How Do FOSS Communities Decide to Accept Pull Requests?

Adam Alami
IT University of Copenhagen
Copenhagen, Denmark

Marisa Leavitt Cohn
IT University of Copenhagen
Copenhagen, Denmark

Andrzej Wąsowski
IT University of Copenhagen
Copenhagen, Denmark

ABSTRACT

Pull requests are a method to facilitate review and management of contribution in distributed software development. Software developers author commits, and present them in a pull request to be inspected by maintainers and reviewers. The success and sustainability of communities depends on ongoing contributions, but rejections decrease motivation of contributors. We carried out a a qualitative study to understand the mechanisms of evaluating PRs in open source software (FOSS) communities from developers and maintainers perspective. We interviewed 30 participants from five different FOSS communities. The data shows that acceptance of contributions depends not only on technical criteria, but also significantly on social and strategic aspects. This paper identifies three PR governance styles found in the studied communities: (1) protective, (2) equitable and (3) lenient. Each one of these styles has its particularities. While the protective style values trustworthiness and reliability of the contributor, the lenient style believes in creating a positive and welcoming environment where contributors are mentored to evolve contributions until they meet the community standards. Despite the differences, these governance styles have a commonality, they all safeguard the quality of the software.

CCS CONCEPTS

• **Software and its engineering** → **Open source model**; • **Human-centered computing** → Collaborative and social computing.

KEYWORDS

Open source software, code review, pull request, decision making, FOSS governance, community management

ACM Reference Format:
Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2020. How Do FOSS Communities Decide to Accept Pull Requests?. In *Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383219.3383242

1 INTRODUCTION

FOSS projects are collaborative ventures organized as communities that produce software using specific coding processes and tools such as GitHub. Contributors submit code changes, such as a bug fix or a new feature, in form of a pull request which undergoes an evaluation for appropriateness and quality. The evaluation process is not simple; it involves specific technical and social rituals. Various evaluation patterns (governance styles) emerge in communities. Only 13% of pull requests are rejected due to technical reasons [15] and "the toughest and most frequent challenges encountered by contributors are social in nature." [16]. We attempted to determine the factors affecting pull requests being accepted or rejected by asking the following questions:

RQ1: *How do FOSS communities decide to accept pull requests?*

RQ2: *What are the principles of evaluating pull requests?*

We define PR governance as the system of rules, practices, and norms by which a community directs and control assessment of PRs. It ensures management of the interests of the community and the integrity of its products. Because of the importance of PRs and the effect the governance of FOSS communities has on PRs, this study was designed to investigate PR evaluation governance styles in FOSS communities, using a qualitative approach. We determine the decision-making mechanisms in evaluating PRs based on extensive data transcribed from interviewing 30 FOSS contributors and maintainers from five communities.

The PR evaluation process is a socially loaded practice. Interviewee 25 explains: *"... how contributions are rejected is a major factor in a project's success. The structure of PRs acceptance process is such that it can easily be used to bully people, assert dominance, engage in various forms of emotionally abusive behavior"*. If project's success depends greatly on how PRs are evaluated; then, it is important to understand how communities evaluate PRs.

In this work we identify existing PR governance styles and their underlying believes and norms. We also extract lessons for community leaders and maintainers. For example, our interviewees prefer a governance style that values technical merits over social connections. We highlight the main contributions below.

(1) In response to **RQ1** we propose to distinguish the following three governance models:
   (a) *Protective*: A defensive style of governance where the project leader and his subordinates have absolute power over what is merged into the code. This is known in the FOSS circles as "no by default." This attitude requires prior commitment and trust from the contributor to win the approval of the gatekeeper before the evaluation can take place.
   (b) *Equitable*: A style of governance based on fairness and the ascendancy of evaluation principles. It focuses on a balanced and technically grounded decision. The community principles overrule any leniency toward contributors.
   (c) *Lenient*: A style of governance based on creating a positive and welcoming environment for contributors. This style tolerates some errors and mediocrity. The foundational belief here is that a contribution is an asset that should

not be taken lightly. A contributor carries an enthusiasm that should be leveraged for the benefit of the community. In order not to compromise on quality, the contributor is guided to evolve his or her PR to mergeable quality through mentoring. Our data shows, despite their fundamental differences in governing code changes, these style of governance have one thing in common, safeguarding code quality and ensuring the evolution of the software as per the community roadmap.

(2) In response to **RQ2**, we identified criteria that fit into three categories: (1) Software engineering practices and requirements, (2) social norms, and (3) strategic vision for the product. First, the software engineering criteria include specific rules and recommendations, which contributors need to adhere to in order for their PRs to be of "mergeable" quality. Atomicity is an example. The studied communities require that a PR addresses a single atomic concern. Second, social community norms, like trustworthiness, guide behavior of developers. A contribution from a trustworthy community member, who demonstrated prior commitment, might be prioritized. Third, the product vision must be met, or at least not contradicted, by the intent of the contribution.

FOSS communities are unique social systems and their PRs governance models will reflect this uniqueness. Understanding the values, norms and rituals that are taken into consideration in the PRs evaluation governance, enhances our insight into the evolution, growth, and lifecycle of open source software communities.

To best of our knowledge, all key contributions of this paper (the governance models and the analysis of social and technical interactions in pull request evaluations) are entirely new. We revealed how communities collectively judge and decide on the fate of a code change submission; while previous work focused on chances of acceptance and the factors affecting acceptance and rejections [16, 17, 23, 33, 34, 40].

The paper proceeds by introducing the studied communities in Sect. 2. In Sect. 3, we define our research method and discuss the rationale behind it. Section 4 presents the key findings, and Sect. 5 interprets them. Related work is discussed in Sect. 6. We conclude in Sect. 7.

2 SUBJECT COMMUNITIES

We now present the studied communities. We intentionally sought diversity in our selection. Our subjects build different products, have different participation demographics and history. Each community was selected for a particular reason. FOSSASIA was selected for having a predominately southeast Asian contributors and having an agenda beyond software development. Odoo and DuckDuckGo were selected because both of them are a blend of a community and a commercial enterprise. The Coala community is relatively young (5 years old at the time of the study), but has shown strong signs of growth. While the above communities are rarely studied, many have investigated the Linux Kernel community. Previous studies of the community [2, 20] have shown that the community has its own particularities. Being different culturally and socially from other communities, inspired us to include the Linux Kernel in our selection of communities. Table 1 summarizes the selected communities and the dimensions used for the selection.

FOSSASIA. FOSSASIA was founded in 2009, as a community committed to creating "social change" by using a wide range of technologies. The community projects range from free and open source software, to design, graphics and hardware. Amongst its successful projects, SUSI.AI and EventYaY. SUSI.AI is an artificial intelligent application that provides functionality for personal assistance, Help Desks and Chatbots. EventYaY offers features for organizers to create and manage events.

FOSSASIA welcomes developers to contribute using Github. The PRs management process uses the default Github features of managing PRs. Changes, new features and bug fixes are raised as "issues" and discussed by the community. Contributors voluntarily select and work on issues. Once the code and its companion artifacts (i.e. Test, documentation, etc.) are available, the contributor raise a PR to resolve the issue. The PR is assessed using a set of technical and non-technical criteria.

Odoo. Founded in 2005, Odoo is both a company and a community. The community develops software to manage and record sales, inventory, procurement, and accounting functions: a business intelligence engine with an all-in-one business suite program. The company markets an "enterprise" version of the community platform. The Community version is the open source version. The company version supplements the community version with additional features and services. The community attract developers worldwide.

DuckDuckGo. Similarly to Odoo, DuckDuckGo is a company and a community. Developing an independent search engine and web browser. The company opened source the software to attract a community around it; but their indexing algorithm remains closed source. DuckDuckGo was founded in February 2008. By 2013, there were over 3 million users on DuckDuckGo. In 2014, DuckDuckGo was included in Safari, and it was built into Mozilla.

Coala. Coala is a tool for developers made by developers, a language-independent Linter and analysis toolkit. The primary goal of coala is to make it easier for developers to create rules for a project's code. Coala emphasizes reusing code and the ability to use plugins. Coala provides a unified interface for linting and fixing code with a single configuration file. The software was first released in July 2015. Six additional releases were made in 2016, and two additional versions were released in 2017. The community positions itself as a beginner friendly environment. Coala outlines exactly how to get started as a member of the community, and the first step to meet the community.

Linux Kernel. The Linux Kernel is a free and open source operating system. Now, Linux is the most popular and versatile operating system kernel. It is used on super computers and web-servers, powering up cloud infrastructure, and controlling lots of mobile and embedded devices including all Android devices.

Communities	Dimensions			
	Product(s)	Size (estimated # of contributors in GitHub)	Age (years)	Leadership
FOSSASIA	The Open Event Organizer, SUSI.AI, PSLab Android App, and NeuroLab Android	1500	10	Meritocratic system
Odoo	Enterprise resource planning (ERP)	1000	15	Meritocratic system
DuckDuckGo	Search engine	186	11	Meritocratic system
Linux Kernel	Operating system kernel	15,600 [13]	25	"Benevolent dictator" [2]
Coala	Language independent linter	437 contributors	5	Meritocratic system

Table 1: Selected Communities and the Dimensions used for Selection

3 METHODS

We want to understand the process that takes place in the assessment of PRs in FOSS communities; the what and how of a PR evaluation and decisions are taken to merge or reject PRs. We wanted to explore the human and social aspects of the PR evaluation process. For this reason, we choose a qualitative research method, suitable for exposing and gaining participants' experiences and perspectives, giving rich data. This allows us to gain breadth and depth of understanding. We conducted interviews with contributors and maintainers from five communities, (see section 2).

Interviews. We opted to use semi-structured interviews as they allow the researcher to add questions arising during the interview. A semi-structured interview is suitable for assembling rich data for a qualitative study, as qualitative studies explore topics with a goal of gaining insights into individual beliefs and behaviors.

The questions (Tbl. 2) in our interview fall into three categories: introductory, core, and probing questions. The introductory questions were used to set the tone for the interview and make the interviewee comfortable. The core questions are directly related to the research questions. The probing questions are aimed at exposing details and concrete facts.

intro	Can you talk to me about your community?
	What first motivated you to participate in this community?
core	Can you describe the PR evaluation process in your community?
	Can you talk to us about the experience of having a PR rejected?
	Can you talk to us about the experience of having a PR accepted?
	When you evaluate a PR, how do you go about it?
	What is your community attitude and philosophy regarding evaluating PRs?
probing	What were the reasons for rejecting your PR?
	How did you feel about the rejection?
	What were the reasons for accepting your PR?
	How did you feel about the acceptance?
	What is the maintainer role in the process?

Table 2: Key parts of the interview framework

Subject Selection. We selected five FOSS communities that we felt would give us a deep understanding of the phenomena under study. Section 2 summarizes the choice and the selection process of the communities. We interviewed 30 participants from FOSSASIA, Odoo, DuckDuckGo, Linux and Coala communities. We searched for participants on GitHub repositories with the exception of the Linux and FOSSASIA communities. We randomly (indiscriminately, without a method, or conscious decision) searched for contributors and maintainers with valid emails in their GitHub profiles. Then, we sent them an invite to participate in the study. For the Linux and FOSSASIA communities, we used our contacts in the community to recruit participants. A snowball sampling effect took place in the recruitment of participants. We asked our contacts to introduce us to contributors and maintainers for the purpose of this study. Table 3 summarizes the demographics of the population of the interviewees. The experience is the number of years the interviewee spent in contributing to open source. Maintainers have final responsibility to merge the code and ensure an adequate review has occurred before the merge. They also direct the contributors and reviewers,

Interviewee	Community	Role	Experience [Y]	Country
1	FOSSASIA	Maintainer	4	India
2	FOSSASIA	Maintainer	5	India
3	FOSSASIA	Maintainer	4	India
4	FOSSASIA	Contributor	3	India
5	FOSSASIA	Maintainer	4	India
6	Odoo	Contributor	10	India
7	Odoo	Contributor	10	Greece
8	Odoo	Contributor	12	Belgium
9	Odoo	Contributor	3	Italy
10	Odoo	Contributor	5	India
11	Odoo	Contributor	8	USA
12	Odoo	Contributor	15	Belgium
13	DuckDuckGo	Contributor	6	USA
14	DuckDuckGo	Contributor	8	UK
15	DuckDuckGo	Contributor	5	North Macedonia
16	DuckDuckGo	Contributor	11	India
17	DuckDuckGo	Maintainer	12	USA
18	DuckDuckGo	Contributor	9	Finland
19	DuckDuckGo	Contributor	3	India
20	Linux Kernel	Contributor	12	Finland
21	Linux Kernel	Contributor	10	USA
22	Linux Kernel	Contributor	5	Ukraine
23	Linux Kernel	Contributor	6	India
24	Linux Kernel	Maintainer	8	North Macedonia
25	Linux Kernel	Contributor	30	USA
26	Coala	Contributor	5	India
27	Coala	Contributor	4	South Korea
28	Coala	Contributor	6	India
29	Coala	Maintainer	4	India
30	Coala	Maintainer	6	India

Table 3: The population of the interviewees

making sure that they connect to each other appropriately, often serving as dispatcher. Contributors are developers and sometimes volunteer to review other developers' code.

Data Collection. As the subjects were distributed geographically, all interviews were conducted using Google Meet (a video conferencing tool). The interviews lasted from 40 minutes to an hour, and they generated, in average, 12 pages of verbatim. All interviews were transcribed from recorded interviews.

Analysis. We used thematic coding [6, 14] to analyze the data, following the guidelines of Robson and McCartan [30] and of Miles and coauthors [24]. The iterative analysis begun in the early stages of the data collection and continued throughout the study. The responses were coded by examining the data line-by-line through the lens of the following questions: what is this saying? What does it represent? What is happening here? What are they trying to convey? What is the process being described? Once the responses were coded, we could find patterns in statements and ideas that were then suggestive of a theme (i.e. a concept or implied topic that organizes a group of repeating ideas that help to understand the responses related to the research question). After identifying and giving names to the basic meaning units, we grouped them in categories by similarity. Table 4 shows examples of our codes and their categories.

We stopped conducting interviews, when we attained saturation so when (1) all the data are accounted for, with no outlying codes or categories; (2) every category is sufficiently explained in depth by the data that support it; and (3) there is enough data to ensure the research questions can be answered.

4 FINDINGS

4.1 RQ1: Decision Making in PR Evaluation

We identify three styles of governance for pull requests in our data: (1) protective, (2) equitable, and (3) lenient. Each of these styles has certain characteristics and qualities. Table 5 represents the studied communities governance styles.

4.1.1 Protective. This style is defensive; it values trust, relationships and reliability of the contributor. The Linux Kernel community describe this style of pull request evaluation as "no, by default." In this community, the contributions are often either not thoroughly evaluated or rejected without due diligence. Interviewee 24 stated, *"I communicate with the maintainer a lot. In general, he says no, unless he cannot say no. You know that is kind of his philosophy. I saw this view elsewhere in the Linux community"*. Winning the approval of the gatekeeper is critical. It requires persistence and accumulated trust (reputation). Interviewee 20 said, *"It's easy for me to get patches in because people in this community trust me and know who I am. Basic patches just go in easily because the maintainer trusts me. He knows that I will be around. If I submit a big chunk of code, and he does not know me, I may just disappear. Maintainers are very conscious about whether I know this guy ... the maintainer has to trust that the person will be around"*.

This attitude appears to be a gate that signals specific beliefs, such as the fact that commitment to the community must be demonstrated by the potential contributor, and winning the approval of the gatekeeper is critical. This necessary trust between contributor and gatekeeper comes from an ongoing relationship between the two individuals that exhibits trustworthiness. Once the contributor succeeds in dealing with the "no", then, the contribution is evaluated for its technical merits as explained by this interviewee, *"On some parts of the kernel building trust is essential, and there is a clear social entry barrier. It has some downsides for beginners. Yet it's understandable, as changes in the kernel always come with some kind of maintenance overhead, and maintainers want people that have proven to take ownership of their contributions ... However, once a patch is considered, then it goes through thorough vetting."* (interviewee 24)

> ***Protective*** is the PR governance style that relies on trust, relationship building and the contributor's reliability.

The interviews data shows that the Linux Kernel community PRs evaluation process exhibits the characteristics of a protective style. It appears that the protective style of governance is distinctive to the Linux kernel community that prioritize trust, reliability, and the contributor-maintainer relationship.

4.1.2 Equitable. The equitable governance style is about being fair and impartial regardless of who is the contributor. It is transactional in nature. The PRs submission evaluation is concerned with technicalities and less with social aspects. Interviewee 3 stated, *"We try to be very impartial, we try not to make interactions very personal because code change isn't about friends it's not about being friendly it's about managing a technology. And so there is a very straightforward mechanism of submitting code changes"*. This was echoed by many interviewees in several communities. Another interviewee stated, *"It's very transactional, and that's just one way of doing it and it's a way that we like because it keeps personalities out of it and it makes rejections not personal ... Yes we tend to keep personalities to minimum"* (Interviewee 9).

In this style, the community principles overrule any leniency toward contributors. Rejection is not loosely applied, but it is a social responsibility. Interviewee 7 stated, *"Rejections of pull requests are a social responsibility and are taken with a fairness in mind"*. The community exhibiting an equitable style applies a set of principles seriously during the evaluation process. Interviewee 8 stated, *"There are principles for evaluating pull requests, and we religiously obey them...and we will usually reject a pull request if it doesn't hold up to these principles"*. FOSSASIA and Odoo communities appear to be equitable.

> ***Equitable*** is the PR governance styles that values fairness and rigorous application of community principles.

4.1.3 Lenient. The lenient style of pull request reviews is a tolerant and compassionate style of governance prioritizing growth and openness of the community. The lenient governance style was prominent in the data collected from the Coala community. Interviewee 27 explained, *"We accept errors. Instead of rejection, we embrace the enthusiasm of the contribution"*. *"My first PR was reviewed 65 times but not rejected"* (Interviewee 26). The community is willing to invest in the contributors abilities by mentoring them to learn how to submit PRs that meet the community standards. This investment has to pay off at one stage, as this interviewee explains,

Category	Code or Theme	Definition	Example of verbatim
Software Engineering Principles	**Quality**	Quality is a subjective concept to FOSS contributors. This subjectivity is offsetted by reaching a consensus about when a piece of code make a "quality" contribution.	*I am not sure there is a specific way to assess quality. We can read through the code and we know good code from bad code. It is quite subjective. However, in our community, there is a requirement for a minimum 3 reviewers to approve code. That makes it objective.* Interviewee 28
	Avoid Technical Debt	Technical debt is the owing inherited from a contribution when it doesn't meet certain quality and design requirements.	*I will not add something that increases my maintenance burden unless it's very compelling functionality or an obvious bugfix. I can't maintain a system I don't fully understand, so I like keeping things lighter and cutting off edge cases rather than adding technical debt I don't have time to pay off.* Interviewee 8.
Social Norms	**Trust**	Trust is the unyielding belief that the person is truthful and reliable.	*There are obviously criteria that have to do with the contributor, I would mainly look for reliability and trustworthiness of the contributor.* Interviewee 9.
	Mentoring	Mentoring is establishing a support relationship between a mentor and a newcomer. A mentor is someone who partners with a newcomer during his or her early period of engagement with the community. The mentor offers advice and guidance to help foster and promote the development of a newcomer. The mentor knows the community, its products and processes, and can be an effective source of advice and encouragement.	*I had a mentor for 3 years. He helped me to become a better developer and an effective member of the community* Interviewee 27
Product Vision	**Feature within the community vision for the product**	Some FOSS communities set a vision for their products. Contributions have to fit within the defined vision and goals.	*My pull request have been rejected because generally, the maintainer does not find the feature aligned with the goals of the project.* Interviewee 16
PR Governance	**Protective**	Protective means designed or intended to guard or shield the code base from undesired and low quality contributions. It operates based on trust, relationship building and the contributor's reliability.	*It's easy for me to get patches in because people in this community trust me and know who I am.* Interviewee 20
	Equitable	Equitable means fair and impartial, all contributions are judged for their technical merits and suitability for the community product's vision.	*Contributions are assessed fairly and based on their quality not the contributor. Sometimes, it feels transactional and unsupportive.* Interviewee 9
	Lenient	Lenient means tolerant for errors but at the same time it does not compromise quality. Contributors are mentored to elevate the quality of their contribution to mergeable standards.	*When I joined the community, my pull requests were not rejected. Instead, I was shown by the mentor how to improve them and make them mergeable. Now, I produce high quality code, because I learned.* Interviewee 27

Table 4: Examples of Categories and Themes with definitions

Communities	Protective	Equitable	Lenient
FOSSASIA		✓	
Odoo		✓	
DuckDuckGo			✓
Linux Kernel	✓		
Coala			✓

Table 5: The studied communities PR governance styles

"You can't spoon-feed the developers all the time either. They have to demonstrate their abilities" (Interviewee 27).

The lenient governance is based on the belief that any contribution is an asset that should not be ignored. A contribution carries an enthusiasm that should be leveraged for the benefit of the community. Interviewee 26 stated, *"We have a rule in our community that we never, ever reject a PR. Instead, we manage the contribution and improve it. We make every PR mergeable"*. Another said, *"Rejections kill motivation and, it is a rude thing. We instead steer the contribution to a positive direction by making it better, and get it merged"* (Interviewee 18). However, this is not a compromise of quality. Lenient communities ensure quality by mentoring contributors to elevate their contributions to mergeable standards. We observed that the DuckDuckGo and Coala communities appear to be lenient.

> ***Lenient*** is the PR governance style that reduces social barriers and assumes that every contribution can be elevated to a mergeable state.

The literature suggests that socio-technical factors interfere with these perceived strategic styles of governance. FOSS reviewers review social signals more than they reported they did [12]. Ford, et al. report that, while reviewers reviewed code most (64%), they also reviewed technical (28%) and social signals (17%). Even when they do not realize it, reviewers consider social signals. Developers

should stay aware of their image on various social networks. Sharing one's image on social networks makes one trust that a person is who they say they are. If one feels unsure about revealing their own identity, they should use a pseudonym frequently enough to make it recognized. Completing the online profile is also important. In summary, identity is very important in today's open source communities [12].

The protective and lenient PR governance styles show that in some instances, the person and the code matter, while, the equitable style focuses on the code quality. Interviewee 6 stated *"the quality is more important than the person"*. The commonality across these PR governance styles is safeguarding quality. Interviewee 25 explains, *"the willingness to insist on quality is key to the success of PRs processes in FOSS projects". "The process in place seeks the best. The best code quality possible"* (Interviewee 20). PR governance delivers good outcomes. It is achieved by both the creation and use of systems that ensure consistency and repeatability of processes.

Each PR governance reduces the threat of poor code. Consistency and governance create a culture of excellence. *"The quality is the main driver that drive our decision to either accept or reject a PR. The processes are there to support and control the decision-making"* (Interviewee 2). This perspective contribute to the sustainability of software quality in FOSS. Interviewee 2 further explains, *"first reliability of the code. Open source is ever changing, people come and go. High quality code and the ability to read the code and understand it is critical"*. This belief was echoed across the various PR governance styles. A voice from a lenient community said, *"We keep contribution's code quality in the check, but at the same time we are trying to be lenient towards contributors to really help them out to get the codes to the level where it can be merged"* (Interviewee 29). These beliefs and behavior create a sustainable culture for quality.

4.2 RQ2: Principles of Evaluating Pull Requests

4.2.1 Software Engineering Principles. In the three styles of governance, once the PR is considered, it goes through an evaluation against a set of software engineering principles. The proposed change must also add clear value to the project. As this interviewee explained, *"We measure the success of a pull request by its ability to add value to the application or the community. It could be for example a legitimate feature, a payment of technical debts, etc."* (Interviewee 27).

There is a strong belief among the studied communities that quality is supreme, and quality is seen as a necessary quality of pull requests. Interviewee 15 stated, *"In open source projects, we like to achieve higher code quality because it is open source and we will need to get good quality code"*. Another one asked how he evaluates PRs, he replied, *"quality, quality, quality ... it always comes first"* (Interviewee 3). Interviewee 20 went so far as to claim that *"there are people who give up; not everybody can write the required quality of code."*

In the studied communities, quality is constructed of seven principles: (1) PR atomicity, (2) maintainability, (3) avoiding technical debt, (4) passing peer code review, (5) Compliance with best practices, (6) documentation, and (7) passing tests. These principles are not always documented and communicated. However, reviewers are aware of them and claim to rigorously apply them. Interviewee 8 stated, *"we have a well-established set of principles by which we evaluate PRs and we say 'no' when a PR doesn't meet our standards"* (Interviewee 8).

PR Atomicity. Atomicity is a requirement that the PR should be composed from relatively independent parts that can be understood separately and (possibly) reused. Our interviewees were aware of the evaluation criteria for PR atomicity, and they made it clear that atomicity is a key aspect of quality. Interviewee 27 stated, *"a pull request should be addressing one atomic concern and not more"*. The concept of atomicity is a common belief. Interviewee 9 stated, *"messy and bulky code is no good in open source"*. That it seems atomicity is ingrained in FOSS contributors' behavior. *"Anything more than 50 lines of changes, and my brain doesn't have the capacity to do a good code review"* (Interviewee 8).

Maintainability. Coleman, et al. [5] define maintainability as *"the ease with which a software system of component can be modified to correct faults, improve performance or other attribute, or adapt to a change environment"*. The interviewees strive to achieve code maintainability. Interviewee 8 states *"this is open source, we have to keep maintainability in mind all the time. The code must be neat and tidy and caters for long term changes"*. Maintainability also includes looking after the long-term of the project. Interviewee 24 states *"so many projects get derailed by accepting too many new features without evaluating them for long-term maintainability, and it is a problem that is avoided by a simple two-letter word - no."*

Technical Debt. The term "technical debt" describes a universal problem that software engineers face, which is the problem of how to balance immediate value with long-term quality. The term refers to a shortcut made for expediency, bad code, or inadequate code. This "debt" accumulates and causes increasing costs, or interest, to system quality in maintenance and evolution. This debt can be taken on deliberately, and then monitored and managed as principal repaid in order to achieve business value. Architectural choices are the major source of technical debt and often occur as a result of emphasis on fast delivery of features and limited budget [11].

Some interviewees in this study indicate an awareness of technical debt and its effects. They actively look at avoiding it. *"I will not accept something that increases my maintenance burden ... I can't maintain a system that I don't fully understand, so I like keeping things lighter and cutting edge. I strive to avoid technical debt, which we do not have time to pay off"* (Interviewee 17). Our data shows that maintainability and avoiding technical debts are tightly connected. Avoiding technical debts enhances maintainability and assuring maintainability encompass avoiding technical debt. Technical debt is a contingent liability with impact on the internal software qualities, primarily, maintainability and the evolution of the software.

Peer review. Before a code contribution can be added to the code repository, it must receive a positive review by a pre-determined number of reviewers, usually three to five. *"We have a definite principle that we have five reviewers that must approve the pull request"* (Interviewee 4). Each reviewer examines the code visually and subjectively to assess its quality. Reviewers provide necessary feedback concerning the code review. If they submitted code that does not meet the reviewers' judgment of quality code, then the code goes through cycles of iterative improvements until it is deemed good

enough for the code repository [1]. The studied FOSS communities believe that peer review is the mechanism that assures quality, that it is a valuable quality assurance practice. Peer code review is religiously adopted in the studied FOSS communities. *"There is no PR assessment without code review obviously. We have this non-negotiable rule that every PR must pass code review"* (Interviewee 1).

Best practices. The studied communities have agreed on best practices for the programming languages they use. During the evaluation of PRs, reviewers make sure that these best practices are applied. *"Pull requests reviews must follow the community best practices"* (Interviewee 4). In some communities, best practices go beyond the coding conventions and guidelines. For example, in FOSSASIA, the contributors' conduct is also covered with a best practice evaluation. *"First thing is when we sign up for FOSSASIA contributing, there is a list of rules that we have to follow, and these include being nice to people who are around you, and secondly is the code and standards for the code. The next thing is that we do not merge anything and everything that comes to the repositories"* (Interviewee 5).

Documentation. In FOSS, the documentation usually explains how the code operates, how decisions are made during the programming, and how to use and amend the code. *"We really focus on documentation because we believe a project can strive in a community with knowledge being documented"* (Interviewee 3).

Tests. The communities that we studied use various types of testing, such as unit testing, continuous integration, and integration testing. During pull request reviews, reviewers look to see if the PR has passed the necessary tests. *"We make sure there are proper tests to verify that a pull request works as expected. Pull requests will not be accepted without the proper tests"* (Interviewee 27).

> Once a PR is considered for a review, a set of software engineering principles are applied to assess its eligibility to be merged.

4.2.2 Social Norms. "Norms are properties of a group, they describe the typical or desirable behavior of a certain social group." Individuals know what behaviors are expected of them because social norms are communicated through verbal messages and modeled behaviors. Those not abiding by social norms are identified informally by social cues such as being isolated or rejected. Social norms are powerful and effective, and they are less resource intensive than incentive based or punishment systems [26]. We identified three social norms:

Trust. Trust is defined as the willingness of the community to rely on the contributor, the principle of trusting the contributor as a precondition for considering his or her code change. We observed this in the Linux Kernel community. This principle is unique to the protective governance style. Interviewee 24 stated, *"Changes to the kernel can be complex! I need to be able to trust the contributor to the point that I know he will be around to take ownership of the code"*.

Establishing trust requires time. This time element makes it an entry barrier for newcomers. Other communities seem to have addressed this type of barrier and aligned the process to work solely with community principles. Interviewee 30 stated, *"We don't have entry barriers, but we ask the newcomers to obey our principles"*.

Contributor-Maintainer Relationship. Having a relationship with the maintainer is an advantage in the process of getting a pull request accepted. Interviewee 21 stated, *"What helped is that I meet these people in person. It's a basic human thing. When you meet a person, it's not like a mailing list. Actually, it's a physical thing; you release a chemical called oxytocin"*.

Mentoring. Mentoring is a practice put in place by some communities to help less experienced contributors to meet the community standards. Experienced contributors and evaluators take the time to work with the contributor to improve her submission. This action encourages additional submissions by that person and other observers as well.

Mentoring was observed in FOSSASIA, Coala, and DuckDuckGo among others. Interviewee 13 described mentoring, *"Pull requests that cannot be merged require mentoring. We have enough patience to work with the contributor to get it into a mergeable state. We mentor the contributor to do so"* (Interviewee 13).

> Trust, the contributor-maintainer relationship and mentoring are norms that take place during the evaluation process of PRs.

4.2.3 Product Vision. Some communities define a roadmap for their product and document it. During the evaluation of PRs, the proposed change is assessed whether it fits within the defined roadmap. *"We do not like to say no but we do to protect the evolution of the project"* (Interviewee 9).

> Pull requests proposed changes must adhere to the community roadmap for its products, in order to increase their chances of being accepted.

4.3 Trustworthiness and Limitations

Qualitative researchers pursuit trustworthiness for validity and credibility [31]. Trustworthiness is ensured by the establishment of these four traits: credibility, transferability, confirmability and dependability. Credibility refers to the confidence that the qualitative researcher includes the truth in the research study's findings. Transferability is the quality of the research demonstrating how the qualitative research can be applied to other contexts, that is similar situations, similar populations, and similar phenomena. Confirmability refers to how neutral the findings of the research study are, or how true the premise is that the responses are neutral and do not show any potential bias or personal motivations of the researcher. Dependability refers to the assurance that the study could be repeated by other researchers and that the findings would be consistent [18].

To establish credibility we used peer debriefing and member checking. One author conducted the coding and the other two authors validated the emerging codes and categories against the raw data. Six debriefing sessions were organized. We also used members checks to enhance the validity. We used it for narrative accuracy checks, and interpretive validity. We sent the interviews transcripts and description of the findings to the participants for validation. We collected data from five communities. This should strengthen the transferability of the findings.

We also used an audit trail to document and track the decisions we made throughout the study. This allowed us to meet confirmability requirements. An audit trail is the details of the process of data collection, data analysis, and interpretation of the data. To ensure dependability we compiled a research method that is s logical, traceable, and clearly documented [38]. When the research process is described thoroughly, the research audience is in a better position to judge the dependability of the research. If the process of the research can be audited, then it can ensure dependability [18].

Limitations. Linux and FOSSASIA interviewees were recruited through our contacts in the community. This make the Linux and FOSSASIA participants sample convenient. Convenience sampling has its criticism; it may not be representative of the targeted population.

5 DISCUSSION

The pull request governance styles have a reason to exist. Some are legacy, a result of years of ingrained culture and practice. Some are well crafted strategies put in place after years of trial and error learning. In all cases, these governance styles impacts their respective communities.

5.1 Protective

The protective governance style may create a *"clique"* culture difficult to access for newcomers. Newcomers may feel less important than the established core members of the community. Yet the community always needs newcomers, as creativity requires fresh minds and an ongoing flow of ideas and new contributions. Ostracizing those who are not inside the community may hinder its evolution and sustainability.

However, the protective style remains a good fit in some circumstances, such as when the FOSS project requires tight control over its code. It may be the appropriate style for a community to choose when they have an ongoing project where an influx of newcomers is not important, or when it is high anyways. Tsay et al. write that well-established and mature projects are more conservative in accepting pull requests [39]. The Linux community is a successful and mature project. Berger and coauthors note that the Linux community is a "closed platform" using heavy-weight processes. They also describe it as being a "centralized" structure; patches have to pass thorough reviews through the maintainer hierarchy [3]. They find it a justified practice when the project is developing a highly technical system, with a high barrier of entry, and high risk of introducing critical problems.

The protective style assigns relatively higher importance to commitment, relationships, and trust. Dabbish et al. report that both the contributor and the community look for signals of commitment. Frequency of recent submissions and the volume of activity by developers is a useful signal to the maintainer, while historical activity allows potential contributors to infer how well the project was managed. Visible actions on artifacts indicate the intentions, competence, and experience of the developers. Community support is inferred from the attention given, such as following, watching, and comment activity [7, 8, 23].

Relationships were shown to influence the evaluation of PRs. A chance of acceptance is higher for submitters already known to the core members of a project [40]. Also, maintainers interact more politely in discussions with core members than with new submitters [40]. The social connections between members of each of these groups can be measured on social distance and prior interaction values. Strong social connections increase the likelihood of acceptance, as they are markers of trust and allow to lower the assessment and coordination costs [33, 39].

5.2 Equitable

Although an equitable style of governance focuses on fair assessment, it does remain quite rigid. It is not a suitable style for communities that want to grow fast and attract new contributors, especially those with limited programming experience. Yet, this is the most preferred style of governance among the respondents overall.

The equitable style is suitable for communities aiming to attract experienced developers who are able to understand and incorporate advanced software engineering principles into their contributions. However, it does entry barriers for newcomers. Steinmacher et al. identified a list of barriers for newcomers, amongst them the need for orientation and technical hurdles [35, 36]. Communities which opt for this style of governance should communicate their evaluation principles clearly to contributors. They should educate contributors about their software engineering principles in their documentation.

5.3 Lenient

Mentoring contributors is a key part of the lenient governance style. This style is particularly well suited to communities with contributors with varied but limited experience in software development. An acceptance of the first contribution is an important step in a newcomer's socialization. She or he can learn the conventions and contribution rules through observation, lurking, and direct mentoring from more experienced members. Successful socialization allows potential contributors to learn the project norms and to identify the core members, where newcomers need to recruit allies [40]. After an initial period of observation, lurking, newcomers can assimilate the norms and values of the community. Then they begin to build an identity and become more visible to the core members, enrolling allies in the community. Once they demonstrate that they have the technical expertise, they are accepted by a community. Then they become an insider, not simply crafting material artifacts, but maintaining social relationships as well. They become a maintainer of the project, coaching and mentoring newcomers [10].

Attracting newcomers to communities is a major challenge. Fear of rejection that may harm reputation hinders some from contributing [16]. Lenient communities are aware of this issue and employ a strategy that minimizes rejections. Project members should show empathy toward new contributors, be engaged, and demonstrate fairness and positive attitude as mentors. Responsiveness and clear roadmap have also been identified by others as important factors encouraging newcomers [16, 17]. Berger et al. define variability encouragement as an open attitude to contributions from a broader ecosystem [3], and observed that some very fast growing ecosystems have openly and actively designed their processes and architectures to encourage external innovation.

Sim and Holt explain that a major downside of mentoring is that it is very time consuming for the senior developers in the community [32]. To some extent, the time required is compensated by attracting newcomers more easily.

5.4 Community Governance Vs. PR Governance

"Every development organization makes decisions and has some form of governance – this may be done explicitly or implicitly" [4]. FOSS projects are characterized by a specific framework through the lens of transactional cost economics called "bazaar" governance. This mode of governance is neither market nor hierarchy nor network, but is a governance system in its own right [9, 19].

FOSS governance is seen as the means of achieving the direction, control and coordination of autonomous individuals or organizations [22]. Community managed governance features are independence, pluralism, representation, decentralized decision making, and autonomous participation. The communities have a diverse group of participants that rests with the members of the community itself. An independent community allows decisions to be made at the lowest levels of the hierarchy, volunteers who may not be paid for their work. An independent community is deemed independent by its basis of material support, decision making structure, and independence from authority. A pluralistic community has a geographically diverse base of developers, community members who use a variety of ways to manage conflict, and leaders that emerge. Decision making occurs at the code level, the sub-project level, and the community wide level. Examining how members gain code level access rights, decision making rights, and the degree to which project communications and activities are publicly available lets one determine the mode of decision making [25].

PR governance is a process governance. Richardson [28] characterizes process governance as "consists of the set of guidelines and resources that an organization uses to facilitate collaboration and communication when it undertakes enterprise process initiatives." PR governance is the set of rules and controls that take place during the process of pull requests evaluations. It is doing what is required to assure that quality is produced by the process in the most efficient and effective manner possible. This is governance at the operational level of the community. Whether there is a link between community level governance and operational level governance is not something we explored.

6 RELATED WORK

The topic of PR-based collaboration has attracted some attention recently [16, 17, 21, 23, 27, 33, 34, 41–44]. To our best knowledge, no prior work attempts to conceptualize and distinguish the different governance styles in PR-based collaboration.

Soares and coauthors [33] find that the chance of a merge is 32% lower for first time contributions, supporting our intuition that the protective and equitable styles of governance are unfriendly to newcomers. In general, the chance of acceptance for a PR is 17% higher when tests are included, and 26.2% lower when many lines of code are changed [39]. This is inline with our findings, that passing tests and modularity of contributions are key criteria applied in evaluating PRs. The study has also shown that social distance and prior interaction with the maintainer are key influencers on acceptance chances [39]. This is consistent with our observation that social connections, trust, relationship building and commitment to the community, are considered in the PR evaluation processes.

Tsay et al. [40] note that maintainers were particularly concerned with the appropriateness of the contribution's actual content and direction. Appropriateness in this study is defined as fitting the product vision set by the community. We concur that adhering to the product vision is one of the evaluation criteria for PRs in the studied communities.

Marlow, et al. [23] study examines how interpersonal impressions influence evaluations of others' contributions. The analysis identified three scenarios where users sought out more information about each other. These scenarios are discovery, informing interaction, and skill assessment. Individuals form impressions about specific areas of expertise so that they can assess ways the coder can assist the project. They also make judgments about individual's personality. Arguments or rudeness in posting often are seen as indicators of uncooperativeness or arrogance [23]. This study concurs and complement our findings. It confirms that social inferences are part of the PR evaluation process. It complements our findings by suggesting that GitHub social signals are leveraged to make social inferences about contributors.

In FOSS communities, proper evaluation is seen as more important than addition of a feature. Developers prefer to postpone reviews rather than rush through them [29]. We observed similar attitude amongst our interviewees. They prefer investing great care and attention to detail rather than following a pre-defined checklist. This rigour coupled with the passion for the project lead to excellence in the evaluation process.

7 CONCLUSION

Modern software engineering heavily relies on open-source software. FOSS communities mostly emerges and organizes organically [37]. Measuring and tracking the organizational structure type and characteristics of an observable community is critical to achieve quality because identification of these systems provide organization problems that recur, such as motivation or trust, isomorphism, software failures, lack of centralized management of leadership, and stagnation. Software engineering research still lacks reference quality and models, but measuring community quality models can improve quality of software [37].

The PR governance styles foster a productive development and ensures high code quality. The controls and rules aim to improve the quality of source code changes made by developers, and it is a transparent process.

The PR evaluation process has a significant impact on contributor's motivation, so it is important to understand it. There is much more to learn from contribution evaluation than just simply whether the contribution is accepted or rejected, for instance we can understand better how to behave as community managers, reviewers, members, and how to enter communities more effectively. PR governance styles can be protective, with tight control of contributions; equitable, with a focus on technical fairness; and lenient, prioritizing community growth and retention by means of mentorship. Clearly, software engineering principles are not the

only criteria applied in PR evaluation; social and strategic criteria are also of high importance.

ACKNOWLEDGMENTS

Work supported by the ROSIN programme EU's H2020, grant No 732287. We thank the interviewees for making this research possible.

REFERENCES

[1] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wasowski. 2019. Why does code review work for open source software communities?. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 1073–1083.

[2] Maria Antikainen, Timo Aaltonen, and Jaani Väisänen. 2007. The role of trust in OSS communities—case Linux Kernel community. In *IFIP International Conference on Open Source Systems*. Springer, 223–228.

[3] Thorsten Berger, Rolf-Helge Pfeiffer, Reinhard Tartler, Steffen Dienst, Krzysztof Czarnecki, Andrzej Wąsowski, and Steven She. 2014. Variability mechanisms in software ecosystems. *Information and Software Technology* 56, 11 (2014), 1520–1535.

[4] Sunita Chulani, Clay Williams, and Avi Yaeli. 2008. Software development governance and its concerns. In *Proceedings of the 1st international workshop on Software development governance*. ACM, 3–6.

[5] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. 1994. Using metrics to evaluate software system maintainability. *Computer* 27, 8 (1994), 44–49.

[6] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 275–284.

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 1277–1286.

[8] Paul B De Laat. 2010. How can contributors to open-source communities be trusted? On the assumption, inference, and substitution of trust. *Ethics and information technology* 12, 4 (2010), 327–341.

[9] Benoit Demil and Xavier Lecocq. 2006. Neither market nor hierarchy nor network: The emergence of bazaar governance. *Organization studies* 27, 10 (2006), 1447–1466.

[10] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368.

[11] Neil A Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L Nord, and Ian Gorton. 2015. Measure it? manage it? ignore it? software practitioners and technical debt. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 50–60.

[12] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. 2019. Beyond the code itself: how programmers really look at pull requests. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*. IEEE Press, 51–60.

[13] Linux Foundation. [n. d.]. 2017 Linux Kernel Report Highlights Developers' Roles and Accelerating Pace of Change. https://www.linuxfoundation.org/blog/2017/10/2017-linux-kernel-report-highlights-developers-roles-accelerating-pace-change/

[14] Graham R Gibbs. 2007. Thematic coding and categorizing. *Analyzing qualitative data* 703 (2007), 38–56.

[15] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 345–355.

[16] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 285–296.

[17] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 358–368.

[18] E. G. Guba and Y. S. Lincoln. 1985. Naturalistic inquiry (Vol. 75). *Beverly Hills, CA: Sage* (1985).

[19] Kieran Healy and Alan Schussman. 2003. *The ecology of open-source software development*. Technical Report. Technical report, University of Arizona, USA.

[20] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research policy* 32, 7 (2003).

[21] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. 2017. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology* 84 (2017), 48–62.

[22] M. Lynne Markus. 2007. The governance of free/open source software projects: monolithic, multidimensional, or configurational? *Journal of Management & Governance* 11, 2 (2007).

[23] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 117–128.

[24] Matthew B Miles, A Michael Huberman, and Johnny Saldana. 2014. *Qualitative data analysis: A methods sourcebook. 3rd.* Thousand Oaks, CA: Sage.

[25] Siobhán O'Mahony. 2007. The governance of open source initiatives: what does it mean to be community managed? *Journal of Management & Governance* 11, 2 (2007), 139–150.

[26] Elizabeth Levy Paluck and Laurie Ball. 2010. Social Norms Marketing to Reduce Gender Based Violence. *IRC Policy Briefcase* (2010).

[27] Mohammad Masudur Rahman and Chanchal K Roy. 2014. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 364–367.

[28] Clay Richardson. 2006. Process governance best practices: Building a BPM center of excellence. *Business Process Trends* (2006).

[29] Peter C Rigby and Margaret-Anne Storey. 2011. Understanding broadcast based peer review on open source software projects. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 541–550.

[30] Colin Robson and Kieran McCartan. 2016. *Real world research*. John Wiley & Sons.

[31] Andrew K. Shenton. 2004. Strategies for ensuring trustworthiness in qualitative research projects. *Education for information* 22, 2 (2004).

[32] Susan Elliott Sim and Richard C Holt. 1998. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In *Proceedings of the 20th international conference on Software engineering*. IEEE, 361–370.

[33] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 1541–1546.

[34] Daricélio Moreira Soares, Manoel L de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 960–965.

[35] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*. ACM, 1379–1392.

[36] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. 2018. Almost there: A study on quasi-contributors in open-source software projects. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 256–266.

[37] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. 2019. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering* 24, 3 (2019), 1369–1417.

[38] Sarah J Tracy. 2010. Qualitative quality: Eight "big-tent" criteria for excellent qualitative research. *Qualitative inquiry* 16, 10 (2010), 837–851.

[39] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. ACM, 356–366.

[40] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 144–154.

[41] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for it: determinants of pull request evaluation latency on GitHub. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 367–371.

[42] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1. IEEE, 335–342.

[43] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[44] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 871–882.