Pull Requests or Commits? Which Method Should We Use to Study Contributors' Behavior?

Marcus Vinicius Bertoncello
Univ. Estadual de Maringá
Maringá, PR, Brazil
pg400012@uem.br

Gustavo Pinto
Univ. Federal do Pará
Belém, PA, Brazil
gpinto@ufpa.br

Igor Scaliante Wiese
Univ. Tecn. Federal do Paraná
Campo Mourão, PR, Brazil
igor@utfpr.edu.br

Igor Steinmacher
Northern Arizona University
Flagstaff, AZ, USA
igor.steinmacher@nau.edu

***Abstract*—Social coding environments have been consistently growing since the popularization of the contribution model known as pull-based. This model has facilitated how developers make their contributions; developers can easily place a few pull requests without further commitment. Developers without strong ties to a project, the so-called casual contributors, often make a single contribution before disappearing. Interestingly, some studies about the topic use the number of commits made to identify the casual contributors, while others use the number of merged pull requests. Does the method used influence the results? In this paper, we replicate a study about casual contributors that relied on commits to identify and analyze these contributors. To achieve this goal, we analyzed the same set of GitHub-hosted software repositories used in the original paper. By using pull requests, we found an average of 66% casual contributors (in comparison to 48.98% when using commits), who were responsible for 12.5% of the contributions accepted (1.73% when using commits). We used a sample of 442 developers to investigate the accuracy of the method. We found that 11.3% of the contributors identified using the pull requests were misclassified (26.2% using commits). We also evidenced that using pull requests is more precise for determining the number of contributions, given that GitHub projects mostly follow the pull-based process. Our results indicate that the method used for mining contributors' data has the potential to influence the results. With this replication, it may be possible to improve previous results and reduce future efforts for new researchers when conducting studies that rely on the number of contributions.**

***Index Terms*—Replication, Open source, Casual contributors**

I. Introduction

Part of the recent growth of Open Source Software (OSS) is related to the rise of social coding environments, like GitHub. One key point of GitHub's success—in addition to its social features—was the introduction of the pull-based model [1]. In this model, contributors *fork* a repository and make their changes independent of each other. When the set of changes is ready for review, they *pull* the contribution using a *pull request* back to the main repository. Each pull request is reviewed by project members and may be discussed and revised before a decision is made about whether it will be merged into the main repository or not [1], [2].

One crucial point is that pull requests are, ultimately, composed of commits submitted to a `git` upstream repository. This point brings some flexibility to projects that do not want to completely adhere to the pull request model, enabling them to adopt different methods to integrate the contributions submitted via pull requests. For example, while some maintainers prefer to merge commits via the command line, the Linux kernel is famous for still using patches sent to the mailing list.[1] While this flexibility may benefit practitioners, it poses many challenges to researchers analyzing data mined from GitHub. For example, when one chooses to conduct their analysis in the commit-level, problems related to author disambiguation [3] may appear. On the other hand, many obstacles arise when using pull requests [4], such as not considering the pull requests manually merged or the commits pushed directly to the upstream repository.

Does the use of pull requests or commits to mine contributors' behavior influence the results of an empirical study? To answer this question, we replicated the study conducted by Pinto et al. [5] published on SANER 2016 conference, which investigated the rise of the casual contributor phenomenon, and the comparised both results. Pinto and his colleagues define these contributors as those who performed a single contribution to a given project. However, as briefly discussed above, the concept of contribution can be misleading when we consider the pull-based model since a pull request may have multiple commits. We chose to replicate this particular topic since this casual contributor phenomenon has recently gained traction, with researchers studying it from different angles [6], [7], [5], [8]. Moreover, the nature of this phenomenon makes the results of this study of interest for research investigating other aspects of OSS communities tied to the number of contributions (e.g.,quasi-contributions [9], developers disengagement [10], [11], students participation [12], women engaging in OSS [13], among many others). All these studies rely, somehow, on data collected directly from GitHub public API [14] or existing datasets like GHTorrent [15].

Why is this replication needed? This paper can help researchers to avoid potential threats to validity in their empirical studies when deciding the approach they will take to analyze data from repositories. Although other research works highlighted that threats in mining repository activities exist [16], [4], none of them focus on these two particular approaches: commits and pull requests. In this paper, we bring to light the differences that may occur when looking at contributions from different abstraction levels. Given that more projects are migrating to GitHub and following a pull-based model, and that many other platforms now offer similar features (e.g.,

[1] https://kernelnewbies.org/FirstKernelPatch

GitLab, Phabricator, BitBucket), it is essential to understand the implications of this choice, and how to make the decision aligned to the goals of the work.

In the context of Software Engineering, a replication consists of repeating the original study by reusing its materials, design, or analysis procedures [17]. The original paper focused on obtaining all its data using command line `git log` utility to evaluate the contributions. In this replication, however, we focused only on data available via the GitHub API and stored at GHTorrent [15]. Using this data while replaying the same procedure of the original work, we observed our main finding: *the method employed to gather contributors' data does matter!*

We also found about 35% more casual contributors when using the pull request method. However, we found false-positives using both methods. Regardless of the method, researchers need to be cautious when mining contributors' data.

The main contributions of this paper are: (i) Comparison between two different ways used to measure the contributing behavior of casual contributors in OSS projects (commits and pull requests); (ii) Replication of a study that used the commit method for identifying casual contributors—we complemented the analysis using pull requests for the same purpose; (iii) Discussion about how the two approaches could yield similar and different results; (iv) A list of lessons learned to help other researchers gauge which method would better suit their needs.

II. Related work

A. Casual Contributors

The majority of past empirical studies on the social aspects of OSS projects and communities has focused on understanding the role of core and peripheral OSS developers [7]. Researchers have often neglected those contributors who do not seek long-term commitment [5], [18], [6], [19], [8], [20], [21]. However, there is a growing body of knowledge related to the so-called casual contributors [5]. Other authors use different terminologies when discussing this phenomenon. For example, Lee and Carver [7] use the term "one-time-contributor" to define those contributors who have had exactly one code contribution—the same defined by Pinto and his colleagues. "Episodic contributor" is another term—coined by Barcomb et al. [20]—aimed to define short-term, erratic, and conditional participation in OSS projects. In this paper, we keep Pinto's definition: *casual contributors are those contributors who have had exactly one contribution to an open software project.* We want to explore the differences that may occur when the concept of "contribution" changes from commit to pull request.

This "behind the scenes" role is important to OSS projects. According to the literature [5], [6], casual contributors make far from only trivial contributions. Pinto et al. [5], for example, found that more than half of their contributions relate to fixing bugs, implementing new features, and refactoring existing code. Furthermore, their contributions are of high quality and compare to those made by regular contributors [19].

When analyzing the methods of the papers that deal with casual contributors, we noticed that none of them considered pull requests when analyzing contributions. Those papers that studied GitHub contributions [5], [19], [21] conducted their analysis at commit-level. The other studies only leveraged data from surveys and interviews [8], [20], or used data from projects that are not hosted on GitHub [7], [18]—in commit level. No study investigated this phenomenon from the perspective of pull requests. More interestingly, for those studies that analyzed data from GitHub hosted projects, there is no discussion about the potential threats of analyzing commits from projects that potentially accept and merge contributions via pull request.

B. Replication Studies in Software Engineering

Replications are gaining increasing presence in the research agenda of software engineering studies, as evidenced by the increasing number of replications published in recent years [22].

Campbell and Stanley [23] argue that experiments need to be replicated in different contexts, at different times, and under different conditions before they can produce generalizable knowledge. Thus, replications can help improve the understanding of a phenomenon, since the results reported by one study do not always directly transfer to other contexts [24].

Shull *et al.* [25] identified two types of replications: (i) exact replications, when researchers apply the same procedures to answer the same research questions as closely as possible; and (ii) conceptual replications, that happens when researchers investigate the same research question by using a different experimental procedure.

Guidelines exist to help researchers to conduct replications [26], [27], [22]. According to Carver *et al.* [26] a replication paper should report the research questions, design, participants, artifacts, context variables, and summary of the results. We present each in the following section.

III. The Original Study

The original paper [5] included the following questions:

RQ1. How common are casual contributors in OSS projects?

RQ2. What are the characteristics of contributions made by casual contributors?

RQ3. How do casual contributors and project maintainers perceive contributions made by casual contributors?

The original study was the first study aimed at gaining an in-depth understanding of casual contributors, as well as their benefits and the problems they introduce. **RQ1** provided an overview of the existence of casual contributors in the set of studied projects. In **RQ2**, the authors conducted a manual inspection and a quantitative analysis to understand the casual contributor's intention when submitting their contribution. The authors presented the results of a survey conducted with casual contributors and maintainers in **RQ3**, in which they investigated causal contributors' motivations, as well as the benefits and problems associated with their presence.

Design. The original study conducted a quantitative and qualitative analysis of data from OSS projects. To select representative OSS projects, the authors queried GitHubArchive[2]

[2]https://www.gharchive.org/

to find the most popular projects in terms of the number of stars. The authors cloned the projects and used `git` commands to extract commit logs and file contents. They collected the qualitative data by surveying 197 casual contributors and 65 project maintainers. To understand the intention of the casual contributors, they manually categorized 384 contributions by analyzing commits made by casual contributors.

Projects. The authors selected the top 20 most popular projects from 16 programming languages: C, C++, Clojure, CoffeeScript, Erlang, Go, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, Ruby, Scala, and TypeScript. After curating the dataset by removing projects that were not software projects or were not active, the final dataset comprised 275 GitHub Projects. These projects had a total of 73,960 contributors who performed 2,039,376 contributions.

Artifacts. The original study shared the raw data collected. However, the dataset presents only commit data collected via `git log`, making it impossible to analyze pull requests. In this replication, we created a new dataset, relying on data obtained directly via GitHub API, and from GHTorrent,[3] comprising both pull request and commit data we could compare the two. As we shall see in Section IV-A2, we made small modifications in the original methodology to enable pull request analysis.

Context Variables. According to Basili *et al.* [28], drawing general conclusions from empirical studies is difficult because any process depends on a potentially large number of relevant context variables. In this replication, we started by selecting the same set of OSS projects curated in the original study. Besides using commit data, we complemented the data from these projects with pull request data.

IV. The Replication Study

In this section, we state our research questions and the research approach used to collect and analyze data. Similar to Pinto *et al.* [5], to collect data from casual contributors, we conducted a mixed-method approach to quantify the presence of casual contributors in software projects and qualitatively inspect the results. Our target population is the same set of projects collected by the original study. For this study, we included pull request data.

As mentioned before, in this replication, we kept the first two research questions of the original study [5]. Since our goal is to understand whether pull requests could be an interesting approach to categorize casual contributors' contributions, our intention with **RQ1** is to assess whether, say, the proportion of these contributions (using pull requests) is similar to the original study (commits). Moreover, in this study, it is important in **RQ2** to evaluate whether the same kind of contributions can be found using the pull request research method (e.g., since pull requests can contain more than one commit, contributors could keep submitting changes to the same pull request until integrators decide that the contribution is ready to be merged).

In this replication paper, we decided not to replicate **RQ3**, because we aim to better understand the method used to identify the casual contributors, and how that may affect the main results of the original paper. RQ3 aimed to understand the perceptions of casual contributors, without considering the level of granularity analyzed. More concretely, we consider out of the scope of this work a further understanding of the benefits or challenges related to contributions made by casual contributors. We thus focus on the *method*.

A. Data Collection

1) Curating the corpus of OSS projects: We start our data collection process by selecting the 275 OSS projects used in the original study. For each one of these projects, we collected data about their pull requests (merged and unmerged) and the commits made to the master branch of the project. We opted to dismiss the open pull requests, because we cannot predict the acceptance/rejection of these pull requests.

When conducting the collection process, we noticed that we would not be able to reuse the complete original dataset of projects. This happened due to three concerns:
1) **projects had less than five pull requests:** five OSS projects are popular but have a small number of pull requests—as a consequence, fewer contributors—(e.g., LULZLABS/AIRCHAT,[4]).
2) **projects were inactive:** we detected four projects that were abandoned by the maintainers (e.g., OSTINELLI/MISULTIN[5]).
3) **projects that do not accept pull requests:** three projects do not use the pull request workflow to receive patches (e.g.,TORVALDS/LINUX).and CLOJURE/CLOJURESCRIPT,[6]).

After discarding these projects, we curated a list of 263 active and popular OSS projects that use the pull request system. Besides removing these OSS projects, we also merged the commit history of the project TURBOLINKS/TURBOLINKS with the most recent version of it, TURBOLINKS/TURBOLINKS-CLASSIC. We did this to the history of the projects, allowing us to consider their pre-migration pull requests.

2) Curating the contributing metrics: In this study, we focused on comparing two contributing metrics: commits and pull requests. To retrieve both commit and pull request data, we used a GHTorrent snapshot, complemented with data from GitHub API. This choice sets the stage for a fair comparison since the data collected for commits and pull requests represents the same set of contributions in the same time frame (from the beginning of the project until the collection date: October 2017).

The data collected for the commits include the author and date, in addition to lines added, deleted, and the number of files modified. For the pull requests, we collected the decision about the pull request (i.e., whether it was merged or not), in addition to the number of files changed per pull request, lines added and removed, number of commits, and other related events (e.g., close, force-push, merge).

After manually investigating some of the pull requests collected, we noticed an interesting behavior. Some contributions

[3]http://ghtorrent.org

[4]https://github.com/lulzlabs/AirChat
[5]https://github.com/ostinelli/misultin
[6]https://github.com/clojure/clojurescript

that were part of pull requests flagged as unmerged were in fact accepted and merged in the repository. This occurred for at least three reasons: (1) commits had been cherry-picked, (2) the contribution was squashed into a different commit, (3) or the contents of the patch were copied to a separate commit. These false-negative contributions and proposed approaches to deal with them have been identified in the software engineering literature [2], [4]. We decided to apply a conservative version of the set heuristics introduced by Gousious and colleagues [2], to minimize the effect of false-negative contributions (i.e., the ones accepted and merged in the repository codebase with pull requests flagged as unmerged). The heuristics are the following:

- **The presence of commits in the master branch.** We checked whether at least one of the commits associated with the pull request appears in the master branch of the target project.
- **The use of keywords that suggest merging activities right after closing the pull requests.** The last comment before closing the pull request was analyzed to verify the existence of words related to the acceptance of the contribution. The words chosen were based on the heuristic by Gousios [2] ("pulled," "pushed," "integrated"), including some new ones noticed in our manual analysis ("landed," "LGTM," "cherry pick").
- **The presence of SHAs in the last comment.** The latest comment before the close event contains a commit SHA identifier (40-characters or a shortened version) that exists in the project's master branch.
- **The presence of SHAs in the "close" event.** By manually analyzing the pull requests, we noticed that in some projects the members provided with the commit SHA in the close events (e.g., ANGULAR/ANGULAR-JS[7]). This heuristic was not originally proposed by Gousios et al. [2], but it helps to identify false-negatives.

Table I provides numeric information about our data. The data used in this work is available for replication purposes at the companion website.[8] We conducted the data collection in October 2017. In general, there are around five commits for every pull request.

TABLE I: Information about our dataset (per language)

	# Pull Requests			
	Merged	Unmerged	Total	# Commits
C	16,793	5,937	22,730	185,738
C++	40,253	10,620	50,873	263,603
Clojure	2,914	657	3,571	25,966
CoffeeScript	7,449	2,515	9,964	45,583
Erlang	3,126	787	3,913	41,700
Go	16,604	3,228	19,832	68,867
Haskell	6,049	1,068	7,117	50,936
Java	23,245	5,601	28,846	118,020
JavaScript	33,997	14,458	48,455	131,512
Objective-C	6,813	2,385	9,198	37,923
Perl	2,218	768	2,986	28,965
PHP	24,357	8,268	32,625	142,307
Python	21,324	6,724	28,048	118,406
Ruby	64,101	14,549	78,650	228,909
Scala	17,772	2,923	20,695	88,670
Typescript	6,277	956	7,233	71,675
Total	293,292	81,444	374,736	1,648,780

B. Data analysis

In this section, we discuss how do we analyze our data.

1) Avoiding false-negatives: The first step in our analysis was to improve the misclassification of "unmerged" pull requests. This step is important since we perceived that although these pull requests appear unmerged, they were actually merged. After a manual analysis, we noticed that they were, indeed, merged. We then worked to avoid potential false-negative pull requests. To determine casual contributors using the pull request approach, we first applied the heuristics previously mentioned in Section IV-A2 to all "unmerged" pull requests. For the cases in which we succeed in identifying merged contributions in the unmerged sea (61,529 of 143,099 unmerged pull requests, 42%), we flagged them with the heuristic that helped us to spot the false-negatives. We then selected all known merged pull requests and defined the casual contributors as those with one single pull request accepted.

2) Manual Analysis: As in Pinto's study [5], we performed a manual analysis of the data obtained. To answer RQ2, we investigated the content of the contributions made by casual contributors to understand their intended goal. To do so, we selected a statistically significant sample of 384 contributions (95% confidence and 5% margin of error)) performed in 263 different projects. This sample was randomly selected considering all the pull requests in our population. For each contribution, we studied the pull request title and pull request description and discussion, in addition to the commit message and code changes. We used Pinto's categories from the original study [5] to classify our pull requests and make a fair comparison.

Next, we intended to compare the effectiveness of these two approaches (commits and pull requests). To do so, we investigated a sample of 442 contributors (95% confidence and 5% margin of error) from our sample. We manually classified these contributors into casual (when we identified only a single contribution), non-casual (when the developer had more than one contribution merged to the repository) or non-contributor (when no contribution was accepted at all). To analyze the contributions and assure whether the contribution had been accepted or not, we carefully investigated the commits made by the user in the master branch of the project, the pull requests submitted, and the discussions of the pull request. After manually classifying the sample, we compared this classification to the results obtained by using commits and pull requests. Two researchers performed both manual analyses. They worked independently and, when necessary, resolved any discrepancies after a discussion meeting with a third researcher. All the contributions in the sample had been analyzed and discussed, and a consensus was reached for all of them.

3) Statistical Analysis: We performed statistical tests to compare the percentage of casual contributors identified using

[7]https://github.com/angular/angular.js/pull/9158

[8]https://markaumvb.github.io/RENE/

pull requests and commits. After confirming that the data did not follow a normal distribution for any project—by applying the Shapiro-Wilk normality test—we applied the Wilcoxon signed-rank test [29] to compare the percentage of casual contributors per project using commits and pull requests. We also applied Cliff's delta [30], an effect size measure, to know the mean distance between the samples. The results are interpreted using the thresholds provided in Romano et al. [31], i.e.,$delta < 0.147$ (*negligible*), $delta < 0.33$ (*small*), $delta < 0.474$ (*medium*), and $delta >= 0.474$ (*large*).

V. Study Results

In this section, we report the results of our replication study, answering each research question. We also discuss the results comparing them to the original study.

A. RQ1. How common are casual contributors?

As in the original study, we analyzed the number of contributions made per contributor in our sample. The histograms presented in Figure 1 show the overall picture of the number of pull requests made by contributor, considering all the projects in our sample (contributions per contributor: median=1, mean=4.6, Q3=2, std. deviation=33.27). Our first finding sharply aligns with the original study (histogram on the left of the figure): regardless of the programming language, few contributors are responsible for the majority of the pull requests, while most developers perform very few pull requests. The same is true of the commits approach, where most contributors just commit once.

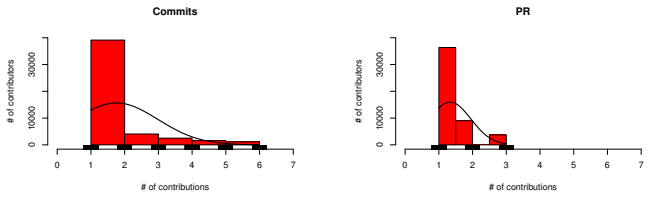

Fig. 1: Distribution of contributions and contributors in the nalyzed projects. Outliers were removed to ease visualization.

As expected, we noticed that, by using pull requests, we found a larger (relative) number of not only casual contributors but also contributions made by these contributors, compared to the original study.

Contributions per project per programming language. To better understand the landscape, in Figure 3 we present an overview of the casual contributors per project in the form of boxplots (removing the outliers to ease visualization). Each boxplot represents the data from all projects analyzed, grouped by their main programming language. One of the first noticeable observations from this figure is that by using pull requests the ratio of casual contributors is higher regardless of the programming language. The difference can be as small as 8.43% (Ruby), or as high 22.16%, which is the case of projects written in TypeScript. For every project in this particular language, we identified a higher percentage of casual contributors

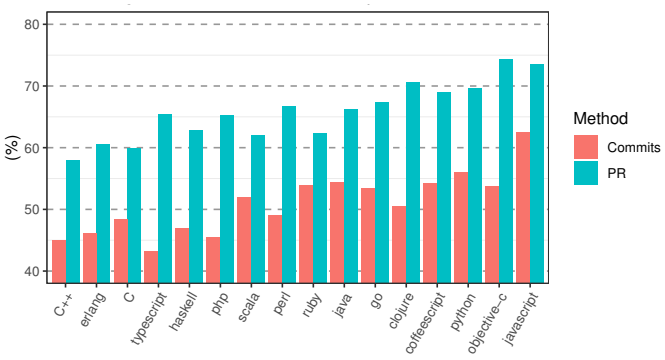

Fig. 2: Percentage of casual contributors per prog. language

when using pull requests than when using commits. That is, the approach that one uses to mine contribution data has a non-trivial influence on the results.

Overall results. We found that 66.02% of the contributors of the analyzed projects had a single pull request merged (to a given project); thus, they had been classified as a casual contributor. Compared to the number presented in the original study [5] (48.98%), this represents an increase of ∼35%. To make a fair comparison, the percentage of casual contributors found analyzing commits for the same sample and period used for the pull request analysis (until October 2017) was 52.88% at the commit level. Thus, the difference of analyzing pull requests is far from negligible (∼25%).

At project level. By looking deeper—at project level—at the extremes of our sample we found nine projects with more than 90% casual contributors. On the other side of the spectrum, we found 18 projects with less than 50% casual contributors (two of them with less than 35%). This finding is particularly interesting: the projects that have the least number of casual contributors, still have a non-negligible amount of them. Besides, eight of the 18 projects with the least casual contributors were written in C or C++ (4 each), and other 4 in Scala. This maybe explains the initiatives from Scala Center to attract more contributors.[9]

More concretely, we noticed that the projects among the 20% with smaller percentage of casuals (ranging between ∼7% to 10%) include large, consolidated projects such as ANGULAR/ANGULAR-JS, NODE/NODE-JS, BITCOIN/BITCOIN, SPREE/SPREE, ELASTIC/ELASTICSEARCH, and HOMEBREW/HOMEBREW-CASK. On the opposite side, for the projects among the 20% with greater variation (ranging from 24% to 84%), we made two interesting observations: (i) they comprise newer/less popular projects (such as ENGELBERG/INSTAPARSE, MOJOJOLO/TEXTTEASER, WINJS/WINJS); (ii) 12 projects were written in Clojure and 7 were written in Typescript. This may indicate that newer projects and those developed using new or trending languages adhere more to the pull-based model, making more extensive use of pull request as a unit composed of many commits that represent a single contribution.

[9]https://github.com/scalacenter/sprees

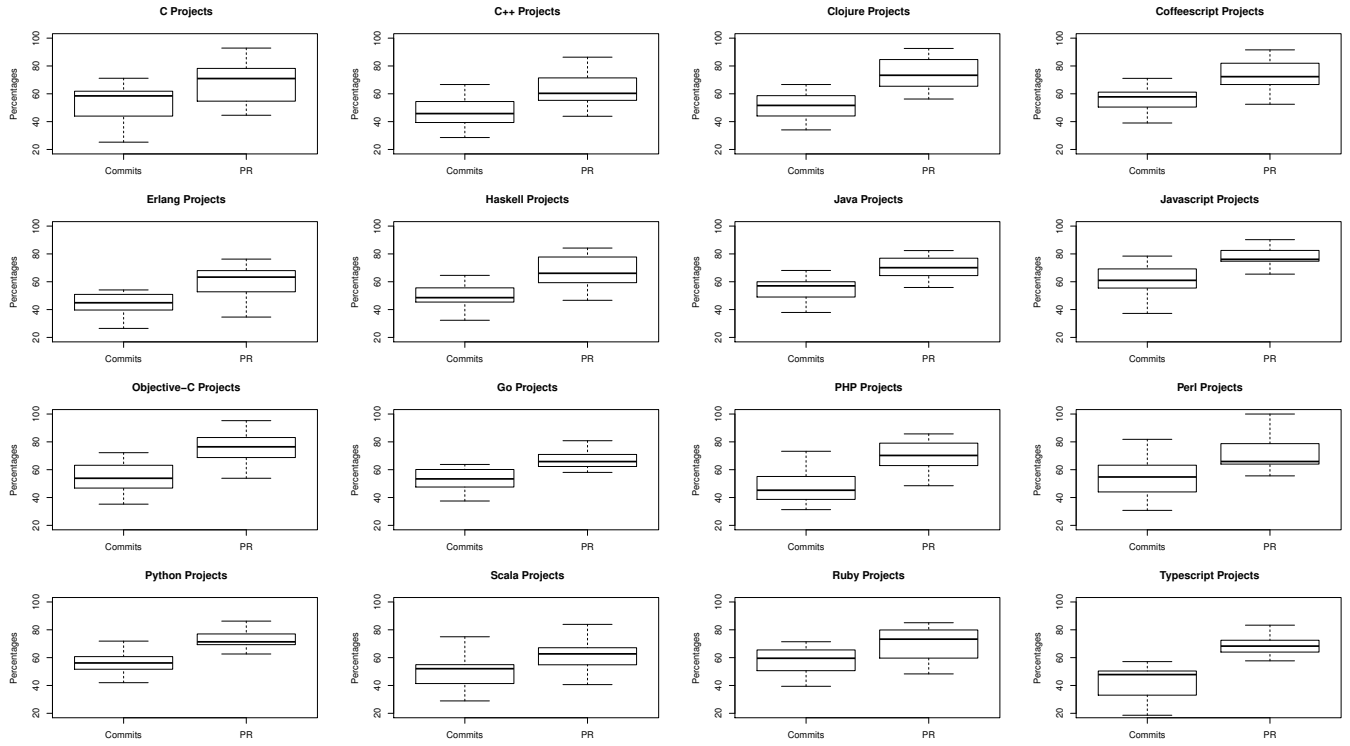

Fig. 3: Distribution of casual contributors per project (grouped by programming language). The boxplots on the left indicate commits, whereas the boxplots on the right indicate pull requests.

When we look at the number of pull requests, we observe that casual contributors submitted 12.5% of the total contributions made to the projects. This number is higher than the proportion presented in the original study [5] (1.73%). The result using the commit data collected for this replication (same period as the data used for the pull requests), showed a result close to that reported in the original study (contributors with a single commit had submitted 1.83% of the total commits). The higher percentage of "contributions" in the form of pull requests was expected because: (1) there is a higher percentage of casual contributors identified when using pull requests (Fig. 3) and (2) the number of pull requests is, in general, one order of magnitude smaller than the number of commits (see Table I) because pull requests may contain multiple commits.

By analyzing the percentage of pull requests made by casual contributors per project, it was possible to notice a sparse distribution. While we have 13 projects with casual contributors responsible for more than 65% of the pull requests, the other 15 accounts presented less than 5% of pull requests from casual contributors. The projects FACEBOOK/SHIMMER and WEAVEJESTER/COMPOJURE respectively received 90.91% and 86.21% of their contributions from casual contributors, while in SCALA-JS/SCALA-JS and MOZILLA/SHUMWAY contributions from the casuals account for 1.82% and 2.21%.

RQ1 Summary: We provided additional evidence that casual contributors are rather common; the number of casual contributors is non-negligible, averaging 66% in our sample, and reaching more than 90% in some projects. Moreover, the number of pull requests made by the casual contributors average 12.5% and reach more than 50% in several projects. Compared to the commit level approach [5], we see an increase both in terms of contributors and the number of patches submitted when analyzing the phenomenon at the pull request level.

B. RQ2. What are the characteristics of contributions made by casual contributors?

In this section, we analyze the number of files changed and the number of additions and deletions performed by the casual contributors in their pull requests.

Quantitative Analysis. Each pull request has, on average, 1,804 lines added (Q1=2; median=11; Q3=55), 890 lines deleted (Q1=1; median=3; Q3=16), 16.78 files changed (Q1=1; median=2; Q3=4), and 14.37 commits (Q1=1; median=1; Q3=2). Figure 4 presents the distribution of the number of additions, deletions, changed files, and commits per pull request comparing casual and non-casual contributors. To ease visualization, we filtered out the outliers from the figures. These figures show interesting patterns. First, when focusing on the medians and upper quartiles, contributions from casuals are smaller (i.e., they have fewer additions and deletions and touch fewer files).

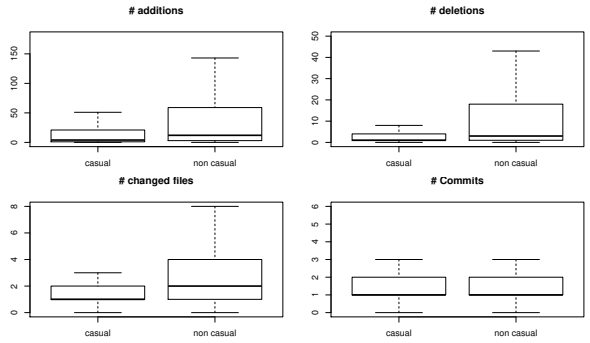

Fig. 4: Characteristics of the pull requests

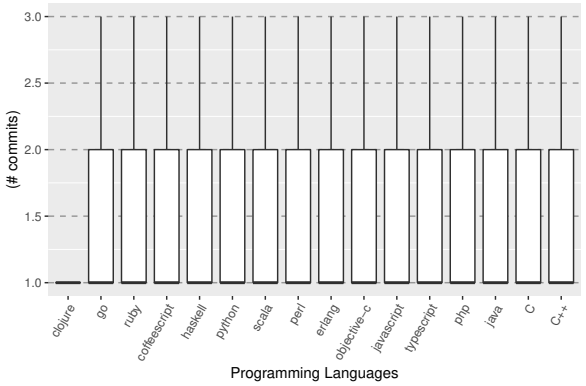

Fig. 5: Number of commits per pull request made by casual contributors

Second, as pointed out in the original study [5], a non-negligible part of the contributions presents multiple files changed, with multiple additions and deletions. Also, both groups (casuals and non-casuals) perform a similar number of commits. In particular, the median number of commits per pull request is one (two is the third quartile, and three is the maximum). In fact, 70.5% of the pull requests comprised only one commit (and only 14.7% comprised more than two commits). This result is especially aligned with the finding of Gousios and colleagues, who observed that the majority of the pull requests comprise a single commit [2], suggesting that these pull requests may not be the subject of a thorough code review cycle (which may incur in additional commits to accommodate the changes requested). In a manual analysis over a small sample of 20 pull requests with only one commit, we observed that only three of them had code reviews.

Figure 5 shows the distribution of the number of commits per pull requests made by casual contributors per language (without outliers). We found that the distribution is the same for all languages (median=1, upper quartile=2) but Clojure, which has a flat boxplot (with median and the upper quartile = 1). When taking a closer look at the outliers, we observed that 3,930 ($\sim$9%) of the 42,092 pull requests have more than three commits, which correspond to 112,056 out of 160,970 (69.61%) commits within casual contributors pull requests.

Qualitative Analysis. To further investigate the goal of the contributions placed by casual contributors, we manually analyzed a sample of 384 pull requests made by casual contributors as we described in Section IV-B. Table II summarizes the results of the manual analysis and the comparison with the original study. Besides a few differences, it is possible to notice that the types of contributions made by casual contributors are in line with those found in the original study. In particular, we found a remarkable similarity in the category: "bug fix". In this category, both the absolute and percentage numbers matched the ones in the original study. Additionally, we found contributions ranging from simple typos[10] to fixes that require expertise in given topics, like a fix sent to XBMC project[11] that resolves a video glitch when adjusting the volume from the XBMC Android remote, which requires knowledge about cache and Linux kernel.

As for the differences, we found 12 contributions that could not be classified according to the original categories, so we labeled them as "Others." Examples of "Others" include, creating script utility for creating an rpm file,[12] enhancing the Continuous Integration configuration,[13] and bulking merged commits into a given version branch.[14]

TABLE II: The categorization of the contributions made by casual contributors.

Category	Pull requests #	Pull requests %	Pinto et al. #	Pinto et al. %
Bug Fix	116	30.20%	116	30.20%
Documentation	99	25.78%	110	28.64%
Add New Feature	39	10.10%	72	18.75%
Refactoring	46	11.98%	34	8.85%
Update Version/Dependencies	37	9.64%	25	6.51%
Improve Error/Help Messages	19	4.95%	14	3.64%
Improve Resource Usage	5	1.30%	8	2.08%
Add/Fix Test Cases	12	3.12%	5	1.30%
Other	12	3.12%	–	–

RQ2 Summary: We obtained similar results compared to the original study when we categorized the casual contributors' contributions: most of the contributions are bug fixes and documentation. Casual contributors' pull requests are, in general, smaller than non-casual contributors', in terms of lines added, removed, and files changed. However, casual contributors also make significant contributions.

VI. Comparing the results

According to Carver [26], one of the main values of a replication study is the comparison of its results with the results of the original study. Table III summarizes the main

[10]https://github.com/thoughtbot/paperclip/pull/822

[11]https://github.com/xbmc/xbmc/pull/2065

[12]https://github.com/ariya/phantomjs/pull/342

[13]https://github.com/tornadoweb/tornado/pull/539

[14]https://github.com/trinitycore/TrinityCore/pull/13703

findings of both studies. We also provide a comparison in terms of (1) consistent results and (2) differences in results.

Consistent results. Some of the findings that aligned with the original study include: (1) a non-negligible number of casual contributors—comparing the number of casuals obtained, at the project level, we had 259 projects in which the percentage of casual contributors was higher using the pull requests, versus only four projects when using the commits. An example of this is the project NINENINES/COWBOY, in which by selecting casual contributors using the commits we found that 70,33% of the contributors were casual; while by using pull requests we found 68.00%. (2) Contributions made by casual contributors are not trivial. Some require an in-depth knowledge of the source code and technologies involved. Additionally, many contributions touch different files and deal with improvements in resource usage and refactoring. (3) Although many contributions relate to documentation (typo/grammar fixes, adding links, changing headers), most of the contributions are related to code (bug fixes, adding new features, refactoring).

Different results. We also perceived some differences between the two studies. As it is possible to notice by going through the results, the main difference of the results of the original study and this replication revolves around the size of the casual contributors' population. Using pull requests, the average percentage of casual contributors increased by more than 30%. We confirmed this by running a Wilcoxon signed-rank test for every programming language, followed by a Cliff's delta test to assess the effect size. All the differences are, indeed, significant (all p-values$<$0.05), except for the projects written in C, for which the difference was marginally significant (p-value=0.06). For all cases, we found a large effect size according to the Cliff's delta (all $>$0.5).

VII. Discussion

Although we found about 30% more casual contributors when using the pull request approach, this does not mean that this replication outperforms the original study by any means. To shed additional light on this matter, we compared how effective each approach is in terms of correctly identifying the contributors as casual or not. To do so, we manually investigated a sample of 442 contributors (golden-set) that we identified as having a merged pull request to one of the studied projects. We found 50 (11.3%) cases of users misclassified using the pull request level, versus 116 (26.2%) cases when using commits. We present more detail in Table IV, and discuss some of the results in the following subsection.

The fact that we observed more casual contributors or contributions is not the main outcome of this replication. Rather, the main point is to show how different the results can be when analyzing contributions using different concepts of contributions. For this specific case, we saw that using pull requests is more precise, given that GitHub projects mostly follow the pull-based process—in which one "contribution" maps to one "pull request."

A. Lessons Learned

In this section, we report some lessons that we learned during the journey of this research analyzing both commits and pull requests. We hope others may also learn from them and, as a consequence, mitigate the problems that they introduce.

GitHub contributors without commits. The first interesting observation from this analysis was that 109 of the contributors manually analyzed (24.7%) did not have any commits authored or identified by their GitHub usernames. In fact, 82 of them had some contributions accepted via pull requests (in 22 cases more than one pull request). By analyzing the commits under those pull requests, we found that the root cause is that the contributors have made commits using an e-mail address different from the one linked to their GitHub account—which leads to an unknown user in the GitHub system—or included third-party commits. When analyzing the pull request level, this is usually avoided because the pull requests retain the developers' usernames. Exceptions may occur, for example, when users remove their account.

Commits may miss casual contributors. The commits approach led us to miss 30 casual contributors (6.8% of our sample). This happened since these contributors had more than one commit in a single pull request. This was an expected drawback of the commits approach, since pull requests are designed to group commits related to the same contribution—reducing the issues with different commit habits [32]. This is clear in cases like the example identified on IPYTHON/IPYTHON project.[15] In such cases, pull requests capture the casual contributors' phenomenon better than commits. This is the main reason why analyzing at the pull request level may be more precise when the number of contributions is central to the research.

The pull requests heuristics' also have flaws. Although our heuristics helped us to uncover 61,529 out of 143,099 (43%) pull requests flagged as unmerged but in fact potentially merged, we still found some misclassifications. For example, we found 15 actual casual contributors (3.4%) that had been classified as non-casual when using the pull requests. Also, we identified 27 developers (6.1%) with no commits or pull requests merged (non-contributors). This happened because the heuristics applied to the unmerged pull requests identified some false-positives (e.g., https://github.com/spree/spree/pull/938. Although the employed heuristics led to false-positives, we highly encourage their use when analyzing pull requests. However, this also suggests that there is still room to improve the heuristics. By delving deep into the results provided by the heuristics, we found that the "use of keywords that suggest merging activities right after closing the pull requests" causes most of the false-negatives. The "merge" keyword had a higher incidence, but it depends on the context in which it applies (e.g., "not agree to merge"). On the other hand, using the heuristics "SHAs" and "commits in the master branch" is more precision (we did not find any misclassification during our manual analysis).

Developers could use commits *and* pull requests. We identified eight false-positives that had been classified as casual contributors using pull requests, but that had more than one

[15] https://github.com/ipython/ipython/pull/4302

TABLE III: Summary of the results of both studies.

	Original Study	**Replication Study**
Method	Commits	Pull Requests
Sample Size	275 Projects	263 Projects
Findings	**RQ1**: Pinto et al. [5] found that 48.98% of the contributors analyzed were casual contributors, who were responsible for 1.73% of the total contributions of the analyzed projects. **RQ2**: After manually inspecting a sample of contributions, Pinto et al. [5] found that 28.64% were related to documentation, 30.20% of them fix bugs, 18.75% propose new features, and 8.85% refactor code.	**RQ1**: by using pull requests, we found that the overall number of casual contributors is 66% in our sample (more than 90% in some projects). Casual contributors are responsible for 12.5% of the total pull requests in our sample. **RQ2**: By analyzing pull requests made by casual contributors, we found that 25.78% are related to documentation, 30.20% fix bugs, 11.98% refactor code, and 10.10% add new features. We also found pull requests enhancing CI configuration and bulking previously merged commits into another branch.

TABLE IV: Manual analysis of casual contributors.

Manual	**Pull requests**		**Commits**		
Classif	**Casual**	**Non-casual**	**Casual**	**Non-casual**	N/A
Casual	251	15	176	30	60
Non-Casual	8	141	4	123	22
Non-Contributor	26	1	–	–	27
Total	285	157	180	153	109

accepted contribution. The main reason for this is that the developers had commits submitted directly to the repository, without opening a pull request. This happened in at least two identified scenarios: (i) contributions made via git that skipped the pull request flow;[16] (ii) commits made before the migration to GitHub.[17] One way to mitigate this situation would be to follow a mixed-approach: first analyzing pull requests, and then checking the commits made by the same user (but not included in any pull request). On the one hand, this process may bring some drawbacks of using commits; on the other hand, it may provide a more comprehensive analysis.

Git brings a certain messiness to the analysis. When performing our manual analysis, we observed that project maintainers employ git commands such as rebase (i.e., rewrites the commit history) and cherry-pick (i.e., does not alter the existing history; instead, it adds to the history) that alter the structure of a commit. These operations have the potential to exclude commits (with rebase) or skip commits (when manually copying contributions), which in turn hinders the analysis of contributions made by casual contributors. Many of these cases had been captured by the heuristics applied to pull requests originally classified as unmerged.

Developers can also bring messiness. In one example, a developer had a pull request merged into the master branch and was requested to open another pull request targeting the current stable release branch (which was also merged). Therefore, the developer submitted the same commit to two pull requests; henceforth, when we analyzed the commits to master, we found only one, but by analyzing the number of pull requests merged, we found two. We also found contributions *copied* into a different commit and manually added via command line.

[16]https://github.com/cocos2d/cocos2d-x/commits?author=zhukaixy

[17] https://github.com/fzaninotto/Faker/commits?author=paulvalla

VIII. Threats to Validity

Like any empirical study, our research presents some threats to validity. First, it is limited to OSS projects hosted on GitHub. Although we studied hundreds of them, we did not explore all possible OSS projects available online that use the pull-based model. Moreover, we tried to use the same projects investigated in the original study. However, some of these projects became inactive or were not found at all. As a consequence, our number of studied projects differs a bit from the original study (275 in the original study and 263 in our research). We do not expect significant differences in the results due to the projects not reused in this research.

Moreover, the original study relied on data gathered via `git log` utility to conduct its analysis. In our study, we opted to use the GitHub API and GHTorrent, which is more reliable when it comes to disambiguated contributor identification. However, this benefit comes with a challenge: when fetching data from the GitHub API, we observed that some commits did not have the author identification. This particular threat happened because one GitHub contributor can do commits using an unknown git configuration (which prevents GitHub from linking the commit to the username). To mitigate this threat, we removed the commits without user identification from our research. Regarding the use of the GitHub API, however, one may argue that this decision may introduce some noise on the commit data when compared to the original study. Moreover, we kept only those projects that used the pull request workflow, discarding those that do not accept pull requests (see Sect. IV-A1). We observed, though, that the results using commits sharply align with the original study. For example, the percentage of contributions by casual contributors was 1.73% in the original study, and we found 1.83% using the GitHub API with data from 2 years later. This strengthens the confidence in the results of the original study.

We also acknowledge that the method of using pull requests does not take into account the fact that there may be open pull requests from a casual contributor. This may influence the number of classification, given that the developers may have other pull requests that waiting for review (still open) at the moment of the collection.

Similar to projects that become inactive, GitHub users can also become inactive or change their usernames, which could introduce biases to our analysis. Another threat relates to contributors that open a pull request with one user account, but make commits with another user account (this problem

was reported as "GitHub contributors without commits" in Section VII-A). Due to the scale of our study (i.e., more than 293,292 pull requests and 1,028,172 commits were analyzed), we were unable to verify or fix these concerns. Still regarding the pull requests, we observed that although a pull request can be found in the API, it could be deleted from the repository, creating inconsistencies in the dataset. Finally, we downloaded our data in October 2017. Since the studied projects are still under evolution, we expect that the overall number of pull request increased during 2018–2019 time window. We do not expect, however, changes in the contributing behavior, like the ones we highlighted throughout this study.

IX. Conclusions

In this paper, we replicated a study on casual contributors [5], a recent phenomenon in OSS communities. To conduct this replication, we used the same subjects, and the same method used to infer casual contributors—the commit method. Additionally, we used a second method—the pull request method. Our intention with the use of pull request was to understand whether different methods used for the same purpose could yield the same results.

Although the results may at first glance seem obvious, existing studies using commits may overlook the precise picture of casual contributors in OSS projects. Moreover, the results may be useful not only for the context of casual contributors, but also for researchers analyzing other aspects of OSS communities tied to the number of contributions that may be impacted by the concept of contribution.

Among the findings, we found that with the use of pull requests to find casual contributors the number of casual contributors is higher than those reported with the commit method. Moreover, by manually analyzing a set of contributors, we evidenced that using pull requests results in a smaller number of false-negatives, making pull requests better than commits for capturing the casual contributors' phenomenon. With this replication, we also thereby expect to benefit other researchers toward designing better mining approaches.

References

[1] 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.

[2] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *ICSE 2014*. ACM, 2014, pp. 345–355.

[3] I. S. Wiese, J. T. da Silva, I. Steinmacher, C. Treude, and M. A. Gerosa, "Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant," in *ICSME 2016*, Oct 2016, pp. 345–355.

[4] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[5] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *SANER 2016*, 2016, pp. 112–123.

[6] A. Lee and J. C. Carver, "Are one-time contributors different?: A comparison to core and periphery developers in floss repositories," in *ESEM 2017*, 2017, pp. 1–10.

[7] A. Lee, "One-time contributors to floss: Surveys and data analysis," *SIGSOFT Softw. Eng. Notes*, vol. 43, no. 1, pp. 1–6, Mar. 2018.

[8] A. Barcomb, "Episodic volunteering in open source communities," in *20th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2016, p. 3.

[9] 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.

[10] C. Miller, D. Widder, C. Kästner, and B. Vasilescu, "Why do people give up FLOSSing? a study of contributor disengagement in open source," in *International Conference on Open Source Systems*. Springer, 2019.

[11] G. Iaffaldano, I. Steinmacher, F. Calefato, M. Gerosa, and F. Lanubile, "Why do developers take breaks from contributingto oss projects? a preliminary analysis," in *SoHeal 2019*, 2019, p. 8pp.

[12] S. da Silva Amorim, J. D. McGregor, E. S. de Almeida, and C. von Flach G. Chavez, "Educating to achieve healthy open source ecosystems," in *ECSA 2018*, 2018, pp. 28:1–28:7.

[13] 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.

[14] "Github developers," accessed: 2018-05-10. [Online]. Available: https://developer.github.com/v3/

[15] G. Gousios, "The GHTorrent dataset and tool suite," in *MSR 2013*, 2013, pp. 233–236.

[16] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. Germán, and P. T. Devanbu, "The promises and perils of mining git," in *MSR 2009*, 2009, pp. 1–10.

[17] J. Miller, "Replicating software engineering experiments: A poisoned chalice or the holy grail," *Inf. Softw. Technol.*, vol. 47, no. 4, pp. 233–244, Mar. 2005.

[18] A. Lee, J. C. Carver, and A. Bosu, "Understanding the impressions, motivations, and barriers of one time code contributors to floss projects: A survey," in *ICSE 2017*, 2017, pp. 187–197.

[19] Y. Lu, X. Mao, Z. Li, Y. Zhang, T. Wang, and G. Yin, "Does the role matter? an investigation of the code quality of casual contributors in github," in *APSEC 2016*. IEEE, 2016, pp. 49–56.

[20] A. Barcomb, A. Kaufmann, D. Riehle, K.-J. Stol, and B. Fitzgerald, "Uncovering the periphery: A qualitative survey of episodic volunteering in free/libre and open source software communities," *IEEE Transactions on Software Engineering*, 2018.

[21] M. Rebouças, R. O. Santos, G. Pinto, and F. Castor, "How does contributors' involvement influence the build status of an open-source software project?" in *MSR 2017*, 2017, pp. 475–478.

[22] R. M. M. Bezerra, F. Q. B. da Silva, A. M. Santana, C. V. C. Magalhaes, and R. E. S. Santos, "Replication of empirical studies in software engineering: An update of a systematic mapping study," in *ESEM 2015*, Oct 2015, pp. 1–4.

[23] D. T. Campbell, "Experimental and quasi-experimental designs for research on teaching," *Handbook of Research on Teaching*, vol. 5, pp. 171–246, 1963.

[24] T. T. Dinh-Trong and J. M. Bieman, "The freebsd project: A replication case study of open source development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, 2005.

[25] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *Empirical Softw. Engg.*, vol. 13, no. 2, pp. 211–218, Apr. 2008.

[26] J. C. Carver, "Towards reporting guidelines for experimental replications: A proposal," in *1st international workshop on replication in empirical software engineering*. Citeseer, 2010, pp. 2–5.

[27] M. Shepperd, N. Ajienka, and S. Counsell, "The role and value of replication in empirical software engineering results," *Information and Software Technology*, vol. 99, pp. 120 – 132, 2018.

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

[29] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[30] R. J. Grissom and J. J. Kim, *Effect sizes for research: Univariate and multivariate applications*. Routledge, 2012.

[31] J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?" in *Annual Meeting of the Florida Association of Institutional Research*, 2006, pp. 1–3.

[32] K. Herzig, S. Just, and A. Zeller, "The impact of tangled code changes on defect prediction models," *Empirical Software Engineering*, vol. 21, no. 2, pp. 303–336, Apr 2016.