A Dataset of Duplicate Pull-requests in GitHub

Yue Yu*, Zhixing Li*, Gang Yin, Tao Wang, Huaimin Wang
College of Computer, National University of Defense Technology
Changsha, China
{yuyue,lizhixing15,yingang,taowang2005,hmwang}@nudt.edu.cn

ABSTRACT

In GitHub, the pull-based development model enables community contributors to collaborate in a more efficient way. However, the distributed and parallel characteristics of this model pose a potential risk for developers to submit duplicate pull-requests (PRs), which increase the extra cost of project maintenance. To facilitate the further studies to better understand and solve the issues introduced by duplicate PRs, we construct a large dataset of historical duplicate PRs extracted from 26 popular open source projects in GitHub by using a semi-automatic approach. Furthermore, we present some preliminary applications to illustrate how further researches can be conducted based on this dataset.

KEYWORDS

Duplicate pull-request, distributed software development, GitHub

ACM Reference Format:
Yue Yu*, Zhixing Li*, Gang Yin, Tao Wang, Huaimin Wang. 2018. A Dataset of Duplicate Pull-requests in GitHub. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196398.3196455

1 INTRODUCTION

In GiHub, the pull-based mechanism [1, 4, 10] lowers the contribution entry for community developers and prompts the development and evolution of numerous open source software projects. Any contributor can fork (*i.e.*, clone) a repository and edit the forked repository locally without disturbing the original repository. After finishing their local work (*e.g.*, fixing bugs or proposing features), contributors package the code changes into a new Pull-Request (PR) and submit it to the original repository. And then the core members of the project and community users will launch the process of code review [3, 8] to detect potential defects contained in the submitted PR and discuss how to improve its quality. Finally, the PR which have went through several rounds of rigorous evaluations will be merged or rejected depending on its eventual quality by an integrator of the original repository.

However, due to the parallel and distributed nature of pull-based development model, more than one contributors would submit PRs to achieve a similar objective (*i.e.*, duplicate PRs [5]). Especially for the popular projects which attract thousands of volunteers and continuously receive incoming PRs [7, 11], it is hard to appropriately coordinate contributors' activities, because most of them work distributively and tend to lack information of others progress. Duplicate PRs increase the maintenance cost of GitHub and result in the waste of time spent on the redundant effort of evaluating each of them separately [1, 3]. Moreover, contributors may iteratively update and improve their PRs in several rounds of code reviews [11] driven by the feedbacks provided by reviewers. Therefore, the more late the duplicate relations between PRs are identified, the more efforts of contributors and reviewers may be wasted. Furthermore, improper management of duplicates may also lead the contributors to be more frustrated [6] and get doubtful about the core team.

Although several research has been conducted on analyzing the popularity [1], challenges [3, 5] and evaluations [10, 11] of PRs, the problem of duplicate PRs is left not well studied. More research, including empirical studies on the cause, outcome, challenge, and even influencing factor of duplicate PRs and automatic tool development used to help reviewers to detect and choose duplicates, need to be conducted to better understand and solve the issues introduced by duplicate PRs. To facilitate the further studies, we constructed a large dataset of historical duplicate PRs (called *DupPR*) extracted from 26 open source projects in GitHub. Each pair of duplicate PRs in *DupPR* has been manually verified after an automatic identification process, which would guarantee the quality of this dataset. We make the dataset and the source code available online, [1] in hope it will foster more interest in the following studies.

- Analyzing how much redundant effort would be wasted by duplicate PRs. This would give researchers a straightforward impression about how duplicate PRs negatively affect software development process.
- Investigating how reviewers make dicisions among similar contributions. It is necessary to build automatic tools that make more targeted comparisons between PRs and assist reviewers in managing duplicates.
- Training and evaluating the intelligent models for detecting duplicate PRs. Detecting duplicates at submission time can avoid redundant effort spent in quality evaluation.
- Exploring the factors that affect the occurrence probability of duplicate PRs. This makes it possible to recognize inefficient collaborative patterns that are more likely to generate duplicate contributions, and hence core members can propose corresponding strategies to avoid them.

* Yue Yu and Zhixing Li are both first authors, and contributed equally to the work.

[1] https://github.com/whystar/MSR2018-DupPR

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MSR '18, May 28–29, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196455

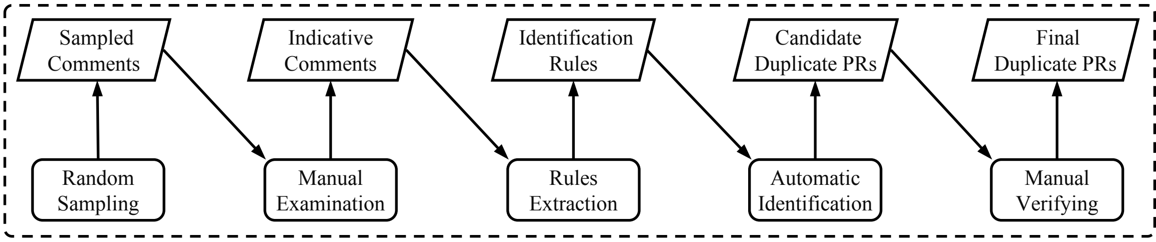

Figure 1: Process of collecting duplicate Pull-requests

2 DATA COLLECTION

2.1 Studied Projects

We study on 26 open source projects hosted in GitHub, involving 12 programming languages and various application domains (*e.g.*, web-application framework, database and scientific computing library). Table 1 presents some of statistical characteristics about the project scale and popularity, *e.g.*, the number of PRs, contributors and forks, which show that they have attracted plenty of attentions from the community. Also, we can assure that the studied projects have full-fledged and heavy usage of PR mechanism (minimum number of PRs is 5,050). More details can be found in the released dataset.

Table 1: The statistical information of studied projects

Statistic	Min	Max	Mean	Median
#PR	5,050	31,600	10,912	12,753
#Contributor	518	3,395	1,283	1,034
#Fork	1,759	55,075	9,317	5,131
#Star	1,277	117,220	25,290	16,917
#Watch	112	7,116	1,759	1,303

2.2 Method

Unlike Stack Overflow, which indicates duplicate posts with a signal "[duplicate]" at the end of question titles, GitHub provides no explicit and unified mechanism to indicate duplicate PRs. Although reviewers are encouraged to use the pre-defined reply template [2] when they intend to point out a PR is duplicate to another one, a variety of other comment presentations can also be applied. Therefore, to collect a comprehensive dataset of duplicate PRs in GitHub, we have to analyze and examine the historical review comments carefully. The raw data of projects, pull-requests and comments is easily available with the official API provided by GitHub, and hence the key point is how to collect duplicate PRs based on those raw data. In this paper, we design a mixed approach that combines automatic identification and manual examination, as illustrated in Figure 1, where the rounded rectangles stand for the actions we have taken and parallelograms represent the input data or output data of the actions. The details of our novel collecting process are discussed as follows.

2.2.1 Random sampling. For each project, we randomly sampled 200 review comments which contain at least one reference (*i.e.*, the number or the url of a PR) to another PR. Cross-PR references in review comments are the evidence that some kind of relation exists between two PRs. In fact, this is also the necessary condition for finding duplicate PRs because reviewers have to reference other PRs when they want to point out the duplicate relation among PRs. Using cross-PR references as a filter criteria in sampling can reduce the proportion of noise data in the sampled comments to be processed in the following action (*i.e.*, Manual examination) and therefore improve the examination efficiency.

2.2.2 Manual examination. For each sampled comment, we manually examine whether it is a comment that some reviewer uses to point out the duplicate relation among PRs. We call such kind of comments *indicative comments* which can help us to re-construct the duplicate relations. We would like to note that quite a number of sampled comments are not indicative comments. Cross-PR references can also be used to indicate the relation of conflict, dependency, or association among PRs.

2.2.3 Rules extraction. We review all the manually identified indicative comments and tried to extract rules which can be applied lately to automatically judge whether a given comment is an indicative comment. Actually, some phrases frequently occur when reviewers are stating the duplicate relation between PRs. Similarly, we call such phrases as *indicative phrases*. The followings are several example comments containing indicative phrases.

- *"dup of #31372 "*
- *"Closed by https://github.com/rails/rails/pull/13867"*
- *"This has been addressed in #27768."*

In the above example comments, *"dup of"*, *"closed by"*, and *"addressed in"* are all the typical indicative phrases. Together with PR references, these indicative phrases can be used to compose the identification rules. An identification rule can be implemented as a regular expression which is applied to match comment text to identify duplicate relations. The following items are some simplified rules, and the complete set of our rules can be found online. [3]

- `closed by (?:\w+:? ){,5} (?:#(\d+))`
- `(?:#(\d+)):? (?:\w+:? ){,5} dup(?:licate)?`

2.2.4 Automatic identification. According to the extracted identification rules, we can automatically identify the indicative comments and then discover the duplicate PRs. If a review comment is identified as an indicative comment, the PR references contained in the comment will be extracted immediately. Each of the extracted PRs and the PR that the indicative comment belongs to form a

[2] https://help.github.com/articles/about-duplicate-issues-and-pull-requests

[3] https://github.com/whystar/MSR2018-DupPR/blob/master/code/rules.py

couple of candidate duplicates. Actually, we have introduced some preliminary constrains for candidate duplicate PRs. For example, a couple of candidate duplicate PRs cannot be submitted by the same contributor. It is obviously that the same author is aware of the existence of both PRs which means the duplicate is intentional and the author submit duplicate PRs for some purpose. This kind of intentional duplicates are not taken into account in our dataset. Moreover, PRs and issues share the same numbering system in GitHub and issues may also be referenced by the same format as PRs like "#[number]". Therefore, we have to verify the extracted “PR” is really a PR, rather than an issue.

2.2.5 Manual verifying. It is inevitable that automatic identification may introduce false-positive errors, that is some identified candidate duplicate PRs are not really duplicate in fact. To further clean the automatically identified dataset, we manually examine and verify all the candidate duplicate PRs. For a couple of candidate duplicates, the early submitted one is called *master PR* and the late submitted one is called *duplicate PR* in the paper. And then we review each couple of candidate duplicates and label them as “really duplicate” if they meet the following criteria: (a) the author of duplicate PR is not aware of the existence of the master PR. It is the default assumption unless we can find obvious contrary evidence in the discussion history of both PRs. (b) reviewers have reached a consensus on the duplicate. When some reviewer points out a PR is duplicate to another one, it is necessary that this declaration is responded and affirmed. One of the most common responses is an immediate close of one of the duplicate PRs.

After manual verifying is finished, the final dataset of 2,323 pairs of duplicate PRs is constructed.

3 DESCRIPTION OF DATASET

We store the dataset *DupPR* in a MySQL database, and make the script files available online in GitHub. Figure 2 illustrates the schema of *DupPR*. There are four tables in *DupPR* and the fields in these tables are defined as follows.

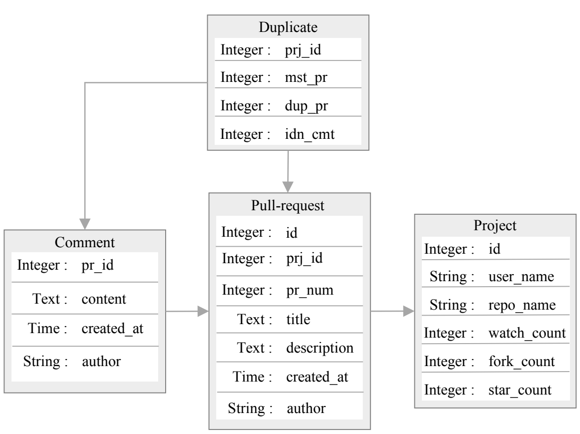

Figure 2: The Schema of *DupPR* dataset

- Table `Project` stores the basic information of studied projects. Field `user_name` is the name of the user owning the project in GitHub, and field `repo_name` is the name of the project. These two fields, together with the domain name of GitHub, can be used to compose the resource locator of the project in GitHub. Other fields in table `Project` present some statistical characteristics of a project, for example `fork_count` is the number of forks.
- For each project, all the PRs belonged to it are stored in table `Pull-request`. Field `prj_id` is the value of `id` of the project. and `pr_num`, `title` and `description` represent the number label (generated by GitHub), the title and the description of a PR respectively. Moreover, field `pr_num` can be used to uniquely locate a PR in the addressing space of a project in GitHub. Fields `author` and `created_at` mean a PR is submitted by the GitHub user named `author` at the time of `created_at`.
- For a pull-request in Table `Pull-request`, comments on it are stored in table `Comment`. For table `Comment`, filed `pr_id` is the value of `id` of the pull-request. The text content, the creation time and the author of a comment are represented by fields `content`, `created_at`, and `author` respectively.
- Table `Duplicate` contains all the duplicate PR-pairs. Field `prj_id` is the value of `id` of the project that a pair of duplicate PRs belong to. For a pair of duplicate PRs, field `mst_pr` is the number of the PR that is submitted early and filed `dup_pr` is the number of the PR that is submitted late. Field `idn_cmt` is the first indicative comment that points out the duplicate relation between `mst_pr` and `dup_pr`.

4 APPLICATIONS

To foster more interest in studying pull-based development based on this dataset (maybe sometimes together with GHTorrent [2] and GitHub API), we present some of our preliminary investigations.

4.1 Detection latency & redundant effort

First, we have explored the *detection latency* of duplicates. In this paper, detection latency is used to measure how long it takes to detect the duplicate relation between two PRs. It is defined as the time period from the submission time of a new PR to the time when the duplicate relation between it and a historical PR is identified. For each item in table `Duplicate`, the property `created_at` of `dup_pr` in table `Pull-request` is used as the submission time, and the property `created_at` of `idn_cmt` in table `Comment` is used as the identification time. Figure 3 shows the statistical distribution of the detection latency based on our dataset. There are nearly 21% (486) duplicates are detected after a relative long latency (more than one week). Those PRs probably have already consumed a lot of unnecessary manpower and computational resources (*e.g.*, continuous integration [9, 10]). In addition, we focus on how much redundant review effort has been costed by calculating the number of different reviewers and comments that are involved in the evolution process of duplicate PRs. According to our statistics, there are on average 2.5 reviewers participating in the redundant review discussions and 5.2 review comments are generated before the duplicate relation is identified.

4.2 Preference of choice

For each pair of duplicate PRs, reviewers have to make a choice between them or, in rare cases, make a combination. We have tried

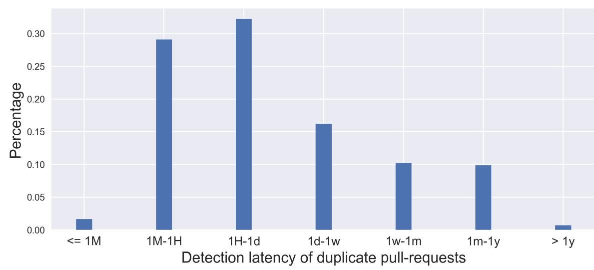

Figure 3: Distribution of detection latency

to figure out the reasons why integrators would prefer one pull-request to the other one. In brief, a winner PR contains some of prominent indicators: (a) correct implementation; (b) early submission (*i.e.*, first-come first-merge); (c) better implementation (*e.g.*, less changed codes or better performance); (d) providing test of changed codes; and (e) submitted by new contributors (reviewers prefer new contributors to encourage them to continuously contribute). Obviously, there are other factors that can affect reviewers' preference of choice, and we will conduct further research on this topic and analyze the influences of these factors. We would also investigate the effectiveness of the current practices of choice. For example, we are studying whether the preference for new contributors would increase the probability of introducing potential bugs, and whether it is necessary to dynamically adjust the strategy of choice according to the development status of a project.

4.3 Training & evaluating models

The dataset *DupPR* is constructed through a rigorous process which involves careful manual verifying. Thus, it can act as a ground truth to train and evaluate intelligent models (*e.g.*, classification model). Here, we conduct a preliminary experiment to automatically identify duplicate PRs. By employing natural language processing and calculating the overlap of changes, we measure the similarity between two PRs, and then return a candidate list of top-k historical PRs that are most similar with the submitted PR. We use half of *DupPR* to train an automatic detection model and use the rest to evaluate its performance. Figure 4 shows the identification results measured by *recall-rate@k*, which can achive nearly 70% when the size of candidate list is set to be 20.

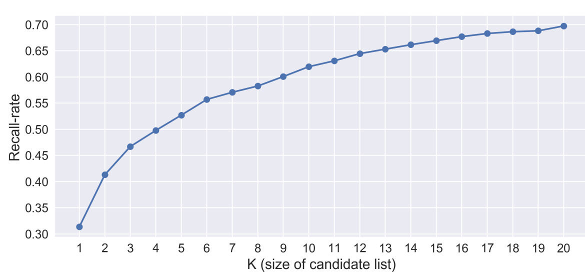

Figure 4: Performance of the automatic detection model

5 CONCLUSION

The distributed and parallel characteristics of pull-based development model on one hand enable community users to collaborative in a more efficient and effective way, but on the other hand carry contributors a potential risk of submitting duplicate PRs.

In this paper, we present a large dataset containing 2,323 pairs of duplicate PRs, collected from 26 popular open source projects hosted in GitHub. The dataset includes duplicate relations between PRs, the meta-data of PRs and reviews (*e.g.*, creation time, text content and author), and the basic information of the studied projects.

The dataset allows us to conduct empirical studies to understand the outcomes and issues of duplicates, explore the underlying causes and the corresponding prevention strategies, and analyze the practices and challenges of integrators and contributors in dealing with duplicates. Moreover, this dataset enables us to train and evaluate automatic models that can detect duplicate historical PRs for a newly submitted PR.

However, this dataset still has several limitations. The studied projects are only a relatively small proportion of all the projects hosted in GitHub. We plan to enrich the dataset by taking more projects into consideration. In addition, identification rules are extracted based on sampled comments and therefore the set of rules might be incomplete which would result in false negatives in the dataset. In future work, we would like to continually improve the identification method. At the meantime, by sharing both the dataset and guidelines for recreation, we intend to encourage other researchers to validate and extend the dataset.

REFERENCES

[1] 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, ICSE.* 345–355.

[2] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: Github's data from a firehose. In *Proceedings of the 9th Working Conference on Mining Software Repositories, MSR.* 12–21.

[3] Georgios Gousios, Margaret Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference Software Engineering, ICSE.* 285–296.

[4] Georgios Gousios and Andy Zaidman. 2014. A Dataset for Pull-based Development Research. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR.* 368–371.

[5] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering, ICSE.* 358–368.

[6] Wenjian Huang, Tun Lu, Haiyi Zhu, Guo Li, and Ning Gu. 2016. Effectiveness of Conflict Management Strategies in Peer Review Process of Online Collaboration Projects. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing.* 717–728.

[7] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Kenichi Matsumoto. 2015. Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER.* 141–150.

[8] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the 22Nd ACM International Symposium on Foundations of Software Engineering, FSE.* 144–154.

[9] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, FSE.* 805–816.

[10] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for it: Determinants of pull request evaluation latency on GitHub. In *Proceedings of the 12th Working Conference on Mining Software Repositories, MSR.* 367–371.

[11] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.