FLOW

PLURALSIGHT

RETROSPECTIVES

Facilitate visibility and continuous learning retrospective-over-retrospective

Retrospectives are a time for the team to reflect on the previous sprint and discuss any successes or failures, and to collaborate on what actions they can take to improve the process and the product in the next iteration.

Start incorporating data about *what* was shipped (meaning the *product)* and *how* it was shipped (meaning the *process*). By bringing this information in, you can spend less time talking about what happened and more time talking about why - and what the team would like to start, stop, or keep.

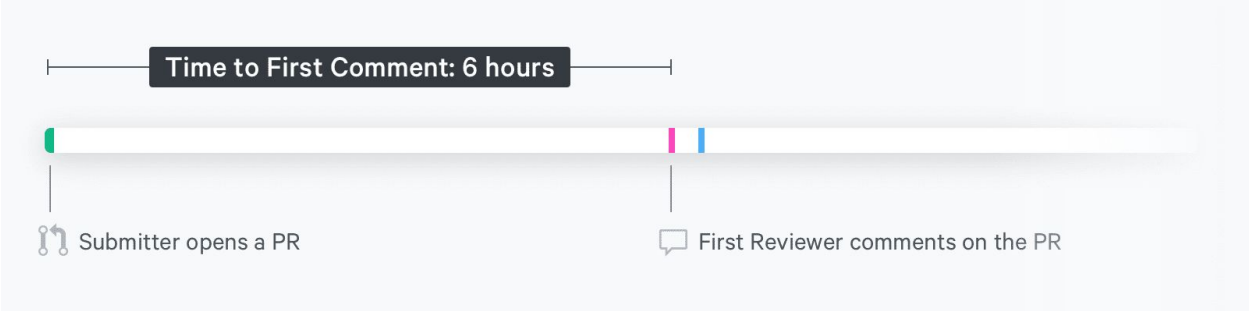
Here's how to start incorporating analytics about the delivery process in your retrospectives:

1. Prepare by identifying the most relevant metrics to your team, given your team's workflows.
2. Then, gain buy-in from the team and establish target health ranges (work agreements) for where we expect these metrics to be for our team
3. Use that information to facilitate a conversation about the "why" -- which may mean any deltas seen between the team's target ranges and the actuals.
4. Have a discussion about what learnings could be applied to the next sprint, including what the team should stop, start, or keep doing.

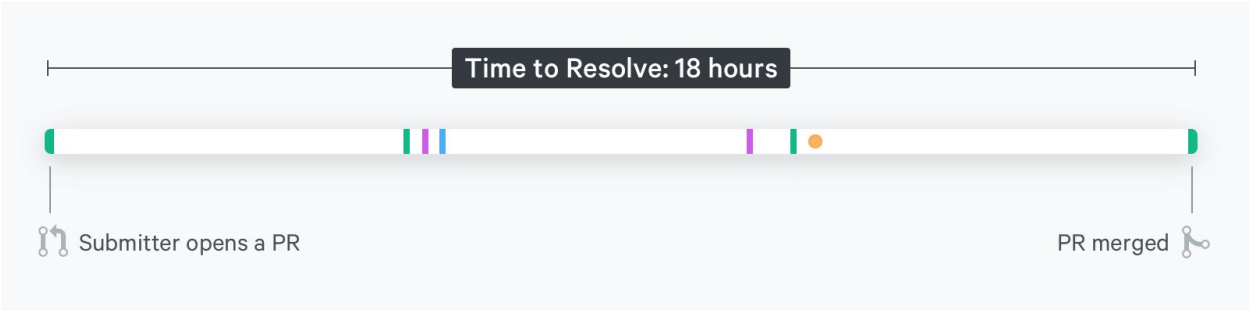
Metrics to consider including in your retrospectives

In the PR Resolution report:

There are two metrics in the *PR Resolution* report that we most commonly see teams use in their retrospectives: *Time to First Comment*, and *Time to Resolve*.



Time to First Comment is the median time, in hours, between when a pull request is opened and the first reviewer comments on it. This helps the team see how quickly they're responding to PRs once they're opened. This helps us understand the length of our first wait state, and whether we are delivering on our work agreements to help each other move our work forward.

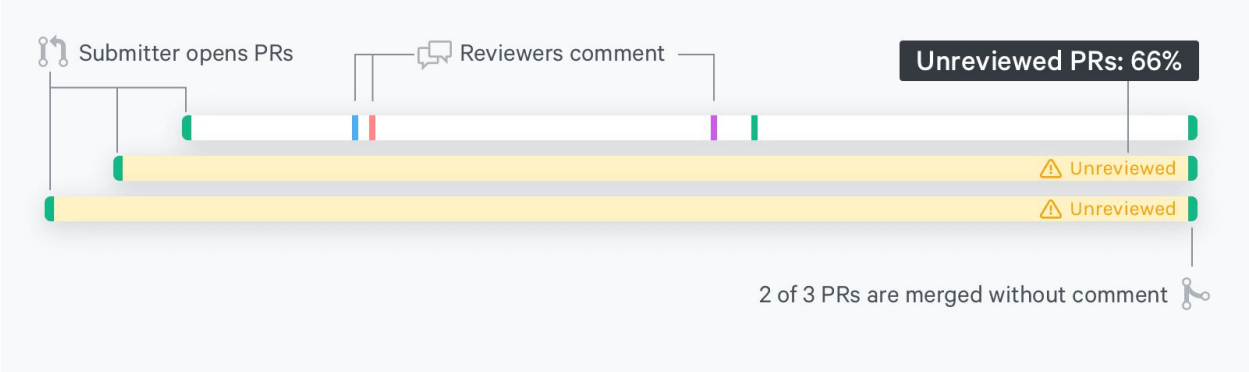


Time to Resolve is the average time, in hours, that it takes a pull request to be resolved - whether that means merging it, closing it or withdrawing it. This metric looks at a broader picture of how work is being handled during the review process and can help the team visualize how slight changes in activities like *Time to First Comment, Responsiveness,* and *Reaction Time* can influence the team's overall *Time to Resolve*.

Ideally, you can bring this report into retrospectives with a few hypotheses to spark a discussion, but remember that this is the *team's* discussion about what patterns or trends they're seeing that are causing PRs to take a longer time to resolve. Encourage the team to get familiar with the metrics and to begin it in their discussions.

In the Submit Fundamentals report:

The *Submit Fundamentals* report can help you visualize how team members are handling their own work in the review process. There is one metric in this report that is a great starting point for incorporating data into your retrospectives. That metric is *Unreviewed PRs*.



Unreviewed PRs measures the percentage of PRs that are opened and merged without ever receiving a comment nor an approval.

This is an exceptionally important metric for us to understand our general tolerance for risk with solutions that we are moving forward to our customers. The higher the rate of *Unreviewed PRs*, the more likely it is that we're introducing errors to the codebase.

Though there are exceptions, most engineering leaders agree that Unreviewed PRs should be as near zero as possible.

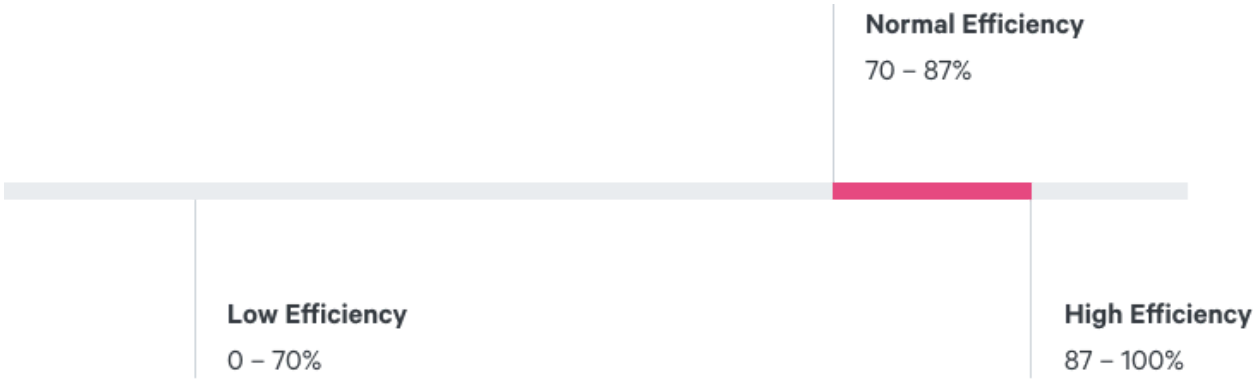
In the Code Fundamentals report:

There are two metrics in this report that we most commonly see teams use in their retrospectives: *Active Days per Week*, and *Efficiency*.

Active Days measures the average number of days per week that the team checked in at least one commit. It is a simplistic metric that can generate complex, powerful conversations. This metric will help your team identify whether there are any upstream or external bottlenecks to moving work forward in the codebase, or whether there's anything in the delivery process that's causing friction and slowing the team down.

Efficiency is the percentage of code that survives beyond three weeks. This metric helps us understand the stability of our codebase throughput; it can be a simple way to spot broad deviations from your normal baselines and understand whether there are any key findings that caused the deviation from the norm.

Efficiency levels will vary between individuals, teams, and projects, so it can be helpful to get a feel for what "normal" looks like in the context of your team. That way, it'll be easier to notice when something is amiss.



Source: The Fundamentals Metrics, a 2017 GitPrime Data Science Study of 7M commits from 87,000 professional software developers

Efficiency is trying to answer the following question: “How much of an author’s work is not re-writing their code?” Attempting to answer this question relates directly to the concept of code churn—the amount of an author’s work that is self re-written within three weeks. For a deeper review of the causes of shifts in Churn, read this guide.

NOTES: