

A Dataset for Pull-Based Development Research

Georgios Gousios
Delft University of Technology
Delft, the Netherlands
g.gousios@tudelft.nl

Andy Zaidman
Delft University of Technology
Delft, the Netherlands
a.e.zaidman@tudelft.nl

ABSTRACT

Pull requests form a new method for collaborating in distributed software development. To study the pull request distributed development model, we constructed a dataset of almost 900 projects and 350,000 pull requests, including some of the largest users of pull requests on Github. In this paper, we describe how the project selection was done, we analyze the selected features and present a machine learning tool set for the R statistics environment.

Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—*Version control*; D.2.9 [Software Engineering]: Management—*Programming teams*

General Terms

Management

Keywords

pull-based development, pull request, distributed software development, empirical software engineering

1. INTRODUCTION

Pull requests as a distributed development model in general, and as implemented by Github in particular, form a new method for collaborating on distributed software development. In the pull-based development model, the project's main repository is not shared among potential contributors; instead, contributors *fork* (clone) the repository and make their changes independent of each other. When a set of changes is ready to be submitted to the main repository, they create a pull request, which specifies a local branch to be merged with a branch in the main repository. A member of the project's core team is then responsible to inspect the changes and pull them to the project's master branch. If changes are considered unsatisfactory, more changes may be requested; in that case, contributors need to update their local branches with new commits. Furthermore, as pull requests only specify branches from which certain

commits can be pulled, there is nothing that forbids their use in the shared repository approach (cross-branch pull requests).

To understand what the underlying principles that guide pull-based development are, we created *pullreqs*, a curated dataset of almost 900 projects along with a set of tools for its analysis. A previous version of the dataset has been used to quantitatively study the pull request development process [8]. The *pullreqs* dataset is based on our previous work on GHTorrent [7], albeit only for its construction. While GHTorrent is a full mirror of all data offered by the Github API, the *pullreqs* dataset includes many features extracted by combining GHTorrent and the project's repository; the dataset is offered in a format ready to be processed by statistical software. In this paper, we describe the construction process of the dataset and outline directions for further research with it.

2. FEATURE SELECTION

The feature selection was based on prior work in the areas of patch submission and acceptance [12, 3, 15, 2], code reviewing [14], bug triaging [1, 6] and also on semi-structured interviews of Github developers [5, 13]. The selected features are split into three categories:

Pull request characteristics. These features attempt to quantify the impact of the pull request on the affected code base. When examining external code contributions, the size of the patch is affecting both acceptance and acceptance time [15]. There are various metrics to determine the size of a patch that have been used by researchers: code churn [12], changed files [12] and number of commits. In the particular case of pull requests, developers reported that the presence of tests in a pull request increases their confidence to merge it [13]. To investigate this, we split the churn feature into two features, namely *src_churn* and *test_churn*. The number of participants has been shown to influence the time required to do a code review [14]. Finally, through our own experience analyzing pull requests, we have found that in many cases conflicts are reported explicitly in pull request comments while in other cases pull requests include links to other related pull requests. We therefore included corresponding binary features in the dataset.

Project characteristics. These features quantify how receptive to pull requests a project is. If the project's process is open to external contributions, then we expect to see an increased ratio of external contributors over team members. The project's size may be a detrimental factor to the speed of processing a pull request, as its impact may be more difficult to assess. Also, incoming changes tend to cluster over time (the "yesterday's weather" change pattern), so it is natural to assume that pull requests affecting a part of the system that is under active development will be more likely to merge. Testing plays a role in speed of processing; according to [13], projects struggling with a constant flux of contributors use

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'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...\$15.00
<http://dx.doi.org/10.1145/2597073.2597122>

testing, manual or preferably automated, as a safety net to handle contributions from unknown developers.

Developer. Developer-based features quantify the influence that the person who created the pull request has on the decision to merge it and the time to process it. In particular, the developer who created the patch has been shown to influence the patch acceptance decision [10]. To abstract the results across projects with different developers, we include features that quantify the developer’s track record [5], namely the number of previous pull requests and their acceptance rate; the former has been identified as a strong indicator of pull request quality [13]. Bird et al. [4], presented evidence that social reputation has an impact on whether a patch will be merged; in our dataset, the number of followers on Github can be seen as a proxy for reputation.

All features must be calculated at the time a pull request has been closed or merged, to evaluate the effect of intermediate updates to the pull request as a result of the ensuing discussion. Features that contain a temporal dimension in their calculation (e.g., `team_size` or `commits_on_files_touched`) are calculated over the three-month time period before the pull request was opened.

The full list of features can be seen in Table 1.

3. DATASET CONSTRUCTION

The distribution of pull requests per project in Github is extremely skewed (quantiles 5%: 1, 95%: 68, mean: 26, median: 4). By the end of 2013, 255,914 projects had received a pull request while only 8,600 had received more than 100. To ensure that the selected projects used pull requests as part of the project development cycle, rather than just occasional external contributions, we only selected the top 1% of projects by total number of pull requests created. The initial selection led to 2,551 projects. In addition, to evaluate testing related features as described above, we needed a way to determine whether a source code file in a project repository represented test code. For that, we exploited the convention-based project layout in the Ruby (Gem), Python, Java and Scala (both Maven) language ecosystems, so our project selection was limited to those languages. 1,517 projects were thus filtered out.

For the remaining 1,034 repositories, the full history (including pull requests, issues and commits) of the included projects was downloaded and features were extracted by querying the GHTorrent databases and analyzing each project’s Git repository.

Merge detection. To identify pull requests that are merged outside Github, we resorted to the following heuristics, listed here in order of application:

- H_1 At least one of the commits associated with the pull request appears in the target project’s master branch.
- H_2 A commit closes the pull request (using the `fixes:` convention advocated by Github) and that commit appears in the project’s master branch. This means that the pull request commits were squashed onto one commit and this commit was merged.
- H_3 One of the last 3 (in order of appearance) discussion comments contain a commit unique identifier, this commit appears in the project’s master branch and the corresponding comment can be matched by the following regular expression:

`(?:merg|appl|pull|push|integrat)(?:ing|i?ed)`

- H_4 The latest comment prior to closing the pull request matches the regular expression above.

If none of the above heuristics identifies a merge, the pull request is identified as unmerged. The heuristics proposed above are not complete, i.e. they may not identify all merged pull requests, nor sound, i.e. they might lead to false positives (especially H_4).

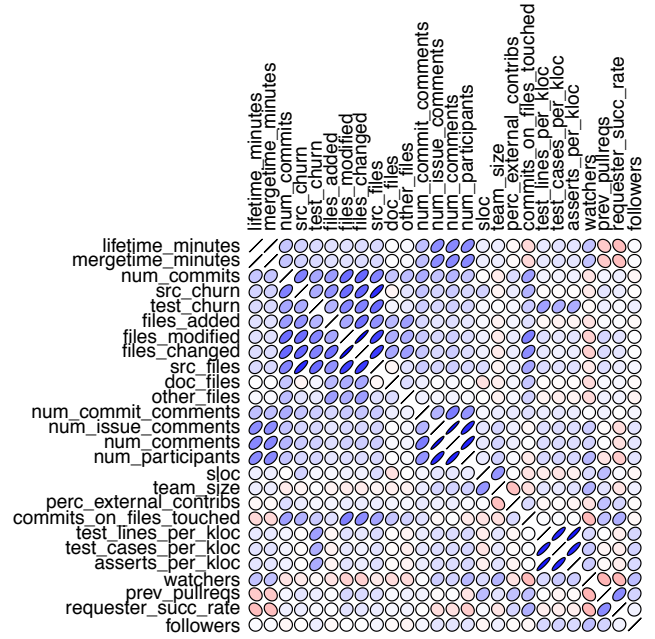





























Figure 1: Cross-correlation (Spearman) among all dataset features. Blue color (or right slant) indicates positive correlation, red color (or left slant) is negative correlation. The darker the color, the stronger the correlation.

Test case detection. In *Ruby* projects, files under the `/test/` and `/spec/` directories are considered test files. Test cases are recognized by scanning through the test files lines for method name patterns as required by the RUnit,RSpec,Shoulda and Minitest frameworks. The popular Cucumber behaviour driven development testing framework is excluded from the analysis due to inexistent naming conventions. In *Python*, project conventions do not specify specific directories for test files, so test file detection is based solely on whether the file name contains the word “test” as prefix or suffix. Test cases and asserts are discovered both in source code and also in API examples embedded in documentation comments (also known as *doctests*).

Following Maven conventions, in *Java* projects, files in directories under a `test/` branch of the file tree are considered test files. junit4 test cases are recognized using the `@Test` tag. For junit3, methods starting with test are considered as test methods. Asserts are counted by searching through the source code lines for `assert` statements. Finally, as Maven underlies *Scala*’s default build system (`sbt`), the same conventions as in the Java case can be used to discover test files. In addition to junit, the process can discover test cases and assert statements as defined by the `scalatest` and `specs2` testing frameworks.

Counting lines, files and file types. In the `pullreqs` dataset, a line of code is an executable statement, excluding blank lines and comments. To measure lines, we developed custom comment strippers for all the programming languages the dataset supports, as we could not find any tool that can count lines reliably (block comments in Ruby and Python were a particular problem). Moreover, we delegated the identification of file types to a Ruby library called `Linguist` (the same that Github uses), which supports more than 250 file types.

Table 1: Selected features and descriptive statistics. A data point is a pull request. Histograms are in log scale.

Feature	Description	5%	mean	median	95%	histogram
Pull Request Characteristics						
lifetime_minutes	Minutes between opening and closing	0.00	15,418	581.00	72,508	
mergetime_minutes	Minutes between opening and merging (only for merged pull requests)	0.00	10,506	418.00	44,234	
num_commits	Number of commits	1.00	4.42	1.00	12.00	
src_churn	Number of lines changed (added + deleted)	0.00	282.95	10.00	846.00	
test_churn	Number of test lines changed	0.00	79.74	0.00	248.00	
files_added	Number of files added	0.00	4.01	0.00	7.00	
files_deleted	Number of files deleted	0.00	2.05	0.00	1.00	
files_modified	Number of files modified	1.00	7.56	2.00	21.00	
files_changed	Number of files touched (sum of the above)	1.00	13.62	2.00	32.00	
src_files	Number of source code files touched by the pull request	0.00	7.64	1.00	20.00	
doc_files	Number of documentation (markup) files touched	0.00	2.36	0.00	6.00	
other_files	Number of non-source, non-documentation files touched	0.00	2.74	0.00	4.00	
num_commit_comments	The total number of code review comments	0.00	0.73	0.00	4.00	
num_issue_comments	The total number of discussion comments	0.00	1.84	0.00	8.00	
num_comments	The total number of comments (discussion and code review).	0.00	2.57	1.00	11.00	
num_participants	Number of participants in the discussion	0.00	1.27	1.00	4.00	
Project Characteristics						
sloc	Executable lines of code at creation time.	458.00	53,801	18,019	275,058	
team_size	Number of active core team members during the last 3 months prior to creation.	1.00	20.64	7.00	93.00	
perc_external_contribs	The ratio of commits from external members over core team members in the last 3 months prior to creation.	8.00	54.01	56.00	95.00	
commits_on_files_touched	Number of total commits on files touched by the pull request 3 months before the creation time.	0.00	51.65	4.00	209.00	
test_lines_per_kloc	Executable lines of test code per 1,000 lines of source code	0.00	1,297	355.21	2,097	
test_cases_per_kloc	Number of test cases per 1,000 lines of source code	0.00	83.74	14.55	181.03	
asserts_per_kloc	Number of assert statements per 1,000 lines of source code	0.00	200.30	40.37	479.11	
watchers	Project watchers (stars) at creation	4.00	1,778	310.00	11,114	
Developer Characteristics						
prev_pullreqs	Number of pull requests submitted by a specific developer, prior to the examined one	0.00	42.81	11.00	196.00	
requester_succ_rate	The percentage of the developer's pull requests that have been merged up to the creation of the examined one	0.00	0.51	0.62	1.00	
followers	Followers to the developer at creation	0.00	20.93	4.00	80.00	

Further Quality Control. After the dataset was constructed, the following criteria were applied to ensure homogeneity:

- The number of pull requests in the data files should be more than 70% of the pull requests in the database. The typical reason for some pull requests missing from the output is that many projects use a dedicated branch for documentation which includes no source code; the data generation script skips such pull requests. Other reasons might include the project has been deleted from Github or no source code files could be identified in a specific version. 119 projects were filtered out.
- The ratio of merged pull requests should be within reasonable limits from the mean merge ratio across all Github projects. In the GHTorrent dataset, the mean merge ratio is 72%. The pullreqs dataset uses heuristics to identify merges done outside Github, so we generally expect a mean merge ratio close to or higher of the one across Github. For this reason, we filtered out the lowest 5% of the projects. The remaining projects exhibit a merge ratio more than 50.8%. 43 projects were filtered out.

Results. The final dataset consists of 865 projects (303 Python, 253 Java, 274 Ruby, 35 Scala) and 336,502 pull requests (120,475; 83,960; 113,665 and 18,402 for Python, Ruby, Java and Scala projects respectively). 60% of the pull requests are merged using Github's facilities while 16% are identified as unmerged. The remaining 24% is identified as merged using the heuristics described above (H_1 : 11%, H_2 : 3%, H_3 : 4%, H_4 : 6%). Moreover, as Figure 1 shows, the selected features are fairly orthogonal, with very few strong correlations between them. This indicates that they capture a wide range of pull request activities with minimal overlap.

4. TOOLS

The pullreqs dataset is accompanied by an extensive analysis toolkit written in the R statistics language. The framework allows researchers to load selections of projects in R, provides basic statistics in tabular (e.g. Table 1) and graphical form (e.g. Figure 1), and exposes a command line base interface that tools can extend. Moreover, it implements a modular, multi-step data mining framework that researchers can easily re-use in similar machine learning experiments. The data mining parts include all the necessary steps

for successfully executing a data mining experiment, such as distribution plotting, cross correlation among features, pluggable model and algorithm definitions, n -fold cross-validation and plotting of important model variables such as Area Under Curve (AUC) and accuracy across cross-validation runs. To cope with the `pullreqs` dataset size, several parts of the tooling (mainly the cross validation and variable importance steps) employ parallelism and can thus exploit multi-core machines.

5. LIMITATIONS

The dataset, while extensive, is not entirely representative of all pull request activity on Github. Specifically, while it covers almost 10% of the pull requests available in the GHTorrent database, the project selection process and consequent filtering leave out important aspects (e.g. development in Javascript, the most popular language on Github). Moreover, to analyze the projects, we extracted data from i) the GHTorrent relational database ii) the GHTorrent raw database iii) each project's Git repository. Differences in the data abstraction afforded by each data source may lead to different results in the following cases: i) Number of commits in a pull request: During their lifecycle, pull requests may be updated with new commits. However, when developers use commit squashing, the number of commits is reduced to one. Therefore the number of commits feature is often an idealized version of the actual work that took place. ii) Number of files and commits on touched files: The commits reported in a pull request also contain commits that merge branches, which the developer may have merged prior to performing his changes. These commits may contain several files not related to the pull request itself, which in turn affects our results. We filtered out such commits, but this may not reflect the contents of certain pull requests.

6. RELATED WORK

The `pullreqs` dataset is similar to the code review datasets published by Hamasaki et al. [9] and Mukadam et al. [11]. While code reviewing is an inherent characteristic of pull requests, pull requests cover a wider range of project activities, such as discussion on project features, integration of attached external tools (continuous integration, code quality evaluation), management of project goals through milestones etc. Thus the `pullreqs` dataset improves upon those two datasets not only by covering a wider array of projects and languages but also by offering more precise related data, such as file counts and test cases and covering a wider range of activities. To the best of our knowledge, this is the only publicly available dataset covering pull-based development and certain aspects of distributed software development in general.

7. RESEARCH OPPORTUNITIES

The `pullreqs` dataset can be used for a multitude of studies, apart from basic exploration of the pull-based distributed development model. Making the pull request process more efficient requires research in topics such as *pull request triaging* and *patch quality evaluation*. Tools that recommend appropriate developers for doing code reviews or label pull requests according to their characteristics might be possible. The dataset can then be used as a benchmark for comparing recommender systems. *Code reviews* are a core ingredient of the pull request model. Using this dataset, a researcher can readily get quantitative data for thousands of code reviews, which can then be combined with qualitative data for specific projects. Finally, Github has been praised for lowering the entry barrier for casual contributions to projects, but do projects actually take advantage of this? *Community openness* is an important property for

projects hosted on Github; the `pullreqs` dataset can be used to evaluate aspects of it.

We presented the `pullreqs` dataset, a curated dataset of almost 900 projects, along with a statistical tool set for its analysis. The dataset itself can be extended to support more programming languages and more projects. All source code and data is available on the Github repository `gousiosg/pullreqs`. Data on the `ghtorrent.org` web site permit full replication of the construction process or the expansion of this dataset. Pull requests are especially welcome!

8. ACKNOWLEDGEMENTS

This work is supported by the NWO 639.022.314 — TestRoots project.

9. REFERENCES

- [1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of ICSE '06*, pages 361–370. ACM, 2006.
- [2] O. Baysal, R. Holmes, and M. W. Godfrey. Mining usage data and development artifacts. In *Proceedings of MSR '12*, pages 98–107. IEEE.
- [3] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in oss projects. In *Proceedings of MSR '07*, page 26. IEEE Computer Society, 2007.
- [4] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? Immigration in open source projects. In *Proceedings of MSR '07*, page 6. IEEE Computer Society, 2007.
- [5] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in Github: transparency and collaboration in an open software repository. In *Proceedings of CSCW '12*, pages 1277–1286. ACM, 2012.
- [6] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of RSSE '10*, pages 52–56. ACM, 2010.
- [7] G. Gousios. The GHTorrent dataset and tool suite. In *Proceedings of MSR '13*, pages 233–236. IEEE, 2013.
- [8] G. Gousios, M. Pinzger, and A. van Deursen. An exploration of the pull-based software development model. June 2014. To appear at ICSE 2014.
- [9] K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida. Who does what during a code review? datasets of oss peer review repositories. In *Proceedings of MSR '13*, pages 49–52. IEEE, 2013.
- [10] G. Jeong, S. Kim, T. Zimmermann, and K. Yi. Improving code review by predicting reviewers and acceptance of patches. *Research on Software Analysis for Error-free Computing Center Tech-Memo (ROSAEC MEMO)*, 2009.
- [11] M. Mukadam, C. Bird, and P. C. Rigby. Gerrit software code review data from Android. In *Proceedings of MSR '13*, pages 45–48. IEEE, 2013.
- [12] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of ICSE '05*, pages 284–292. ACM, 2005.
- [13] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of ICSE '13*, pages 112–121. IEEE, 2013.
- [14] P. C. Rigby and C. Bird. Convergent software peer review practices. In *Proceedings of FSE '13*, 2013.
- [15] P. Weißgerber, D. Neu, and S. Diehl. Small patches get in! In *Proceedings of MSR '08*, pages 67–76. ACM, 2008.