Poster:An Exploratory Study of the Pull-Based Software Development Model

Han-Ping Lin
North Carolina State University
Raleigh, North Carolina
hlin14@ncsu.edu

ABSTRACT

The advent of distributed version control systems has led to the development of a new paradigm for distributed software development; instead of pushing changes to a central repository, developers pull them from other repositories and merge them locally. In this work, we explore how pull-based software development works, first on the GHTorrent corpus and then on a carefully selected sample of 291 projects. We find that the pull request model offers fast turnaround, increased opportunities for community engagement and decreased time to incorporate contributions. We show that a relatively small number of factors affect both the decision to merge a pull request and the time to process it. We also examine the reasons for pull request rejection and find that technical ones are only a small minority.

1 INTRODUCTION

Pull-based development is an emerging paradigm for distributed software development.A unique characteristic of such sites is that they allow any user to clone any public repository.The goal of this work is to obtain a deep understanding of pull request usage and to analyze the factors that affect the efficiency of the pull-based software development model. Specifically, the questions we are trying to answer are:

RQ1: How popular is the pull based development model?

RQ2: What are the lifecycle characteristics of pull requests?

RQ3: What factors affect the decision and the time required to merge a pull request?

RQ4: Why are some pull requests not merged?

2 RESEARCH DESIGN

RQ1: We investigate such questions as how many projects actually make use of pull requests, how many of the projects are original repositories (versus, e.g., forks), and how pull requests relate to Github issue tracking facilities.

RQ2 and RQ3: Determine a set of suitable candidate features through consultation of related work in the fields of patch submission, bug triaging, code reviewing and distributed collaboration. Then, we clean it up through cross-correlation analysis to obtain a set of features with maximum predictive power.Using the data from the extracted features, we perform a detailed statistical analysis of pull request characteristics to answer RQ2 Next, we use machine learning to retrieve the dominant features. Prior to running the classification algorithms, we automatically labeled each pull request with an outcome factor; in the case of the merge decision classification task, the label signifies whether the pull request has been merged. For the merge time task, we first filter out pull requests that have not been merged and then split the remaining data points into three classes (hour, day, more than a day) according to the time required to merge the pull request.

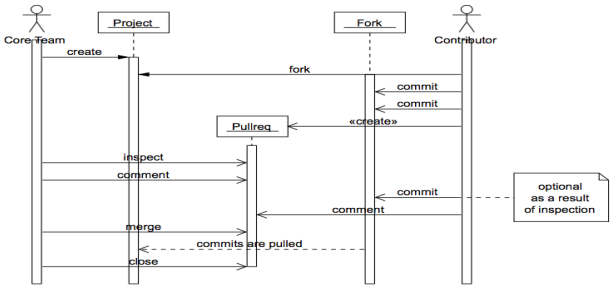

Figure 1: The pull request process.

Figure 1: A sample black and white graphic that needs to span two columns of text.

RQ4: To examine why some pull requests are not merged, we qualitatively analyze a set of randomly chosen non-merged pull requests in depth. We use open coding (a grounded theory tool) to come up with an inclusive set of reasons of why pull requests are not merged as follows: the first author read the pull request discussion on Github for randomly selected pull requests and summarized the reasons for closing them into one sentence per sample; during a second pass, the descriptions were aggregated and codes were extracted.

3 DATA

3.1 Github Data

We used Github data as provided through our GHTorrent project, an off-line mirror of the data offered through the Github API.The project has been collecting data since February 2012. Up to August 2013, 1.9 million pull requests from more than two hundred thousand projects have been collected.

3.2 Pull Request Project Sample

Project selection. To make the analysis practical, while avoiding to examine toy projects, we use a dataset consisting of all projects for which GHTorrent recorded more than 200 pull requests in the period between February 2012 and August 2013. **Feature Extraction.** The feature selection was based on prior work in the areas of patch submission and acceptance , code reviewing, bug triaging and also on semi- structured interviews of Github developers . The selected features are split into three categories:

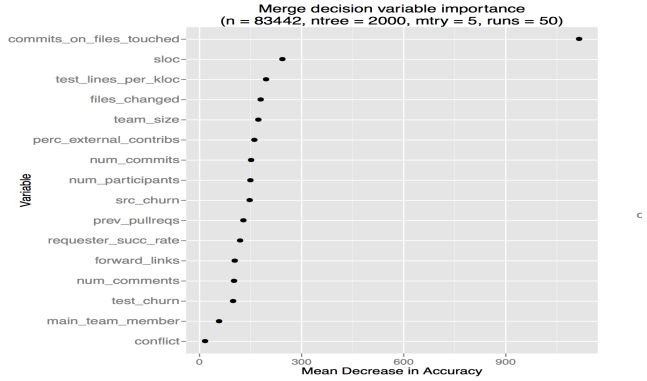

Table 3: Reasons for closing pull requests without merging.

Reason	Description	%
obsolete	The PR is no longer relevant, as the project has progressed.	4
conflict	There feature is currently being implemented by other PR or in another branch.	5
superseded	A new PR solves the problem better.	18
duplicate	The functionality had been in the project prior to the submission of the PR	2
superfluous	PR doesn't solve an existing problem or add a feature needed by the project.	6
deferred	Proposed change delayed for further investigation in the future.	8
process	The PR does not follow the correct project conventions for sending and handling pull requests.	9
tests	Tests failed to run.	1
incorrect implementation	The implementation of the feature is incorrect, missing or not following project standards.	13
merged	The PR was identified as merged by the human examiner	19
unknown	The PR could not be classified due to lacking information	15

(1)*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 .

(2)*Project characteristics.* These features quantify how receptive to pull requests the project is. If the project process is open to external contributions, then we expect to see an increased ratio of external contributors over team members.

(3)*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.

4 POPULARITY OF PULL-BASED DEVELOPMENT

In the period from February to August 2012, 315,522 original repositories received a single commit. From those, 53,866 (17%) received at least one pull request, while 54,205 (18%) used the shared repository approach, having received commits by more than one developers and no pull requests. The situation is similar during the same period in 2013; from the 1,157,625 repositories that re- ceived a single commit, 120,104 (10%) repositories received a pull request while 124,316 (11%) used the shared repository approach exclusively. In both cases, the remaining 65% and 79% are single developer projects. Across both years, 14% of the active repositories use pull requests.

5 PULL REQUEST LIFECYCLE

For merged pull requests, an important property is the time required to process and merge them. The time to merge distribution is highly skewed, with the great majority of merges happening very fast. Measured in days, 95% of the pull requests are merged in 26, 90% in 10 and 80% in 3.7 days.This means that pull requests are either processed fast or left lingering for long before they are closed.

Sizes of pull requests. A pull request bundles together a set of commits; the number of commits on a pull request is generally less than 10 (95% percentile: 12, 90% percentile: 6, 80% percentile: 3), with a median of 1. The number of files that are changed by a pull request is generally less than 20 (95% percentile: 36, 90% percentile: 17, 80% percentile: 7), with median number of 2. The number of total lines changed by pull requests is on average less than 500 (95% percentile: 1227, 90% percentile: 497, 80% percentile: 168) with a median number of 20.

6 MERGING AND MERGE TIME

For the merge decision task, the feature importance result is dominated by the commits_on_files_touched feature. By re-running the cross validation process, we conclude that it suffices to use the features commits_on_files_touched, sloc and files_changed to predict whether a pull request will be merged. Therefore, we can conclude that the decision to merge a pull request is affected by whether it touches an actively developed part of the system, how large the project source code base is and how many files the pull request changes.

For the mergetime task, there is no dominant feature. The time to merge is influenced by the developer previous track record, the size of the project and its test coverage and the project openness to external contributions.

7 UNMERGED PULL REQUESTS

53% of pull requests are rejected for reasons having to do with the distributed nature of pull based development. Only 13% of the pull requests are rejected due to technical reasons.

8 CONCLUSION

1. The pull-based model is not as popular as we had anticipated: Only 14% of the active projects use pull requests, but this number is equal to the number of projects using the shared repository approach (Section 5). 2. Most pull requests affect just a few dozen lines of code, and 60% are processed (merged or discarded) in less than a day. The merge decision is not affected by the presence of test code. Core members and external developers have equal chances to get their pull request accepted (Section 6). 3. The decision to merge is mainly affected by whether the pull request modifies recently modified code. The time to merge is influenced by various factors, including the developerâĂŹs track record, and the projectâĂŹs test coverage. 4. 53% of non-merged pull requests are rejected for reasons related to the distributed nature of pull-based development. Only 13 of the pull requests are rejected due to technical reasons.

9 RERFERENCE

[1] Georgios Gousios,Martin Pinzger,Arie van Deursen.An Exploratory Study of the Pull-Based Software Development Model