Pull Request Governance In Open Source Communities

Adam Alami, Raúl Pardo, Marisa Leavitt Cohn, and Andrzej Wąsowski, *Member, IEEE*

Abstract—Pull requests facilitate inclusion and improvement of contributions in distributed software projects, especially in open source communities. An author makes a pull request to present a contribution as a candidate for inclusion in a code base. The request is inspected by maintainers and reviewers. The initiated process of review and collaborative improvement can be loaded with debates, opinions, and emotions. It heavily influences the atmosphere in the community. It can demotivate and detract contributors or it can fail to guard the code quality. Both problems put the existence of a community at risk. This mixed methods study aims to elucidate the mechanisms of evaluating pull requests in diverse open source software communities from the perspectives of developers and maintainers. We interviewed 30 participants from five different communities and conducted a survey with N=387 respondents. The data shows that acceptance of contributions in open source depends not only on technical criteria, but also significantly on social and strategic aspects. As a result, we identify three governance styles for pull requests: (1) protective, (2) equitable, and (3) lenient. 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 the community standards are met. Each of the governance styles safeguards the quality of the project code in different ways. We hope that this material will help researchers and community managers to obtain a more nuanced view on the peculiarities of different communities and the strengths and weakness of their pull requests evaluation process.

1 Introduction

Into Scylla he fell, wishing to avoid Charybdis; the brave project maintainer treading between the prospect of a lonely struggle without enough contributors and a busy processing of overwhelmingly many low quality pull requests. The pull request (PR) evaluation process is one of the very few instruments she has at hand to control the influx of contributions. As one of our interviewees explains: *"how contributions are rejected is a major factor in a project's success. The structure of the acceptance process is such that it can easily be used to bully people, assert dominance, engage in various forms of emotionally abusive behavior."* Only 13% of PRs are rejected for technical reasons! [1] *"The toughest and most frequent challenges encountered by contributors are social in nature."* [2] The process involves specific technical and social rituals, and various behavioral patterns emerge throughout. It is formative for the community as a social whole and for the individual members. In this paper, we set off to deeply investigate the mechanisms, norms, and propensities displayed by participants in the PR evaluation. We ask the following questions:

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

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

We address these questions with a mixed methods study, based on extensive data transcribed from 30 interviews with contributors and maintainers from five communities, and a survey with N=387 respondents from 15 communities.

We define PR governance as the *system of rules, practices, and norms by which a community directs and controls the assessment of PRs.* We identify three archetypical PR governance styles and their underlying believes and norms. From this understanding, we extract lessons for community leaders and maintainers. For example, our interviewees value governance by technical merit over social connections. We identify a mixture of the following three governance styles in the studied communities (**RQ1**):

Protective: A defensive style of governance where the project leader and his subordinates have absolute power over what contributions are accepted; sometimes described in FOSS circles as *"no-by-default."* This style requires commitment from the contributor to win the trust and approval of the gatekeeper before the evaluation can take place.

Equitable: A style of governance based on fairness and the ascendancy of evaluation principles. It aims at a balanced and technically grounded decision. The community principles overrule any leniency toward contributors.

Lenient: A style tolerating some errors and mediocrity in favour of a positive and welcoming environment for contributors. The foundational belief here is that a contribution is an asset that should not be taken lightly. The contributor's enthusiasm should be leveraged for the benefit of the community. In order not to compromise the quality, the contributor is mentored to evolve the PR to acceptable quality.

Our data shows that each open source community, as a social group, tends to display a mixture of the three styles, often with a clear tendency towards one of them. We also see that despite the fundamental differences in governance, all three styles aim to safeguard quality and ensure the evolution of the project.

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 project. The software engineering criteria are specific rules and recommendations that must be adhered to by a PR to be assessed as *"mergeable."* For instance, many communities require that a PR addresses a single atomic concern. Social norms, like trustworthiness, guide behavior of developers. A contribution from a trustworthy community member who demonstrated prior commitment might be prioritized. Finally, the project vision must be met, or at least not contradicted, by the intent of the contribution.

FOSS communities are idiosyncratic social systems and the PR governance styles reflect their uniqueness. Understanding the values, norms and rituals of PR evaluation gives insight into the evolution, growth, and life cycle of these social systems. For example, a typical Coala contributor prefers mentoring a contributor to improve her

• *The authors are affiliated with IT University of Copenhagen in Denmark, E-mails: {adaa,raup,mcoh,wasowski}@itu.dk*

PR, instead of rejecting the contribution (94% probability according to our survey). On the other hand, 47% of the surveyed Linux Kernel maintainers agreed that prior trust is important in accepting a pull request, and 58% think the reliability of the contributor is a key factor (note that both are personal qualities, in principle independent from the quality of a PR in question). In other communities these numbers are only 7% and 11% respectively. We hope that our findings will inspire researchers to nuance the understanding and study of different FOSS communities, and that we can help community managers and influential maintainers to reflect about their practices in community interactions.

A shorter version of this work was presented before [3]. Here we extend that qualitative study with a quantitative survey of contributors and maintainers from 15 communities (N=387) to increase the credibility of the results. While our earlier work [3] sought depth and richness by using qualitative methods, with a narrow focus on five communities, we now aim to understand how the original findings manifest in a larger base of FOSS communities. For example, our claims regarding the governance style of the Coala community and the protective style of the Linux Kernel community, and the contrast between the too, are well visible in the quantitative data. The use of multiple methods can neutralize some of the disadvantages of using a single method and strengthen a study [4]. The quantitative data is analyzed using Bayesian data analysis [5], which is well suited to data with small bucket sizes (a small number of participants per community). Our use of Bayesian inference also demonstrates its merits for software engineering research, where it is still rarely employed [6].

We describe the study design and analysis methods in Sect. 2. Sections 3 to 5 discuss the key findings and reflect on limitations Section 5. We contrast our contribution with the related work in Sect. 6 and conclude in Sect. 7.

2 Methods

We want to understand, the *what* and *how* of the evaluation; how decisions are made, and how they *affect* the participating developers and the fate of the contributions. Given the uncharted nature of the question, we opted for a mixed qualitative and quantitative design, to identify the propensities of governance, to understand how they are manifested across a selected set of communities. The qualitative phase precedes the quantitative phase, and the design of the latter depends on the constructs and themes resulting from the former, aiming to broaden the empirical coverage of the data we used to draw our conclusions [4].

TABLE 1: A brief characterization of the FOSS communities that participated in this study

AngularJS is an open source platform that makes it easy to build applications for the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

Apache. The community known as Apache develops the Apache web server (one of the many projects under the Apache Foundation). Apache is claimed to be the most popular web server worldwide. The community is a group of volunteers distributed geographically and using Internet for collaborative development.

Coala develops a language-independent program analysis toolkit (linting and fixing) written in Python since 2015. Coala portrays itself as a community open to newcomers. Directions for newcomers are clear and encouraging. The community documentation outlines how to get started, beginning with meeting others in chat rooms and Gitter.

DuckDuckGo is a community that designs, develops and maintains a search engine committed to privacy. It was founded in 2008 by Gabriel Weinberg, the community grew as a search engine that does not track the user's searches nor sell information about it. By 2013, there were over three million users of DuckDuckGo. In 2014, DuckDuckGo was included in Safari and interfaced into Mozilla/Firefox.

FOSSASIA is a community that aims to facilitate the creation of a "sharing society," expanding knowledge, tools and opportunities, freedom of communication and expression for everyone. The community runs many diverse projects including SUSI.AI (an artificial intelligent personal assistance, help desks, and chatbot) and EventYaY (event management).

JQuery is an open and transparent FOSS community that designs the JQuery UI library. The community members work as a team to develop widgets, animations, and class names that can be used as themes or styles in mobile, desktop, and web applications. Because of the international character and audience, the project is developed to work in a variety of languages and cultures.

Linux Kernel. The Linux Kernel is a free and open source operating system kernel developed by the Linux Kernel community. Initiated in 1991 by Linus Torvalds, Linux Kernel operates under a GNU license and now reaches a large user base around the world.

Mozilla Firefox is a community developing a Web browser. Mozilla Firefox is a browser strives to adhere to (and contribute to) web standards and to guard users' privacy. Firefox is built on a mission of openness, innovation, and opportunity.

Node.js is a cross-platform JavaScript runtime environment using the V8 JavaScript engine from Google Chrome. Node.js provides a set of asynchronous I/O primitives that prevent JavaScript code from blocking. Libraries are written in a non-blocking manner, with blocking behavior being an exception.

Odoo is a community of 1500+ members building an ERP/CRM system. Odoo meets complex user needs with a solution that is easy to use and upgrade, feature-rich, integrated, supporting management of sales, inventory, procurement, accounting, business intelligence, etc. (30+ applications). Claimed the most installed business suite for small and large entities.

OpenGenus is a community developing tools for work with bad Internet connectivity: buffering data sources, saving web-pages and browse history, playing games, and to searching web, images and code offline. It operates online communities (of freelancers, entrepreneurs, and programmers) that facilitate discussion, mentoring, and knowledge exchange.

OpenSUSE is a Linux distribution providing a user-friendly desktop. OpenSUSE includes openQA (an automated testing service for build&release updates), OSEM (an event management tool), Jangouts (videoconferencing), YaST (a configuration tool), and Kiwi (management of binary images for diverse hardware).

Plone is an Enterprise CMS built on the Zope application server, powering intranets and web sites of large organizations, including the U.S. Federal Bureau of Investigation, Brazilian Government, United Nations, City of Bern (Switzerland), New South Wales Government (Australia), and European Environment Agency. It is recognized for its security and accessibility.

ReactJS is a JavaScript library for building user interfaces based on the reactive programming paradigm. Declarative views make code more predictable and easier to debug. React allows to build encapsulated components that manage own state and compose into complex UIs. Originally released in 2013 by Facebook, it has attracted huge popularity since.

ROS is a flexible open-source framework for robotics, built around a communication middleware with asynchronous and synchronous message passing and a distributed parameter system. ROS builds an ecosystem of robotics modules for geometry modeling, robot description, diagnostics, pose estimation, localization, mapping, and navigation.

2.1 Phase I: Qualitative Exploration and Understanding

We conducted 30 semi-structured interviews with contributors and maintainers from FOSSASIA, Coala, Odoo, DuckDuckGo, and Linux Kernel. A semi-structured interview allows the researcher to add and refine questions during conversation. This method is effective for gathering rich data, exploring a topic, and gaining insight into individual beliefs and behaviors. The used interview questions fall into introductory, core, and probing categories (Tbl. 2). The introductory questions 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. As developers often contribute to multiple projects, and often in different roles, we have asked them to speak only for the community they consider central for them. During the conversation, we clarified what is the nature and the tenure of their contributions.

We selected the FOSS communities that we felt would give us a deep understanding of the phenomena under study. We sought diversity in the studied communities (see Sampling sub-section below). We interviewed 30 maintainers and contributors in total. We randomly (indiscriminately, without a method, or conscious decision) searched for contributors and maintainers with valid emails in their GitHub profiles. We sent 87 invites to participate in the interviews. Thirty-two accepted our invites, two did not show up to the interviews. For Linux and FOSSASIA communities, we used prior contacts in the community to recruit participants. Later, a snowball sampling developed: we asked our contacts to introduce us to further contributors and maintainers. Table Table 3 summarizes the resulting demographics of the interviewees. The experience column shows the number of years each 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, making sure that they connect to each other appropriately, often serving as dispatcher. *Contributors* are developers and sometimes volunteer to review other developers' code.

As the subjects were distributed geographically, all interviews were conducted using Google Meet. The interviews lasted from 40 minutes to an hour, and generated in average twelve pages of verbatim transcripts.

We used thematic coding [7], [8] to analyze the data, following the guidelines of Robson and McCartan [9] and of Miles and coauthors [10]. 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 the codes and interviewees' statements 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 7 shows examples of our codes and their categories.

We stopped interviewing after attaining saturation [11] so when (1) all the data were accounted for, with no outlying codes or categories; (2) every category was explained in depth by the data supporting it; and (3) there was enough data to answer the research questions. To demonstrate evidence of saturation during the recruitment and analysis processes, we compiled a table of examples, which exemplify the reasoning we used to make decisions on gathering further data, available at https://github.com/itu-square/statistical-analysis-foss-governance-styles.

TABLE 2: Key parts of the interview framework

Intro. Can you talk to me about your community? What first motivated you to participate in this community?
Core. 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?

2.2 Phase II: Quantitative Survey

The qualitative analysis allowed to identify the PR evaluation constructs important to the five communities selected. In the

TABLE 3: The population of the interviewees in Phase I

Interviewee	Community	Role	Exp.[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

second phase, we extend the coverage to ten new communities, aiming at exploring the presence of the constructs in other communities, how the identified exemplary styles of governance manifest elsewhere. To this end, we posed the hypotheses in Tbl. 5.

Instrumentation: The survey had a total of twelve questions. All but the demographic questions, were derived from the qualitative findings. Tbl. 9 in appendix 1 lists the questions asked to the maintainers and contributors. The tables also explain the rationale of including the questions, i.e., how they were derived from the qualitative findings.

Sampling: The sampling happens at two levels in this study: the selection of FOSS communities and the selection of the survey participants, i.e., the maintainers and contributors. For FOSS communities (Tbl. 1), we used *purposive sampling* to select the cases, a strategy that uses specific characteristics relevant to the study's objective [12]. Our strategy is to include communities that are as diverse as possible. To ensure diversity, we used five dimensions in the selection: products, contributors demographics, the size of the community, the community history and the leadership style in the community. Table 4 presents and defines these dimensions.

For the selection of participants, we used random sampling with the exception of the Linux Kernel, where we used "respondent-driven sampling" [13]. We recruited three maintainers from the community and asked them to recruit other participants by circulating the invite to participate in the survey. This choice is due to the constraint exercised by the community to prevent unsolicited messages coming from outside the community. For the other communities, we used "Simple Random Sampling"; Ghauri and Gronhaug explain that this sampling implies that "every case of the population has an equal probability of inclusion in sample." [14] We used various techniques to implement this sampling strategy (see data collection sub-section below).

We sought the participation of maintainers and contributors actively contributing to their respective communities. The interviewees from Phase I did not participate in the survey of Phase II. To control our sample, we asked whether the visitor (to the survey first page) is an open source contributor or maintainer, allowing the choice of both and a negative response to both simultaneously. In the latter case the survey was stopped. The participants have not been compensated for participation.

Data Collection: We employed three techniques to attract participants: direct emailing potential participants, sending invites to mailing lists of the communities, and posting in the selected community fora.

- *Direct emailing:* We used GitHub and GitLab to manually search for potential participants in the selected communities. We examined contributors' and maintainers' profiles of the selected communities and randomly (without a specific logic) selected participants (active contributors and maintainers). We approached a total of 211 contributors and maintainers using this technique.
- *Invites to mailing lists:* We sent invites to participate in the survey to the available mailing lists. Some of communities have multiple mailing lists. It was not always possible to use all available lists, due to restrictions on the content of messages. We managed to send 22 invitation emails using this technique.
- *Invites in online fora:* Five of the selected communities use online discussion fora (e.g., the ROS community discussion forum at https://discourse.ros.org/). We posted invitations in the fora of ROS, FOSSASIA, Coala, Plone and OpenSUSE. In some cases, we first had to get the forum's administrator permission to post an invite.

Behavioral research rarely justifies quantitative sample sizes [15]. While we do not know the response rate, we are convinced that the obtained set of responses is of very high quality, and likely to represent opinions of the

TABLE 5: The study hypotheses

ID	Hypothesis statement
H1	The Coala community adopts a lenient style of governance for its pull request process.
H2	The Linux Kernel community adopts a protective style of governance for its code change process.
H3	The Coala community is more lenient than the Linux Kernel community
H4	The FOSSASIA community adopts an equitable style of governance for its pull request process.
H5	The Odoo community adopts an equitable style of governance for its pull request process.
H6	Each of the 15 FOSS communities adopts a governance style, either protective, equitable or lenient, for its pull request process.

TABLE 4: The dimensions used to select FOSS communities for the survey (purposive sampling)

Dimension	Definition	Examples from our cases
Products	The products developed by a community. We aimed for a sample including various types of software products.	Operating systems (Linux Kernel), content management systems (Plone), robotics software (ROS), web libraries (JQuery), etc.
Demographics	The ethnicity, age, and (national) culture of contributors and maintainers.	We aimed at diverse demographics. For example, FOSSASIA has predominantly south east Asian contributors. The Linux Kernel and ROS communities attract mostly contributors from Europe and North America. Other communities like Odoo and Apache are globally distributed.
Size	The number of contributors (including maintainers) estimated by GitHub/GitLab (The Linux Foundation estimate for the Linux kernel).	We selected communities with sizes ranging from large to small. The Linux Kernel reports 15,600 contributors, ROS has 5,600 and JQuery has 275.
History	The history of the community, its inception and development, possibly influencing behavior, attitudes, norms, and social contexts.	The ROS community started as a research project, evolved to a start-up, and later progressed into an open source community. FOSSASIA had evolved from a movement for "social change" advocated by three open source enthusiasts in south east Asia.
Leadership style	The style that the community leaders impact on the population of contributions.	While Linus Thorvalds (the Linux Kernel) is characterised as a "Benevolent dictator", other communities follow more meritocratic systems.

involved FOSS community members. We used dedicated and reliable community channels. Participation in the study was voluntary, there was no hidden, or monetary incentive to attract responses. Given these conditions, the set of responses is considerably large. We received 473 responses. The survey data are available at https://github.com/itu-square/statistical-analysis-foss-governance-styles.

We validated the responses using the following criteria:

1) The respondent must be an active contributor or maintainer in one of the selected FOSS communities,
2) All mandatory questions have been answered with valid input, and
3) Only one entry is allowed per respondent, controlled using IP addresses.

Participants who failed the check were excluded, leaving N=387 valid and complete answers.Out of these answers, 188 are from maintainers and 199 from contributors. Governance style questions (v_{27} - v_{33}) are maintainer specific, and they were not available for non-maintaners.

We used Bayesian statistics, also called *explicit probabilistic inference*, to analyze the quantitative data. This method provides formal means of dealing with uncertainty in scientific inference, by approximating unknown parameters using probability distributions. A Bayesian analysis begins with modeling prior distributions, capturing pre-existing knowledge of the problem. When new evidence is collected, the analysis reallocates probability in the model to revise the prior beliefs. To answer yes/no questions, we often use a simplifying abstraction of *High Density Interval* (HDI). The HDI describes the range of most credible conclusions accumulating 95% of probability mass. This limits the chance of erroneous conclusions to 5% [16]. For further robustness, a magnitude of effect can be identified known as the *region of practical equivalence* (ROPE). The ROPE is a decision threshold that is chosen in the context of current theory and measurement precision. If the ROPE, or region of values, does not include the high density values, then the value is rejected, but if the ROPE completely includes 95% of the high density values, the value is accepted because the high density values are the most credible values [16]. This allows Bayesian analyses to both accept and reject hypotheses, unlike in frequentist statistics.

Prior to analyzing the data, we linked the PR governance styles to the survey questions as shown in Tbl. 6. A response was categorized as positive if the 95% HDI falls entirely below 3 on a linear scale derived from the scale: 1 strongly agree, 2 agree, 3 neutral, 4 disagree, 5 strongly disagree.

In order to test the hypotheses of Tbl. 5, we use Bayesian inference to estimate the underlying distributions of the responses that each FOSS community gives to each question v_{27}–v_{33} employing *ordinal regression*; the standard model for analyzing Likert data [5, Chp. 23]. We assume that the answers to the questions are normally distributed (Gaussian likelihood function). We use a neutral prior, letting the inference find the values of the parameters for the underlying distribution that best accommodate the evidence from each community. This is appropriate as there are no preexisting studies of PR governance in FOSS to inform the prior. Our subjective understanding of the problem based on the qualitative analysis has instead informed the structure of the model and the choice of the hypotheses. We implemented the analyses in PyMC3 [17], a state of the art package for Bayesian statistical modeling. The Jupyter notebook is available in the paper data repository, linked above.

3 Findings

3.1 RQ1: Decision Making in PR Evaluation

We identify three archetypical styles of governance that have strongly manifested in the qualitative data: (1) lenient, (2) protective, and (3) equitable. Table 8 summarizes which governance styles are present in communities as identified during qualitative analysis.

Lenient Governance Style

The lenient style of PR evaluation is tolerant and compassionate, prioritizing growth and openness of the community. The lenient governance style was prominent in the data collected from Coala. 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 invests in the contributors' abilities by mentoring them on how to prepare PRs that meet the quality standards. This investment has to pay off at one stage, as Interviewee 27 clarifies: *"You can't spoon-feed the developers all the time either. They have to demonstrate their abilities"*.

The lenient governance rests on the belief that no contribution should be ignored. Each contribution carries enthusiasm that should be leveraged for the benefit of the community. Interviewee 26 says *"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). Lenient communities, do not compromise on quality—they *ensure quality* by mentoring contributors to elevate their contributions to mergeable standards. DuckDuckGo and Coala communities appear to be lenient based on the interviews.

We investigated the distribution of answers to v_{33} from Coala (**H1**).[1] To this end, we look at the distribution of

1. The participation of DuckDuckGo in the quantitative survey of Phase II was too small to permit statistical analysis.

TABLE 6: PR governance styles defined by survey questions

Protective: A community is protective when it responds positively to at least one of the questions v_{27} (*"In general I say no to most pull requests (PR)/patches"*), v_{28} (*"I don't consider a pull request/patch, unless I trust the contributor"*), v_{29} (*"I don't consider a pull request/patch, unless the contributor is reliable."*), and v_{30} (*"I don't consider a pull request/patch, unless I have a strong relationship with the contributor."*).

Equitable: A community is equitable when the response from this community is positive to at least one of the questions: v_{31} (*"I assess every pull request/patch in the same manner irrespective of the contributor."*) and v_{32} (*"I assess pull requests/patches purely on technical grounds."*).

Lenient: A community is lenient when the response to Question v_{33} (*"I never say no to a pull request/patch. If the quality of the PR/patch is not mergeable, then I mentor the contributor to elevate his/her PR/patch to a mergeable state."*) is positive.

the mean for v_{33}, the central tendency of the answers for the community. The distribution of the mean for lenient communities should be shifted towards low values, with concentration below 3 (Tbl. 6). Figure 1 shows that for Coala 98.8% of the density is below 3, and the entire 95% HDI, specifically (0.93,2.6), is below 3. We consider a ROPE of size 0.8 (from 2.6 to 3.4) which does not overlap with the HDI. We conclude that, given a ROPE of (2.6,3.4), the Coala community has a tendency towards a lenient governance style also in the survey data, and **H1** holds.

> **The adopters of *Lenient*** governance style are concerned with reducing social barriers. They assume that every contribution can be elevated to a mergeable state.

Protective Governance Style

This style of pull request evaluation is defensive. Our interviewees reported that during the evaluation process values such as trust, relationships and the reliability of the contributor are considered. The Linux Kernel community describes it 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 says: *"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 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: *"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*

TABLE 8: The studied communities PR governance styles

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

TABLE 7: Examples of categories and themes with definitions

Category	Code	Definition	Example of verbatim
software engineering principles	**quality**	Quality is a subjective concept to FOSS contributors. This subjectivity is offset 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's 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 partners with a newcomer during their early period of engagement with the community. She offers advice and guidance to help foster and promote the development of the 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
pull request 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

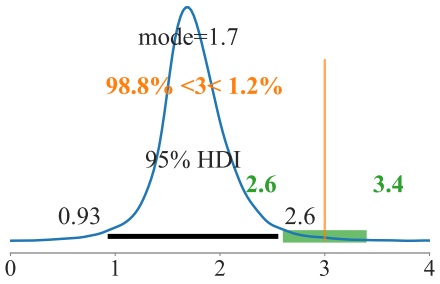
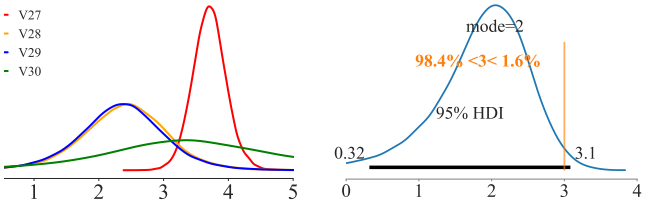


Fig. 1: Mean distribution for v_{33} of the Coala community



(a) Mean dists. for v_{27}–v_{30} (b) $\Pr(\min(v_{27}, v_{28}, v_{29}, v_{30}))$

Fig. 2: Protectiveness Analysis for Linux Kernel

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).

Hypothesis **H2** is set up to check how widespread this behavior is in the Linux community. We analyze answers to questions v_{27}-v_{30} that reveal the traits of protectiveness (cf. Tbl. 9). We will conclude that a community is protective if at least one of v_{27}-v_{30} receives predominantly positive answers. The mean answers should be distributed in the low part of the scale, with density concentrated below three.

Figure 2a shows that distributions of v_{28} and v_{29} are predominantly located in the interval [0,3], indicating that the Linux Kernel community indeed shows strong protective tendencies (data from 19 respondents, with a hierarchical model inferred using the answers of maintainers). To increase confidence, we plot the distribution of the minimum of mean answers for the four questions, which captures whether a typical Linux respondent exhibits at least one protective position (answer). Figure 2b shows that 98% of the density is below 3, and the 95% HDI is within the interval [0,3.1]. Thus the Kernel community shows a clear tendency towards protective governance. However, the tendency is not radical, with some overlap with neutral answers ($\geq$ 3). There are noticeably many respondents who do not exhibit protective attitudes. This might be a manifestation of transitioning towards a less protective governance style—a claim supported by the mean distributions for v_{27} and v_{30} (Fig. 2a) and open online debates of exclusivity and governance in the Linux Kernel project.

> The adopters of the ***Protective*** choose to rely on trust, relationship building and the contributor's reliability.

The interviewees portray the Kernel community as protective. Prioritizing trust, reliability, and the contributor-maintainer relationships appears to be distinctive of this community. However, the quantitative analysis uncovers that the community is not uniform on this issue. Some of our interviewees did admit that the community is trying to change its attitude toward contributing in general, which could explain the mixed results. Interviewee 25 explains: *"getting a patch accepted in the Linux community can be difficult for newcomers and unfamiliar names in the community. They have preference for trustworthy contributors. This attitude is cascaded from the top ... However, this is changing. I know a lot of subsystem maintainers who want a community with less fences."*

Leniency of Linux versus Coala

To validate that the difference between the Linux and Coala communities is not a random artifact emerging from our sample, we studied hypothesis **H3**. We estimated the distributions of the mean answers to question v_{33} exploring the attitude towards mentoring. We computed the difference between the mean and the standard deviation of the two estimated distributions, and checked the effect size between the two differences. The effect size is a standard method to compare two estimated normal distributions [5].

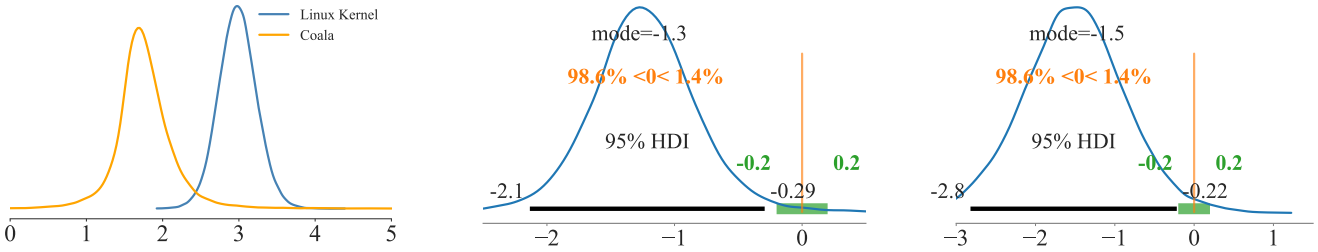
Figure 3a shows the distribution of the answers. The Coala community is more lenient than Linux. However, the density for Linux is concentrated in the interval (2.5,3.5). This means that the respondents from the Linux community mostly opted for a "neutral" response to v_{33}.

Figures 3b and 3c show the differences of distributions of answers to v_{33}, both the difference of means and the effect size. In both plots, the density is mostly located in the negative part of the plot. This means that the mean of the answers of the Linux Kernel community is consistently greater than that of Coala. The Coala community agrees more eagerly with the statement v_{33} than the Linux Kernel developers. We highlighted a ROPE of (-0.2,0.2) in the plots. This region marks what values we consider equal to 0 in practice. The HDI of both the difference of means and the effect size are outside the ROPE. Hence, we can conclude that, given a ROPE of (-0.2,0.2), there exists a difference between the distributions of the answers of the Coala and Linux Kernel communities validating hypothesis **H3**. The clear difference in the survey data reassures us that the qualitative analysis has identified a relevant general phenomenon.

These two communities diverge fundamentally in their strategies for management and evaluation of contributions. The choices are likely rooted in differences of leadership style, and in the history and legacy of both communities. Setting aside the rationale, we observe that each community adopts a style that fits their needs and mirrors their social anatomy.

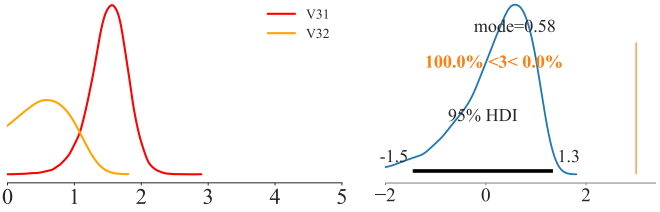
Equitable Governance Style

The equitable governance style believes in being fair and impartial regardless of who is the contributor. It is transactional in nature. The evaluation of a PR is concerned more with technicalities and less with social aspects. Interviewee 3 states: *"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, for example: *"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).



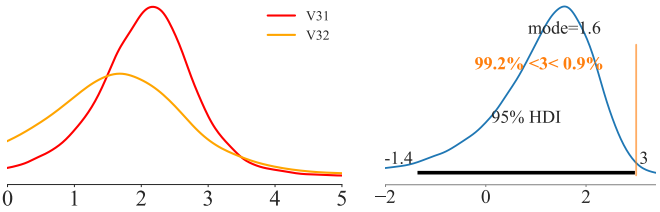
(a) Coala vs Linux Kernel mean dists. (b) Difference of mean dists. (c) Effect size

Fig. 3: Comparison of answers to v_{33} (leniency) for Coala and Linux Kernel



(a) Mean distributions for v_{31}, v_{32} (b) $\Pr(\min(v_{31}, v_{32}))$

Fig. 4: Analysis of equitability for FOSSASIA



(a) Mean distributions for v_{31}, v_{32} (b) $\Pr(\min(v_{31}, v_{32}))$

Fig. 5: Analysis of equitability for Odoo

In this style, the community principles, for evaluating a contribution, overrule any leniency toward contributors, but rejection is not applied lightly; it bears a social responsibility. Interviewee 7: *"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: *"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 appear to be equitable. To check whether a larger population confirms this we tested **H4** and **H5**.

To test **H4** and **H5**, we analyze the answers to questions v_{31} and v_{32}. We say that a community adopts an equitable governance style if it responds predominantly positively to any of these questions. We follow the same method as for the leniency and protectiveness analyses above. The plots in Fig. 4a shows that FOSSASIA exhibits equitable governance tendencies. The density of both distributions is well below 3; in fact, both modes are below 2. Similarly, most of the density of expected answers for Odoo are located below 3, but, in this case, there is non-negligible amount of the density located above 3 (Fig. 5a).

To further affirm this finding, we plot the minimum of the mean distributions (Figs. 4b and 5b). The plot of FOSSASIA clearly indicates the equitable style. The 95% HDI is well below 3, in fact the right limit is 1.3. Almost all members of the community strongly agree. We observe a similar curve for Odoo: the 95% HDI is mostly below, but includes 3. However, the answers are more widespread than for FOSSASIA. The right limit of the 95% HDI interval is 3; indicating that there are some participants that lean towards neutral answers. Still, we conclude that **H4** and **H5** hold.

> The adopters of the ***Equitable*** governance styles value fairness towards contributors and rigorous application of community principles.

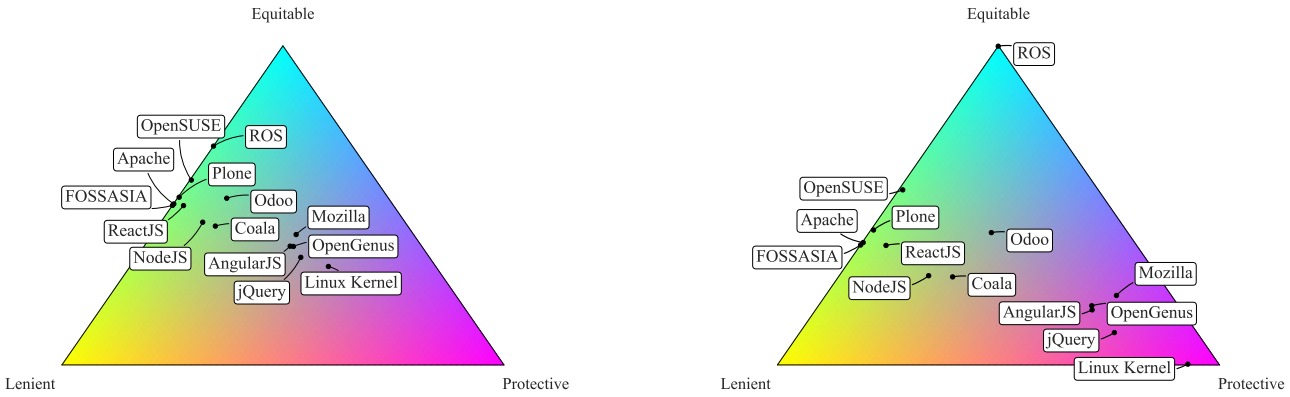
Governance Styles across 15 FOSS Communities

H6 compares the level of protectiveness, equitability and leniency of the communities in an aggregated fashion. We use a ternary plot that translates 3-coordinate points into an interior of equilateral triangle. For each community we establish the probability density below 3 for the respective questions (protective $\Pr(\min(v_{27}, v_{28}, v_{29}, v_{30}) < 3)$, equitable $\Pr(\min(v_{31}, v_{32}) < 3)$) and lenient $\Pr(\min(v_{33}) < 3)$. This characterizes each of them with a triple of numbers between 0 and 1. We normalize the points to sum to 1 and plot in a standard ternary plot.

Most communities exhibit a balance between equitability and leniency. A cluster is formed in the center of the left edge, seemingly equidistant from the lenient and equitable corners. The answers from communities located on the altitude line of the protective vertex balance equitability and leniency. The closer a community is to the protective vertex, more protective it is. For instance, Linux Kernel is the most protective. If **H6** held, we should observe clusters of communities in the corners of the triangle, which is not the case.

Admittedly though, the *extent* of the triangle in Fig. 6a is arbitrary. The corners represent extreme positions. For a community to be placed, say, in the equitable apex, all its members have to answer the equitability questions strongly positively and all the other questions strongly negatively. This is unlikely to happen for independent human subjects answering orthogonal questions. Perhaps the triangle should be drawn larger or smaller reflecting not the absolute position of tendencies on the Likert scale, but rather the relative differences of tendencies between communities?

Figure 6b re-scales the ternary plot. Intuitively, it "zooms in" into the central part of Fig. 6a. We identify the smallest



(a) Ternary plot of all FOSS communities vs styles

(b) Re-scaled ternary plot of all FOSS communities vs styles

Fig. 6: Comparison of all governance styles among FOSS communities

mean answers to equitability and leniency across all communities and use them to define the triangle's extremes. Thus, instead of defining the governance style in an absolute way (as a value of an answer on a Likert scale), we now define the styles relatively (as the difference between answers by communities). For example, the reference point for equitability becomes the *community that is the most equitable in our sample*. This way of interpreting the data is more robust with respect to the formulation of the questionnaire. The case analysis of extremes presented above in the section, confirms that these relative differences are meaningful with significant effects, not just due to random noise so amplifying them is justified.

The scaled figure reveals the differences in governance style in the communities under study. Two are clearly biased towards a single style: ROS (equitable) and Linux Kernel (protective). The others are still located along the altitude of the protective vertex, slightly shifted towards leniency. This means that behaviors reported in the answers from these communities still balance leniency and equitability, with a slight preference towards the former, while we observe quite a range of attitudes towards protectiveness. We cannot confirm **H6**, but learn instead, somewhat expectedly, that a model with three extremes is too simple to characterize complex behavior in a discrete way. Nevertheless, its continuous and relative interpretation provides interesting and valuable insights.

In summary, the Coala community follows a lenient style of governance for its pull request process both according to the qualitative and the quantitative data (**H1**). Similarly, the Linux Kernel exhibits a protective style both in interviews and the Bayesian tests of **H2**. The Coala community is more lenient than the Linux community (**H3**). FOSSASIA (**H4**) and Odoo (**H5**) are examples of an equitable governance style for PR evaluation. Finally, the analysis of Hypothesis **H6** revealed that across the communities adopting a style is more a matter of degree than a unitary discrete choice. Some communities show a clear preference, but a mixture of equitable and lenient attitudes is most common.

As seen in the analysis of data for **H6**, the governance styles do not induce an exclusive taxonomy of governance in PR evaluation in FOSS communities. A community cannot be confined to a particular style. Rather the styles define archetypical dimensions, exhibited to a different degree in the studied communities. Some communities show stronger inclination towards one style and not the others (e.g., ROS and Linux Kernel). The equitable and lenient styles coexist and complement each other's in most communities. Leniency can be either a collective choice (e.g., Coala) or an individual choice adopted by a group of maintainers and not necessarily shared by the majority of maintainers. Furthermore, FOSS communities are not static social systems. Like any social systems, they experience changes over time. In some communities the governance style can be cemented (e.g., ROS) but in others the governance principles undergo changes (e.g., FOSSASIA) that further contributes to the observed mix. This observation is consistent with the interview data, for example interviewee 2 says that his community was rather lenient but it has experienced growth; this has influenced maintainers and senior community members to become more equitable that confirms that the community is experiencing change. He stated: "When I join the community, we use to mentor newcomers and less experienced contributors mainly students. We help them and avoid rejections as much as possible to encourage them to stay and contribute more and hopefully become better. In the last couple years we experienced so much growth, we attract a lot of people. We just have no time to help and mentor. We have more contributors and less maintainers. We become rather strict with people. The code has to be good and meet our expectations. But some maintainers still prefer to mentor when they have time."

Each FOSS community has different culture, history, leadership, and values. Our analysis confirms that FOSS communities are not equal. The governance style exemplifes their believes, norms and culture. It influences the decision-making mechanisms in the PR evaluation process. Communities can also show a mixture of attitudes and undergo migration between the styles in this space.

Researchers suggest that socio-technical factors interfere with perceived strategic governance. Code reviewers interpret social signals more than they are willing to admit [18]. While reviewers focus mostly on code (64%), they also consider technical (28%) and social signals (17%). Developers should stay aware of their image in the community. Complet-

ing an online profile and projecting a consistent image, best a photograph, on social networks helps to associate trust with an identity. Identity is very important in today's open source communities [18]. The protective and lenient PR governance styles admit that not only the code but also the person matter. The equitable style focuses solely on code. *"the quality is more important than the person"* (Interviewee 6).

Still, all styles share the concern for 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). Governance *itself* ensures consistency and repeatability, *not* the individual styles. A consistent governance creates a culture of excellence, contributing to the sustainability of FOSS products' quality. Interviewee 2 explains: *"The quality is the main driver that drives our decision to either accept or reject a PR. The processes are there to support and control the decision-making. (...) 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 sentiment is echoed across the styles; a voice from a lenient project: *"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). A sustainable quality culture emerges.

3.2 RQ2.1: Software Engineering Principles in Evaluating PRs

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 feature 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 achieved by the adherence to 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. In the words of Interviewee 8: *"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."*

PR Atomicity

Atomicity is the requirement that the PR should be composed from relatively independent parts that can be understood separately and possibly reused. Our interviewees are aware of the PR atomicity as a quality criterion, and make it clear that atomicity is a key aspect of quality. *"A pull request should be addressing one atomic concern and not more"* (Interviewee 27). *"Messy and bulky code is no good in open source"* (Interviewee 9). *"Anything more than 50 lines of changes, and my brain doesn't have the capacity to do a good code review"* (Interviewee 8). Atomicity is deeply ingrained in the technical principles they follow.

Maintainability

Coleman et al. [19] define maintainability as *"the ease with which a software system or component can be modified to correct faults, improve performance or other attribute, or adapt to a change in environment."* FOSS developers in our sample are aware of of the relevance of code maintainability in the evaluation process: *"this is open source, we have to keep maintainability in mind all the time. The code must be neat and tidy and cater for long term changes"* (Interviewee 8). Maintainability also includes taking the long-term sustainability: *"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"* (Interviewee 24).

Technical Debt

The term "technical debt" describes a universal problem of software engineering: how to balance immediate value with long-term quality. The term refers to a programming shortcut taken to save time at the cost of creating inadequate code. The debt accumulates and causes increasing cost in terms of future maintenance and evolution. Debt may be accepted deliberately and then monitored and managed over time. Architectural choices are the major source of technical debt, especially in context of fast delivery of features at limited budget [20].

Several interviewees are aware of technical debt and its effects. They actively avoid 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"* (Interviewee 17). Our data confirms that maintainability and technical debt are tightly connected. Avoiding debt improves maintainability and assuring maintainability means avoiding technical debt.

Peer review

Before a code contribution can be accepted into a repository, it must receive a positive review by (usually) three to five reviewers. *"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. If the submitted code does not meet the reviewers' judgment of quality code, it enters cycles of iterative improvement until it is deemed good enough [21]. 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

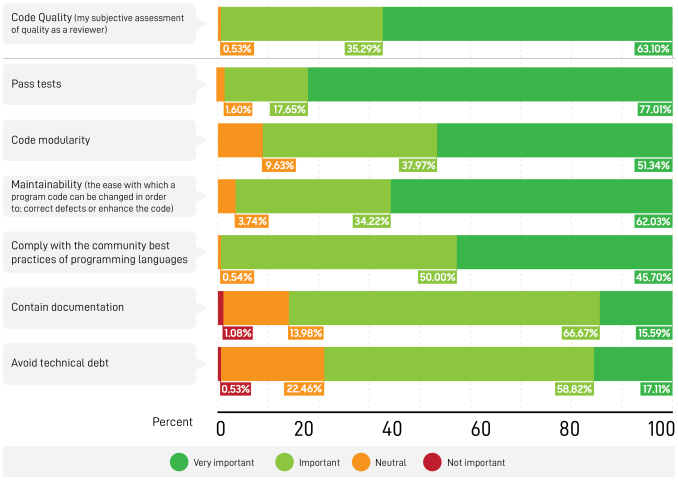


Fig. 7: Engineering criteria in evaluation (all respondents)

of PRs, reviewers make sure that these best practices are followed. *"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 communities we studied, 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 studied communities employ various types of testing, including unit testing and integration testing with continuous integration. During PR reviews, reviewers check 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, software engineering principles are applied to assess its eligibility for merging.

The qualitative analysis of the engineering principles aligns with the quantitative data of Phase II. We observe (Fig. 7) that all the above principles are important for majority of the respondents across all communities. Still, the differences between the importance attached to the various aspects of quality are quite interesting. For example, passing existing tests is almost universally required and almost equated to quality (same acceptance rate). This may be caused by the fact that this is a relatively easy criterion for assessment, especially if continuous integration is used. The remaining quality aspects are perceived with roughly similar weight (technical debt might by undervalued, as the term is less commonly known in some communities).

3.3 RQ2.2: Social Norms in Evaluation of Pull Requests

"Norms are properties of a group, they describe the typical or desirable behavior of a certain social group" [22]. Individuals know what behaviors are expected of them as social norms are communicated through explicit verbal messages and model 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 [22]. We identified three social norms regarding PR evaluation in our data: trust, contributor-maintainer relationships, and mentoring.

Trust

Trust is the willingness of the community to rely on the contributor, a precondition for considering her code change. This principle is unique to the protective governance style. We observed this in the Linux community: *"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"* (Interviewee 24). Establishing trust requires time, which creates a barrier for newcomers. Other communities seem to have addressed this barrier and aligned the principles to create trust. *"We don't have entry barriers, but we ask the newcomers to obey our principles"* (Interviewee 30).

The requirement of prior trust within Linux is supported by quantitative data. Approximately 47.37% of those surveyed agreed that trust is important in accepting a pull request and 57.90% think the reliability of the contributor is an important factor. In the other communities only 6.95% said that trust was important to accepting pull requests and 11.23% of respondents said that reliability was important.

Contributor-Maintainer Relationship

A prior relationship with the maintainer is an advantage when getting a pull request evaluated. Interviewee 21 recalls: *"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."* Interestingly, just 31.58% of the respondents from the Kernel community agree that a relationship with submitters affects acceptance, despite that qualitative data emphasizes its importance. Strikingly though, only 3.21% of respondents, from the other communities, indicated relationships as important in other communities.

Mentoring

Mentoring is a practice used to help less experienced contributors to meet the community standards. Experienced community members and evaluators work with the contributor to improve her submission. This encourages additional submissions from this person, and other observers.

Mentoring was observed in FOSSASIA, Coala, and DuckDuckGo among others. *"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). All Coala community

respondents stated that they mentor contributors (100%). In other communities, the corresponding fraction was 63.21%.

> Trust, the contributor-maintainer relationship, and mentoring are norms exercised in PR evaluation.

3.4 RQ2.3: Product Vision in Evaluation of Pull Requests

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). In our survey, 65.24% of the maintainers agree or strongly agree that they would reject a PR that does not fit within the roadmap set by the community.

> The changes introduced in a PR must adhere to the community roadmap for its products, in order to increase chances of acceptance.

3.5 Governance Styles, Engineering Principles, and Social Norms

To examine the interplay of governance styles, software engineering principles and social norms, we computed Pearson's correlation coefficient for the answers to all pairs of questions related to governance styles (v_{27}-v_{33}), software engineering principles (v_{47}-v_{53}), and social norms (v_{54}-v_{56}). This coefficient allows us to determine whether there is a linear correlation between the questions. Pearson's coefficient is a value from -1 to 1. Values close to 1 or -1 indicate a positive or a negative linear correlation, respectively.

Figure 8 illustrates the correlations. Whiter cells indicate absence of correlation (0), dark brown or blue cells show positive (1) or negative (-1) correlation, respectively. We arranged the figure into several sections corresponding to the group of variables of interest. The middle-left square shows the correlation between governance styles and software engineering principles, the bottom-left conveys governance styles and social norms, and the center-bottom indicates the correlation of software engineering principles and social norms.

Interplay Between Governance Styles and Software Engineering Principles: Governance styles do not favor any particular software engineering principle, as indicated by the low correlation values of the middle-left section in Fig. 8. It appears that the principles are universal, applicable to all FOSS communities in our sample. All communities are committed to high engineering standards and quality irrespective of their norms and values.

Interplay of Social Norms and Software Engineering Principles: Social norms do not appear to favor any particular software engineering principle either; see the low values in the center-bottom section. This is expected, given the governance styles are clusters of these social norms. This also asserts the parallel between the governance styles and social norms.

Interplay of Governance Styles and Social Norms: We evaluated the correlations between the social norms underlining the protective style and the norms forming all styles. The dark brown cells connecting (v_{27}-v_{30}) and (v_{54}-v_{56}) suggest a relatively strong inclination of the protective style towards norms such as trust, relationships and reliability. On the other hand, the cells connecting (v_{31}-v_{33}) and (v_{54}-v_{56}) take on a blue coloration. This implies that the equitable and lenient styles reject these norms (i.e., trust, relationships and reliability). This conclusion cements our claims that the protective style adheres to norms fundamentally different to the ones preferred by the equitable and lenient styles.

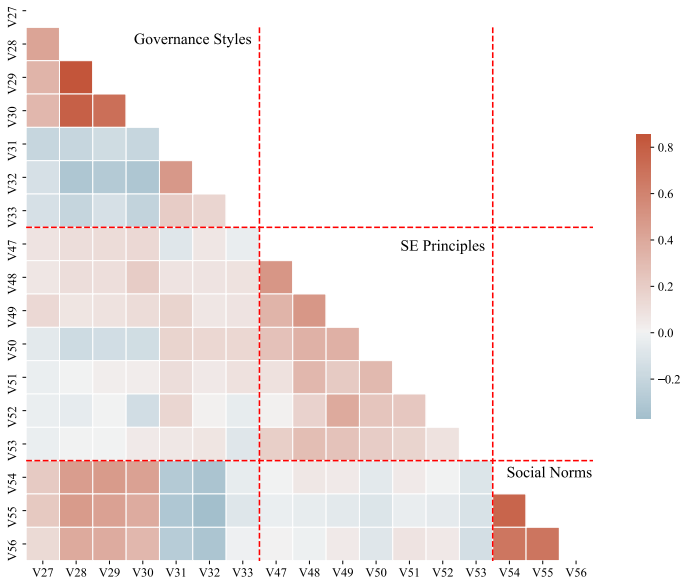


Fig. 8: Heatmap of correlations between questions related to governance style, software engineering principles and social norms. Darker colors indicate higher correlation.

4 Discussion

Governance relate to the social interaction and decision-making among the actors involved in a collective process that creates, reinforces, or reproduces social norms and organized groups [23]. Governance can be undertaken in any social system (e.g., government, institutions, family and informal organizations) [23]. The pull request governance styles have a reason to exist. 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.

4.1 Protective

The protective governance style may create a *"clique"* culture difficult to access for newcomers. Newcomers may feel less important than the established 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.

Still, the protective style helps to maintain tighter control over its code. For example, in a community for which an influx of newcomers is unimportant, or when it is high anyways. Tsay et al. write that well-established and mature projects are more conservative in accepting pull requests [24]. The Linux community is indeed a successful and mature project. The kernel code is a, so called, "closed," highly optimized and integrated platform that requires heavy-weight processes to evolve. This justifies a "centralized" governance, where patches have to pass thorough reviews along

the maintainer hierarchy [25]. Tight control is particularly beneficial if a project develops a highly technical system, with a high barrier of entry, and high risk of critical problems.

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 [26]–[28].

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 [29]. Also, maintainers interact more politely in discussions with core members than with new submitters [29]. 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 [24], [30].

Some of the behavior, e.g., contributor's reliability, underlying the protective style could be a risk mitigation for potential abandoned contributions. Li et al. find that contributors are abandoned for personal reasons, e.g., lack of time (36.6% of the study's respondents) and lack of interest (22.3% of the study's respondents) [31]. Abandoned contributions impact the evolution process of the community product, e.g., cluttered PRs and misspent review effort, and strains the maintainers efforts [31]. The Linux community is one of the oldest FOSS communities; this protective behavior could be a reaction to unpredictable contributors' commitment to the community.

The unpredictability of contributors also impacts the efficiency of the contribution process. Li et al. suggest that behaviors, such as not checking for prior work and not claiming a task before starting it, lead to duplicate pull requests, further increasing the pressure on maintainers [32]. Still, social factors remain relevant when maintainers decide which PR to accept when duplicates occur. Li et al. suggest explain that maintainers may prefer evaluating either the first to be submitted or the most "active" author.

4.2 Equitable

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

The equitable style is attractive for experienced developers who understand and incorporate advanced software engineering principles into their PRs. However, it does pose an entry barrier for newcomers. Steinmacher et al. list the need for orientation and technical hurdles as one of barriers for new contributors [33], [34]. Communities which prefer this style of governance should communicate their PR evaluation principles clearly through documentation and other channels to help the contributors to comply.

4.3 Lenient

Mentoring contributors is a key part of the lenient governance style. This style may help 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 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 [29]. After an initial period of 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 [35].

Attracting newcomers to communities is a major challenge. Fear of rejection that may harm reputation hinders some from contributing [2]. 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 [2], [36]. Berger et al. define variability encouragement as an open attitude to contributions from a broader ecosystem [25], 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 [37]. To some extent, the time required is compensated by attracting newcomers more easily.

4.4 Implications

Individuals' behavior is strongly influenced by the social group transmitted norms and values [38]. Social norms are adhered to by the group reinforcement; individuals also receive group approvals and rewards, when they follow them and punishment when they violate them [38]. It would be imprudent to suggest that a particular governance style with its underlying norms is superior or acceptable but not the others. One particular governance style cannot be universally prescribed to all FOSS communities. The underlying norms of these governance styles have reasons to exist. However, we can point out the costs of these governance styles on contributors participation and this may influence communities leaders to promote different norms for the betterment of their community. Leaders (e.g., senior maintainers, community leaders, and founders) can harness the power of social norms to enhance their groups behaviors to deliver the best possible outcomes and experience for the group's members [39].

Participants motivation to join FOSS community literature suggests that the underlying drivers are either intrinsic or extrinsic [40]–[42]. Von Krogh et al. [43] explain the intrinsic

motives are ideology, altruism, kinship amity, enjoyment and fun. Hars and Ou [41] further suggest community identification being another intrinsic motive. The extrinsic category hosts these sub-categories: reputation, gift economy, learning, own-use value, career and pay [43]. Although this literature is almost 20 years old, recent work [44] on the topic suggests that some of these motives "stood the test of time." They even suggest that the social aspect of participation such as kindship have become more prominent [44].

The protective governance style may alienate participants who are driven by the desire to learn, kinship and community identification. From this perspective, it is surprising that the Linux Kernel community is so large (15,600 contributors). Hertel and Niedner [45] explain this by identification with the prestigious community of the Linux kernel being the main motivation factor to contribute. Still, this governance style is at odds with the fundamental values of open source movement promoted by early pioneers (e.g., Richard Stallman and Eric Raymond). Communities adopting this style distance themselves from the true values of open source movement. Stewart and Gosain [46] explain that helping and cooperation are at heart of open source ideology. The cost of this style is encouraging a drift from the authentic values of open source movement and drive away those with a genuine desire to learn, motivated by the non-economical behavior like kinship, enjoyment and giving.

The equitable style was described by our participants as being "glacial." It attempts to decouple the contribution from the individual under the assumption that PRs submission and evaluation is purely a technical process. This may not be entirely in-line with the motivation of those who want to contribute to learn, identify with the community, kinship and enjoyment. Its transactional nature is at odds with expectation to help and cooperate. The cost might be alienating those who come to the community with limited experience in software development, especially applying advanced software engineering practices such as modularity of the code and submitting high quality code.

The lenient style seems to be strategically tuned toward attracting new contributors to ensure ongoing influx of contributions in order to survive as a community. However, it would be idealistic to suggest that this style is a model for FOSS communities. Mentoring generates a cost for maintainers and mentors. Also, not every contributor has the capability to bring his code to the desired quality. Less experienced contributors and newcomers should be directed to start working on less complex and easier issues. Communities considering this style may go further and propose a learning path to contribute for newcomers and less experienced community members. We observed some of these practices already taking place in communities like Coala and FOSSASIA. FOSSASIA suggests contributions for newcomers by labeling issues "newbie-friendly." Similarly, the Coala community labels the level of issues' difficulty from low to high.

Irrespective of contributors' motivation to join the community, community leaders should leverage the participation to sustain their communities, invest in retaining contributors for prolonged participation. Qiu et al. investigated the relevance of building social capital to prolonged participation amongst GitHub contributors [47]. Social capital is the set of advantages individuals gain from their social networks [48]. Qiu et al. suggest that social capital influences prolonged participation in open source communities. Individual gains, such as exposure to new technologies and ideas, contribute to building the social capital and consequently results in long term participation [47]. Communities leaders should encourage trade-offs between the cost of collective social conducts and ensuring ongoing and prolonged participation for sustainability.

As noted previously, social norms, are part of the social fabric in all cultures and social groups. Even though particular norms vary greatly, people use group norms to justify behavior, to provide order and predictability in social organizations and to make sense of and understanding of each other's actions. Communities leaders can influence the norms to minimize the cost on participation, nurture the non-economical behaviors and align their communities with the authentic values of open source. Below, we summarize the implications discussed in this section.

> Community leaders should take these costs (listed below) and weigh them in the context of their particular community. What is more effective depends greatly on the circumstances of a community, its history, the demographics of contributors and its need to remain sustainable.
>
> 1) **Protective:**
>
> 1.1. This style is not aligned with the authentic values of open source movement. This may create tension between the community and those who join because they are driven by these values.
>
> 1.2. This clash of values and social norms may drive away those with a genuine desire to learn, motivated by the non-economical behavior like kinship, enjoyment and giving.
>
> 2) **Equitable**
>
> 2.1. Motives such as cooperation, learning and enjoyment are not compatible with a transactional and rigid interactions.
>
> 2.2. This tension may frustrate contributors who want a more supportive community.
>
> 3) **Lenient:**
>
> 3.1. Maintainers and mentors may feel or become drained by the burden of mentoring.

5 Limitations & Threats to Validity

Trustworthiness (Phase I and II)

We used a combination of directly emailing potential participants, and announcements in the selected communities forums and mailing lists. This method may have biased our sample towards contributors who actively check their respective communities forum and mailing lists announcements. Especially, the Linux interviewees were recruited through our contacts in the community (convenience sampling). Convenience sampling is criticized as potentially not representative of the targeted population. However, both the qualitative and quantitative data from other communities clearly indicates that there is a clear trend towards protective behavior with Linux in the limit (Fig. 6b).

While we cannot be sure if the positioning of Linux is not overdriven, the existence of the trend is unquestionable.

There is always a risk that the survey questions could not be understood by the respondents. We mitigated this risk by ensuring that the statement of our questions are well elaborated and clearly stated. Self-determination is a standard issue in interview studies and surveys. It is possible that some participants answer the questions conveying their intentions (self-interest when answering survey questions) rather than what happens in reality. This is partly mitigated by the sample size in the quantitative analysis, and using comparative, rather than absolute analysis.

Internal and Construct Validity (Phase II)

Our survey measures subjective constructs. Although, we did our best to write clear and comprehensive statements in the survey, a risk of misalignment in understanding the questions by researchers and respondents remains. To strengthen the statistical conclusions we informed the Bayesian model of the Likert scale with a joint prior for all communities (which means that the interpretation of cut off points between the scale answers is consistent across all communities). The analysis scripts and anonymized quantitative data are available for scrutiny at our GitHub repository.

In Bayesian data analysis, the choice of ROPE is domain specific and is typically used to provide insights on the robustness of the results. Larger the ROPE, more robust the conclusion is. Following a common practice, we reported the largest ROPEs for which the hypotheses hold to make explicit the robustness of their validity. For H3, we opted for a ROPE of $(-0.2, 0.2)$. A ROPE of $(-0.1, 0.1)$ is conventionally accepted for effect size [49]—half of the size used here. For H1, we consider a ROPE of size 0.8. This ROPE constitutes 16% of the Likert scale values $(1-5)$, which, we think, gives sufficient robustness for the validity of this hypothesis.

External Validity (Phase II)

The findings of this study are confined to the selected communities. Conscious of this potential limitation, we selected 15 communities with different backgrounds and attributes. The statistical analysis of N=387 subjects in Phase II alleviates some concerns of the (naturally) small qualitative sample of Phase I.

All respondents from the DuckDuckGo community were contributors so they have not answered the maintainer oriented questions v_{27}–v_{33}. Consequently, we excluded DuckDuckGo from the statistical analysis. We thus cannot make any quantitative statements about it. Out of the 387 participants 19 did not report the community they belong to. Their responses were excluded as well.

Participation of maintainers from some communities was low. This may affect the accuracy of the results of the quantitative analysis. However, Bayesian data analysis is known to provide the most accurate results given the data, regardless of how small the data is [5]. The variance of the inferred distributions gives an indication of the certainty of the results. Furthermore, to reduce the impact of this issue, our model uses *shrinkage* (e.g., [5, Sect. 9.3]) which means that the responses of individual communities are informed by the answers of the whole dataset of FOSS communities. This feature increases the reliability of the results for communities with lower response rates.

A complete agreement between the qualitative and quantitative results is not always a possible outcome in mixed-methods, especially when the subject of interest is social and behavioral. Data obtained using different methods is bound to show differences. We believe, a perfect agreement of both methods would be even suspicious, given that qualitative data is small, and participants were interviewed in a particular context. In this study, the quantitative component refines the qualitative results. The results of the survey shift the conclusion from a rigid interpretation of the identified governance styles, i.e. a taxonomy, to an archetypical styles of governance that are approximated by the communities. A FOSS community cannot be boxed into a single governance style, which could have been an impression created by the interviewees. The quantitative phase makes it clear that FOSS communities (and their members) form a continuum and an evolving social structure, where different forces coexist together (Fig. 6). This text has also been added to Section 5.

6 Related Work

Pull-request-based collaboration has attracted some attention recently [2], [27], [30], [36], [50]–[56]. Still, to our best knowledge, no prior work conceptualizes the different governance styles across FOSS communities.

Soares and coauthors [30] find that the chance of a merge is 32% lower for first time contributions, supporting our findings 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 [24]. 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 [24]. 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. [29] 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. [27] examine how interpersonal impressions influence evaluations of 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 personality. Arguments or rudeness in posting often are seen as indicators of uncooperativeness or arrogance [27]. Their study concurs and complements our findings. It confirms that social inferences are part of the PR evaluation process, while showing how GitHub social signals are leveraged to make inferences about contributors.

In FOSS, proper evaluation is seen as more important than addition of a feature. Developers prefer to postpone reviews rather than rush through them [57]. We observed similar attitude amongst our interviewees. They prefer investing great care and attention to detail rather than following a predefined checklist. This rigour coupled with the passion for the project lead to excellence in the evaluation process [21].

Gousios et al. [1] asked "what factors affect the decision and the time required to merge a pull request?" and "why are some pull requests not merged?" In response to the first research question, they found that the decision to merge is influenced by whether the pull request modifies recently modified code. The time required to merge is influenced by the developer's previous track record in contributing to the project, the size of the project, and the test coverage of the PR. The project's openness to external contributions is also important factor. In regards to unmerged PRs, they suggest that contributions are rejected because of concurrent modifications, the contributor not having identified the direction of the project correctly, and the project's process and quality requirements not identified or adhered to. Although some of these findings match our conclusions (mainly RQ2), the study focuses solely on the "merge" decision, which is the result end of the entire socially loaded process. In a broader and more holistic manner, we examined the evaluation process granularity and we unfolded some of its social and human aspects. Gousios et al. [1] assert that only 13% of rejections are related to technical reasons. However, it does not go further and explore what are the non-technical reasons.

Gousios et al. [36] investigated several research questions; but the most pertinent here is "what are the challenges of contributing?" They suggest that the challenges to contributing, from the contributor's perspective, are either related to "social aspects" or "code aspects." They cited that some of the social aspects are "responsiveness", "communication" and "fear of rejection." The "code aspects" are mainly "understanding the code base", the test suite is not always available and meeting the code quality expectations. For the social aspects, our work examines the collective actions of the community, while Gousios et al. [36] work suggest the individual social challenges faced by the contributors. Collective actions refer to the actions taken by a community as a group of people, acting based on a collective set of believes, norms and decisions. Our work suggest that what the community collectively does as a social system also influence the decision-making of the PR evaluation process.

Gousios et al. [2] sought to investigate several research questions. The most relevant question to our work states "how do integrators decide whether to accept a contribution?" This question relate to our RQ2. They identified a set of factors which the integrator use to assess contributions. They listed, code quality, documentation, roadmap fit and simplicity amongst others. Our work build on these findings and adds three additional principles, passing peer code review, code maintainability and avoiding technical debt. In addition, we show that these software engineering principles are almost universal to all the communities we studies, irrespective of their social conducts. The interplay with the technical merits of the contribution is decoupled from the collective group actions. The social collective actions influence the interactions and how the individual is treated but not necessary the technical merits of the contributions.

Dey and Mockus investigated the effect of the social and technical factors on PRs quality in the npm community [58]. They find that the technical and social factors are equally significant in the PR evaluation in npm. Although, the study does not add new technical factors compared to previous work, e.g., [2], [24], [29], [36], [59], it accentuates the importance of some social factors like "social proximity" and commitment or loyalty to the community. They also identify a collective behavior, the "leniency of a repository", which influences the PR acceptance [58]. This is consistent with our conclusions that 'social proximity", i.e., how strongly a relation between the contribution's author and the maintainer and the reliability of the contributor are important social determinants in the protective communities. Our work provides further interpretation of the repository level behavior discussed by Dey and Mockus, including archetypical governance styles other than leniency.

The shorter version of this work [3], investigates the topic using solely a qualitative method in five communities. It identifies a community collective behavior taking place in the evaluation process of pull requests, and formulates the three governance styles. This extended version adds the quantitative survey of 15 communities, and carefully combines the two analyses to achieve stronger and more informative results that are both more broader empirically, and more informative about a landscape of FOSS communities.

7 Conclusion

Governance of PRs evaluation process comprises of control believes and assessment rules undertaken by the FOSS communities we studied using a social system norms in order to organize the contribution process.

FOSS communities align their pull request governance to their needs and peculiarities. In some instances the PR governance evolves from one style to another, but it never compromises quality. What shifts is the willingness of these communities to support the person contributing, possibly in a reaction to a strategic need. For example, in the case of the Coala community, this strategic need is to sustain participation. In the Linux Kernel case, a strong influx of contributors combined with a high complexity of the software seem to justify prioritizing exclusivity measures like trust and relationship.

Regardless the style, this disciplinary aspect of FOSS communities contributes to achieving quality by mitigating the risks posed by the contributor and the contribution. Only, the second level of risk mitigation focuses on the technical quality of the PR. We hope that this extensive and nuanced analysis of the mechanism of PR evaluation across various FOSS communities can influence future projects on the socio-technical aspects of open source software engineering, decision making, community management, bias, and alike. Clearly, not all communities are the alike, and the method we used was effective in demonstrating that.

It remains an interesting question for future research, what other aspects of FOSS communities (and how) correlate or influence decision making in the PR process. These dimensions can be organizational, e.g., leadership style, or historical, e.g., well-established community versus a new or emerging community. Another consideration for future

work is whether the technologies, the engineering practices and tools used by FOSS communities, exert any influence on sustaining a particular style of governance.

Acknowledgments

Work supported by the EU's Horizon 2020 Research and Innovation program, grant No. 732287 ROSIN. We thank all the interviewees and the survey respondents.

References

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

[2] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: the contributor's perspective," in *38th International Conference on Software Engineering*, 2016.

[3] A. Alami, M. L. Cohn, and A. Wąsowski, "How do FOSS communities decide to accept pull requests?" in *Proceedings of the Evaluation and Assessment in Software Engineering*, 2020, pp. 220–229.

[4] J. W. Creswell and V. L. P. Clark, *Designing and conducting mixed methods research*. Sage publications, 2017.

[5] J. Kruschke, *Doing Bayesian Data Analysis*. Elsevier, 2015.

[6] C. A. Furia, R. Feldt, and R. Torkar, "Bayesian data analysis in empirical software engineering research," *IEEE Transactions on Software Engineering*, 2019.

[7] G. R. Gibbs, "Thematic coding and categorizing," *Analyzing qualitative data*, vol. 703, pp. 38–56, 2007.

[8] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011.

[9] C. Robson and K. McCartan, *Real world research*. John Wiley & Sons, 2016.

[10] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative data analysis: A methods sourcebook. 3rd*. Sage, 2014.

[11] B. Saunders, J. Sim, T. Kingstone, S. Baker, J. Waterfield, B. Bartlam, H. Burroughs, and C. Jinks, "Saturation in qualitative research: exploring its conceptualization and operationalization," *Quality & quantity*, vol. 52, no. 4, pp. 1893–1907, 2018.

[12] M. Q. Patton, *Qualitative evaluation and research methods: Integrating theory and practice*. Sage Publications, 2014.

[13] D. D. Heckathorn, "Respondent-driven sampling: a new approach to the study of hidden populations," *Social problems*, vol. 44, no. 2, pp. 174–199, 1997.

[14] P. Ghauri, K. Grønhaug, and R. Strange, *Research methods in business studies*. Cambridge University Press, 2020.

[15] P. L. Alreck, P. L. Alreck, R. B. Settle, and S. Robert, *The survey research handbook*. McGraw-Hill/Irwin, 1995.

[16] J. K. Kruschke and T. M. Liddell, "Bayesian data analysis for newcomers," *Psychonomic bulletin & review*, vol. 25, no. 1, 2018.

[17] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in python using PyMC3," *PeerJ Computer Science*, vol. 2, p. e55, 2016.

[18] D. Ford, M. Behroozi, A. Serebrenik, and C. Parnin, "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, 2019, pp. 51–60.

[19] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, 1994.

[20] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "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, 2015, pp. 50–60.

[21] A. Alami, M. L. Cohn, and A. Wąsowski, "Why does code review work for open source software communities?" in *The 41st International Conference on Software Engineering*, 2019.

[22] E. L. Paluck and L. Ball, "Social norms marketing to reduce gender based violence," *IRC Policy Briefcase*, 2010.

[23] M. Bevir, *Governance: A very short introduction*. OUP Oxford, 2012.

[24] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *The 36th International Conference on Software engineering*. ACM, 2014.

[25] T. Berger, R.-H. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wąsowski, and S. She, "Variability mechanisms in software ecosystems," *Information and Software Techn.*, vol. 56, no. 11, 2014.

[26] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "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, 2012, pp. 1277–1286.

[27] J. Marlow, L. Dabbish, and J. Herbsleb, "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, 2013, pp. 117–128.

[28] P. B. De Laat, "How can contributors to open-source communities be trusted? on the assumption, inference, and substitution of trust," *Ethics and information technology*, vol. 12, no. 4, pp. 327–341, 2010.

[29] J. Tsay, L. Dabbish, and J. Herbsleb, "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, 2014, pp. 144–154.

[30] D. M. Soares, M. L. de Lima Júnior, L. Murta, and A. Plastino, "Acceptance factors of pull requests in open-source projects," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1541–1546.

[31] Z. Li, Y. Yu, T. Wang, G. Yin, S. Li, and H. Wang, "Are you still working on this an empirical study on pull request abandonment," *IEEE Transactions on Software Engineering*, 2021.

[32] Z. Li, Y. Yu, M. Zhou, T. Wang, G. Yin, L. Lan, and H. Wang, "Redundancy, context, and preference: An empirical study of duplicate pull requests in oss projects," *IEEE Transactions on Software Engineering*, 2020.

[33] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *The 18th ACM conference on Computer supported cooperative work & social computing*. ACM, 2015.

[34] I. Steinmacher, G. Pinto, I. S. Wiese, and M. A. Gerosa, "Almost there: A study on quasi-contributors in open-source software projects," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 256–266.

[35] N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, pp. 323–368, 2005.

[36] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen, "Work practices and challenges in pull-based development: the integrator's perspective," in *The 37th International Conference on Software Engineering-Volume 1*, 2015.

[37] S. E. Sim and R. C. Holt, "The ramp-up problem in software projects: A case study of how software immigrants naturalize," in *The 20th International Conference on Software Engineering*, 1998.

[38] S. Gavrilets and P. J. Richerson, "Collective action and the evolution of social norm internalization," *Proceedings of the National Academy of Sciences*, vol. 114, no. 23, pp. 6068–6073, 2017.

[39] S. Albert and D. A. Whetten, "Organizational identity." *Research in organizational behavior*, 1985.

[40] J. Bitzer, W. Schrettl, and P. J. H. Schröder, "Intrinsic motivation in open source software development," *Journal of Comparative Economics*, vol. 35, no. 1, 2007.

[41] S. O. Alexander Hars, "Working for free? motivations for participating in open-source projects," *International journal of electronic commerce*, vol. 6, no. 3, pp. 25–39, 2002.

[42] K. R. Lakhani and R. G. Wolf, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," *Open Source Software Projects (September 2003)*, 2003.

[43] G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, "Carrots and rainbows: motivation and social practice in open source software development." *MIS quarterly*, vol. 36, no. 2, 2012.

[44] M. Gerosa, I. Wiese, B. Trinkenreich, G. Link, G. Robles, C. Treude, I. Steinmacher, and A. Sarma, "The shifting sands of motivation: Revisiting what drives contributors in open source," in *ICSE'21*, 2021.

[45] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel," *Research policy*, vol. 32, no. 7, 2003.

[46] K. Stewart and S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *Mis Quarterly*, 2006.

[47] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu, "Going farther together: The impact of social capital on sustained

participation in open source," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019.
[48] P. S. Adler and S.-W. Kwon, "Social capital: Prospects for a new concept," *Academy of management review*, vol. 27, no. 1, pp. 17–40, 2002.
[49] J. K. Kruschke, "Rejecting or accepting parameter values in bayesian estimation," *Advances in Methods and Practices in Psychological Science*, vol. 1, no. 2, pp. 270–280, 2018.
[50] J. Zhu, M. Zhou, and A. Mockus, "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, 2016, pp. 871–882.
[51] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: determinants of pull request evaluation latency on github," in *The 12th Working Conference on Mining Software Repositories*, 2015.
[52] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration," in *21st Asia-Pacific Software Engineering Conf.*, 2014.
[53] M. M. Rahman and C. K. Roy, "An insight into the pull requests of github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 364–367.
[54] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, "Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development," *Information and Software Technology*, vol. 84, pp. 48–62, 2017.
[55] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Techn.*, vol. 74, 2016.
[56] D. M. Soares, M. L. de Lima Júnior, L. Murta, and A. Plastino, "Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates," in *14th International Conference on Machine Learning and Applications*, 2015.
[57] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 541–550.
[58] T. Dey and A. Mockus, "Effect of technical and social factors on pull request quality for the npm ecosystem," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.
[59] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 67–76.

Adam Alami is a postdoctoral researcher with the IT University of Copenhagen, Denmark. He has broad experience in information technology practices. His career began in software development, before progressing to include business analysis and project management. Involvement in major IT transformation projects has for twenty years been the mainstay of his work. His chosen fields of research fit within the broad topic of cooperative, social, and human aspects of software engineering. In striving to better understand how software development teams might achieve the quality of their output, he dissects process and ceremony to identify behaviors, norms, and traditions that are fundamental to their quest for quality. He holds a PhD degree in Computer Science from the IT University of Copenhagen, Denmark (2020), a Master degree on Computer Science from the University of Technology (UTS) (2006), Sydney, and a Bachelor degree in Software Engineering from the Université du Québec à Montréal (UQÀM) (1998),

Raúl Pardo a Postdoctoral researcher at the IT University of Copenhagen. His research is focused on developing rigorous techniques to design, analyze and build software to protect online privacy. His interests lie at the intersection of formal methods, online privacy and computer security. He has done research on privacy for social networks, Internet of Things (IoT) and data analytics. Within these topics, he is working on privacy risk analysis, formal verification of privacy legal requirements, probabilistic programming, and Bayesian data analysis. He holds a PhD degree from Chalmers University of Technology.

Marisa Leavitt Cohn is an Associate Professor in the Business IT Department at the IT University of Copenhagen. She is member of the Technologies in Practice Research group and co-director of the ETHOS laboratory, a feminist methods lab fostering critical approaches to digital methods. Cohn combines approaches from Anthropology, Science and Technology Studies, and Human Computer Interaction to the study of infrastructure and software. Her work examines the temporalities of technological change in long-lived software systems, looking at issues related to legacy, maintenance, and obsolescence. Cohn holds a PhD in Information and Computer Sciences from the University of California Irvine Department of Informatics (2013) and a BA in Anthropology from Barnard College (2002).

Andrzej Wąsowski is Professor of Software Engineering at the IT University of Copenhagen. He has also worked at Aalborg University in Denmark, and as visiting professor at INRIA Rennes and University of Waterloo, Ontario. His interests are in software quality, reliability, and safety in high-stake high-value software projects. This includes semantic foundations and tool support for model-driven development, program analysis tools, testing tools and methods, as well as processes for improving and maintain quality in software projects. Many of his projects involve commercial or open-source partners, primarily in the domain of robotics and safety-critical embedded systems. Recently he coordinates the Marie-Curie training network on Reliable AI for Marine Robotics (REMARO). Wąsowski holds a PhD degree from the IT University of Copenhagen, Denmark (2005) and a MSC Eng degree from the Warsaw University of Technology, Poland (2000).

APPENDIX

TABLE 9: Survey questions relevant to RQ1 (governance styles) and RQ2 (social norms and software engineering principles). Each criterion had a 5-point Likert scale: i) strongly agree, agree, neutral, disagree and strongly disagree for governance styles; and ii) very important, important, neutral, not important, not important at all for software engineering principles and social norms. Governance style questions were only answered by maintainers. Software engineering principles and social norms questions were answered by maintainers and contributors.

	ID	Question	Rationale for inclusion
Governance Styles	v_{27}	*In general I say no to most pull requests (PR)/patches. The contributor has to be persistent and prove that the PR/patch worth evaluating.*	This question was included to test the widespread of the inclusivity behavior we observed in the protective style, which is unique to the Linux Kernel community. Inclusivity is a code that emerged in the qualitative analysis and belongs to the Protective Governance style theme.
	v_{28}	*I don't consider a pull request/patch, unless I trust the contributor.*	Trust emerged as a code under the Protective Style theme. This question was included to measure the presence of this behavior within the protective communities.
	v_{29}	*I don't consider a pull request/patch, unless the contributor is reliable.*	Reliability is another code characteristic of the protective style. This code emerged in our qualitative data analysis, we included this question to evaluate the widespread of this behavior within the protective communities.
	v_{30}	*I don't consider a pull request/patch, unless I have a strong relationship with the contributor.*	The maintainer-contributor relationship is another qualitative data related code. This question represent this code and intended to test its relevance in the protective communities.
	v_{31}	*I assess every pull request/patch in the same manner irrespective of the contributor.*	This question is to evaluate the claim of "fair conduct towards Each other's", which we observed in our coding of the interviews of the equitable communities.
	v_{32}	*I assess pull requests/patches purely on technical grounds.*	Technically grounded decision is another code under the equitable governance style theme. This question is to evaluate its presence in the equitable communities.
	v_{33}	*I never say no to a pull request/patch. If the quality of the PR/patch is not mergeable, then I mentor the contributor to elevate his/her PR/patch to a mergeable state.*	Helping each other's and avoiding rejection are two codes characteristic of the Lenient style. This question evaluate these two codes.
Software Engineering Principles	v_{47}	*Code Quality (my subjective assessment of quality as a reviewer)*	This is an evaluation of the adherence to the software engineering criteria we identified in our qualitative analysis. It measures the importance of these evaluation principles to the PR reviewer among our survey participants.
	v_{48}	*Code modularity*	
	v_{49}	*Maintainability (the ease with which a program code can be changed in order to: correct defects or enhance the code)*	
	v_{50}	*Comply with the community best practices of programming languages*	
	v_{51}	*Contain documentation*	
	v_{52}	*Avoid technical debt*	
	v_{53}	*Pass tests*	
Social norms	v_{54}	*Trust is more important than the contribution of the PR/patch during the assessment of PRs/patches.*	These social norms were identified in our qualitative analysis. This question evaluate the importance of these norms to the participants in our survey.
	v_{55}	*Past reliability of the contributor is more important than the contribution of the PR/patch during the assessment of PRs/patches.*	
	v_{56}	*A good contributor/maintainer relationship is more important than the contribution of the PR/patch during the assessment of PRs/patches.*	