Branch Use in Practice

A Large-Scale Empirical Study of 2,923 Projects on GitHub

Weiqin Zou*, Weiqiang Zhang*, Xin Xia†, Reid Holmes‡, Zhenyu Chen*
*State Key Lab of Novel Software Technology, Nanjing University, China
†Faculty of Information Technology, Monash University, Australia
‡Department of Computer Science, University of British Columbia, Canada
wqzou, zhangweiqiang@smail.nju.edu.cn, xin.xia@monash.edu, rtholmes@cs.ubc.ca, zychen@nju.edu.cn

***Abstract*—Branching is often used to help developers work in parallel during distributed software development. Previous studies have examined branch usage in practice. However, most studies perform branch analysis on industrial projects or only a small number of open source software (OSS) systems. There are no broad examinations of how branches are used across OSS communities. Due to the rapidly increasing popularity of collaboration in OSS projects, it is important to gain insights into the practice of branch usage in these communities. In this paper, we performed an empirical study on branch usage for 2,923 projects developed on GitHub. Our work mainly studies the way developers use branches and the effects of branching on the overall productivity of these projects. Our results show that: 1) Most projects use a few branches (<5) during development; 2) Large scale projects tend to use more branches than small scale projects. 3) Branches are mainly used to implement new features, conduct version iteration, and fix bugs. 4) Almost all master branches have been requested by contributors to merge their contributions; 5) There always exists a branch playing a more important role in merging contributions than other branches; 6) Almost all commits of more than 75% branches are included in the master branches; 7) The number of branches used in a project has a positive effect on a project's productivity but the effect size is small, and there is no statistically significantly difference between personal projects and organizational projects.**

***Index Terms*—branch use, GitHub, exploratory study**

I. Introduction

Along with the rapid growth of both project scale and team size in modern software development, there comes an important and challenging problem: enabling developers to collaborate and develop projects in parallel without interfering with one another [5]. One common method to address this problem is with branches within version control systems [3]. Many advanced version control systems, such as Git[1] and SVN[2], have provided good support for the feature of branches.

When developers plan to perform specific tasks such as bug fixing or feature implementation without affecting the main stream development, they often create a branch. Then they will work on this new branch independently without interfering with other developers. After they finish coding and testing, they then merge their branch that they were changing back into the branch they originally branched from, or they invite another developer to help perform the merge for them [32]. In this way, branching makes it possible for developers to work on their own workspace without being disturbed or disturbing others unnecessarily.

With the above benefits of branching, many OSS projects (such as Python) and commercial companies (such as Microsoft) adopt branching strategies to facilitate the process of software development [5]. However, branching has a cost. Some developers do not fully understand branching and abusing of branches can hinder development [1]. This can result in problems such as integration failures and release delays if branches are used incorrectly.

To help developers better use branches, some researchers have created branching best practices [55], [40], [1]. Others try to learn the branch usage in practice and its potential impact on software development [43], [39]. There also exist approaches that propose solutions to problems introduced by using branches [5], [36]. Unfortunately, most existing research studies are either largely based on researchers' own experience, are targeted at a small number of OSS projects, or are limited to individual industrial projects. To the best of our knowledge, there are no large-scale empirical studies on developers' behaviours of branch usage in practice. As such, we do not yet have an overview about how branches are used in practice across the breadth of OSS communities.

Fortunately, GitHub[3] makes it possible to deeply investigate branch usage across a large number of practical projects. GitHub is a platform based on Git, which provides code hosting and distributed collaboration [26]. As of January 2017 more than 50 million repositories were hosted on GitHub.

In this paper, we investigated the current state of branch usage in OSS communities. Specifically, we conducted an empirical analysis on 2,923 GitHub projects that have been developed over at least five years. We first obtained an overview of branch usage in GitHub. Then we studied the purposes of branching. Next, we investigated the roles that branches take in coping with contributions by others. We also studied the commits flowing from non-master branches into master branches. Finally, we studied the impact of using branches on the overall productivity of projects. Our major contributions are listed as follows:

[1]http://git-scm.com/
[2]http://subversion.apache.org/
[3]http://github.com/

- We conduct a large-scale empirical analysis on branch usage on OSS communities. In total, we investigate 2,923 projects on GitHub. Our study sheds light into the practice of branching, and the large number of projects can improve the generalizability of our results.
- We investigate several aspects of branch usage practice on GitHub, including the number of branches used, branching purpose, the roles of branches in coping with contributions by others, the commit flow, and the impact of using branches on the overall productivity of projects. Our findings increase the understanding of branching in practice.

II. Research Questions

In this paper, we aim to understand how branches are used in real OSS communities. Specifically, we seek to understand the characteristics of branch usage and its potential impact on project productivity. In order to achieve these goals, we study the following five research questions.

RQ1. How many branches does a project have and what kind of projects are more likely to have more branches?

Although some researchers have conducted a series of studies on branching [5], [43], to the best of our knowledge, there are no broad investigations into the branching practices of OSS communities. In this RQ, we try to gain some insights into how prevalently branches are used in GitHub. We further try to find out what kind of projects tend to use branches more than others. The answer of RQ1 helps to gain a better understanding of current situation of branch usage in large-scale OSS communities.

RQ2. What are branches used for?

Philips et al. [39] studied the purposes of branching in industry by surveying 140 practitioners. They found that branching for release and feature implementation are the most two most popular branch use types. Survey results represent the belief of participants, which are primarily formed based on personal experience, rather than on evidence found in empirical research [14]. In this RQ, to complement Philips et al.'s study, we try to understand why developers create branches in GitHub by analyzing data collected from OSS communities. The answer to this RQ can help us gain a better understanding about the branching practice in OSS communities. It can also provide a good complement to existing industrial studies, thus to make people hold a more complete view over branching in practice.

RQ3. How many branches are requested to merge PRs? On GitHub, developers can contribute to any project via pull requests (PRs) [20]. Developers only need to fork the targeted project and modify code in the forked repository. If they finish coding and want to merge their code change into the targeted project, they can submit a PR. When submitting a PR, they have to specify which branch they want to merge the PR into.

Although contributors can submit any PR to an arbitrary branch as they wish, however, when we read contribution documents of projects in GitHub, we found that different projects have different requirements on which branch the contributors should push PRs into. For example, some projects may allow contributors to submit PRs to master branch while some projects forbid this practice. For example, project `orientechnologies/orientdb` [4] explicitly requires contributors to never submit PRs to their master branch because the master branch is only used for releasing [5].

Considering the arbitrariness of specifying targeted branch by contributors and the specific requirement by projects, it is important to get a macro view of the current situation of branches that are requested by contributors to merge their PRs. In this RQ, we study the role of different branches in terms of receiving outside contributions from PRs. Our results can be an initial basis for practitioners or researchers to help contributors submit their PRs to the right branches more easily.

RQ4. Do all commits go into the master branch? Generally speaking, developers tend to create branches from the master branch to complete specific tasks such as bug fixing or feature implementation. When they finish their work on the branch, they may want to merge their work (in the form of commits) into the master branch [9]. This inspires us to figure out whether all such commits flow into the master branch. In cases where this does not happen, we study how many commits are left alone on the unmerged branch itself.

The answer of RQ4 can gain a high-level view of the current commit flow of non-master branches in the whole OSS community. Besides, for developers of a project, knowing detailed commit flow of non-master branches can also help them understand what is going on within their project, which can help better manage work on their branches. For example, they may timely merge useful commits from non-master branches into their master branches.

RQ5. Does branch usage affect the overall productivity of a project? Branching is proposed to help developers focus on their own work to avoid unnecessary interruptions by others [9]. In intuition, branching can facilitate the process of developing software. Nevertheless, little work has been done to check the potential impact of branching on software development in OSS communities. In this RQ, we investigate the effect of branch usage on project productivity. Specifically, we aim to determine whether the number of branches used has an impact on a project's productivity. The answer of this RQ can help better identify the functionality of branch use.

III. Methodology

In this section, we describe three aspects of our methodology: project selection, RQ design, and RQ statistical method selection.

A. Studied Projects

Our experimental subjects are selected from projects in GitHub. We select projects that have been existed on for at

[4] http://github.com/orientechnologies/orientdb

[5] http://orientdb.com/docs/last/Contribute-to-OrientDB.html

TABLE I
SUMMARY OF ATTRIBUTES ON 2,923 GITHUB PROJECTS.

Attribute	Min	Median	Max	Mean	St. Dev.
Age (years)	5.0	6.8	8.9	6.9	0.7
Commits	100	422	25,052	918.3	1,483.2
Total PRs	1	32	4,418	91.1	192.5
Total issues	1	47	3,465	142.4	284.5
Size (MB)	0.1	2.0	928.4	12.4	42.9
Developers	3	15	690	29.8	45.2

least five years (last update timestamp minus project creation timestamp) by January of 2016. This condition ensures that the projects are long-lived. With this criterion, we were left with 431,879 projects. Among those projects, we only consider *non-forked* projects (i.e., projects that are not cloned from other projects), as forked projects may bring bias to our conclusions with their duplicate code and documents. With this condition, only 304,694 projects were left.

We also filter out projects with fewer than 100 commits to avoid small scale and toy projects. If a project involves fewer than 3 contributors, we also remove it from our data set. Additionally, since RQ3 studies the role of branches on coping with contributions (PRs in GitHub) by others, each selected project should have at least one closed PR (contributors can submit a PR to any branch as they wish). Considering the number of commits, contributors and closed PRs have been listed in each project's website in GitHub, we wrote a web crawler to retrieve these numbers and directly compared them with our threshold values. After filtering, only 2,938 projects met all the above criteria.

When we analyzed the remaining projects, we found that there were 15 projects that do not specifically target developing software but instead focus on documenting tasks. For example, the project `msgpack/msgpack`[6], mainly focuses on managing the specification of MessagePack format. These projects were also eliminated from the 2,938 projects. Ultimately, 2,923 GitHub projects were selected as our experimental subjects. Table I shows more details on our collected project data.

In our experiment, we downloaded each project using *git clone*. Then, for each project we retrieved all branches used (using *git branch -r*). In total, we retrieved 19,389 branches created for the 2,923 projects. We used these collected data to conduct our experiments and answer the five RQs.

B. Design of RQs

We attempt to gain insight into the branch usage in OSS projects via five RQs. In RQ1, we provide an overall view of branches used in 2,923 projects. We further investigate some characteristics of projects that use relatively more branches during development. In RQ2, we investigate branching purpose in GitHub. In RQ3, we study the branches that are requested for merging PRs. We further study whether all commits of other branches make it into master branches in RQ4. Finally in RQ5, we examine the impact of branches usage on projects' productivity. We combine bar plots, non-parametric statistical hypothesis tests, and regression models to answer our RQs. In RQ1, we use bar plot and Wilcoxon Rank Sum test [34] to observe the overall usage of branches across the 2,923 projects. In RQ2, we manually check each branch's content and then provide some descriptive statistics on the purpose of branch use. In RQ3, we calculate the number of pushed pull requests (PRs) in each branch, and investigate the role of branches in coping with PRs in GitHub. In RQ4, we count the number of commits that are unique to non-master branches and perform statistics to understand the flowing of commits from non-master branches to master branches. In RQ5, we use a multiple linear regression model [25] to measure the impact of branch usage on the overall productivity of projects. In particular, we adopt Cohen's f^2 to capture the effect size of branching on project's productivity. The detailed description about the regression model and Choen's f^2 will be presented in Section III-C. During our model building for RQ5, we also use eight confounding factors that may affect project's productivity. These confounding factors are listed as follows:

Forks: The number of forks of a project. Outsiders (developers who cannot directly commit to a project) must fork a project if they want to contribute to it. More forks may indicate that more developers are involved in a project's development. This may affect a project's final productivity.

Watchers: The number of watchers of a project. In general, the more watchers a project has, the more popular it is among developers; a project's popularity may affect its overall productivity.

Project age: Project's age since its creation. A project's age is calculated as the last update timestamp minus the creation timestamp. A longer-lived project may have more commits than a relatively short-lived project.

Project size: Project's size measured as total physical space a project needs to store its data (MB). Generally speaking, different projects with different sizes may behavior differently, thus leading to different project productivity.

Number of PRs: The number of PRs requested to merge into the project. PR is a form of contributions. More PRs indicate that more contributions are requested to merge into the project. This may result in an increase of project productivity.

Number of PR comments: The number of comments made on PRs. More comments on PRs may indicate an active response to contributions by developers. This may result in a quicker resolution to PRs, which may further affect a project's productivity.

Number of issue comments: The number of comments made on issues. More comments on issues may indicate active interactions among developers to fix bugs. This may facilitate a project's development and finally affect a project's productivity [7].

Number of developers: The number of developers contributing to a project. More developers contributing to a project may indicate a higher project's productivity.

The details about above eight confounding factors can be found in Table II.

[6]https://github.com/msgpack/msgpack

TABLE II
SUMMARY OF EIGHT CONFOUNDING FACTORS ON 2,923 GITHUB PROJECTS.

Confound	Min	Median	Max	Mean	St. Dev.
Forks	0	51	9,609	161.7	426.5
Watchers	1	18	3,081	47.0	115.9
Age (years)	5.0	6.8	8.9	6.9	0.7
Size (MB)	0.1	2.0	928.4	12.4	42.9
Total PRs	1	32	4,418	91.1	192.5
PR comments	0	43	13,366	243.8	767.5
Issue comments	0	95	13,248	439.4	1,030.2
Developers	3	15	690	29.8	45.2

C. Statistical methods

1) Multiple Linear Regression: Regression models are often used to measure the effects of several explanatory variables on a response variable. In our study, we use a multiple linear regression model [25] to measure the impact of branches used on a project's overall productivity, i.e., the total number of commits within a project. While building the regression models, we also perform the regression diagnostics with recommended criteria [14]. We log transform (i.e., `x->log(x)`) the response variable (i.e., the number of commits) to assure acceptable normality. We also do the log transformation on several explanatory variables as this can make the variance more stable and can always improve the model fit [11]. Variance inflation is also controlled within the recommended range [25]. All outliers are removed to avoid possible disturbance to the experimental results. In this paper, we use `outlierTest` function in the "car" package of R to detect outliers[7]. All regression models are conducted within the R statistic environment [46].

2) Cohen's Effect Size: Based on the results of multiple regression models, we further use Cohen's effect size [25] to gauge the effect of branches on the project's productivity. Specifically, we use Cohen's f^2 to measure the effect size for a linear regression model. *Cohen's* f^2 is calculated as

$$\frac{R^2_{AB} - R^2_B}{1 - R^2_{AB}}$$

Here R^2_B is the variance accounted by variable set B. The R^2_{AB} is the variance accounted by variable set B and A together. In this paper, B is the set of eight confounding factors as mentioned in Table II and A is the number of branches used. A threshold value of 0.02 for f^2 is suggested as a minimum value to determine that the effect is `small` [25]. 0.15 and 0.35 is suggested as the minimum values for `median` and `large` effects.

IV. EXPERIMENTAL RESULTS

In this section, we present empirical results for the five RQs.

A. RQ1: How many branches does a project have and what kind of projects are more likely to have more branches?

Results. In RQ1, we study the general characteristics of branches used in GitHub. Specifically, we investigate how many branches are used by developers during development.

[7]http://CRAN.R-project.org/package=car.

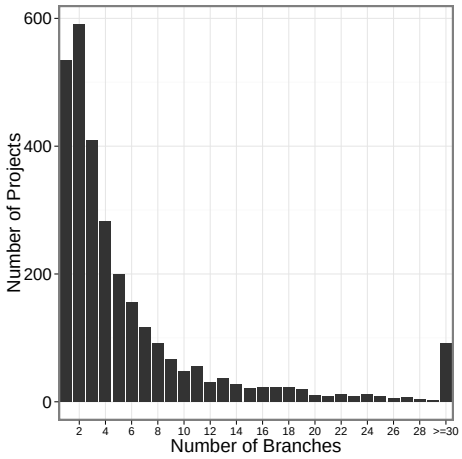


Fig. 1. General statistics on the number of branches used in 2,923 projects.

Additionally, we try to identify what kind of projects tend to use more branches. We conduct our experiments on 2,923 GitHub projects. For each project, we first retrieve all branches created within the project. Then we count how many branches each project uses.

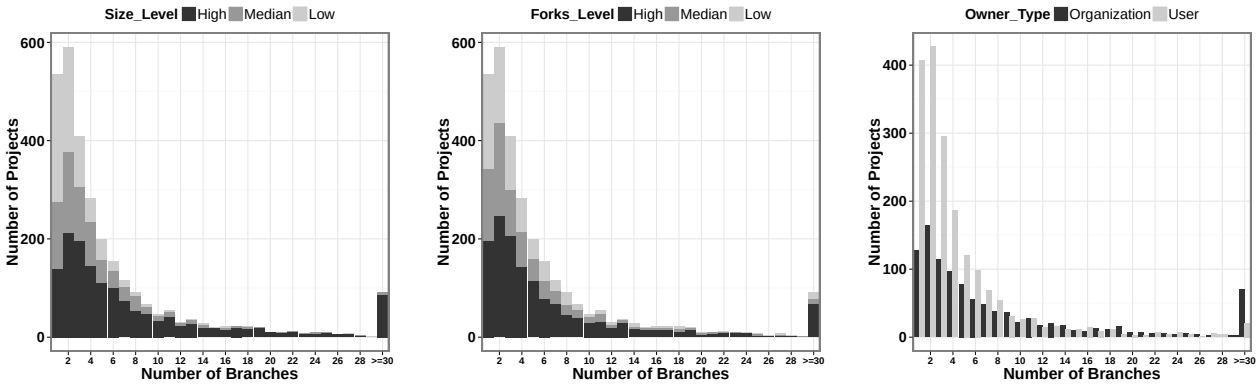
Figure 1 shows the general statistics on the number of branches used in 2,923 projects. The horizontal axis is the number of branches created within a project. The vertical axis represents the number of projects containing a certain number of branches during development. Most projects (87.9%) create less than five branches. Only 91 projects create more than 30 branches, while 534 projects have only one branch. In other words, 18.3% (534/2,923) projects do not create other branches but directly develop on the master branch.

Finding 1. Most projects (87.9%) create fewer than five branches during their development. A small number of projects (3.1%) create at least 30 branches; 18.3% projects only develop on their master branches.

In Figure 1, we also observe that different projects create different numbers of branches during their development. Some use only a few branches while some others create many branches. To understand what kind of projects tend to use more or fewer branches,we divide 2,923 projects into different groups by specific project attributes (such as project's scale and popularity) to examine the branch usage in those projects. We consider three project features, namely *project size*, *forks number*, and *owner type*.

Project size measures a project's scale, which represents how much physical space a project needs to store its data (KB). The Forks count describes how many times a project is forked by others. In GitHub, if a developer wants to contribute to a project, he/she generally needs to fork it to his/her own account. Thus, forks number can be used as a proxy metric of a project's popularity among developers. Owner type can tell us whether a project is owned by a personal user or an organization, which is obtained by searching relevant item (i.e., "type") in the details of individual repository by using GitHub API (i.e., "GET /repos/:owner/:repo").

We divide projects into groups based on each of the above



(a) branches used in projects grouped by size (b) branches used in projects grouped by forks number (c) branches used in projects grouped by owner type

Fig. 2. Branches used in projects grouped by owner type, forks number and size.

three project attributes. Unlike owner type, which has only two enumeration values ("personal user" and "organization"), project size and forks number are numeric values. For each of the numeric project attributes, we divide the numeric values into three groups: small, median, and large. For project size and forks number, we first compute the 1st quartile, the median, and the 3rd quartile of all values. Projects with the measure smaller than the 1st quartile are assigned to the small group. If the value is larger than the 3rd quartile, the project is assigned to the large group. The remaining projects are assigned to the median group. Figure 2(a)-2(c) presents branches used in projects grouped by size, forks number and owner type respectively. The horizontal axis is the number of branches used in a project. The vertical axis is the number of projects which have a certain number of branches. Different colours represent different groups.

In Figure 2(a), when the number of branches is no more than three, the numbers of projects with small size, median size, and large size are similar. When the number of branches increases above 3, large scale projects tend to represent a much larger proportion compared to small and median projects. This means large-scale projects are more likely to use more branches during development. A similar phenomenon can also be observed in Figure 2(b). Thus, a more popular project tends to use more branches during development. As shown in Figure 2(c), when there are less than eight branches, personal projects occupy a larger proportion than organization-owned projects. As the branch count increases, the difference between two types of projects becomes small. For projects with more than 30 branches, organization-based projects are the majority. This suggests that projects created by personal users are more likely to use fewer branches; while projects created by organizations are more likely to use more branches during their development.

Next, we apply Wilcoxon Rank Sum test with Bonferroni correction [2] and Cliff Delta Effect Size [10] to check the statistical significance of the above-observed differences between different groups, i.e., user vs. organization, high vs. median, median vs. low, and high vs. low. We apply Bonferroni correction to adjust p-values from multiple comparisons. We find that the differences in branch usage between different groups are all statistical significant (`p-value<0.05`): for project size or owner type, all `p-values`$<2.2e^{-16}$; for forks number, the p-values for high vs. median, median vs. low and high vs. low are 0.024, $2.68e^{-13}$ and $< 2.2e^{-16}$ respectively. However, all differences except two are small evaluated by the Cliff Delta effect size (`|d|<0.147 "negligible", |d|<0.33 "small", |d|<0.474 "medium", otherwise "large"`): for project size, the difference between high and low groups is large, with effect size value being 0.543 (`>0.474`); for forks number, the difference between median and low groups is negligible, with effect size value being 0.067 (`<0.147`); all other differences are small with their effect size values being larger than 0.147 but smaller than 0.33.

Finding 2. Large scale projects tend to use significantly more branches than small scale projects; popular projects or projects owned by organizations tend to use slightly more branches during software development.

Implications. In Finding 1, we note that most (87.9%) projects use no more than five branches during their development. However, it also can be observed that either some projects (18.3%) only work on the master branch or some projects (3.1%) use many branches (`>30` branches). For these projects, in-depth analysis is desired. For example, we are not clear on why some projects create a lot of branches, what problems they face when they try to manage many branches. Moreover, why do some projects, including organizational projects which may involve many developers, only develop on master branch?

In Finding 2, we find that more popular, larger or organizational projects tend to use more branches than others, which propose more research questions to investigate, e.g., whether these projects might face more challenges when managing more branches, and what problems they exactly face and how do they try to address them. The answers of these research questions can 1) provide some guidance for other practitioners who face similar problems, and 2) inspire more researchers or

practitioners make more efforts to help solve real problems in branching practice in OSS communities.

B. RQ2: What are branches used for?

In RQ2, we try to gain insights into the problem of why branches are created in GitHub. Particularly, considering master branches are used for projects' main line development by default, in this RQ, we only explore the branching purpose of non-master branches. Specifically, we randomly sample 300 branches from 16,466 non-master branches. This sample size is comparable with existing work [18] that 350 out of `>20,000` unmerged PRs are randomly selected to study why PRs are not merged.

We adopt a three-step process to conduct the branching purpose categorization: First, the first two authors selected the first 100 branches and discussed their categories. Specifically, for each non-master branch, they manually read its commit logs, changed code and relevant conversations of commits/issues/PRs on GitHub, in order to understand what activities (such as bug fixing, testing) developers perform on it. Then they took the major activity as the branching purpose for each branch. The results contain 6 categories shown in Table III (with Dependency Configuration and Others being excluded).

Then they tried to classify the remaining 200 branches based on the 6 categories independently. They left the branches which cannot be classified for later discussion. Fleiss Kappa [15] was used to measure the overall agreement between the two labelers. The Kappa value was 0.78, which indicates a substantial agreement between them.

At last, for those branches that they cannot classify into the initial categories or disagree with each other on certain branches, they invited another senior developer with 12 years development experience in industry to help them make the final decision. As a result, a new category, i.e., Dependency Configuration was added to the original 6 categories. There are 7 branches being created for other purposes (such as doing nothing, experimenting with a framework) or unknown purpose that we cannot identify after reading relevant data items. We placed them into the Others category. Table III shows the final categories of branching purposes of the sampled 300 non-master branches.

As shown in Table III, we summarized 8 categories of reasons for which developers create branches, from bug fixing to version iteration. From the table, we can find that approximately 83% branches are created for feature implementation (41.7%), version iteration (24.7%) and bug fixing (16.7%). The remaining branches (approx. 15%) target at testing, documentation, etc. Our findings are consistent with [39]'s study, which also found that feature implementation and version iteration are also the most two popular branching types among developers.

Specially, 58.4% branches target at two major activities during software development, i.e., bug fixing and feature implementation. This to some extent, implies that developers may have recognized branches' potential in helping them better perform independent single task (i.e., bug or feature) in practice; After diving into those branches (24.7%) that aim at version iteration, we find that 1) 25% branches are created for version release; 2) 12% branches are created for a single version development; 3) others (63%) mainly focus on version bumping, i.e., upgrading product to a higher version. This to some extent, indicates that branches may play an important role in helping developers manage their products' versions.

TABLE III
CATEGORIES OF BRANCHING PURPOSE.

Purpose	Description	Num.	%
Bug Fixing	A branch aims to fix bugs.	50	16.7
Feature Implementation	A branch aims to implement features.	125	41.7
Testing	A branch aims to test code, deploy test platform or maintain test cases, etc.	16	5.3
Code Structure Optimization	A branch aims to perform code refactoring or style formatting.	12	4
Documentation	A branch aims to maintain a project's documents, such as its website, license declaration, etc.	10	3.3
Dependency Configuration	A branch aims to declare a project's plugins or dependent libraries' versions.	6	2
Version Iteration	A branch aims to certain version development, prepare for release, and version upgrade. In such a branch, bug fixing and feature implementation are two major activities.	74	24.7
Others	A branch which cannot be put into the above categories goes to this category. E.g., a branch is created for doing nothing.	7	2.3

Despite the importance of testing during development, it seems that developers are less likely to build specific branches to conduct pure testing tasks, with only 5.3% branches focusing on testing. Similarly, other kinds of activities, such as documentation, refactoring, are also not reflected too much in a single particular branch.

Finding 3. Developers mainly use branches to implement features, fix bugs, and conduct version iteration. Very few developers would create particular branches for testing, documentation, etc.

Implications. We find that branches are mainly still focusing on bug fixing and feature implementation (both of these two activities play an important role in version iteration). In practice, developers can also submit pull requests to perform bug fixing and feature implementation activities. Thus, it would be interesting to investigate the difference between the pull requests and branching in bug fixing and feature implementation.

Besides, during our investigation into branching purpose, we find that it is prevalent that branches are not fully described in an explicit way. For example, we find that only 69 out of 300 sampled branches have easy-to-understand names. One may have to spend lots of time on reading changed code and commit logs, to infer the branching purpose. This may place a great burden on project moderators who need to manage

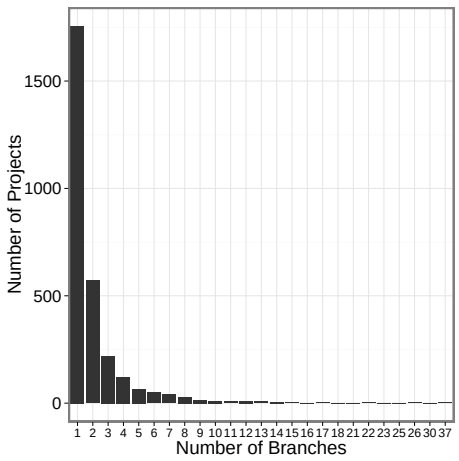


Fig. 3. Branches that are requested to merge PRs.

and maintain lots of branches. Thus, it would be valuable if techniques can be developed to automatically provide a descriptive summary for a branch, by collecting and analyzing different kinds of branching information, e.g., changed code, commit logs, and branch names.

C. RQ3: How many branches are requested to merge PRs?

In GitHub, developers use PRs to contribute to others' projects. A PR contains detailed code change or document modification to the target project by developers. Developers specify the destination branch (i.e., the branch that developers want to have their PRs merged into) for the PRs. Since a project may have several branches during their development, which inspires us to further investigate the prevalence of branches within a project that are requested by developers to merge their PRs.

For each project, we download all PRs that were submitted by developers. We then retrieve the destination branch of each PR with the help of GitHub APIs for PRs[8]. Finally, we count the number of PRs submitted to each destination branch within the project.

After recording each branch's state of receiving PRs, we find that only 6,159 out of 19,389 branches have ever been requested to merge PRs. For those 6,159 branches, we further investigate how many branches have been requested to merge PRs in a project. Figure 3 shows the detailed results. In Figure 3, the horizontal axis is the number of branches that are requested to merge PRs. The vertical axis is the number of projects.

From Figure 3, we can find that more than 90% projects (2,723/2,923=93.2%) have less than 5 branches that are requested to merge PRs by contributors, among which more than 60% projects (1,756/2,723=64.5%) have only one branch having been requested by merging PRs. There are extremely few projects (1.9%) that have more than 10 branches requested by merging PRs. Next, we specifically study master branches and find that almost all projects' master branches (2,909/2,923=99.5%) have been requested to merge PRs by contributors.

After analyzing the overall usage of PRs in each project, we further investigate those projects which have more than two branches with PR merging requests. We find that 1,135 projects have more than two branches having been asked for merging PRs. For those projects, we investigate whether all branches weight the same on being requested by merging PRs. The data shows that 1,121 out of 1,135 projects always have one branch having more than 50% PRs merging requests.

Finding 4. 90% projects have less than 5 branches that are requested to merge PRs in GitHub. There always exists a branch that is mainly targeted by contributors to merge PRs for each project. About 60% projects have only one branch requested to merge PRs. Almost all master branches (99.5%) have been requested to merge PRs.

Implications. In RQ3, we mainly study the current situation of branches that are requested to merge PRs by contributors. One interesting observation is that almost all projects' (99.5%) master branches are pushed PRs by contributors. However, as mentioned previously in Section II, some projects have their own requirement in to which branch a PR is supposed to submit (e.g., contributors should not submit PRs to the targeted project's master branch). This contrary to some extent indicates that not all contributors are aware of such requirements. Those PRs with incorrectly targeted branches may be rejected and be asked to re-merge to the right branches, which will affect contributors' passion on further contribution, and increase project maintainers' workload in coping with PRs. Thus, an effective way to identify the right targeted branch for contributors is very desirable. Researchers or practitioners may need to build an automated approach that can help contributors more easily find the right branch to merge into, such as developing recommending tools or providing more explicit and indicative information of targeted branches.

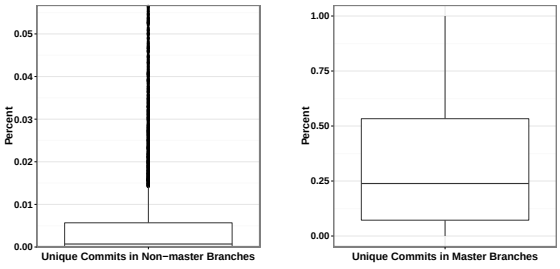
D. RQ4: Do all commits go into the master branch?

In this RQ, we investigate the prevalence of commits flowing into the master branch from other branches. In other words, we try to gain some insights into the validity of work on other branches. In total, we examined 19,389 branches on 2,923 projects (2,923 master branches and 16,466 other branches). For each project, we first find its master branch[9] (M) and other branches (BranchSetA). Then for each branch (B) within BranchSetA, we use the commands "git log --pretty=oneline B --not M --" and "git log --pretty=oneline M --not B --" to retrieve the number of commits which are only in the branch B or in the master branch M. Since the projects we collected have various number of commits in them, we use percentage of total commits that are only in master or only in a non-master branch, in order to better observe the whole situation of commit flow within GitHub.

[8]https://developer.github.com/v3/pulls/

[9]The master branches are identified by retrieving the default base branch of projects hosted in GitHub.

Figure 4 shows the commit flow of master and non-master branches. In Figure 4(a), we find that on the whole situation, the percentage of commits which are left on the non-master branches is very low. It seems that at least 75% branches' all commits are almost contained in master branches (with at most 0.56% commits are not included in the master branch). We further find that 50% branches have only one commit that is not in master branches; and about 40% (6,460/16,466) branches' all commits go into their master branches. In Figure 4(b), we observe that the minimum value of the boxplot is also 0, which means there are branches (actually 373 branches from 278 projects) containing all commits from their master branches. From the 50% quantile value, we can find that, nearly 25% commits of half master branches are not contained in their non-master branches, which is much more than the number (0.7%) of commits unique to a non-master branch. Comparing quantile values of Figure 4(a) with those of Figure 4(b), we can find that master branches always have many more commits that are not in non-master branches.



(a) non-master branches (b) master branches

Fig. 4. Commit flow of master and non-master branches

Finding 5. Almost all commits of most branches (>75%) finally flow into their master branches. In comparison, more than 97% branches (16,093/16.466=97.7%) do not include all commits from their master branches.

Implications. From the results of RQ4, we find that some commits only belong to non-master branches or master branches. For those unique commits, it would be helpful if some techniques can be proposed to summarize these unique commits and provide supportive information to further indicate what is going on in each branch. In this way, developers working on either non-master branches or master branches can feel more free to merge work from other branches without unnecessary conflicts or duplicate work.

E. RQ5: Does branch usage affect the overall productivity of a project?

Branching is a method that can help developers focus on their own development task without hindering, or being hindered by others. In this way, we assume that branching may facilitate development. To our best knowledge, the impact of branch usage on development in large OSS communities has not been studied. In this RQ, we quantitatively measure the impact of branch usage on the overall productivity of projects.

Our experiment includes two parts. The first part studies the impact of branches used on project's productivity. The second part investigates whether the results are different between personal projects and organization-based projects. We use a multiple linear regression model to measure the impact of branch usage on project's productivity. Since code contributions are always encompassed in the form of commits, the number of commits is used as a proxy measure of developers' productivity [13]. Following existing studies [51], [17], [50], we also use the commit count to measure projects' productivity. Specifically, we use the number of commits within master branch as a proxy of a project's productivity and use the number of branches as a proxy of branch usage. We aim to discover whether the more branches are used, the more likely a project will have more commits on the master branch. Since there are other factors (such as project popularity and developers' interaction) that can also affect project's productivity, we also use other eight factors listed in Table II as confounding factors.

As mentioned in Section III, we use *Cohen's* f^2 to measure the effect size of branch usage on a project's productivity. To measure effect size, we first build a model with all explanatory variables, i.e., the factor `brCnt` (the number of branches) and other eight confounding factors. Then we build another model with only eight confounding factors. We use each models' Adjusted R-squared to calculate the `Cohen's` f^2 as the branches' effect size on each project's productivity.

Since some explanatory variables have large numeric scales, we performed log transformation on them and outliers are removed. The residual distribution is found to have acceptable normality and the variance inflation is also within recommendable ranges [25].

Table IV and Table V show the results of two models. From the two tables, we can find that the project's final productivity (i.e., the number of commits) increases with the number of branches, developers, PRs, issue comments and project age; the final productivity decreases with the number of forks and watchers of projects.

As can be seen in Table IV and Table V, the R^2 for models that includes and excludes `brCnt` is separately 0.625 and 0.616. Thus the `Cohen's` f^2 is (0.625-0.616)/(1-0.625)=0.024. Since 0.024 is bigger than the minimum value 0.02 for a `small` effect but smaller than the minimum value 0.15 for the `median` effect, we can conclude that the branch usage does have a positive effect on project's productivity, although the effect is small.

Finding 6. The number of branches used within a project has a positive impact on a project's productivity, despite of a small effect measured by Cohen's effect size. This means, to some extent, the more branches a project has, the more final commits a project has.

After examining the effect of branch usage on project productivity on all 2,923 projects, we further investigate whether the effect is different between projects owned by personal users and by organizations. We divide 2,923 projects into two groups: personal projects and organizational projects. There are 1,891 personal projects and 1,032 organizational projects,

TABLE IV
MULTIPLE REGRESSION MODEL FOR BRANCH USAGE (BRCNT) AND EIGHT CONFOUNDING FACTORS ON 2,923 PROJECTS.

Variable	T Value	Significance
(Intercept)	17.4	p<.001 ***
brCnt	8.8	p<.001 ***
log(forks_count+0.5)	-9.2	p<.001 ***
log(watchers_count+0.5)	-3.5	p<.001 ***
age	6.6	p<.001 ***
log(project size)	43.7	p<.001 ***
developer	9.4	p<.001 ***
PRs	4.8	p<.001 ***
log(PR comments)	4.7	p<.001 ***
log(issue comments)	8.3	p<.001 ***
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1		
F-statistic = 542.9, p-value: <0.001. R^2 = 0.625		
The response variable is log(total commits).		

containing 9,290 and 10,099 branches respectively.

For personal projects and organizational projects, we separately build two models in Tables IV and V to find the R^2 to further calculate the Cohens' f^2. The F Statistics of four models are all significant (p-value<0.001). The detailed results can be found in Table VI

From Table VI, we can find that the Cohen's f^2 values of both models are larger than 0.02 (the minimum value for a small effect) but are smaller than 0.15 (the minimum value for a median effect). This means the number of branches indeed has a small effect on the productivity of both personal and organizational projects. However, despite organizational projects tend to use more branches than personal projects (which can be observed in RQ1), the number of branches does not have a stronger effect on the productivity of organizational projects than that of personal projects (with both Cohen's f^2 values being almost equal, i.e., 0.0277 vs. 0.0265).

Finding 7. The number of branches has a small effect on the final productivity of projects created by personal users and organizations, and there is no statistically significantly difference between personal projects and organizational projects.

Implications. In this paper, we focus on how branch usage affects a project's commits productivity. We find the branch use has a small effect on a project's overall productivity measured by Cohen's f^2. It would be interesting to investigate how branch usage affects other OSS development activities such as developer's productivity (e.g., bug fixing or feature implementation) or software maintainability (e.g., conflicts inducing by merging branches). By doing so, we are likely to gain a more complete understanding of the functions of branches.

V. THREATS TO VALIDITY

Generalization of Findings. In this paper, we try to gain some insights into the branch usage in OSS communities by conducting experiments on 2,923 GitHub projects. Since there are also other OSS communities (such as SourceForge and BitBucket), we cannot guarantee that our conclusions on GitHub can be generalized to other OSS communities. However, with the great popularity of GitHub for OSS development and the large scale of our experiments, we think the findings on GitHub can reveal some insights into the practice of branching in OSS communities. In future, we plan to perform our experiments on other OSS platforms to better generalize our findings.

TABLE V
MULTIPLE REGRESSION MODEL FOR CONFOUNDING FACTORS.

Variable	T Value	Significance
(Intercept)	16.7	p<.001 ***
log(forks_count+0.5)	-9.8	p<.001 ***
log(watchers_count+0.5)	-3.5	p<.001 ***
age	6.5	p<.001 ***
log(project size)	45.9	p<.001 ***
developer	9.3	p<.001 ***
PRs	5.5	p<.001 ***
log(PR comments)	5.7	p<.001 ***
log(issue comments)	8.2	p<.001 ***
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1		
F-statistic = 585.9, p-value: <0.001. R^2 = 0.616		
The response variable is log(total commits).		

TABLE VI
ADJUSTED R-SQUARED (R^2) OF REGRESSION MODELS BUILT ON PROJECTS OWNED BY PERSONAL USERS AND ORGANIZATIONS.

owner	with brCnt	with no brCnt	F value	Conhen's f^2
users	0.5774	0.5657	p<0.001	0.0277
Org.	0.6268	0.6169	p<0.001	0.0265

Private Projects. In this paper, we only consider projects that are available in GitHub. This may make us miss some private projects that are inaccessible to the public in GitHub. These missing projects may bring some bias to our concluded finding across the GitHub platform. However, considering the large scale of our experiments, we believe our findings can still be generalized to other projects.

Implicit Merged Commits. When examining commit flows in RQ4, we take commits only occurring in non-master branches as commits that are not merged into the master branch. However, since developers may use `rebase` or `cherry-pick` provided by Git to manipulate commits, some seemingly unmerged commits may have been merged into the master branch [26]. These implicit merged commits may indicate a higher flowing ratio of commits into the master branch than we have observed in RQ4. Although we are unable to track these commits, we think our findings of RQ4 can shed some light into the trend of how commits flow between branches.

Measurement of Project Productivity. We use the number of commits as a project's overall productivity, as has prior work [51], [17]. In particular, we only take the commits of master branch into account without considering other branches. Since some useful work may stay in other branches without being merged into the master branch, this might bring some bias to our arrived conclusions. However, since it is hard to identify whether all unmerged commits from other branches are useful, and it is very likely that most useful commits would go into the master branch, we think only using the commits of master branch is also acceptable.

VI. RELATED WORK

Studies on Branching. There are some studies similar to ours. Bird et al. showed that developers working on Windows

use branches to divide work into tasks and teams [6]. They also found some problems related to merging branches in Windows [5]. Phillips et al. conducted a survey among industrial developers and gained insights into the best branching strategy in practice [39]. Shihab et al. investigated the effect of different branching strategies on software quality [43]. Hua found that branches are often used during Linux kernel development, and cross-branch porting is frequent and error-prone in practice [21]. These studies mainly focus on industrial branching practice or are limited to a small number of specific OSS projects. To the best of our knowledge, we are the first to conduct a large-scale empirical study of branch usage on a broad collection of OSS communities.

Researchers also have made several efforts to tackle the problems of using branches in practice. Bird et al. proposed to remove low benefit but high cost branches to reduce the integration delays [5]. Tarvo et al. [45] and Michaud [36] proposed to track commits from different branches to improve the maintainability of software. Kerzazi proposed to use social network to build better branch strategies to facilitate the software development [28]. Costa et al. developed TIPMerge to recommend developers to merge commits from different branches [12]. Kovalenko et al. emphasized that it was important to properly cope with branches when calculating file modification histories [29]. We studied a different problem and our findings also projected some insights into the best practices of branching and potential related problems to be solved.

Some researchers have also developed some guidelines for using branches. Berczuk and Appleton guided people when to branch and how to adopt the right branch structure [4]. Wingerd et al. gave some high-level best practices of using branches in software management [55]. Arve studied how to use advanced branching strategies to facilitate the development in Agile projects [3]. Appleton et al. described some branch patterns for parallel software development [1]. Premraj et al. discussed the pros and cons of branches and presented some guidelines on when and how to branch [40]. There also exist some lively discussions or specific documents on branching and merging over Internet. For example, Linus Torvalds provided some branching and merging guidance for Git users in an email thread [31]. Microsoft documented some practical advice on branching strategies in the Team Foundation Server branching guide [22]. These studies mainly provided some best practices of branching based on personal experience. We studied the actual practice of branching in 2,923 real GitHub projects and investigated different aspects of branch usage.

Studies on GitHub. A great number of studies have been conducted on GitHub, with focuses ranging from pull-based mechanism, social coding to testing and being productive on GitHub [57], [37], [38], [51], [26], [53].

Zhu et al. investigated the effectiveness of pull-based mechanism compared to patch-based methods [57]. Gousios et al. studied the challenges of pull-based development from both the contributors' and integrators' perspectives [19], [20]. They further found that 40% PRs do not appear as merged even though they have been merged [27]. Some researchers explored how social and technical factors may affect the evaluation of PRs [47], [18], [56]. Veen et al. proposed to rank PRs to help developers better handle PRs [49]. Different from above studies, we studied a new aspect of PRs, i.e., the destination branches of PRs.

Vasilescu et al. found that the gender and tenure diversity, the adoption of continuous integration, would affect a project's productivity [51], [52]. They also studied how context-switch would affect developers' productivity [52]. Casalnuovo et al. found that both social links and language experience have an influence on both developers' and projects' outcomes [8]. We looked into a new factor, i.e., branch usage, that affects a project's productivity.

Vendome et al. studied the license changing problem [54]. Ray et al. measured the naturalness of buggy source code [41] and explored how programming languages would affect code quality [42]. Silva et al. and Tufang et al. did an exploration into the code smells [44] and test smells [48]. Fowkes et al. tried to mine APIs across GitHub [16]. Ma et al. explored how developers coped with cross-project bugs [33]. Lee et al. found that developers used rockstars to better choose and contribute to projects in GitHub [30]. Jiang et al. found that creators' status affected developers' forking behavior [23]. Marlow et al. studied impression formation in GitHub [35]. Jiang et al. studied how projects get disseminated across GitHub [24]. Unlike them, we considered branch usage in GitHub.

VII. Summary

In this paper, we conduct an exploratory study of branch usage in 2,923 OSS projects hosted on GitHub. We first investigate how branches are used in 2,923 projects. We find that most projects have less than five branches during development and less than 20% projects only develop on their own master branches without creating other branches. Large scale projects tend to use significantly more branches than small scale projects. Then, we further study the branching purpose in GitHub, we find that 83% branches are created by developers to conduct feature implementation, version iteration, and bug fixing.

We also find that almost all master branches of 2,923 projects have been requested to merge contributions by developers. Each project has less than two other branches that have received pull requests on average. When studying the commits of branches, we find that 75% branches' most commits are contained on their master branches.

Finally, we investigate the impact of branch usage on project's productivity. We find that the more branches a project has, the more commits the project has. But the impact is considered small measured by Cohen's effect size; and such impact shows no difference between personal and organizational projects. Our findings provide insight into the situation of the current practice of branching in OSS communities on GitHub. Our experimental data sets are available online: https://github.com/SurfGitHub/branchStudy.

REFERENCES

[1] B. Appleton, S. Berczuk, R. Cabrera, and R. Orenstein. Streamed lines: Branching patterns for parallel software development. In *Proceedings of PloP*, volume 98, 1998.

[2] R. A. Armstrong. When to use the bonferroni correction. *Ophthalmic and Physiological Optics*, 34(5):502–508, 2014.

[3] D. Arve. Branching strategies with distributed version control in agile projects. pages 1–12, 2010.

[4] S. P. Berczuk and B. Appleton. *Software configuration management patterns: effective teamwork, practical integration*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[5] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 45. ACM, 2012.

[6] C. Bird, T. Zimmermann, and A. Teterev. A theory of branches as goals and virtual teams. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 53–56. ACM, 2011.

[7] C. Casalnuovo, P. Devanbu, A. Oliveira, V. Filkov, and B. Ray. Assert use in github projects. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 755–766. IEEE Press, 2015.

[8] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov. Developer onboarding in github: The role of prior social links and language experience. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 817–828. ACM, 2015.

[9] S. Chacon and B. Straub. *Pro git*. Apress, 2014.

[10] N. Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[11] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[12] C. Costa, J. Figueiredo, L. Murta, and A. Sarma. Tipmerge: recommending experts for integrating changes across branches. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 523–534. ACM, 2016.

[13] S. Daniel, R. Agarwal, and K. J. Stewart. The effects of diversity in global, distributed collectives: A study of open source project success. *Information Systems Research*, 24(2):312–333, 2013.

[14] P. Devanbu, T. Zimmermann, and C. Bird. Belief & evidence in empirical software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 108–119. ACM, 2016.

[15] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[16] J. Fowkes and C. Sutton. Parameter-free probabilistic api mining across github. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 254–265. ACM, 2016.

[17] J. Gamalielsson and B. Lundell. Long-term sustainability of open source software communities beyond a fork: A case study of libreoffice. In *IFIP International Conference on Open Source Systems*, pages 29–47. Springer, 2012.

[18] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM, 2014.

[19] G. Gousios, M.-A. Storey, and A. Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, pages 285–296. ACM, 2016.

[20] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 358–368. IEEE Press, 2015.

[21] J. Hua. *A case study of cross-branch porting in Linux Kernel*. PhD thesis, 2014.

[22] B. Javidi, J. Pickell, B. Heys, T. Erwee, and W.-P. Schaub. Microsoft Visual Studio Team Foundation Server Branching Guidance 2010.Microsoft Corporation, 2010 edition, 2009. 2010.

[23] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. Why and how developers fork what from whom in github. *Empirical Software Engineering*, 22(1):547–578, 2017.

[24] J. Jiang, L. Zhang, and L. Li. Understanding project dissemination on a social coding site. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 132–141. IEEE, 2013.

[25] R. Kabacoff. *R in action: data analysis and graphics with R*. Manning Publications Co., 2015.

[26] 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*, pages 92–101. ACM, 2014.

[27] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035–2071, 2016.

[28] N. Kerzazi. Branching strategies based on social networks. In *Proceedings of the 1st International Workshop on Release Engineering*, pages 25–28. IEEE Press, 2013.

[29] V. Kovalenko, F. Palomba, and A. Bacchelli. Mining file histories: should we consider branches? In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 202–213. ACM, 2018.

[30] M. J. Lee, B. Ferwerda, J. Choi, J. Hahn, J. Y. Moon, and J. Kim. Github developers use rockstars to overcome overflow of news. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 133–138. ACM, 2013.

[31] Linus, Torvalds. [git pull] drm-next, March 2009. http://www.mail-archive.com/dri-devel@lists.sourceforge.net/msg39091.html, 2009.

[32] J. Loeliger and M. McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[33] W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu. How do developers fix cross-project correlated bugs?: a case study on the github scientific python ecosystem. In *Proceedings of the 39th International Conference on Software Engineering*, pages 381–392. IEEE Press, 2017.

[34] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

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

[36] H. M. Michaud. *Detection of Named Branch Origin for Git Commits*. PhD thesis, University of Akron, July 2015.

[37] R. Pham. Improving the software testing skills of novices during onboarding through social transparency. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 803–806. ACM, 2014.

[38] R. Pham, L. Singer, and K. Schneider. Building test suites in social coding sites by leveraging drive-by commits. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1209–1212. IEEE Press, 2013.

[39] S. Phillips, J. Sillito, and R. Walker. Branching and merging: an investigation into current version control practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 9–15. ACM, 2011.

[40] R. Premraj, A. Tang, N. Linssen, H. Geraats, and H. van Vliet. To branch or not to branch? In *Proceedings of the 2011 International Conference on Software and Systems Process*, ICSSP '11, pages 81–90, New York, NY, USA, 2011. ACM.

[41] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu. On the naturalness of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*, pages 428–439. ACM, 2016.

[42] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165. ACM, 2014.

[43] E. Shihab, C. Bird, and T. Zimmermann. The effect of branching strategies on software quality. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 301–310. IEEE, 2012.

[44] D. Silva, N. Tsantalis, and M. T. Valente. Why we refactor? confessions of github contributors. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 858–870. ACM, 2016.

[45] A. Tarvo, T. Zimmermann, and J. Czerwonka. An integration resolution algorithm for mining multiple branches in version control systems. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 402–411. IEEE, 2011.

[46] R. C. Team et al. R: A language and environment for statistical computing. 2013.

[47] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366. ACM, 2014.

[48] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. An empirical investigation into the nature of test smells. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 4–15. ACM, 2016.

[49] E. van der Veen, G. Gousios, and A. Zaidman. Automatically prioritizing pull requests. In *12th IEEE/ACM Working Conference on Mining Software Repositories, Florence, Italy, May 16-17, 2015*, pages 357–361, 2015.

[50] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov. The sky is not the limit: multitasking across github projects. In *Proceedings of the 38th International Conference on Software Engineering*, pages 994–1005. ACM, 2016.

[51] 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 *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3789–3798. ACM, 2015.

[52] 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*, pages 805–816. ACM, 2015.

[53] C. Vendome. A large scale study of license usage on github. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 772–774. IEEE, 2015.

[54] C. Vendome, G. Bavota, M. Di Penta, M. Linares-Vásquez, D. German, and D. Poshyvanyk. License usage and changes: a large-scale study on github. *Empirical Software Engineering*, pages 1–41, 2017.

[55] L. Wingerd and C. Seiwald. High-level best practices in software configuration management. In *International Workshop on Software Configuration Management*, pages 57–66. Springer, 1998.

[56] 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*, pages 367–371. IEEE, 2015.

[57] 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*, pages 871–882. ACM, 2016.