Detecting Duplicate Pull-requests in GitHub

Zhixing Li, Gang Yin*, Yue Yu, Tao Wang, Huaimin Wang
National Laboratory for Parallel and Distributed Processing
College of Computer, National University of Defense Technology
Changsha, China 410073
{lizhixing15,yingang,yuyue,taowang2005,hmwang}@nudt.edu.cn

ABSTRACT

The widespread use of pull-requests boosts the development and evolution for many open source software projects. However, due to the parallel and uncoordinated nature of development process in GitHub, duplicate pull-requests may be submitted by different contributors to solve the same problem. Duplicate pull-requests increase the maintenance cost of GitHub, result in the waste of time spent on the redundant effort of code review, and even frustrate developers' willing to offer continuous contribution. In this paper, we investigate using text information to automatically detect duplicate pull-requests in GitHub. For a new-arriving pull-request, we compare the textual similarity between it and other existing pull-requests, and then return a candidate list of the most similar ones. We evaluate our approach on three popular projects hosted in GitHub, namely Rails, Elasticsearch and Angular.JS. The evaluation shows that about 55.3% - 71.0% of the duplicates can be found when we use the combination of title similarity and description similarity.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; **Collaboration in software development**;

KEYWORDS

Pull-request, code review, duplicate detection, textual similarity

ACM Reference format:
Zhixing Li, Gang Yin*, Yue Yu, Tao Wang, Huaimin Wang. 2017. Detecting Duplicate Pull-requests in GitHub. In *Proceedings of Internetware'17, Shanghai, China, September 23, 2017,* 6 pages.
https://doi.org/10.1145/3131704.3131725

1 INTRODUCTION

A critical factor of the rapid development and evolution for many open source software projects in GitHub is the use of pull-based development model [5–7, 28]; this model allows any developer (core members of a project or external developers) to contribute to a public project by submitting *pull-requests*. Contributors fork (*i.e.*, clone) a fascinating repository and make their changes on the cloned repository locally without disturbing the original repository [5, 28]. On the base of cloned repository, contributors can fix bugs, add new features, improve documents and etc.. When their changes are ready to merge back into the main repository, they create a pull-request to notify the core team of the project to review the submitted changes [16, 23, 26], which is an important software quality assurance.

Contributing in pull-based model is a parallel and uncoordinated process [2, 6, 7]. Therefore, duplicate pull-requests may be created by different developers to address exactly the same problem, especially for the popular projects which attract numerous contributors and receive plenty of pull-requests everyday. Duplicate pull-requests increase the maintenance cost of GitHub and result in the waste of time spent on the redundant effort of reviewing each of them separately. The statistics based on our data reveals that before a duplicate pull-request is identified, there are on average 2.6 reviewers participating in the review discussion and 5.2 review comments are generated. Furthermore, contributors often continuously improve their pull-requests driven by the code review feedback [12, 26], and therefore late identification of duplicates tend to lead the contributors to be more frustrated [8] and get doubtful about the management team when they have paid plenty of effort in several round of contribution improvements and code reviews, especially if their pull-requests are superseded as duplicate of the subsequently created ones.

The current practice is to count on the code reviewers to identify these duplicate pull-requests manually. Unfortunately, the number of pull-requests submitted daily, however, can be too large to cope with for reviewers of popular projects [5, 26]. Moreover, it is not realistic for reviewers to keep all the historical pull-requests in mind and compare each of them with the newly-submitted one. As a result, many duplicate pull-requests cannot be identified in time. In spite of so much effort that have been spent on the evaluation of pull-requests [3, 9, 21–23, 27], very few research is conducted to assist pull-request management. This highlights the need for an automated tool which can be used to detect duplicate pull-requests at an early stage.

In this paper, we explored whether text information, that is textual similarity, can be used to automatically detect duplicate pull-requests in GitHub. For a given pull-request, we compute the textual similarity between it and other existing pull-requests, and then return a candidate list of the most similar ones. Textual information of a pull-request consists of two parts: title and description. Therefore, we investigated the detection performance of title similarity, description similarity and the combined similarity of them respectively. Based on the test dataset of duplicate pull-requests that we collected from three popular projects hosted in GitHub,

*Corresponding author

Internetware'17, September 23, 2017, Shanghai, China

ACM ISBN 978-1-4503-5313-7/17/09…$15.00
https://doi.org/10.1145/3131704.3131725

namely Rails, Elasticsearch and Angular.JS. We evaluate our approach in terms of recall-rate. The evaluation result shows that about 55.3% - 71.0% of the duplicates can be found when we use the combination of title similarity and description similarity.

To the best of our knowledge, we are the first to investigate how to automatically detect duplicate pull-requests in GitHub. The key contributions of this study include the following:

- We propose the problem of detecting duplicate pull-requests in GitHub. And we construct a dataset[1] of duplicate pull-requests by automatic identification and manual examination.
- We explore using textual similarity including title similarity, description similarity and their combination to detect duplicate pull-requests.
- The evaluation result shows textual similarity can be used to detect 55.3% - 71.0% duplicates which means it has practical value and can be integrated into the issue tracking systems.

The rest of paper is organized as follows: Section 2 illustrates the background. Section 3 presents the approach of our study in detail, and Section 4 elaborates the conducted experiments and reports the evaluation result. Threats and related work can be found in Section 5 and Section 6. Finally, we draw our conclusion in section 7.

2 BACKGROUND

2.1 Pull-based development model

In GitHub, a growing number of developers contribute to the open source projects by submitting pull-requests [5, 28]. As illustrated in Figure 1, a typical contribution process based on pull-based development model in GitHub involves the following actions.

- *Fork:* A contributor can find an interesting project by following several well-known developers and watching their projects. Before contributing, the contributor has to fork the original project.
- *Edit:* Based on the clone repository, the contributor can edit locally without disturbing the main branch in the original repository. He is free to do whatever he wants, such as implementing a new feature or fixing bugs.
- *Submit:* When his work is finished, the contributor submits the changed codes from the forked repository to its source by a pull-request. Except for commits, the submitter needs to provide a title and description to elaborate on the objective of his pull-request.
- *Review:* All developers in the community have the chance to discuss that pull-request in the issue tracker, with the existence of pull-request description, changed files, and test results provided by the Continuous Integration (CI) server. They have to ensure the pull-request does not break the current runnable state or raise other code defects.
- *Update:* After receiving the feedback from reviewers, the contributor updates his pull-request by attaching new commits for another round review.
- *Decide:* A responsible manager of the core team considers all the opinions of reviewers and merges or rejects the pull-request.

[1] https://www.trustie.net/attachments/download/198047/duppr.txt

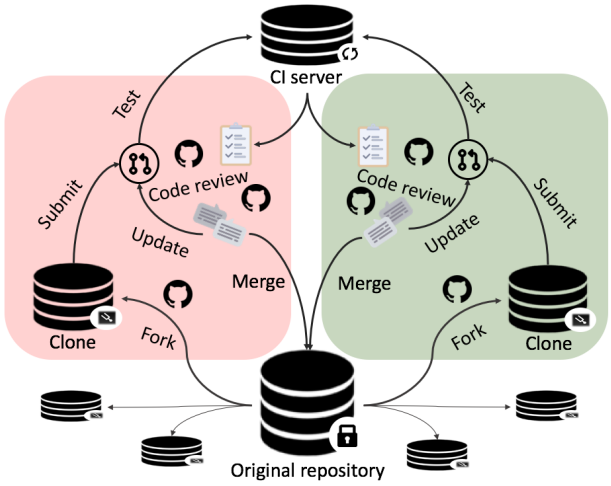


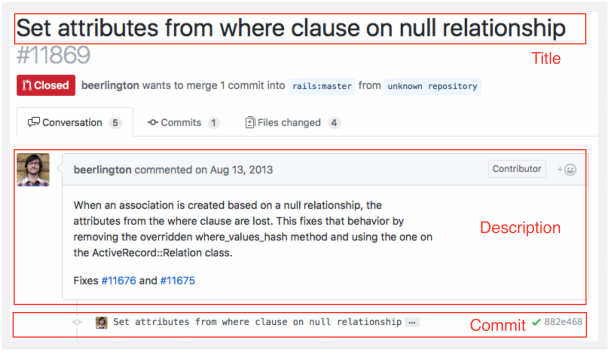
Figure 1: Pull-based work flow on GitHub

Pull-based development model lowers the contribution entry for outside developers and an increasing number of developers are motivated to participate in the development of open source software. However, there is a problem with this model when multiple developers are contributing voluntarily without appropriate coordination. This results in the probability of duplicates. On the other hand, although each submitted pull-request in GitHub has to go through a rigorous process of manual code review, it is not possible that duplicate pull-requests are detected immediately due to the huge volume of incoming and active pull-requests. Duplicates result in redundant effort of not only the initial work before their submission but also the review and update activities around them after the submission. The above challenges highlight the need for an automated tool to detect duplicate pull-requests at an early stage.

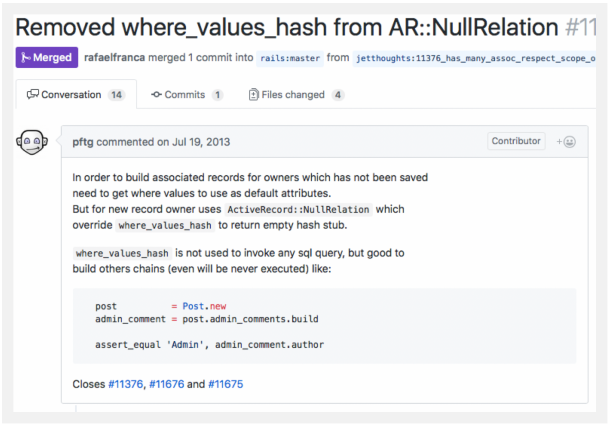
2.2 Duplicate pull-requests

Duplicate pull-requests are intended to achieve the same objective, *e.g.*, fixing the same bug or proposing features of equivalent function. Multiple pull-requests can be reviewed in parallel. As a consequence, sometimes a pull-request which is submitted early but is still in progress may be recognized by reviewers as duplicate to a subsequent pull-request that has been approved or rejected. In the paper, for the consistency of study, we always denote the first submitted pull-request as the *master pull-request* and the subsequent pull-requests are referred to the *duplicate pull-requests*.

For example, Figure 2 shows a duplicate pull-request (*Rails #11869*) and its master pull-request (*Rails #11496*). Both of them intend to resolve the problem of association which is based on null relationship. From the figure, we can that the titles and descriptions of these two pull-requests share some same words. Textual similarity has been actually applied by many precious studies [11, 13, 18, 25] to detect duplicate contents in software development (*e.g.*, bug reports). Therefore, in the paper, we explore whether textual similarity can be used to detect duplicate pull-requests.



(a) github.com/rails/rails/pull/11869



(b) github.com/rails/rails/pull/11496

Figure 2: Two duplicate pull-requests of Rails in GitHub. (a)Pull-request #11869 (b)Pull-request #11496

3 APPROACH

The goal of our work is automatically detecting duplicate pull-requests. As shown in Figure 3, for a given pull-request, our method returns a list of candidate duplicate pull-requests by computing and ranking the textual similarities between it and other history pull-requests. We determine the textual similarity between pull-requests from different perspectives: *title similarity*, *description similarity* and their combination. In the following sections, we will elaborate each step in detail.

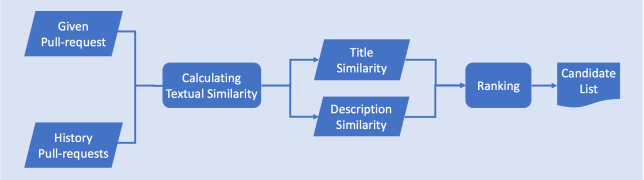


Figure 3: Overall framework of our method

3.1 Calculating Textual Similarity

Our method adopts the traditional natural language processing (NLP) techniques [10] to calculate textual similarity between two pull-requests. We mainly use the text information from the title and description of pull-requests.

Preprocessing. We perform the standard preprocessing in the extracted text including tokenization, stemming and stop words removal. Different strategies can be applied to split a sentence into tokens depending on the type of data and application domain [18]. There are some types of text which are split into multiple tokens in common settings but we want to treat them as a single token in the context of pull-requests. For example, code paths and hyper links usually indicate one concept and they should not be divided into separate words. As a result of that, we use the regular tokenizer to parse the raw text. The following are some example regular expressions and the matched text.

- code path
  - \w+(?:\:\:\w+)*
  - *"ActionDispatch::Http::URL"*
- number of pull-requests
  - \#\d+
  - *"#10319"*

After tokenization, each word will be stemmed to its root form (*e.g.*, *"was"* to *"be"* and *"errors"* to *"error"*) with the help of Porter stemming algorithm [14]. Finally, common stop words (*e.g.*, *"the"*, *"we"*, *"a"*), which appear so frequently that they contribute very few in distinguishing different documents, will be removed.

Transformation. We then transform the preprocessed text into multi-dimensional vector which is computable in Vector Space Model (VSM). A text is represented as: $TextVec_i = (w_{i,1}, w_{i,2}, ..., w_{i,v})$. Each dimension of the vector corresponds to an unique word in the corpus built by all the text. The value of $w_{i,k}$, the weight of the k-th item of the vecotr of i-th text, is computed by the TF-IDF model [20]:

$$w_{i,k} = tf_{i,k} \times idf_k \tag{1}$$

In the above formula, $tf_{i,k}$ denotes the *term frequency* which is the frequency of k-th term appearing in the i-th text and $idf_{i,k}$ denotes the *inverse term frequency* which measures the distinguishing characteristic of a term.

Similarity. After transforming text into vectors, we compute the similarity between a pair of text using *Cosine* [10] measure which is calculated by the following formula:

$$\begin{aligned} Sim(i,j) &= \frac{TextVec_i \cdot TextVec_j}{|TextVec_i||TextVec_j|} \\ &= \frac{\sum_{m=1}^{m=v}(w_{i,m} \times w_{j,m})}{\sqrt{\sum_{m=1}^{m=v} w_{i,m}^2}\sqrt{\sum_{m=1}^{m=v} w_{j,m}^2}} \end{aligned} \tag{2}$$

By applying the this measure, we can obtain two kinds of similarities between two pull-requests: $Sim_{title}(i,j)$ and $Sim_{desc}(i,j)$. $Sim_{title}(i,j)$ measures the similarity between titles while $Sim_{desc}(i,j)$ measures the similarity between descriptions.

3.2 Ranking Candidate List

After Sim_{title}, and Sim_{desc} between pull-request pairs are calculated, we are able to retrieve the target pull-requests according to these two kinds of similarities. To produce the candidate list of top-k pull-requests for the given pull-request, we use the combined similarity [25] computed by the following formula.

$$SimText(i,j) = Sim_{title}(i,j) + Sim_{desc}(i,j) \quad (3)$$

In the formula, we use a straightforward heuristic computing the arithmetic average of $Sim_{title}(i,j)$ and $Sim_{desc}(i,j)$ to get the final textual similarity which is the most widely used combination function. Finally, the top-k pull-requests which get the most-high *Sim* with the given pull-request will be returned in the candidate list.

4 EXPERIMENT & EVALUATION

4.1 Experimental Setup

We conducted the case study on three large projects. To increase the generalization of our study, these three projects are selected as of different domains and different programming languages. As shown in Table 1, Rails is a well-known web framework written in Ruby, Elasticsearch is a search server written in Java, and Angular.js is a popular front-end framework . Ruby, Java and JavaScript are three of the top most popular programming languages used on GitHub. The dataset includes meta data (*i.e.*, title, description) and review history of pull-requests.

Table 1: Dataset of our experiments

Projects	Language	Application Area	Hosted_at	#PR
Rails	Ruby	Web Framework	2009	18321
Elasticsearch	Java	Search Server	2010	10683
Angular.js	JavaScript	Front-end Framework	2010	7371

We would like to point out that there is no explicit mark for duplicate pull-requests in GitHub. Hence, we have to first figure out the duplicates from the historical data and construct a standard dataset for the subsequent analysis. To achieve this, we hold two criterion: (a) some reviewer comments that two pull-requests are duplicate to each other, and (b) other reviewers come to an agreement at this and close one of them. The construction process consists of two step: *automatic identification* and *manual examination.*

Automatic identification. When reviewers are commenting on a pull-request, they mention other related pull-requests by leaving a link to them. The following are some example comments which reference the duplicate pull-request.

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

From these comments, we recognize some indicative phrases such as *"dup of"*, *"in favaor of"* and *"closed by"*. These phrases co-occur frequently with links in form of pull-request numbers (*e.g.*, the first example comment) or hyper links (*i.e.*, the second example comment). This enables us to match these comments using regular expressions and the following is part of the expression set:

```
(dup|addressed|close) (\w+ )?(by|of) (#\d+|http)
```

If a comment of a pull-request is matched by the regular expression above, we will extract the referred pull-request with the link from the comment and construct them as a candidate duplicate pair. However, not all matched links actually direct to pull-requests, and are therefore not possible to extract to get the referenced pull-request. Those invalid links are usually release versions (*e.g.*, *"addressed in rails 5.1"*) and hash codes of a specific commit (*e.g.*, *"closed by efxxxx"*).

Manual examination. Regular expression match may introduce false-positive error. Therefore, we manually examined each candidate duplicate pair and removed the false matches. Moreover, we also eliminate duplicate pull-requests pairs that are submitted by the same author and those whose submitters are aware of the existence of the corresponding master pull-request and submit the duplicate pull-request for improvement purpose or just merging the master pull-request with higher privilege. At last, a dataset of 746 duplicate pull-requests is constructed.

4.2 Evaluation Metrics

To evaluate the performance of our method, we apply the *recall-rate@k* metric proposed by *Runeson et al.* [18] which has been widely applied by other studies [19, 20]. Formula 4 defines how recall-rate@k is calculated.

$$recall\text{-}rate@k = \frac{N_{detected}}{N_{total}} \quad (4)$$

In the equation, $N_{detected}$ is the number of duplicate pull-requests whose corresponding masters are detected in the candidate list of top-K pull-reqeusts, while N_{total} is the total number of duplicate pull-requests that are used for testing. In terms of Recall-rate, detection approaches can be assessed by calculating the percentage of duplicates which find their masters in the candidate list. Moreover, the k in recall-rate varies from 1 to 20.

4.3 Experiment Results

Previous work has studied how to use text information to detect duplicate development tasks such as bug reports. In the paper, we want to investigate whether the text information can be used to detect duplicate pull-requests. To answer this question, we conduct experiments with several options: using title similarity, using description similarity, using title similarity and description similarity. Moreover, *Runeson et al.* [18] suggested that title should be weighted more than the regular description for its more concise description. Therefore, we also set an option using doubled title similarity and description similarity. To measure the performance of each option, we compute recall-rate@k (k ranges from 1 to 20) when different options are applied.

In Figure 4, we present the recall-rate@1 to recall-rate@20 scores achieved by each option on the three software projects. We use *TS*, *DS*, *TDS* and *T2DS* as the abbreviations for the four options respectively (*i.e.*, *TS*: Title Similarity, *DS*: Description Similarity, *TDS*: Title Similarity + Description Similarity, *T2DS*: doubled Title Similarity + Description Similarity).

From the result we can see that title similarity (*TS*) always achieves better performance than description similarity for Rails . For example, setting the size of top list to 20, title similarity finds

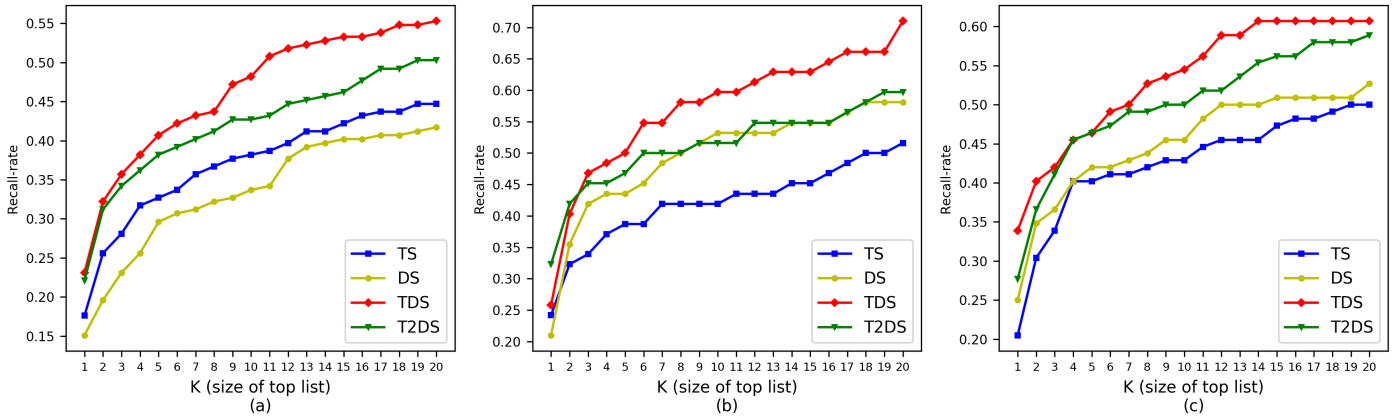


Figure 4: Detection performance of each kind of similarity measure method. (a) Rails, (b) Elasticsearch, (c) Angular.JS

44.7% duplicates compared with 41.7% which are found by description similarity. Therefore, the improvement of recall-rate@20 of title similarity is 7.2% for Rails. While for ElasticSearch and Angular.JS, description similarity is better than title similarity and the improvements of recall-rate@20 are 12.6% (58.1% vs 51.6%) and 5.4% (52.7% vs 50.0%) respectively.

We also find that combination of title similarity and description similarity produces higher recall-rate scores than any single similarity. The recall-rate@20 scores of *TDS* and *T2DS* are 55.3% and 50.3% for Rails, 71.0% and 59.7% for ElasticSearch and 60.7% and 58.9% for Angular.JS respectively. Obviously, compared with the single similarity (*TS* or *DS*), the improvement of combination similarity is Significant. However, the surprising is that using double title similarity does not achieve better performance than not using double title similarity which is inconsistent with the finding of *Runeson et al.* [18].

We can note that textual similarity can achieve a recall-rate@20 of 55.3% - 71.0% which we believe to be reasonably good. This means that detecting duplicate pull-requests using textual information has practical value and can be integrated into the issue tracking systems.

5 THREATS TO VALIDITY

In this section, we discuss some threats to validity which may affect the experiment results of our study.

Our experiment results are based on some of the popular open source projects hosted on GitHub. The projects are developed by various programming languages and are applied in different domains. However, it is unknown whether our method can be generalized to all the projects in GitHub, other open source projects hosted on other platforms.

In addition, our collection approach, which is used to construct the test dataset, may not find all the duplicate pull-requests because the regular expressions may not match all the review comments that indicate a pull-request is duplicate. Moreover, some reviewers may just close the duplicate pull-requests and do not leave any comments. In the future, we plan to collect more projects and enrich our test dataset to further validate the effectiveness of our method.

6 RELATED WORK

6.1 Duplicate Detection

Although vefy few work studies on duplicate detection of pull-requests, many work investigate how to recognise duplicate bug reports. Runeson *et al.* [18] is one of the first such studies. They evaluated how NLP techniques support duplicates identification and found NLP techniques can found 40% marked duplicates. Wang *et al.* [25] proposed an approach to detect duplicate bug reports by comparing the natural language information and execution information between the new report and the existing reports. Sun *et al.* [20] used discriminative models to detect duplicates and their evaluation on three large software bug repositories showed that their method achieved improvements compared with methods using natural language. Nguyen *et al.* [13] modeled each bug report as a textual document and took advantage of both IR-based features and topic-based features to learn the sets of different terms used to describe the same problems. Lazar *et al.* [11] made use of a set of new textual features and trained several binary classification models to improve the detection performance. Moreover, Zhang *et al.* [29] investigated to detect duplicate questions in Stack Overflow. They measured the similarity of two questions by comparing observable factors including titles, descriptions, and tags of the questions and latent factors corresponding to the topic distributions learned from the descriptions of the questions.

6.2 Pull-request & code review

Although research on pull-requests is in its early stages, several relevant studies have been conducted. Gousios *et al.* [5, 15] conducted a statistical analysis of millions of pull-requests from GitHub and analyzed the popularity of pull-requests, the factors affecting the decision to merge or reject a pull-request, and the time to merge a pull-request. Tsay *et al.* [22] examined how social and technical information are used to evaluate pull-requests. Yu *et al.* [28] conducted a quantitative study on pull-request evaluation in the context of CI. Moreover, Yue *et al.* [27] proposed an approach that combines information retrieval and social network analysis to

recommend potential reviewers. Veen *et al.* [24] presented PRioritizer, a prototype pull-request prioritization tool, which works to recommend the top pull-requests the project owner should focus on.

Code review is employed by many software projects to examine the change made by others in source codes, find potential defects, and ensure software quality before they are merged [3]. Traditional code review, which is also well known as the code inspection proposed by Fagan [4], has been performed since the 1970s. However, its cumbersome and synchronous characteristics have hampered its universal application in practice [1]. With the occurrence and development of VCS and collaboration tools, Modern Code Review (MCR) [17] is adopted by many software companies and teams. Different from formal code inspections, MCR is a lightweight mechanism [21] that is less time consuming and supported by various tools. Code review has been widely studied by from several perspectives including automation of revIew task [21], factors influencing review outcomes [3, 22] and problems found in code review [1].

7 CONCLUSION

In this paper, we conducted a preliminary study on automatically detecting duplicate pull-requests in GitHub using natural language information. Our method uses title and description of pull-request to calculate the textual similarity between two pull-requests and return a candidate list of the most similar one with the given pull-request. We constructed a test dataset of duplicates through a semi-automatic way from three popular projects hosted in GitHub including Rails, Elasticsearch and Angular.JS. The evaluation result shows that the combination of title similarity and description similarity can find about 55.3% - 71.0% of the duplicates.

In the future, we plan to enrich our test dataset and evaluate our method with datasets from more software projects. In addition, we would also develop better techniques to improve the detection effectiveness by considering other information such as code changes.

ACKNOWLEDGMENT

The research is supported by the National Natural Science Foundation of China (Grant No.61432020, 61303064, 61472430, 61502512) and National Grand R&D Plan (Grant No. 2016YFB1000805).

REFERENCES

[1] A Bacchelli and C Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *International Conference on Software Engineering*. 712–721.
[2] Earl T Barr, Christian Bird, Peter C Rigby, Abram Hindle, Daniel M German, and Premkumar Devanbu. 2012. Cohesive and isolated development with branches. In *International Conference on Fundamental Approaches To Software Engineering*. 316–331.
[3] Olga Baysal, Oleksii Kononenko, and Reid Holmes. 2016. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering* 21, 3 (2016), 1–28.
[4] Michael E Fagan. 2001. Design and code inspections to reduce errors in program development. In *Pioneers and Their Contributions to Software Engineering*. Springer, 301–334.
[5] Georgios Gousios, Martin Pinzger, and Arie Van Deursen. 2014. An exploratory study of the pull-based software development model. In *International Conference Software Engineering*. 345–355.
[6] Georgios Gousios, Margaret Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *International Conference Software Engineering*. 285–296.
[7] 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*. IEEE, 358–368.
[8] 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 *ACM Conference on Computer-Supported Cooperative Work & Social Computing*. 717–728.
[9] Jing Jiang, Jia Huan He, and Xue Yuan Chen. 2015. CoreDevRec: Automatic Core Member Recommendation for Contribution Evaluation. *Journal of Computer Science and Technology* 30, 5 (2015), 998–1016.
[10] Paul Kantor. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press,. pags. 91–92 pages.
[11] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Working Conference on Mining Software Repositories*. 308–311.
[12] Zhixing Li, Yue Yu, Gang Yin, Tao Wang, Qiang Fan, and Huaimin Wang. 2017. Automatic Classification of Review Comments in Pull-based Development Model. In *International Conference on Software Engineering and Knowledge Engineering*.
[13] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *ase*. 70–79.
[14] M. F Porter. 1997. *An algorithm for suffix stripping*. Morgan Kaufmann Publishers Inc. 130 – 137 pages.
[15] Peter C Rigby, Alberto Bacchelli, Georgios Gousios, and Murtuza Mukadam. 2014. *A Mixed Methods Approach to Mining Code Review Data: Examples and a study of multi-commit reviews and pull requests*. 231–255 pages.
[16] Peter C Rigby, Alberto Bacchelli, Georgios Gousios, and Murtuza Mukadam. 2015. *A Mixed Methods Approach to Mining Code Review Data: Examples and a study of multi-commit reviews and pull requests*. 231–255 pages.
[17] Peter C. Rigby and Margaret Anne Storey. 2011. Understanding broadcast based peer review on open source software projects. In *International Conference on Software Engineering*. 541–550.
[18] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of Duplicate Defect Reports Using Natural Language Processing. In *International Conference on Software Engineering*. 499–510.
[19] Chengnian Sun, D Lo, Siau Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *Ieee/acm International Conference on Automated Software Engineering*. 253–262.
[20] Chengnian Sun, D Lo, Xiaoyin Wang, and Jing Jiang. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *ACM/IEEE International Conference on Software Engineering*. 45–54.
[21] 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 *IEEE International Conference on Software Analysis, Evolution, and Reengineering*. 141–150.
[22] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *ICSE*. 356–366.
[23] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *The ACM Sigsoft International Symposium*. 144–154.
[24] Erik Van Der Veen, Georgios Gousios, and Andy Zaidman. 2015. Automatically prioritizing pull requests. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 357–361.
[25] Xiaoyin Wang and Lu Zhang. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *icse*. 461–470.
[26] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Reviewer recommender of pull-requests in GitHub. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 609–612.
[27] 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.
[28] 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), 1–14.
[29] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. 2015. Multi-Factor Duplicate Question Detection in Stack Overflow. *Journa of Computer Science and Technology* 30, 5 (2015), 981–997.