Case Study: Moving To The Pull Request Workflow & Integrating Quality Engineers On The Team For Better Software Quality

By Christopher W. H. Davis, Continuous Improvement

This is an excerpt of a real world scenario from the end of Chapter 4 of Continuous Improvement that shows the reader how to apply source control data toward continuous improvement. We've been through how to collect and analyze SCM and CI data so it's time to see it in action.

The book Continuous Improvement walks the reader through the collection and analysis of data to provide metrics that guide the continuous improvement of agile teams. Each chapter ends in a case study outlining the application of this technique with specific types of data. This is an excerpt of a real world scenario from the end of Chapter 4 that shows the reader how to apply source control data toward continuous improvement. We've been through how to collect and analyze SCM and CI data so it's time to see it in action.

This real world scenario will show you how this can be applied to your process.

The team in question was generating a lot of bugs. Many of them were small issues that likely should have been caught much earlier in the development process. Regardless, every time they would cut a release they would deploy their code, turn it over to the Quality Management (QM) team, and wait for the bugs to come in. They decided to make a change to get to better quality.

After discussing the issue, the team decided to try out the pull request workflow. They were already using Git, but developers were all committing their code to a branch and merging it all into master before cutting a release. They decided to start tracking commits, pull-requests and bugs to see if using pull-requests decreased their bug count. After a few sprints they had the graph shown in Figure 1.

Figure 1 Bugs aren't trending down as the team starts doing pull requests.

For source code, sample chapters, the Online Author Forum, and other resources, go to http:// http://www.manning.com/davis2/
To make trends a bit easier to see, we will divide pull requests and commits by 2 so there isn’t such a discrepancy between the metrics. That is shown in Figure 2.

![Figure 2](image)

**Figure 2** The same data with variance decreases between bugs and other data

That makes it a lot easier to see the variance. As you can see from Figure 2 not much changed; even though there is a big dip from bugs from sprint 18 to 19, we aren’t decreasing over time – there was just a big jump in bugs in sprint 18. After discussing it a bit more, the team decided to add some more data points to the mix. To see how much collaboration was happening in the pull requests, they started adding comments to their graphs as well. That resulted in the chart shown in Figure 3. To keep consistent, we’ll divide comments by 2 as well.

![Figure 3](image)

**Figure 3** Adding comments to our graph and finding an ugly trend

Figure 3 shows that there aren’t many comments along with the pull request, which implies there wasn’t much collaboration going on at all. Since the bug trend wasn’t changing it looked like the changes to their process wasn’t quite taking effect yet. The workflow itself wasn’t effecting the change they wanted, they needed to make a bigger impact to their process. To do this, they decided to make their developers act like the QM team when they were put on a pull request. The perspective they needed wasn’t just “is this code going to solve the problem?”, but “is this code well-built and what can go wrong with it”? There was some concern about developers getting less done if they had to spend a lot of time commenting on other developers’ code and acting like the QM team. To help coach them, they moved one of their QM members over to the development team and the team agreed that if this could result in fewer bugs then the time spent up front was well spent. They started taking the time to comment on each other’s code and ended up iterating quite a bit more on tasks before checking them in. A few sprints of this resulted in Figure 4.
Figure 4 Everything is trending in the right direction!

Figure 4 shows that as collaboration between development and quality increased, in this case shown through comments in pull-requests, the number of bugs was going down. This was great news to the team so they decided to take the process one step further. They brought another member of the QM team down to work with the developers on code reviews and quality checks to avoid throwing code over the wall to the QM team.

**Test Engineers**

For a long time, the role of the quality department in software has involved checking to make sure features were implemented to spec. That is not an engineering discipline, and as result many people in the quality assurance (QA) and quality management (QM) space were not engineers. To truly have an autonomous team, quality engineering has to be a significant part of the team. The role of the quality engineer, aka QE aka SDET aka Test Engineer has started to become more and more popular. However as quality moves from one state to another in the world of software engineering, this role is not very clearly defined and often you have either someone with an old quality background who recently learned to write code, or you have an expert in test running technology. Neither of these actually works; you need a senior engineer with a quality mindset.

As shown in Figure 4, over time commits and pull requests started increasing as well. As the development team started thinking with a quality mindset they started writing better code and producing fewer bugs. Also by combining the QM team with the development team many issues were found and fixed before deploying code out to the test environment.