Appendix to

Code Reviewing in the Trenches: Understanding Challenges, Best Practices and Tool Needs

Michaela Greiler,
Christian Bird,
Margaret Anne Storey,
Laura MacLeod,
and
Jacek Czerwonka

June 22nd, 2016
Technical Report
MSR-TR-2016-27

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Table of Contents

Abstract

This technical report is a companion document to the IEEE Software article "Code Reviewing in the Trenches: Understanding Challenges, Best Practices and Tool Needs". It is intended to give a thorough description of our study such that the article in IEEE Software can focus on results and insights that are most relevant to practitioners. In this report, we provide a more in-depth description of the methodology used to conduct our study of code review at Microsoft (see 1. Code Review Study) and share more detailed and comprehensive analyses of the survey results (see Appendix: In-Depth Survey Analysis and Appendix: Survey Slices). We also present the survey that we deployed in its entirety (see Appendix: Complete Survey), as well as many raw results (Appendix: Raw Results).

1. Code Review Study

Much of the previous research on code reviews focused on a retrospective analysis of code review traces, captured by one or more tools (e.g., Codeflow (Bacchelli and Bird), Github pull requests (Gousios, Pinzger and v. Deursen) or emails (Thongtanunam, Kula and Cruz)) who mostly conducted their research using open source projects. Less frequently, researchers directly communicated with developers, although some examples include surveys (Gousios, Pinzger and v. Deursen) (Gurbani, Garvert and Herbsleb). Mostly, those studies relied on a retrospective memory of what occurred to understand the motivation and challenges with code review. One notable exception is the research by (Bacchelli and Bird), as they conducted interviews during code reviewing activity at Microsoft. This work provides insights into motivations, outcomes and expectations that developers perceive about code reviewing but only report some high level recommendations for improving code review practices.

For our study, we wished to gain a more in depth understanding of the human and social dimensions that drive the code review lifecycle in a large industrial context, to uncover challenges experienced by developers and to reveal best practices that can be applied at different stages of the lifecycle and to identify pitfalls that should be avoided. We also wished to understand the broader constellation of tools and communication channels developers rely on during the different code reviewing activities and to consider not just the reviewers, but also the authors of the code to be reviewed, as well as other stakeholders such as team leads. Finally, we wished to uncover contextual factors that may influence the observed and reported best practices and tools used. As code reviewing is a socially situated activity (involving many stakeholders and influenced by numerous social factors), we followed an ethnographic approach (using observations and contextual interviews), complemented with a broad survey to discern if our findings from the observations of a select set of teams resonated with a much broader population of developers at Microsoft. In the following, we provide more details on the ethnographic observations and contextual interviews and the survey we conducted.

1.1 Ethnographic observations and interviews

We interviewed 18 developers from 4 different project teams and conducted observations with four other teams, sitting with each of these four teams for approximately one week each monitoring their code reviewing activities. The observations allowed us to witness

interactions concerning code review that would not be visible in the trace data captured by the code reviewing tools, and to witness cultural and social issues that may be difficult or even impossible to elicit through surveys and interviews. We further conducted semi-structured contextual interviews with 18 different developers from the four observed teams either while they were conducting a code review activity, or shortly thereafter (thus bringing situated insights).

For example, in three instances while sitting with developers working on a code review, we observed the conversations they had with teammates, asking questions to clarify their understanding of the code change. These cases highlighted the challenges developers face in trying to understand a code change, and the use of face-to-face communication as a quick method to get content and information. In a similar case while observing Team 3, the code reviewer couldn't ask the author questions about the change because the author was working from home for the day,

"If it was in person I would just ask him real quick because I don't want to do look up this code to go find out what does this do.... I could, but I would just ask him why does he need it?" (Participant 15)

In another conversation, we observed a code reviewer and author having a 90+ minute debugging session to walk through the edge cases and alternate approaches to a particularly complex problem in a code review. While the reviewer was satisfied with the end result, the author described the checked in solution as 'duct tape'.

The teams we observed consisted of between 5 and 14 developers, working on newer projects to legacy systems, which were a mix of products from internal use to external use. Three of the four teams collaborated with either remote team members or contract developers, and two of these collaborated with members in different time zones. Our interviewees were a mix of new developers, senior developers and managers. During these interviews, we asked questions about the developers' roles in the code review (whether they were the author or a reviewer), their experience during the review and their insights and opinions on the overall process. This elicited responses about team member relationships and their approaches to code reviewing.

"I have always the persona in my head of the Code Review Hammer. 'I am the code review guardian and no bug will get through my review and I'm gonna comment on everything'…But like as you get to know the developers you normalize, and you get to know what the developer's strengths and weaknesses are." (Participant 16)

Since the teams all used the same code review tool, CodeFlow – an internal and homegrown tool of Microsoft that supports a diverse set of code reviewing practices – common practices emerged. Change authors would create a code review, add a description with varying levels of detail about the change, and then choose code reviewers to send the review to.

Typically authors added one or two required developers to the review, and then included broader team mailing list as optional to give the rest of the team visibility. One interviewee described how knowing who to add to the review can be a difficult task for new developers:

"Its actually really hard for new people. New people email out and are like 'I'm modifying this thing. Please help. I don't know who owns this thing" (Participant 5)

The interviews and observations helped us understand how the teams approached code reviewing, the policies used, as well as elicit human and social factors not obvious from looking at trace data alone. In particular, we learned that code reviewing has a common lifecycle across all the teams which involves (in varying order depending on the policy):

- **Preparation of the code** to be reviewed (by the author),
- **Selection of reviewers** (automatically or manually, the number of reviewers and the requirements for who should be and how they should be selected depend on the policy and tools used),
- **Notification** of the selected reviewers and other stakeholders (where the policy may dictate who should be informed and how)
- **Feedback** provided by reviewers to authors and potentially other stakeholders (the tools and practices that are used depend on the policy and/or team culture)
- **Iteration** between the author and a reviewer, which may involve extensive or little communication between the two and further work by both the author and the reviewer.
- **Check-in** of the reviewed code to the target system. Note that for some team policies, code may be committed before review and in some cases review is an optional activity.

In addition to insights about the variability about how each phase of review is conducted, we gained initial insights into many challenges (technical and social) faced by authors and reviewers, as well as insights concerning the policies they follow and the tools they use. We used the findings from this phase of our study to design an in-depth survey which was sent to a much broader set of developers.

In this survey, we asked about

- the **demographics** of the respondents,
- the **team policies** used for code reviews,
- the **reviewing and communication tools** used, as well as which informal communication channels were used and why these were used,
- the **motivations** for conducting code reviews,
- **challenges** authors and reviewers faced,
- how diligent developers were at reviewing and the **steps authors took before submitting** code for a review.
- We also inquired about a number of **personal issues** such as if developers felt they were evaluated or judged based on their reviewing activity (or the reviewing of their code), and how reviewing influenced their relationships with other developers.

We discuss some background findings from the survey next, while other insights are mentioned throughout the remainder of the article. The interested reader can find the full survey in (Appendix: Complete Survey), as well as additional in-depth analysis of the survey responses in (Appendix: In-Depth Survey Analysis) and (Appendix: Survey Slices).

1.2 Survey Results

Participants for the survey were selected based on job title `Software Developer' and `Software Developer in Test' and they were contacted via email. Participation was optional

(with two prizes of nominal Amazon gift cards awarded as a small incentive). The survey was sent to 4300 developers and answered by 911 developers. First we share some details about the survey demographics, occurrence and frequency of code review activities, motivations for reviewing code, some background on different code reviewing policies used at Microsoft and some findings how social interactions shape code review and vice versa at Microsoft.

Demographics: Of the 911 developers, 87% of the respondents indicated that they had at least 2 years in the software industry. Figure 1 below provides an overview of the survey respondents demographics, and more information can be found in the Appendix. 70% had more than 6 years experience in the software industry, and 40% had more than 10 years of experience. Similarly, 72% of respondents worked for Microsoft for at least 2-5 years, and 43% for at least 6-10 years, and 17% had been at Microsoft for longer than 10 years. Most of the respondents (80.3%) who practice code reviewing had at least 2-5 years of experience with code reviewing, and 22% had more than 10 years of code review experience. A group of respondents (18.6%) reported that they had 6 or more years of experience in the software industry, but had been practicing code reviews for less than 2 years.

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

Figure 1: Demographics of Microsoft employees who responded to the survey.

Code reviewing occurrence and frequency: Most respondents indicate they review code at least once a day (39%) with 21% reviewing multiple changes per day. Whereas, 36% review changes a couple of times during the week. The remainder indicated they review changes once during the week (12%). 13% indicated they had not done a review in the week before the survey. Since our survey was focused on code reviewing, these percentages do not accurately reflect how many developers conduct code review at Microsoft as perhaps developers that do not do code review did not respond, but nevertheless it does demonstrate it is a broadly occurring activity. Respondents indicated they author code (to be reviewed) less often than they act as reviewer, with 17% authoring code to be reviewed at least once a day. Almost half (48%) said they author code that needs review a couple of times per week, and the rest authored code for review once during the week (21%). 14% did not author any code to be reviewed in the week before the survey.

Motivations for code reviewing: The motivations behind code reviewing have been reported by others (see (Bacchelli and Bird) paper for example). We found some similar results. We asked the respondents to rank reasons that are important to them for performing code reviews (see Figure 1). The top ranked reasons were code improvements and finding defects, followed by increased knowledge transfer, and finding alternative solutions.

In the free text, several respondents added additional or slightly different reasons to review code. One of the more frequent reasons given is to **teach junior** or less experienced developers. Similarly, several respondents indicated that **self-improvement** and learning is an important reason for code reviewing. Another reason is that code reviews allow the team to **develop a coding culture**, to develop **best coding practices** and to avoid anti-patterns or detect issues faster. Code reviewing therefore promotes coherent code bases. Similarly, several respondents indicate the need to **enforce a quality bar**, coding standards and style guidelines. Also, **increasing maintainability and readability** of the code was also mentioned numerous times. Another reason is to **build awareness** among the team, to inform others, as well as to get subject matter or area experts' opinions. Some respondents said that the effect of knowing that others look at the changes **increases code quality and accountability**. Code review was used as a tool to **perform design, security, architecture reviews** as well as a way to **support test planning and verify test coverage**.

Code Review Policies: Developers typically appreciate the value of code reviewing, with 94% of the survey respondents ranking it as either important (37%) or very important (57%). As noted previously, teams at Microsoft use varying policies for conducting code reviews, 94% of our respondents reported that their teams require a code review before check- in. In some cases, code review could be skipped (to avoid bottlenecks and thus increase code velocity), in other cases any code that is committed must be reviewed and rigorously tested. Other researchers have studied the impact of code review policies, specifically in open source (Rigby, Cleary and Painchaud).

However, the respondents were divided between those who indicated that their team has rules or policies around code reviews (54%) and those that had no explicit policies or rules in place (46%). Interestingly, fewer remote respondents indicated that a code review is needed before committing a code change (86.7% versus 94.1%), but this difference could be explained by other contextual factors rather than merely distance.

Code Review tools: Through the survey, we asked which tools are used to support their code reviewing activities. An internal tool, CodeFlow, was reported to be the most widely used code review tool at Microsoft (see Figure 2). This tool is well explained in (Bacchelli and Bird). While email is used by about 15% of developers, respondents also reported using a variety of other communication channels for code review-related tasks: face-to-face discussions, discussions at the whiteboard, video and voice chats, and IM. Email was the top reported choice for scheduling meetings (71.9%) and coordinating with other teams (65.3%). Face-to-face discussions were used by 61% to communicate issues that might reflect badly on someone else. For a fast response, face- to-face was the preferred method by 43.6%, while asking questions about the code being reviewed was done through the code reviewing tool (CodeFlow, GitHub, VisualStudio Team Services, and Atlassian).

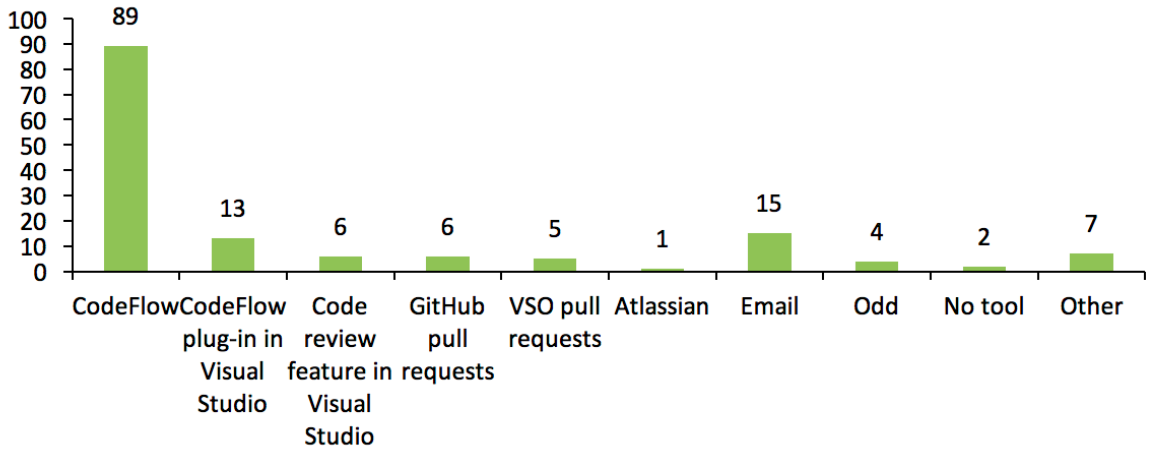


Figure 2: Percentage of survey respondents that reported using the various tools. The vast majority use CodeFlow (89%) and/or the CodeFlow plugin for Visual Studio (13%).

The social shaping of code reviewing: We asked participants a variety of questions related to the social interactions that shape code reviews. When authoring a code review 84.9% of respondents agreed or strongly agreed that they **appreciated** the feedback provided by reviewers. Respondents reported that having others review their changes **improved their confidence** (83.3% agreed or strongly agreed) and that they felt that they were **more thorough** because they knew that their work would be reviewed (75.9%). Respondents also reported that reviewing the changes of others improved their confidence (73.1%). When asked about whether they worry about others judging them or whether the personal relationships impact the code review process our respondents were split. 34% of the respondents indicated that they **worry about being judged**, and 31% indicated that their **personal relationships** impact their code review process. These responses demonstrate that the inter-personal relationships between developers have impact on the code review process, and should not be overlooked. We discuss later the importance of organizations taking these findings into consideration.

2. Code Review Challenges

Our survey respondents and the developers we interviewed and observed reported a number of challenges when requesting or performing code reviews. We detail these challenges from

two perspectives: an author of a change requesting a review and a change reviewer providing feedback. Many of these findings also reinforce the challenges reported by other researchers. Organizational challenges are discussed in Section 4, as they mainly concern tradeoffs.

2.1 Challenges faced by code change authors

The first challenge faced by the change authors is receiving feedback on their code in a timely manner. This was also listed as the top code reviewing challenge from respondents to the survey.

"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)

In addition, our participants reported that it was difficult finding appropriate or willing reviewers. Seven interviewees explained that knowing who to ask for review is challenging as well. Thongtanunam et al. also reported challenges faced finding expert reviewers (Thongtanunam, Kula and Cruz).

Another challenge mentioned by five developers we interviewed is **obtaining insightful feedback** on their code. The 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)

When preparing for a review, interviewed authors were troubled by **how to best document changes for review**. It was interesting that only 26% 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 was ephemeral** and not easy to refer to after the fact, especially if they use communication channels such as face-to-face rather than a code reviewing tool that maintains a history of discussion. Traceability of review activity was also reported as a challenge by (Rigby and Bird).

Richer channels may be preferred when trying to reach consensus about next steps though some discussed how it can be a challenge to **manage multiple communication channels**.

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

2.2 Challenges faced by code reviewers

Developers reviewing code changes made by their teammates struggle with **large reviews** (a challenge also reported by others (Barnett, Bird and Brunet) (Rigby, Cleary and Painchaud) (Tao, Dang and Xie). A team lead we interviewed explained how code review size was an issue for him:

“Yesterday I got a huge one that took at least an hour to even look over. Due to the sheer size, it’s really hard to see what's happening." (Participant 10)”

Additionally, our interviewees struggle with **finding time** to perform all the code reviews requested of them, as well as **understanding the code's purpose**, **understanding the motivations for the change**, and **understanding how the change was implemented**. For code changes that are 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, code reviewers reported challenges with **finding relevant documentation** about changes. This was brought up by 11 interviewees and also recognized in the survey. One interviewee provided his thoughts on what a good description of a change is:

``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 some survey respondents included a **lack of training** on the review process itself and that their reviewing activities were perceived as **not being valued** enough. Some also discussed that they lacked insights into how their code review activities **impact job evaluations**.

3. Best Practices

Our interviewees and respondents also shared ideas on how some of the challenges they discussed with us could be avoided or mitigated. To get a taste for the original data we distilled the practices from, see Figure 3 to see a quote for each perspective.

We summarize their responses into a set of suggested best practices categorized by practices for authors of code changes, code reviewers, and practices that the team or an organization could follow. Many of these best practices are also founded in the related literature (and thus are found in other development contexts including open source projects). Where appropriate, we reference existing literature to strengthen the case for the best practice. Note, we do not suggest that these practices will apply to all development contexts nor to all developers.

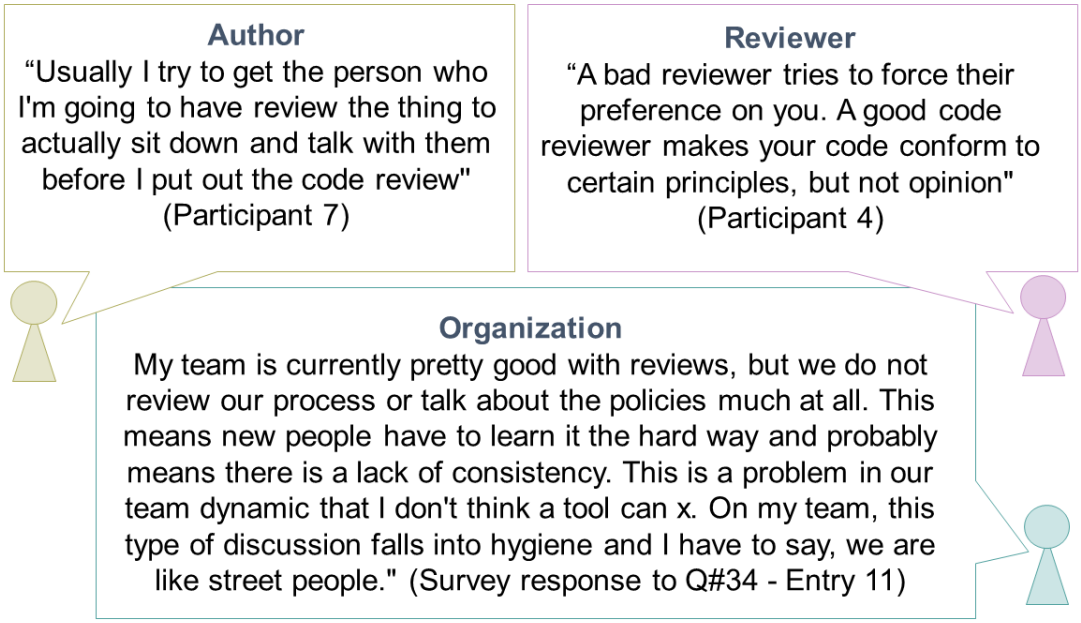


Figure 3: Excepts of best practices from "the trenches".

3.1 Author's perspective

For each task from preparing a change, over receiving feedback until submission of a change, authors can keep in mind the following best practices:

While preparing a change for review, authors should:

- **Be conscientious** and **read thoroughly through changes** before sending them out to be reviewed. Often seeing changes in the code review tool which visually presents them differently, makes it easier for the author to identify simple issues e.g. related to coding style. This best practice was also suggested by (Cohen, Brown and DuRette).
- **Aim for small, incremental changes**, and **easy to understand code changes**. Incremental changes are seen as especially important for novices whose understanding of the codebase can still be superficial. (Rigby, German and Storey) also noted this practice was followed in a number of open source projects.
- **Cluster related changes** and **submit the change including context** for the reviewers.
- **Document the motivation, annotate and describe the change** while providing reviewing directions to the reviewer. This practice was also suggested by (Bacchelli and Bird).
- **Test the change** before sending it out for review and if no test exists, **create a test** for the change.
- **Run automated tools** to check for formatting and low level issues (that can be caught through simple code analysis).
- **Know when to skip a review**, check the code review policy (if one exists) and confirm that the type of change should be sent out for review. Based on interviewees and survey respondents, typical reasons for skipping a review include small or trivial

changes that did not change the logic of the code, such as commenting or formatting issues, or renaming of local variables and stylistic fixes.

While selecting reviewers, authors should:

- **Decide how many reviewers** should be selected consulting a policy if one exists. Similar to the findings in Rigby et al., two of the leads we interviewed explicitly recommended two reviewers was their ideal.
- Select appropriate reviewers with the right expertise, or who are in charge of the code maintenance.
- **Allow reviewers to volunteer to perform a review** (if consistent with other policies). (Rigby, German and Storey) also suggested this best practice noting it was important for a number of open source projects.
- **Carefully select people to notify** who will benefit from being exposed to this code change and resulting discussion, i.e. engineers new to the project or teams heavily dependent on the project or the change being made. Also **decide who should not be informed**. Senior engineers especially are asked to participate in many reviews and lessening their reviewing burden would be advantageous,
- **Notify potential reviewers in advance** that a challenging or unexpected review may be forthcoming, earning their buy-in for such cases. If necessary, **explain the change**.

While responding to a review, authors should:

- **Show gratitude to the reviewers** and carefully consider their feedback in a respectful manner.
- **Promote dialog with the reviewers**. Use rich communication channels to discuss the reviews in progress if necessary, e.g., large or more complex changes.
- **Track and confirm problems are fixed** after receiving feedback.

Author best practices	
While preparing a change for review	✓ Be conscientious and read thoroughly through changes ✓ Aim for small, incremental changes, and easy to understand code changes. ✓ Cluster related changes and submit the change including context ✓ Document the motivation, annotate and describe the change ✓ Test the change, if needed create tests for the change ✓ Run automated tools ✓ Know when to skip a review
While selecting reviewers	✓ Decide how many reviewers to include ✓ Select appropriate reviewers with the right expertise ✓ Allow reviewers to volunteer to perform a review ✓ Carefully select people to notify ✓ Notify potential reviewers in advance
While responding to a review	✓ Show gratitude to the reviewers ✓ Promote dialog with the reviewers ✓ Track and confirm problems are fixed

Table 1: A summary of author code reviewing best practices from Section 3.1.

3.2 Reviewer's perspective

Also for reviewers several best practices should be considered. The goal of a good reviewer is to provide constructive feedback to a change. As one of the interviewees explained,

While conducting the review, reviewers should:

- **Set dedicated but bounded time aside** for reviewing, taking enough time to understand the code.
- **Review frequently**; fewer changes at a time but more often. This was also suggested by (Rigby and Bird).
- **Provide feedback as soon as possible** which will help with code velocity and capitalize on people's recent memory of the change.
- **Focus on core issues** first. The need to avoid emphasizing small problems at the expense of the design or logic problems was also discussed by Rigby et al. (Rigby, German and Storey)
- **Use a checklist for the review**, ideally one that is customized for the project's particular context. Cohen et al. (Cohen, Brown and DuRette) also suggested this best practice following their case study of code review at Cisco.

When giving feedback on the review, reviewers should:

- **Choose communication channels carefully**. Richer channels such as face-to-face meetings may be preferred for contentious issues or to review complex code that needs high degree of interaction to describe and understand.
- For non contentious or sensitive issues, **use tools that provide traceabilit**y.
- Be aware how to **give constructive** and **respectful feedback**.
- **Justify and explain the reason** for rejecting a change.

Reviewer best practices	
While conducting the review	✓ Set dedicated but bounded time aside for reviewing, ✓ Review frequently ✓ Provide feedback as soon as possible. ✓ Focus on core issues first. ✓ Use a checklist for the review
When giving feedback on the review	✓ Choose communication channels carefully ✓ For non-contentious or sensitive issues, use tools that provide traceability ✓ Give constructive and respectful feedback ✓ Justify and explain the reason for rejecting a change

Table 2: A summary of reviewer code reviewing best practices from Section 3.2.

3.3 Organizational perspective

How an organization (whether a product team or company) sets the stage for reviewing activities, and how it supports and values code reviewing, is critical to the success of code reviews. In the following are emerged best practices for the organization, once again stressing that these may not apply to all development contexts.

To maximize the value of code reviewing, a review policy that promotes the following practices should be established:

- **Build a positive review culture** that sets the tone for feedback style. The importance of reciprocity in the review process was also discussed by (Petre and Wilson) when they studied how scientific programmers review code.
- **Ensure time spent reviewing is ``counted'' and ``expected''** and that it is seen as an important part of the development life cycle.

- But also **watch for negative impacts of employee assessment or incentives** that may be based on or linked to code reviewing activity. While promoting engineers who spend considerable effort reviewing their peer's code is to be encouraged, penalizing engineers who do not (often with a good reason), will lead to gaming the metrics.
- **Ensure appropriate communication channels are used** that match the desired reviewing culture and that tools are widely adopted and integrate well with other tools. Keep in mind the needs of distributed teams. The need for richer communication channels was noted by (Bacchelli and Bird), whereby others have noted the need for light-weight tools (Barnett, Bird and Brunet), (Cohen, Brown and DuRette).
- **Ensure the automated tools support the desired reviewing process**: tools may enforce or help with certain steps such as finding reviewers, automating feedback. It is important that developers understand how automated tools are used and how they relate to explicit or informal policies.
- **Develop and constantly reflect and revise** on code reviewing policies and checklists. The organization should measure the impact of its policies and tools used on its overall output (speed of development, development efficiency, product quality and employee satisfaction).
- **Discovered bottlenecks should be resolved**, e.g. policy can help reduce notification overload or define which reviews can be skipped.
- **Ensure the reviewing process is free of bottlenecks** and leads to fast turnaround. The policy can help decide which reviews may be skipped (some reasons to skip were discussed above).
- Organizations should have a process in place for **identifying non appropriate or aggressive communication**.
- Organizations should ensure that there is **sufficient training** in place for code reviewing activities and associated tools. One approach here is for junior developers to work alongside senior developers during code review. (Petre and Wilson) also noted the importance of training.

Organizational best practices
✓ **Build a positive review culture**
✓ **Ensure time spent reviewing is ``counted'' and ``expected''**
✓ **Watch for negative impacts of employee assessment or incentives**
✓ **Ensure the automated tools support the desired reviewing process**
✓ **Develop and constantly reflect and revise** on code review processed
✓ **Ensure the reviewing process is free of bottlenecks** and resolve discovered bottlenecks
✓ Watch for **non appropriate or aggressive communication**
✓ **Ensure that there is sufficient training in place for code reviewing activities**

Table 3: A summary of organizational best practices from Section 3.3

4. Code Review Tradeoffs

The practices suggested above may not be applicable in all contexts and some even conflict with each other. All development teams face resource, time, and scope constraints that influence the choice of workflow and practices used. We discuss some of these trade-offs here.

When faced with time constraints, it may be necessary to choose **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 to 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 availability** of those reviewers. 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 assign the review to ideal experts 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 rarely documented. In some situations, recording every decision might be required for legal compliance.

The policy and tools **promoting 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.

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 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**.

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

5. Tool Needs and Opportunities

Tools play an important role in code review. (Rigby, German and Storey) studies investigated how email is used for broadcasting and conducting reviews; (Cohen, Brown and DuRette) studied how Code Collaborator enforces certain practices and collects metrics at Cisco; (Bacchelli and Bird) report how CodeFlow is used at Microsoft; and (Gousios, Pinzger and v. Deursen) and (Tsay, Dabbish and Herbsleb) report how pull requests support code review in open and closed source projects. These studies also touch on the fact that additional tools or communication channels are used in the code review process, although how these other channels are used is not investigated in depth.

At Microsoft, the most commonly used tool is CodeFlow. In the course of our study, participants shared with us their needs with regards to code review tools. Not surprisingly, respondents value **ease of use and performance of a tool**. **Integration with other services and tools** to reduce friction and help with the reviewing effort were also requested. In particular, integration with static analysis, testing and continuous integration tools was mentioned. Further, participants ask for integration of coding style plugins and search as well as features that ease **editing of code during review** and the ability to execute code that is being reviewed. Features were requested to aid in **describing a change to reviewers** such as ability to create a code change narrative, attach enough auxiliary information to the review to provide context, support **discussion during review**, integration with note-taking and documentation tools, where architecture may be described, templates for common code review comments. In general, seamless integration with external communication tools that are also used to support code review activities is needed. Participants wish for communication tools that integrate well with the code reviewing tool, supporting **informal communication** and **building awareness**. Features were also requested to help **manage reviews** with better notifications and tracking of the lifecycle of the feedback provided. Although this was not requested explicitly, we noticed the importance of **training and increasing awareness** because many of the requested features already exist in the tools used.

The research community has previously recognized improving tool support as an important way to address the code reviewing challenges. Our respondents also brought up many of the features previously reported:

- **enforce a reviewing workflow** to help engineers follow a team practice such as desired sign-off criteria or checklists (Cohen, Brown and DuRette), (Rigby, Cleary and Painchaud);
- help **find right experts** to review changes (Thongtanunam, Kula and Cruz);
- **show test coverage** of code under review (Morales, McIntosh and Khomh);
- **cluster changes** into groups of related entities for easier comprehension (Barnett, Bird and Brunet);
- **automate feedback** when consistent patterns emerge thus freeing reviewing time (Bacchelli and Bird);
- provide **support for traceability** of reviews (Bacchelli and Bird);
- provide **dashboards to show metrics and pending reviews** (Rigby and Bird).**;**
- reduce and manage **more pertinent notifications** (Rigby and Bird).**;** and
- support **integration with other communication tools** (Rigby and Bird).

We expect to see more elaborate tools becoming available for practitioners and open source developers in the near future. However, some of the trade-offs we mentioned above should be considered when such tools are selected or deployed. The tools will shape the practices that are used and vice versa. Studying impact of the tools on the workflow is paramount.

6. Conclusion

In this article, we reported on a large industrial study where we closely observed developers during code reviewing to bring situated insights, strengthened by a large follow-up survey. We relate our findings to other literature, bringing some order to the widespread suggestions published so far. We hope that these combined insights are useful to both practitioners and researchers and will improve future code reviewing activities.

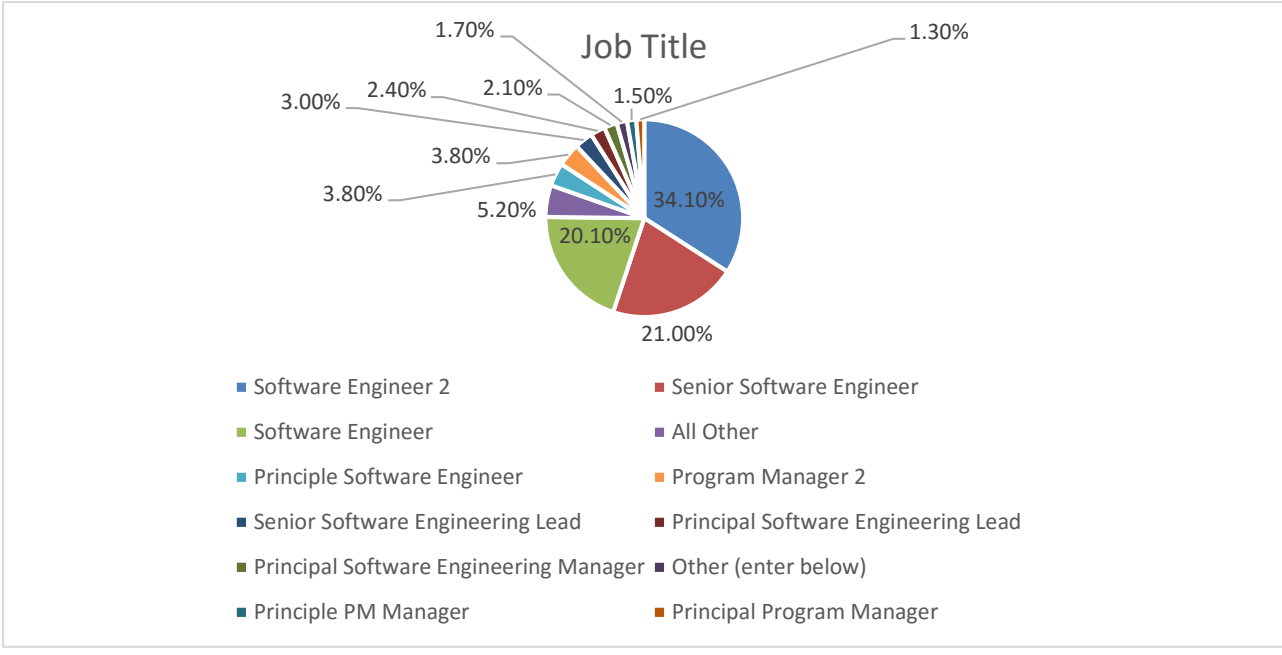
Appendix: In-Depth Survey Analysis

This document details the outcome of a survey concentrating on code review practices and communication during code reviewing. The survey was conducted by Laura MacLeod, Michaela Greiler, Chris Bird and Margaret-Anne Storey and was online in March 2015. 911 respondents shared their opinions about code reviewing, the challenges and its benefits. This section highlights aggregated data of all respondents who indicated to practice code reviewing.

Demographics

Job Title. Most of the respondents (~75%) are either Software Engineers (~20%), Software Engineers 2 (~34%) or Senior Software Engineers (~21%). The rest consist mostly of Principal Software Engineers, SE Leads, SE Managers as well as Program Managers (2 Principals). Details are illustrated in Table 1.

Table 1 Job title of respondents to the code review survey



Most of the managers (~82%) indicate to regularly participate in code reviews. Only few of the respondents manage other mangers (~7%).

The average team size is around 13 people, and the respondents indicate to work directly with 7 people on average.

Experience. 87% of the respondents indicate that they worked at least 2 years in the software industry. 70% more than 6 years, and 40% indicate to have more than 10 years of experience.

Similar, 72% indicate to work for Microsoft for at least 2-5 years, whereby 43% work at MS for at least 6-10 years. 17% indicate to work at MS longer than 10 years.

Most of the respondents (~80.3%) who indicate to practice code reviewing have at least 2-5 years of experience, whereby almost 22% indicate more than 10 years of code review experience.

Interestingly, many of the respondents who report not to practice code reviews are managers with a long experience in the industry.

Co-location. Most teams are completely co-located (73%). Only 11% of the respondents indicate that less than half or none of their team mates are close enough to get a coffee with them.

When it comes to the people respondents interact with during code reviews, we see that code review teams are more distributed than the actual team of the respondents (see Table 2 and Table 3). Still, most respondents indicate to be collocated with at least half of their peers who they interact on code reviews (86%), whereby roughly half of all respondents have all their peers close enough to get a coffee with them (48%). Only 4% indicate to have none of their peers they interact during code review near them.

Table 2 Co-location of team: Of the people you work with on a daily bases what percentage of those people work near you?

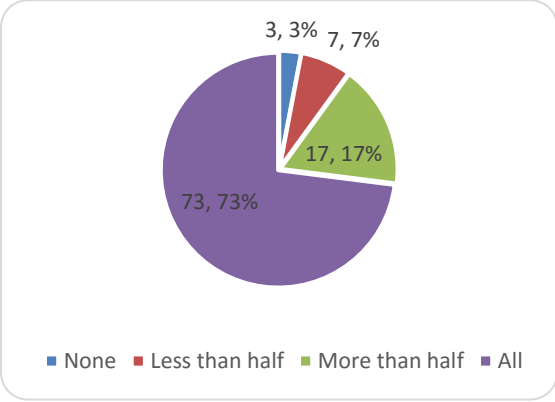
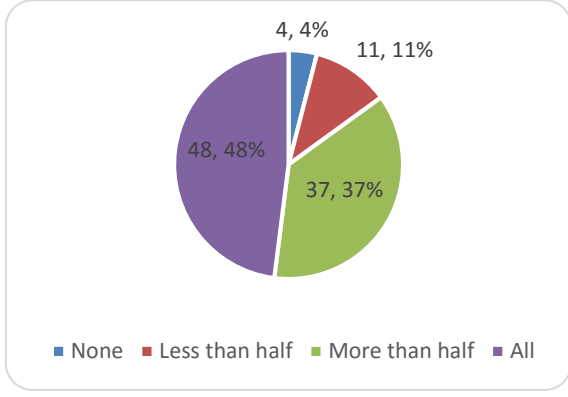


Table 3 Table 2 Co-location of code review team: Of the people you work with on code reviews what percentage of those people work near you?

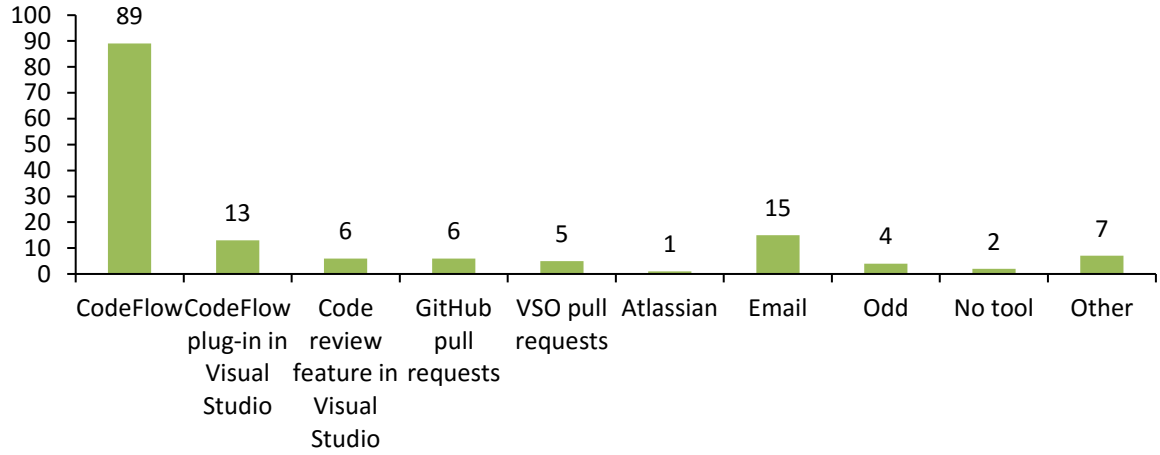


Technical set-up

SourceDepot is still the predominant version control system within the selected population (64%), followed by TFS (41%), and Git with 29%. Other version control systems only account for 4%.[1]

Code review tool usage. A large majority of the respondents (89%) indicate to use CodeFlow as their code reviewing tool. This is followed by Email used as code review tool (15%) and the CodeFlow extension (13%) in Visual studio. Details are illustrated in Table 4. In the category of Other: 2% use Collaborator from SmartBear, and 5% use one of the 30 other named tools.

Table 4 Code review tool usage



Development practices

A majority of the respondents indicate to use an agile development process (77%) or to practice Scrum (69%). Also, 68% indicate to use automated tool support for code checkins like Checkin Wizard.

On the other hand, only 16% indicate to practices pair programming, and even fewer (8%) say they have a formal training on code review practices.

Code reviews

Frequency of performing code reviews. Most respondents indicate to review changes of others at least once a day (39%), whereby 21% review even multiple changes per day. The other large group indicates to review changes a couple of times during the week (36%). The rest indicated to review changes once during the week (12%), or that they did not act as a review during the last week (13%).

Naturally, respondents indicate to author code reviews less often than they act as reviewer. Here, 17% indicate to author code reviews at least once a day, and of those only 5% says they author several code reviews per day. Almost half (48%) say they author code reviews

[1] Percentages don't add up to 100% as many respondents use more than one source control solution.

couple of times during the week, and the rest either indicates to have authored a review once during the week (21%) or that they did not act as a review author in the last week (14%).

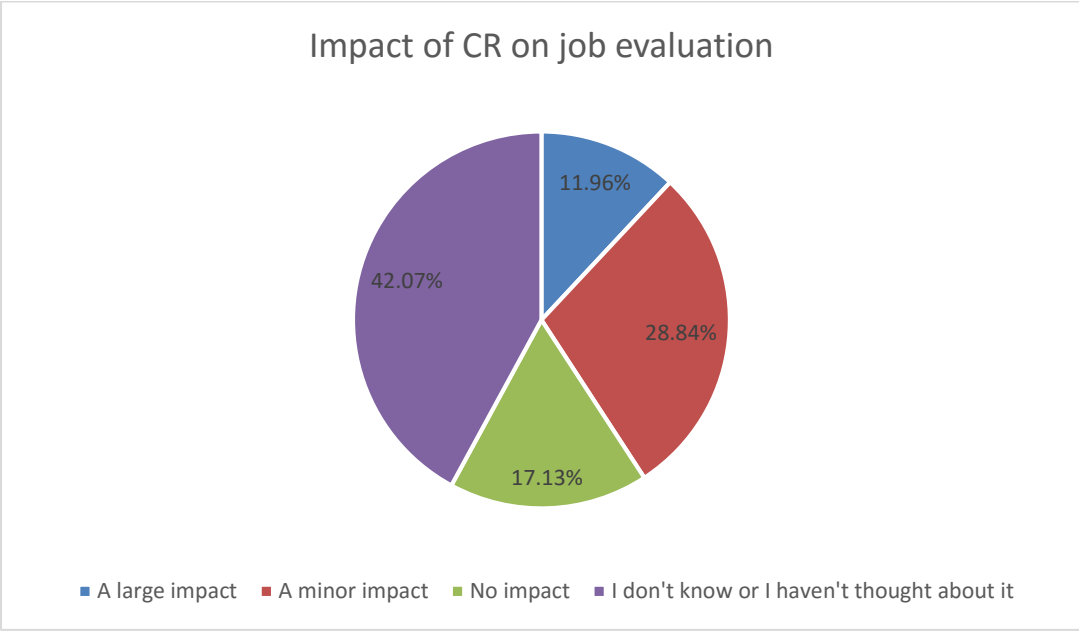
Importance of code reviews. 88% indicate that code reviewing is seen by their team as important or even very important (43%). Only 3% say that their team perceives code review as unimportant or very unimportant.

If they reflect on their own attitude towards code review, respondents paint an even more positive picture. 94% of the respondents indicate that they perceive code reviewing as very important (57%) or important (37%). Only 6% are either neutral (3%) or perceive code review as unimportant or very unimportant.

Policies. It became very clear that most teams require a code review before a code change can be checked in (94%). Also, 84.1% indicate that they have mechanisms in place to keep team members aware of each other's code reviews. On the other hand, respondents are split between those that indicate that their team has rules or policies around code reviews (54%) and those that indicate they have no policies or rules in place (46%). Similar 52% indicate that their team reflects on their code review process, and 48% say they do not.

Code review impact. A large portion of the respondents indicate that the do not know or haven't thought about to what degree their performance in code reviews impacts their job evaluation (42%) (see Table 5). Also, 29% indicate this has a minor impact, and even 17% think it has no impact on their job evaluation. Only 12% think it plays a large impact for their job evaluation.

Table 5 Perceived impact of code reviewing on job evaluation



Reasons for code reviews. The respondents had to rank several reasons that are important to them for performing code reviews as listed in detail in Table 6. The top ranked reasons were code improvements, followed by increased knowledge transfer, and finding alternative solutions.

Table 6 Ranked reasons for code reviewing

Reason for code reviewing	Score*	Overall Rank
Code improvement	2835	1
Find defects	2749	2
Increase knowledge transfer	1528	3
Find alternative solutions	1199	4
Improve the development process	979	5
Avoid breaking builds	957	6
Build team awareness	790	7
Lead to shared code ownership	717	8
Team assessment	235	9

Score is a weighted calculation. Items ranked first are valued higher than the following ranks, the score is the sum of all weighted rank counts.

In the free text, several respondents added additional or slightly different from the pre-defined reasons to review code. One of the reasons that came up most often for performing code reviews is to teach junior or less experienced developers, and let them learn from more experienced developers on the team. Slightly different but on the same track, several respondents indicated that self-improvement, learning and improvement of coding skills is an important reason for code reviewing. Another often named reason to perform code reviews is that code reviews allow the team to develop a coding culture, be exposed to what is seen as best practice within the team, and to learn new coding patterns and to avoid anti-patterns or detect issues. Code reviewing therefore allows to build coherent solutions and code bases. Similarly, several respondents indicate the need to enforce a quality bar, coding standards, enforce clean code and style guidelines. Also, increasing maintainability and readability of the code was also among the often appearing answers.

Another often expressed reason is to build awareness among the team, inform others as well as to get subject matter or area experts' opinions. Therefore, code reviews also help to put the change into perspective, i.e., to get the bigger picture. Some respondents said that the effect of knowing that others look at the changes increases code quality and accountability.

Code review as a tool to perform design, security and architecture reviews and therefore improve the code with respect to those areas was also mentioned. Also testing, especially verifying test coverage and supporting test planning was mentioned as reasons for code reviews. Few respondents said that code reviewing helps them to transition from SDETs to SEs.

Skipping code reviews. More than 400 respondents answered this free text from question on when code review can be skipped. Around 5-7% indicate in their answers that code reviews should never be skipped. During the analysis of the answer for reasons to skip code reviews several common opinions emerged. First, the most common reason respondents believe code reviewing can be skipped is for **small, trivial or minor changes**. The definition of small or minor deviates obviously, but a common understanding of a small, trivial or minor change is that it does **not change the logic of the code**, but addresses things like typos in comments, formatting issues, renames of local variables, removal of dead code, changes to string literals or style issues. Others are more liberal with their definition of small and mainly go by **lines of code touches**. Here very often respondents indicate that one line changes, or changes touching only few lines can be checked in without prior code review. Others think that such small changes should be code reviewed by over the shoulder reviewing, so less formally than through a tool chain.

Another very frequent occurring reason for skipping code reviews are build breaks. Here, some respondents explicitly mention the time pressure of the **build break** as an additional factor for permitting skipping the code review, whereby others such focus on the size of the fix (i.e., if it is small or well-defined then skipping a CR is okay). Also quite a few respondents talk about emergency situation, including build breaks, hot fixes during odd times or issues with live sites where the **time aspect has priority** and code reviews can be skipped. Some indicate to do after the fact code reviews for changes that are related to time critical issues.

Integrations, FIs/RIs, merges without conflicts or code moves appear among the changes that many respondents indicate as valid for skipping code review.

Also several respondents say that **configuration changes** do not necessarily have to be reviewed. Here, some indicate general configuration changes, whereby others explicitly state that the changes to the configuration must be small and/or well understood.

Other situations that several respondents feel permit skipping code review are changes to code that is **non-production code, private code, prototypes, internal tools or test code**. Few also talk about low-priority parts of the code base, and that changes in those areas might skip code review.

Also code that has been developed during **pair programming** can be permitted into the code base without additional code reviewing.

Another situation which permits skipping code review in the opinion of several respondents is if the **author of the change is the subject matter expert** or the only person knowledgeable in the area or with this part of the code base. Slightly related, some respondents think that code review can be skipped if the change is small and the **developer is confident** that the change is low risk, safe and does not break anything or that **the fix is well known**.

Another category of changes that allow skipping code review has to do with the **type of the change**. Many respondents indicate that non code changes (like changes to binaries, packages, markup or data) can skip a code review. A few respondents also think version number changes, script changes, changes related to logging or build can be skipped. Also some indicate that changes to the UI that cannot break the build can be skipped during code review.

Changes that only **roll back or revert a previous change** can also be skipped according to the opinion of some respondents.

Some respondents talk about that **changes that have been discussed before** with team member or the team lead or that were reviewed otherwise can skip the formal code review process.

Also time constraints like **deadlines and tight schedules** might lead to a skip of code reviewing practices.

Less frequent named reasons for skipping code reviews are if the code is well covered with and **verified by automated tests**, if the change happens in **legacy code**, the code is the same between **several platforms** or branches.

Very few indicate to only perform code reviews for very complex or large changes.

Challenges. The five main challenges developers face during code reviewing are receiving feedback in a timely manner, the review size, managing time constraints and understanding the code's purpose (see Table 7). Other higher ranked challenges are understanding the motivation for the change, obtaining insightful feedback and disputing minor issues while more serious ones are overlooked.

Table 7 Ranked challenges faced during code review

Challenges faced during code reviewing	Score*	Overall Rank
Receiving feedback in a timely manner	1944	1
Review size	1406	2
Managing time constraints	1250	3
Understanding the code's purpose	1243	4
Understanding the motivations for the change	962	5
Obtaining insightful feedback	917	6
Bikeshedding (disputing minor issues while more serious ones are overlooked)	883	7
Understanding how the change was implemented	687	8
Maintaining code quality	686	9
Reaching consensus	548	10
Finding relevant documentation	501	11
Managing multiple communication channels	315	12
Identifying who to talk to	286	13

Score is a weighted calculation. Items ranked first are valued higher than the following ranks, the score is the sum of all weighted rank counts.

When it comes to **acting as a reviewer**, the majority of respondents (73%) indicate that reviewing changes of others improves their confidence as programmers, as can be seen from Table 8. Also 80% believe that they are thorough when looking through changes of others and 89% say that the feel their feedback is respected and that the author considers the feedback.

A less clear picture emerges from answers regarding relationships and judgmental behavior during code reviewing. Here, around half of the respondents (53%) indicate that they do not worry about others judging their abilities as programmers during reviewing. 20% are neutral, 22% agree and 5% strongly agree that they worry about having their abilities judged during code reviewing.

Respondents are split between whether or not the personal relationships with those involved in review have an impact on the code review. 44% believe this is not the case, whereby 34% believe that their personal relationships do impact code reviews, and 22% are undecided[2].

Table 8 Acting as a reviewer: Perception results

[2] Detailed results about respondents' perceptions as reviewer can be found in the appendix Table 20 Acting as a reviewer: Detailed perception resultsTable 20.

	Strongly disagree or disagree	Neutral	Agree or strongly agree
When reviewing, I worry about others judging my abilities as a programmer.	52.50%	20.20%	27.40%
It improves my confidence as a programmer when I review the changes of others.	6.00%	21.00%	73.10%
I am thorough when I review the work of others.	2.20%	18.00%	79.80%
As a code reviewer I feel that my feedback is respected.	1.70%	9.60%	88.70%
My personal relationships with those involved in a review have an impact on my code review.	43.70%	22.40%	33.80%
I am confident that the author considers my feedback.	2.10%	9.30%	88.70%

As a **review author**, almost all respondents (96%) indicate that they appreciate the feedback of the reviewers, as depicted in Table 9. Also, the majority of the respondents claim to express appreciation to reviewers (85%), indicate that reviewing improves their confidence (83%) and that they learn a lot when others review their code (78%). Also, 76% indicate that they are more thorough because they know that the code will be reviewed. On the other hand, a less clear picture emerges when respondents are asked about whether they worry about being judged by others and whether or not the personal relationships impact the code review process. Here, 34% of the respondents indicate to worry about being judged, and 31% indicate that the personal relationships impact the code review process[3].

We can observe that respondents indicate that the appreciate feedback they receive as authors more positive, as they perceive that their feedback is respected during performing code reviews. Also as review authors, respondents indicate a slight higher concern about judgments of their skills then when acting as reviewers. Nevertheless, the observed differences are indeed small.

Table 9 Acting as an author: Perception results

	Strongly disagree or disagree	Neutral	Agree or strongly agree
As a review author, I appreciate the feedback I receive from reviewers.	0.70%	3.20%	96.10%
When others review my changes, I worry about them judging my abilities as a programmer.	44.00%	22.50%	33.50%
It improves my confidence when others review my changes	2.80%	13.90%	83.30%

[3] Detailed results about respondents' perceptions as authors can be found in the appendix Table 21.

I feel that I am more thorough because I know my code will be reviewed.	8.90%	15.20%	75.90%
My personal relationship to reviewers has an impact when I author a code review.	42.00%	27.30%	30.70%
I express thankfulness to those who review my code.	3.10%	12.00%	84.90%
I learn a lot when other developers review my code.	3.90%	17.60%	78.40%

Additional resources. To gather additional information relevant to code reviews, respondents indicate to use the following three resources most often: contact the review author (49% often, 10% always, 33% sometimes), look at the source code history in the repository (32% often, 39% sometimes and 7% always), and look at source code not in the code review (40% sometimes, 30% often, and 6% always).

On the other hand, the following three resources are not used or are used sparingly: 1) mailing lists (43% never, 29% rarely), 2) style guides (29% never, 33% rarely) and 3) design documentation (27% never, 33% rarely). More details can be found in the appendix in Table 22.

Table 10 Additional resources used during code review

Resources	Never or rarely	Sometimes	Often or always
Bug reports	49.00%	32.10%	18.90%
Contacting the review author	8.00%	33.20%	58.90%
Contacting subject experts (besides the author)	47.10%	35.30%	17.70%
Source code not in the review	23.90%	40.20%	35.90%
Design documentation	60.00%	27.30%	12.60%
Mailing lists	71.70%	20.50%	7.80%
Style guides	62.00%	25.80%	12.10%
Source code history in the repository	23.10%	38.50%	38.40%

Communication channel choices per task. For getting a fast response, F2F discussions (44%) and IM (38%) are the tool of preference for the respondents. Details are shown in Table 11. Especially if there are issues that might reflect badly on someone, F2F communication is preferred by 61% or the respondents compared with all other options. Whereby the code review tool is the tool of choice ([38%-48%]) for asking questions, either about the code change, its history or the reason for the change. The second ranked choice for asking questions is the F2F discussion [24-26%]. To reach a consensus, negotiate a change or find alternative solutions, respondents chose to use F2F discussions ([33-36%]) as well as the code review tool ([27-38%]). Email is the tool of choice for coordination tasks such as

scheduling a meeting (72%) or coordinating with other teams (65%). Voice or video chat as well as telephone are almost never used by respondents.

Table 11 Communication channel choice for certain tasks

	Code review tool	F2F discussion	F2F discussion at a whiteboard	Video or voice chat	Telephone	Email	IM	Responses
Get a fast response	7.20%	43.60%	3.70%	1.80%	2.00%	4.20%	37.60%	764
Explore alternative approaches	27.30%	32.50%	23.60%	1.20%	0.30%	11.10%	4.10%	758
Communicate issues that may reflect badly on someone	8.60%	61.00%	5.70%	1.10%	0.50%	12.00%	11.20%	753
Reach a consensus	33.60%	32.80%	14.30%	2.20%	0.80%	12.20%	4.10%	760
Schedule a meeting	1.90%	11.10%	3.50%	2.50%	0.70%	71.90%	8.50%	750
Coordinate with other teams	13.30%	8.50%	4.30%	2.50%	0.50%	65.30%	5.60%	645
Negotiate changes	38.10%	35.80%	11.10%	1.70%	0.00%	8.60%	4.80%	651
Ask questions about the code in general	44.20%	25.80%	4.30%	0.60%	0.00%	13.50%	11.50%	651
Ask questions about the history of the code	38.40%	26.30%	2.60%	1.20%	0.20%	16.80%	14.50%	649
Ask questions to understand a change	48.10%	24.20%	6.40%	1.40%	0.20%	8.40%	11.30%	653
Ask questions to understand the reasons for a change	45.60%	25.90%	4.40%	0.90%	0.50%	9.40%	13.30%	652

Not all tasks are faced equally often as highlighted in Table 12. Regarding which tasks the respondents face most often during code reviews, the most often ask a question about the change (45% often, 8% always), reach a consensus (37% often, 10% always) and get a fast response (39% often, 5% always). On the other hand, the rarely or never schedule a meeting (28% never, 52% rarely), communicate issues that may reflect badly (15% never, 51% rarely), and coordinate with other teams (40% rarely, 10% never). More details can be found in Table 23.

Table 12 Frequency of tasks faced during code reviewing

Tasks	Never or rarely	Sometimes	Often or always
Get a fast response	12.70%	43.10%	44.30%
Explore alternative approaches	14.60%	58.60%	26.80%
Communicate issues that may reflect badly on someone	65.90%	28.20%	5.90%
Reach a consensus	14.50%	39.10%	46.40%
Schedule a meeting	79.80%	17.10%	3.10%
Coordinate with other teams	50.20%	37.90%	12.00%
Negotiate changes	22.50%	50.50%	27.00%
Ask questions about the code in general	15.90%	42.90%	41.10%
Ask questions about the history of the code	44.80%	40.00%	15.20%
Ask questions to understand a change	7.60%	39.50%	52.90%
Ask questions to understand the reasons for a change	9.80%	45.60%	44.70%

Before sending out a code review, the majority of the respondents (65%) indicate to always read through their changes looking for errors, and 48% also always run the tests. In total, 92% indicate to always or often read through changes, 79% run tests often or always before sending out the review, and about half indicate to often or always write tests for a change. Even though respondents indicate the importance of writing a detailed description about the change, only 26% of them indicate to always follow this practice. Still, roughly half of the respondents indicate to write a detailed description either often or sometimes. 17% indicate to never or rarely write such a description. Respondents are split almost evenly on whether or not they give their peers a heads-up on the change to review (35% sometimes, 33% rarely or never, and 32% often or always). The practice less often used is to run static analysis. Here, 44% indicate that the never (25%) or rarely (19%) run static analysis before sending out a code review. The results are highlighted in Table 13and more details can be found in the appendix in Table 24.

Table 13 Tasks performed before sending out a code review

Before sending out a review	Never or rarely	Sometimes	Often or Always
Read through the changes looking for mistakes	2.80%	5.10%	92.20%
Write a detailed description of the code to be reviewed	17.00%	28.30%	54.60%
Get advice from subject matter experts	21.70%	40.40%	37.90%
Give reviewers a heads-up about the review	32.70%	35.40%	31.90%
Run static analysis	44.30%	16.20%	39.50%

Run tests	8.70%	12.50%	78.80%
Create tests	17.80%	28.90%	53.30%

Increase feedback speed. Almost 500 developers used the free text format to express their opinion on how to increase the feedback speed for code reviews. Among the many answers, few very clear categories emerged. The most common suggestion of respondents was to **contact the code reviewers**. Here, they either mentioned to ping or remind the reviewers about the review either F2F, or by IM, email or phone. The also suggested to organize a short code review meeting, and/or to let the reviewers know in advance that they are needed for a code review. Several said that you have to ping early and often or/and set reminders.

Another very frequent occurring suggestion is to **improve the code review** or the code review package. Improvement suggestions include to do small, incremental code reviews, to be rigorous about providing a good description, title, and eventually add comments to explain some code changes. In general, respondents highlighted the need to explain the reason, the background and the motivation for the change to the reviewers.

Another coherent category is the need to **build the right team culture** and perception about code reviews. Respondents stress that code review must be an essential part of the development process, and this includes that it can account for time and also is rewarded. Several respondents say that code review must be seen as top priority and acted upon (i.e., code reviews are done immediately).

Several respondents also expressed the need to **ask the right reviewers** to review the code. This means to include people that are knowledgeable about the area, but also that have a stake or interest in the code change. Also several respondents stress that it is important to only include few reviewers on the code review and avoid sending out to whole teams or mailing lists.

The last very frequent occurring suggestion was to **review fast yourself** (i.e., be part of the solution not the problem).

Appendix: Survey Slices

Distributed teams versus collocated teams

Remote respondents are slightly more experienced with code reviewing i.e., they indicate less often to have less than 2 years of experience and to not practice code reviewing. Also, they indicate to have worked slightly longer in software industry, but appear to have similar working times at MS.

Remote respondents said that specific practices are used less often, in particular practices like scrum or agile methods. Remote managers indicate to participate less often in code reviews, than their collocated counterparts (68% vs. 81%).

Regarding the importance of code reviews, the remote respondents rate the importance of code reviews slightly less high in their teams' perception than collocated respondents. The same is true for their own opinion on code review importance.

Also, remote participants say less often that a code review is needed before checking in (86.7% versus 94.1%).

Respondents that work remote from their team say that they use email more often as code reviewing tool than the overall population (27% versus 15%).

Interestingly, even though less than half or even none of the immediate team works near the respondents, 10% say that more than half of the people the interact with during code review are near them, and another 5% say that all people the code review with are near them.

Distributed respondents rank "Understanding the motivations for the change" as the second most occurring challenge during code review. For collocated teams this seems less troublesome and only appears on rank 7. Also, distributed respondents rank "Understanding the code's purpose" higher than "managing time constraints" – differing from collocated teams.

Naturally, when choosing the "tool" of choice for several tasks related to code review, remote participants count more on IM, the Code review tool and Email than on F2F discussions. F2F discussions are only the main tool to communicate issues that may reflect badly on others. To reach a consensus most participants use the code review tool, and also 13% of the participants use video conversation. Remote respondents also indicate to use IM (22-23%) and Email (20-29%) much more frequent to ask questions about a code review than collocated teams which prefer the code review tool (40-50%) and F2F conversations (25-28%).

Interestingly, remote respondents indicate to worry less about others judging their abilities as programmers when reviewing other people changes (64% remote respondents disagree to worry vs. 51% that are collocated) and also indicate that they are less worried about others judging their ability as programmers when sending out code review (59% remote respondents disagree versus 43% or collocated) (see Table 14 and Table 15).

Remote respondents also indicate to more frequently express thankfulness than their collocated counterparts (93% vs. 84%).

Remote respondents believe less that code reviewing makes them more thorough during coding (64% vs. 77% agree to be more thorough).And they also indicate to be less thorough when reviewing changes of others (68% vs 81% agree).

We tested the effects of remoteness for both, either the team of the respondent is not near or the people that are on code reviews are not near the respondent. We could see similar effects for both populations.

Table 14 Distributed versus Collocated respondents' perception about reviewing others changes

Distributed Respondents

	Strongly	Neutral	Agree

Collocated Respondents

	Strongly	Neutral	Agree or

	disagree or disagree		or strongly agree
When reviewing, I worry about others judging my abilities as a programmer.	63.51%	17.57 %	18.92%
It improves my confidence as a programmer when I review the changes of others.	4.11%	13.70 %	82.19%
I am thorough when I review the work of others.	1.37%	30.14 %	68.49%
As a code reviewer I feel that my feedback is respected.	0.00%	9.59%	90.41%
My personal relationships with those involved in a review have an impact on my code review.	44.59%	28.38 %	27.03%
I am confident that the author considers my feedback.	1.35%	9.46%	89.19%

	disagree or disagree		strongly agree
When reviewing, I worry about others judging my abilities as a programmer.	51.27%	20.54 %	28.19%
It improves my confidence as a programmer when I review the changes of others.	6.37%	21.81 %	71.81%
I am thorough when I review the work of others.	2.26%	16.69 %	81.05%
As a code reviewer I feel that my feedback is respected.	1.84%	9.75%	88.42%
My personal relationships with those involved in a review have an impact on my code review.	43.79%	21.75 %	34.46%
I am confident that the author considers my feedback.	2.13%	9.22%	88.65%

Table 15 Distributed versus Collocated respondents' perception as author

Distributed Respondents

	Strongly disagree or disagree	Neutral	Agree or strongly agree
As a review author, I appreciate the feedback I receive from reviewers.	0.00%	1.43%	98.57%

Collocated Respondents

	Strongly disagree or disagree	Neutral	Agree or strongly agree
As a review author, I appreciate the feedback I receive from reviewers.	0.74%	3.27%	95.99%

When others review my changes, I worry about them judging my abilities as a programmer.	59.42%	21.74%	18.84%
It improves my confidence when others review my changes	0.00%	13.04%	86.96%
I feel that I am more thorough because I know my code will be reviewed.	11.59%	24.64%	63.77%
My personal relationship to reviewers has an impact when I author a code review.	40.00%	31.43%	28.57%
I express thankfulness to those who review my code.	0.00%	7.14%	92.86%
I learn a lot when other developers review my code.	0.00%	21.43%	74.29%

When others review my changes, I worry about them judging my abilities as a programmer.	42.56%	22.62%	34.82%
It improves my confidence when others review my changes	3.27%	13.82%	82.91%
I feel that I am more thorough because I know my code will be reviewed.	8.59%	14.37%	77.04%
My personal relationship to reviewers has an impact when I author a code review.	42.35%	26.89%	30.76%
I express thankfulness to those who review my code.	3.56%	12.46%	83.98%
I learn a lot when other developers review my code.	4.30%	17.36%	78.34%

Impact in the job evaluation

Respondents that say that code review has no impact on their job evaluation are also less likely to practice some software methodologies such as scrum (63% vs. 73%), or agile development (70% vs. 80%) compared with respondents that think code reviewing has a large impact on their job evaluation. They also use less frequently automated tool support for checkins (62% vs. 71%).

Respondents that think CR has no impact on their job evaluation ($Respondents_{no}$) also indicate that code reviewing is seen as less important than the respondents that think CR has a large impact ($Respondents_{large}$). 78% of the $Respondents_{no}$ say that code reviewing is important (51%) or very important (27%), versus 94% of the $Respondents_{large}$ say that CR is very important (67.3%) or important (26.5%) in their teams perspective.

Similarly, when judging their own attitude torwards code reviewing, we see a significant shift in perceived importance between $Respondents_{no}$ and $Respondents_{large.}$ 73% of the $Respondents_{large}$ say that code reviewing is very important, compared to 43.5% of $Respondents_{no.}$ Most other $Respondents_{no}$ (47%) say it is important, compared to 21% of $Respondents_{large.}$

As to be expected, respondents that say code review has no impact on their job evaluation also report less rigorous practices around code reviews (as highlighted in Table 1Table 16).

Table 16 Slice Impact on Job evaluation: differences between code review process

CR has a large impact on job evaluation			**CR has no impact on job evaluation**		
	Yes	No		Yes	No
Does your team subscribe to rules or policies for conducting code reviews?	61.90%	38.10%	Does your team subscribe to rules or policies for conducting code reviews?	50.40%	49.60%
Does a code change normally require a code review before it can be checked in?	95.90%	4.10%	Does a code change normally require a code review before it can be checked in?	84.80%	15.20%
Does your team have mechanisms to keep team members aware of each other's reviews?	89.70%	10.30%	Does your team have mechanisms to keep team members aware of each other's reviews?	77.50%	22.50%
Does your team review and reflect on their code review process?	69.10%	30.90%	Does your team review and reflect on their code review process?	35.80%	64.20%

$Respondents_{no}$ also participated less frequently in code reviews during the last week, both as authors and as reviewers. Whereby 32% of the $Respondents_{large}$ say the reviewed multiple times a day, only 17% of the $Respondents_{no}$ indicated to do so, and 10% of $Respondents_{large}$ acted as a author compared with 3% $Respondents_{no}$. Also 19% of $Respondents_{no}$ say they did not act as a reviewer compared with 9% of $Respondents_{large}$. 20% $Respondents_{no}$ say they did not act as an author compared to 12% $Respondents_{large.}$

Respondents who indicate that code review does not have an impact on job evaluation, also are less likely to experience that their confidence is improved when the review changes of others (63% vs. 82%), they are less thorough when reviewing the work of others (63% vs. 82%), and slightly feel that their feedback is less respected (83% vs. 89%). See Table 17 for more details. Also, they indicate to learn less during code reviewing, to express thankfulness less often and are less likely to indicate that it improves their confidence when others review their changes (for details see Table 18).

Table 17 Slice impact on job evaluation: perception as reviewer

CR has a large impact on job evaluation	CR has no impact on job evaluation

	Strongly disagree or disagree	Neutral	Agree or strongly agree
When reviewing, I worry about others judging my abilities as a programmer.	56.52%	14.13%	29.35%
It improves my confidence as a programmer when I review the changes of others.	4.35%	14.13%	81.52%
I am thorough when I review the work of others.	1.09%	13.04%	85.87%
As a code reviewer I feel that my feedback is respected.	4.35%	6.52%	89.13%
My personal relationships with those involved in a review have an impact on my code review.	40.22%	18.48%	41.30%
I am confident that the author considers my feedback.	4.35%	6.52%	89.13%

	Strongly disagree or disagree	Neutral	Agree or strongly agree
When reviewing, I worry about others judging my abilities as a programmer.	53.03%	21.21%	25.76%
It improves my confidence as a programmer when I review the changes of others.	10.69%	26.72%	62.60%
I am thorough when I review the work of others.	4.55%	21.21%	74.24%
As a code reviewer I feel that my feedback is respected.	2.27%	14.39%	83.33%
My personal relationships with those involved in a review have an impact on my code review.	37.88%	28.79%	33.33%
I am confident that the author considers my feedback.	3.08%	11.54%	85.38%

Table 18 Slice impact on job evaluation: perception as author

CR has a large impact on job evaluation

	Strongly disagree or disagree	Neutral	Agree or strongly agree
As a review author, I appreciate the feedback I receive from reviewers.	1.15%	4.60%	94.25%

CR has no impact on job evaluation

	Strongly disagree or disagree	Neutral	Agree or strongly agree
As a review author, I appreciate the feedback I receive from reviewers.	1.60%	2.40%	96.00%

When others review my changes, I worry about them judging my abilities as a programmer.	43.02%	27.91%	29.07%
It improves my confidence when others review my changes	2.30%	5.75%	91.95%
I feel that I am more thorough because I know my code will be reviewed.	4.65%	17.44%	77.91%
My personal relationship to reviewers has an impact when I author a code review.	48.28%	21.84%	29.89%
I express thankfulness to those who review my code.	1.15%	6.90%	91.95%
I learn a lot when other developers review my code.	4.60%	11.49%	83.91%

When others review my changes, I worry about them judging my abilities as a programmer.	48.00%	23.20 %	28.80%
It improves my confidence when others review my changes	4.00%	24.00 %	72.00%
I feel that I am more thorough because I know my code will be reviewed.	12.90%	12.90 %	74.19%
My personal relationship to reviewers has an impact when I author a code review.	34.68%	29.03 %	36.29%
I express thankfulness to those who review my code.	7.32%	11.38 %	81.30%
I learn a lot when other developers review my code.	6.45%	27.42 %	66.13%

Respondents that do not see an impact of their performance during code review on their job evaluation are less likely to write a thorough description of the change, to get advice from subject matter experts, to give reviewers a heads-up about the review, or to create tests before sending out the review (see for Table 19 details).

Table 19 Slice impact on job evaluation: tasks before sending code review

CR has a large impact on job evaluation				**CR has no impact on job evaluation**			
	Never or rarely	Some-times	Often or		Never or rarely	Some-times	Often or

			always
Read through the changes looking for mistakes	2.38%	7.10%	90.48%
Write a detailed description of the code to be reviewed	14.29%	20.20%	65.48%
Get advice from subject matter experts	14.12%	40.00%	45.88%
Give reviewers a heads-up about the review	23.53%	36.50%	40.00%
Run static analysis	37.65%	17.60%	44.71%
Run tests	7.06%	11.80%	81.18%
Create tests	14.12%	22.40%	63.53%
Build and run changes	2.53%	3.80%	93.67%

			always
Read through the changes looking for mistakes	4.20%	5.90%	89.92%
Write a detailed description of the code to be reviewed	20.17%	34.50%	45.38%
Get advice from subject matter experts	30.51%	33.10%	36.44%
Give reviewers a heads-up about the review	36.97%	38.70%	24.37%
Run static analysis	47.06%	11.80%	41.18%
Run tests	10.92%	10.90%	78.15%
Create tests	18.49%	26.10%	55.46%
Build and run changes	5.56%	1.10%	93.33%

Appendix: Raw Results

In this section, the interested reader can find more details on the raw results for many of the discussed survey sections.

	Strongly disagree	**Disagree**	**Neutral**	**Agree**	**Strongly agree**	**Responses**
When reviewing, I worry about others judging my abilities as a programmer.	17.70%	34.80%	20.20%	22.00%	5.40%	779
It improves my confidence as a programmer when I review the changes of others.	1.50%	4.50%	21.00%	51.20%	21.90%	778
I am thorough when I review the work of others.	0.10%	2.10%	18.00%	59.40%	20.40%	779
As a code reviewer I feel that my feedback is respected.	0.50%	1.20%	9.60%	61.50%	27.20%	780
My personal relationships with those involved in a review have an impact on my code review.	14.30%	29.40%	22.40%	26.10%	7.70%	781

I am confident that the author considers my feedback.	0.40%	1.70%	9.30%	59.40%	29.30%	778

Table 20 Acting as a reviewer: Detailed perception results

Table 21 Acting as a review author: Detailed perception results

	Strongly disagree	**Disagree**	**Neutral**	**Agree**	**Strongly agree**	**Responses**
As a review author, I appreciate the feedback I receive from reviewers.	0.30%	0.40%	3.20%	41.20%	54.90%	743
When others review my changes, I worry about them judging my abilities as a programmer.	14.20%	29.80%	22.50%	25.90%	7.60%	741
It improves my confidence when others review my changes	0.80%	2.00%	13.90%	52.00%	31.30%	742
I feel that I am more thorough because I know my code will be reviewed.	2.30%	6.60%	15.20%	47.30%	28.60%	744
My personal relationship to reviewers has an impact when I author a code review.	14.40%	27.60%	27.30%	24.60%	6.10%	743
I express thankfulness to those who review my code.	0.80%	2.30%	12.00%	51.60%	33.30%	744
I learn a lot when other developers review my code.	0.70%	3.20%	17.60%	47.80%	30.60%	744

Table 22 Additional resources used during code review: Detailed results

Resources	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**	**Total**
Bug reports	19.40%	29.60%	32.10%	17.00%	1.90%	100% (747)
Contacting the review author	3.10%	4.90%	33.20%	49.10%	9.80%	100% (754)
Contacting subject experts (besides the author)	17.20%	29.90%	35.30%	16.20%	1.50%	100% (746)
Source code not in the review	7.90%	16.00%	40.20%	29.70%	6.20%	100% (744)
Design documentation	26.80%	33.20%	27.30%	10.70%	1.90%	100% (746)
Mailing lists	42.80%	28.90%	20.50%	7.30%	0.50%	100% (743)
Style guides	29.40%	32.60%	25.80%	10.00%	2.10%	100% (751)
Source code history in the repository	6.70%	16.40%	38.50%	31.60%	6.80%	100% (749)

Table 23 Frequency of tasks faced during code reviewing: Detailed results

Tasks	Never	Rarely	Sometimes	Often	Always	Responses
Get a fast response	1.90%	10.80%	43.10%	39.00%	5.30%	641
Explore alternative approaches	1.10%	13.50%	58.60%	24.30%	2.50%	643
Communicate issues that may reflect badly on someone	15.30%	50.60%	28.20%	4.80%	1.10%	642
Reach a consensus	1.60%	12.90%	39.10%	36.60%	9.80%	644
Schedule a meeting	27.60%	52.20%	17.10%	2.80%	0.30%	644
Coordinate with other teams	10.00%	40.20%	37.90%	11.20%	0.80%	642
Negotiate changes	1.70%	20.80%	50.50%	24.50%	2.50%	644
Ask questions about the code in general	0.60%	15.30%	42.90%	34.70%	6.40%	645
Ask questions about the history of the code	4.80%	40.00%	40.00%	14.10%	1.10%	645
Ask questions to understand a change	0.60%	7.00%	39.50%	45.10%	7.80%	643
Ask questions to understand the reasons for a change	1.10%	8.70%	45.60%	38.30%	6.40%	643

Table 24 Tasks performed before sending out code review: Detailed results

Before sending out a review	Never	Rarely	Sometimes	Often	Always	Total
Read through the changes looking for mistakes	1.40%	1.40%	5.10%	27.20%	65.00%	725
Write a detailed description of the code to be reviewed	4.50%	12.50%	28.30%	28.90%	25.70%	727
Get advice from subject matter experts	8.10%	13.60%	40.40%	28.90%	9.00%	726
Give reviewers a heads-up about the review	14.40%	18.30%	35.40%	21.30%	10.60%	727
Run static analysis	25.10%	19.20%	16.20%	16.80%	22.70%	728
Run tests	3.60%	5.10%	12.50%	30.40%	48.40%	727

Create tests	6.50%	11.30%	28.90%	32.30%	21.00%	727

Appendix: Complete Survey

For the purposes of completeness and replication, we provide the complete text from the survey deployed for this study below.

We are researchers from the Tools for Software Engineers team and Microsoft Research investigating the code review work practices of developers at Microsoft. We would be greatly appreciative if you would be willing to answer the following questions. The survey shouldn't take more than 15 minutes.

This survey is completely anonymous and all questions are optional. No personal information is required for particpation in this survey. If you have any questions or if you'd rather not participate and want no further contact, please email Laura MacLeod or Christian Bird. For survey participation, we are also hosting a raffle for two $50 Amazon gift cards. Instructions for the raffle appear after participants submit their responses.

We invited participants by randomly selecting employees at Microsoft that fit our demographic criteria such as their role at Microsoft. We are interested in hearing from employees who have experience with code reviews (as either an author of changes, a reviewer of changes or both). If you do not participate in code reviews, we ask that you still complete the first two questions.

Demographics
The following questions ask about your background and role within Microsoft.

1) What is your title?

2) How many years have you practiced code reviewing?

( ) I do not practice code review ( ) Less than 2 ( ) 2-5 years

( ) 6-10 years ( ) More than 10 years

If you do not participate in code reviews, please scroll to the bottom of the survey and click submit so that we can still get your answers to the first two questions. Thanks!

3) If you are a manager, please answer the following questions.

	Yes	**No**

Do you regularly participate in code reviews?	( )	( )
Do you manage other managers?	( )	( )

4) How many years have you worked in the software industry?

( ) Less than 2 ( ) 2-5 years ( ) 6-10 years ( ) More than 10 years

5) How many years have you worked at Microsoft?

( ) Less than 2 years ( ) 2-5 years ( ) 6-10 years ( ) More than 10 years

Team Demographics
The following questions ask about your team's characteristics.

6) How many people make up your immediate team (including yourself)?

7) Of the number you listed above, how many people on your team do you directly work with?

8) Of those people, what percentage work near you? (i.e. you could get a cup of coffee with them)

( ) None

( ) Less than half

( ) More than half

( ) All

9) What version control system does your team currently use? (Please check all that apply)

[ ] TFS

[ ] SourceDepot

[ ] Git

[ ] Other: _____

10) For the following table, please indicate if your team implements any of the following practices

	Yes	**No**
Pair programming	( )	( )
Uses automated tool support for code check-ins (e.g., a Checkin Wizard).	( )	( )
Provides formal training on code reviews practices	( )	( )
Scrum	( )	( )
An agile development process	( )	( )

11) Based on your experiences, which of the following best describes your team's attitude towards code reviews?
They consider it to be:

( ) Very unimportant ( ) Unimportant ( ) Neutral ( ) Important ( ) Very important

Team Code Reviews
The following questions ask about your team's code review practices.

12) Please answer if your team does any of the following:

	Yes	**No**
Does your team subscribe to rules or policies for conducting code reviews?	( )	( )
Does a code change normally require a code review before it can be checked in?	( )	( )
Does your team have mechanisms to keep team members aware of each other's reviews?	( )	( )
Does your team review and reflect on their code review process?	( )	( )

13) What code review tools does your team currently use? (Choose all that apply)

[ ] CodeFlow

[ ] CodeFlow plug-in in Visual Studio

[ ] Code review feature in Visual Studio

[ ] GitHub pull requests

[ ] VSO pull requests

[ ] Atlassian

[ ] Email

[ ] Odd

[ ] No tool

[ ] Other: _____

14) Of the people you work with on code reviews (either as an author or reviewer of changes), what percentage of those people work near you? (i.e. you could get a cup of coffee with them)

( ) None

( ) Less than half

( ) More than half

( ) All

15) To what degree does your performance in code reviews impact your job evaluation? It has:

( ) A large impact ( ) A minor impact ( ) No impact

( ) I don't know or I haven't thought about it

Code Reviews
The next questions ask about why you do code reviews and for your opinions on the process.

16) Based on your experience, which of the following best describes your attitude towards code reviews?

( ) Very unimportant ( ) Unimportant ( ) Neutral ( ) Important ( ) Very important

17) Why do you do code reviews? Below is a list of **reasons developers do code reviews**. Please choose and rank your top 5 items (with 1 being the most important).

_____Avoid breaking builds

_____Code improvement

_____Lead to shared code ownership

_____Find defects

_____Find alternative solutions

_____Improve the development process

_____Build team awareness

_____Increase knowledge transfer

_____Team assessment

18) If you have other important motivations you wish to share, please briefly explain them below and indicate their level of importance.

19) Do situations occur where you find code reviews can be skipped? If so, briefly describe those situations.

20) Below is a list of challenges developers face in code reviews.
Please choose and rank your top 5 challenges to code reviews (with 1 being the greatest challenge).

_____Understanding the motivations for the change

_____Review size

_____Understanding the code's purpose

_____Finding relevant documentation

_____Identifying who to talk to

_____Obtaining insightful feedback

_____Understanding how the change was implemented

_____Receiving feedback in a timely manner

_____Managing time constraints

_____Maintaining code quality

_____Reaching consensus

_____Bikeshedding (disputing minor issues while more serious ones are overlooked)

_____Managing multiple communication channels

Code Reviewing
The following question asks about your thoughts and actions as a reviewer on code reviews (reviewing changes, not authoring them).

For the next set of questions we want you to think about the recent code reviews you have been a part of in the past week and reflect on those experiences. If you did not act as a reviewer, please answer the next question and skip the rest of the questions in this section.

21) In the past week, how often did you act as a reviewer on code reviews (reviewing changes, not authoring them).

( ) I did not act as a reviewer

( ) Once during the week

( ) A couple times during the week

( ) At least once a day

( ) Multiple times a day

22) To what degree the following statements align with your recent experiences as a reviewer on code reviews.

	Strongly disagree	**Disagree**	**Neutral**	**Agree**	**Strongly agree**
When reviewing, I worry about others judging my abilities as a programmer.	( )	( )	( )	( )	( )
It improves my confidence as a programmer when I review the changes of others.	( )	( )	( )	( )	( )
I am thorough when I review the work of others.	( )	( )	( )	( )	( )
As a code reviewer I feel that my feedback is respected.	( )	( )	( )	( )	( )
My personal relationships with those involved in a review have an impact on my code review.	( )	( )	( )	( )	( )
I am confident that the author	( )	( )	( )	( )	( )

considers my feedback.					

23) Thinking about your actions as a reviewer on code reviews over the past week, how often do you make use of the following resources to gather additional information relevant to code reviews?

	Never	**Rarely**	**Sometimes**	**Often**	**Always**
Bug reports	( )	( )	( )	( )	( )
Contacting the review author	( )	( )	( )	( )	( )
Contacting subject experts (besides the author)	( )	( )	( )	( )	( )
Source code not in the review	( )	( )	( )	( )	( )
Design documentation	( )	( )	( )	( )	( )
Mailing lists	( )	( )	( )	( )	( )
Style guides	( )	( )	( )	( )	( )
Source code history in the repository	( )	( )	( )	( )	( )

24) Generally as a reviewer on code reviews, indicate which communication channel you would turn to first to do the following tasks:

	Code review tool	**Face to face discussion**	**Face to face discussion at a whiteboard**	**Video or voice chat**	**Telephone**	**Email**	**IM**
Get a fast response	( )	( )	( )	( )	( )	( )	( )
Explore alternative approaches	( )	( )	( )	( )	( )	( )	( )
Communicate issues that may reflect badly on someone	( )	( )	( )	( )	( )	( )	( )
Reach a consensus	( )	( )	( )	( )	( )	( )	( )

Schedule a meeting	( )	( )	( )	( )	( )	( )	( )
Coordinate with other teams	( )	( )	( )	( )	( )	( )	( )
Negotiate changes	( )	( )	( )	( )	( )	( )	( )
Ask questions about the code in general	( )	( )	( )	( )	( )	( )	( )
Ask questions about the history of the code	( )	( )	( )	( )	( )	( )	( )
Ask questions to understand a change	( )	( )	( )	( )	( )	( )	( )
Ask questions to understand the reasons for a change	( )	( )	( )	( )	( )	( )	( )

25) Generally as a reviewer on code reviews, indicate the frequency of which you find yourself doing the tasks mentioned above:

	Never	**Rarely**	**Sometimes**	**Often**	**Always**
Get a fast response	( )	( )	( )	( )	( )
Explore alternative approaches	( )	( )	( )	( )	( )
Communicate issues that may reflect badly on someone	( )	( )	( )	( )	( )
Reach a consensus	( )	( )	( )	( )	( )
Schedule a meeting	( )	( )	( )	( )	( )
Coordinate with other teams	( )	( )	( )	( )	( )
Negotiate changes	( )	( )	( )	( )	( )
Ask questions about the code in general	( )	( )	( )	( )	( )

Ask questions about the history of the code	( )	( )	( )	( )	( )
Ask questions to understand a change	( )	( )	( )	( )	( )
Ask questions to understand the reasons for a change	( )	( )	( )	( )	( )

Code Authoring
The following question asks about your thoughts and actions as an author of code reviews (submitting changes for others to look at).

For the next set of questions we ask you to think about the recent code reviews you have been a part of in the past week, and to reflect on those experiences. If you did not act as an author, please answer the next question and skip the rest of the questions in this section.

26) In the past week, how often did you act as an author of code reviews (submitting changes for others to look at).

( ) I did not act as an author

( ) Once during the week

( ) A couple times during the week

( ) At least once a day

( ) Multiple times a day

27) Thinking as an author of code reviews this past week, how often did you do any of the following before you sent out changes for review?

	Never	**Rarely**	**Sometimes**	**Often**	**Always**
Read through the changes looking for mistakes	( )	( )	( )	( )	( )
Write a detailed description of the code to be reviewed	( )	( )	( )	( )	( )
Get advice from subject matter experts	( )	( )	( )	( )	( )

Give reviewers a heads-up about the review	( )	( )	( )	( )	( )
Run static analysis	( )	( )	( )	( )	( )
Run tests	( )	( )	( )	( )	( )
Create tests	( )	( )	( )	( )	( )
Build and run changes	( )	( )	( )	( )	( )

28) To what degree do you agree with the following statements based on your recent experiences as an author of code reviews.

	Strongly disagree	**Disagree**	**Neutral**	**Agree**	**Strongly agree**
As a review author, I appreciate the feedback I receive from reviewers.	( )	( )	( )	( )	( )
When others review my changes, I worry about them judging my abilities as a programmer.	( )	( )	( )	( )	( )
It improves my confidence when others review my changes	( )	( )	( )	( )	( )
I feel that I am more thorough because I know my code will be reviewed.	( )	( )	( )	( )	( )
My personal relationship to reviewers has an impact when I author a code review.	( )	( )	( )	( )	( )
I express thankfulness to those who review my code.	( )	( )	( )	( )	( )
I learn a lot when other developers review my code.	( )	( )	( )	( )	( )

Best practices
The following questions ask about best practices for code reviews.

29) What do you think is the most important thing developers can do to increase feedback speed on code reviews?

30) What do you think is the most important thing developers can do to increase feedback usefulness on code reviews?

31) What do you think is the most important thing developers can do to increase code review productivity?

32) Please list the top impediment to productivity you encounter on code reviews.

Thank you for taking the time to respond to our survey. We hope that the results of this study will provide meaningful feedback, leading to changes in code review tool support and practices.

33) If you are interested in participating in follow up sessions regarding this survey, or in future studies, please enter your alias below. (Note: this step is completely voluntary. If you wish to participate, but not associate your alias with the answers given in this survey, you may email us separately)

34) Please use the following text box if you have any additional feedback or comments that you feel would be helpful to our research in this area.

If you found anything unclear, or should be changed in this survey, we would love to hear your feedback.

Bibliography

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

Barnett, M., et al. "Helping developers help themselves: Automatic decomposition of code review changesets." *IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 2015. 134-144.

Cohen, J., et al. *Best kept secrets of peer code review*. Smart Bear, 2006.

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

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

Morales, R., S. McIntosh and F. Khomh. "Do code review practices impact design quality? a case study of the qt, vtk and itk projects." *Software Analysis, Evolution and Reengineering (SANER)*. 2015. 171-180.

Petre, M. and G. Wilson. "Code review for and by scientists." arXiv preprint arXiv, 2014.

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

Rigby, P. C., D. M. German and M.-A. Storey. "Open source software peer review practices: A case study of the apache server." *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. IEEE, 2008. 541-550.

Rigby, P., et al. "Contemporary peer review in action: Lessons from open source development." *Software* Nov 2012: 56–61.

Tao, Y., et al. "How do software engineers understand code changes?: an exploratory study in industry." *roceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012. 15.

Thongtanunam, P., R. G. Kula and A. E. C. Cruz. "Improving code review effectiveness through reviewer recommendations." *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2014. 119-122.

Tsay, J., L. Dabbish and J. Herbsleb. "Influence of social and technical factors for evaluating contribution in github." *Proceedings of the 36th international conference on Software engineering*. ACM, 2014. 356–366.