Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development as well as Industrial Practice at Microsoft

Amiangshu Bosu, *Member, IEEE,* Jeffrey C. Carver, *Senior Member, IEEE,* Christian Bird, *Member, IEEE* Jonathan Orbeck, *Member, IEEE,* and Christopher Chockley, *Member, IEEE,*

Abstract—Many open source and commercial developers practice contemporary code review, a lightweight, informal, tool-based code review process. To better understand this process and its benefits, we gathered information about code review practices via surveys of open source software developers and developers from Microsoft. The results of our analysis suggest that developers spend approximately 10-15% of their time in code reviews, with the amount of effort increasing with experience. Developers consider code review important, stating that in addition to finding defects, code reviews offer other benefits, including knowledge sharing, community building, and maintaining code quality. The quality of the code submitted for review helps reviewers form impressions about their teammates, which can influence future collaborations. We found a large amount of similarity between the Microsoft and OSS respondents. One interesting difference is that while OSS respondents view code review as an important method of impression formation, Microsoft respondents found knowledge dissemination to be more important. Finally, we found little difference between distributed and co-located Microsoft teams. Based on our findings, three key areas that warrant focused research are: 1) exploring the non-technical benefits of code reviews, 2) helping developers in articulating review comments, and 3) assisting reviewers' program comprehension during code reviews.

Index Terms—Code Review, Open Source, OSS, Survey, Peer impressions, Commercial projects

1 Introduction

In recent years, many open source software (OSS) and commercial projects have adopted peer code review [2], a practice where developers subject their code to scrutiny by their peers. While the underlying concepts of *contemporary code review* [45] are similar to the traditional Fagan inspection [22], there are also marked differences. A Fagan inspection is a heavyweight process requiring synchronous meetings among the participants in multiple phases. Conversely, *contemporary code review* is defined as being lightweight, more informal, asynchronous, and supported by specialized tools [2]. Despite studies that show Fagan inspections improve software quality [21], their typically high cost and formality have prevented widespread adoption [29], [54]. Conversely, contemporary code review has addressed many shortcomings of Fagan inspection and has shown increasing adoption in industry and OSS contexts [3], [38], [45].

Because many OSS projects, e.g. Apache [47], Chromium[1], Mozilla[2], Qt[3], and Android[4], now require peer-review prior to merging new code into the main project codebase, there are a large number of developers regularly participating in code reviews. From the industrial perspective, Google [56] and Facebook [33] have adopted mandatory code reviews and approximately 50,000 Microsoft developers actively practice code reviews [14]. Our recent survey of OSS developers found that they spend approximately six hours per week in code review [11]. Given the large number of developers who practice code review, the total time devoted to code review is quite significant. Therefore, increasing the effectiveness of contemporary code review can greatly improve software development productivity.

To improve a process or practice, empirical researchers use a three-step approach: (1) understand the current process to identify improvement opportunities; (2) evaluate the current process and new ideas; and (3) improve the process by incorporating suggestions [58]. The high-level goal of this study, which addresses the first step in this empirical framework, is to *better understand the contemporary code review process and its benefits.* Specifically, the goal of the study is to provide that understanding by gathering information about 1) the code review process, 2) developers' expectations from code review, and 3) how code review impacts developers' impressions of their peers. To gather this information, we rigorously designed and validated a survey instrument. We sent the survey to code review participants from 36 popular OSS projects and received 287 responses. We have already published the validation of the survey instruments along with partial results of the OSS survey [11].

Prior research has identified differences between the software

A. Bosu is with the Department of Computer Science, Southern Illinois University, Carbondale, IL. Email: abosu@cs.siu.edu.
J. Carver, J. Orbeck, and C. Chockley are with the Department of Computer Science, University of Alabama, Tuscaloosa, AL. E-mail: carver@cs.ua.edu, jdorbeck@ua.edu, cmchockley@ua.edu.
C. Bird is with Microsoft Research, Redmond, WA. Email: cbird@microsoft.com
Manuscript received: TBD

1 https://www.chromium.org/developers/contributing-code
2 https://www.mozilla.org/hacking/reviewers.html
3 https://wiki.qt.io/Qt_Contribution_Guidelines
4 https://source.android.com/source/life-of-a-patch.html

engineering practices in OSS and commercial contexts [34], [42]. Because this prior work did not specifically address code review, we replicated the survey in a commercial context (i.e., Microsoft) to analyze whether there were any differences between OSS and a commercial organization relative to our study goal. To provide additional insight into the similarities or differences between OSS and Microsoft developers, we specifically recruited two types of survey participants from Microsoft: those that work on collocated projects and those that work on distributed projects. Because code review is an interactive process, we hypothesized that the Microsoft participants who work on distributed projects would have similar views about code as the OSS participants (whose projects are also distributed). Our Microsoft survey received 416 responses.

The primary contributions of this study are:

- A better understanding of developers' perception about contemporary code review;
- A better understanding of why and how developers collaborate during code reviews;
- Empirical evidence regarding the perceived non-technical benefits of code reviews;
- A comparison of code review practices between OSS and Microsoft projects; and
- An illustration of the process of systematically designing and analyzing a software engineering (SE) survey.

The remainder of the paper is organized as follows. Section 2 provides a brief description of contemporary code review process and prior literature on code reviews. Section 3 defines the research questions. Section 4 describes the research method. Section 5 characterizes the study participants. Section 6 provides the results. Section 7 discusses the implications of the results. Section 8 describes the threats to validity. Finally, Section 9 provides directions for future work and concludes the paper.

2 Background

This section provides a brief background and prior research on contemporary code review.

2.1 Contemporary Code Review Workflow

One key aspect of contemporary code review is that it is tool-based. Some popular code review tools include: Gerrit[5], Phabricator[6], and ReviewBoard[7]. Figure 1 provides a simplified overview of the contemporary code review workflow. First, the author creates a *patch-set* (i.e. all files added or modified in a single revision), along with a description of the changes, and submits that information to the code review tool. Then the author (or someone else) selects the reviewer(s) for the patch-set. The code review tool then notifies the reviewer(s) about the incoming review. During the review, the tools highlight the changes between revisions in a side-by-side display. The reviewers and the author can insert comments into the code. After the review, the author can address the comments and upload a new patch-set to initiate a new review iteration. This review cycle repeats until either the reviewers approve the changes or the author abandons the change. If the reviewers approve the changes, then the author commits the patchset or asks a project committer to commit the patchset to the project repository.

[5] https://www.gerritcodereview.com/
[6] http://phabricator.org/
[7] https://www.reviewboard.org/

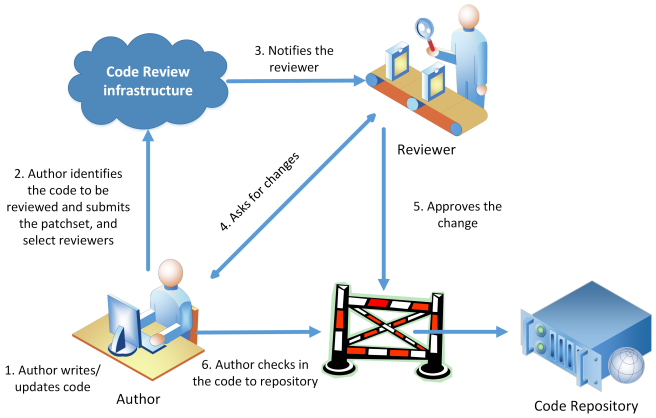

Fig. 1: Simplified code review workflow

2.2 Overview of Contemporary Code Review Research

In recent years, there have been several studies on understanding contemporary code review practice. Rigby has published a series of studies examining informal peer code review practices in OSS projects [46], [47], [48], and comparing the review practices between commercial and open source projects [45]. To characterize the code review practices, Rigby and German proposed a set of code-review metrics (i.e. acceptance rate, reviewer characteristics, top reviewer vs. top committer, review frequency, number of reviewers per patch, and patch size) [46]. Other researchers calculated similar metrics for five OSS projects and concluded that code review practices vary across OSS projects based on age and culture of the projects [1]. However, these findings were contradicted in a later study, which found that despite large differences among five OSS projects and several commercial projects, their code review metrics were largely similar [45].

After seeing the successful adoption of code review practices by OSS projects, many commercial organizations have recently adopted peer code review practices [2], [3], [44], [53]. Contrary to OSS projects, code review participants at Microsoft use both synchronous and asynchronous communication media. They also consider communications during code reviews essential for understanding code changes and design rationale. Microsoft developers expressed a need to retain code review communications for later information needs [53]. Another study at Microsoft found that although finding defects is a primary motivation for code reviews, other benefits (e.g. knowledge dissemination, team awareness, and identifying better solutions) may be more important. The major challenge is understanding the code changes [2].

While these studies characterized the code reviews in commercial projects, only one study [45], which focused on quantitative aspects of code reviews, has compared and contrasted between the code review practices of OSS and commercial projects. Since developers' motivations and project governance differ between OSS and commercial organizations [32], [42], code review collaborations may also differ between OSS and commercial projects. This lack of research was one of the motivations for gathering information about contemporary code review from both OSS and commercial developers.

While most of the earlier exploratory studies focused on understanding the code review practices, a few recent studies have focused on understanding the impact of different factors on code reviews. Code review characteristics such as review size,

component, priority, organization, reviewer characteristics, and author experience significantly influence both review completion time and outcome [5]. Moreover, a reviewer's prior experience in changing or reviewing the artifact and the reviewer's project experience increases the likelihood that s/he will provide useful feedback [14]. While these studies focused on technical human factors and characteristics of the code changes, no studies have focused on the non-technical human factors (i.e., author's reputation, and relationship between an author and a reviewer). Because code review facilitates direct collaboration between people, a better understanding of the impacts of various human factors is crucial to improve the code review practices.

A few recent studies have investigated various technical benefits of code reviews. Although the primary goal of code reviews is defect detection, because three-fourths of the review comments are related to maintainability issues code review may be more beneficial for projects which require highly maintainable code [6]. Code reviews have significant impact on software quality. A recent study found that both low code review coverage (i.e., the proportion of changes that have been reviewed) and low review participation (i.e., the number of reviewers) often increase the likelihood of post-release defects [38]. While these studies focused on the technical benefits of code review, only one study [2] has explored the non-technical benefits of code reviews. The evidence about the non-technical benefits (i.e., impressions formation, knowledge sharing, and mentoring) has been mostly anecdotal. Empirical evidence regarding various benefits of code reviews can encourage project managers to adopt code reviews for their projects.

While this prior work provides several important insights into contemporary code review, a number of key aspects are yet unexplored. First, developers who regularly use code reviews should be able describe scenarios when code reviews can be helpful or not useful. Second, experienced reviewers should also be able to describe the best strategies for code review and help other developers write acceptable code. Finally, because code reviews involve direct collaboration between participants, various types of social interactions are crucial for successful code reviews. However, these three aspects of code review have not received enough attention from researchers. One of the goals of our work is to provide a better understanding of these aspects to guide project managers' decisions about the usefulness of code review and help improve code review effectiveness.

2.3 Our Previous Survey

The work in this paper builds on the results of our previous survey of contemporary code review practices in OSS projects [11]. In that paper, we developed and validated the survey instrument described in Section 4. We used that survey to gather data from 287 OSS developers who had been active in contemporary code review. That paper reported one of the primary quantitative results from the survey, specifically that there is a high level of trust, reliability, perception of expertise, and friendship between OSS peers who have participated in contemporary code review for a period of time. In this paper, we expand these results to include qualitative data from the first survey as well as to compare the results with those from the a second survey conducted with commercial developers.

3 Research Questions

To address the study goal, we explore eight research questions. The remainder of this section defines each question.

3.1 Importance of Code Review

Code reviews require significant effort. They delay merging of code to the main branch by 1-2 days [45]. However, recent studies indicate that only one-fourth of code review comments relate to functional defects [6], [19], which raises questions whether developers perceive the effort spent in code review as beneficial. To better understand how developers view the importance of code review, the first research question is:

RQ1: *Why do developers consider code reviews important (or not important) for their projects?*

3.2 Code Review Process

Because projects often mandate the use of code review, developers spend a significant amount of time performing code reviews. To quantify this effort, the second research question is:

RQ2: *How many hours, on average, do developers spend in code reviews?*

In our prior study of code review-based social networks in OSS projects, we observed the presence of sub-communities and a higher volume of interactions between some developer pairs [12]. Subsequently, we found that, in OSS projects, experienced developers received more timely feedback on review requests than newcomers [13]. These results suggest that a history of interactions may influence a reviewer's acceptance and prioritization of particular reviews. The next research question investigates this phenomenon.

RQ3: *How do developers decide whether to accept an incoming code review request?*

Reviewers may use different criteria to determine whether a code change is of high quality. For example, reviewers may have different opinions on the effects of coding style on quality [14]. The next research question seeks to better understand these factors:

RQ4: *Which code characteristics are indicative of low quality code?*

The goal of code review is not only to identify issues in a code change but also to help the author resolve those issues. Experienced reviewers can mentor code authors regarding coding techniques, project design, or API usage. The next question seeks to better understand this mentoring process:

RQ5: *How do reviewers help authors of low quality code improve it to the level required for inclusion in the project?*

3.3 Impact of Code Review on Peer Impressions

The intense interactions during the code review process allow participants the opportunity to gain a unique insight into the abilities of their peers. For example, if a reviewer repeatedly finds the contributions of a particular code author to be of high quality, the reviewer may consider that author to be highly competent or intelligent. As a result, code review collaborations may help the participants form accurate perceptions of each other. Moreover, a reviewer may be more likely to trust project-related decisions made by an author known to be competent. Because some of the primary benefits of contemporary code review are non-technical, i.e. beyond defect detection [2], it is important to understand what those non-technical benefits may be. To help identify these

benefits, we present three research questions. The first question, on the positive side:

RQ6: *How does the use of a high-quality or an outstanding problem-solving approach affect the reviewers' perception of the code author?*

Conversely, low-quality code may result in negative impressions. Therefore, the next question is:

RQ7: *How does a poorly written code affect the reviewers' perception of the code author?*

As addressed in previous research questions, code reviews could have either positive or negative impacts on the impressions that teammates have of each other. To help judge the overall value of code reviews, we ask:

RQ8: *What is the effect of code review on peer impression?*

4 Research Method

We conducted two surveys to answer the research questions and compare OSS projects to commercial projects. The first survey targeted OSS developers. The second survey targeted Microsoft developers. We have published partial results of the first survey [11]. The remainder of this section describes the survey design process, participant selection criteria, pilot tests, data collection, and data analysis methods.

4.1 Survey Design

Because our goal was to measure peer impression constructs, we followed well-regarded social and behavioral research methods to build scales [20], [23]. In this approach, rather than directly asking the participants about each of the constructs of interest, the researchers define a number of scale items that focus on different aspects of the same underlying construct. Then, during analysis, the researchers are able to gain a more complete understanding of the construct based on the diverse set of scale items.

To understand peer impression, we identified four key constructs. For each construct, we defined a set of statements (scale items). We drew these statements from well-established scales in psychology, information science, or organizational behavior. To ensure they were complete for software engineering, we added a few additional statements. The four constructs along with the sources for the statements are:

1) trustworthiness [30], [37], [43], [51],
2) reliability [30], [43], [49],
3) perception of expertise [49], and
4) friendship [16], [49].

Table 1 lists the statements for each construct. For each statement, the respondents used a 7-point scale to indicate whether it better described a *code review partner*[8] or a *non-code review partner*[9]. We defined the scale as follows: 1 = *describes a code-review partner and NOT a non-code review partner*, 4 = *describes both equally* and 7 = *describes a non-code-review partner and NOT a code-review partner*. To avoid any bias, the survey tool presented the statements in a random order without the name of the corresponding construct.

[8] a person who reviews your code or whose code you review on a regular basis

[9] a person who has been a peer for some time, but you have rarely reviewed their code

The survey also contained four multiple choice questions, fourteen open-ended questions, and one rating-scale question to address the research questions and gather demographics. Table 2 lists the those additional questions (renumbered for the sake of simplicity). In the remainder of the paper, we will refer to the questions by those numbers.

Note that for both sets of questions, there were some minor differences between the OSS survey and the Microsoft survey. Section 4.4 explains these differences.

4.2 Participant Selection

Developers must have participated in a sufficient number of contemporary code reviews (as authors or reviewers) before they can accurately understand the code review process, the non-technical benefits of code review, and the effects on peer impression formation. To ensure valid results, we only surveyed developers with sufficient experience. For Survey 1, we mined the code review repositories of 34 OSS projects that used either Gerrit, ReviewBoard, or RietVeld, to identify developers who had participated in at least 30 code review requests (either as the author or the reviewer) and identified 2,207 developers. Similarly, for Survey 2, we queried Microsoft's CodeFlow analytics platform [8] to select 2,000 developers who had participated in at least 30 contemporary code reviews.

One of the study goals was to analyze whether developers from a commercial organization behaved differently depending on whether their project was collocated or distributed. We hypothesized that commercial developers who worked on distributed projects would be more likely to behave like OSS developers (whose projects are also distributed). To test this hypothesis across all research questions, we specifically recruited Microsoft developers from two types of projects: 1) projects in which most developers are collocated, 2) projects in which most developers are distributed.

4.3 Pilot Tests

To ensure the comprehensibility and validity of the scale items (statements) with respect to the constructs, we conducted five pilot tests. First, researchers from Psychology, Computer Science, and Management Information Systems reviewed the questions. This review led to several changes, including: (1) increasing the rating scales from 5- to 7-point, (2) rewording some questions to remove bias, and (3) adding questions for a broader perspective of the code review process.

Second, software engineering graduate students piloted the survey to identify any difficulties in understanding the questions and to estimate the time required to complete the survey.

Third, we sent the survey to 20 OSS code review participants from two projects in our database. The completion rate of this version of the survey was low. To address this problem, we rephrased some questions, reformatted the behavioral scale questions (Table 1) so they would appear less daunting, and reordered some questions.

Fourth, we sent the improved survey to 24 OSS code review participants from another project in our database. We received enough responses to analyze the internal consistencies of the four peer impression constructs. This analysis indicated that *reliability* scale had questionable internal consistency. Therefore, we added two questions to that construct.

TABLE 1: Behavioral Scale Questions

Construct	Item	Question	OSS	MS
Trust	trust_1	My communication with him/her is more informal (e.g. unofficial, friendly)	✓	✓
	trust_2	S/he is less likely to intentionally misrepresent my point of view to others	✓	✓
	trust_3	S/he is less likely to take advantage of me (e.g. exploit or deceive for personal benefit)	✓*	✓
	trust_4	I am more likely to share my personal information (e.g. feelings, opinions, or achievements) with him/her	✓	✓
	trust_5	I am more likely to consider contributing to a new Free/Open Source project at his/her request	✓	
Perception of Expertise	expertise_1	It is easier for me to identify whether s/he has the ability to provide help in a specific project area	✓	✓
	expertise_2	I more easily know if s/he is the best person to contribute to a specific project area	✓	✓
	expertise_3	It is easier for me to identify if s/he is the right person to fix a given bug report	✓	✓
	expertise_4	I am more likely to seek his/her help in a project-related area (e.g. coding problem, design decision, task assignment, documentation)	✓	✓
	expertise_5	I am more aware of his/her level of work and dedication to the project	✓*	✓*
Reliability	reliability_1	I am more comfortable assigning a critical task to him or her	✓*	✓
	reliability_2	I am more likely to be satisfied with the results of a task assigned to him or her	✓	✓*
	reliability_3	S/he is more likely to complete a project-related task (not just code review) s/he accepts even if it requires a large amount of work	✓	✓
	reliability_4	S/he is more likely to follow project coding and design guidelines	✓	✓
	reliability_5	I am more willing to accept his/her advice	✓	✓
Friendship	friendship_1	I would feel a stronger sense of loss at his/her departure from the project	✓	✓
	friendship_2	S/he has a better understanding of me (e.g. aware of my preferences and feelings)	✓	✓
	friendship_3	S/he is more likely to respond to my mailing list posts	✓*	
	friendship_4	I communicate more frequently with him/her	✓	✓
	friendship_5	If s/he does something I do not like, I am more likely to talk to him/ her about it.		✓
	friendship_6	I am more likely to consent working together with him/her on a project design task.		✓

* - *items dropped during analysis of that survey to improve the internal consistencies of scales.*

TABLE 2: Survey Questions

Q1. (OSS)	Which Free/Open Source project are you most actively involved in?
Q1. (MS)	What product do you currently work on?
Q2.	How many years have you worked in software development?
Q3.	On average, how many people contribute code or review code in a given month for the project?
Q4.	What proportion of the overall code commits in the project undergo peer code review? ◯ Less than 10% ◯ 11% - 25% ◯ 26% - 50% ◯ 51% - 75% ◯ More than 75%
Q5. (OSS)	Do you receive financial compensation for your participation in the project? ◯ Yes ◯ No
Q5. (MS)	Do most of the code change reviews that you are asked to participate in come from authors who are at your site or from a distributed site (a site that is in another time zone or more than 100 miles away)? ◯ Most come from people that are at a different site than myself ◯ Most come from people in the same site as myself ◯ Approximately half from people in same site and half from distributed sites
Q6.	Do you think peer code review is important in your project? ◯ Yes ◯ No
Q7.	Why do you think peer code review is important (or not important) for your project?
Q8.	On the project, how many hours per week, on average, do you spend reviewing other contributors code?
Q9.	From approximately how many different contributors do you review code each week for the project?
Q10.	What proportion of your code commits do you submit for peer code review? ◯ Less than 10% ◯ 11% - 25% ◯ 26% - 50% ◯ 51% - 75% ◯ More than 75%
Q11.	Is the identity of the contributor relevant to you when you decide whether to review a code contribution? ◯ Yes ◯ No
Q12.	Please explain why the identity of the contributor is relevant to you when you decide whether to review a code contribution.
Q13.	When you review poorly written code, does it affect your perception of the code author? ◯ Yes ◯ No
Q14.	Please explain how does poorly written code affect your perception of the code author.
Q15.	When you see poorly written code, how do you help the code reach the level required to be included in the project (if at all)?
Q16.	Please rate the following factors on a 6-point scale based on how strongly each indicates that code your are reviewing is poorly written. (5 - Most important, 4 - Second most important, 3 - Third most important, 2 - Forth most important, 1 - Fifth most important, 0 - Not in top five) __ Poor readability __ Lack of comments __ Does not maintain application integrity __ Poor performance __ Unnecessary complexity __ Lack of modularity (large functions / classes) __ Does not follow coding convention of the project __ Inadequate exception handling __ Duplicated code (Identical or similar code exists in more than one location) __ A large parameter list to a function
Q17.	When you review code of high quality or that has an outstanding approach to solve a problem, does it affect your perception of the code author? ◯ Yes ◯ No
Q18.	Please explain how does high quality or an outstanding approach to solve a problem affect your perception of the code author.

Finally, we sent the survey to 117 OSS code review participants from 11 other projects. The completion rate was near 20% and the internal consistency of each construct was sufficient (Cronbach's $\alpha > 0.7$[10]). Therefore, we deemed the survey ready for broad distribution. We incorporated feedback from the pilots into the final survey.

4.4 Data Collection

There were a few differences in the data collection process for the two surveys.

4.4.1 OSS Survey

In February 2013, we sent a survey invitation to each of the 2,046[11] active contemporary code review participants in our OSS database. Of those, 231 emails were undeliverable, leaving 1,815 valid invitations. Two weeks later we sent a reminder email. Approximately two weeks later we closed the survey after the daily response rate decreased to almost zero. We received 287 responses (response rate of ~16%).

4.4.2 Microsoft Survey

Questions Q1 and Q5 on the OSS survey were specific to OSS developments. For the Microsoft survey, we modified these questions to fit the commercial development context. Two scale items (trust_5 and friendship_3) were specific to the OSS development context and therefore excluded from the Microsoft survey. As a result of those excluded items, the friendship scale had only three remaining scale items. To ensure that all scales had an adequate number of items, we added two items to the friendship scale. The last two columns of Table 1 indicate exactly which items were included in each survey. In September 2013, we sent an invitation to the revised survey to 2,000 Microsoft developers who had participated in contemporary code review. After one week, we closed the survey. We received 416 responses (response rate of ~21%).

4.5 Data Processing and Analysis

The following subsections describe the data processing and analysis steps for the behavioral scale questions and the five open-ended questions.

4.5.1 Behavioral Scale Questions

We analyzed three forms of validity for the survey. First, we used expert researchers from Psychology, Management, and Computer Science to review the survey question for face validity [36]. Second, we carefully chose appropriate items from prior validated scales to ensure content validity [36]. Finally, we performed a principal components analysis with VARIMAX rotation to measure the construct validity [36] of the scale items. Table 3 reports the α coefficients measures for the scales and their interpretations based on the two surveys.

This approach ensured high reliability and validity of the behavioral scales used in this study. In a prior article, we have detailed the reliability and validity measures of the survey instruments [11]. We also validated the reliability and validity of the four modified scales used in the Microsoft survey using a similar process.

[10] Chronbach's α is widely used to calculate the internal consistency of multiple-item measurements (e.g. the behavioral scale constructs). The results of this test are interpreted as follows: $\alpha \geq .9 \rightarrow$ *excellent*, $\alpha \geq .8 \rightarrow$ *good*, $\alpha \geq .7 \rightarrow$ *acceptable*, and $\alpha < .7 \rightarrow$ *questionable*.

[11] 2,207 total less the 161 used in the pilots

TABLE 3: Coefficient Alpha (Cronbach's α) of the scales

Construct	OSS		Microsoft	
	α	Interpretation	α	Interpretation
Trust	.756	Acceptable	.798	Acceptable
Perception of Expertise	.811	Good	.839	Good
Reliability	.810	Good	.736	Acceptable
Friendship	.700	Acceptable	.758	Acceptable

4.5.2 Open-ended Questions

For the open-ended questions, we followed a systematic qualitative data analysis process. First, two analysts (Research Experience for Undergraduates (REU) students) extracted the general theme from each response to the OSS survey. Next, the first two authors worked with those analysts to develop an agreed-upon coding scheme for each question. Using these coding schemes, the two analysts independently coded the responses. After coding, they examined their results to identify any discrepancies. They discussed and resolved those discrepancies. For the Microsoft survey, the last two authors used the same process to analyze the qualitative data. For the five open-ended questions in the two surveys, we coded a total of 2626 responses.

5 Demographics

To provide proper context for the results, this section describes the demographics of the projects represented by the respondents and of the respondents themselves.

5.1 Projects represented

Table 4 provides the results to Question Q1 (Table 2) about respondents' primary projects. The number in parenthesis represents the number of respondents who listed that project. As an indication of the frequency of code review in these projects, approximately 83% of the OSS respondents and 90% Microsoft respondents indicated that more than 75% of the code changes in their projects undergo code review. Furthermore, almost two-thirds of the respondents in both surveys indicated that they submit every code change for code reviews. Therefore, the survey respondents are actively using contemporary code review.

5.2 Respondent demographics

Table 5 shows the results from survey questions 2, 8 and 9. In each case, we grouped the responses into four categories by analyzing the frequency distributions of the responses. We then checked these categories to ensure they also made logical sense.

For the OSS survey, 60% of the respondents were paid to work on the project and 40% were volunteers (Q5). The percentage of paid participants is not surprising because the list of projects we drew from included some large sponsored projects (e.g. Android, Chromium OS, Qt project, and OpenStack), in which most of the participants are employees of the sponsoring companies. This distribution is slightly higher, but in the same range as previous OSS surveys that had 40%-50% paid OSS participants [10], [31].

For the Microsoft survey, approximately 69% of the respondents indicated that most of the code review requests come from developers at the same site, 13% come from a different site, and the remainder are equally split.

TABLE 4: Respondents' primary projects

Open Source Projects			Microsoft Projects		
Qt Project (36)	OpenStack (32)	CyanogenMod (28)	Bing (54)	Windows (48)	Office (43)
TYPO3 (20)	Android (19)	MediaWiki (17)	Azure (24)	Visual Studio (22)	XBox (13)
oVirt (17)	Linux kernel (16)	Chromium OS (14)	Ad Center (11)	Exchange (10)	Dynamic AX (12)
Eclipse (9)	Gromacs (9)	ITK (9)	Windows Phone (10)	IE (6)	SQL Server (6)
LibreOffice (8)	OpenAFS (6)	Scilab (5)	Dynamic CRM (4)	Sharepoint (4)	Skype (3)
VTK (5)	Gerrit (4)	AOKP (2)	Halo (3)	TFS (3)	HPC (3)
Couchbase (2)	Debian (2)	Others (24)	Autopilot (3)	Lync (3)	Others (131)

TABLE 5: Demographics of the respondents

Question	Mean		Median		Category description	% of Respondents	
	OSS	Microsoft	OSS	Microsoft		OSS	Microsoft
Q2. Experience in software development	7 years	10.7 years	5 years	9 years	*Low*: Less than 2 years	20%	8%
					Medium: 3 to 5 years	33%	19%
					High: 6 to 10 years	26%	33%
					Veteran: More than 10 years	21%	40%
Q8. Average number of hours per week spent in reviewing other contributors' code	6.4 hours	4.7 hours	5 hours	4 hours	*Low*: Less than 2 hours	30%	26%
					Medium: 3 to 5 hours	32%	48%
					High: 6 to 10 hours	26%	21%
					Very High: More than 10 hours	12%	5%
Q9.Number of contributors' code reviewed each week	6.3 peers	5.5 peers	5 peers	5 peers	*Small*: Less than 2 peers	20%	12%
					Medium: 3 to 5 peers	45%	58%
					High: 6 to 10 peers	27%	25%
					Very High: More than 10 peers	8%	5%

6 RESULTS

The following subsections describe the results of the two surveys. For each of the eight research questions introduced in Section 3, we compare the results from the OSS survey with the results from the Microsoft survey. To help clarify the results, we also include excerpts from the qualitative responses to the open-ended questions. In this section, we identify each of the respondents using a unique identifier, with *OSS-XXX* and *MS-XXX* indicating respondents from the OSS Survey and the Microsoft Survey, respectively. Unless explicitly stated, the opinions of the OSS and Microsoft respondents were similar. Therefore, the chosen quotations best represent the set of responses from both samples (OSS and Microsoft).

As a result of the coding process (Section 4.5.2), each of the open-ended questions had a large number of detailed categories. For this presentation of the results, we abstracted the detailed categories into a smaller number of high-level categories. Further analysis of the data using the more detailed categories can be found on a supplemental website[12]. In a qualitative analysis, each open-ended response could match multiple codes. Therefore, the sum of the percentages can be greater than 100%.

For each question, we tested the normality of the answer distribution using the Shapiro-Wilk test [52]. In cases where the distribution was non-normal, we used non-parametric statistics.

6.1 RQ1: Why are code reviews important?

In response to Q6, 98.6% of the OSS respondents and 100% of the Microsoft respondents considered code reviews to be important for their project. Figure 2 shows the reasons why the respondents found code reviews to be important for code quality (Q7). Although the relative order of the responses was the same in the both surveys, the distribution of answers was significantly different between the OSS respondents and the Microsoft respondents ($\chi^2 = 21.38, df = 5, p < .001$).

The OSS respondents emphasized maintainability slightly more than the Microsoft respondents did. Because OSS participants come from diverse locations, backgrounds and expertise levels, the quality of submitted code can vary greatly. Therefore, OSS reviewers have to focus more on maintaining consistent code quality. Conversely, there is less quality variation in code from Microsoft developers. Therefore, Microsoft respondents are able to focus more on finding defects and improving project awareness during code reviews.

Microsoft developers told us that knowledge sharing is one of the primary purposes of code review. Newcomers to a team often are included on reviews so they can learn more about the codebase and how code reviews are conducted. In some cases, there is an explicit mentor-mentee relationship between an expert and a less experienced developer that is manifested in code reviews. We are unaware of a similar use for code reviews in OSS projects.

Interestingly, *eliminating functional defects* was only the third most important reason for code reviews in both surveys. This result is consistent with earlier findings that the other benefits of contemporary code review i.e. knowledge transfer and identifying better solutions, may be more important than defect detection [2]. Prior research on software inspection also reported the following benefits provided by software inspections: defect identification [27], [50], knowledge sharing [15], [50], [57], increased project awareness [15], [50], and reduced development costs [26], [40]. Moreover, our prior work found that approximately half of code review comments relate to maintainability issues, with less than a quarter related to functional defects [14]. The following subsections provide details on the reasons why developers consider code reviews important for their projects.

6.1.1 Improve Maintainability

The majority of respondents from each survey (OSS: 71%, Microsoft: 61%) indicated that code review improves project main-

[12] http://carver.cs.ua.edu/Data/Journals/CodeReviewSurvey/

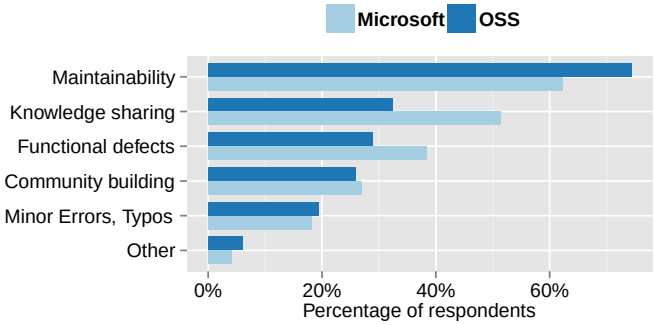

Fig. 2: Importance of code reviews

tainability in terms of efficiency as well as other maintainability attributes, i.e. legibility, testability, adherence to style guidelines, adherence to application integrity, and conformance to project requirements. In general, developers are more cautious when they know that changes are subject to peer review.

If you know someone is going to look at you, you dress better. When you know someone is going to question you for certain decisions, either you don't make them or you are prepared to defend it. So, in general, it improves quality and makes you better developer by forcing you to look at your code the way others look. [MS-50]

In addition to code quality, reviewers evaluate the code's conformance to project requirements. This conformance is especially important in OSS projects where contributors can have different personal goals.

By requiring approval from core maintainers, it also helps to keep undesirable code out. [OSS-48]

Code reviews and the subsequent discussions also help maintaining project design constraints and result in better designs.

It allows experts on particular areas of the (vast) codebase to detect issues with changes early, helps generate improved design ideas. [OSS-118]

Similarly,

..reconsidered crucial design decisions and ended up thinking "wow, the way I was doing it is stupid for a lot of reasons." [MS-80]

Another benefit of code review is the production of more readable code. To help reviewers understand the code easily, developers use documentation, comments, and appropriate indentation to make code more readable.

Code review helps to see if "my code" is read and interpreted the way it should be. [MS-25]

Readable code also helps long-term maintainability, especially for large-scale and long-lived projects.

The code is our most valuable asset as well as our biggest liability. But we rarely have the time to re-invest in features done, so it is vital that whatever we checkin is of the right quality. You can test so-and-so much, but you really cannot test maintainability and how easy a codebase is to debug. So to avoid bug-farms you really have to review. [MS-312]

Code review also helps enforce a common coding style, which is one of key characteristics of maintainable code.

Code style is even important because code is written once but read so many times more. In fact, code can be maintained by other devs so it's important it follows guidelines. We have strict coding guidelines published. [MS-290]

6.1.2 Facilitate Knowledge Sharing

Code review facilitates multiple types of knowledge sharing. Code review interactions help both authors and reviewers learn how to solve problems using new approaches. Reviewers often not only identify issues but also explain why the author's approach could lead to potential problems. Reviews also help socialize project details, e.g. architecture, common APIs, and existing libraries.

... spread information to more people so all knowledge of a system is not lost if someone is out sick, on vacation, or leaves the team, assist in sharing knowledge of helpful utilities so that we do not end up with duplicate systems doing the same things. [MS-196]

Code reviews increase project awareness among the project members by ensuring that at least one or two reviewers are also aware of code changes.

... it helps to ensure that more than one member of a group is familiar with any changes, it makes sure that all changes are (at least somewhat) sane, and it helps to foster feedback from people who will be affected by a change before the change actually happens. [OSS-194]

Code reviews also allow senior project members to mentor newcomers.

... code reviews are often one of the primary methods of knowledge transfer and brainstorming about software between developers. They're part of the critical path to ramp-up new developers on both the project and technologies, and they're often where experienced developers share tricks-of-the-trade and knowledge in context. [MS-190]

In addition to newcomers, more experienced project members can also learn through code reviews.

... allows us to leverage the lessons learned by each person in the code base that not everyone will encounter. [MS-355]

6.1.3 Eliminate Functional Defects

Reviewers often find logical errors, corner cases, security issues, or general incompatibility problems that the author may have overlooked.

Code review dramatically reduces bug count, in my experience. It is very rare for a change to be accepted without some suggested improvements or notations of deficiencies by reviewers. [OSS-145]

Experienced security reviewers are often able to identify critical security flaws during code reviews.

More people see more, you can not let anyone from the community to merge anything to your code (security risk). [OSS-105]

Finally, code reviews help to inform a wider audience about agreed upon changes and thus help avoiding incompatibility issues (i.e., a broken build).

... makes sure the feature/bug fixing can integrate into other parts of project, done by other developers. [MS-241]

6.1.4 Encourage Community Building / Collaboration

By fostering direct collaboration between developers and reviewers, code reviews encourage community building and collaborations. While community building was mentioned by both Microsoft respondents and OSS respondents, it was seen as very important in OSS projects.

.. helps the developer feel part of the F/OSS community. It also provides a framework for participation, and allows people to voice opinions on submitted changes and feel involved. [OSS-39]

In addition, the inclusion of various stakeholders in the review process fosters collaboration among the team members.

It enables collaborative contributions to the implementation of the system. Everyone on the team has a say on every part of the system.[MS-100]

Finally, code reviews also help developers gain a better perception of each others' expertise and build relationships.

Gives a sense of collaboration and camaraderie among engineers who would not typically work together (employees of competing companies, for instance). [OSS-37]

6.1.5 Identify Minor Errors, Typos

Developers often do not notice their own minor errors and typos. Without code reviews, identifying those minor issues may be time-consuming. In addition, developers may forget to keep comments updated, which is crucial for long term maintainability of the project. In most cases, the majority of the minor errors or typos are identified during code reviews.

It helps catch human errors/typos. Two pairs of eyes are always better than one. [MS-182]

6.2 RQ2: How much time is spent in code reviews?

According to Q8, the median time spent in code review each week is 5 hours for OSS developers and 4 hours for Microsoft developers. Considering 40 work-hours per week, this result indicates that *developers spend 10% - 15% of their time in code reviews.* Moreover, OSS developers spend significantly (Mann-Whitney U, p=0.05) more time in code review than Microsoft developers

Because a less experienced developer would be more likely to invite an experienced teammate to perform a code review, we hypothesize that experienced developers would spend more time performing code reviews. Figure 3 shows development experience vs. median hours spent in code review. Since the distribution of review hours per week significantly differs from a normal distribution, we used non-parametric ANOVA (i.e., Kruskal Wallis H), which indicates that those differences are statistically significant (OSS: $\chi^2 = 8.16, p = 0.043$, Microsoft: $\chi^2 = 8.43, p = 0.038$).

For the OSS respondents, the paid contributors spend significantly more time in code review than volunteer participants, median of 5 hours vs. 3 hours (Mann-Whitney U, $p < .001$). This results makes sense because paid contributors often act as gatekeepers to maintain the integrity of the software by preventing buggy, unwanted, or malicious code. As a gatekeeper, the paid participant will therefore review code from many different peers and spend more time in code reviews. To support this observations, the results of Q9 indicate that paid contributors review code from significantly more peers each week than volunteers do, median of 5 peers vs. 4 peers (Mann-Whitney U, p=.009).

6.3 RQ3: How do developers decide whether to accept review request?

More than half of the OSS respondents and two-thirds of the Microsoft respondents indicated that the identity of the author was important in accepting a code review request (Q11). Figure 4 shows the reasons why respondents found the code author's identity important (Q7). Although the factors identified were common between the two surveys, the distribution of responses was significantly different ($\chi^2 = 24.09, df = 4, p < .001$). For example, the OSS respondents emphasized the non-technical factors (i.e., reputation and relationship), while the Microsoft

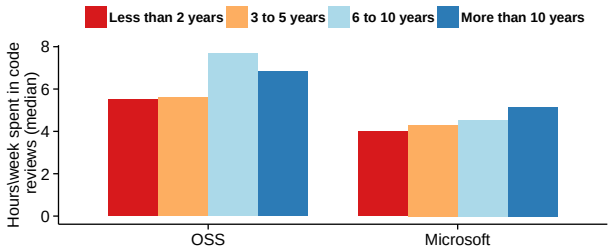

Fig. 3: Hours spent in code reviews vs. Experience

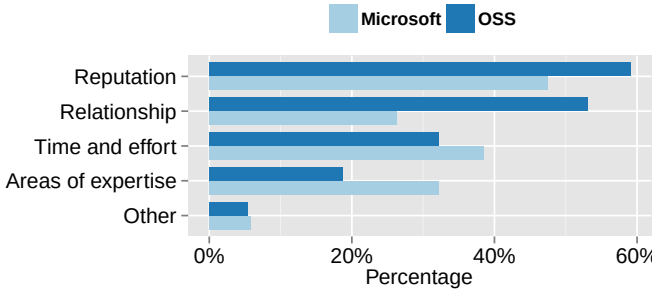

Fig. 4: Why the identity of a code author is relevant

respondents emphasized the technical factors (i.e. time/effort and expertise). This result reinforces the emphasis that OSS developer place on reputation and relationships found in other research [32], [41].

Conversely, the areas of expertise of the contributor and time / effort required for the review were the priority considerations for the Microsoft respondents. Discussions with developers shed some light on the reasons for this result. With respect to expertise, an experienced Microsoft developer often receives a large number of review requests. Therefore, to minimize review time, s/he is more likely to accept reviews for which s/he has expertise. In addition, since Microsoft developers must manage competing demands for their time and products have tight timelines, developers must frequently make decisions based on the time required to complete a task. Thus, the choice to participate in a code review depends heavily on the estimated time required. The following subsections provide details on each factor.

6.3.1 Relationship with the Author

A reviewer's relationship and history of interaction with the code author often affects the decision of whether to accept a review request. Relationship with the code author was very important particularly for the OSS developers who were almost twice more likely than the MS developers to consider their relationship with the code authors when accepting code reviews.

We know each other. We know each others strengths and weaknesses and we can change the way we review to meet to needs of the specific developer. It is an optimization that humans naturally perform. Is it a net positive? I think so. [OSS-60]

Furthermore, to optimize the time spent reviewing code, a reviewer often chooses to review code from authors who have already reviewed his/her code.

It feels like a "quid pro quo" - if the contributor has reviewed my code in the past in a thorough/timely fashion, I like to return the favor. [OSS-20]

Next, because code changes from a trustworthy author are more likely to require less reviewing effort, the level of trust between the reviewer and the author is important.

If you review code from someone you already know and trust very well, you can only focus more on detecting careless mistakes and less on overall design of the code change. [OSS-276]

Finally, reviewers prioritize requests from their teammates or co-workers over others.

... there are other programmers at my company that also work on the project, and if they've submitted something that is time-sensitive, I'm likely to prioritize that review to keep things moving. First-time contributors tend to get down-prioritized a bit. [OSS-227]

6.3.2 Reputation of the Author

Depending upon their goals or project roles, some reviewers may seek out authors with positive reputations, while other may focus on those with poorer reputations. To leverage the time spent in code reviews, some reviewers favor review requests from authors they consider to be capable of producing high-quality code.

Often time to do reviews is limited. I prefer changes from contributors where I know that they are proposing good changes (high code quality, good commit message, small scope, focused on one thing), because I know that I can finish the review quickly. I also prefer changes from contributors that themselves give feedback by doing code review on changes of others. [OSS-106]

Conversely, some reviewers are gatekeepers that focus on code changes from new or troublesome authors.

I am more likely to review changes by developers new to the team, as well as developers who have a history of poor adherence to coding standards or lots of significant comments on their reviews.[MS-90]

Sometime the experience of the code author also influences the decision to accept or reject a review request. Developers will often accept review requests from experienced contributors expecting to learn about outstanding techniques or designs. Conversely, experts and/or code owners may be more likely to review code coming from inexperienced developers in an effort to maintain high code quality.

Some people do stellar work, and I want to learn from them, so I review their CR to see what they did, even though I almost never find problems. [MS-319]

6.3.3 Area of Expertise

Many reviewers prefer reviewing code changes that are closely related to their areas of work or expertise. The identity of the code author can often help them to determine whether the change is relevant. The area of expertise is a very important factor especially for the Microsoft developers to prioritize incoming reviews.

I get a lot of code review requests from multiple teams. I only review things that are in my area, and author of change often helps to determine if changes are relevant to me. [MS-325]

Some reviewers even decline code reviews that are not related to their areas of expertise.

Area ownership and expertise matters. If I'm unfamiliar with an area and depending on the complexity, I'll decline or partially review it.[MS-12]

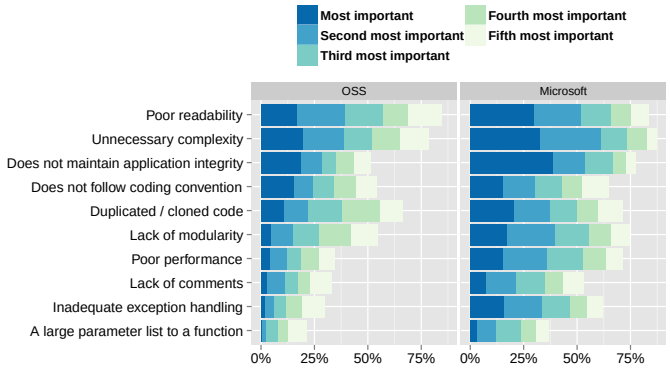

Fig. 5: Characteristics indicating poor code (Sorted based on the ranks in the OSS survey)

6.3.4 Anticipated Time / Effort to Review

Based on the author's identity, reviewers can often anticipate the amount of effort required. To maximize the utility of time spent in code reviews, some reviewers focus on areas that require the most attention.

My time is inherently limited, so I choose to prioritize code review for less experienced developers. For code from people that I know have a history of quality contributions, I'm less likely to spend time reviewing. [OSS-176]

Conversely, to reduce their effort, some reviewers prefer to avoid changes from known poor coders.

Sometimes you want to quickly review code from contributors whose work you trust greatly. Other times you might choose to ignore work from a known contributor who typically produces poor work. [OSS-114]

6.4 RQ4: Which characteristics indicate low quality code?

From our previous work [10] and common code smells (any symptom in the source code that usually corresponds to a deeper problem in the system) [25], we identified ten characteristics of low-quality code. We asked the respondents to rank order those characteristics based on their importance during code reviews. Due to the limitations of the survey tools, in the Microsoft survey, respondents rated each characteristic on a 6-point scale rather than rank ordering them (i.e. they could rate multiple characteristics as *most important* instead of only one). As a result, the total for *most important* is greater than 100% for the Microsoft survey. However, we believe that this may not be an issue, since we are interested only in comparing the ranks of the characteristics between the two surveys. For the two surveys, we separately calculated the ranks of the characteristics using the number of top two ratings (i.e. Most important, and Second most important). Figure 5 summarizes how the respondents rated the relative importance of each characteristic (Q17), where the characteristics are sorted based on their ranks in the OSS survey.

The fact that *unnecessary complexity* and *poor readability* were among the top three characteristics in each survey, suggests that code which is simple (not complex) and readable is easier to review. It is interesting to note that *lack of comments* ranked very low (OSS: 8th and Microsoft: 9th) in both of the surveys. The combinations of these results suggest that reviewers expect code to be straightforward and self-documenting rather than requiring

extensive comments to explain it. Because a reviewer has to understand the code to properly review it, a complex approach, even if well-commented, will likely take longer to review.

Does not maintain application integrity was also among the top three characteristics in both surveys. For long-term project maintaining a consistent design is very important. A feature that violates project design not only adds burden for future maintenance but also opens bugs or even vulnerabilities. Such code generally indicates either the author lacks knowledge about the project design, or the author lacks care / dedication for the project. Therefore, authors should be careful that submitted code changes maintain application design constraints.

The ranking of eight of the nine characteristics was similar for both surveys (with a difference of no more than two ranks). The exception was the characteristic: *does not follow coding convention of the project* (fourth in OSS and eighth in Microsoft). There are two possible explanations for this result. First, while Microsoft respondents may consider coding convention issues important, they may not judge code quality based those problems because they are easier to fix. Second, because Microsoft developers often use automated tools to identify and fix coding convention issues, they may focus less on these issues during code review.

6.5 RQ5: How do developers help improve low quality code?

Figure 6 lists the approaches the respondents used to help poorly written code reach the level of quality required for inclusion in the project (Q15). The distribution of responses was significantly different between the OSS respondents and the Microsoft respondents ($\chi^2 = 93.29, df = 6, p < .001$). This difference was largely due to two factors.

First, the Microsoft respondents are more likely to communicate with the author using other channels (i.e., face-to-face, Skype, instant messenger, or email). They found those communications helpful in quickly resolving any misunderstandings. Conversely, face-to-face communication may not be an option in an OSS project. Interestingly, OSS developers could use some of the tools (e.g., Skype or other voice/video over internet technologies), but they do not.

Second, when other methods are unsuccessful, the OSS respondents are more likely to rewrite the code themselves. Because OSS participants may not be obliged to follow up, they may not make the changes required to make the code acceptable. If a code change is important, then the reviewer may choose to just fix the problem rather than waiting on the original author. Conversely, the Microsoft respondents rarely rewrite poor code themselves, for two primary reasons: 1) Microsoft developers are required to follow up, and 2) reviewers know that they have to mentor authors of low quality code to help them learn how to write better code.

6.5.1 Provide Comments

More than 80% of the respondents from each survey provide comments through the code review tool to help authors improve poorly-written code. The reviewers typically indicate specific shortcomings of the code and ask the author to fix those issues.

> *For issues specific to the patch in question, or small coding style/convention issues, I'll reply to the patch with point-by-point feedback and suggestions. For major systemic issues, such as pervasive use of incorrect coding style/conventions, or fundamental architectural issues, I'll reply to the first instance of such an issue with a summary of the problems and an indication that many more exist that I didn't quote or comment on. [OSS-46]*

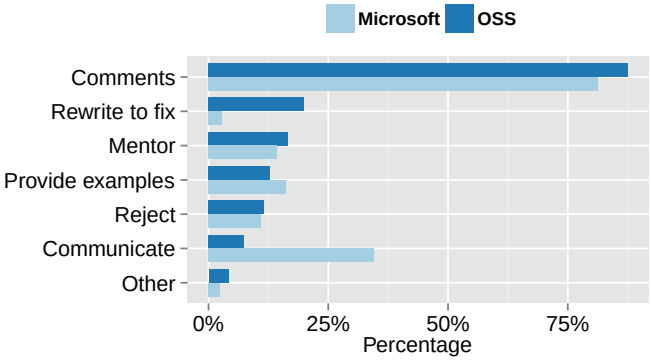

Fig. 6: How reviewers assist to fix poor code

Many reviewers provide hints or suggestions to refactor the code to make it more readable.

> *I try and give syntax tips or suggest improvements (like rewriting a function to reduce complexity, pointing out where we have duplicate code and how it might be shared and suggesting to split a large function into smaller ones). [MS-403]*

A few respondents also mentioned the importance of constructive criticism to avoid hurting the feelings of the code author.

> *Critique the code, not the author. Describe better approaches, don't just denigrate the chosen one. Ask questions about why an approach was chosen, don't attack the choice. Point out possible edge cases where they are overlooked. Refer directly to the coding standard where appropriate. [MS-73]*

6.5.2 Rewrite/Fix the Code

In projects where authors are not required to respond to reviews, i.e. some OSS projects, some reviewers find it quicker to just fix the low-quality code rather than providing comments.

> *... in some circumstances I massage the code change myself and explain to the submitter why I have made the follow-up change. [OSS-276]*

The reviewers do acknowledge that this practice may not be the best approach.

> *Usually it boils down to rejecting it or fixing it by writing it myself. I am aware that this is not a good practice, but we're all volunteers. [OSS-154]*

6.5.3 Provide Mentoring

In cases where the author of low-quality code lacks project or programming knowledge, mentoring may be the best approach to improve code quality.

> *I try to provide design guidance to the contributor when I think it will benefit the project. I may also provide some language-specific mentoring or at least refer the contributor to relevant documentation I believe to be helpful. [OSS-71]*

Another type of mentoring is to ask questions to help the author understand potential code problems.

> *Ask questions such as what happens in scenarios to guide him through this understanding. If this is due to lack of understanding of fundamental technology then give pointers to bring up the knowledge. [MS-126]*

6.5.4 Provide Examples

Some reviewers prefer providing example code or directing an author to other well-written code in the project.

> *I will usually point the original author towards existing examples of code in the project to look at for reference. [OSS-58]*

6.5.5 Reject Until Good

Some reviewers prefer to reject code changes until they meet the project quality standards.

> *I do not sign off until I am convinced that the code change meets the team criteria for quality. If the author does not understand or agree with my feedback, I typically will sit down with them to discuss in detail. [MS-132]*

6.5.6 Communicate with the Author

Over a quarter of the Microsoft respondents preferred discussing code changes via email or instant messenger rather than inside the code review tool. They found this type of communication helpful for avoiding long discussions in the code review tool and embarrassment of the author. This percentage was much higher than for the OSS respondents.

> *Sometimes, for more difficult issues, I create an email thread on the side to have a better back-and-forth discussion about the change as a whole instead of a discussion about one small part indicated in the review. If I consider the change very poorly written, I tend to keep the side-conversation more private to avoid embarrassing the author. [MS-145]*

If feasible, some Microsoft reviewers also prefer to meet the author face-to-face and discuss the issue to resolve potential misunderstandings.

> *Usually the best option is to go to their office and see where they are coming from and whether they made oversights or were missing information. [MS-34]*

6.6 RQ6: What is the impact of high quality code?

More than 85% of the respondents from each survey indicated that high quality code or use of an outstanding approach to solve a problem affects their perception of the code author (Q17). As shown in Figure 7, the aspects of peer perceptions that are influenced by high quality code (Q18) differ significantly between OSS respondents and the Microsoft respondents ($\chi^2 = 25.81, df = 3, p < .001$).

For the OSS respondents, the largest impact of high quality code is increase in positive impressions about the personal characteristics of the code author. Because of the lack of physical interaction among OSS participants, socio-technical interactions (e.g., via code reviews) become more influential in the formation of impressions about the personal characteristics of teammates [10]. The lower importance of this factor for Microsoft respondents may be because developers in industrial organizations have other methods of observing and assessing the characteristics of their peers, e.g. participating in face-to-face meetings, working in close proximity, communicating frequently, and participating in non-work social activities like lunch.

Conversely, the Microsoft respondents indicated that the largest impact of high quality code is stronger relationships and future collaborations with the code authors. Because approximately 75% of the code reviews at Microsoft are performed by teammates of the code author [14], the reviewers are likely already aware of the personal characteristics of an author. Instead, code reviews help the reviewers judge the intellect and coding skill of the code author. High quality code can lead to increased respect, admiration and trust.

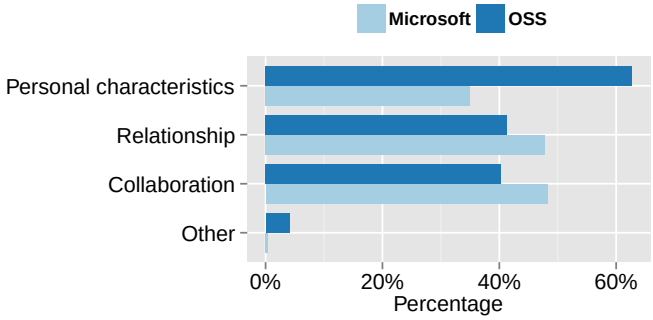

Fig. 7: Impact of high quality code

6.6.1 Creates Positive Perceptions about Personal Characteristics

Approximately 60% of the OSS respondents and 36% of the Microsoft respondents indicated that high quality code or an outstanding problem solving approach were evidences of the personal characteristics of the code author. First, high quality code can indicate that the author is competent, an expert in the area, high performing, and posses professional skills.

> *I will assume that the author is experienced or skilled in a particular area if I see good work in that area. If the author tends to produce clean, well structured code that complies with the coding style guide, I will assume that he/she works in well organized and thorough manner. [MS-68]*

Second, high quality code can be a sign of the contributor's ability.

> *It's my opinion that you can infer the quality of a developer by the quality of their changes. Changes that are not well thought out indicate sloppy thinking, but changes that are neat and tight indicate an accuracy of thought that I appreciate. [MS-216]*

Third, the quality of code changes can indicate the level of the dedication of the code author.

> *High quality code shows that the author cares about the project and has considered the ramifications of their changes. High quality code also can elevate the project as new ideas are injected into the community. [OSS-31]*

Finally, high quality code can be a sign that the author has a good understanding of the project.

> *This means the author has spent a large amount of time working hard to understand the problem at hand and has come up with a great solution. [OSS-220]*

6.6.2 Helps Build Relationships

Approximately 37% of the OSS respondents and 47% of the Microsoft respondents indicated their desire to build relationships with outstanding code authors. They believed that authors of good code are trustworthy and should have additional tasks and privileges. Trust is very important in OSS projects, because gaining the trust of the core members is the only way a contributor can earn commit privileges.

> *.. "outstanding approaches" are rare, but well written, well-documented code indicates that the author can be trusted. He/She may get approver rights or become maintainer of a module. [OSS-167]*

In addition, respondents reported an increase in respect and admiration for authors of high quality code.

High quality code speaks about its author. I see software development more as an art than an actual engineering discipline. From this perspective, a person writing high quality code is someone that deserves respect and recognition, as he combines his knowledge/experience with his intellect to create unique solutions. [OSS-180]

Finally, authoring high quality code can also lead the author to earning a better reputation within the community.

Impressed by ability to solve the problem, this is really just meritocracy at work. People who put in the time and solve problems with outstanding approaches build a strong reputation. [OSS-144]

6.6.3 Encourages Future Collaborations

Approximately 36% of the OSS respondents and 48% of the Microsoft respondents indicated their desire for future collaborations with authors of high quality code because they viewed those authors as expert contributors from whom they could learn.

If someone writes some code with an "Outstanding approach" such that it impresses me, I'll probably read all of their code reviews after that, in order to learn more. [MS-103]

Some developers often volunteer to review code changes submitted by these authors primarily to learn.

I will be more likely to consult this person in the future as they are a proven performer in code matters. I may pay more attention to future work by them by signing up for code reviews, but more to understand their work than to pick apart code line by line. [MS-333]

Apart from learning, developers also seek assistance from outstanding code authors when having difficulty solving a problem.

I know that this is a guy to go to when faced with difficult problems, and that he can be counted on to give proper reviews and suggest improvements to my code. [OSS-109]

Consistently submitting high quality code can improve the trustworthiness of the author as respondents stated that, if busy, they would spend less time reviewing code from these authors.

.. if this person sends another code review on a day that I'm really busy, I won't worry about looking it because I trust that they also "did the right thing" in this new code review. [MS-128]

Finally, authoring high quality code changes resulted in an increased perception of reliability and the assignment of more complex or critical tasks.

Seeing well written code increases my confidence in the author and I know I will be able to rely on that author for future tasks of high complexity or high importance. [MS-341]

6.7 RQ7: What is the impact of low quality code?

More than three-quarters of the respondents from each survey indicated that poorly written code negatively influences their perceptions of the code author (Q13). As shown in Figure 8, the aspects of peer perception that are influenced by poorly written code (Q14) differ significantly between OSS respondents and Microsoft respondents ($\chi^2 = 52.79, df = 3, p < .001$). The OSS respondents were more likely to form negative impressions about the experience and personal work habits of authors of poorly written code. The reasons for forming negative impressions are largely the same as those for forming positive impressions

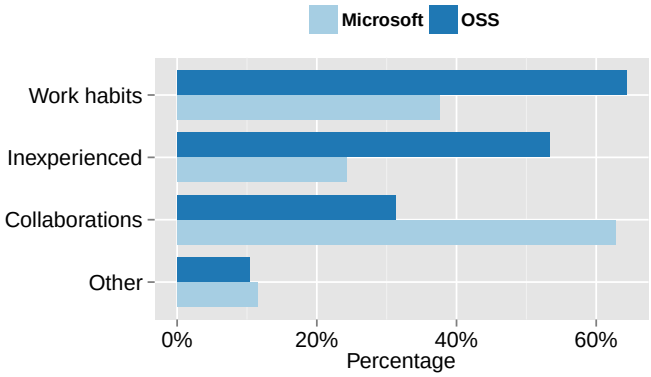

Fig. 8: Impact of a poor quality code

(RQ6). In addition, the Microsoft respondents considered authors of low quality code to be incompetent. They were less likely to collaborate with those authors in the future due to the expected increase in time and effort such collaborations would require.

6.7.1 Creates Negative Perceptions about the Work Habits

Approximately 62% of the OSS respondents and 38% of the Microsoft respondents indicated that low quality code negatively affects their perceptions about the work habits of the author. As before, Microsoft respondents were likely to find this factor less important than OSS respondents because of the many other ways that developers interact and are able to assess each other's characteristics and work habits. More specifically, reviewers consider the submission of low quality code to be a sign of the author's carelessness or lack of respect for his/her peers.

Code that is poorly written, does not follow guidelines etc. makes a bad impression, it gives a sloppy impression, like the person writing the code does not take the time to provide the code with proper quality, yet expect it to be reviewed. Providing sloppy code is in my opinion a sign of lack of respect. [OSS-260]

Second, reviewers think that authors of low quality code were lazy or lacking dedication to the project.

Author either didn't put enough time to investigate the task he is solving or does not have understanding of the system he is building. Either case he should have spent more time to understand what is he doing. [MS-41]

While low quality code does affect perception, reviewers realize that people do make mistakes, so they are likely to excuse the first few mistakes and begin forming a negative impression when the author is unable or unwilling to learn from previous mistakes.

For occasionally poor code, not much effect - people have off days, or may just misunderstand the particular area of code they are modifying. But if the author continues to makes similar mistakes, then that strongly degrades my opinion of their competence. [OSS-176]

Furthermore, not all the mistakes have the same impact. Reviewers are more accepting of mistakes due to lack of knowledge or understandings than they are to easily avoidable mistakes.

It's ok if the code contains bugs and I will not think the author is careless. However if the code contains coding style, readability, duplicated code and other easily avoidable problems, I will think the author is careless and is not dedicated to the project. [OSS-197]

6.7.2 Suggests Inexperience

Approximately 51% of the OSS respondents and 24% of the Microsoft respondents indicated that low quality code suggests that the author is inexperienced in either the project or in programming.

> *It can mean that they haven't yet reached understanding of the how the code they are modifying works, and thus any contributions from them may need to be reviewed extremely carefully. [OSS-196]*

Reviewers also think that the authors of low quality code may be incompetent or lacking intelligence.

> *No matter what level of experience a programmer is at they can write clear, readable, robust code. Surprisingly people can go a long ways without learning to do this. It makes me question their native intelligence and dedication to the project. [OSS-25]*

6.7.3 Impacts Future collaborations

Approximately 62% of the Microsoft respondents and 27% of the OSS respondents indicated that the impressions formed about the authors of poorly written code affected their future collaborations. Many of the respondents doubted a the abilities of the authors of poor code.

> *Poorly written code usually indicates to me that the author is lacking coding experience or technical skill, which (negatively) affects how I perceive the author's general performance at work. [MS-45]*

Loss of respect is another factor hampering future collaboration.

> *Depending on the 'severity' of the bad code, I can feel that I lose respect for the person and their intelligence as a code author in extreme cases. For example thinking 'did they even try to build it/test it', or 'why did they think this is the right approach. It should be much simpler but they are to clueless to know'. [OSS-78]*

In the extreme case, reviewers can lose trust in the author of low quality code and carefully examine future changes from that author.

> *I trust the developer less, and know that I'll have to code review future changes in even greater detail. [MS-349]*

Finally, because poorly written code takes longer to review, some reviewers are less likely to accept review requests from these authors.

> *Poorly written changes also require more review time, so I feel that a person who consistently makes poorly written changes will waste a lot of people's time in the long run. I also end up expecting that person's changes to be poorly written and do not look forward to the prospect of reviewing the changes. [MS-166]*

6.8 RQ8: What is code review's effect on peer impressions?

Using behavioral scales, we focused on understanding the impact of code reviews on four aspects of peer impression formation: trust, reliability, perception of expertise, and friendship. To ease analysis and presentation of results, we recoded the scale to make the effect of the scale items on impression formation more evident. The recoded scale is: -3: *describes a non-code review partner, NOT a code review partner*, 0: *describes both equally*, and 3: *describes a code review partner, NOT a non-code review partner*. To avoid biasing the results with negative scale values, we did not use this scale during data collection.

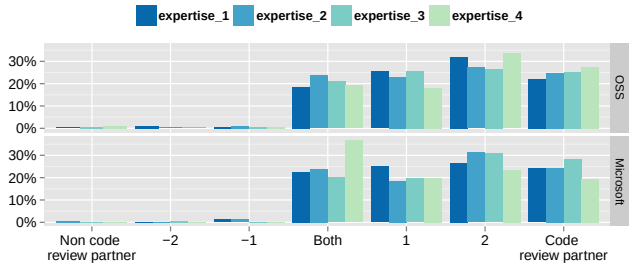

Fig. 9: Distribution of ratings for the scale items: Perception of expertise

TABLE 6: Behavioral scale means and effects

Construct	Scale Mean		Effect Size (Cohen's *d**)	
	OSS	Microsoft	OSS	Microsoft
Trust	0.699	0.843	0.76	0.93
Perception of Expertise	1.526	1.467	1.72	1.53
Reliability	1.023	1.088	1.11	1.04
Friendship	1.473	1.115	1.53	1.33

*Cohen's d interpretation: $d \geq .8$ indicating large effect, $d \geq .5$ indicating medium effect, and $d \geq .2$ indicating small effect [17]

As an example, Figure 9 shows that for the four perception of expertise scale items, most of the respondents (approximately 70% to 80%) thought they had a better perception of the expertise of their *code review partners* than their *non-code review partners*. All four scales exhibited a similar trend.

Table 6 shows the item means for the four behavioral scale items for the two surveys. The scale means were positive and significantly higher[13] than the mid-point of the scale (0 - Both Equally) in all cases. We did not observe any significant differences between the results of the two surveys for the behavioral scale questions. We also estimated effect size using Cohen's d (rightmost two columns of Table 6). In seven out of the eight cases there were large effects. Only the trust scale in OSS survey showed medium effect size. The results suggest that code review had overall large positive impact on building four types of peer impressions (i.e., trust, perception of expertise, friendship, and reliability) between code review participants in both OSS and MS projects.

These results provide some insight into the results from RQ6 (impact of high quality code) and RQ7 (impact of poor code), which show that code reviews can have both positive and negative impacts on impression formation. This analysis shows that 1) code reviews have a large positive impact on impressions formation, and 2) the majority of the respondents had better perceptions of their code review partners than their non-code review partners.

7 Discussion

This section provides further discussion on the detailed analysis of the survey results described in Section 6. In particular, this section highlights seven themes that emerged from the results.

7.1 Differences Between OSS and Microsoft

The OSS respondents differ significantly from the Microsoft respondents in the aspects of code review emphasized as most

[13] one sample t-test, $p < 0.001$ in all cases

important. The focus for OSS reviewers is on building relationships with core team members. When forming impressions of their teammates through code reviews, the OSS respondents indicated that the personal characteristics and work habits of the code author were most important. This emphasis makes sense because members of OSS teams may not have the opportunity to form impressions of their teammates through the more traditional types of interactions (i.e. face-to-face work and social interactions) that members of a commercial organization, like Microsoft, would have. As a result, code reviews become even more important for forming impressions of teammates. Conversely, the Microsoft respondents consider the knowledge dissemination aspects more important. Code reviews work as a medium to mentor new teammates about the project design, coding conventions, and available API or libraries.

Similarly, when deciding whether to accept a code review request, the most important factors for OSS respondents are their relationship with the code author and the reputation of the code author. This focus is driven by the desire to maintain current relationships and to improve relationships with reputed developers. Conversely, for Microsoft respondents, when deciding whether to accept a review, the most important factors were the expertise of the code author (i.e., if a developer writes good code that s/he can learn) and the effort required to review the change. When deciding who to invite to review their code, the most important factor was the expertise of the reviewer (i.e., whether s/he has expertise to review that code and will be able to provide useful feedback).

Other than the differences mentioned here, the results were similar for the OSS and Microsoft developers. Sections 7.2-7.5 describe results that were similar across the OSS respondents and the Microsoft respondents.

7.2 Benefits of Code Review

While there is empirical evidence that code review improves software quality [18], [57], the benefits of code review are much broader. Evidence about these other benefits has been mostly anecdotal. The results of these surveys begin to provide more evidence for these benefits. Nearly all survey respondents in both surveys found code reviews important for their projects for reasons including: knowledge dissemination, relationship building, better designs, and ensuring maintainable code. For large-scale and long term projects, those benefits may be very important and hard to achieve through other means. Therefore, even projects with highly-skilled developers who rarely commit low-quality changes can still benefit from the practice of code review.

7.3 Peer Impression Formation

One of the key non-technical benefits of code review for both OSS and MS participants is its role in impression formation, that is how developers form opinions of teammates. The results of the surveys show that the most important social factor of code reviews is in obtaining an accurate perception of the expertise of teammates. The quality of the code submitted for review is an important aspect of the formation of teammate perceptions. For example a code change that is simple, easy to understand, self documenting and requires minimum review time is highly appreciated by the reviewers and may lead to improved social status. Whether a developer is positive or negative towards a teammate may influence future code reviews. For example, respondents indicated that they perform more thorough reviews of code submitted by teammates who are untrustworthy than of code submitted by teammates they view as experts. In addition, the impressions formed during code review could also affect future collaborations, relationships, and work practices. Therefore, code review is a critical practice not only for ensuring the quality of code changes but also for forming the social underpinning of successful projects.

7.4 Effects on Future Collaborations

More than three-fourth of the code review participants from the both surveys had strong positive impressions about their peers, suggesting that code reviews may influence future collaborations. When a reviewer finds high quality code from an author it can increase collaborations in two ways. First, s/he is more likely to sign up to review the author's future changes in order to learn from them. Second, s/he considers the author an expert who is able to provide suggestions to improve his/her code and add the author as a reviewer for his/her changes. On the contrary, poorly written code often takes more effort to review and a reviewer may not accept future review requests from the author of poor code. Moreover, uncertainty about their level of expertise may lead a developer to avoid asking the author of poor code to review his/her code. Combining these two scenarios, a developer's code review partners are more likely those peers who s/he considers as good authors as well as experts. Conversely, the peers who s/he judged as poor code authors will become non-code review partners due to infrequent interactions.

7.5 Effects of Perceived Expertise

A reviewer's perception of the expertise of the code author not only influence the acceptance or rejection of code review requests but also influences the level of scrutiny for the code. First, in terms of accepting incoming code reviews (See Section 6.3), perceived expertise has mixed influence. Some respondents preferred to review code from experts to learn and to minimize time spent in reviews. Other respondents prioritized reviews from newcomers or focused on areas requiring the most attention. Both of these approaches may be correct, depending upon the expertise of the reviewer.

Second, in terms of the level of scrutiny given to the code, if a reviewer is uncertain about a particular design choice, s/he is more likely to trust the design choice of experts and to question the rationale of non-experts. In addition, if a reviewer does not have adequate time to perform a thorough review of code from an expert author, s/he will approve that code after only a cursory review, based upon an assumption that the expert author implemented correctly, as usual. However, an author only receives this expert status after consistently submitting high-quality code changes.

7.6 Effects of Distributed vs. Collocated Teams

One of the goals of replicating the survey with Microsoft developers was to investigate how much the distributed nature of OSS projects factored in to the understanding and practice of code review. To investigate this effect, we surveyed members of distributed teams and members of collocated teams within Microsoft. We anticipated that members of distributed teams would emphasize the human relationship aspects of code review due to their limited ability to form these relationships in person (as members of collocated teams have). Interestingly, when analyzing the data from the Microsoft survey we found very little difference

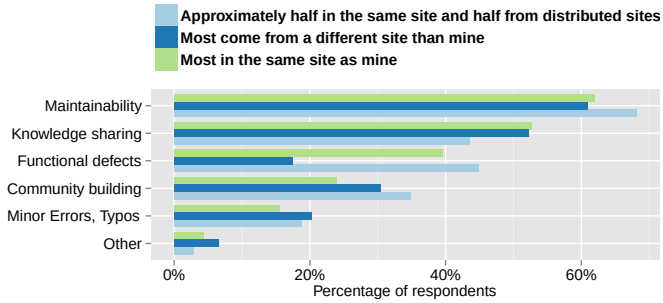

Fig. 10: Importance of code reviews (Grouped by three types of Microsoft respondents)

between the responses of developers from collocated teams and the responses of developers from distributed teams. For example, Figure 10 shows how the three groups of Microsoft respondents considered code reviews to be important for their projects. In fact, there were much larger differences between respondents to the OSS survey and respondents to the Microsoft survey, in general (regardless of whether the respondent was on collocated team or a distributed team), as discussed in Section 7.1. Therefore, our initial hypothesis that respondents from distributed teams at Microsoft would respond similarly to respondents from OSS teams is not supported.

This surprising result suggests two possible explanations: 1) some types of impressions that are impacted by code reviews (e.g. coding ability) may not depend on face-to-face interactions, and 2) the code review process (e.g., review acceptance and future collaborations) may depend more on project culture (i.e. OSS projects have a different culture than commercial projects [28]) than on the physical location of the developers.

7.7 Paid vs. Volunteer OSS developers

The motivation of OSS participants may be affected by whether or not they receive financial compensation for their contributions. In addition, paid OSS participants may have different goals than volunteer OSS participants. Prior research has identified several differences between paid and volunteer participants in terms of impressions formation (i.e., paid participants were more likely to form impressions based on meeting in person) and perceived experiences (i.e., volunteers were more likely to perceive negative experiences working with peers) [10].

In response to RQ2, we found that paid OSS participants collaborate with significantly more peers and spend significantly more time in code reviews than volunteer OSS participants. This observation may indicate that paid OSS participants serve as the gatekeepers for the OSS projects.

For the remaining research questions, which dealt with the importance of code review, the code review process, and the impact of code reviews, the results did not show any differences between the paid OSS participants and the volunteer OSS participants. Therefore, whether an OSS participant is paid or is a volunteer does not seem to impact the key aspects of the code review process or how code reviews impact peer impression formation.

8 Threats to Validity

This section discusses the addressed and unaddressed threats to validity. It is organized around the four common types of validity threats.

8.1 Internal validity

Participant selection is the primary threat to internal validity. The subject population consisted of reviewers who had participated in at least 30 code reviews (either as the author or reviewer). It is possible that using a different threshold would have produced different results, but we have no evidence to suggest this situation. Because seven of the eight research questions (RQ2-8) are related to the code review process (i.e., accepting review request, judging poor code, improving poor code, impact of good/bad codes), we strongly believe that without having adequate code review experiences, a developer cannot provide appropriate answers to these questions.

In addition, there is the threat that only those subjects who had positive experiences with code review took time to respond to the survey. There is no evidence to suggest that this self-selection occurred. But, even if it did, because the goal of the survey was to gather information about various aspects of code review, those who had positive experiences could likely provide the best feedback.

8.2 Construct validity

The survey design process specifically focused on reducing construct validity threats. This process took approximately eight months and included both expert reviews and multiple pilot tests. The design process included the following bias-reducing practices:

- placing the questions about the topics of interest after the other survey questions to prevent *hypothesis-guessing*,
- presenting the scale questions in random order,
- providing clear definitions of *code review partner* and *non-code review partner* on all relevant pages, and
- carefully wording questions in an unbiased manner.

Third, we conducted multiple reliability and validity tests, with widely-used and highly recommended measures, to ensure construct validity.

8.3 External validity

Due to the wide diversity within the OSS community, it is possible that the results may not be representative of all OSS projects. In fact, as most respondents came from well-known, successful OSS projects, they may have been among the higher skilled and more motivated OSS developers. The impacts of code review on software quality and on the social fabric of the team may differ in other types of OSS projects.

In terms of the Microsoft developers, they may not be representative of all commercial organizations. To reduce this potential threat, the respondents came from teams that differ in *development process* (e.g. waterfall vs. agile), *hardware platform* (e.g., mobile, desktop, server, and data center software), *deployment method* (boxed products versus web services), *operating system* (iOS, Windows, Windows Phone, and Linux), *location* (U.S., Europe, and Asia), and *workflow* (e.g., some teams require two reviews on all sign-offs, others are more lax; some want review prior to checkin and test, others do review afterwards; some include testers and development leads on reviews and some do not). The software development processes, project management, and release cycles across different projects in Microsoft are quite varied. Interestingly, in a prior study of code review, Rigby and Bird investigated code review practices and metrics in multiple

commercial organizations and open source projects [45]. Surprisingly, they found little difference between the different systems studied in terms of code review, supporting the notion that findings about code review at one company may be relevant for other organizations.

In addition, the code review workflow at Microsoft is similar to that used by other large commercial organizations such as Facebook [33], Google [56], VMWare [3], Cisco [18], and Oracle [55] that have adopted mandatory code review practices. In those organizations, code review has become an important software quality assurance practice similar to testing, tracking/fixing bugs, and automated build systems, which all are aspects of mature software engineering projects.

A common misconception about industrial research at large companies such as Microsoft is that software projects at Microsoft are not representative of other software projects. While projects might be larger in size, most development practices at Microsoft are adapted from the general software engineering community and also used outside Microsoft. Another frequent misconception is that empirical research within one company or one project is not good enough, provides little value for the academic community, and does not contribute to scientific development. Historical evidence shows otherwise. Flyvbjerg provides several examples of individual cases that contributed to discovery in physics, economics, and social science [24]. Again, W.I. Beveridge observed for social sciences: "*More discoveries have arisen from intense observation than from statistics applied to large groups*" [7]. Even in SE domain, prior case studies at large commercial companies such as: Microsoft [9], [39], Google [35], and Cisco [18] have provided useful insights. Please note that this argument should not be interpreted as a criticism of research that focuses on large samples or entire populations. For the development of an empirical body of knowledge as championed by Basili [4], both types of research are essential.

8.4 Conclusion validity

The number of responses to each survey was sufficiently large to mitigate any threats arising from small sample sizes. In addition, the Chi-square test (used most frequently in this analysis) does not assume normality in the data. For variables that were not normally distributed (according to the Shapiro-Wilk test), we used non-parametric tests.

9 Conclusion

This paper describes the results of two surveys to better understand the practice and motivation for performing code reviews. These results have several implications for researchers and for practitioners. First, although only one-fourth of the code review comments are about functional defects, practitioners should not be discouraged to practice code reviews. Code review offers several other benefits (i.e., knowledge dissemination, relationship building, better designs, and ensuring maintainable code) that are crucial for large scale or long term projects. Interestingly, most code review research focuses on defect detection. These other aspects of code review that are considered more important by the developers have not received much attention. Therefore, these other aspects of code reviews (i.e., relationship building, knowledge sharing, achieving better designs) warrant additional focused research.

Second, the results of these surveys indicate that code reviews have a large impact on relationship building and future collaborations. Carelessness in wording a review comment can lead to negative feelings from the code author and hinder future collaborations. For example, an author is more likely to make the required changes if the reviewer provides constructive criticism and is more likely to argue with the reviewer if the reviewer comments are viewed as an attack. Therefore, reviewers should carefully consider how their review comments will be heard by the code author. This finding warrants further research in two directions: 1) empirical validation of how the expression of sentiment (i.e., positive or negative) in code review comments influences the code review outcomes and long term collaborations, and 2) how to assist reviewers in articulating appropriate comments during code reviews.

Finally, effective code reviews require a significant amount of effort from the reviewers to thoroughly understand the code. The results of this study suggest that reviewers prefer to review code changes that are simple, self-documenting and easy to comprehend. Authors should keep those code characteristics in mind when submitting code changes for review. This result could also be of interest to program comprehension researchers. The large amount of time devoted to understanding code changes could be improved with appropriate program comprehension techniques.

Acknowledgments

This research is partially supported by the US National Science Foundation Grant No. 1322276, 1156563. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would also like to thank Brook Bowers and Luis Aguiar for assistance in data analysis.

References

[1] J. Asundi and R. Jayant, "Patch review processes in open source software development communities: A comparative case study," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, 2007, pp. 166c–166c.

[2] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 712–721.

[3] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 2013 International Confernece on Software Engineering*, 2013, pp. 931–940.

[4] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456–473, 1999.

[5] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review," in *Proceedings of the 20th Working Conference on Reverse Engineering*, 2013, pp. 122–131.

[6] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 202–211.

[7] W. I. B. Beveridge *et al.*, "The art of scientific investigation." *The art of scientific investigation.*, 1950.

[8] C. Bird, T. Carnahan, and M. Greiler, "Lessons Learned from Building and Deploying a Code Review Analytics Platform," in *Proceedings of the 12th International Conference on Mining Software Repositories*, 2015, pp. 191–201.

[9] C. Bird and T. Zimmermann, "Assessing the value of branches with what-if analysis," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 45.

[10] A. Bosu, J. Carver, R. Guadagno, B. Bassett, D. McCallum, and L. Hochstein, "Peer impressions in open source organizations: A survey," *Journal of Systems and Software*, vol. 94, pp. 4 – 15, 2014.

[11] A. Bosu and J. C. Carver, "Impact of peer code review on peer impression formation: A survey," in *Proceedings of the 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 133–142.

[12] ——, "How Do Social Interaction Networks Influence Peer Impressions Formation? A Case Study," in *Open Source Software: Mobile Open Source Technologies*, ser. IFIP Advances in Information and Communication Technology, L. Corral, A. Sillitti, G. Succi, J. Vlasenko, and A. Wasserman, Eds. Springer, Berlin, Heidelberg, 2014, vol. 427, pp. 31–40.

[13] ——, "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 33:1–33:10.

[14] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 146–156.

[15] L. C. Briand, B. Freimut, and F. Vollei, "Using multiple adaptive regression splines to support decision making in code inspections," *Journal of Systems and Software*, vol. 73, no. 2, pp. 205–217, 2004.

[16] W. M. Bukowski, B. Hoza, and M. Boivin, "Measuring friendship quality during pre-and early adolescence: The development and psychometric properties of the friendship qualities scale," *Journal of Social and Personal Relationships*, vol. 11, no. 3, pp. 471–484, 1994.

[17] J. Cohen, *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum, 1988.

[18] J. Cohen, E. Brown, B. DuRette, and S. Teleki, *Best Kept Secrets of Peer Code Review*. SmartBear Software, 2006.

[19] J. Czerwonka, M. Greiler, and J. Tilford, "Code reviews do not find bugs. how the current code review best practice slows us down," in *Proceedings of 2015 International Conference on Software Engineering - Companion*, May 2015, pp. 27–28.

[20] R. F. DeVellis, *Scale development: Theory and applications*. Sage Publications, 2011, vol. 26.

[21] M. Fagan, "A history of software inspections," *Software Pioneers*, pp. 562–573, 2002.

[22] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[23] A. Fink, *The survey handbook*. Sage Publications, 2003, vol. 1.

[24] B. Flyvbjerg, "Five misunderstandings about case-study research," *Qualitative Inquiry*, vol. 12, no. 2, pp. 219–245, 2006.

[25] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[26] T. Gilb, D. Graham, and S. Finzi, *Software inspection*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[27] L. Harjumaa, I. Tervonen, and A. Huttunen, "Peer reviews in real life-motivators and demotivators," in *Fifth International Conference on Quality Software*. IEEE, 2005, pp. 29–36.

[28] E. v. Hippel and G. v. Krogh, "Open source software and the "private-collective" innovation model: Issues for organization science," *Organization science*, vol. 14, no. 2, pp. 209–223, 2003.

[29] P. M. Johnson, "Reengineering inspection," *Communications of the ACM*, vol. 41, no. 2, pp. 49–52, 1998.

[30] C. Johnson-George and W. C. Swap, "Measurement of specific interpersonal trust: Construction and validation of a scale to assess trust in a specific other." *Journal of Personality and Social Psychology*, vol. 43, no. 6, p. 1306, 1982.

[31] K. Lakhani, B. Wolf, J. Bates, and C. DiBona, "The boston consulting group hacker survey," *Boston, The Boston Consulting Group*, 2002.

[32] K. Lakhani and R. G. Wolf, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," *MIT Sloan working paper*, 2003, paper No. 4425-03.

[33] Y. Lee. (2011) How Facebook ships code. [Online]. Available: https://framethink.wordpress.com/2011/01/17/how-facebook-ships-code/

[34] J. Lerner and J. Tirole, "Some simple economics of open source," *The Journal of Industrial Economics*, vol. 50, no. 2, pp. 197–234, 2002. [Online]. Available: http://www.jstor.org/stable/3569837

[35] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead, "Does bug prediction support human developers? Findings from a Google case study," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 372–381.

[36] M. S. Litwin, *How to measure survey reliability and validity*. Sage Publications, 1995, vol. 7.

[37] D. J. McAllister, "Affect- and cognition-based trust as foundations for interpersonal cooperation in organizations," *The Academy of Management Journal*, vol. 38, no. 1, pp. pp. 24–59, 1995.

[38] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 192–201.

[39] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The Design Space of Bug Fixes and How Developers Navigate It," *IEEE Transactions on Software Engineering*, 2015.

[40] A. Porter, L. G. Votta Jr, V. R. Basili *et al.*, "Comparing detection methods for software requirements inspections: A replicated experiment," *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp. 563–575, 1995.

[41] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.

[42] E. S. Raymond, "Homesteading the noosphere," 1998.

[43] J. K. Rempel, J. G. Holmes, and M. P. Zanna, "Trust in close relationships." *Journal of Personality and Social Psychology*, vol. 49, no. 1, p. 95, 1985.

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

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

[46] P. C. Rigby and D. M. German, "A preliminary examination of code review processes in open source projects," University of Victoria, Tech. Rep. DCS-305-IR, January 2006.

[47] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 541–550.

[48] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA: ACM, 2011, pp. 541–550.

[49] J. Robinson, L. Wrightsman, and F. Andrews, *Measures of personality and social psychological attitudes*. Academic Press, 1991, vol. 1.

[50] D. Rombach, M. Ciolkowski, R. Jeffery, O. Laitenberger, F. McGarry, and F. Shull, "Impact of research on practice in the field of inspections, reviews and walkthroughs: learning from successful industrial uses," *ACM SIGSOFT Software Engineering Notes*, vol. 33, no. 6, pp. 26–35, 2008.

[51] J. B. Rotter, "A new scale for the measurement of interpersonal trust1," *Journal of Personality*, vol. 35, no. 4, pp. 651–665, 1967.

[52] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, pp. 591–611, 1965.

[53] A. Sutherland and G. Venolia, "Can peer code reviews be exploited for later information needs?" in *Proceedings of the 31st International Conference on Software Engineering-Companion*, 2009, pp. 259–262.

[54] L. G. Votta, Jr., "Does every inspection need a meeting?" in *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering*, 1993, pp. 107–114.

[55] J. Waldo. (2009) Code reviews. [Online]. Available: https://blogs.oracle.com/scalinggames/entry/code_reviews

[56] M. Welsh. (2012) My love affair with code reviews. [Online]. Available: http://matt-welsh.blogspot.com/2012/02/my-love-affair-with-code-reviews.html

[57] K. E. Wiegers, *Peer reviews in software: A practical guide*. Addison-Wesley Boston, 2002.

[58] C. Wohlin, M. Höst, and K. Henningsson, "Empirical research methods in software engineering," in *Empirical Methods and Studies in Software Engineering*, ser. Lecture Notes in Computer Science, R. Conradi and A. Wang, Eds. Springer Berlin Heidelberg, 2003, vol. 2765, pp. 7–23.