Assessing the Characteristics of FOSS Contributions in Network Automation Projects

John Anderson
Clemson University
jda3@clemson.edu

Igor Steinmacher
Northern Arizona University
Igor.Steinmacher@nau.edu

Paige Rodeghero
Clemson University
prodegh@clemson.edu

***Abstract*—Network Automation seeks to integrate software solutions that aid in the management and maintenance of modern networks. In industry, large organizations see dedicated software engineering resources within a networking team. However, in the broader industry, it is more common to see traditional network engineers working on network automation. With the growth of Free and Open Source Software (FOSS), network automation software solutions also started to adhere to this development model. However, as it is known from the literature, onboarding to FOSS projects is not a trivial task and may be more challenging for people without a software development background. In this paper, we study network automation FOSS projects, which are seeing a large number of new contributors who do not have traditional software engineering skills. We analyze a set of data collected from pull requests and issues collected from 81 GitHub projects (71 network automation projects, and 10 top-projects from other domains), to identify the characteristics that are specific to first-time project contributors in the network automation domain. Our results show that pull requests in the Network Automation domain differ from those in the Top-10 set and the existing literature. At the same time that Network Automation projects are more inclusive (rejection rate: 12% vs. 28% on Top-10), the pull request latency is longer in this specific domain, especially for first-timers.**

***Index Terms*—software engineering, onboarding, networking automation, Open Source**

I. Introduction

Network Automation is a recently expanding field within the area of computer networking that seeks to integrate software solutions to aid the management and maintenance of modern networks. Network operators are increasingly mixing traditional network engineering with software engineering fundamentals and skill sets to do this.

In larger organizations, it is not uncommon to see dedicated software engineering resources within a networking team. In the broader industry, however, it is much more common to see these once traditional network engineers learning how to program and picking up software engineering experience. Also of note in this transition is the networking industry's growing reliance on Free and Open Source Software (FOSS) solutions, a trend that is no doubt noticed in other disciplines [1].

Naturally, this combination is ripe for FOSS project creation and contribution. One of the pillars of FOSS is community collaboration, and this means FOSS projects often see participation from a variety of software engineering skill levels [2]. Given the circumstances surrounding network automation, we desire to understand specific contribution metrics that will allow us to make statements about the current state of the network automation FOSS landscape. As it is well known that first-time contributors often face challenges to onboard [3], it is important to understand how these contributors interact with the projects in different domains. In this context, the long-term goal of this work is to aid the industry in onboarding and provide guidelines for best practices. More specifically, the research objective of this paper is to understand better how new contributors interact with projects in network automation FOSS projects via issue tracker and pull requests. Therefore, we propose two research questions (RQs):

- RQ_1 How do pull requests submitted by first-time contributors in network automation projects compare to those by experienced members and to those in other domains?
- RQ_2 How do issues and pull requests resulting from issues submitted by first-time contributors compare to those by experienced members in network automation and to those in other domains?

The goal of the first research question is to identify if network automation projects are seeing quality concerns in the contributions made by first-time contributors which may determine the fate of the contribution. While many different factors can cause this, if the relationship of the characteristic and the result of the contribution is more significant when compared to the broader industry, it may indicate a need for more attention to the onboarding practices within network automation projects. To analyze the pull requests, we rely on metrics used in related literature [4]–[6]: number of commits, pull request status, code churn, number of comments, and latency (time to close the pull request).

The goal of the second research question is to explore issue collaboration with first-time contributors. GitHub issues are often used as a forum to submit feature requests and bug reports. We analyzed the data to explore a possible correlation between first-time contributors and the fate of their feature requests and bug reports. Should a relationship be found, it could be an indicator that new collaborators misunderstand the intended use of these issues. To analyze the issues, we used the following metrics: number of comments, latency, and link between issues and pull requests.

In this paper, we conduct a statistical study of data surrounding FOSS project collaboration in the domain of network au-

tomation. We identified 71 network automation-related FOSS projects on GitHub and, using the GitHub Archive [7], we collected data from each of these projects for the entire year of 2019. This data set includes event details from pull requests, issue comments, commit pushes, repository popularity metrics, and more. We compared this data to another subset that contains the same events but for a list of 10 repositories called out by GitHub as the top repositories for 2019 [8]. We found that the characteristics of pull requests found in Network Automation projects are quite different from those found in the Top-10 GitHub projects. Pull requests in Network Automation are usually larger (in terms of commits, additions, and files touched) and foster less discussion when compared to the Top-10 pull requests. Furthermore, the pull requests in Network Automation take longer to be closed. In terms of issues, we found similar behavior: they take longer to be closed and foster less discussion. On the other hand, the Network Automation community looks more welcoming with a higher pull request acceptance rate (88% vs. 72% in the Top-10). We found that Network Automation projects do not follow the practices well-known in mature FOSS projects, showing that knowledge related to FOSS practices and software development and maintenance are no longer optional.

II. Background

The field of network automation represents a unique subsection of the broader industry, one that is not only interdisciplinary but also one that only relatively recently came into existence in the formal sense. In terms of general software engineering, this case study can provide an example of an industry that is turning to more programmatic solutions. With that transition comes inherent problems related to software engineering, which deserve attention.

Network operators, driven by the ever-changing business landscape, desire to run their networks with increased efficiency and reduced costs. Many solutions have arisen like Software Defined Networking (SDN) [9], Network Function Virtualization (NFV) [10] that serve to implement new technologies to solve networking problems. While these solutions have their place in the industry, operators are more often than not tied to their existing network deployments and unable to re-architect them to meet their automation goals. Operators are looking for ways to leverage their existing infrastructure in more efficient manners. In terms of automation, this usually begins by examining business processes and identifying areas in which manual, human-driven workflows can be automated.

In smaller-scale situations such as small to medium enterprises, these solutions are typically not full-blown custom software projects, but instead integration efforts between existing products, often FOSS projects. Sometimes this requires code to be written to "glue" systems together. Larger enterprises tend to run networks with more scale and often see those networks as providing definitive business value. They are typically looking for custom solutions. These efforts may be to build a system from the ground up, but often businesses realize the importance of making use of FOSS, and thus this becomes an issue of upstream contribution to filling gaps. Sometimes these businesses have the software engineering resources in house to take on these projects, but this tends to be the exception, not the rule, which prompts our research efforts. We are interested in the segment of the market that desires these sorts of implementations but does not have additional resources. This situation often leads to network engineering and operations personal to pick up this slack.

III. Related Work

In this section, we discuss the related work on pull request analyses, issue analyses, data quality, and FOSS project onboarding. We discuss related work to compare and contrast our approach while reminding the reader a portion of our contribution is the application of this study to one particular, growing field identifiable within the broader GitHub ecosystem.

The driving force of FOSS is open community collaboration. Therefore, the process to onboard onto a project is a complex process composed of different stages and a set of forces that push contributors towards or away from the project [11]. As shown before [3], potential new contributors face several barriers while attempting to contribute for the first time to FOSS projects. The new collaboration model brought by GitHub last decade facilitates onboarding by providing users with a single contribution process, known as the pull-based model [6], [12], which allowed it to become the *de facto* watering hole for FOSS projects [13]. The growth of GitHub and FOSS, increased the attention given to research related onboarding of newcomers onboarding [14]–[16], how outsiders are received in company-owned projects [17], how the developers' emotions and sentiments when interacting [18], [19], and how inclusive the projects are [20], [21].

The two major backbone features of the pull-based model on GitHub are pull requests and issues [22]. The pull-based model followed by GitHub [6], [12] is based on the concept of pull requests. In this model, the work is distributed between a team, which submits the pull requests to be considered for merging, and a core team, which oversees the merging process, provides feedback, conducts tests, requests changes, and accepts the contributions [6].

Given the relative openness of public projects available on GitHub, several studies analyzed the interaction of developers in pull requests, more specifically focusing on pull request acceptance/rejection. Gousios *et al.* [6], for example, found that 53% of rejected pull requests are rejected for reasons related to the distributed nature of pull-based model, and only 13% of the pull-requests are rejected for technical reasons. They report that the decision to merge a pull request is mainly influenced by whether the pull-request includes recently modified code and that the time to merge is influenced by the developer's history, the project's size, test coverage, and the openness to external contributions. Size and maturity of the project are also reported as influential by Rahman et al. [23]. Padhye *et al.* [24] complement this list, reporting that bug fixes are more likely to be merged than feature enhancements, while Hellendoorn *et al.* [25] found that code style is an important

aspect. From a less technical perspective, Tsay *et al.* [26] reported that the social connection between the submitter and project manager matters, and that highly discussed pull requests are less likely to be accepted. Tao *et al.* [27] also found that bad timing of patch submission and a lack of communication with team members can lead to rejection. More recently, Alami *et al.* [28] discussed that the governance model needs to be considered since the communities are different. They classify the pull request governance model in protective, equitable, and lenient. They also summarize the principles used to evaluate the acceptance of pull requests that are classified as software engineering practices, requirements, social norms, and strategic vision for the product. While these studies analyzed the reasons behind contribution rejection, we complement this literature by expanding them to target first-time contributions in a specific domain.

In fact, the onboarding of first-timers is (marginally) considered in the studies that involve analysis of pull requests. By interviewing maintainers and contributors, German *et al.* [29] found that first-time contributors may be treated unfairly and that maintainers are socially biased in favor of known people. Soares *et al*. [30] also found that pull requests made by an outsider (the first pull request by a developer) influence the acceptance decision. Finally, while analyzing quasi-contributors (people who attempted to contribute but did not have their contribution accepted), Steinmacher *et al.* [31] found that the main reasons for pull request rejection from the quasi-contributors' perspective were "superseded/duplicated pull-request" and "mismatch between developer's and team's vision/opinion." Our work complements these by looking specifically for first-timers in Network Automation projects, compared with other projects.

As mentioned before, another main component of GitHub is the issues. GitHub "issues" are a place to foster discussion around new software features and bug reports [32], and often a place to ask questions to the community [33]. In terms of a typical flow within a GitHub repository, issues are used in intake bug reports and feature requests, allowing for a lengthy discussion about these ideas before and during implementation. Once implementation is prepared, the relevance of the issue is migrated and linked to one or more pull requests. The existing literature analyzed the use of issues on GitHub in a more exploratory sense. Bissyandé *et al.* [34] investigated the adoption of issue trackers on GitHub and found that issues are more thoroughly used by large projects, there is a small correlation between the numbers of issue reporters and the time-to-close delays, and that bugs and features are equally reported. Kikas *et al.* [35] showed that the number of opened issues is stable over time while pending issues are constantly growing. Still, by analyzing labels assigned to issues, Cabot *et al.* [36] found that the use of labels positively impacts the issue evolution by facilitating collaboration and leading to an increase in the number of issues solved. Although many studies explore issues, in this paper, we take the perspective of first-time contributors, to understand how they collaborate with network automation communities using the issue tracker.

IV. Research Methodology

In this paper, we seek to shed light on the current state of network automation FOSS development, concerning new developers and skillsets. To replicate our study, please see Section IV-A. We collected data from the GitHub Archive project [7], which continuously collects and catalogs events from all of GitHub. Due to the scope and nature of this data set, it has been used many times before in the domain of software repository mining [37]–[39] .

We identified 71 GitHub repositories related to the domain of network automation, by applying the following criteria:

- we identified activity in 2019,
- relevance to the field, judged by community references (e.g., Awesome Network Automation list[1]), and the expertise of one of the authors (who works in the field), and
- being an actual software project and not merely a source of other reference material or knowledge (through a manual analysis).

With these 71 repositories identified, we executed a query on the BigQuery platform[2] to return all data related to those 71 repositories for the entirety of the year 2019. We created a local copy with the events for these projects, containing the tuple of event, a json blob with the event data itself, date of creation, organization and repository name, and the actor responsible for the event.

TABLE I
Top-10 Ranked GitHub Repositories

ID	GitHub project
1	https://github.com/aspnet/AspNetCore
2	https://github.com/flutter/flutter
3	https://github.com/MicrosoftDocs/vsts-docs
4	https://github.com/istio/istio
5	https://github.com/aws-amplify/amplify-js
6	https://github.com/helm/charts
7	https://github.com/ValveSoftware/Proton
8	https://github.com/gatsbyjs/gatsby
9	https://github.com/storybookjs/storybook
10	https://github.com/cypress-io/cypress

Having identified the subset of repositories by which to study our niche domain, we wanted to have another data set to compare to represent the broader software engineering landscape better. For this, we chose to use GitHub's list of Top-10 trending repositories, according to the Octoverse (see Table I) [8]. These projects represent trending repositories by the number of new contributors. This factor makes these specific projects a fair selection to compare with the network automation projects since our analysis is based on contribution quality characteristics. It is important to note this Top-10 list also comes from the year 2019, which ensures that we are comparing events and data over a consistent timeline. The data for these repositories comes from GitHub Archive.

[1]https://github.com/networktocode/awesome-network-automation
[2]https://cloud.google.com/bigquery

With this data set, we perform our analyses by collecting data through a series of queries in pursuit of our research questions. To identify first-time contributors, we used the classification of the contributors provided by GitHub. In this classification, project contributors are broken down into five categories: (1) Collaborator, (2) Member, (3) Owner, (4) Contributor, and (5) None. These categories are referred to as the pull request or issue author's association with the project. Although we know that issues and pull requests do not hold the temporal nature of these categories, GitHub Archive does keep this information given the incremental nature of the collection.

For both issues and pull requests, the category of None represents the authors that never successfully had a commit merged in the default branch of the project. We considered those contributors in this category as our *first time contributors*. The Owner category is the person or persons who make up the lead maintainers role of the project. The Member and Collaborator categories are roles granted to persons, which give them some degree of permissions within the repository. Generally speaking, within the realm of FOSS, these people have a track record of existing contributions to the project to be afforded these roles. Finally, the Contributor role makes up anyone else that has successfully committed to the default branch and does not fall into one of the other categories. Some of our analysis aggregates all roles other than None, and in other instances, we call out specifics for each of the individual roles. In either case, we make an effort to differentiate.

We also want to call out we will refer to the category of None as either "first-time collaborators," "first-time contributors," or "first-timers," all of which represent the same set of first-time users to a project. Likewise, we will refer to the all the remaining categories as an aggregate with either "existing collaborators" or "existing contributors," who again, make up any user having previously contributed to the project.

Our study performs a statistical analysis of the data across several metrics, described below. We want to point out that we ran the Anderson-Darling normality test on our data sets and in all cases the data are non-normalized. This is largely due to the context surrounding pull request and issues on GitHub and not a strike against the data itself. Therefore, we used the non-parametric Mann-Whitney [40] test when comparing potential indicators. This test determines the significance of a difference in central tendency of two unpaired samples. We always report the p-value adjusted for ties.

We explore the data analyzed to answer each RQ in the following (Section V for RQ1, and Section VI for RQ2).

A. Reproducibility

For the purposes of reproducibility and independent study, we have made all data, scripts, and a list of network automation repositories available via an online appendix.[3]

V. RQ1: On Pull Request Characteristics

RQ1 seeks to determine if there is a relationship between pull requests opened by first-time contributors and the ultimate fate of those pull requests. We analyze this comparing to experienced members and projects in other domains. We consider a pull request to be successful if it has been merged into the project's codebase. Pull requests are unsuccessful whenever they are closed for any other reason.

We approach this question from a few different angles, and for each, we offer our analysis of the results and potential factors leading to the examined numbers.

A. State at Closure

One of the first and most key elements of the pull request data that exampled was the state of a pull request at the time it was closed. As we mentioned before, a pull request can either be merged or rejected. When rejected, there is additional state information that is key to the determination of why that pull request may have been closed. Those states include clean, dirty, draft, unstable, and unknown. Each of these states carries a particular context, especially when a repository is using Continuous Integration (CI) checks on commits and pull requests. A *clean* state indicates the pull request will not produce any merge conflicts against the base brace (target destination) branch, and it has passed any CI checks that may have been defined. The dirty state says that there will be one or more merge conflicts that must be resolved before the pull request can be automatically merged. A *dirty* state can sometimes be misleading in terms of quality constraints because it is not necessarily the fault of a pull request author if his pull request enters this state. For instance, if an earlier pull request that makes modifications to overlapping areas of the code base is merged, there is a good chance the second pull request will become dirty because, in terms of the git timeline, it needs to be manually updated. A *draft* state indicates the pull request was submitted in draft mode, meaning the author is not ready for this pull request to be merged and is likely under active development. It is worth noting that the draft state was only added as a GitHub feature in February of 2019 which means it is not fully represented in our overall data set, however, our findings revealed a very small number of draft pull-requests, so we generally regard them as inconsequential [41]. The *unstable* state is perhaps the most interesting to our current research because it indicates that one or more CI checks are failing, and this is usually a direct indication of a quality problem with the proposed contribution. As pointed out by Vasilescu *et al.* [42], the purpose of CI pipelines is to improve the overall quality of the codebase by enforcing a passing bar for new code. The *unknown* state is a bit of a catch-all, but usually, it means the pull request was closed before GitHub could determine the auto-merge state or before CI checks completed and reported back to GitHub.

When looking at the Top-10 projects, a total of 29,962 pull requests were opened and 30,470 were closed, of which 8,445 were rejected. The discrepancy in open vs. closed is explained by closure of pull requested opened before our 2019 data set window, and further backed up by our analysis of pull request age, later. When compared to the network automation data set, there were 3,578 pull requests opened and 3,589 closed,

[3]https://doi.org/10.6084/m9.figshare.12333143

of which 437 were rejected a rejection rate of 28% for the Top-10 and 12% for network automation. It is also worth noting at this point that there is a small number of pull requests that were opened in 2019 in both data sets but were not acted upon by the end of the year and thus are not included in our results which look at only those pull requests which were definitively closed for some reason.

In terms of closure state, the Top-10 saw a 66% unstable rate for existing contributors vs. just 47% for that of first-time contributors. Network automation, on the other hand, saw a 35% unstable state rate for existing contributors and 34% for first-time contributors. Another significant closure state we looked at is the dirty state. The Top-10 saw 15% for existing contributors and 17% for first-time contributors, which compares to 22% and 24% for network automation, respectively. Table II summarizes this information. The noticeable aspects of these numbers are that network automation sees a higher rate of closures due to dirty pull requests and that first-time contributors, in general, are responsible for slightly more of these dirty pull requests.

TABLE II
PERCENTAGE OF PULL REQUESTS CLOSED IN UNSTABLE AND DIRTY STATES

	Unstable		Dirty	
Contributor	**Top-10**	**Net. Auto.**	**Top-10**	**Net. Auto.**
Existing	66%	35%	15%	22%
First Time	47%	34%	17%	24%

B. Number of Commits

Pull requests are made up of one or more commits, which are small units of work. It is well understood that "good" pull requests, i.e., those more likely to be successfully merged, are small in scope and that generally correlates to a lower number of total commits contained within a pull request [5]. Again, while this sort of research already exists, we put it in the context of new contributors.

We present Figure 1 as a summary of our findings for the number of commits per pull request. From these, we can observe that there is a difference in the distribution of merged pull requests from first-timers when we compare network automation and Top-10 projects (p-value=0.028)—given the size of our sample, the difference can be considered marginal. Besides, we also found that there is a marginal difference for rejected pull requests (p-value=0.025). With this information, by analyzing the boxplots and the data, we observed that pull requests by first-timers in network automation projects have more commits that in the Top-10 projects. When analyzing the pull requests by experienced developers, we found the same trend for the merged pull requests (p-value$<$0.001). However, we found no significant difference for the rejections.

When analyzing rejected pull requests comparing first-timers and experienced members, we found that for the rejections experienced contributors' have more commits than those of first-timers' pull requests (p=0.026 for network automation projects, p$<$0.001 for Top-10). We highlight that, due to sample size the difference observed for the network automation can be considered marginal. For the merged pull requests, the apparent difference in the boxplots is confirmed for the Top-10 projects (p$<$0.001). However, we could not confirm the difference between first-timers and experienced contributors for the network automation projects (p=0.84).

We also analyzed the differences between pull requests merged and rejected according to the contributor type. While we could not observe statistical difference comparing first-timers for Network Automation, we found that there is a marginal difference for the Top-10 (p=0.028). On the other hand, both for Top-10 and network automation, it was possible to note differences between merged and rejected pull requests by experienced members.

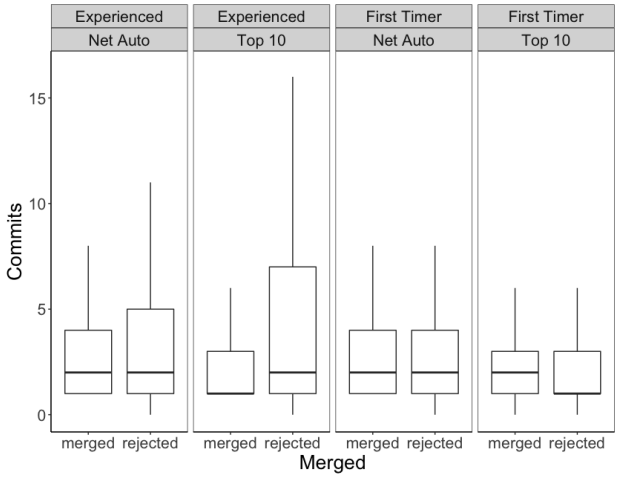

Fig. 1. Pull Request Commits

C. Size of Changes

Along the same lines as the number of commits in a pull request, the size of the overall changes to the code base proposed in a pull request also correlates to the end state of said pull request [5]. We identify the details of trends as they relate to first-time contributors. The size of a change is broken into three metrics: the number of additions, the number of deletions, and the number of files touched. These are clear indicators of the size of a proposed change and directly correlate to the acceptability of a change [43].

Right off the bat, the boxplot in Figure 2 shows some trends, notably that network automation pull requests contain more additions when compared to the Top-10. The Mann Whitney tests confirm the differences. First, when analyzing the first-timers' pull requests, we found a significant difference for both merged and rejected (p=0.029 for merged, p$<$0.001 for those rejected). When analyzing the pull requests by experienced members, we could not find statistical significance.

When analyzing only Network Automation projects, we found that additions in merged and rejected pull requests by first-timers are different (p=0.015). Moreover, we can see that rejected pull requests are different when comparing first-timers

and experienced in Network Automation(p=0.001). Interestingly, we could not find these differences when analyzing the pull requests in the Top-10 set.

Figures 2, 3, and 4 show our findings for pull request additions, deletions, and files touched from both Network Automation and Top-10 projects. In line with the established literature, we observe that the amount of changes on rejected pull requests is usually larger when compared to merged. Interestingly, we see that situation is exaggerated within the network automation projects. For first-timers, it is clear that they have more additions in their rejected pull requests over their merged ones (p=0.015); but the same comparison to both deletions and files touched was not significant. The Top-10 first timers, on the other hand, do not show a significant difference in their additions or deletions across merged and rejected pull requests, but their files touched show difference for rejected pull requests (p=0.030). First-timer pull requests in network automation see fewer overall changes within merged pull requests compared to experienced member's pull requests (p<0.001 for each of additions, deletions, and files touched).

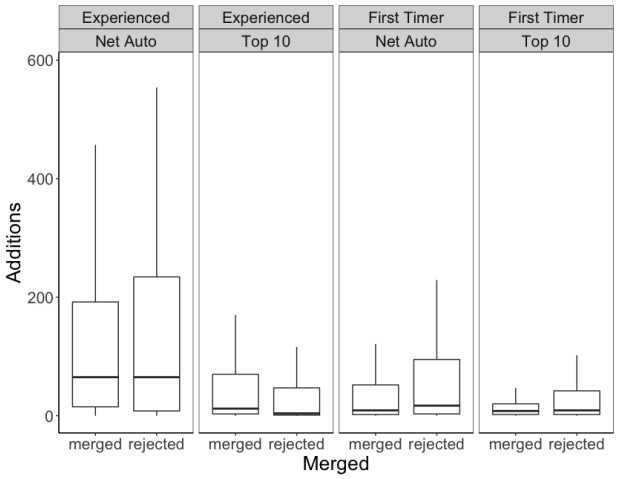

Fig. 2. Pull Request Commit Additions

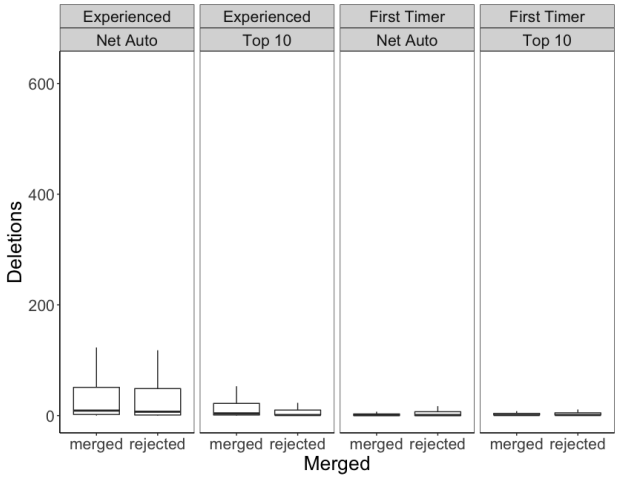

Fig. 3. Pull Request Commit Deletions

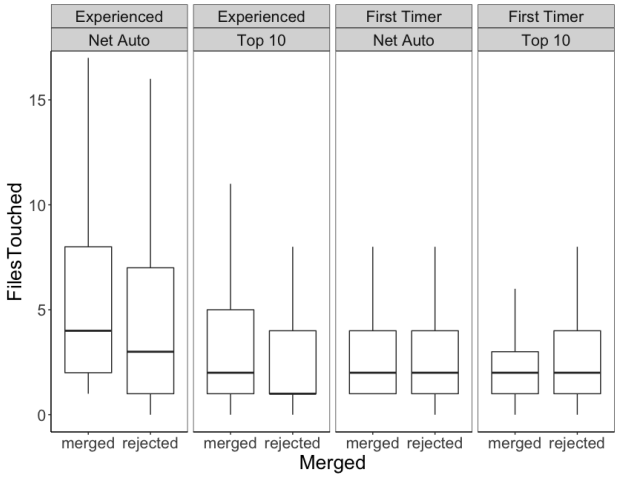

Fig. 4. Pull Request Commit Files Touched

D. Pull Request Comments

Pull request comments are a vital metric to study because they relate directly to community collaboration within a project between developers [44]. Pull request comments come in two forms, and formally are used for two different purposes. General pull request comments can be added at any point and are generally used to discuss higher-level details of the pull request. Review comments are a part of a formal code review of the pull request. These reviews are an asynchronous way to perform peer review of a proposed change and allow participants to tie comments to files and even individual lines of code. We studied the number of each of these types of comments on each pull request.

The first thing that stands out from the Figure 5 is network automation communities are not engaged in conversation through pull requests comments as much as the Top-10. We see this in the comparison of pull requests made by experienced contributors (p<0.001 for merged) and across first timer pull requests (p<0.001 for both merged and rejected). We also noted first timers in network automation see fewer comments compared to their experienced counterparts for both merged and rejected pull requests (p<0.001). This is interesting because the exact opposite is true in the Top-10 (p<0.001 for both merged and rejected).

Review comments are hard to analyze because the average pull request receives very few. That being said, it appears network automation projects are not engaging in peer review through these comments nearly as much as the Top-10 when comparing pull requests from experienced contributors (p<0.001 for merged and p<0.001 for rejected). We also found that first timers in network automation are receiving marginally fewer review comments on rejected pull requests vs the Top-10 (p=0.033).

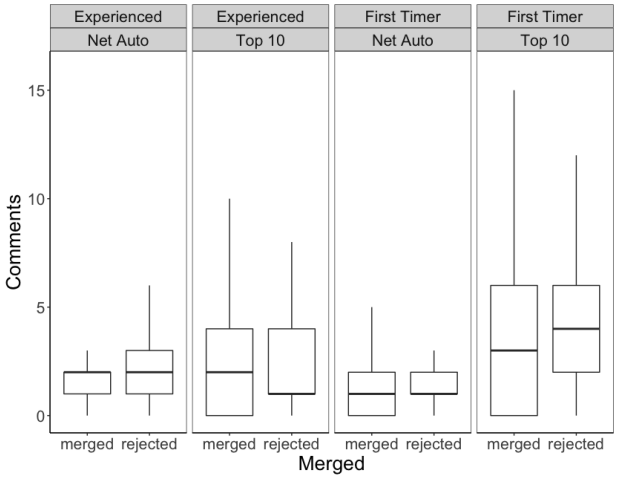

Fig. 5. Pull Request Comments

E. Pull Request Age (latency)

The final metric we investigated concerning pull requests was the age of a pull request when it is closed, also known as latency [4]. This metric is relevant because it can help to determine the staleness of contributions, which ultimately is a detriment to both the maintainers and the submitter of a pull request. If a pull request sits too long, it risks becoming dirty, and the longer a first-time contributor waits to hear final feedback on a pull request, the more disheartened or jaded he may become about a project or the process in general. On the flip side, maintainers of FOSS projects are sometimes donating time, and thus the time needed to review and decide on open pull requests comes at a premium [4].

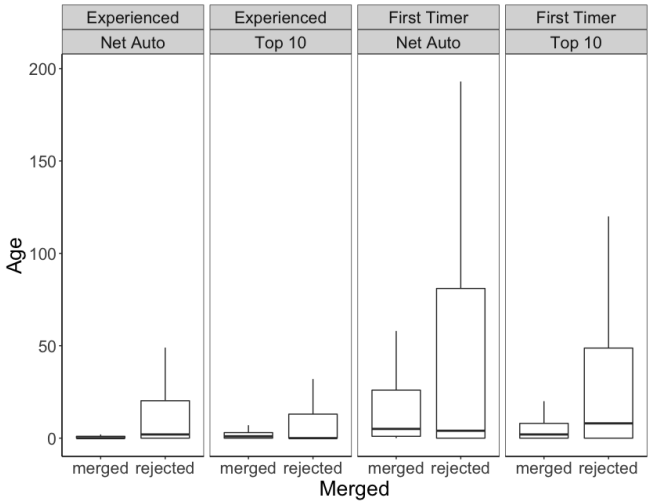

Fig. 6. Pull Request Age in Days

As it is possible to observe in the Figure 6, there are clear differences in the distribution of latency of pull requests. Most noticeably, first-timers' pull requests take longer than those submitted by experienced members, both for the Top-10 projects and Network Automation ($p<0.001$). Moreover, the boxplots show that the rejected pull requests take longer than merged ones. Looking at Mann-Whitney results, this holds for all configurations ($p<0.001$) except for first-timers of Network Automation projects; in which we could not find statistical significance when comparing merged and rejected ($p=0.545$).

TABLE III
DESCRIPTIVE STATISTICS FOR PULL REQUEST AGE IN DAYS

			N	Avg	Q1	Med	Q3	Max
Exp.	**Top-10**	Merg	18476	5.20	0.00	1.00	3.00	608.00
		Rej	6137	17.91	0.00	0.00	13.00	750.00
	Net Auto	Merg	2931	4.27	0.00	0.00	1.00	538.00
		Rej	260	53.18	0.00	2.00	20.75	639.00
First Timer	**Top-10**	Merg	3549	10.07	0.00	2.00	8.00	430.00
		Rej	2302	38.22	0.00	8.00	49.00	819.00
	Net Auto	Merg	221	40.94	1.00	5.00	23.50	746.00
		Rej	177	72.00	0.00	4.00	82.00	1023.00

RQ1 Summary. We found marginal differences in first-timer pull requests related to the number of commits in both data sets, and we found in network automation first-timers submit more commits than experienced members. We found that network automation pull requests contain larger change sets in general, but for first-timers, we only found significance in their pull request additions over merged and rejected states. The comments metric showed us that network automation is not engaging in pull request conversation as much as the Top-10 and that first-time contributors receive significantly fewer of these comments. Pull request latency, perhaps, is the most telling metric, as it shows pull requests from first-time contributors sit unresolved significantly longer than experienced contributors, which is a fairly clear indicator of the quality constraints which live therein and presents opportunity for future study into those constraints.

VI. RQ2: On Issue Characteristics

RQ2 centers around the other core feature of GitHub, issues. To answer RQ2, we identified several metrics for issues. We wanted to know if there is a relationship between issues and first-time collaborators. Issues, however, have a different context than pull requests. While pull requests are used to submit code changes, issues are the GitHub mechanism for asking questions, proposing ideas, and reporting bugs. They are used as a communication thread but can also be linked to pull requests and other issues [45]. A study of the Apache webserver project by Mockus *et al.* [33] revealed that issue trackers on platforms like GitHub are used in profoundly different ways, which is consistent with our own findings.

The timing semantics surrounding issues work a bit differently than those of pull requests. Issues track ideas, feature requests, and bug reports, while pull requests track proposed low-level code changes. For this reason, we spend more time examining the open and close dates of issues to identify boundaries within the year 2019.

In the Top-10 data set, 40,645 issues were opened in 2019, and 36,249 total issues closed, but many of those were opened

before the beginning of 2019. In total, 31,322 were opened and closed in 2019. In the network automation data set, there were 1,750 issues opened, and 1,261 were closed within the year. There were a total of 1,604 issues closed in 2019, but similarly to pull requests, some of these were opened before 2019.

A. Issue Comments

As a starting point, we analyzed the total number of comments that existed in an issue thread when it was closed. This is one metric that allows us to begin to comprehend the rigor of the conversation being undertaken in issues. As noted by Liu *et al.* [46], the social aspect of FOSS is a major contributing factor in the success of any project, so it stands to reason a correlation with issue comments would be significant.

We present Figure 7 to summarize the issue comments data. Keeping in mind the relative difference in population size between the two data sets, we first noted that network automation sees fewer comments from both first timer and experienced contributors compared to the Top-10 ($p<0.001$ in either case). Second, within network automation, experienced member's issues received fewer comments than first timers ($p<0.001$). While visually it appears the opposite is true of the Top-10, we were unable to find any significance in the Mann-Whitney test result.

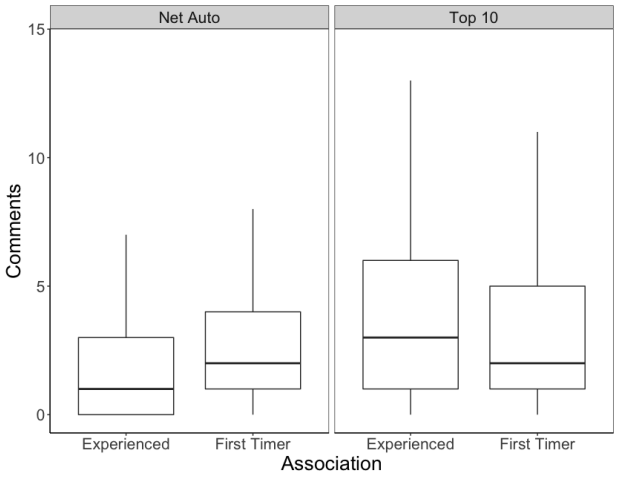

Fig. 7. Number of Issue Comments

B. Issue Age

Like pull requests, it is also essential that we consider the age of issues when they are finally closed. For this we present our summary in Figure 8. First, we can see a large difference in the age of issues within network automation ($p<0.001$) and we found the same to also be true for the Top-10 data set ($p<0.001$). We can also make out that experienced collaborators in network automation have an issue open for much longer than those experienced members in the Top-10 ($p=0.009$). Finally, we found that first timers in network automation have issues closed sooner than those of first timers in the Top-10 ($p<0.001$).

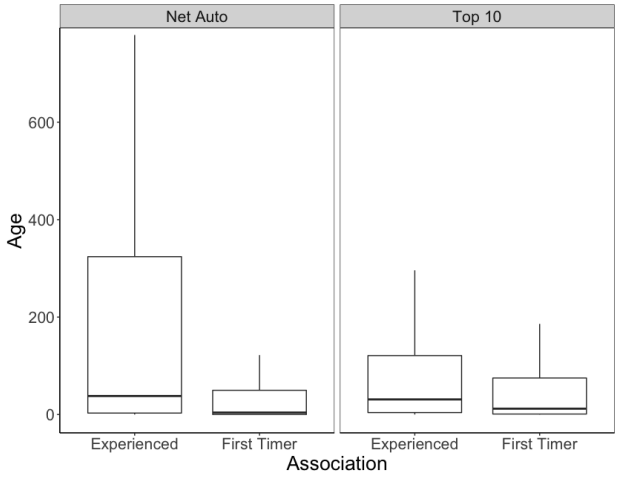

Fig. 8. Issue Age in Days

C. Issues Resulting in Pull Request

One of the primary uses of issues is to facilitate discussion before the submission of a pull request. For this reason, the final and potentially most telling metrics we studied was the identification of issues that resulted in the submission of a pull request. Li *et al.* has conducted extensive research in the way GitHub issues are linked to pull requests and other issues and found that the practice is not only useful but popular among developers on GitHub [47].

We did our typical analysis and found for the Top-10 that 9,005 pull requests were submitted that resulted from issues opened *by existing users*. Of those, just 4,028 (45%) were merged. We dug a little deeper and discovered of those merged pull requests, 905 were submitted by first-time contributors (about 25% of them). Of the rejected pull requests in this category, just 23 were opened by first-time users.

Next, we compared these results to issues opened *by first-time contributors* in the Top-10 data sets and found 2,051 issues resulted in a pull request submission. Of those, 899 (44%) were merged, and the same first-time collaborators submitted all. Of the rejected pull requests, 1,176 were submitted by the first time contributors. The discrepancy in comparison to the total is due to a specific small percentage of items opened before 2019 but closed within the year. We only look at items that have been closed for any reason.

To continue our analysis, we then studied the network automation data set. We found 375 pull requests that linked to issues submitted by existing contributors vs. 112 for issues opened by first-time collaborators. Of the 375, a total of 226 (60%) pull requests were merged, and first-timers submitted 35 (15%) of those. Again we saw the same phenomena in which 112 pull requests from first time issues, 9 were merged, and they were all from first time collaborators. From the existing contributors' issues, 2 resulting pull requests were rejected and had been submitted by first-time contributors, compared to 106 rejections from first-time contributor's issues.

RQ2 Summary. We found significance in fewer issue comments in network automation, and for both data sets that first-time contributors have their issues closed sooner. We also identified a low "pick-up" rate of pull requests submitted by first-time contributors resulting from issues opened by experienced members. This culminates in us declaring that the number of issue comments, issue age, and analysis of pull requests linked to issues is good metrics for comparative analysis and we would want to further study their relation as quality constraints.

VII. Discussion

Our work initially set out to explore two questions related to first time contribution. To reach this goal, we devised a framework of metrics by which to analyze GitHub event data to identify trends. Our framework is based entirely off of first order attributes available with the event data. We feel this makes our approach very palatable to all sorts of individual use cases and skillsets. Much of the existing work on GitHub data mining relies on complex code analysis, natural language processing, or machine learning techniques [48]. We do not mean to diminish this research or their approaches; instead, we are arguing that the relative simplicity of the technical requirements in our methodology makes it easier to reproduce.

We apply our method to two different data sets with the same structure; one that represents the niche domain of network automation and another, which is comprised of the Top-10 trending repositories on GitHub, which is meant to be used as a control of the broader industry. Part of our goal in this work was to glean information from this niche domain, so have a basis by which to compare was vital. This means that our work can be reused to examine other subdomains within the software engineering field.

For network automation, we found that for many metrics, this domain is performing on par with the broader industry. There were several cases of deviation, however, that we attempted to address. Some of this is certainly due to the nature of network automation projects, which often tend to be low-level tools intended for use by a specific set of people in the industry. Still though, there are several large scale projects in the place just as NetBox [49], Batfish [50], NSoT [51], NAPALM [52], Netmiko [53], and others which do attract a proper gathering of FOSS involvement. It is these projects for which our interest is first time contribution is intended, and we feel we have contributed some meaningful findings.

Different acceptance rate: Our results showed that the acceptance rate for pull requests in Network Automation (88%) is considerably higher than the Top-10 projects (72%). Interestingly, the number for the Top-10 is in line with previous studies (64% [54], 71% [55]). We believe that the domain specifics, and the policies of the projects influence these numbers. Top-10 projects, are well-recognized and mature projects with potentially high standards, while Network Automation projects are smaller, and may welcome more contributors seeking to create a community.

Instability of pull requests: When analyzing the state of the pull requests at closure (Section V-A), it appears that there are more unstable pull requests in the Top-10 than in the Network Automation projects. It is important to consider that the Top-10 projects see many more contributions, and that have tools and processes in place to help triaging pull requests. For example, analyzing the Top-10 projects, it was possible to notice that Continuous Integration is set running. Due to the nature of the Network Automation projects, this may be the case that triaging processes or CI pipelines are not commonplace. This can also be the case that only 31% of projects state [56].

Size of pull requests: According to the existing literature on FOSS [1], [6], smaller pull requests have more chances to be merged. We found that pull requests in Network Automation domain are larger than in Top-10 projects in terms of the number of commits and lines added (Sections V-B and V-C). This may be related to the practices in the Networks Automation community, which is frequently composed of contributors without a software development background.

Pull requests take longer than usual to be processed: It was clear that the pace of Network Automation is slower than Top-10 projects. There are a few potential explanations for the apparent delay of first-time submissions, but the most logical is to remember the pull request size and scope disparities evaluated earlier. This is higher than the numbers reported by Pinto et al. [57], who found that pull requests made by volunteers in company-owned projects take, in average, 11.37 days to be processed (q3=5, stdev=55). For the sake of comparison, first-timers rejected pull requests take on average 72 days (q3=82, stdev=135), while merged take 40.94 (q3=23.5, stdev=5.93). It is likely that project maintainers simply look at the first timer's pull request as a bit of a daunting task because they have proven to be much larger and thus take more time to review and understand realistically. Another factor leading to long review times is duplicate pull requests. Li *et al.* showed that duplicates are abundant in larger projects and can be hard to weed out [58], [59].

First timers' issues are closed faster: First-timers' issues are staying open for roughly half the total time of issues from existing members of a project. There are several potential reasons for this, including the fact that regular users often use issues as a software support mechanism. We say regular users here because this is a class of GitHub users that are not engaging in software development but are merely looking for help in using software from an end-user standpoint. This is still a quality constraint in our minds because it is a project maintenance burden. While issues are commonly used as a support forum, they quickly become noise when the issue backlog is primarily being used to manage the software development life cycle. Other factors playing into this ratio may include incorrect bug reports, duplicates, malformed styling, or language barriers, to name a few [60]. Some maintainers have even turned to custom solutions like bots and web apps to help stem the tide and review of issues, as pointed out in work conducted by Ristemi *et al.* [61], [62].

Pull requests linked to issues: When analyzing RQ2, we could see a 45% acceptance rate of pull requests that are linked to issues. We see a disproportionately lower number of successful pull requests from first-time contributors within the network automation data set. We also observed, in the Top-10 dataset, that first-time contributors are attempting to address 24% of issues they did not open (relative to all submitted pull requests), but in the network automation data, first-timers are only attempting 10% of all issue related pull requests. We call this the issue "pick up rate" as it designates work that someone is taking on that was initially proposed by someone else. Jiang *et al.* touched on this in their work, which analyzed the proportion of all available users within a project that can contribute vs. those that do [63].

Call for action: The results show that the characteristics of pull requests on Network Automation projects differ from the Top-10 and from what is presented by the current literature. This may be attributed to the fact that the contributors to the projects under this domain lack a software engineering background. Therefore, the way that the projects function, including the processes and tools used to maintain the contribution flow, is different from the state-of-the-art of mature software projects. For *researchers*, this is an opportunity to understand how the process differs in various domains, and how they adhere to well-known practices and tools used in mature projects. An open gap here regards the potential impact of these differences in the code in terms of, for example, maintainability, efficiency, smells, and reliability. It is clear that knowledge related to software development and maintenance is no longer optional. For *educators*, it is necessary to include basic knowledge of the development process in the undergraduate curriculum, focusing on a non-software developers cohort.

VIII. Limitations

There are examples in software engineering literature about perils of mining GitHub data, specifically such as consistent use of platform features, differing project organizational models, data staleness, and uses of GitHub unrelated to software engineering [37], [64], [65]. In this paper, we address some of these issues with specific remedies within our work, but generally, our study makes use of a hand-selected set of repositories to weed out potentially problematic and unrelated data from the very beginning.

As we mentioned in our method, the author association attribute does not hold temporally. Once someone becomes a contributor to a project, for example, all their pull request and issues will carry the "contributor" association, becoming not possible to find, for example, the issues that this person opened when their association was "none." In our case, we relied on the GitHub Archive dataset, which holds the temporal aspect of the issues and pull requests, making it possible to trace back the association at the time of the creation of the issues. Still, while there exist some edge case in which such a person has submitted a successful pull request that was merged to a non-default branch, in the vast majority of cases, authors designated as None, are first-time contributors.

Kalliamvakou et al. [66] stated that many merged pull requests appear non-merged, because commits were merged through rebase or squash. This may have affected our validity since we consider the number of merged/rejected pull requests.

We acknowledge that issues and pull requests may be reopened multiple times during their lifetime. We deliberately did not factor in pull requests and issues which are reopened. Such events may be an additional indicator of quality constraints as, in some cases, they imply an underlying issue with the initial implementation or desire to rehash a conversation. We wish to analyze this further, and in particular, look into work produced by Mohamed *et al.* on the subject [67].

The size of the data set used in this research [7] is a double-edged sword. It contains a broad array of information that we analyze and compare, but to do so at scale requires expertise in big data analysis. For reference, the entire year of 2019 in this data set is 1.7TB in size. Previous work makes use of Google's BigQuery online service for which the GitHub Archive has a direct integration. While BigQuery performs efficient data processing, in this context, it is not cost-effective. To combat this, we reduced the target data set that of only events relevant to the 81 projects analyzed. The resulting subset of data was 6GB large. We created a local copy of this dataset into a PostgreSQL database, which we queried consistently without other constraints of BigQuery. From this more manageable data set, we further extracted data directly related to our prescribed metrics to run our statistical analysis.

We used the Top-10 projects from Octoverse as our baseline. Although this set makes up an excellent set to compare, they do not represent the average projects available on GitHub. To alleviate this problem, in discussion, we compare our results with existing literature that analyze the same kind of data.

IX. Conclusion

In this paper, we explored the characteristics of first-time contributions to FOSS projects. We devised a method of data analysis that uses existing data sets of GitHub events from across many repositories. We set up a baseline to explore trends in the broader industry and apply our methods to a specific domain. Part of our contribution also involves the application of our methods to the domain of network automation, which is a growing area seeing the introduction of developers without traditional software engineering skill sets.

Throughout our research, we identified metrics and factors that can be used to understand the characteristics of pull requests and issues in FOSS projects. While the Network Automation projects are more welcoming (only 12% of rejection rate), the characteristics of the pull requests and issues in this domain are different from mature and well-known projects. These differences may be related to the lack of knowledge about software engineering best practices and tools, which may ultimately impact code quality. Future studies may focus on differences in other domains, and in differences in terms of the quality of the code produced by these communities.

REFERENCES

[1] G. Pinto, I. Steinmacher, L. F. Dias, and M. Gerosa, "On the challenges of open-sourcing proprietary software projects," *Empirical Softw. Engg.*, vol. 23, no. 6, p. 3221–3247, Dec. 2018. [Online]. Available: https://doi.org/10.1007/s10664-018-9609-6

[2] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder," *IEEE Software*, vol. 30, no. 1, p. 52–66, 2013.

[3] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE'16. New York, NY, USA: ACM, 2016, pp. 273–284.

[4] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *2015 IEEE/ACM 12th working conference on mining software repositories*. IEEE, 2015, pp. 367–371.

[5] C. Treude, L. Leite, and M. Aniche, "Unusual events in github repositories," *Journal of Systems and Software*, May 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300876?via=ihub

[6] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 345–355. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568260

[7] "Gh archive." [Online]. Available: https://www.gharchive.org/

[8] "The state of the octoverse," 2019. [Online]. Available: https://octoverse.github.com/

[9] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[10] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[11] I. Steinmacher, M. A. Gerosa, and D. Redmiles, "Attracting, onboarding, and retaining newcomer developers in open source software projects," in *Proceedings of the Workshop on Global Software Development in a CSCW Perspective*, ser. CSCW '14 Workshops, 2014.

[12] R. Pham, L. Singer, and K. Schneider, "Building test suites in social coding sites by leveraging drive-by commits," in *ICSE 2013*. IEEE, 2013, pp. 1209–1212.

[13] Y. Hu, J. Zhang, X. Bai, S. Yu, and Z. Yang, "Influence analysis of github repositories," *SpringerPlus*, vol. 5, no. 1, pp. 1–19, 2016.

[14] F. Fronchetti, I. Wiese, G. Pinto, and I. Steinmacher, "What attract newcomers to onboard on oss projects? tl; dr: Popularity," in *15th International Conference on Open Source Systems (OSS)*, 2019.

[15] I. Steinmacher, M. A. Gerosa, and D. Redmiles, "Attracting, onboarding, and retaining newcomer developers in open source software projects," in *Workshop on Global Software Development in a CSCW Perspective*, 2014.

[16] A. Rastogi, N. Nagappan, G. Gousios, and A. van der Hoek, "Relationship between geographical location and evaluation of developer contributions in GitHub," in *12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2018, pp. 22:1–22:8. [Online]. Available: http://doi.acm.org/10.1145/3239235.3240504

[17] G. Pinto, L. F. Dias, and I. Steinmacher, "Who gets a patch accepted first? comparing the contributions of employees and volunteers," in *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2018, pp. 110–113.

[18] V. Sinha, A. Lazar, and B. Sharif, "Analyzing developer sentiment in commit logs," in *MSR 2016*. New York, NY, USA: ACM, 2016, pp. 520–523. [Online]. Available: http://doi.acm.org/10.1145/2901739.2903501

[19] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in GitHub: An empirical study," in *11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 352–355. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597118

[20] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in GitHub teams," in *CHI 2015*, 2015, pp. 3789–3798.

[21] C. Mendez, H. S. Padala, Z. Steine-Hanson, C. Hilderbrand, A. Horvath, C. Hill, L. Simpson, N. Patil, A. Sarma, and M. Burnett, "Open source barriers to entry, revisited: A sociotechnical perspective," in *40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 1004–1015. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180241

[22] J. Tantisuwankul, Y. S. Nugroho, R. G. Kula, H. Hata, A. Rungsawang, P. Leelaprute, and K. Matsumoto, "A topological analysis of communication channels for knowledge sharing in contemporary github projects," Sep 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219301906?via=ihub

[23] M. M. Rahman, C. K. Roy, and J. A. Collins, "Correct: Code reviewer recommendation in github based on cross-project and technology experience," in *38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 222–231. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889244

[24] R. Padhye, S. Mani, and V. S. Sinha, "A study of external community contribution to open-source projects on github," in *11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 332–335. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597113

[25] V. J. Hellendoorn, P. T. Devanbu, and A. Bacchelli, "Will they like this?: Evaluating code contributions with language models," in *12th Working Conference on Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 157–167. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820539

[26] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *ICSE*, 2014, pp. 356–366.

[27] Y. Tao, D. Han, and S. Kim, "Writing acceptable patches: An empirical study of open source project patches," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sept 2014, pp. 271–280.

[28] A. Alami, M. L. Cohn, and A. Waisowski, "How do foss communities decide to accept pull requests?" in *Proceedings of the Evaluation and Assessment in Software Engineering*, ser. EASE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 220–229. [Online]. Available: https://doi.org/10.1145/3383219.3383242

[29] D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, ""was my contribution fairly reviewed?" a framework to study the perception of fairness in modern code reviews," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 523–534.

[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*, ser. SAC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1541–1546. [Online]. Available: https://doi.org/10.1145/2695664.2695856

[31] I. Steinmacher, G. Pinto, I. S. Wiese, and M. A. Gerosa, "Almost there: A study on quasi-contributors in open-source software projects," in *ICSE 2018*, 2018, pp. 256–266.

[32] A. B. Dhasade, A. S. M. Venigalla, and S. Chimalakonda, "Towards prioritizing github issues," in *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly Known as India Software Engineering Conference*, ser. ISEC 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3385032.3385052

[33] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: The apache server," in *Proceedings of the 22nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 263–272. [Online]. Available: https://doi.org/10.1145/337180.337209

[34] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 188–197.

[35] R. Kikas, M. Dumas, and D. Pfahl, "Issue dynamics in github projects," in *Product-Focused Software Process Improvement*, P. Abrahamsson, L. Corral, M. Oivo, and B. Russo, Eds. Cham: Springer International Publishing, 2015, pp. 295–310.

[36] J. Cabot, J. L. Cánovas Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software

projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 550–554.
[37] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 92–101.
[38] M. Allamanis and C. Sutton, "Mining idioms from source code," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 472–483. [Online]. Available: https://doi.org/10.1145/2635868.2635901
[39] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *Journal of Systems and Software*, vol. 140, pp. 32 – 47, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218300323
[40] H. Mann and D. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.
[41] "Introducing draft pull requests." [Online]. Available: https://github.blog/2019-02-14-introducing-draft-pull-requests/
[42] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 805–816.
[43] K. Muthukumaran, A. Choudhary, and N. L. B. Murthy, "Mining github for novel change metrics to predict buggy files in software systems," in *2015 International Conference on Computational Intelligence and Networks*, 2015, pp. 15–20.
[44] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "A exploratory study of @-mention in github's pull-requests," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1, 2014, pp. 343–350.
[45] N. Bleiel, "Collaborating in github," in *2016 IEEE International Professional Communication Conference (IPCC)*, 2016, pp. 1–3.
[46] J. Liu, J. Li, and L. He, "A comparative study of the effects of pull request on github projects," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 313–322.
[47] L. Li, Z. Ren, X. Li, W. Zou, and H. Jiang, "How are issue units linked? empirical study on the linking behavior in github," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 386–395.
[48] N. Khadke, M. H. Teh, and M. Shen, "Predicting acceptance of github pull requests," Stanford–CS 229, Tech. Rep., 2012.
[49] Netbox-Community, "netbox-community/netbox," Apr 2020. [Online]. Available: https://github.com/netbox-community/netbox
[50] Batfish, "batfish/batfish," May 2020. [Online]. Available: https://github.com/batfish/batfish
[51] Dropbox, "dropbox/nsot," Oct 2019. [Online]. Available: https://github.com/dropbox/nsot
[52] Napalm-Automation, "napalm-automation/napalm," May 2020. [Online]. Available: https://github.com/napalm-automation/napalm
[53] Ktbyers, "ktbyers/netmiko," Apr 2020. [Online]. Available: https://github.com/ktbyers/netmiko
[54] 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*, 2016, pp. 871–882.
[55] J. Terrell, A. Kofink, J. Middleton, C. Rainear, E. Murphy-Hill, C. Parnin, and J. Stallings, "Gender differences and bias in open source: pull request acceptance of women versus men," *PeerJ Computer Science*, vol. 3, p. e111, May 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.111
[56] O. Elazhary, M. Storey, N. Ernst, and A. Zaidman, "Do as i do, not as i say: Do contribution guidelines match the github contribution process?" in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 286–290.
[57] G. Pinto, L. F. Dias, and I. Steinmacher, "Who gets a patch accepted first? comparing the contributions of employees and volunteers," in *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2018, pp. 110–113.
[58] Z. Li, G. Yin, Y. Yu, T. Wang, and H. Wang, "Detecting duplicate pull-requests in github," in *Proceedings of the 9th Asia-Pacific Symposium on Internetware*, ser. Internetware'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3131704.3131725
[59] X. Zhang, Y. Chen, Y. Gu, W. Zou, X. Xie, X. Jia, and J. Xuan, "How do multiple pull requests change the same code: A study of competing pull requests in github," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 228–239.
[60] D. Kavaler, S. Sirovica, V. Hellendoorn, R. Aranovich, and V. Filkov, "Perceived language complexity in github issue discussions and their effect on issue resolution," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 72–83.
[61] I. Ristemi, M. A. Trpkovska, and B. Cico, "Mygitissues web application as a solution in dealing with issues on github," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, 2019, pp. 1–4.
[62] Z. Hu and E. F. Gehringer, "Improving feedback on github pull requests: A bots approach," in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–9.
[63] J. Jiang, D. Lo, X. Ma, F. Feng, and L. Zhang, "Understanding inactive yet available assignees in github," Jun 2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0950584917304457?via=ihub
[64] 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*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 364–367. [Online]. Available: https://doi.org/10.1145/2597073.2597121
[65] P. S. Kochhar, D. Wijedasa, and D. Lo, "A large scale study of multiple programming languages and code quality," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 563–573.
[66] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 92–101. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597074
[67] A. Mohamed, L. Zhang, J. Jiang, and A. Ktob, "Predicting which pull requests will get reopened in github," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 375–385.