



[Agile](#) / Metrics

# Five agile metrics you won't hate

Stats and charts are powerful tools. Use them for good, dear agilists... use them for good.



BY DAN RADIGAN

Metrics are a touchy subject.

On the one hand, we've all been on a project where no data of any kind was tracked, and it was hard to tell whether we're on track for release or getting more efficient as we go along. On the other hand, many of us have had the misfortune of being on a projects where stats were used as a weapon, pitting one team against another or justifying mandatory weekend work. So it's no surprise that most teams have a love/hate relationship with metrics.

But it doesn't have to be this way. Tracking and sharing sound agile metrics can reduce confusion

Up Next  
[Advantage](#) →

## Know your business

"Done" only tells half the story. It's about building the right product, at the right time, for the right market. Staying on track throughout the program means collecting and analysing some data along the way. In any agile program, it's important to track both business metrics and agile metrics. Business metrics focus on whether the solution is meeting the market need, and agile metrics measure aspects of the development process.

### A program's business metrics should be rooted in its roadmap.

For each initiative on the roadmap, include several key performance indicators (KPIs) that map to the program's goals. In addition, include success criteria for each product [requirement](#) such as adoption rate by end-users or percentage of code covered by automated tests. These success criteria feed into the program's agile metrics. And the more teams learn, the better they can adapt and evolve.

## How to use agile metrics to optimize your delivery

The agile metrics discussed below focus on the delivery of software. Whether you are a [scrum](#) or [kanban](#) team, each of these agile metrics will help the team better understand their development process, making [releasing](#) software easier.

### Sprint burndown

Scrum teams organize development into time-boxed [sprints](#). At the outset of the sprint, the team

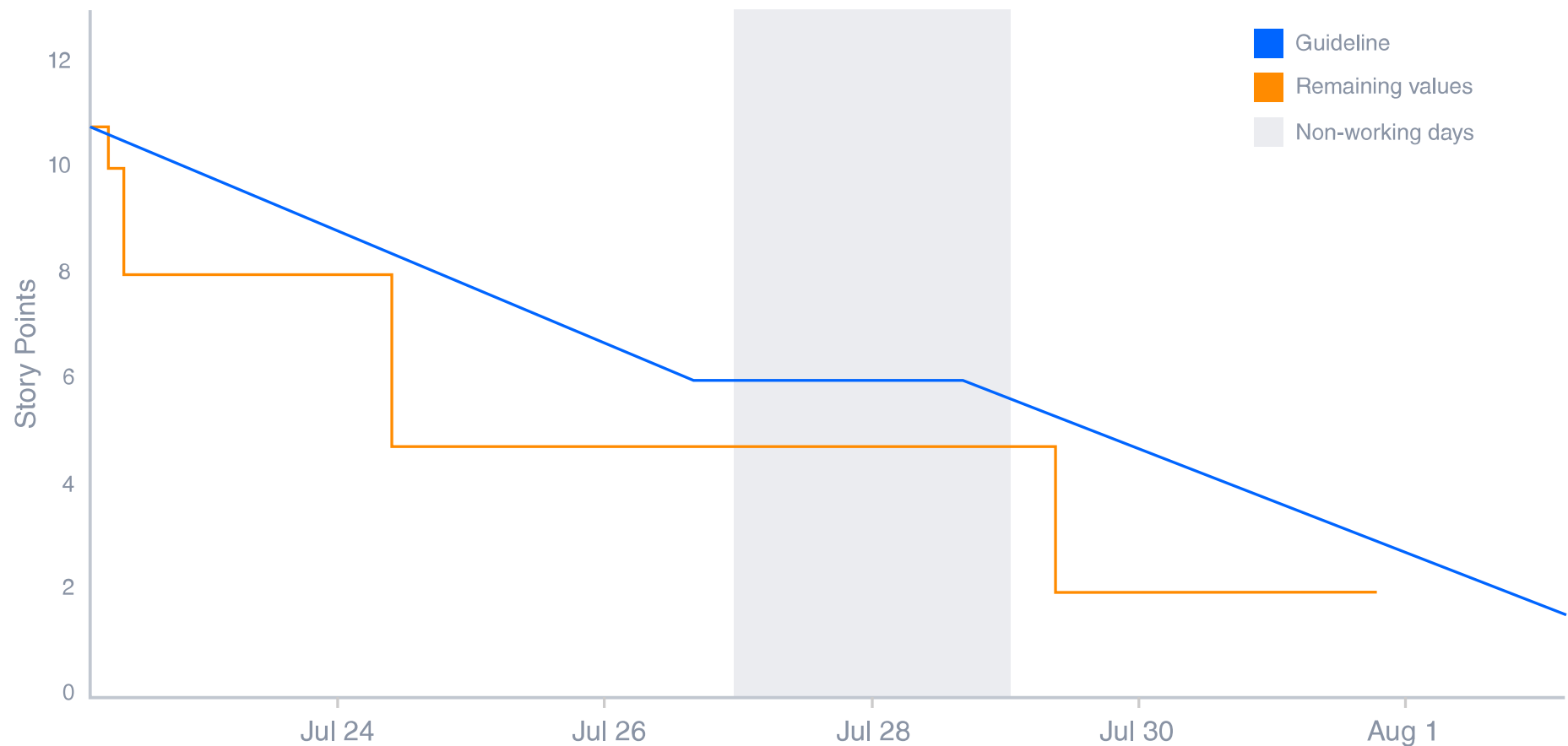


refers to the amount of work left to complete, measured in either story points or hours. The goal is to have all the forecasted work completed by the end of the sprint.

A team that consistently meets its forecast is a compelling advertisement for agile in their organization. But don't let that tempt you to fudge the numbers by declaring an item complete before it really is. It may look good in the short term, but in the long run only hampers learning and improvement.



# Burndown Chart



## ANTI-PATTERNS TO WATCH FOR

- The team finishes early sprint after sprint because they aren't committing to enough work.
- The team misses their forecast sprint after sprint because they're committing to too much work. →

- The burndown line makes steep drops rather than a more gradual burndown because the work hasn't been broken down into granular pieces.
- The product owner adds or changes the scope mid-sprint.

## Epic and release burndown

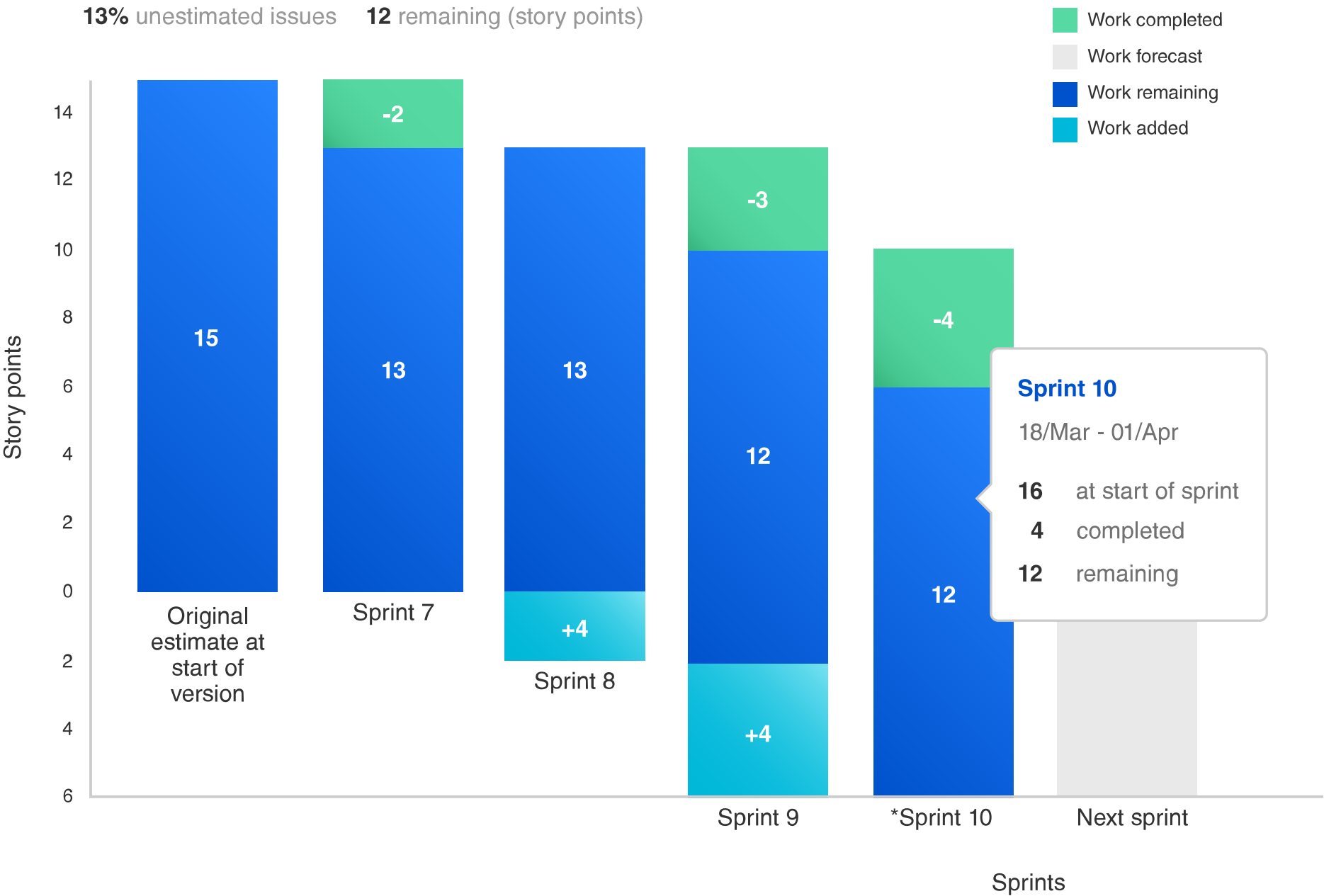
**Epic and release** (or version) burndown charts track the progress of development over a larger body of work than the sprint burndown, and guide development for both scrum and kanban teams. Since a sprint (for scrum teams) may contain work from several epics and versions, it's important to track both the progress of individual sprints as well as epics and versions.

"Scope creep" is the injection of more requirements into a previously-defined project. For example, if the team is delivering a new website for the company, scope creep would be asking for new features after the initial **requirements** had been sketched out. While tolerating scope creep during a sprint is bad practice, scope change within epics and versions is a natural consequence of agile development. As the team moves through the project, the product owner may decide to take on or remove work based on what they're learning. The epic and release burndown charts keep everyone aware of the ebb and flow of work inside the epic and version.



# Epic Burndown

13% unestimated issues    12 remaining (story points)




### ANTI-PATTERNS TO WATCH FOR

- Epic or release forecasts aren't updated as the team churns through the work.
- No progress is made over a period of several iterations.
- Chronic scope creep, which may be a sign that the product owner doesn't fully understand the problem that body of work is trying to solve.
- Scope grows faster than the team can absorb it.
- The team isn't shipping incremental releases throughout the development of an epic.

## Velocity

Velocity is the average amount of work a scrum team completes during a sprint, measured in either story points or hours, and is very useful for forecasting. The product owner can use velocity to predict how quickly a team can work through the backlog, because the report tracks the forecasted and completed work over several iterations—the more iterations, the more accurate the forecast.

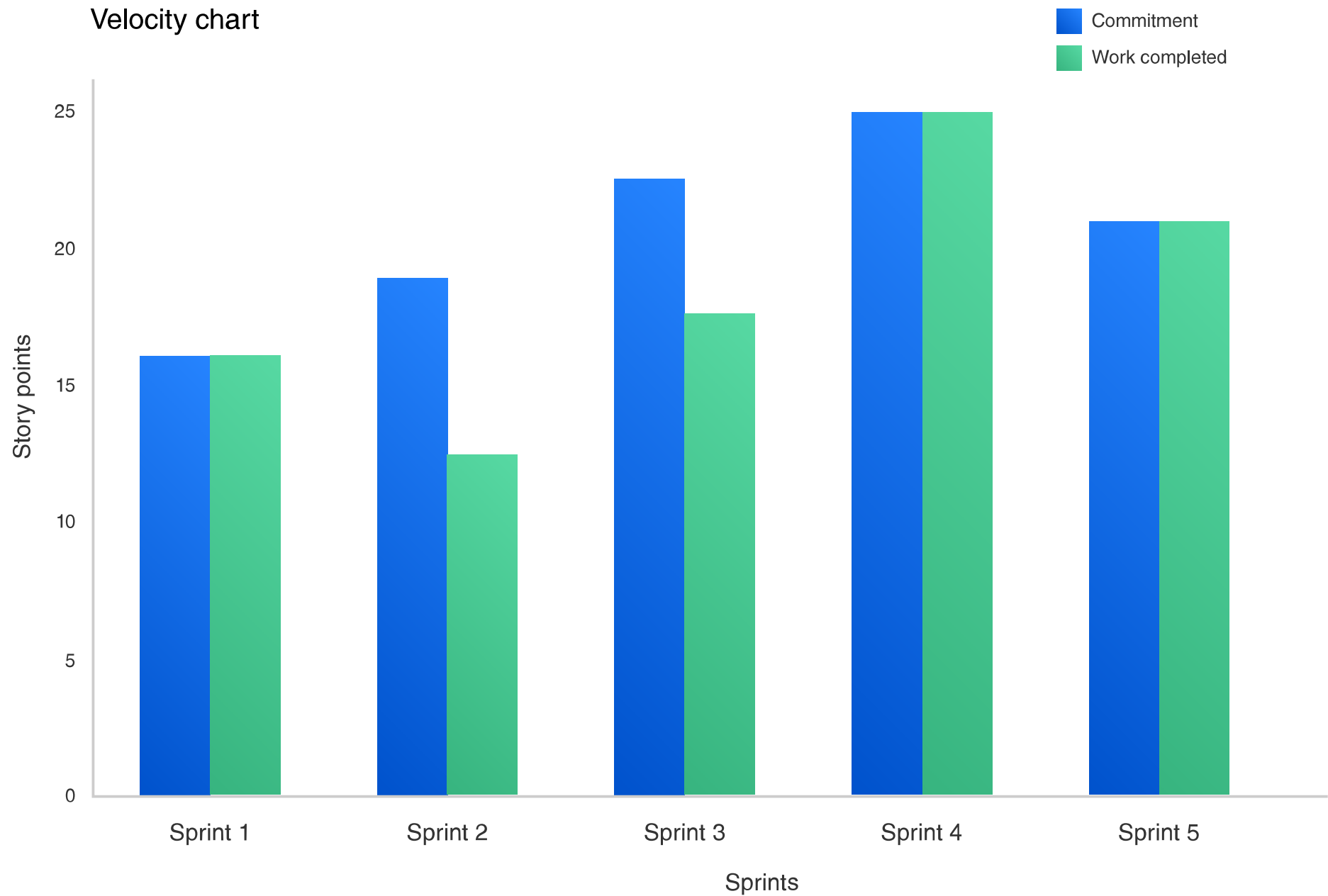
Let's say the product owner wants to complete 500 story points in the backlog. We know that the development team generally completes 50 story points per iteration. The product owner can reasonably assume the team will need 10 iterations (give or take) to complete the required work.

It's important to monitor how velocity evolves over time. New teams can expect to see an increase in velocity as the team optimizes relationships and the work process. Existing teams can track their velocity to ensure consistent performance over time, and can confirm that a particular process change made improvements or not. A decrease in average velocity is usually .

sign that some part of the team's development process has become inefficient and should be brought up at the next retrospective.







When velocity is erratic over a long period of time, always revisit the team's estimation practices. During the team's retrospective, ask the following questions:

- Are there unforeseen development challenges we didn't account for when estimating this work?  
How can we better break down work to uncover some of these challenges?
- Is there outside business pressure pushing the team beyond its limits? Is adherence to development best practices suffering as a result?
- As a team, are we overzealous in forecasting for the sprint?

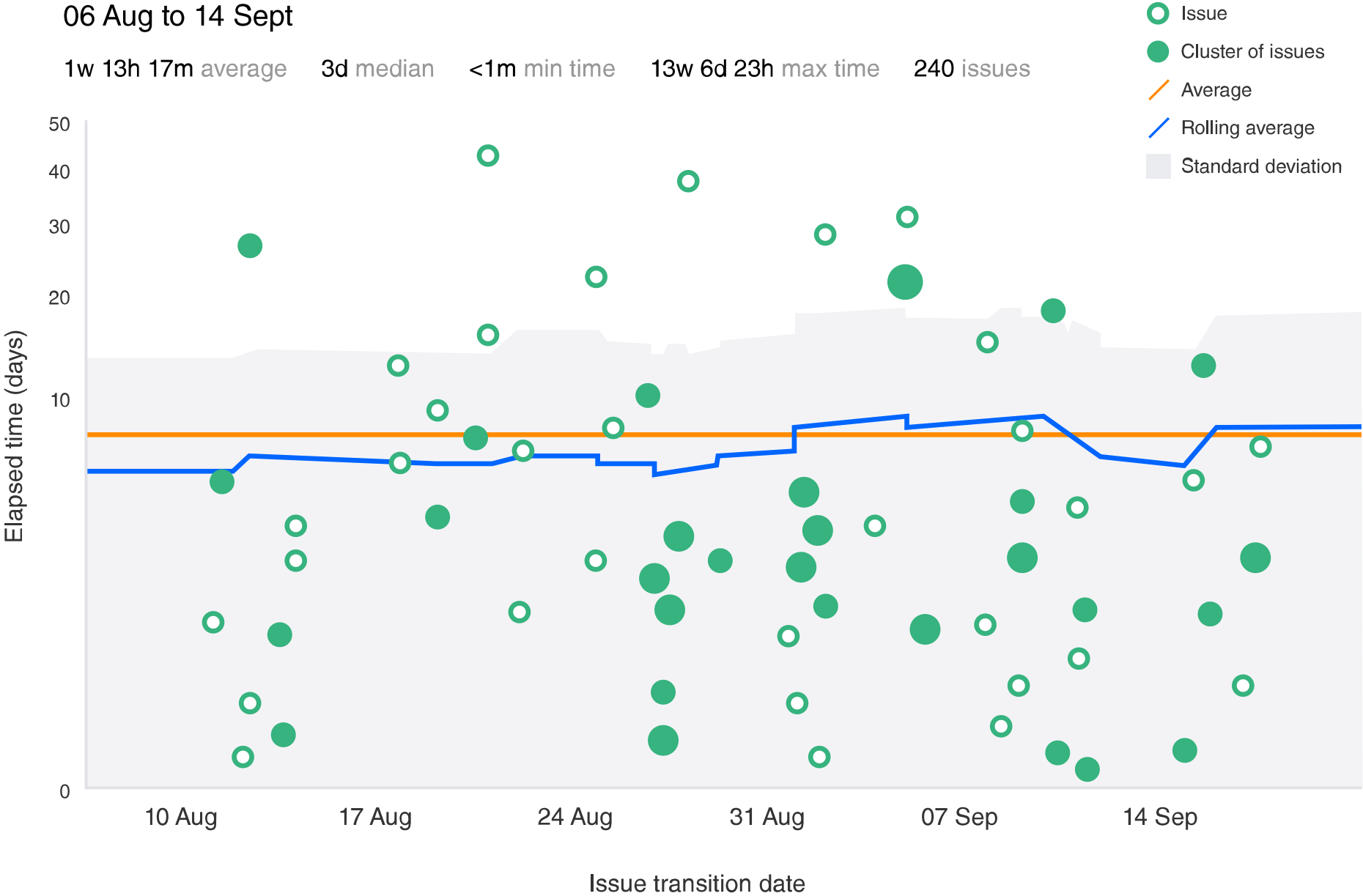
Each team's velocity is unique. If team A has a velocity of 50 and team B has a velocity of 75, it doesn't mean that team B has higher throughput. Since each team's [estimation](#) culture is unique, their velocity will be as well. Resist the temptation to compare velocity across teams. Measure the level of effort and output of work based on each team's unique interpretation of story points.

## Control Chart

Control charts focus on the cycle time of individual issues—the total time from "in progress" to "done". Teams with shorter cycle times are likely to have higher throughput, and teams with consistent cycle times across many issues are more predictable in delivering work. While cycle time is a primary metric for kanban teams, scrum teams can benefit from optimized cycle time as well.

Measuring cycle time is an efficient and flexible way to improve a team's processes because the results of changes are discernable almost immediately, allowing them to make any further adjustments right away. The end goal is to have a consistent and short cycle time, regardless of the type of work (new feature, [technical debt](#), etc.).





ANTI-PATTERNS TO WATCH FOR



Control charts can appear fickle at first. Don't be so concerned with every outlier. Look for trends. Here are two areas to watch out for:

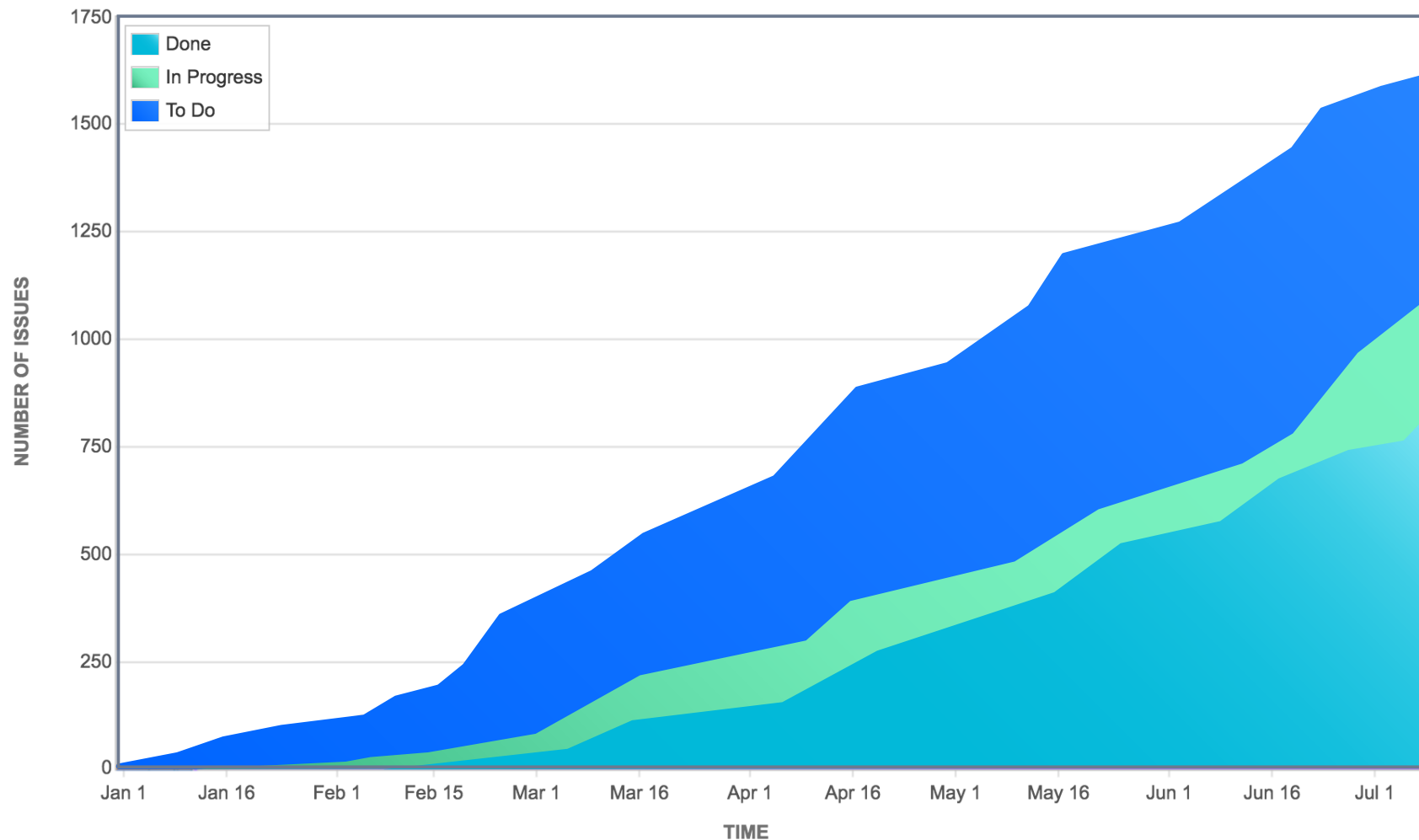
- Increasing cycle time -Increasing cycle time saps the team of it's hard-earned agility. In the team's retrospective take time to understand an increase. One exception: if the team's definition of done has expanded, cycle time will probably expand too.
- Erratic cycle time – The goal is to have consistent cycle time for work items which have similar story point values. Filter the control chart for each story point value to check for consistency. If cycle time is erratic on small and large story point values, spend time in the retrospective examining the misses and improving future estimation.

## Cumulative flow diagram

The cumulative flow diagram is a key resource for [kanban](#) teams, helping them ensure the flow of work across the team is consistent. With number of issues on the Y axis, time on the X axis, and colors to indicate the various [workflow](#) states, it visually points out shortages and bottlenecks and works in conjunction with [WIP limits](#).

The cumulative flow diagram should look smooth(ish) from left to right. Bubbles or gaps in any one color indicate shortages and bottlenecks, so when you see one, look for ways to smooth out color bands across the chart.





#### ANTI-PATTERNS TO WATCH FOR

- Blocking issues create large backups in some parts of the process and starvation in others.
- Unchecked backlog growth over time. This results from product owners not closing issues that are obsolete or simply too low in priority to ever be pulled in.



## Even more metrics

Good metrics aren't limited to the reports discussed above. For example, quality is an important metric for agile teams and there are a number of traditional metrics that can be applied to agile development:

- How many defects are found...
  - during development?
  - after release to customers?
  - by people outside of the team?
- How many defects are deferred to a future release?
- How many customer support requests are coming in?
- What is the percentage of automated test coverage?

Agile teams should also look at release frequency and delivery speed. At the end of each sprint, the team should release software out to production. How often is that actually happening? Are most release builds getting shipped? In the same vein, how long does it take for the team to release an emergency fix out to production? Is release easy for the team or does it require heroics?

Metrics are just one part in building a team's culture. They give quantitative insight into the team's performance and provide measurable goals for the team. While they're important, don't get obsessed. Listening to the team's feedback during [retrospectives](#) is equally important in



growing trust across the team, quality in the product, and development speed through the release process. Use both the quantitative and qualitative feedback to drive change.

SHARE THIS ARTICLE



DAN RADIGAN

Agile has had a huge impact on me both professionally and personally as I've learned the best experiences are agile, both in code and in life. You'll often find me at the intersection of technology, photography, and motorcycling. Find me on Twitter! @danradigan

TUTORIAL

**Learn burndown charts with Jira Software**



The go-to-guide for burndown charts in Jira Software. Learn how to monitor epics and sprints with burndown charts.

[Try this tutorial →](#)

#### UP NEXT

## The agile advantage - How to start your transformation

Learn how to connect business goals to agile development practices, the advantages of agile, and industry best practices.

[Read this article →](#)

### Agile Topics

Agile project management

Scrum

Kanban

Design





Software development

Product management

Teams

Agile at scale

DevOps

Sign up for more agile articles and tutorials.

Email

email@example.com

Subscribe

