“Understanding Watchers on GitHub”

CIS 5930 - Data Mining in Software Engineering
Project Report

Project Members:
Harish Mandava (hm15h)
Vigneshwar Padmanaban (vp15g)

GitHub:
Repository Link: https://github.com/swift2891/cis5930_github_analytics

1. Introduction
The “watchers” is a new kind of passive project membership for GitHub projects. To understand who these watchers are and how they contribute to the project can be vital information for the efficient management and survival of a project in GitHub. To understand these watchers the author of the paper broke down the task into following three research question.

a. (RQ1) Do Watchers become Contributors?
b. (RQ2) Do Watchers behave differently than normal contributors?
c. (RQ3) Does programming language have any impact on who and when contribute to a project?

2. Methodology
2.1 Overview:

Data : GitHub MSR 2014 dataset (MySQL)
Repositories : 91 repositories
IDE : Eclipse Oxygen
Programming Language : Java
Other Tools : MySQL Workbench

2.2 Approach:

For most of the tasks we followed this below approach,

a. Design the SQL solution
b. Write SQL Query
c. Test the SQL query - Manually checking related Tables
d. Embed SQL query in Java code

e. Process data and display wherever necessary

2.3 Research Question 1:

Task 1: Pearson Correlation - Watchers Vs Forks/ Issues/ Commits

Step 1: Get watchers count.

SQL

```
select count(repo_id) as watchers_count\r\n" +
                            "from (SELECT id FROM github.projects where
ISNULL(forked_from)   =   1)   as   t1   inner   join   watchers   on
watchers.repo_id=t1.id \r\n" +
                            "group by watchers.repo_id
```

Step 2: Get Forks Count

SQL

```
SELECT t1.id, count(*) as forks_count from \r\n" +
                            "(\r\n" +
                            "select   t2.id   as   id,count(repo_id)   as
watchers_count\r\n" +
                            "from (SELECT id FROM github.projects where
ISNULL(forked_from)   =   1)   as   t2   inner   join   watchers   on
watchers.repo_id=t2.id \r\n" +
                            "group by watchers.repo_id\r\n" +
                            ") as t1 inner join projects on t1.id =
projects.forked_from\r\n" +
                            "group by projects.forked_from
```

Step 3: Get Issues Count

SQL

```
SELECT count(*) as issues_count from \r\n" +
                            "(\r\n" +
                            "select   t2.id   as   id,count(repo_id)   as
watchers_count\r\n" +
                            "from (SELECT id FROM github.projects where
ISNULL(forked_from)   =   1)   as   t2   inner   join   watchers   on
watchers.repo_id=t2.id \r\n" +
                            "group by watchers.repo_id\r\n" +
                            ") as t1 inner join issues on t1.id =
issues.repo_id\r\n" +
                            "group by issues.repo_id
```

Step 4: Get Commits Count
SQL

```
SELECT count(*) as commits_count from \r\n" +
                            "(\r\n" +
                            "select  t2.id  as  id,count(repo_id)  as
watchers_count\r\n" +
                            "from (SELECT id FROM github.projects where
ISNULL(forked_from)  =  1)  as  t2  inner  join  watchers  on
watchers.repo_id=t2.id \r\n" +
                            "group by watchers.repo_id\r\n" +
                            ") as t1 inner join commits on t1.id =
commits.project_id\r\n" +
                            "group by commits.project_id
```

Step 5: Using the org.apache.commons.math3.stat.correlation.PearsonsCorrelation class compute Pearson correlation for the data series calculated in above steps.

Task 2: Compute the percentage of watchers who become contributors eventually and Compute the percentage of contributors who are watchers turned contributors.

Step 1: Start gathering all the user contributions for the 72 repositories (which has watcher information). There are 7 types of contributions (commits, issues, pull requests, commit comments, issue comments, pull request comments, issue assignee)

Step 2: For each contribution type design sql solution and write query. For commits type
SQL

```
insert into prj_contr_ts_typ;
select project_id, author_id, min(created_at), "commits" as contr_type
from commits
inner join projects_needed
on commits.project_id = projects_needed.id
group by project_id, author_id
```

Step 3: for issue reported
SQL

```
insert into prj_contr_ts_typ;
SELECT  repo_id,  reporter_id,  min(created_at),  "issues"  FROM  issues
inner join projects_needed
on issues.repo_id = projects_needed.id
where isnull(reporter_id)=0
group by repo_id, reporter_id
```

Step 4: for pull requests

SQL

```
insert into prj_contr_ts_typ;
select t4.project_id, actor_id, min(created_at), "pull_requests" from
pull_request_history
inner join
(select pull_requests.id, projects_needed.id as project_id from
pull_requests inner join projects_needed
on pull_requests.base_repo_id = projects_needed.id) as t4
on pull_request_history.pull_request_id = t4.id
where action="opened"
group by t4.project_id, actor_id
```

Step 5: for commit comments

SQL

```
insert into prj_contr_ts_typ;
select t4.project_id, user_id, min(created_at), "commit_comments" from
commit_comments
inner join
(select commits.id, commits.project_id from commits inner join
projects_needed
on commits.project_id = projects_needed.id) as t4
on commit_comments.commit_id = t4.id
group by t4.project_id, user_id
```

Step 6: for issue comments

SQL

```
insert into prj_contr_ts_typ;
select t4.repo_id,user_id, min(created_at), "issue_comments" from
issue_comments
inner join
(select issues.id, issues.repo_id from issues inner join
projects_needed
on issues.repo_id = projects_needed.id) as t4
on issue_comments.issue_id = t4.id
group by t4.repo_id,user_id
```

Step 7: for pull request comments
SQL

```
insert into prj_contr_ts_typ;
select                                          t4.base_repo_id,user_id,
min(created_at),"pull_request_comments" from pull_request_comments
inner join
(select     pull_requests.id,     pull_requests.base_repo_id     from
pull_requests inner join projects_needed
on pull_requests.base_repo_id = projects_needed.id) as t4
on pull_request_comments.pull_request_id = t4.id
group by t4.base_repo_id,user_id
```

Step 8: for assignee type
SQL

```
insert into prj_contr_ts_typ
SELECT issues.repo_id, assignee_id, min(created_at), "assignee" FROM
issues inner join projects_needed
on issues.repo_id = projects_needed.id
where isnull(assignee_id)=0
group by issues.repo_id, assignee_id
```

Step 9: Calculate Watchers turned Contributors
SQL

```
select count(*) as watch_contr from watchers inner join\r\n" +
                    "(\r\n" +
                    "SELECT project_id, author_id, min(created_at) as
created_at FROM github.prj_contr_ts_typ2\r\n" +
                    "group by project_id, author_id\r\n" +
                    ") as t4\r\n" +
                    "on watchers.user_id = t4.author_id and\r\n" +
                    "watchers.created_at < t4.created_at and \r\n" +
                    "watchers.repo_id = t4.project_id
```

Step 10: Compute the percentages for both cases (a. Percentage of Watchers turned Contributors b. Percentage of contributors who are watchers before)

Task 3: Calculate Mean Time Before First Contribution for each category

Step 1: Using the table computed from before task, containing all the contributions of authors for each project, it will be joined with watcher table containing watcher and timestamp.

SQL

```
select avg(time_diff)/86400 as avg_mtbf from(\r\n" +
                            "SELECT                              author_id,
time_to_sec(timediff(t4.created_at,watchers.created_at))  as  time_diff,
t4.contr_type FROM \r\n" +
                            "(SELECT       project_id,       author_id,
min(created_at)       as       created_at,       contr_type       FROM
github.prj_contr_ts_typ2\r\n" +
                            "group by project_id, author_id) as t4\r\n"
+
                            "inner join watchers\r\n" +
                            "on  watchers.repo_id  =  t4.project_id  and
\r\n" +
                            "watchers.user_id  =  t4.author_id  and  \r\n"
+
                            "watchers.created_at  <  t4.created_at  \r\n"
+
                            ") as t7
```

Step 2: for commits type

SQL

```
select avg(time_diff)/86400 as avg_mtbf from(\r\n" +
                            "SELECT                              author_id,
time_to_sec(timediff(t4.created_at,watchers.created_at))  as  time_diff,
t4.contr_type FROM \r\n" +
                            "(SELECT       project_id,       author_id,
min(created_at)       as       created_at,       contr_type       FROM
github.prj_contr_ts_typ2\r\n" +
                            "group by project_id, author_id) as t4\r\n"
+
                            "inner join watchers\r\n" +
                            "on  watchers.repo_id  =  t4.project_id  and
\r\n" +
                            "watchers.user_id  =  t4.author_id  and  \r\n"
+
                            "watchers.created_at  <  t4.created_at  and
\r\n" +
                            "t4.contr_type = \"commits\"\r\n" +
                            ") as t7
```

Step 3: Similarly, for rest of contribution types the query will be executed and results are recorded.

2.4 Research Question 2:

The contributions like commits, pull requests and issues reporting are considered as confident actions compared to the rest.

Task 1: For watchers turned contributors compute the confidence for each major 6 types

Step 1: Using the table previously constructed(Table containing all the contributions), we can calculate the count of authors who did a particular type of contribution

SQL

```
select count(*) as commit_conf from watchers inner join\r\n" +
                            "(\r\n" +
                            "SELECT       project_id,       author_id,
min(created_at)     as     created_at,     contr_type          FROM
github.prj_contr_ts_typ2\r\n" +
                            "group by project_id, author_id\r\n" +
                            ") as t4\r\n" +
                            "on    watchers.user_id    =    t4.author_id
and\r\n" +
                            "watchers.created_at  <  t4.created_at  and
\r\n" +
                            "watchers.repo_id = t4.project_id  and\r\n"
+
                            "t4.contr_type=\"commits\";
```

Step 2: Similarly we can execute a sql query for rest of the contribution types.

Step 3: To get a total of watchers confidence all the counts from the first three contribution types. (commits, issues, pull_requests)

Task 2: For Other Contributors, compute the confidence of 6 major types.

Step 1: Build a new table containing only the “Other contributors”. Then we can get the stats for each of the contribution types.

SQL

```
select count(*) as commit_conf from \r\n" +
                            "(\r\n" +
                            "select     t4.project_id,     t4.author_id,
t4.contr_type, t4.created_at,count(*) from \r\n" +
                            "(\r\n" +
                            "select  repo_id  as  project_id,  author_id,
created_at, contr_type from watch_contr\r\n" +
                            "union all\r\n" +
```

```
                            "select                project_id,author_id,
min(created_at) as created_at, contr_type from prj_contr_ts_typ2\r\n"
+
                            "group by project_id, author_id\r\n" +
                            ") as t4\r\n" +
                            "group by project_id, author_id\r\n" +
                            "having count(*)=1\r\n" +
                            ") as t6 \r\n" +
                            "where t6.contr_type = \"commits\"
```

Step 2: Compute the stats for total confidence for “Other contributors”.

2.5 Research Question 3:

Task 1: For the top 13 programming language find the percentage of first contribution for the top five contribution types.

Step 1: Construct a new table (using the existing table of contributions) containing the needed column “Language” for each project.

Step 2: Compute the total rows for each language.

SQL

```
select count(*) as row_count from prj_contr_ts_typ_lan\r\n" +
                            "where lang = \"C\"
```

Step 3: Similarly find the count for remaining languages.

Step 4: After getting the total count for each language, compute the count for each contribution type.

SQL

```
SELECT count(*) as row_count FROM prj_contr_ts_typ_lan \r\n" +
                            "where lang = \"C\" and \r\n" +
                            "contr_type = \"issues\"
```

Step 5: Similarly compute the count for remaining contribution types.

Task 2: For the top 13 programming language find the mean time before first contribution duration.

Step 1: Using the table constructed the the previous task (containing all contributions with language information), we can join the table with watcher table to get the duration of contributor wait time. We can compute the average for each programming language.

SQL

```
select avg(time_diff)/86400 as avg_time from\r\n" +
                            "(\r\n" +
```

```
                            "SELECT                          t4.author_id,
time_to_sec(timediff(t4.created_at,watchers.created_at))  as  time_diff,
t4.contr_type \r\n" +
                            "FROM \r\n" +
                            "(\r\n" +
                            "SELECT  project_id,  author_id,  created_at,
contr_type FROM github.prj_contr_ts_typ_lan\r\n" +
                            "where lang = \"C\"\r\n" +
                            ") as t4\r\n" +
                            "inner join watchers\r\n" +
                            "on  watchers.repo_id  =  t4.project_id  and
\r\n" +
                            "watchers.user_id  =  t4.author_id  and  \r\n"
+
                            "watchers.created_at  <  t4.created_at  \r\n"
+
                            ") as t7
```

Step 2: We can do the same for rest of the programming languages.

3. Results

Note: The dataset used was not the same as the paper's version. It was updated as mentioned by the owner in the link. Probably this influenced our results to certain amount.

3.1 Research Question 1:

<table>
<tr><th>#</th><th>Tasks</th><th>Our Results</th><th>Paper Results</th></tr>
<tr><td>1</td><td>Total no. of Watchers</td><td>141300</td><td>141300</td></tr>
<tr><td>2</td><td>Total no. of Contributors</td><td>44280</td><td>55265</td></tr>
<tr><td>3</td><td>Pearson correlation</td><td><table><tr><td>Stats</td><td>Correlation</td></tr><tr><td>#Forks</td><td>0.81</td></tr><tr><td>#Issues</td><td>0.47</td></tr><tr><td>#Commits</td><td>0.04</td></tr></table></td><td><table><tr><td>Stats</td><td>Correlation</td></tr><tr><td>#Forks</td><td>0.81</td></tr><tr><td>#Issues</td><td>0.47</td></tr><tr><td>#Commits</td><td>0.07</td></tr></table></td></tr>
<tr><td>4</td><td>Percentage of Watchers turned Contributors</td><td>12.2%</td><td>4.7%</td></tr>
<tr><td>5</td><td>Percentage of Contributors who are Watchers before</td><td>38.9%</td><td>20.7%</td></tr>
</table>

6	Mean Time Before First Contribution (weeks)	Types	Duration		Tasks	Duration
		Commit Comment	28.5		Commit Comment	36.7
		Assigned Issue	29.1		Assigned Issue	37.3
		Submit Code	31.08		Submit Code	51.8
		Report Issue	30.9		Report Issue	56.1
		Create PR	30.8		Create PR	61.4
		Comment PR	32.5		Comment PR	63.1
		Issue Comment	33		Issue Comment	91.1

3.2 Research Question 2:

#	Tasks	Our Results			Paper Results		
1	Percentage of Watchers turned Contributors doing Confident actions	72.9%			87.6		
2	Percentage of Other Contributors doing Confident actions	53.2%			56.2%		
3	Percentage of First Contribution for each Type (Both Watchers and Others) %	Type	Watcher	Others	Type	Watcher	Others
		Issue report	28.8	20	Issue report	43.9	24.5
		Submit code	26.8	29.1	Submit code	26.3	27.5
		Pull Request	17.2	4.1	Pull Request	17.4	4.2
		Comment Issue		15.3	Comment Issue	2.1	39.4
		Comment commit	4.9	3.9	Comment commit	8.7	4.1
		Comment pull request	.15	.06	Comment pull request	1.2	.2

3.3 Research Question 3:

Task 1:

Language	Report Issue		Submit code		Pull request		Commit comment		Issue comment	
	Our Results	Paper's	Our Results	Paper's	Our Results	Paper's	Our Results	Paper's	Our Results	Paper's
C	29.4	26.1	40.19	27.5	9.23	5.9	4.99	.34	16	36.4
C#	8.6	9.6	63.6	46.1	16	17.4	5.6	2.2	5.9	24.3
C++	18.	17	42.44	31.9	4.9	4.1	17.9	18	16.5	28.5
CSS	41.3	65.5	2.71	4.5	2.3	3.1	0.02	.2	53.4	26.4
Go	18.7	18.5	49.2	53.5	10.1	8.8	3	3.3	18.2	15.1
Java	27.6	26.2	32	23.1	8.6	6.2	8.1	8.5	23.4	35.4
JavaScript	49.6	45.1	18.9	16.1	5.5	3.5	1.71	2.22	24	32.7
PHP	9.2	8.6	61.8	39.9	14.8	12	5.1	3.7	8.67	34.3
Python	20.3	23.2	48.9	35.1	12.5	11.1	3.9	3.1	14	26.7
R	53.2	42.4	26.49	15.8	6.9	2.6	0.7	.5	12.6	38.5
Ruby	35.6	42.3	23.8	19.3	7.2	5.1	2.44	2.4	30.7	30.2
Scala	26.2	22	50.1	31.8	7.2	4	3.2	2	12.2	38.2
TypeScript	30.5	20.1	33.5	29.7	10.7	3.1	2.54	.4	22.3	42.4

Task 2:

Language	Mean Time Before First Contribution (Our Results)	Mean Time Before First Contribution (Paper Results)
C	32.63	67.1
C#	30.6	54.1
C++	28.3	39
CSS	32.4	36.8
Go	30.9	39
Java	31	47.5
JavaScript	29.1	44.8
PHP	30.2	50
Python	31	52.4
R	31..8	42.9
Ruby	33.8	74.7
Scala	32.3	74.4
TypeScript	27.4	38.5

4. Conclusion

There are certain results which didn't exactly match with the paper's version. This we believe could be attributed by the changed version of GitHub dataset. While implementing the paper's results, there were lot of unanswered/ambiguous situations, where we had to make assumptions or best probable guess. Even this could have some factor in changed outcomes.

Despite the few discrepancies, our results do infer the same conclusion as did the paper's result. The answers for the three research questions are shown below:

a. The watchers are indeed become contributors. Even though very few of them eventually contribute (12.2%), the proportion of contributors with watchers turned contributors is large (38.9%).
b. The watchers are more confident than the other contributors who were not watchers before. As first group did confident actions more (72.9%) than the latter (53.2%) as first type of contribution.
c. The programming language has different characteristics based on the type of contributions as shown in the Results.

5. Work Distribution

(Tasks# – Based on Results Table for each RQ)

(HM – Harish; VP - Vignesh)

#	Task	Developer Responsible
1	RQ1 – Task 1, 2, 3	VP
2	RQ1 – Task 4, 5	HM
3	RQ1 – Task 6	HM, VP
4	RQ2 – Task 1	VP
5	RQ2 – Task 2	HM
6	RQ2 – Task 3	VP, HM
7	RQ3 – Task 1	HM
8	RQ3 – Task 2	VP
9	Report	VP
10	Readme	HM