# CodeFlow

## Improving the Code Review Process at Microsoft

case study

A DISCUSSION WITH JACEK CZERWONKA, MICHAELA GREILER, CHRISTIAN BIRD, LUCAS PANJER, AND TERRY COATTA

You may be wondering, "Code review process? Isn't that obvious?" But code reviews are pervasive. Any developer is likely to be asked at any time to review someone else's code. And you can be sure your code is reviewed. For some developers, code reviews take up a portion of each day. So there's your answer: large numbers of very well-compensated people spend a great deal of time on this activity, meaning the aggregate costs are substantial. If you're talking about a development shop the size of, say, Microsoft... well, then, the investment regularly made in code reviews can amount to something quite impressive indeed.

That's only one of the reasons that Jacek Czerwonka and his TSE (Tools for Software Engineers) team at Microsoft set out to study how the code-review process plays out across the company. Another had to do with taking on a challenge they found interesting in the sense that, beyond their important role in software-engineering integration, code reviews involve some rather complex social dynamics that elude simple modeling.

Then there also was the fact that Microsoft's code-review tool represented an opportunity to touch every developer throughout the entire company. For a group

charged with boosting developer productivity, that's just the sort of lever dreams are made of. What's more, the tool also offered TSE's researchers something they could instrument to collect data and generate metrics that, in turn, could be used to enable further research.

So, that's why the group set out on this journey. To recount what it was like, where it led, and what was learned along the way, Czerwonka discusses the undertaking here, along with fellow researchers Michaela Greiler and Christian Bird. Also on hand to help steer the discussion are Lucas Panjer, the senior director of engineering at Tasktop, and Terry Coatta, the CTO at Marine Learning Systems, a Vancouver-based startup working to develop a learning platform.

LUCAS PANJER What exactly is it that initially moved you to zero in on the code-review process?

JACEK CZERWONKA This group was formed several years ago with the goal of encouraging the adoption of a common set of software-engineering tools across the whole of Microsoft. We've been on this path for a while now. We're not done yet. But there are a few places where we've managed to centralize the tools quickly, and one of those is in code-review tooling.

Clearly, in looking at that aspect of the engineering workflow, we saw there were already some tools in place, so we just concentrated on determining what we could do to make improvements. First we wanted to learn what we could from actual experience since you always want to start with a foundation grounded in practice, as well as theory. So, we started looking at any qualitative or

quantitative data we could get our hands on that had to do with the code-review tooling and process already in place at Microsoft. That's how we started on this journey of trying to understand where the process originated and how it has evolved over time. What are the factors that drove that evolution? How is the process currently applied? How does it work with open source? How does it work within Microsoft? And what happens when we find ourselves collaborating with others?

LP What did you end up initially focusing on?

CHRISTIAN BIRD In general, we wanted to find out what prompted people to do code reviews in the first place. How many people were usually involved? What types of issues were being raised? What was it that led people to make changes? And what typically led people not to make changes?

TERRY COATTA Were the engineering teams themselves pushing for this line of inquiry? That is, were people coming to you to say, "We're sure spending a lot of time with code reviews, but it doesn't seem like we're getting all that much out of it"?

CB Mostly it was because this was an area where the data was both plentiful and readily available. With that being said, once people found out what we were doing, they proved to be quite receptive. It wasn't like they wondered why we were doing this research. In fact, it was just the opposite. People generally were very supportive of improving the code-review process and, if anything, said they wished it was treated as a first-class citizen. Also, many were pretty excited to learn there was data available they would be able to track themselves.

LP Once people engaged with you and told you what they thought was valuable, did they also let you know what else they wanted?

CB What people wanted for the most part was the ability to do their own tracking, along with a way to look at how they were doing in comparison to other teams. We came up with metrics that align with some of the targets teams at Microsoft have for what they want to achieve at different points in the software-development process. For example, they would want to know if they were on track for getting a commit into master within a month. Or they would want to see if they were well on their way to achieving 80 percent test coverage.

Similarly, for code review some teams had targets, while others did not since they didn't have a way to measure that. So, they might decide that at least two people should sign off on every code review and that each review would have to be completed within a 24-hour period. Until we started collecting the data around code reviews, analyzing it, and then making it more generally available, teams had no way of measuring that. Yet they *wanted* to be able to do that since they were already measuring other parts of their development process. As a consequence, people started coming to tell us what metrics they would find useful. Then we would just add those to metrics we were already collecting. It turns out that much of our effort was actually driven by what the development teams themselves were telling us they wanted to be able to measure.

TC Since you say this tooling for code reviews is something everybody at Microsoft now uses, can you give us a brief description of the features it offers and how you think

those compare with what's available to most people outside of Microsoft?

JC Well, we're talking now about things we did with our tool [called CodeFlow] a few years ago, and tooling has a way of converging out in the world at large over that much time. So, some of the changes we made back then might now seem fairly obvious to people who are using other code-review tools that have since come to work in much the same way.

The brief summary is that we made a number of changes to finely tune the underlying subsystem. We also trained the tool to be super-precise in terms of tracking changes as people move through numerous software iterations. That is, as you move from one revision to the next, you can imagine that your code changes end up moving around as some code gets deleted, some new lines are added, and chunks of code are shuffled around. That can throw your comment tracking severely out of sync with what you had once intended. Overcoming that took work, but we now know from feedback that it's greatly appreciated and thus well worth the effort.

**CodeFlow remains a tool that works client-side, meaning you can download your change first and then interact with it, which makes switching between files and different regions very, very fast.**

*—Jacek Czerwonka*

Another thing we focused on was performance. For that reason, even today CodeFlow remains a tool that works client-side, meaning you can download your change first and then interact with it, which makes switching between files and different regions very, very fast.

It also helps that CodeFlow has essentially become ubiquitous throughout Microsoft. That's because we used something like a viral marketing strategy in that the moment you were added as a reviewer, you received a notification, which allowed you to open the review by

simply clicking on a link. Then the CodeFlow client would be installed and the review would be opened. So, soon after the tool was introduced to a group, it would start to permeate the fabric of that team pretty much all on its own. The choice not to require a special install for CodeFlow proved to be a really good one.

LP Is there anything in particular from the user's perspective that distinguishes CodeFlow from either Git or Gerrit? How would you say it differs from what you find with pull requests and patch set-based tooling?

CB It comes down to being a native app rather than a Web capability, meaning it enables much richer interactions than you would get otherwise. I've been through the Git and the pull request stuff, and it's absolutely the case that you can easily jump around from comment to comment, and you also get things that work like score boxes. Which is to say they feel like rich native clients, so I realize you can accomplish this with a web experience.

As for Git and Gerrit code reviews, what you get there just amounts to lists of diffs. I mean, you also can add comments, but, in the end, that just makes it more difficult to track things or navigate everything effectively.

So, the fact that CodeFlow is native and is treated like a first-class citizen on the desktop makes it more usable.

MICHAELA GREILER I also really like the richness of CodeFlow's commenting features. You can, for example, mark just a single character within a line instead of calling out the whole line of code. That way, people can immediately see exactly where the issue is.

Also, to this day, very few code-review tools let you span regions, but with CodeFlow you can attach a

comment at the same time to a number of deleted lines and inserted lines—and then track all of that through succeeding iterations. Another feature worth pointing out is comment threading, which lets you resolve an entire thread of comments at the same time rather than dealing with each comment individually.

Code reviews generally conjure up notions of troubleshooting. More specifically, people tend to associate them with the never-ending search for bugs.

It turns out that's not nearly as central to the code-review process as you might think. Which is not to say that finding bugs is unimportant or discouraged. And yet it seems the real win comes in the form of improved long-term code maintainability.

LP Which problems did you decide to attack first?

JC Most of the issues we chose to focus on were process oriented. The tool itself is quite flexible and adaptable to practically any process. We spent a lot of time trying to understand the benefits of code review and what was getting in the way of achieving those advantages. Also, we wanted to understand how the existing code-review tool was being used. We were interested in learning more about the costs and the turnaround times in hopes we would be better able to see what the drivers were.

MG Also, one of the issues we looked at was how to create a reviewer recommender since programmers had been complaining to us about how difficult it was to find the

**How many people should you include in a code review? Is there a number beyond which it becomes counterproductive?**

*—Jacek Czerwonka*

right people to look over their code. Chris started working on a tool that would deliver a listing of people with the expertise to match the sorts of problems addressed by your code, along with suggestions as to which of these people you might want to add to a review.

Something else Chris and I studied for a while was code-review usefulness. That wasn't a problem we were trying to solve, of course, but we did want to understand which aspects of code reviews tend to be most valued by engineers—that is, by both reviewers and programmers. What did they see as being most useful? It didn't take us long to conclude that it was not the mere decision to accept or rework the code that the reviewers were interested in, it was the comments that added to the value of the review. On the other hand, some comments just increase the burden of the code review and slow down the development process. So then we wanted to know what *kinds* of comments they found most useful, since we could then start thinking about how to encourage and lend greater emphasis to those.

JC Just as this interesting question of usefulness led to practical implications later on, the same might be said of the work that was done to look into other process-related questions. For example, how many people should you include in a code review? Is there a number beyond which it becomes counterproductive? We all intuitively feel that smaller reviews are better, but where exactly to draw that line? And what's the optimal amount of time to allow for a review?

MG Another interesting thing we found is that, while the popular notion is that code reviews are mostly about finding bugs, only a very small percentage of the code-

review comments we studied actually had anything to do with bugs at all. In fact, most of the comments were about structural issues and style problems. Sometimes they were even about really minor issues, like spelling. Basically, what we found was that many reviewers were using their commenting platform to *discuss* these issues and share their knowledge.

We found it very enlightening to categorize these comments and do some mappings to determine which ones were thought to be the most interesting or useful. It turns out that generally proved to be comments that identified functional issues, pointed out missing validation checks, or offered suggestions related to API usage or best practices.

LP Just for context, can you also speak to the scale of this research—the size of the codebase you were working with, the number of code reviews you analyzed, or the number of developers who were involved?

CB We did a number of different studies, many of which were more quantitative than observational. In one case, we did an initial study where it became clear that the depth of knowledge someone has of a certain piece of code will definitely show up in the quality of feedback they're able to offer as a reviewer. Which is to say, to get higher-quality comments, you need reviews from people who have some experience with that particular piece of software. Then, to check out that conclusion, we spoke with and observed some engineers who had submitted reviews for code already familiar to them. We also observed some engineers who had been asked to review code they had no prior experience with. That was a small study, but it left us with some definite impressions.

There also were those studies Michaela just mentioned, where we considered comment usefulness. That was based on data gathered from across all of Microsoft and then fed into a machine-learning classifier we had built to categorize code reviews. We ended up using that to classify 3 million reviews of code that had been written by tens of thousands of developers and drawn from every codebase across the whole of Microsoft—meaning we're easily talking about hundreds of millions of lines of code. Obviously, the quantitative data analysis we were able to perform there was based on a substantial amount of data. The qualitative observational studies, on the other hand, were typically much smaller.

MG We definitely had a tremendous amount of data available—essentially all the code written for Office, Windows, Windows Phone, Azure, and Visual Studio, as well as many smaller projects.

JC We also enjoy an advantage here at Microsoft in that we have so many different product types. We look at the work people do on operating systems, as well as apps and large-scale services and small-scale services and everything in between. We're very aware of the different demands in each of these areas, and we make a point of keeping that in mind as we do our studies.

LP In those cases where you could derive data from the use of CodeFlow, were you also able to further instrument the tool to augment your studies?

JC One of the most interesting things to surface from instrumenting CodeFlow was just how much time people were actively spending in the review tool. That's because

we've found that people will often open multiple instances of the tool and then, as they get a bit of free time, do a small review here and then another small review there. So, just because you can see the tool has been open for a certain amount of time doesn't mean you can assume there has been activity for that whole time. We have the telemetry to determine just how long you were navigating around within the app. That has allowed us to determine that people, on average, spend about 20 minutes per day actively working in CodeFlow—which amounts to a significant amount of time once you multiply that by 40,000 people.

CB From all that, we've been able to make a number of general observations we're always happy to pass along as recommendations. In fact, one suggestion I would offer to anyone looking to do something similar to what we've done in analyzing their own organization's code-review process is that, in considering what data to collect, stay as close as possible to the actual object model employed by the application itself. For example, there's almost a 1:1 correspondence between the tables in our database and the classes in the application. As a result, we didn't have to think very hard about whether to collect something or not. We just grabbed everything.

So, we ended up collecting all this raw data, and one advantage of that is, even if you don't see an immediate need for some of that data, you might find a use for it later as new studies come up. Which means you won't be faced with needing to go back and update your data-collection system to provide for that. The downside is that you'll also have all this raw information on your hands that hasn't

been processed for use, which means some engineer is going to have to come along later to build a metrics layer on top of all that. That will leave you with two levels of data—the analytics layer and another layer containing the raw object model data—which people can dive into later if they're looking to get their hands really dirty.

That sort of layering turned out to be a really smart move for us since we now can cater not only to the casual user who simply wants to look at metrics and reviews but also to someone who wants to dive into things.

LP Are you saying that after you've created these tools for your research purposes, other teams will go on to use them to reflect on their own processes?

CB Yes. In fact, we did a study a few years ago where we contacted some of the teams that were using our data to discover exactly what they were doing with it, as well as to see whether they had managed to improve the process in any way. We thought that this might be a way to find where we needed to take our own research.

We found that some teams were using the data to generate scorecards, whereas some were using it to discover where people were having problems understanding the codebase and then using those insights to drive their training programs. We ended up talking with at least another dozen teams, and it was interesting and surprising to learn about the different ways some of those teams had used our data.

LP What were some of the bigger surprises?

CB The biggest surprise for me was learning that some teams would use our tools to identify code reviews that took too long or contained only a few comments. Then

they would open the code reviews based on that data, and the reviews would tell them what code had been used and what part of the code was being reviewed. They would dig into that and quickly determine, "Oh, it looks like people are having a tough time reviewing code that uses this particular API." That's how they would determine that their next training session ought to be devoted to that API.

TC Have you developed any metrics for essentially grading the quality of code reviews?

CB Not as such, but I know some teams have built live dashboards around this data. Some development teams have mounted a massive TV monitor right on the wall where metrics like "Time since last bug" or "Time to delivery of next release" can be displayed. One team told us they also put code-review data up on their scoreboard so people could see how many code reviews are on backlog or how much time on average is required to complete a code review. From what they told us, it seems that having that data up on a realtime dashboard, mission-control style, has proved to be quite motivating.

Delivering a new set of capabilities for managing and improving Microsoft's code-review process was the primary goal right from the start. In the course of accomplishing that, much was also learned about certain general code-review principles—guidelines that might also be applied to beneficial effect elsewhere. In fact, subsequent research has offered surprising evidence of just how similar the impact can be when many of these principles are followed

at companies other than Microsoft—or, for that matter, by open-source projects.

LP Looking back to when you first started this project, what would you say came up most whenever you questioned people about their primary motives for doing code reviews?

MG We did a survey where we asked people to rank their reasons. What came out of that tended to be fairly obvious: improving the code, finding defects, knowledge transfer... that sort of thing. But then, when we launched this other study to categorize the comments that had been left in the actual code, we found they only rarely aligned with those stated motivations.

LP Interesting. What did those comments chiefly focus on?

MG There were a lot of comments about the documentation, of course. And you would see some remarks having to do with alternative solutions. There also were comments about validation, which admittedly leaned in the direction of bug resolution since people would say, "You know, if this particular corner case went away, you would be able to eliminate some of these problems." People also had things to say about API usage—and best practices as well. On the whole, I'd say these sorts of comments far outweighed any that focused on specific defects.

JC To Michaela's point regarding this mismatch between expectations and reality, despite the fact that people consistently said their primary reason for doing code reviews was to discover bugs in code, only 15 percent of the comments we found in code actually related to bugs.

**Only 15 percent of the comments we found in code actually related to bugs.**

*—Jacek Czerwonka*

For example, we would find comments about control-flow issues or use of the wrong API—or even use of the right API but in the wrong way. On the other hand, at least half of the comments were about maintainability. So, it would seem that for the reviewers themselves, identifying maintainability issues proves to be more of a priority than uncovering bugs.

LP Now that your work has been out there for a number of years, what sort of impact have you seen on code-review policies and practices across all the different development teams?

JC One of our top goals was to reduce the amount of time required to do a code review on average. We looked to discover where it was that people seemed to be spending an inordinate amount of time, and that's what led to the creation of a reviewer recommender. It's such a simple thing, really, but it can be hard to find people with the right experience if you're part of a large team. Having an automated system to identify those engineers who have some familiarity with the file where some changes have been made can help cut down on the time required to get those changes reviewed.

Something else we've done, quite recently, is to give the developers a way to explain what it was they were trying to accomplish. This is because a complaint we commonly hear from reviewers is that it can be quite challenging to understand the reasoning behind a code change. Which is to say they would like some way to get into the mindset of the person who made that change so they can better understand whether it actually makes any sense or not.

One way of dealing with this is to show more than just

the isolated section of code where a change has been made. Instead, we show entire files so reviewers can get a better sense of the code around each change. We also wanted to provide some means for the author of a change to offer additional information so reviewers could better understand their reasoning. Toward that end, our system now lets authors put tags on files and regions to indicate which files are at the heart of a change and so should probably be given particular attention. For example, the tags can be used to quickly indicate which changes have been made to test cases as opposed to the product codes. Or they can be used to call out certain files or changes with potential security implications.

LP Do you have any other new capabilities in the works?

JC The fundamental underlying factor we're trying to address is the size of code reviews since that affects both the time required to produce a review and the usefulness of the comments that come out of it. It's a difficult problem to address because some of the issues are cultural in nature, and some relate to workflow. Still, there are times when two unrelated concerns end up getting crammed into a single review, so we're hoping we'll be able to untangle some reviews by automatically splitting those concerns into two smaller reviews. On average, that ought to lead to better turnaround times, as well as better outcomes.

LP Have you taken any steps to get development teams to focus their code-review time on correctness and content versus style? Have any tool changes or process changes been implemented toward that end?

JC We haven't done a proper study of that, but there is a team here that's done something along those lines. This

is something that had to do with some factoring changes they considered to be low-risk—such as the renaming of methods or local variables. For example, this might involve putting a special tag on a review to say, "We don't really need to have two people look at this. One is enough since it's very unlikely we'll have any functionality issues here." Modest as that might seem, it can also prove profound since it turns out there are many changes like this floating through a legacy system—*clogging* the system.

The thing to remember is that it's not just about making one change go faster, since what you're dealing with here is a pipeline of changes—meaning that any change you can redirect to a lighter-weight path is going to lower the load on your key people and get it out of the way of other changes waiting to be reviewed. That's just the sort of thing that makes for a more efficient system all the way around.

TC With an eye to the people outside of Microsoft that don't have your tooling, do you have any recommendations from your experience that might prove relevant?

JC I'd say the one thing to recognize is that comments related to maintainability are primarily what you're going to get out of the code-review process. Contrary to popular opinion, locating bugs is not the primary outcome. The other important thing to bear in mind is that the smaller a review is, the better it's going to be. In our case, we've found that if a review contains more than 20 files, it's too big already. In fact, from our study of all the data at our disposal, we've concluded that for more than 20 files the density and usefulness of comments degrades significantly. This is actually more a rule of thumb than a

precise limit, but it is useful to keep in mind.

Also, if your organization has data from past reviews, I'd suggest investing in a recommender system that can help make some of the administrative steps a little less tedious. You can even use these systems to automatically address some of your maintainability issues, which is something we're starting to get into these days. That is, you can imagine that some of these maintainability issues are essentially things that might be autodiscovered and flagged, which means you then don't have to expend any human resources to get this accomplished.

Another thing, as we just discussed, is the idea that two signoffs on every change might be too many. If you look at the distribution of comments made by either the first or the second reviewer, you'll find that your first reviewer typically discovers the most egregious problems. In many cases, waiting for a second reviewer to corroborate those findings before allowing the commit into the main source tree might be less efficient.

MG My biggest takeaway from the survey is to always make the burden of code reviews just as small as you possibly can. Part of that comes down to having a good code-review process that enables and encourages comments that can be easily reviewed.

Another important consideration has to do with supporting the reviewers themselves by giving them advance notice about any reviews that might be coming up and giving them enough context so they'll be able to dive right into a review without having first to figure all that out for themselves. Doing what you can to reduce the size of reviews can also be helpful. But I think what's really

important is to make the reviews just as uncomplicated as possible, since, otherwise, you may end up with reviewers who have no clue about where even to start.

Also, organizations need to show they recognize the value of code reviews since there's no question that they take away from the time developers could otherwise be using to create code. But if developers are rewarded only for adding functionality, that's going to end up crippling the code-review process, which in turn will almost certainly have an adverse effect on the maintainability of the code that's generated.

CB One thing I would like to add is that the code-review process we now have at Microsoft has more or less grown organically—through experimentation—from the grassroots. I mention this only because I think it might also work well for smaller companies, instead of having some process that's mandated from the top down.

Also, each product group at Microsoft does code reviews a little differently, with each group using its own set of policies that have essentially came together organically. While this probably won't come as a groundbreaking revelation, it can definitely be said that there's no one-size-fits-all solution for code reviews. This only serves to reinforce the importance of being willing to let your approach evolve organically such that it ends up fitting in with your work processes with the least amount of friction while putting the lightest burden possible on your developers.

Another important point is something Michaela talked about earlier, which is that treating code review as a first-class citizen—just as many companies are likely to treat

testing—is probably the best way to get the most bang for your buck. If, instead, it becomes something you're just expected to do, like flossing your teeth daily, then you'll find people aren't going to embrace it. But if you say this is important and so will be tracked and evaluated, then people are likely to respond to that. Certainly, that's how it has worked out here.

And then the other thing I would add is that it's instructive to think in some depth about what it is you're really looking to get out of code reviews. Then, of course, you should also think about how you can go about measuring that. To the degree that you can track those metrics and set targets, you're always going to achieve more.