# Code Reviewing in the Trenches: Understanding Challenges and Best Practices

Laura MacLeod
Microsoft

Michaela Greiler
Microsoft

Margaret-Anne Storey
University of Victoria

Christian Bird
Microsoft Research

Jacek Czerwonka
Microsoft

## 1. INTRODUCTION

Code review is a software practice that is widely adopted by and adapted to open source and industrial projects. Code review practices have been researched extensively, with most studies relying on trace data from tool reviews, augmented by surveys and interviews in a few cases. Several recent industrial research studies—in addition to blog posts and white papers—have exposed additional insights on code reviewing "from the trenches".

Unfortunately, the lessons learned about code reviewing are widely dispersed and poorly summarized by existing literature. In particular, practitioners wishing to adopt or reflect on a new or existing code review process may find it difficult to know which challenges to expect and which best practices to adopt for their specific development context.

Building on the existing literature, we add insights from a recent large-scale study of the code review practices of Microsoft developers to summarize the **challenges** faced by code change authors and reviewers, suggest **best practices** for code reviewing, and mention **trade-offs** that practitioners should consider.

## 2. CODE REVIEW STUDY

To understand code review processes, researchers generally focus on a retrospective analysis of code review trace data (e.g., CodeFlow [1], GitHub pull requests [2], and emails [3]). But some researchers have conducted interviews and/or surveys [2, 4] to reveal motivations and the challenges faced during code review. Bacchelli and Bird [1] further interviewed developers *while* they performed code reviews.

To gain a more in-depth understanding of the human and social factors that drive code review in a large industrial context, we observed and interviewed several teams at Microsoft. We complemented this with a survey to validate our initial findings about tools use, developer motivations, and the challenges faced with a broader set of developers. The survey was distributed to 4,300 developers and received 911 responses. Figure 1 summarizes the respondents' demographics.

For ethnographic style observations, we sat with four Microsoft teams for one week each to directly observe their code reviewing activities. The teams were comprised of new developers, senior developers, and team leads working on a range of projects—from new software to legacy systems—and a mix of internal and external products. We conducted semi-structured contextual interviews with 18 different developers from these four teams, either during or shortly after they performed a code review activity (bringing situated insights). Our observations of their code review activities allowed us to reveal cultural and social issues. Together with the interviews, we gained an understanding of how the teams approached code reviews and what policies they used.

This article comprises the main findings and lessons learned from our study that may be of interest to practitioners. A companion technical academic report [5] provides details about the observations, interviews, and surveys.

### Code review at Microsoft

Our survey and observations reveal that Microsoft's developers recognize the value of code reviews and feel it is an important activity. Developers appreciate reviewer feedback and are more thorough when they know their code is going to be reviewed. Whether they are a code author or reviewer, the process also helps them become more confident. Interestingly, not all teams have explicit rules or policies around code reviews and code review policies vary. Still, teams at Microsoft share a common code review life cycle that includes the following steps:

- **Preparation** of the code to be reviewed (by the author).
- **Selection** of reviewers (automatically or manually), with varying requirements for who should be selected and how.
- **Notification** of the selected reviewers as well as other stakeholders, with team policy dictating who should be informed and how.
- **Feedback** provided by reviewers to authors and other stakeholders.
- **Iteration** involving communication between the author and reviewer and further work by both.


*WOODSTOCK* '97 El Paso, Texas USA

- **Check-in** of the code change to the target system (in some teams, this happens before review).

These steps are performed by all teams, but the order can vary slightly depending on a team's policies, culture, and tools. A vast majority of engineers use an internal code review tool called CodeFlow [1]. This tool also supports and guides engineers through the reviewing steps. The most typical review cycle starts with the preparation of the code change by the code author. The author then selects the reviewer(s) (with some tool support) and the tool sends automatic notifications to selected reviewers. The reviewers then give feedback on the code change and a feedback notification is sent to the author. The author can then react on the received feedback and may iterate over the change. At some point, the author and the reviewer will be satisfied with the change and the change can be checked into the code base, or it will be rejected. Most communication between author and reviewer occurs through the code review tool, but other communication channels, such as face-to-face discussions, whiteboard sessions, video and voice chats, and IM, are used for contentious issue (i.e., issues that might reflect badly on someone) or to ensure a fast response. Almost all teams require a review before code can be checked in—a few teams allow exceptions, especially for trivial changes.

Microsoft engineers perform code reviews to improve code, find defects, transfer knowledge, explore alternative solutions, improve the development process, avoid build breaks, increase team awareness, share code ownership, and to assess the team (see Figure 1).

## 3. CODE REVIEWING CHALLENGES

Our interviewees and survey respondents reported a number of challenges (see bottom half of Fig. 1) which we discuss from two perspectives: the author of code to be reviewed, and the reviewer providing feedback. Organizational challenges are discussed in Section 5 as they mainly concern trade-offs that must be made. Some of those challenges are also reported by other researchers [6, 7, 8, 9] as discussed in our companion report.

### Challenges faced by code change authors

Authors of code changes discussed how it's hard **getting feedback in a timely manner**. The survey respondents listed this as their top code reviewing challenge.

> "Usually you write up some code and then you send it out for review, and then about a day later you ping them to remind them... and then about half a day later you go to their office and knock on their door." (Participant 7)

Another challenge concerned **obtaining insightful feedback** on code. Five interviewees mentioned that reviewers sometimes **focus on insignificant details** rather than looking for larger issues.

> "There is a lot of style [comments] a lot of the time, which I find annoying. And people will be like, *Maybe you should use this name?*" (Participant 7)

Participants mentioned that it's difficult **finding appropriate or willing reviewers**. And interviewees said that knowing who to ask is challenging as well.

| Demographics | |
|---|---|
| Total Respondents | 911/4,300 |
| Years of Experience in the Software Industry | 2+: 87%<br>6+: 70%<br>10+: 40 % |
| Time Employed at Microsoft | 2+ years: 72%<br>6+ years: 43%<br>10+ years: 17% |
| Experience Code Reviewing | 2-5 years: 80.3%<br>10+: 22% |

| Code Review Participation | | |
|---|---|---|
| During the previous week how often did you.. | How often do you act as a code reviewer? | How often do you author code reviews? |
| At least once per day | 39% | 17% |
| A couple times during the week | 36% | 48% |
| Once during the week | 12% | 21% |
| None | 13% | 14% |

| Motivations for Code Reviews | |
|---|---|
| Survey respondents were asked to pick and rank their top 5 reasons for doing code reviews. The below items ranked first are valued higher than the following ranks. | |
| Reason for Code Reviewing | Overall Rank |
| Code improvement | 1 |
| Find defects | 2 |
| Increase knowledge transfer | 3 |
| Find alternative solutions | 4 |
| Improve the development process | 5 |
| Avoid breaking builds | 6 |
| Build team awareness | 7 |
| Lead to shared code ownership | 8 |
| Team assessment | 9 |

| Challenges faced during code reviewing | |
|---|---|
| Survey respondents were asked to pick and rank their top 5 challenges during doing code reviews. The below items ranked first are valued higher than the following ranks. | |
| Challenge during Code Reviewing | Overall Rank |
| Receiving feedback in a timely manner | 1 |
| Review size | 2 |
| Managing time constraints | 3 |
| Understanding the code's purpose | 4 |
| Understanding the motivations for the change | 5 |
| Obtaining insightful feedback | 6 |
| Bikeshedding (disputing minor issues while more serious ones are overlooked) | 7 |
| Understanding how the change was implemented | 8 |
| Maintaining code quality | 9 |
| Reaching consensus | 10 |
| Finding relevant documentation | 11 |
| Managing multiple communication channels | 12 |
| Identifying who to talk to | 13 |

**Figure 1: Overview of selected responses from the code review survey.**

When preparing for a review, interviewees said they are unsure how to **document changes for review**. It was interesting that less than one third of respondents reported writing descriptions of the change when they prepared code for review, but that many more recognize it should be done more often and more thoroughly.

Some interviewees noted that receiving a **rejection can be harsh** and that they prefer being given a **reason** why a change is rejected. Others also noted that the feedback and discussion around **code review is ephemeral** and not easy to refer to later, especially if they use communication channels such as face-to-face rather than a code reviewing tool that maintains a history of discussions.

Richer channels may be preferred when trying to reach consensus about next steps, though some discussed how it can be tough managing **multiple communication channels**.

Some of our interviewees also stated that **tooling slows down code velocity** and tools should be modified to better suit the team's context, workflow, and policies.

## Challenges faced by code reviewers

Our code reviewers said they struggle with **large reviews**.

Participants discussed how it's hard **finding time** to perform all the code reviews requested of them, as well as **understanding the code's purpose, the motivations for the change, and how the change was implemented**. For code changes that are large and difficult to understand, one developer expressed frustration around the value of his review:

> "It's just this big incomprehensible mess... then you can't add any value because they are just going to explain it to you and you're going to parrot back what they say." (Participant 13)

Related to comprehension, **finding relevant documentation** about changes was another frequently reported challenge. One interviewee provided his thoughts on good documentation:

> "Typically [a good code review] has a good description of what the problem was, what the solution is, and if it's a big change, it has [documentation explaining] what it's doing and how it's integrated with everything else." (Participant 4)

From our interviews, we also learned that **understanding the history of comments** was an issue. Other challenges reported by survey respondents include a **lack of training** on the review process itself, and that their reviewing activities are perceived as **not being valued** enough. Some mentioned that they lack insights into how their code review activities **impact job evaluations**.

# 4. BEST PRACTICES

Our participants shared insights on how to avoid or mitigate some of the challenges they face. For example, one participant shared how to get reviewer buy-in:

> "Usually I try to get the person who I'm going to have review the thing to actually sit down and talk with them before I put out the code review" (Participant 7)

We synthesized insights from our survey into best practices for code change authors, code reviewers, as well as teams or organizations. These best practices are summarized in Fig. 2, organized by the different phases of the code review process (underlined below) and by stakeholder (author, reviewer, organization). Many of these best practices (shown in **bold**) have been suggested by other researchers who studied different development contexts, including open source projects [1, 7, 10, 11].

## Best practices for code change authors

To save reviewers time, while preparing a change for review, authors should **be conscientious** and **read through the change thoroughly** before sending it out for review. Viewing changes in a code review tool can expose simple issues (such as code style) to the author.

Authors should **aim for small, incremental changes** that are be easier to understand. This is especially important for novices whose understanding of the codebase can be superficial. Furthermore, **clustering related changes**, **documenting the motivation for a change**, and **describing the change and how to approach the review** will help reviewers. Authors should also take time to **test their changes**, and if no test exists, they should create one. **Running automated analysis tools** can expose formatting and low-level issues that would otherwise waste reviewers' time.

Finally, code change authors should carefully **consider when to skip a review** while referring to their organization's code review policy (if one exists). Many survey respondents suggested that reviews should be skipped for small or trivial changes that do not change the logic of the code, i.e., commenting or formatting issues, renaming of local variables, or stylistic fixes.

Once code has been prepared for review, authors need to select their reviewers. In particular, they need to carefully **decide how many reviewers** are needed, consulting their organization's policy if needed. Similar to the findings by Rigby *et al.*, our study participants explicitly recommended using two reviewers. It is important to **Select appropriate reviewers**—authors may select based on code expertise, they may select individuals who are responsible for the code, or they may choose reviewers to build expertise. If not against a team policy, it may be advisable to **allow reviewers to volunteer** for motivational reasons.

In addition, authors need to **consider who to notify**, choosing people that will benefit from being exposed to the code change and the resulting discussion, but they should also **decide who should NOT be informed**. Reducing the load for senior engineers was reported as an important consideration in our study. We also found that **notifying potential reviewers in advance** and explaining the upcoming change could help achieve buy-in and speed up reviews.

While responding to a review after their code has been reviewed, authors should **show gratitude to their reviewers** and carefully consider their feedback in a respectful manner. It is also important to **promote ongoing dialog with the reviewers** while **tracking and confirming problems are fixed** after receiving feedback.

Finally, when it comes time to commit code changes, authors should confirm that any **decisions made are documented**, and periodically **reflect on the process** as there

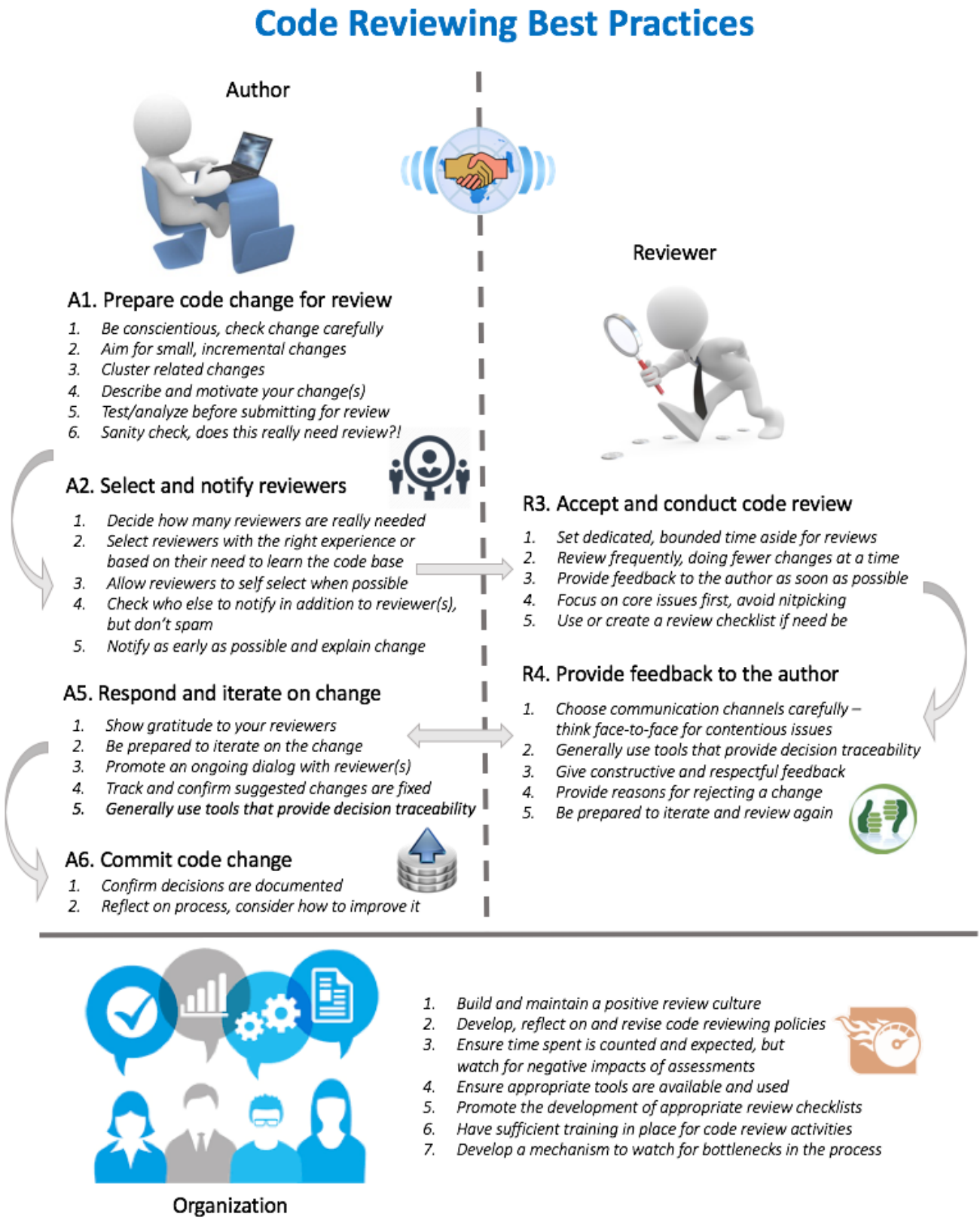


**Figure 2: An overview of the best practices we suggest for authors of code changes, reviewers of changes, and organizations. The diagram also shows the main steps of the code review life cycle.**

may be ways to improve their process and how they interact with their reviewers.

## Best practices for code reviewers

There are several best practices for reviewers to consider to help address the challenges they experience, but also to address author challenges. Although reviewers find it hard to find time to conduct their reviews, they should **set dedicated but bounded time aside** for reviewing, taking enough time to carefully understand the code of each review. Our participants also suggested that it is important to **review frequently** but review fewer changes at a time.

To help authors, it is important to **provide feedback as soon as possible** so that the authors will remember their change. It is also important to **focus on core issues first**, not wasting time on small problems at the expense of the design or logic problems. We further suggest to **create and use a review checklist** that is customized for the project's particular context.

While giving feedback on a review, reviewers should **choose communication channels carefully**. Richer channels, such as face-to-face or voice, are preferred for contentious issues or for discussing complex code changes. While for non-contentious or sensitive issues, **tools that provide traceability are preferred**. It is also an important reviewing skill to know how to **give constructive and respectful feedback** while also clearly **justifying and explaining the reasons for rejecting a change**.

## Best practices for organizations to consider

Whether a product team or company, how an organization sets the stage for reviewing activities and how it supports and values code reviewing is critical to the success of code reviews. We share the following quote from a participant to motivate the importance of best practices for organizations:

> "My team is currently pretty good with reviews, but we do not review our process or talk about the policies much at all. This means new people have to learn it the hard way and probably means there is a lack of consistency. This is a problem in our team dynamic that I don't think a tool can fix. On my team, this type of discussion falls into hygiene and I have to say, we are like street people." (Survey response to Q#34 - Entry 11)

To maximize the value of code reviews, an organization should consider establishing a **code review policy**. Such a policy should help in **building a positive review culture** that sets the tone for constructive review feedback and discussion.

The organization should also consider how to **ensure time spent reviewing is "counted" and "expected"** and is seen as an important part of the development life cycle. But the organization or team should **watch for negative impacts of employee assessment or incentives** that may be linked to code reviewing activities. While rewarding engineers who spend considerable effort reviewing others' code is encouraged, penalizing engineers who do not (often with a good reason) may lead to gaming of the system.

It is also important to **ensure that appropriate tools are used** and that they match the desired reviewing culture and defined process (if there is one). Tools may support certain steps in the process, such as finding and notifying reviewers, automating feedback, running style checkers, and testing. Reviewing tools should be lightweight and integrate well with other developer tools, especially with informal communication channels. Distributed teams may have additional tool needs. New tools for supporting code reviewing activities are emerging all the time and an organization may wish to stay abreast of these developments.

| To address author challenges: | Best practices that may help: |
|---|---|
| Receiving feedback in a timely manner | R3.3 |
| Receiving insightful feedback | R3.4, R3.5 |
| Find reviewers | A2.2, A2.3, A2.4, A2.5 |
| How to document changes for a review | A1.4, A1.5 |
| Dealing with harsh rejections | R4.3, R4.4, R4.5 |
| Code review decision traceability | R4.2, O4, A5.5 |
| Tooling slows things down | O4, O7 |
| Dealing with multiple tools | O4, O7 |
| **To address reviewer challenges:** | **Best practices that may help:** |
| Large reviews are hard to manage | A1.2 |
| Finding time to do reviews | A1.1, A1.4, A1.5, A1.6, A2.1, R3.1, R3.2 |
| Understanding a change and its motivation | A1.2, A1.3, A1.4 |
| Finding relevant documentation | A1.4, A1.5 |
| Understanding history of changes, decisions | O4 R4.2, A5.5 |
| Insufficient training for reviews | O6, R3.5 |
| Reviewing not valued enough | O1, O3 |
| Unsure of impact on job evaluations | O3 |

**Figure 3: Mapping suggested best practices to the reported code reviewing challenges.**

To address challenges concerning knowing the expected process or how to use desired tools, an organization can ensure there is **sufficient training** in place. Informal training through mentorship may be all that is required. Finally, an organization should encourage all stakeholders to **develop, reflect on, and revise** code reviewing policies and checklists. Organizations should continuously measure the impact of the policies and tools used on their overall output (speed of development, development efficiency, product quality, employee satisfaction); any discovered **bottlenecks should be resolved**, e.g., a policy can help reduce notification overload or define which reviews can be skipped.

# 5. TRADEOFFS TO CONSIDER WHEN APPLYING BEST PRACTICES

The practices we suggest for authors, reviewers, and organizations may help address the challenges that emerged from our study. In Fig. 3, we suggest which of the best practices may help address particular challenges. For example, authors that take time to carefully consider which code changes really need a review (see best practice A1.6 in Fig. 2) may save the reviewers' time. However, we acknowledge that not all of the practices may be applicable across all development or project contexts and that some of these practices may conflict with one another. Development teams face unique resource, time, and scope constraints that influence the choice of workflow and practices used. We discuss some of the inevitable **trade-offs** here.

When faced with time constraints, it may be necessary to trade-off **speed of the review over rigor**. For a blocking change, a code review should be done quickly to avoid impacting other developers' work, but only if the change does not impact a critical or consistently buggy part of the

system.

**Rigid policies**, such as always requiring two sign-offs or execution of a complete test suite, can lead to long delays in committing code. Developers, aware of the process burden, might avoid making the change, or will bundle it with others, causing reviews to become larger, less coherent, and harder to review. However, **lax or unclear policies** might reduce the value a team gets from code reviews.

Several trade-offs have to be considered when choosing **practices regarding reviewer selection**. Getting feedback from experts and senior developers must be balanced with several things. First of all, it may mean fewer opportunities for junior team members to learn and be mentored or fewer opportunities for knowledge dissemination while also distracting the senior developers from directly working on other coding tasks. Furthermore, requiring expert feedback might also create delays due to a lack of reviewer availability. Thus requesting less experienced reviewers can increase review speed and balance the team's workload. In terms of whether reviewers volunteer or not, reviewers who volunteer may be motivated to do a good job, but in some cases it may be more efficient to directly assign the review to experts rather than waiting for experts to self-select.

It may be prudent to trade **traceability of review activities with richer communication channels**. Particularly tense situations call for face-to-face discussions but these discussions are hard to capture and are rarely documented. In some situations, recording every decision might be required for legal compliance.

The policy and tools that **promote awareness can lead to notification overload**. A developer may want to notify a large group about a review, but overload leads to notifications being ignored. Likewise, the use of **sophisticated tooling may save or waste time**. Tools can automate some tedious tasks (e.g., check code formatting) but may incur huge costs for configuration and familiarization, or may even slow down processes (e.g., handling false positives of static analysis tools). Automation in the tool chain increases consistency but may lead to a feeling of loss of control.

In summary, the only way to manage these trade-offs is to be aware of them, to search for additional trade-offs, and to periodically evaluate not just workflow velocity and code quality but also the impact the practices have on developer satisfaction, personal goals, and team culture.

# 6. CONCLUDING REMARKS

Code review has been a popular research topic in the past few years and it continues to be an ongoing topic of importance to practitioners and researchers. Through this article, we aimed to gather insights from the dispersed research to date and add findings from a large industrial study where we closely observed and surveyed developers that author or review code changes. We presented the key challenges faced by authors and reviewers of code changes, and provided a number of suggested best practices for authors, reviewers, and organizations to consider that may alleviate these challenges. We discuss the inevitable trade-offs practitioners may face. We hope that these insights will be useful to researchers and practitioners alike as new tools, processes, and research emerge from our community.


## Acknowledgments

We thank our study participants, the CodeFlow team for their input on our research designs, and Cassandra Petrachenko for editing our paper.


# 7. REFERENCES


[1] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 712–721.

[2] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 345–355.

[3] P. Rigby, B. Cleary, F. Painchaud, M. Storey, and D. German, "Contemporary peer review in action: Lessons from open source development," *Software, IEEE*, vol. 29, no. 6, pp. 56–61, Nov 2012.

[4] V. K. Gurbani, A. Garvert, and J. D. Herbsleb, "A case study of a corporate open source development model," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 472–481.

[5] M. Greiler *et al.*, "Appendix to code reviewing in the trenches: Understanding challenges, best practices and tool needs," Microsoft Corp., Tech. Rep. MSR-TR-2016-27, May 2016, (http://research.microsoft.com/apps/pubs/default.aspx?id=266476). [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=266476

[6] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2014, pp. 119–122.

[7] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2013, pp. 202–212.

[8] M. Barnett, C. Bird, J. Brunet, and S. Lahiri, "Helping developers help themselves: Automatic decomposition of code review changesets."

[9] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes?: an exploratory study in industry," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 51.

[10] J. Cohen, E. Brown, B. DuRette, and S. Teleki, *Best kept secrets of peer code review*. Smart Bear, 2006.

[11] M. Petre and G. Wilson, "Code review for and by scientists," *arXiv preprint arXiv:1407.5648*, 2014.


# 8. AUTHOR BIOS

**Laura MacLeod** is program manager at Microsoft.

**Michaela Greiler** works as software engineer at Microsoft.

**Margaret-Anne Storey** is a Professor at the University of Victoria.

**Christian Bird** is a researcher at Microsoft Research.

**Jacek Czerwonka** works as a lead engineer at Microsoft.