# Effect of Follow and Watch Relationships in Pull Requests

Di Yang
Mondego Lab
Department of Informatics,
Bren School of ICS
University of California, Irvine
diy4@uci.edu

Aftab Hussain
Mondego Lab
Department of Informatics,
Bren School of ICS
University of California, Irvine
aftabh@uci.edu

Cristina Videira Lopes
Mondego Lab
Department of Informatics,
Bren School of ICS
University of California, Irvine
lopes@ics.uci.edu

## ABSTRACT

GitHub is both a code repository system and a social networking site. The social graph is built by "following a user" or "watching a project." Making contributions to other users' projects involves sending a pull request by the person committing the code, and the consequent acceptance of the pull request (i.e. merging the committed code) by the project owner at some later time. The goal of our study is to investigate whether the relationships built by following users and watching projects influence the response time of pull requests. From the given data set of 90 projects from GitHub, we retrieve the response times of all the pull requests, and identify those pull requests made by followers and watchers. We perform the Kruskal Wallis test to compare each pair of these user groups in order to assess any significant differences between the mean response times of these groups.

Our findings are twofold. First, we found that very few pull requests are made by the followers of the project owner or watchers of the project, that is, the social network does not reflect strongly the actual pull requests. Second, we did not find statistically significant differences on the difference of means among these different groups.

## General Terms

Follower, Watcher, Response time, Kruskal Wallis Test

## 1. INTRODUCTION

GitHub is both a web-based hosting service for software development projects and a social network for developers. One of the interesting features in GitHub is the ability to see what other people are working on and who they are connecting with. When developers "follow a user," they will get notifications on their dashboard about that user's GitHub activity. Developers can also stay up-to-date with a specific project by "watching a project."

Previous work ([1], [2]) observed the importance of social network relations on work performance. In this light, we hypothesized that there might be a correlation between the social graphs and the actions that developers perform on their projects. Specifically, we investigated whether *follow* and *watch* relations affect the amount of pull requests and the response time of those pull requests. That is, if the committer of the pull request is a follower of the owner or a watcher to the project, will that influence the response time for merging their pull requests? If so, will it decrease the response time? If this were to be the case, project planners and recruiters could look for such relations and create teams that contain these relations.

This paper is organized as follows: in Section 2, we explain in detail about how we get the response time and the follower/watcher relationships from the given data sets. In Section 3, we present our experimental results and analysis. Finally in Section 4, we conclude our report and present other research problems in the area that could be explored in the future.

## 2. RESEARCH METHODOLOGY

We used the MYSQL dump provided for this Data Challenge consisting of data for 90 GitHub projects. After carefully analyzing the given data set and the schema, we find four key tables that are related to the question, whose columns are shown in Table 1. In the following subsections we shall demonstrate how we obtained each of these relational database tables.

**Table 1: Tables used from MySQL dump**

| Database Table | Fields |
|---|---|
| `pull_request_history` | *id, pull_request_id, created_at, ext_ref_id, action, actor_id* |
| `pull_request_commits` | *pull_request_id, commits_id* |
| `commits` | *id, sha, author_id, committer_id, project_id, created_at, ext_ref_id* |
| `followers` | *user_id, follower_id, created_at, ext_ref_id* |
| `watchers` | *repo_id, user_id, created_at, ext_ref_id* |

### 2.1 Building the response_time table

In order to untangle the relationship between social network and the response time of a pull request, we need to get the response time first. The table *pull_request_history* provides us with the information. There are three values in

```
create table response_time(
pull_request_id int,
response_time int,
actor_id int
);
insert into response_time (pull_request_id, response_time, actor_id)
select distinct p1.pull_request_id, timestampdiff(second, p1.created_at, p2.created_at), p1.actor_id
from pull_request_history as p1, pull_request_history as p2
where p1.pull_request_id = p2.pull_request_id and p1.action = 'opened' and p2.action = 'merged';
```

**Figure 1: SQL Query for retrieving the response_time table for all users.**

```
#followers
select author_id, committer_id, project_id, prc.pull_request_id, count(commit_id), response_time
from commits as c join followers as f join pull_request_commits as prc join response_time as rt
on c.committer_id=f.follower_id and c.author_id=f.user_id and c.id=prc.commit_id
    and rt.pull_request_id=prc.pull_request_id and author_id<>committer_id
GROUP BY prc.pull_request_id;
```

**Figure 2: SQL Query for retrieving the response_time table for followers.**

```
#watchers
select author_id, committer_id, project_id, prc.pull_request_id, count(commit_id), response_time
from commits as c join watchers as w join pull_request_commits as prc join response_time as rt
on c.project_id=w.repo_id and c.committer_id=w.user_id and c.id=prc.commit_id
    and prc.pull_request_id=rt.pull_request_id and author_id<>committer_id
GROUP BY prc.pull_request_id;
```

**Figure 3: SQL Query for retrieving the response_time table for watchers.**

the *action* column of the table: *opened, merged, and closed*, which have their created time respectively in the *created_at* column. In GitHub, a pull request can have two life cycles: created, merged, then closed, or created and closed. In our study, we only consider the pull requests that have been approved, aka merged. So we calculate the time difference between *opened* and *merged*. Then the *response_time* table can be created by the SQL query as given in Figure 1.

## 2.2 Connect response_time with followers

After obtaining the response time, we need to find out whether the committer of the pull request is a follower of the owner of the project. From Table 1, we can observe that the commits table contains the information of the author (i.e owner) and the committer. Also, the *pull_request_commits* table connects the *pull_request* table and the commits table with *pull_request_id* and commit_id. Followers' information is stored in the followers table, which contains the relation between user (i.e owner) and followers. From all the information we have above, we can identify if the committer of the pull request is the follower of the owner of the project, and what the response time is. This is achieved using the query given in Figure 2. One pull request can have multiple commits, so there can be many duplicates. To eliminate the duplicates, we group the commits by pull request.

## 2.3 Connect response_time with watchers

The difference between followers and watchers is that followers follow another user's activities, while watchers watch a specific project. The *repo_id* in the *watchers* table is the *project_id* for a user to watch. With the *committer_id* and *project_id* present in the commits table, we can again connect the pull request response time with the watcher' information, using the query shown in Figure 3.

## 3. RESULTS AND ANALYSIS

In this section, we present the results of our study and elaborate upon the analysis techniques that have been used.

Table 3 shows the number of approved pull requests for each user group. As shown in the table, the total number of approved pull requests we observed was 34,126. Figures 4 and 5 show the response times for the pull requests for the three user groups. These graphs clearly suggest that the response times are not normally distributed.

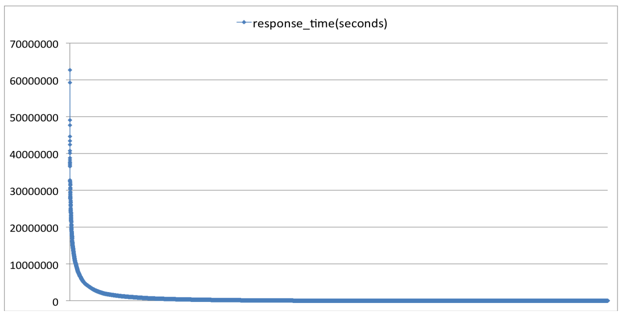

**Figure 4: Response time series for all user groups.**

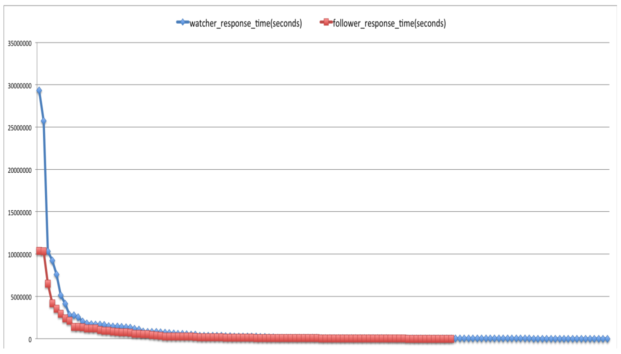

**Figure 5: Response time series for followers and watchers.**

## 3.1 Merged Pull Requests

**Table 2: Number of approved pull requests by type of parent requester**

| Pull Requester Type | No. of Pull Requests |
|---|---|
| `All users` | 34,125 |
| `Followers` | 94 |
| `Watchers` | 132 |

**Table 3: Mean response times of pull requests**

| Pull Requester Type | Mean Response Time (s) |
|---|---|
| `All users` | 608,476 |
| `Followers` | 473,653 |
| `Watchers` | 1,037,830 |

The first interesting, and somewhat surprising, finding of our study is that out of 34,125 merged pull requests, only 0.28 % are from followers of the projects' owners, and only 0.39 % are from watchers of the projects. This can either suggest that followers seldom perform pull requests to projects of people whom they are following (and watchers seldom perform pull requests towards the projects they are watching), or it may suggest project owners seldom merge pull requests coming from their followers or from their projects' watchers.

## 3.2 Mean Response Times

Table 3 shows the mean response times for pull requests for the three different groups of interest. As seen, the mean response times for followers' requests is around 22.2% smaller than that for all requests. The mean response times for watchers' requests is around 70.56% larger than that for all requests. Considering the fact that the variance of these data points is high, there is the need to assess whether these differences in means are statistically significant. This is explained in the following subsections.

## 3.3 Impact of Follower Relationships

In order to ascertain whether there is any significant difference between the response times for followers' requests and the response times for all requests, we performed the Kruskal Wallis test [3] on the two samples. In particular we tested whether the samples have identical distribution functions. Thus, our null hypothesis was that the two samples are identical. We obtained a $p$-value of 0.4805 for the test. Controlling for a 95% type I error, we did not reject the null hypothesis, and thus concluded that the samples' response times are not significantly different.

## 3.4 Impact of Watcher Relationships

In order to ascertain whether there is any significant difference between the response times for watchers' requests and the response times for all requests, we also performed the Kruskal Wallis test on the two samples in the same manner as above. Our null hypothesis was that the two samples are identical. We obtained a $p$-value of 0.4655 for the test. Controlling for a 95% type I error, we did not reject the null hypothesis, and thus concluded that the samples' response times are not significantly different.

## 3.5 Differences Between Follower and Watcher Relationships

In order to ascertain whether there is any significant difference between the response times for watchers' requests and the response times for followers' requests, we again performed the Kruskal Wallis test on the two samples.Our null hypothesis was that the two samples are identical. We obtained a $p$-value of 0.4805 for the test. Controlling for a 95% type I error, we did not reject the null hypothesis, and thus concluded that the samples' response times are not significantly different.

## 3.6 Outliers

We have identified and discarded two outliers from our sample of followers, which had extremely high response time values; both were in the order of 10 million seconds, which were much larger than both the maximum value from the remaining sample set (6 million s), and the mean of the remaining sample set (473653 s).

## 4. CONCLUSION AND DISCUSSION

The purpose of our study was to investigate whether the committer being a follower of the owner of a project or being a watcher of a project affects the owner's response time to merging the pull requests. After examining the detailed meanings of each table, we worked out the connections between five key tables and built the response time tables for all pull requests: the pull requests by followers, and the ones by watchers. As we can see from the analysis results, pull requests from followers have a smaller response time, but the differences are not statistically significant. Another finding is that very few people who are followers of project owners actually send pull requests. Therefore, the social network does not strongly reflect the actual behavior of the pull requests and there is a gap between the social networking aspect and code sharing aspect of GitHub.

In future, it would be interesting to investigate how users react to pull requests that come from those whom they follow. Intuitively, it would be reasonable to assume that people generally follow those users in GitHub whose codes interest them. Therefore, it is likely that pull requests coming from those whom a user follows may be approved sooner by the user. An analysis to prove or disprove this assumption would be of interest. In regard to response time there may be other factors that may have an effect. For example the owner's own activity habit. Some owners may be very active and check their accounts every day, others may seldom visit GitHub, which may have significant influence to their response time to pull requests.

## 5. ACKNOWLEDGMENTS

We would like to thank Thomas Debeauvais, Vaibhav Pratap Singh Saini, and Nicolas Lopez-Giraldo for their valuable advice on statistical analysis techniques.

## 6. REFERENCES

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286, 2012.

[2] J. DiMicco, D. Millen, W. Geyer, and C. Dugan. Motivations for social networking at work. In *In Proc CSCW 2008*, pages 711–720, 2008.

[3] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.