"Was my contribution fairly reviewed?" A Framework to Study the Perception of Fairness in Modern Code Reviews

Daniel M. German
University of Victoria, Canada
dmg@uvic.ca

Gregorio Robles
Universidad Rey Juan Carlos, Spain
grex@gsyc.urjc.es

Germán Poo-Caamaño
University of Victoria, Canada
gpoo@uvic.ca

Xin Yang
Osaka University, Japan
xinyang@ist.osaka-u.ac.jp

Hajimu Iida
Nara Institute of Technology, Japan
iida@itc.naist.jp

Katsuro Inoue
Osaka University, Japan
inoue@ist.osaka-u.ac.jp

ABSTRACT

Modern code reviews improve the quality of software products. Although modern code reviews rely heavily on human interactions, little is known regarding whether they are performed *fairly*. Fairness plays a role in any process where decisions that affect others are made. When a system is perceived to be unfair, it affects negatively the productivity and motivation of its participants. In this paper, using fairness theory we create a framework that describes how fairness affects modern code reviews. To demonstrate its applicability, and the importance of fairness in code reviews, we conducted an empirical study that asked developers of a large industrial open source ecosystem (OpenStack) what their perceptions are regarding fairness in their code reviewing process. Our study shows that, in general, the code review process in OpenStack is perceived as fair; however, a significant portion of respondents perceive it as unfair. We also show that the variability in the way they prioritize code reviews signals a lack of consistency and the existence of bias (potentially increasing the perception of unfairness). The contributions of this paper are: (1) we propose a framework—based on fairness theory—for studying and managing social behaviour in modern code reviews, (2) we provide support for the framework through the results of a case study on a large industrial-backed open source project, (3) we present evidence that fairness is an issue in the code review process of a large open source ecosystem, and, (4) we present a set of guidelines for practitioners to address unfairness in modern code reviews.

KEYWORDS

Fairness, Software Development, Modern Code Review, Open Source Software, Human and Social Aspects, Transparency

ACM Reference Format:
Daniel M. German, Gregorio Robles, Germán Poo-Caamaño, Xin Yang, Hajimu Iida, and Katsuro Inoue. 2018. "Was my contribution fairly reviewed?" A Framework to Study the Perception of Fairness in Modern Code Reviews. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180217

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180217

1 INTRODUCTION

The goal of modern code reviews is to ensure the high quality of software [1, 4, 46, 52]. Code reviewing is a process in which a reviewer makes a decision on somebody else's contribution. As with many other human intensive activities, code reviewing is prone to bias; reviewers might have their personal points of view, interests and circumstances that affect the decisions that they make [6, 19, 38, 61].

When an author has a patch rejected, it is natural to ask if the decision was *fair*. If an author has the perception that her patch was treated unfairly, it might lead towards resentment and might result in actions that are not in the best interest of the project, such as selfishness, retaliation and leaving the project [9, 17]. These actions might affect the long term success and survival of a software project, especially Open Source Software (OSS) [49]. In OSS projects, it is reasonable that its participants expect to be treated *fairly*, even if they might not agree on what *fair* is. In projects that use code reviews, the review process becomes not only a way to guarantee quality, but it could also stifle some contributors —eventually turning them away.

One important goal of any system that involves human decisions is to guarantee that the allocation of resources, and decisions are fair to both, the participants and the system as a whole. Fairness involves many different aspects. One of them is bias; however, a fair system can be biased towards certain participants; for example, it is widely accepted that important bug fixes should be reviewed first, or that reviewers should give more constructive comments to newcomers than regular contributors. Fairness is also concerned with the creation of procedures and appealing mechanisms, the manner in which actors interact, etc.

We have developed a framework grounded on fairness theory to explain fairness in modern code review process. A theoretical framework is the structure that makes the context of a case study clear, particularly when introducing a theory from a different field [44, 54]. We use this framework to conduct an empirical study of the perception of fairness in the code review process of a major OSS project.

The contributions of this paper are: we present (1) a framework—based on fairness theory—for studying and managing social behavior in modern code reviews; (2) support for the framework through the results of a case study on a large industrial-backed open source project, (3) evidence that fairness is an issue in the modern code review process of a major industrial OSS ecosystem; and, (4) a set of guidelines to address unfairness in modern code reviews.

2 ON THE CONCEPT OF FAIRNESS

Any time individuals form a group, from very small to entire societies, they face the challenge of allocating resources and rewards among its members. Thus, individuals are continuously subject to the decisions of others. These decisions have "both economic and socioemotional consequences" [16], and "the way that the group deals with these issues has a great impact on its effectiveness, and on the satisfaction of its members" [43]. The importance of these decisions prompts those subject to a decision to ask: "Was that fair?" [16].

At the core of this issue is the notion of *fairness*: how to decide and allocate resources in a way that is fair and just to the individuals and the group as a whole. Philosophy, psychology, law, sociology, management sciences and economics –among many other disciplines– have extensively researched these issues in what is known as justice theory (also known as fairness theory) [18] (fairness and justice are synonyms in these fields). According to Folger and Cropanzano [24, 25] fairness theory –within the context of organizations– "predicts that individuals will react negatively to decision making events when the decision maker (1) could have done something differently; (2) should have done something differently; or (3) the current state of well-being would have been better if the event had played out differently." and focuses on "explaining when authorities should be held accountable for unfavorable events."

In particular, organizational justice is the study of fairness within an organization; it is divided into four main areas of study [16, 17, 32] (see Table 1): (1) **Distributive Fairness** concerns the allocation of outcomes (decisions and rewards) according to the participation of the individuals in the system [42], (2) **Procedural Fairness** concerns the decision process that leads towards an outcome, (3) **Interactional Fairness** is defined as "the interpersonal treatment of people" [8] and the degree to which the people affected by an outcome are treated with dignity and respect [66]; and, (4) **Informational Fairness** focuses on the dissemination of information about why procedures exist and how outcomes are reached [28].

2.1 Distributive Fairness

Distributive Fairness (also known as Distributive Justice) is concerned with how to balance the distribution of outcomes with the amount of participation of individuals in a system. According to Cohen, distributive fairness is concerned with four dimensions [15]: (1) outcomes are allotted to (2) persons, whose relative shares can be described (3) by some functional rule, and by (4) some standard.

In other words, the decision to allot an outcome (including a resource, a treatment, or a decision) to an individual might depend on both, her contribution to the system, and certain minimal standards as defined by the system. These latter two aspects highlight the fundamental dichotomy of distributive fairness: it must find a balance between **equity** (an individual who contributes more deserves more) and **equality** (every individual deserves the same outcomes, regardless of contribution). Organizations favor equity because it rewards those who contribute more to the organization [42]. However, in an organization that emphasizes equity, there is still a minimal **need** of its members to receive an allocation of outcomes in order to guarantee their continuing participation in the system. Distributive fairness is also concerned with how **newcomers** should be treated, as they have not had an opportunity to contribute to the system [42].

Equity's fundamental challenge is how to effectively measure the contributions of an individual. Imperfection of this measurement might prompt participants to "game" the system. Also, equity might emphasize selfishness in detriment of the well-being of the overall system. In practice, an organization might implement equity in some aspects (e.g., salary) and equality in others (e.g., benefits). Distributive fairness attempts to address the challenges created by self-interest ("if I contribute more, I should receive more") in favor of a long term benefit that comes from the restrain of this self-interest in benefit of others [72].

2.2 Procedural Fairness

Procedural fairness refers to the perceptions of fairness in the procedural components used to determine outcomes in a system and focuses on the events that precede the determination of the outcome [43]. The determination of the outcome is done by decision makers who control the process. When this process is perceived as fair, the final distribution is likely to be accepted as fair—even in the presence of bias (e.g., such as existence of equity) [43].

Procedural fairness relies upon the application of several rules. When these rules are upheld, the procedure is perceived to be fair [16]. These rules are: 1) **consistency**: the process is applied consistently across all instances of decisions and across time; 2) **bias suppression**: decision makers are neutral; 3) **accuracy of information**: the decision is made based on accurate information; 4) **control**: those affected by the decision have a voice that can alter the outcome; 5) **correctability**: there exist appeal processes for correcting bad outcomes; and, 6) **ethicality**: the process upholds personal and organizational standards of ethics and morality.

Procedural fairness requires the documentation of the decision process, including how to select those making the decisions, and a clear establishment of policies and guidelines that cover the rules cited above. Note that consistency and bias suppression run orthogonal to equity. Equity presupposes a bias: those that contribute more should receive more; within this bias, procedural fairness should strive to be consistent across all decisions and without (other) bias.

Accuracy of information implies that decisions should be made based upon correct information. A transparent decision process allows the verification of its accuracy. We will discuss information transparency further under Informational Fairness.

Control and Correctability give a voice to those affected by the outcome. Control implies that the decision process allows those affected to add or correct information or to question the decision process (while in progress). Correctability implies that those affected by a decision can appeal the process, ideally to a different decision maker. Having a voice in the decision process has been found to improve the satisfaction and perception of fairness in a system (known as the *fair voice effect*) [26].

Ethicality implies a certain common set of values in the organization. An agent that acts unethically favors certain individuals or ideas over others against the set of values of the system. Ethicality overlaps with Interactional Fairness (described below).

To be effective, procedural fairness requires a set of policies that serve both as a guideline on how a process should be performed, and a social contract indicating the responsibilities and obligations of those making decisions, and how they can be appealed. And it also requires that decision makers abide by these policies.

Table 1: Dimensions of Fairness in organizational justice. Derived from Colquitt [16, 17].

	Rules	Process
Distributive Fairness	Equity	Resources, treatment and outcomes are distributed in accordance with one's contribution
	Equality	All participants deserve the same resources, treatment and outcomes regardless of contribution
	Need	The resources, treatment and outcome meet the minimal requirements of the recipient
	Newcomers	Newcomers should receive a positive bias
Procedural Fairness	Control	The abilities to (1) voice one's views and (2) influence the outcome
	Consistency	The process is applied consistently across time and persons
	Bias suppression	Decision makers are neutral
	Information Accuracy	Decisions are not based on inaccurate information
	Correctability	Appeal procedures exist for correcting bad outcomes
	Ethicality	The process upholds personal and organizational standards of ethics and morality
Interactional Fairness	Respect	Actions and signals should reflect the intrinsic value that a person has for another
	Psychological safety	A mechanism to prevent harassment and protect victims when unacceptable treatments occur
Informational Fairness	Truthfulness	Need to avoid deception
	Adequacy of explanations	Need of individuals to receive explanations that are reasonable and timely

2.3 Interactional Fairness

Interactional fairness is based on the fact that individuals are not only concerned with outcomes, but also with the way they are treated during the process that reaches those outcomes [9]. Its main concern is that individuals should be treated with dignity and respect during the entire decision process.

Bies argues [9] that people have a view of themselves that is *sacred* and when this view is violated, it results in a painfully intense experience; and that there are three main violations to this dignity that are of concern: disrespect, invasion of privacy and exposure to personal danger. **Respect** are actions and signals that reflect the intrinsic value that a person has for another and disrespect are actions and signals that demean the perceived value of an individual (her own perceived value –self-respect– or her value as perceived by others–the respect of others) [29]. The most typical forms of disrespect are inconsiderate actions (e.g., not receiving timely feedback, not receiving an explanation for a decision), abusive words or actions (e.g., rudeness, public criticism, berating –that humiliates a person, or prejudicial statements– sexism or racism). **Invasion of privacy** is concerned with the disclosure of secrets, including asking improper questions. Finally, exposure to personal danger refers –within the organization context– to actions that violate the **psychological safety** of the individual (e.g., increase of stress due to an abuse) [21].

2.4 Informational Fairness

Informational fairness is concerned with two dimensions: truthfulness and adequacy of explanations. **Truthfulness** is the need to avoid deception, as people who feel "lied to" become resentful [9]. Truthfulness also includes not having others stealing somebody else's ideas and/or divulging false information that might incorrectly affect other's perception of a person. **Adequacy of explanations** is the need of individuals to receive explanations that are perceived to be "adequate" (reasonable, timely, and specific) [16].

3 FAIRNESS AND CODE REVIEWS

Code reviewing is a decision process. Fundamentally, the decision to accept or not a patch should be based on the merits of the patch itself. However, how this process is performed and how the different participants interact can have negative effects on the process and long term success of the system. For these reasons, fairness is an intrinsic property of code reviewing. How the review is conducted (treatment), the resources it gets (its resource allocation), and how the person submitting the patch is treated by the decision makers (the reviewers) are issues of concern to fairness theory. Applying fairness theory to code reviews has the potential to increase the satisfaction, effectiveness and productivity of its participants, the quality of these reviews, and overall, the success of the system.

In the rest of this section we describe a framework that we have developed to describe how the different types of fairness can be applied to code reviewing (see Table 2). Many of these issues are already addressed in many code review processes currently in use. This framework can be used to analyze these processes with the goal of identifying their strengths and potential areas of improvement.

3.1 Distributive Fairness

A system that favors **equality** in code reviews would give every patch contributor and every patch the same allocation of resources and the same treatment. In the context of the contributor, who the author of the patch is should not affect how the review is performed (its prioritization, the quality of the review, the overall time to complete the review, etc.). In the context of a patch, equality would imply that every patch would have to be treated equally (e.g., a critical bug fix should not be prioritized first).

A system that favors **equity** assumes that who the author of the patch is/or characteristics of the patch should affect the allocation of resources and treatment. Equity at the contributor level would imply that those that contribute more (e.g., those who author more code or more difficult code, or those who donate more money to the project, etc.) should receive preferential treatment in the review process. Similarly, equity at the patch level would imply that patches that contribute more (e.g., implement a more desirable feature, fix a more important defect, etc.) should receive better treatment.

As mentioned before, equity has been found to favor productivity [31]. It can be argued that the measure of contribution of a patch is its importance: a more important patch deserves more than a less important one. It is natural to expect that a code review system would consider—with regards to the treatment of patches—equity as a more desirable property than equality. With respect to the treatment of individuals, it is less obvious if a system should benefit those that contribute more (e.g., core developers *versus* less frequent contributors). A system might give preferential treatment to core developers [10] (e.g., their patches should be prioritized) and/or to

Table 2: A framework of fairness applied to Modern Code Reviews

Categories	Rules	Concerns
Distributive Fairness	Equity —Patch level	What types of patches are *important* and should receive preferential treatment
	Need —Patch level	What is the *minimal* treatment that a patch should receive
	Equity vs Equality–Individuals	Decide if those who *contribute more* should be given preferential treatment, and how to measure *contribution*
	Newcomers	Decide what preferential treatment to give them
Procedural Fairness	Consistency and bias suppression	Define standards of quality of treatment and requirements for successful patches
	Control and Information Accuracy	Allow authors and other non-reviewers to comment on the review in progress
	Correctability	Create an appeal process for rejected reviews
	Ethicality	Define ethical standards regarding how reviews should be conducted, including potential conflict of interest
Interactional Fairness	Respect Psychological safety	Define a Code of Conduct and a mechanism to enforce it
Informational Fairness	Truthfulness	Conduct code reviews with full, ongoing transparency to all involved parties, including the publication of metrics
	Adequacy	Reviewers should provide good feedback; explain delays and unexpected treatment and outcomes in a timely manner

newcomers –who should have their patches reviewed in a more detailed and constructive way than seasoned developers. However, giving preferential treatment to core developer might deter non-core developers from participating [40].

Nonetheless, a review system must satisfy a minimal level of **need**. Hence, need determines the minimal level of quality a code review should have, such as being reviewed by a minimum number of reviewers, should not stay in the reviewing queue for too long, and/or should have a minimal level of quality in their feedback [77].

Another aspect relevant to distributive fairness is the distribution of the decision making power to reviewers. Some systems award to certain reviewers more weight than others (such as core reviewers in OpenStack and super-reviewers in Apache [30, 52] whose votes weight more than other reviewers), giving them the ability to have a higher influence on the direction of the project. Similarly, some systems might only allow certain contributors to be reviewers.

3.2 Procedural Fairness

A basic prerequisite of procedural fairness in code reviews is the enactment of policies that indicate who the reviewers are (the decision makers), and how the code reviews should be performed. Usually, the selection of reviewers is clearly documented; in some cases, a system might allow any developer to review, in others, reviewers are selected by vote or by appointment in the organization [74, 79]. As mentioned above, some reviewers might have more weight than others in the review process.

Documenting the way to perform reviews should address multiple aspects, including: how to select a patch to review (prioritization), the minimal standards that a review should satisfy, the requirements that a patch should satisfy to be accepted, etc. Organizations might already have some of these policies in place [68].

Consistency and **bias-suppression** apply primarily to prioritization of reviews, and the expected quality of treatment for a patch under review. Prioritization is mainly concerned with the order in which reviewers should process patches. This ordering should be well documented and applied consistently. For example, due to equity, there might exist different classes of patches, each with a different priority; consistency implies that within each class patches should be prioritized according to their time of arrival (or any other property of the patch). The expected quality of treatment determines what an author of a patch should expect from a review, and what a reviewer is expected to do, including how to provide feedback in case of rejection, especially to newcomers [60]. Bias suppression indicates undesirable treatments to avoid. Individuals can be used to do quality control to help reach consistency and bias-suppression, and could anonymize the authors or reviewers of the patch.

Regarding **control**, the desirability of the *fair voice effect* implies that the code review process should allow the author of the patch to contribute its discussion before a decision is made. Control is usually satisfied by allowing the author to reply to reviewers' comments; reviewers can consider this rebuttal (if submitted) before making a decision. Rebuttals will also improve **information accuracy** as it will be possible to correct misunderstandings and consider any missing information.

Correctability implies that there is an appeal process in place for those situations where the author feels that the decision was wrong. Such process might include some type of penalty to reviewers who are found at fault during the review process.

With respect to **ethicality**, it should be expected that reviewers only accept *good* patches, although determining *what a good patch is* is highly subjective. As stated above, ethicality also implies a set of organizational values and practices (explicit or implicit). Approval of own-patches by core developers, or bias towards colleagues of the same organization might be perceived as ethical in some systems and unethical in others; what is important is to know if this is acceptable behavior or not. Nonetheless, the existence of an acceptable bias should not imply lowering the quality of reviews (i.e., accepting bad patches from oneself or somebody from the reviewer's organization). Another example of unethical behavior is two reviewers colluding to have the patches of each other accepted, irrespective of their quality.

3.3 Interactional Fairness

Of all the types of fairness, interactional fairness is arguably the one that has been addressed the most in code reviews –and in software development in general– in the form of **Codes of Conduct**, which are expected to cover dignity, respect and psychological safety. Tourani *et al.* [70] found that codes of conduct are becoming pervasive in OSS projects; they usually require participants to be respectful and avoid disrespectful language (such as insults, sexism, violence). In most cases, they offer a mechanism to report violations.

Code reviews place the author of the code being reviewed in a perilous situation. While it is the code that is being judged, the author might feel –by extension– that her qualities (as a software developer) are also being judged. Code reviews require language that concentrates on critiquing the code, not the person. Linus Torvalds has been criticized for his verbal abuse and public shaming of other kernel developers [12], leading to some developers leaving the project [57]. In response, Linux adopted a "Code of Conflict" that emphasizes an adversarial process of "critique and criticism" that has been "proven to create the most robust operating system kernel ever"; it does plea, however, for respect and psychological safety [13, 55].

Many codes of conduct refer violators to a committee who decides what action should be taken [70]. However, there is not a lot of experience on how to act in such instances. In fact, the punishment (or lack of) of a violator can trigger feelings of unfairness (either for the one being judged or for the accusers). This is exemplified recently in Node.js, where one of its technical leaders was reported to have violated its code of conduct; the process and resolution of this complaint has been acrimonious and has lead to a split of its community [35].

3.4 Informational Fairness

Since their conception, modern code reviews have strived to be open [33]. The evaluation of the code is done in an open environment, accessible to the author, the reviewers and the rest of the development team. Usually, OSS projects make sure that all decisions are documented for others to evaluate. The use of code review metrics is becoming a common method to guarantee informational fairness [34]. Such procedures maximize **truthfulness** in the code review process. **Adequacy** in code reviews requires specific feedback, in a timely manner (requirements of procedural fairness also). If a review takes too long or has an unexpected outcome, an explanation decreases the likelihood that the author feels unfairly treated.

4 CASE STUDY

We conducted an empirical study [53] on OpenStack to evaluate the role of fairness in its code review process. OpenStack is an OSS cloud computing platform initiated as a joint project of Rackspace Hosting and NASA composed of 21 different official components, each developed as an independent, but all sharing common development infrastructure. As of today, more than 600 companies, including major players, such as Ericsson, Hauwei, HP, IBM, Intel, among others, actively contribute to the ecosystem [62]. We chose OpenStack as our case study due to its perceived novelty, its high inter-networked nature that involves many firms and individual contributors, its heterogeneity involving both start-ups and high-tech corporate giants, its market-size ($1.2bn in revenues in 2015, which may grow to $3.4bn by 2018, as claimed by 51 Research [37]), and its size (over 60,000 community members [63], over 500,000 code reviews [65] and more than 5 million lines of code [64]).

We conducted a qualitative study [56] to better understand the perception of fairness that authors and reviewers might have. We asked OpenStack developers about their perception of whether the code review process of OpenStack was fair, and whether they had experienced unfairness (as an author) or acted unfairly (as a reviewer). For this purpose we conducted a web survey composed of four questions, taking into consideration the design and implementation procedures suggested by Punter *et al.* [48].

We selected as potential respondents all OpenStack developers who had participated in code reviews in the last four years (as authors of code to be reviewed or as reviewers). We invited the 2,870 OpenStack developers who matched the previous criteria to participate in the survey via email. The survey was open for two weeks. We received 213 responses (response rate of 7.4%, above the average in Software Engineering studies [58]), 208 were both authors and reviewers (4 were only authors and 1 only reviewer). The respondents were from eight different official projects of OpenStack.

For each question, respondents were invited to add comments and clarifications in (free) textual form. These open-ended responses were analyzed by the authors and categorized into the types of fairness of code reviews presented in Section 3. We followed a case study protocol [76] to mitigate the bias of the categorization. Two authors of this paper performed the coding separately, then their results were cross-validated, and with the help of a third author to achieve agreement.

5 RESULTS

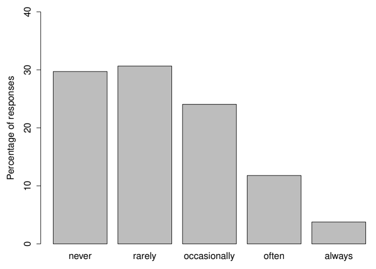
Since our goal was to explore if there was perception of unfairness and what it was, we kept our questions simple and we did not define what *fair* was. The questions were: (1) Have your contributions been treated fairly? (2) Have you treated others' contributions fairly? (3) Do you think the project process is fair? (4) When you review code, how do you prioritize what contribution to review? Each question had an answer in a 5-point Likert scale. In addition, for each question, respondents were asked to "explain or provide evidence" for their answer. The fourth question included a list of potential prioritization strategies, asking reviewers to rank them in importance.

5.1 Regarding fairness in OpenStack

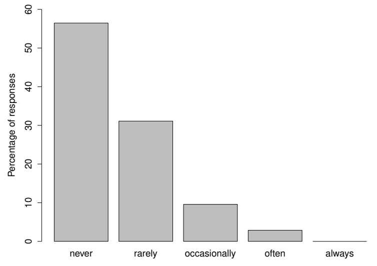
Figure 1 shows the summary of the responses to the first three questions. As an author of a patch, many participants considered they had being subjected to unfair treatments: 24% (51) consider this happens occasionally, and 15% (33) believe is occurs often or always. As patch reviewers, the 60% (124) claimed to never perform unfair reviews; however, 40% (83) considered they do unfair reviews rarely or occasionally. While the majority agreed that the review process of OpenStack is fair, 13% (28) respondents disagree or strongly disagree that the entire review process is fair.

These results imply that, while the process is in general perceived as fair, still a significant proportion of participants perceive unfairness (some claiming that it occurs frequently). This perception of unfairness is more common in authors than in reviewers.

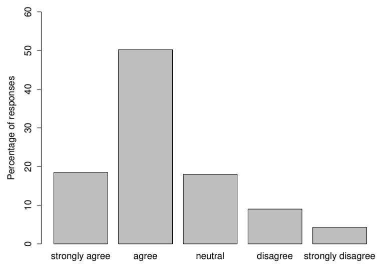
Two authors of this study separately open coded each open-ended answer. In particular, we concentrated only on responses where the respondent feels there is unfairness. Because this study is exploratory and our objective is to show that fairness is an issue—at least to some contributors—those that imply the system is fair were ignored (as shown in Figure 1c most respondents feel that the review process of OpenStack is fair). These responses were classified according to the fairness categories presented in Section 2. After classifying, they discussed the responses that were not consistently classified to reach an agreement. Table 3 summarizes the results.



(a) "Have your contributions been treated unfairly?" (212 responses)



(b) "According to your experience as a reviewer, do you perform code reviews unfairly?" (209 responses)



(c) "In general, is the code review process in OpenStack fair?" (213 responses)

Figure 1: Responses to Likert-scale questions in the web survey on the perception of fairness in OpenStack (a) as patch author, (b) as code reviewer, and (c) in general of the OpenStack code review process

Table 3: Result of the categorization of the open-ended responses of the perception of unfairness in OpenStack

Category	Rule	Frequency* (as author)	Frequency* (as reviewer)	Frequency* (General)
Distributive	Equity & Equality	15 (28%)	14 (34%)	19 (45%)
	Need	12 (22%)	4 (10%)	4 (10%)
Procedural	Control	0 (0%)	0 (0%)	3 (7%)
	Consistency	19 (35%)	6 (15%)	7 (17%)
	Bias suppression	16 (30%)	14 (34%)	19 (45%)
	Information Accuracy	5 (9%)	3 (7%)	4 (10%)
	Correctability	2 (4%)	0 (0%)	0 (0%)
	Ethicality	4 (8%)	0 (0%)	0 (0%)
Interactional	Respect	2 (4%)	1 (2%)	1 (2%)
	Psychological Safety	0 (0%)	0 (0%)	0 (0%)
Informational	Truthfulness	0 (0%)	0 (0%)	0 (0%)
	Adequacy	5 (10%)	0 (0%)	0 (0%)
Non-applicable	Out of context, too broad, etc.	9 (17%)	18 (44%)	13 (31%)

*Some responses fall into several categories and rules, hence the sum of the frequency is higher than the total number of the responses.

As it can be seen, Distribute Fairness and Procedural Fairness (mainly consistency and bias-suppression) were the most common issues mentioned. This is in part because equity affects consistency. In the following subsections, we proceed to show examples of comments from respondents by category.

Distributive Fairness

Equity & Equality: There is a dissonance in the expectations of different contributors. Some expect equity and see equality as unfair:

> *"Contributing members in good standing that regularly and unselfishly carry their share of the workload can and should expect to receive higher priority for reviews."*

Other respondents expect equality, and see equity as unfair:

> *"A submitter is presented with the illusion that all submissions are equal, when in reality the community prioritizes [pull requests] from known submitters."*

Newcomers: rather than receive special attention, newcomers appear to sometimes suffer a negative bias. A respondent describes a textbook example of the impact of unfairness on her motivation to be a contributor:

> *I have only tried to make one contribution. I was so discouraged by the response from the reviewer I have not tried again.*

While a reviewer acknowledges having a bias against newcomers:

> *"When short on time [I] tend to prefer patch styles I'm familiar with (and agree with) over patch styles unfamiliar. This leads to me preferring people I'm already familiar and that I know write good patches over newcomers."*

Need: Respondents stated that some projects do not have enough core reviewers, which delays the entire process.

> *"As the OpenStack projects are becoming bigger, the review process is slowed down by the small number of core reviewers."*

Some respondents also believe that some reviewers do not invest the appropriate effort to review their code (e.g., a reviewer judges a patch before completely understanding it).

Procedural Fairness

Consistency: Respondents stated that reviews are not performed consistently. One case is when some reviewers tend to complain about minor issues more than others:

"... many reviewers in OpenStack ... give a negative review not for functionality but rather for a typo or an extra space."

Some respondents, however, do not regard this type of behavior as being unfair. Respondents also stated that consistency varies from one project to another (names of projects have been replaced):

"I have contributed and reviewed in 3 projects: [Project A], [Project B], and [Project C]. I found that [A] and [B] to be quite reasonable and fair. As stated above, [C] is a disaster."

Bias suppression: Bias was frequently mentioned; however, bias was seen in many different aspects. One major bias is cross-project collaboration and code cleanup:

"... each project tends to reject code that could be implemented in the other projects in an effort to minimize their own code base. But if/when they do that, important features cant land because no one will take a fair share of the code."

"The review process is sufficiently onerous and unpredictable that contributors are actively dissuaded from code cleanup, as this is not directly bug or feature driven."

Control and correctability were rarely mentioned. Control was positively mentioned to avoid unfairness:

"The culture in [Project A] is to gain mind share for new features through discussion on M[ailing] L[ists] and summits."

Regarding correctability, one respondent mentioned the lack of an appeal process:

"[A] core reviewer gave me a -1 which was not understandable. I asked him 4 times on IRC to discuss [it] and he ignored it."

Ethicality: Some responses can arguably fall into this area. In particular, where reviewers explicitly benefit people that they know:

"People look out for friends instead of looking out for their projects."

Interactional Fairness

This type of fairness was only mentioned in the context of **Respect**, primarily by satellite developers and who feel disrespected by more central contributors. For example:

Requests for reviews from core members in the IRC channel have sometimes been responded to with contempt.

Another writes how she feels demeaned by reviewer's lack of attention:

Contributions are ignored unless I beg for attention. I might have time to contribute a minor improvement. I never have time to beg for attention. If you don't want my help, I got the message loud and clear.

Newcomers appear to be a target, also:

"To a point where I have actually heard Cores [maintainers] indicate that people need to be "hazed", or prove themselves or suffer really hard reviews to "earn their stripes". I cannot imagine how many developers we have chased away because of this fraternity culture."

Informational Fairness

Only **Adequacy** was mentioned in the answers, and it was mentioned only in the context of being an author, not the reviewer. Answers covered both dimensions: the quality (or lack) of feedback:

The reviewer does not explain what the problem is or does not include the context of the argument.

and the lack of explanations for unexpected treatments —in this case, a long delay:

Reviews that take too long are] very demoralizing for potential contributors if they don't hear feedback in a timely manor.

Summary

These responses clearly show that unfairness, especially with respect to equity, equality, consistency and bias suppression are important concerns to some contributors. This is probably because the documented procedures of OpenStack do not address these issues, leaving them to interpretation. In our analysis, we omitted positive responses (i.e., those who considered the review process as fair). Due to this, and the limited number of responses, we can not make any generalization regarding the fairness of the review process in OpenStack.

5.2 Prioritization

OpenStack does not document how reviewers should prioritize patches to review. That is the reason why we asked reviewers how they prioritize what contribution to review. The purpose of this question was to evaluate if reviews were prioritized consistently and without bias. We asked OpenStack reviewers how they prioritize patches to review and offered five possible answers: "a patch is selected based on (1) the developer's expertise, (2) the importance of the patch, (3) the author of the patch, (4) the difficulty (the easiest or the most difficult patch first), or (5) its freshness (the newest or the oldest patch first)", and gave them the option to elaborate on their answer in a free textual answer.

Figure 2 shows the results. At it can be seen, the expertise of the reviewer and the importance of the patch are **consistently** considered the most important prioritization strategies by the majority of the reviewers. *Importance of the patch* implies that, as expected, **equity** is applied to the treatment of patches to review. The most often mentioned reasons a patch is important are: security fix, and other patches depend upon it.

The rest of the options to the question are more divided. A significant number of people chose *easiest first* while others chose *the most difficult first*; similarly, some reviewers chose *oldest first*, and others *newer first*. This result shows a significant lack of **consistency** in the selection of patches to review.

The respondents were very divided on whether to prioritize reviews from people they know. Some of the answers imply that some reviewers might lean towards **equity** (giving priority to patches from frequent over satellite contributors):

"[I] tend to emphasize reviewing code submitted by [my] team, particularly when I don't have much time for review."

In some cases, people look for patches that benefit them or their organization, and this might lead towards **ethicality** issues. Two respondents wrote:

"I try to review the contributions of my co-workers at [company name] primarily."

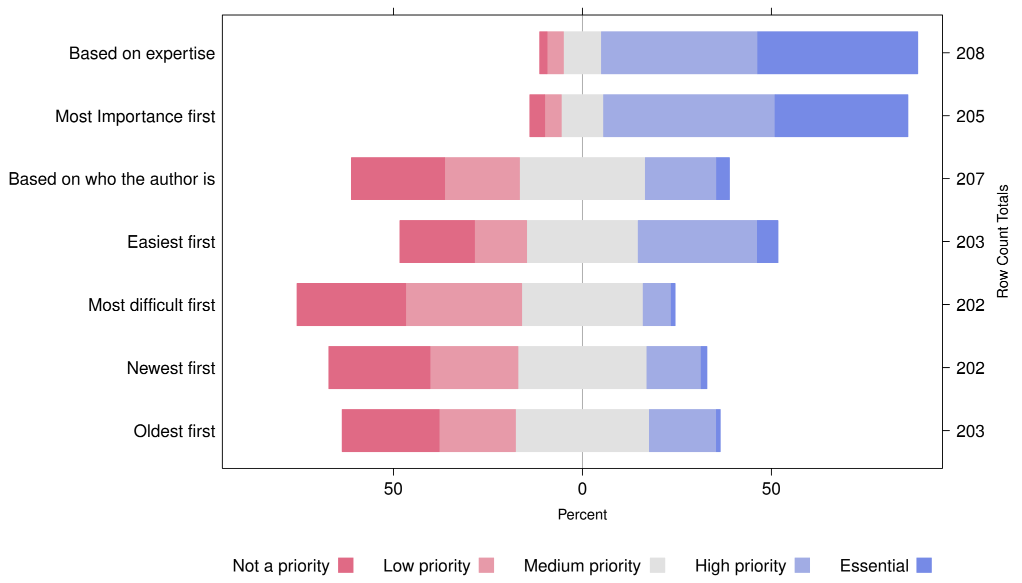


Figure 2: Breakdown of answers to the question: When you review code, how do you prioritize what contribution to review?

"[I review first] patches that align with my interests."

Regarding self interest, another reviewer highlighted that selfishness is not necessarily good for the project:

"Most important for me != most important for the project."

Some reviewers wrote that they tend to review patches after being contacted by the author (who they usually know). This puts **newcomers** at a disadvantage, since they are unlikely to have connections with reviewers.

Overall, we found that reviewers prioritize for a variety of reasons, some grounded on technical aspects and others in social ones. However, without a clear policy of what the reviewing prioritization is, authors of patches could find it difficult to know when the review of their patch should be completed. This was summarized by one respondent:

"[...] For example, a developer might think they are being treated unfairly because other reviewers were completed and merged before theirs."

6 DISCUSSION AND IMPLICATIONS

Modern code reviews have considerable advantages, both to the quality of the system and to the knowledge of its developers. However, social aspects of code reviews cannot be ignored [19]. Every time a person evaluates the work of another, social tensions tend to arise, especially when the outcome is unsuccessful (i.e., the patch is rejected). In code reviews, the individual's work is subjected to the decisions of others and these decisions might have economic and socioemotional consequences to the developers [16].

The literature on code reviews has concentrated on the mechanics of doing code reviews, barely addressing its social interactions. These interactions have been a growing concern for developers, as reflected in blog posts and online discussions [55, 59]. Frequent recommendations to address these issues are to be polite, to be objective, to avoid nit-picking, etc. However, that is not enough. Many other aspects of fairness need to be taken into consideration.

During social interactions, events are likely to produce emotions, especially the negative ones. These emotions affect directly the attitudes and behavior of people [75]. In organizational settings, the frequency and accumulation of the events, rather than their intensity, determine the outcomes [22]. That is, people in an organization can tolerate one or two major events, and yet, have difficulty to overcome relatively minor –but constant– events that affect their work. Thus for example, *"The white-space reviewers drive me crazy!"* is a reflection of an event that triggers a negative emotional response. Within our framework, the developer feels unfairly treated, which may be exacerbated if the code is not reviewed before fixing the white spaces. In the long term, the accumulation of these *small* details may determine the satisfaction in the project, and future participation. Fisher [22] argues that behaviors and attitudes are dynamic, they change with the hassles and uplifts of everyday work. The same occurs with the perception of fairness. Therefore, fairness in code review must be constantly monitored.

Fairness theory provides a solid foundation to create better policies and procedures for code reviewing. Fairness has many dimensions and ignoring any of them might lead to unfairness, even if the rest of the dimensions are properly addressed. The framework that we provide can serve as a starting point of discussion, and as a way to evaluate perceptions of unfairness (as we have demonstrated in our qualitative study).

6.1 Lessons for Practitioners

Guidelines for conduct code reviews usually address issues of interactional fairness (politeness and constructive language) and informational fairness (provide good and precise feedback) [69]. However, distributive and procedural fairness are considered much less.

A reviewing system should start by addressing **distributive fairness**, from which guidelines for procedural fairness are derived. This includes defining: (1) the minimal needs of a patch review, (2) the measurement of importance of patches, (3) the decision of whether to use equity or equality with respect to authors; and (4) how to treat newcomers.

Equity should be used in the treatment of patches. It is clear that some patches are more important than others (e.g., security bug fixes should have higher priority). Implementing equity of patches requires the ability to measure a patch's importance.

Even in the presence of equity, all code reviews should have a minimal level of quality (the needs of the review). Some of these needs might be a function of the patches difficulty (e.g., larger patches require longer reviews), while others should remain above a minimal threshold (e.g., the reviewer will apply the same level of concentration and care to any review). The needs of a patch might also depend on what the patch achieves (i.e., a patch to documentation might require less reviewers or votes than a patch that affects a critical area of the system).

Often the needs of a patch are in direct conflict with the reviewing resources (especially where there is a shortage of reviewers). One way to address this shortage is to allow some patches to have incomplete reviews (or no review at all); this will require that each review clearly documents "level of attention" that the patch has received. For example, certain types of patches could be approved with a mark saying "hastily reviewed" or "not reviewed". This way others are informed of the potential risks of the accepted patch. In such a scenario, the stated minimal need of a patch is lower, but the system provides a way of *signaling* the risks of this process.

It is less obvious if some authors should be treated differently than others. A code reviewing system should determine if equity or equality be implemented. If equity is chosen, it requires a way to measure contribution. It is not uncommon in reviewing systems for core contributors to gain an advantage against other patch authors; but it is important that this advantage is documented – those who are biased against should know why, which reduces their perception of being treated unfairly.

Procedural fairness requires the establishment of clear procedures that document how the review is to be performed. Consistency and bias-suppression go hand-in-hand. The reviews should be consistent and without bias, except when the bias is a result of equity (some patches/authors' patches should have priority). Lack of consistency was one of the major complaints in our survey. Regarding control, modern code review process should always give a voice to the author of the code. It is important that a patch is not rejected (and the review closed) without allowing its author to respond to the reviewer's evaluation. Similarly, a system should be in place that allows the author of a rejected patch to appeal the decision, ideally involving new reviewers who did not participate in the former review process. Due to the open nature of code reviews, information accuracy is often not an issue, especially when the author and other developers can contribute to the review.

Ethicality appeared in some of the responses to our survey. Authors are concerned that some reviewers approve patches from certain authors without actually reviewing them ("quid pro quo"). A code review process can specify some ethical expectations of the reviewer, such as the requirement that a patch will never be approved without a minimal level of review and that both the author and the reviewer will not deceive others, or collude with each other.

Interactional Fairness is being addressed with codes of conduct. In general, these documents address the issues of disrespect, and exposure to personal danger but should also address issues of invasion of privacy, where the reviewer improperly uses information regarding the author (outside the review scope).

Regarding **Informational Fairness**, the openness of code reviews (especially in OSS systems) minimize the issues of truthfulness. With respect to adequacy, there are two main aspects: a) give specific reasons for a rejection –typically documented in guidelines of how to properly do code reviews); and, b) it is important that, when reviews veer off from the expected procedures (such as taking too long) the author is properly notified why. The existence of metrics and analytics of the code review process that document the load of reviewers, the length of the review queue, the turn-around time of reviews, rejection rate, etc. appear to facilitate adequacy, as it creates expectations on the authors regarding the quality of service that they should expect (although we acknowledge that research is required to verify this).

The existence of procedures, guidelines and code of conduct might be sufficient. However, in some cases, it might be necessary to have mechanisms for monitoring compliance and enforcement.

6.2 Implications for Researchers

Evaluating the work of others in a way that is perceived as fair is not trivial, yet, little research has been done to study fairness within the scope of software engineering. Recent work has found that social issues in code reviews (such as organizational membership [5], or anger in communication [47]) affect the outcomes, but work is needed to evaluate if these behaviors are perceived as fair or not.

Fairness needs to be considered an important part of the health of a system. The potential negative effects of unfairness might require its early identification and removal. This requires research in methods to monitor how *fair* the system *is* with respect to the expectations of the organization and its members, and perhaps even more important, how fair the system is *perceived* by its members. Metrics serve an important role for this purpose. However, metrics must be consistent with distributive fairness, and they should consider the merit (equity) of the patches and their (minimal) need (e.g., some patches should be reviewed faster than others, but no patch should take longer than a certain time). Being able to do this requires research in assessing the merits of a patch (e.g., when does a patch require to be prioritized?). Similarly, it might be desirable to automatically monitor the fairness of each reviewer.

Empirical studies can help clarify the more important attributes of fairness. Similarly, studies are needed to evaluate the cost-benefit ratio of implementing fairness policies and mechanisms. This will inform organizations in terms of what actions are more likely to yield results depending on their particular needs and resources.

Our empirical study was exploratory of one OSS system. More research is needed to better understand how developers feel about fairness in code reviews, both in open source and industry and what are the effects of unfairness in both the system being developed, and the individuals who perceive unfairness.

7 THREATS TO VALIDITY

Internal validity: relates to the validity of causal inferences (researcher bias) made by the study. The framework is derived from our interpretation of how the concepts in fairness theory could apply to code reviews, and other researchers may arrive at a slightly different framework to apply. The manual coding of the open-ended responses in the survey may have introduced subjective bias in the results. To reduce this bias, two authors coded independently the responses. The results were merged afterwards. We calculated Cohen's Kappa to evaluate the level of inter-rater agreement between the two authors, which was slightly above 0.7 [14]. In case of disagreement, both authors discussed it with the help of a third author. Particularly, the comments referring to *equity* and *equality* (from distributive fairness) and *bias suppression* were difficult to split into different categories, so we grouped them all into one.

Construct validity: refers to whether the studied parameters are relevant to the research questions, and actually measure the social constructs or concepts intended to be studied. In our study, we may have missed some developer's information, as we can only include the people who have recognizable and available information on the Internet (e.g., email address). We lack information for less than 7% of developers. Considering that OpenStack is an industrial leading OSS project, we may safely ignore those developers. Participation in the web survey was low. However, as stated by Lethbridge *et al.* "if the objective is to understand trends, with reasonable confidence, then low response rates may well be fine" [41].

External validity: is concerned with the extent to which it is possible to generalize the findings. We do not claim that our results apply to other software projects. Our study is exploratory, whose objective is to support the framework to study social behavior in modern code reviews. We presented preliminary evidence from the results that further supports the framework being applicable to modern code reviews, and demonstrates that the issues of fairness exist and should be taken into consideration. Although providing rich data, a single case study has limitations, and further research may help refine the framework presented. Nevertheless, a frequent misconception is that single case studies provide little value for the academic community, and do not contribute to scientific development. Historical evidence shows otherwise. It is known from the social sciences that "more discoveries have arisen from intense observation than from statistics applied to large groups" ([39]–page 95, as cited by Begel and Zimmermann [7]). Similarly, individual cases have contributed to discovery in physics, economics, and social sciences [23]. Both single case studies and research on large samples are essential for the development of an empirical body of knowledge [3].

8 RELATED WORK

With respect to code review studies, Bacchelli and Bird studied the motivations, expectations and outcomes of modern code review [1]. Rigby *et al.* studied the review policies and examined which metrics have the largest impact on review efficacy in OSS projects [50, 51]. von Krogh *et al.* note that some OSS projects have a "socialization" process when accepting technical contributions [73]. Balachandran suggested using review-bot to reduce human effort and improve code review quality [2]. Bosu *et al.* investigated the factor of useful reviews to improve the effectiveness of code reviews [11]. Thongtanunam *et al.* studied traditional code ownership heuristics using code review activities [67]. Baysal *et al.* found the non-technical factors of code review can significantly influence the code review outcomes [5, 6]. McIntosh *et al.* found that there is a negative influence on software quality when the poorly-reviewed code is merged [45]. Jiang *et al.* found that experience of developers impacts the patch acceptance and the reviewing time [36]. Yang *et al.* studied the social relationships between patch authors and reviewers [78]. Dabbish *et al.* [20] found that information transparency motivates participation. Tsay *et al.* [71] found that the level of previous participation of a developer improves the politeness of the review.

Sentiment analysis has been used to study security related pull requests in GitHub where it was found that these discussions tend to have more negative emotions [47]. Gachechiladze *et al.* [27] argued that detecting anger can be helpful to maintain the health of a software project. The use of codes of conduct in OSS projects has been of interest to researchers. Tourani *et al.* [70] studied how codes of conduct are used in a large collection of OSS projects. Schneider *et al.* [55] studied the interactions in the Linux mailing list, where it is highlighted the dispute between kernel developers that lead to Sharp leaving the kernel development. Squire *et al.* [59] provide a collection of data about insults and improper language in OSS projects discussions, and how they relate to their Code of Conducts.

9 CONCLUSIONS

Code reviews are a sociotechnical process where the author is subjected to the decision of others. The exploratory survey of contributors and code reviewers of OpenStack (presented in Section 5) demonstrates that unfairness is starting to be perceived as an issue. Lack of fairness can be demoralizing and affect the productivity and/or participation of some of its members, and can potentially affect the long term success of a project. Codes of conduct are becoming common in software development (especially OSS), but they typically address only interactional fairness. We created a framework that helps study, understand, and manage the potential challenges that code reviews face regarding fairness. We have also contributed a set of guidelines that practitioners can use to improve fairness in their code reviews.

Fairness is likely to be an issue in other decision processes in software engineering. Having a fair development process might be particularly important for OSS projects, where members choose to participate, specially for large industrial-baked OSS projects (where the entities that choose to participate also make large investments).

Fairness is a nascent area of research in software engineering; among many topics, research is needed to i) understand how software developers feel about fairness during their day-to-day activities; ii) to measure the perceivable fairness of a software development project and how it affects its productivity; and iii) how to improve it.

ACKNOWLEDGMENTS

This work was supported by Canada's Natural Science and Engineering Research Council, Japan's JSPS KAKENHI Grant Number JP25220003, and the Spanish Government through project TIN2014-59400-R.

REFERENCES

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proc. of the 35th Intl. Conf. on Software Engineering (ICSE '13)*. IEEE, 712–721.

[2] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proc. of the 2013 Intl. Conf. on Software Engineering*. IEEE, 931–940.

[3] Victor R Basili, Forrest Shull, and Filippo Lanubile. 1999. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering* 25, 4 (1999), 456–473.

[4] Gabriele Bavota and Barbara Russo. 2015. Four eyes are better than two: On the impact of code reviews on software quality. In *Software Maintenance and Evolution (ICSME), 2015 IEEE Intl. Conf. on*. IEEE, 81–90.

[5] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey. 2013. The influence of non-technical factors on code review. In *Proc. of the 20th Intl. Working Conf. on Reverse Engineering (WCRE '13)*. 122–131.

[6] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2016. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering* 21, 3 (2016), 932–959.

[7] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. In *Proc. of the 36th Intl. Conf. on Software Engineering*. ACM, 12–23.

[8] Robert J. Bies and Joseph S. Moag. 1986. Interactional justice: Communication criteria of fairness. In *Research on Negotiation in Organizations*, R.J. Lewicki, B.H. Sheppard, and M.H. Bazerman (Eds.). JAI Press, 43–55.

[9] Robert J. Bies and Debra L. Shapiro. 1987. Interactional fairness judgments: The influence of causal accounts. *Social Justice Research* 1, 2 (01 jun 1987), 199–218.

[10] Amiangshu Bosu and Jeffrey C Carver. 2014. Impact of developer reputation on code review outcomes in OSS projects: An empirical investigation. In *Proc. of the 8th ACM/IEEE Intl. Symp. on Empirical Software Engineering and Measurement*. ACM, 33.

[11] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *Proc. of the 12th Intl. Working Conf. on Mining Software Repositories (MSR '15)*. 146–156.

[12] Jon Brodkin. 2013. Linus Torvalds defends his right to shame Linux kernel developers. Ars Technica. (July 2013).

[13] Mauro Carvalho Chehab. 2016. Code of Conflict. Online. (2016). https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/code-of-conflict.rst?h=v4.13-rc6 Visited 2017-08-23.

[14] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin* 70, 4 (1968), 213.

[15] Ronald L Cohen. 1987. Distributive justice: Theory and research. *Social Justice Research* 1, 1 (1987), 19–40.

[16] Jason A Colquitt. 2001. On the dimensionality of organizational justice: a construct validation of a measure. *Journal of applied psychology* 86, 3 (2001), 386.

[17] Jason A. Colquitt and Jerome M. Chertkoff. 2002. Explaining Injustice: The Interactive Effect of Explanation and Outcome on Fairness Perceptions and Task Motivation. *Journal of Management* 28, 5 (2002), 591–610.

[18] Jason A. Colquitt, Donald E. Conlon, Michael J. Wesson, Christopher O. L. H. Porter, and K. Yee Ng. 2001. Justice at the Millenium: A Meta-Analytic Review of 25 Years of Organizational Justice Research. *Journal of Applied Psychology* 86, 3 (2001), 425–445.

[19] Jacek Czerwonka, Michaela Greiler, and Jack Tilford. 2015. Code Reviews Do Not Find Bugs: How the Current Code Review Best Practice Slows Us Down. In *Proc. of the 37th Intl. Conf. on Software Engineering (ICSE '15)*. IEEE, Piscataway, NJ, USA, 27–28.

[20] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. of the ACM Conf. on Computer Supported Cooperative Work (CSCW '12)*. ACM, 1277–1286.

[21] Amy Edmondson. 1999. Psychological safety and learning behavior in work teams. *Administrative science quarterly* 44, 2 (1999), 350–383.

[22] Cynthia D. Fisher. 2000. Mood and Emotions while Working: Missing Pieces of Job Satisfaction? *Research in Organizational Behavior* 21, 2 (2000), 185–202.

[23] Bent Flyvbjerg. 2006. Five misunderstandings about case-study research. *Qualitative inquiry* 12, 2 (2006), 219–245.

[24] Robert Folger and Russell Cropanzano. 1998. *Toward a General Theory of Fairness*. SAGE, 173–196.

[25] Robert Folger and Russell Cropanzano. 2001. *Fairness Theory: Justice as Accoutability*. Stanford University Press, 1–55.

[26] Robert Folger, David Rosenfield, Janet Grove, and Louise Corkran. 1979. Effects of 'voice' and peer opinions on responses to inequity. *Journal of Personality and Social Psychology* 37, 12 (1979), 2253–2261.

[27] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and its direction in collaborative software development. In *Proc. of the 39th Intl. Conf. on Software Engineering: New Ideas and Emerging Results Track*. IEEE, 11–14.

[28] Jerald Greenberg. 1987. A taxonomy of organizational justice theories. *Academy of Management review* 12, 1 (1987), 9–22.

[29] Steven L. Grover. 2014. Unraveling respect in organization studies. *Human Relations* 67, 1 (2014), 27–51.

[30] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, A. E. Camargo Cruz, Kenji Fujiwara, and Hajimu Iida. 2013. Who does what during a code review? Datasets of OSS peer review repositories. In *Proc. of the 10th Intl. Working Conf. on Mining Software Repositories (MSR '13)*. IEEE, 49–52.

[31] Il-Horn Hann, Jeffrey A Roberts, and Sandra A Slaughter. 2013. All are not equal: An examination of the economic returns to different forms of participation in open source software communities. *Information Systems Research* 24, 3 (2013), 520–538.

[32] Bryan W. Husted and Robert Folger. 2004. Fairness and Transaction Costs: The Contribution of Organizational Justice Theory to an Integrative Model of Economic Organization. *Organization Scienc* 15, 6 (2004), 719–729.

[33] Daniel Izquierdo-Cortazar, Lars Kurth, Jesus M Gonzalez-Barahona, Santiago Dueñas, and Nelson Sekitoleko. 2016. Characterization of the Xen project code review process: an experience report. In *Proc. of the 13th Intl. Conf. on Mining Software Repositories*. ACM, 386–390.

[34] Daniel Izquierdo-Cortazar, Nelson Sekitoleko, Jesus M Gonzalez-Barahona, and Lars Kurth. 2017. Using Metrics to Track Code Review Performance. In *Proc. of the 21st Intl. Conf. on Evaluation and Assessment in Software Engineering*. ACM, 214–223.

[35] Joab Jackson. 2017. Node.js Forked Again Over Complaints of Unresponsive Leadership. The News Stack https://thenewstack.io/node-js-forked-complaints-repeated-harassment/. (Aug 2017).

[36] Yujuan Jiang, Bram Adams, and Daniel M. German. 2013. Will My Patch Make It? And How Fast? Case Study on the Linux Kernel. In *Proc. of the 10th Intl. Working Conf. on Mining Software Repositories (MSR '13)*. 101–110.

[37] Sean Michael Kerner. 2016. OpenStack Revenues Approaching $3.4B: 451 Research. Online. (2016). http://www.eweek.com/cloud/openstack-revenues-approaching-3.4b-451-research Visited 2017-08-23.

[38] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *Software Maintenance and Evolution (ICSME), 2015 IEEE Intl. Conf. on*. IEEE, 111–120.

[39] Adam Kuper and Jessica Kuper. 1985. *The Social Science Encyclopedia*. Routledge.

[40] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *Proc. of the 39th Intl. Conf. on Software Engineering*. IEEE, 187–197.

[41] Timothy C Lethbridge, Susan Elliott Sim, and Janice Singer. 2005. Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering* 10, 3 (2005), 311–341.

[42] Gerald S Leventhal. 1976. The distribution of rewards and resources in groups and organizations. *Advances in experimental social psychology* 9 (1976), 91–131.

[43] Gerald S Leventhal. 1980. *What should be done with equity theory?* Springer.

[44] Susan A. Lynham. 2002. The General Method of Theory-Building Research in Applied Disciplines. *Advances in Developing Human Resources* 4, 3 (2002), 221–241.

[45] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2015. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering (EMSE)* 21, 5 (2015), 1–44.

[46] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[47] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and Emotion: Sentiment Analysis of Security Discussions on GitHub. In *Proc. of the 11th Working Conf. on Mining Software Repositories (MSR 2014)*. 348–351.

[48] Teade Punter, Marcus Ciolkowski, Bernd Freimut, and Isabel John. 2003. Conducting on-line surveys in software engineering. In *Empirical Software Engineering, 2003. ISESE 2003. Proc.. 2003 Intl. Symp. on*. IEEE, 80–88.

[49] Uzma Raja and Marietta J Tretter. 2012. Defining and evaluating a measure of Open Source project survivability. *IEEE Transactions on Software Engineering* 38, 1 (2012), 163–174.

[50] Peter C. Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proc. of the 9th Joint Meeting on Foundations of Software Engineering (FSE '13)*. ACM, 202–212.

[51] Peter C. Rigby, Daniel M. German, Laura Cowen, and Margaret-Anne Storey. 2014. Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. *Transactions on Software Engineering Methodologies* 23, 4, Article 35 (Sept. 2014), 33 pages.

[52] Peter C. Rigby, Daniel M German, and Margaret-Anne Storey. 2008. Open source software peer review practices: a case study of the apache server. In *Proc. of the 30th Intl. Conf. on Software engineering*. ACM, 541–550.

[53] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.

[54] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Blackwell, Hoboken, New Jersey, USA. 256 pages.

[55] Daniel Schneider, Scott Spurlock, and Megan Squire. 2016. Differentiating Communication Styles of Leaders on the Linux Kernel Mailing List. In *Proc. of the 12th Intl. Symp. on Open Collaboration (OpenSym '16)*. ACM, New York, NY, USA, Article 2, 10 pages.

[56] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.

[57] Sarah Sharp. 2015. Closing a door. Online. (2015). http://sarah.thesharps.us/2015/10/05/closing-a-door/ Visited 2017-08-23.

[58] Janice Singer, Susan E. Sim, and Timothy C. Lethbridge. 2008. Software Engineering Data Collection for Field Studies. In *Guide to Advanced Empirical Software Engineering*. Springer, London, UK, 9–34.

[59] Megan Squire and Rebecca Gazda. 2015. FLOSS as a Source for Profanity and Insults: Collecting the Data. In *Proc. of the 48th Hawaii Intl. Conf. on System Sciences*, Vol. HICSS '15. IEEE, 5290–5298.

[60] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.

[61] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. How do software engineers understand code changes?: an exploratory study in industry. In *Proc. of the ACM SIGSOFT 20th Intl. Symp. on the Foundations of Software Engineering*. ACM, 51.

[62] The OpenStack Foundation. 2012. Companies Supporting The OpenStack Foundation. Online. (2012). https://www.openstack.org/foundation/companies/ Visited 2017-08-23.

[63] The OpenStack Foundation. 2012. The OpenStack Foundation. Online. (2012). https://www.openstack.org/foundation/ Visited 2017-08-23.

[64] The OpenStack Foundation. 2017. OpenStack community contribution in all releases | Lines of code. Online. (2017). http://stackalytics.com/?release=all&metric=loc Visited 2017-08-23.

[65] The OpenStack Foundation. 2017. OpenStack com=munity contribution in all releases | Reviews. Online. (2017). http://stackalytics.com/?release=all&metric=marks Visited 2017-08-23.

[66] John W Thibaut and Laurens Walker. 1975. *Procedural justice: A psychological analysis*. L. Erlbaum Associates.

[67] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2016. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proc. of the 38th Intl. Conf. on Software Engineering (ICSE '16)*. IEEE, 1039–1050.

[68] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2017. Review participation in modern code review. *Empirical Software Engineering* 22, 2 (2017), 768–817.

[69] Parastou Tourani and Bram Adams. 2016. The Impact of Human Discussions on Just-In-Time Quality Assurance. In *Proc. of the 23rd Intl. Conf. on Software Analysis, Evolution, and Reengineering (SANER '16)*. 189–200.

[70] Parastou Tourani, Bram Adams, and Alexander Serebrenik. 2017. Code of Conduct in Open Source Projects. In *Proc. of the 24th Intl. Conf. on Software Analysis, Evolution, and Reengineering (SANER '17)*. IEEE, 24–33.

[71] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proc. of the 22nd Intl. Symp. on Foundations of Software Engineering (FSE '14)*. ACM, 144–154.

[72] Tom R. Tyler. 1994. Psychological models of the justice motive: Antecedents of distributive and procedural justice. *Journal of Personality and Social Psychology* 67, 5 (1994), 850 – 863.

[73] Georg Von Krogh, Sebastian Spaeth, and Karim R Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy* 32, 7 (2003), 1217–1241.

[74] Jing Wang, Patrick C Shih, Yu Wu, and John M Carroll. 2015. Comparative case studies of open source software peer review practices. *Information and Software Technology* 67 (2015), 1–12.

[75] Howard M. Weiss and Russell Cropanzano. 1996. *Affective Events Theory: A theoretical discussion of the structure, causes and consequences of affective experiences at work*. Vol. 18. Elsevier, 1–74.

[76] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer.

[77] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. 2015. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In *Software Maintenance and Evolution (ICSME), 2015 IEEE Intl. Conf. on*. 261–270.

[78] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Peer Review Social Network (PeRSoN) in Open Source Projects. *IEICE Transactions on Information and Systems* E99-D, 3 (2016), 661–670.

[79] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2016. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering* 42, 6 (2016), 530–543.