

<Analyses Pull-request Characteristics on Pull Request acceptance in Distributed Software Development>

<Lingyi Jia>

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in partial fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

<Date of submission placed here>

**Abstract**

<Abstract goes here…>

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

<**Please note that you are under no obligation to sign this declaration, but doing so would help future students.>**

Name: Signature:

Acknowledgements

<Acknowledgements go here>

Contents

<Update the table of contents by right-clicking on it and selecting Update Field… and then select page numbers only.>

Chapter 1 Introduction 1

1.1 A section <This is style Heading 2> 1

1.1.1 A subsection <This is style Heading 3> 1

Chapter 2 Background 2

Chapter 3 Related Work 3

Chapter 4 Methodology & Design 4

**Chapter 5 Results & Analysis**

Appendix A <Name of appendix> 1

Appendix B <Another appendix> 2

Bibliography 3

# Introduction

## Motivation

With the rapid development of computer technology, the software development process has become more and more complex. A project often requires many developers to develop together, and version control systems have emerged. In 2001, the emergence of distributed version control systems, especially Git [1], replaced the earlier centralized version control systems. A number of code-hosting platforms were developed based on Git, such as GitHub, Gitlab, and Gitee [2]. These code-hosting platforms provide comprehensive version control features. During the development process, developers fork their projects to their own development branches, then work independently on their own branches. After that, when they finish their development tasks, they request a pull request(PR) to merge their code into the master branch. And a certain number of reviewers from the core team will review the submitted code to ensure code quality.

With the code hosting platform combined with social media features, the relationship between developers and code operations becomes visible. GitHub, as a social development platform, incorporates social media features, provides the infrastructure to facilitate collaborative development, makes work more visual, and fully supports collaboration by managing PRs. PRs consist of one or more commits submitted by contributors and request to be integrated into the repository. A popular repository may receive many PRs, and the project maintainer needs to perform a careful code review to determine which PRs can be merged into the codebase. In practice, not all submitted PRs can be integrated into the repository by the project maintainer. For example, as of 1st December 2021, the freeCodeCamp/freeCodeCamp repository had closed 28,120 PRs, of which only 6,809 were explicitly marked for merging [3]. Therefore, it is important for the open-source community to understand what influences PRs to be accepted or rejected.

There has been a large amount of research on technical and non-technical factors affecting the integration of PRs, where complexity, test inclusion, hotness of a PR, and continuous integration have been commonly mentioned in existing studies and have been shown to be related to the integration of PRs [4,5,6,7]. However, existing studies have used one or some of the factors to represent one aspect and have not combined multiple factors to analyze one aspect comprehensively. For example, where the hotness of PR can be inferred from several characters: number of commits, issue comment count, commit comment count, and number of participants.[4,5,6,7,8]. In this paper, we delve into the relationship between different PRs’ characters and acceptance, as well as the relationship between characters of PRs through the changing size. The exploratory study of the impact of PRs characters on PRs integration will be of further relevance to contributors - providing an opportunity for contributors to make improvements prior to review.

## Purpose

PRs play an important role in distributed software development, especially in open-source projects, where they are the dominant form of contribution. The acceptance and rejection of PRs largely affects the prosperity of the open-source community, yet not all PRs are accepted. As a result, it is crucial to understand the factors that influence the acceptance and rejection of PRs. In this paper, to analyze the relationship between the PRs’ characters and acceptance, we have extracted 23 metrics from nine characters of PRs, including the size of a change, complexity, nature, test inclusion, reference, conflict, hotness, emotions, and continuous integration.

In this paper, we conducted an exploratory study on data from **new\_pullreq**, containing 11,230 open-source software(OSS) projects, and 3,347,937 PRs [9]. We quantitatively analyzed the PR characters based on 23 factors

. We tried to show the relationship between the PR characters and the acceptance of PRs by answering the following 3 research questions (RQs).

1. Do the characteristics of PRs vary across prjects?
2. How do the characteristics of PRs relate to the acceptance of PRs
3. What is the relationship between the time cost of PRs and the time cost of closing PRs

# Background

## Distributed Version Control System

A version control system is one of the main strategies of configuration management that allows developers to control changes to software artifacts through development and maintenance [10]. Its most important role is to keep track of changes to source code, configuration files, and other files in software projects. Specifically, It records information about various change operations in the code repository by developers, when the changes occurred, and who submitted the changes. At the same time, as developers continue to commit their changes, the version control system also increases the version number of the file. Another important function of a version control system is to facilitate collaborative development. A mature software project often requires many developers to work together to complete. And the version control system helps software developers to achieve the function of multi-version integration and synchronization, which improves the efficiency of collaborative development in the software development process, and frees software developers from geographical restrictions, and enables remote work on software projects.

There are two types of version control systems commonly used in today's software project development: centralized version control systems and distributed version control systems. Compared to centralized version control systems, distributed version control systems provide enhanced support for collaborative software development. Distributed version control systems allow developers to control multiple remote repositories, thus allowing different groups to work jointly on the same project at the same time. An example of this type of system is Git, which documents the entire history of software development. In this way, hosted platforms are described as rich sources of information that support the configuration management process, such as GitHub, a mainstream development platform built on the version control system Git that provides code hosting and collaboration.

## Pull-Based Model

Version control systems have evolved from centralized to distributed version control through continuous development. The purpose of distributed development is to enable potential contributors to commit a set of changes to a software project managed by a core team. distributed version control systems provide a development model that is a superset of the centralized version control environment [11], where the pull-based development model is one of the increasingly popular models in distributed software development.

In this development model, project contributors do not have direct access to the main repository, and a unique feature of such sites is that they allow any user to clone any public repository. And the user can clone the main repository and modify the repository without belonging to the development team. In addition, the core teams selectively merge changes from the cloned project to the source repository via PRs.

## Pull Request in Git

Git is the most popular distributed version control system today, and the biggest difference between it and most other version control systems is the way they handle data. Most other version control systems store information in the form of a list of file changes [12][13]. In Git, the local client does not just clone a snapshot of the latest version of the remote repository, but stores all the files in the repository, including all the historical information, as a complete backup of the repository. As you can see in Figure 1, Git treats the data associated with a change as a series of snapshots of the local file system. When a developer commits or saves the state of a project, Git creates a snapshot of all the files in the repository at that time and saves its index. Because Git is designed this way, every PC with a cloned code repository can be considered a server, and if any Git server goes down, all you need is a local repository that is up to date with the latest version of the collaborative development process to quickly recover. In addition, Git developers have extended its small file system-like design with a number of useful and functional tools.

Diagram

Description automatically generated

Figure 1. Store snapshots of project changes over time

Another feature of Git is that almost all operations are local. In Git, except for some operations involving remote code repositories, most development activities performed by software developers require access to local code repositories and resources.

There are five states for files managed by Git:

(1) Untracked: This means that the file exists in your working directory, but is not managed by Git. All changes to the file will not be recorded by Git, so you can add the file to your Git managed files list with the ‘git add’ command and change the file status to unmodified.

(2) Unmodified: This means that the file is managed by Git but has not been changed. If you change the contents of the file, the status of the file will automatically change to Modified, and if you move the file out of Git's managed files list with the ‘git rm’ command, the status will revert to Unmodified.

(3) Modified: This means that the file that Git is tracking has been modified. You can overwrite the current changes with the ‘git checkout’ command, which will change the status of the file to unmodified, or you can save the changes to the staging area with ‘git add’, which will change the status of the file to staged.

(4) Staged: indicates that the changes have been saved to the staging area of the local repository. The ‘git commit’ command commits all changes in the staging area to the local code repository, and the file status is converted to committed.

(5) Committed: indicates that the changes have been saved to the local code repository and can be synchronized to the remote code repository via the git push command.

Committed means that the software developer has made changes to the file and the data has been safely saved to the database; Modified means that changes to the file have been made but not yet saved to the database; Staged means that the current version of a modified file has been marked for inclusion in the snapshot of the next commit.

These five states introduce the concept of four working areas of a Git project: the Working Directory, the Repository, the Remote Directory, and the Stage/Index. The relationship between the file state and the working area in Git is shown in Figure 2.

Diagram

Description automatically generated

Figure 2: The Relationship Between File Status and Workspace in Git

The main workflow for Git-based software development is as follows.

1. the project manager creates a new remote code repository and opens the corresponding permissions for the software developers involved.

2. the software developer clones the remote code repository to the local area, and then develops it in the local workspace.

3. the software developer completes this development task and selectively adds the files to the staging area.

4. the software developer submits an update to persist the files in the staging area to the local code repository in the form of a snapshot.

5. the software developer synchronizes the commit requests from the local code repository to the remote code repository.

6. the project manager reviews the commit content and decides whether to accept it or not.

## Continuous Integration

As one of the typical practices of agile software development, the concept of Continuous Integration (CI) was first originated from Extreme Programming (XP) to reduce the tedious deployment process by automatically upgrading periodic or triggered updates and iterations with pre-defined policies. Continuous integration theory is one of the common approaches used in current software development. By using continuous integration theory, different pieces of code can be integrated into the backbone of the system, and the product can be iteratively updated and the quality of software development can be improved. In the process of continuous integration software development, it is important to test the different pieces of code before they are integrated. Only after passing the tests can the code be integrated, and if problems are found in the code during the tests, the problems need to be solved before the code can be integrated. Continuous integration does not guarantee that there will be no problems in software development, but it does make it easy to find problems in software development. Continuous integration is now supported on GitHub and is enabled in many large projects. CI is an important aspect that affects the acceptance of PRs.

## Summary

# Related Work

The goal of distributed software development is to allow developers to work on the same software product at different times in different locations [14]. By introducing version control systems, distributed software development is widely used. The main feature of the distributed software development approach is the process of integrating a set of requested changes into the project codebase. This process of integrating code changes has undergone several stages of development as collaboration tools have matured and adapted to changing development needs. For example, Mockus [15] analyzed two early OSS communities, Mozilla and Apache, and identified a common model for evaluating code contributions, that is, a commit-and-review process. As an alternative, the Apache community also reviews submission requests via mailing list revision submissions. And Weissgerber[16] studied the email archives of two OSS projects and found that smaller contributions were reviewed faster and had a higher chance of being accepted. Rigby and Storey [17] studied the mailing list of five OSS projects for the review process. They found and resolved technical issues were not the only motivation behind the review process; for example, developers also attempted to address scope and process issues during the review. Baysald [18] studied two industry-led software projects, Blink and WebKit, and they found that technical, organizational, and personal factors all affected the review time and probability of PR acceptance.

With collaborative software development being widely used, distributed version control has become the mainstream version control system. And PR-based development models have been applied in the control system platform. So far, there has been some research work related to PR, such as understanding PR-based development patterns, factors that influence PR evaluation and etc.

In terms of understanding PR-based development patterns, Gousios [19] analyzed the popular features of the pull-based model, studied the lifecycle of PR and the processing time of requests. and explored what factors influence reviewers to merge pull requests. Besides, he also investigated the reasons why merge requests are rejected and found that technical reasons are only a small part of the factors.

Researchers have also explored factors that might have influences on PR evaluation, including both technical and non-technical factors. Gousious and his team [19] are the pioneers of pull-based development model research. They quantitatively surveyed OSS projects hosted on GitHub to understand the factors that influence PRs review time and PRs acceptance. They found that merge time was influenced by several factors, including developers' track records and test coverage in the project, while acceptance was primarily influenced by the number of recent changes to the code. Yue Yu [7] conducted a quantitative study that sought to address what factors influence the merging and latency of request evaluations on GitHub. They modeled CI services with data extracted from a sample of GitHub projects and found that the size of the pull request mattered. To be more exact, the shorter the time, the faster the review of the request. And there are also some other potentially strong influencing factors. For example, whether the first reviewer delayed the review of the request the availability of the CI pipeline, both of which they believe may speed up the review process by improving. Similarly, Vasilescu [21] found that CI resulted in more PRs being merged. Rahman and Roy [22] found that factors such as the maturity of the project and the number of developers involved influenced the decision to merge PRs. Padhye [23] showed that bug-fixing PRs were more likely to be merged compared to feature enhancements.

# Design&Methodology

Our research procedure consists of three parts. First, we selected 9 characters to describe PRs, and then selected 23 factors from **new\_pullreq**. Finally, we explore whether the complexity of PRs is related to the integration of PRs by answering three research questions (RQs).

## Question Design

Using **new\_pullreq** as our data source, we explore the relationship between the characteristics of PRs and PRs acceptance through the following research questions.

RQ1: Do the characteristics of PRs vary across projects? The 11,230 projects in the experiment come from different communities and organizations, and their uses vary to some extent. The characteristics of PRs submitted by contributors to these projects also vary from project to project.

RQ2: Are the characters of PRs related to the acceptance decisions of PRs? Developers want their PRs to be accepted, and this is one of the top concerns of developers. At the same time, the reviewer's merge decision is one of the most important concerns in many similar studies. And we measure the relationship between the characters of PRs and the acceptance decision of PRs by constructing a generalized linear mixed-effects model.

RQ3: Are the characters of PRs related to the time cost of closing PRs? Developers care about the acceptance decision of PRs in addition to the processing time of PRs, and developers want their PRs to be accepted faster. We use the time cost of closing PRs to evaluate the progress of processing submitted PRs. The short time cost of closing PRs means that maintainers can respond quickly to developers' contributions. We measure the relationship between the characters of PRs and the time cost of closing PRs by constructing a linear mixed-effects model.

## DataSet

Based on a large amount of data from previous studies, Xunhui[9] create a new dataset**new\_pullreq**, containing 11, 230 projects and 3, 347, 937 PRs with 96 features. This dataset collects features that may affect PR acceptance from multiple perspectives, such as contributor characteristics, project characteristics, pull request characteristics. It is the most comprehensive and largest dataset for now, and it is also the source of data used in our study.

We try to identify different features to describe PRs, which cover every aspect of PRs as much as possible. As a result, we can measure the relationship between PRs characters and PR acceptance more accurately and comprehensively. We figure out further metrics by summarizing the factors influencing the PR acceptance that have been addressed in previous studies.

Kononenko and Baysal [24] found that developers believe that the size of changes affects the review decision. In general, a PR with large changes is complex, but they do not clarify how to determine the size of a change to a PR. Tsay’s team[25] measure the impact of technical factors on contributions. The considered factors include test inclusion, lines of code changed, and the number of files changed. Baysal[18] has research on the code review process, and the number of changed code lines was addressed. Jiang’s team [5] study accepted patches based on Linux kernel; Yu [7] tries to investigate which factors affect the evaluation delay of PRs on GitHub, which includes the number of commits and the number of lines of changed code.

By summarizing the research of others, we identified nine aspects that characterize PRs, and these criteria address almost every aspect of the inherent properties of PRs. The list and description of these characters are shown in the following table.

|  |  |
| --- | --- |
| Pull Request Characters | Description |
| Merged or not | It describes whether a PR has been accepted and merged |
| Size of change | It is the size of PR, which is measured by the number of changed (added or deleted) code lines. |
| Complexity | It is the complexity of PR, which is reflected by the length of PR’s description. |
| Nature of pull request | It is the nature character of PR, which is represented by bug fix. |
| Test inclusion | It describes whether a PR include any test. |
| Reference of a contributor, issue or pull request | It describes how many times a PR’s contributor, issue, itself are referenced. |
| Conflict | It is the conflict of PR, which is inferred by the number of ‘conflict’ in the PR’s comment. |
| Hotness or relevance | It is the hotness or relevance of PR, which is inferred from the number of PR comments. |
| Emotions | It is the emotion(positive, negative, and neutral) related to a PR. |
| Continuous Integration | It describes whether a PR uses a CI tool. And it also includes characters related to CI. |

And the summary of the dataset is described by mean, min, median, and max value. And more details of these metrics are shown in the appendix.

Table

Description automatically generated

## BoxPlot

The boxplot is a method of describing data using five statistics: minimum, first quartile, median, third quartile, and maximum. It can also give a rough idea of whether the data is symmetrical, the degree of dispersion of the distribution, and other information. It is especially suitable for comparing several samples.

The boxplot can visually identify outliers in the data batch. It is known that the 3σ rule or z-score method is based on a normal distribution. And their criteria for determining outliers are based on the calculation of the mean and standard deviation of the data batch. However, the actual data often do not strictly obey the normal distribution, and the mean and standard deviation are extremely resistant. As a result, the outliers can have a large impact on them. However, the boxplot provides us with a criterion for identifying outliers: an outlier is defined as a value less than Q1-1.5IQR or greater than Q3 + 1.5IQR, which is somewhat different from the classical method of identifying outliers. This criterion is derived from empirical judgments and has shown that it performs well in dealing with data that require special attention. boxplot relies on actual data. It does not require prior assumptions that the data obey a particular form of distribution. Also, it does not impose any restrictive requirements on the data. On the other hand, the boxplot judges outliers based on quartiles and interquartile distances. The quartiles are resistant to a certain degree, and up to 25% of the data can become arbitrarily far away without greatly perturbing the quartiles. So outliers cannot exert an influence on this criterion, and the boxplot identifies outliers with more objective results.

Boxplot allows the comparison of the shape of several batches of data. By arranging the boxplot of several batches of data in parallel on the same axis, information about the shape of the median, tail length, outliers, and distribution interval of the batches of data is clearly visible. The position of data points in a batch of data can be seen by comparing the outliers of each box plot. The size of the quartiles for each batch of data, and whether the distribution of normal values is concentrated or dispersed, is clear by looking at the length of each box and line segment. The skewness of the distribution of each batch of data can also be estimated by analyzing the median line and the location of the outliers. Combining boxplots with these analysis methods helps to make the analysis process easier and faster.

Chart, box and whisker chart

Description automatically generated

**4.4 Linear mixed model**

**4.4.1 Introduction of linear mixed model**

Fitting data with a linear regression model requires assuming that the data satisfy some classical assumptions. Namely, data independence and variance homogeneity. However, in real life, the data encountered often do not satisfy these assumptions, for example, some data can be correlated (longitudinal data, clustered data, etc.). Therefore, in order to be able to deal with such problems, Laird and Ware (1982) first proposed the linear mixed model, which is essentially a linear regression model based on the classical linear regression model with relaxed variance chi-squared conditions, so that the linear mixed model is widely used in various fields.

The linear mixed model is based on the linear regression model and introduces random effects in its linear part. Random effects can reflect not only the correlation between different observations of the same object, but also the heterogeneity between different objects. The general form of the linear mixed model is equation (1).

(1)

where vector Y contains the complete set of all observations, Y1, Y2, … , Yi. 𝛸 is the known fixed-effects design matrix, 𝛽 is the d-dimensional unknown fixed-effects parameter vector, 𝛧 is the random-effects design matrix, bi is the G-dimensional random-effects parameter vector, and 𝜀 is the random error vector. The basic assumptions of the model include.

(1) The covariate 𝑋 is non-random.

(2) The random effects and error terms have a mean of 0 and finite variance.

(3) The random effects and error terms are independent of each other.

**4.4.2 Analysis in linear mixed model**

In GitHub, PRs and projects are structured hierarchically: the PRs level is considered the micro level and the project level is considered the macro level. Given that our data contains a hierarchical structure, we chose to use a mixed effects model in our experiments for RQ2 and RQ3. Mixed effects models allow for accurate estimation of the relationship between individual-level explanatory variables and response variables while taking the population structure of data into account. Using mixed effects models avoids the potential problem of overestimating the importance of individual regression coefficients, which usually occurs when applying linear or logistic regression models to stratified data sets [35].

In our experiments, we build linear mixed-effects models (LMMs)), using the **lmer** function from the **lme4** package of R language. A mixed-effects model is a statistical model that includes both fixed and random effects. The fixed effects estimate the overall level coefficient, which is constant in the aggregate, while the random effects consider the differences in individual responses to a given effect [35]. Accordingly, in order to model mixed effects, we need to specify both fixed factors (with fixed effects) and random factors (with random effects).

Consider that different items may respond differently to the integration of PRs (e.g., some items may be more conservative in the integration of PRs or tend to take a long time to close PRs). To capture this item-level variability in the response variable, we introduce item IDs (11,230 items from 1 to 11,230) as a random factor when constructing the mixed-effects model. Setting the item ID as a random factor allowed the intercept to vary with the item rather than being fixed for all items. 23 metrics for the complexity of the PRs were set as fixed factors.

# Result & Conclusion

## RQ1: Whether the characteristics of PRs vary across projects

There are 663,088 JavaScript projects stored on GitHub, and this includes a wide variety of projects with significant or potential differences in problem areas, team cultures, and development processes. We focused on only 11,230 of these projects in our experiments, and they differed in many ways. However, many developers are involved in different projects, and the PRs submitted by developers may exhibit many similarities. Therefore, we first explored whether the PRs’ characters differ between these software development projects. I choose 3 most variable metrics from nine characters of 5 projects, since 11,230 is too large to deal on SPSS.

We choose to use exponential growth to represent the vertical coordinates, so that we can make the boxplot more representative of the data characteristics. However, exponential growth cannot represent data with a value of 0, and there are many 0 values in our experimental data, and our experimental data are integers. This does not affect our results in any way. Figures 3.2 to 3.10 show the box plots for the nine characters for all PRs in the 5 items, and Table 3.4 gives the variance results of the median data for the nine indicators.

Chart, box and whisker chart

Description automatically generated

Churn\_addition

Chart, box and whisker chart

Description automatically generated

Churn\_deletion

Chart, box and whisker chart

Description automatically generated

Comment number

|  |  |
| --- | --- |
| Churn\_addition | 33.32 |
| Churn\_deletion | 2.13 |
| Num\_comment | 0 |

From Figure 3.2 to 3.6, we can see that the median number of commits of PRs for each project in the experimental data is 1, which means that all projects show at least 50% of PRs with the same number of commits.

The number of lines of code added, lines of code deleted, lower quartile, median, and upper quartile of PRs for each project in the experimental data varies, especially for the number of lines of code added, as can be seen in Table 3.4. In other words, there are significant differences in the number of lines of code added, lines of code deleted, and the number of lines of code deleted for the PRs among the different items in the experimental data.

## RQ2: How do the characteristics of PRs relate to the acceptance of PRs

We collected the time costs of all 3,347,937 PRs in 11,230 projects. As described in Section 4.2 we decided to build a linear mixed-effects model (LMM) to analyze the correlation between the complexity of PRs and the PRs' merger decision. In the LMM, the response variable is the merge status of the closed PRs. The 23 metrics of the PRs introduced in Section 3.1.3 are used as fixed factors, while the item ID is set as a random factor. We standardize the fixed-factor variables using the scale function in R. This not only reduces multicollinearity between explanatory variables and between random intercepts and slopes, but also makes all explanatory variables relatively comparable [35] [42].

Chart, scatter chart

Description automatically generated

## RQ3: What is the relationship between the time cost of PRs and the time cost of closing PRs

# 

###### <Name of appendix>

<Use Heading 6 for the Appendix heading>

Pull Request characteristic influencing pull-based Acceptance

|  |  |  |
| --- | --- | --- |
| PR characters | Feature | Description |
| Size of change | churn\_addition | Number of lines of added code |
| Size of change | churn\_deletion | Number of lines of deleted code |
| Complexity | description\_length | Number of words in PR description |
| Nature of pull request | bug\_fix | Whether the PR fix any bug |
| Test inclusion | test inclusion | Whether the PR include any test |
| Reference of a contributor, issue or pr | hash\_tag | Whether the PR include ‘#’ in comments or description |
| Reference of a contributor, issue or pr | at\_tag | Whether the PR include ‘@’ in comments or description |
| Conflict | comment\_conflict | Whether the PR includes the word ‘conflict’ in the comment. |
| Hotness or relevance of PR | part\_num\_issue | Number of participants in the issue comment |
| Hotness or relevance of PR | part\_num\_commit | Number of participants in the commit comment |
| Hotness or relevance of PR | part\_num\_pr | Number of participants in the PR comment |
| Hotness or relevance of PR | part\_num\_code | Number of participants in the code comment |
| Hotness or relevance of PR | pr\_comment\_num | Number of PR comments |
| Emotions | perc\_pos\_emotion | Percentage of the positive emotion comments |
| Emotions | perc\_neg\_emotion | Percentage of the negative emotion comments |
| Emotions | perc\_neu\_emotion | Percent of the neutral emotion comments |
| Continuous integration | ci\_exists | Whether a PR use CI tools |
| Continuous integration | ci\_latency | The time from PR creation and the first build finish. |
| Continuous integration | ci\_build\_num | Number of Building CI |
| Continuous integration | ci\_test\_passed | Whether PR passed CI builds |
| Continuous integration | ci\_failed\_perc | Percentage of failed CI builds |
| Continuous integration | ci\_first\_build\_status | The status of first builds |
| Continuous integration | ci\_last\_build\_status | The status of last builds |

###### <Another appendix>

# Bibliography

[1] S. Chacon. Pro Git. Expert’s Voice in Software Development. Apress, 1rst edition, Aug 2009

[2] Barr E T , Bird C , Rigby P C , et al. Cohesive and Isolated Development with Branches[C]. fundamental approaches to software engineering, 2012: 316-331.

[3] <https://github.com/freeCodeCamp/freeCodeCamp/pulls?q=is%3Apr+is%3Aclosed>+

[4] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In Proceedings of the 36th International Conference on Software Engineering (Hyderabad, India) (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 345âĂŞ355. <https://doi.org/10.1145/2568225.2568260>

[5] Y. Jiang, B. Adams, and D. M. German. 2013. Will my patch make it? And how fast? Case study on the Linux kernel. In 2013 10th Working Conference on Mining Software Repositories (MSR). 101–110. <https://doi.org/10.1109/MSR.2013.6624016>

[6] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In Proceedings of the 36th International Conference on Software Engineering (Hyderabad, India) (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 356âĂŞ366. <https://doi.org/10.1145/2568225.2568315>

[7] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. 2016. Determinants of pull-based development in the context of continuous integration. Science China Information Sciences 59, 8 (2016), 080104. [https://doi.org/10.1007/s11432- 016-5595-8](https://doi.org/10.1007/s11432-%20016-5595-8)

[8]Georgios Gousios and Andy Zaidman. 2014. A Dataset for Pull-Based Development Research. In Proceedings of the 11th Working Conference on Mining Software Repositories (Hyderabad, India) (MSR 2014). Association for Computing Machinery, New York, NY, USA, 368âĂŞ371. https://doi.org/10.1145/2597073.2597122

[9]

[10] Estublier J . Software configuration management: A roadmap[C]. Proceedings of the Conference on The Future of Software Engineering, ICSE, 2000:279-289.

[11] Bird C , Rigby P C , Barr E T , et al. The Promises and Perils of Mining Git[M]. IEEE, 2009:1-

[12] Pilato C M, Collins-Sussman B, Fitzpatrick B W. Version control with subversion: next generation open source version control[M]. " O'Reilly Media, Inc.", 2008

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

[14] Gumm D C . Distribution Dimensions in Software Development Projects: A Taxonomy[M]. IEEE Computer Society Press, 2006.

[15] Mockus A, Fielding R T, Herbsleb J D, et al. Two case studies of open source software development: Apache and Mozilla[J]. ACM Transactions on Software Engineering and Methodology, 2002, 11(3): 309-346.

[16] Peter Weißgerber, Neu D , Diehl S . Small patches get in![C]. International Working Conference on Mining Software Repositories. ACM, 2008:67-76.

[17] Peter C Rigby , Daniel M German . A preliminary examination of code review processes in open source projects. Technical Report. DCS-305-IR, University of Victoria, 2006.

[18] Baysal O , Kononenko O , Holmes R , et al. Investigating technical and non-technical factors influencing modern code review[J]. Empirical Software Engineering, 2015, 21(3):932-959.

[19] Gousios G , Pinzger M , Deursen A V . An exploratory study of the pull-based software development model[C]// International Conference Software Engineering. ACM, 2014.

[20] Stijnen T，Hamza TH，Ozdemir P． Random effects meta-analysis of event outcome in the framework of the generalized linear mixed model with applications in sparse data． Stat M ed，2010，29( 29) : 3046-3067．

[21] Bogdan Vasilescu, Yue Yu, Huaimin Wang, et al. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub[C]. ESEC/FSE 2015: 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015:805-816.

[22] Rahman M M , Roy C K . An insight into the pull requests of GitHub[C]. Working Conference on Mining Software Repositories. ACM, 2014:364-367.

[23] Padhye R , Mani S , Sinha V S . A study of external community contribution to open-source projects on GitHub[C]. ACM, 2014:332-335.

[24] Kononenko O , Baysal O , Godfrey M W . Code review quality: how developers see it[C]. the 38th International Conference. ACM, 2016:1028-1038.

[25] Tsay J , Dabbish L , Herbsleb J . [ACM Press the 36th International Conference - Hyderabad, India(2014.05.31-2014.06.07)] Proceedings of the 36th International Conference on Software Engineering - ICSE2014 - Influence of social and technical factors for evaluating contribution in GitHub[C]. Icse. 2014:356-366.