GitHub Bad Smells Observation Report

Nishtha Garg
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
ngarg@ncsu.edu

Priysha Pradhan
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
ppradha3@ncsu.edu

Shivam Gulati
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
sgulati2@ncsu.edu

Shifali Jain
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
sjain7@ncsu.edu

ABSTRACT
A bad smell in code is a hint that something has gone wrong somewhere in your code. Using the smell to track down the problem is a good practice in the field of software engineering. One can also identify some early warning which will help the team of developers to correct their process in the starting stages of the project development. The aim of our project was to extract the data of GitHub repositories and analyze it to find the possible bad smells. We collected data in a SQL database using a python script. We defined certain feature detectors and certain queries were run on the data to find the possible bad smells and to analyze the same.

1. DATA COLLECTION

1.1 What we collected
We selected three GitHub repositories, out of which one was our own. For each of team a substantial data was collected and consists of the following:

Issue:
- Issue ID
- Issue Number
- Label assigned to issue
- Issue creation time
- The user, issue is assigned to
- Comments
- Milestone assigned
- Duration of the issue
- Issue closed date
- Actions on the issue
- Issue created by

Commits:
- Commit ID
- The user who committed
- Commit timestamp
- Commit message

Comments:
- Comment ID
- User who commented
- Message
- Created at
- Updated at

Milestones:
- Milestone ID
- Title
- Description of milestone
- Milestone created at
- Milestone due at
- Milestone closed at
- User who created the milestone

1.2 Collection method
We took the gitable.py [1] script and modified [2] it to extract the desired data from GitHub as per our requirements. To have a well-structured data, the script output was directly stored into text files which were mapped to CSV format. To do the analysis on the data and query using SQL, we set up an Oracle environment and imported the CSV files to create the tables for issues, milestones, commits and comments.

2. ANONYMIZATION
To anonymize the data for groups and repository team members, we created a mapping list locally. We then wrote a PL/SQL script to map the repository team and the users as per our list and ran it on all the tables we had created. The PL/SQL script was a set of instructions which assigns users as user1, user2, etc. per team for all the distinct members. A short sample to update the commits table has been mentioned below

```
BEGIN
        UPDATE commits
                set creator='user1' where creator='xyz' and
        team='team1';
        …..
END;
```

3. DATA
Combining the data [3] from the three project repositories, we grabbed the below numbers.

Milestones	Issues	Comments	Commits
32	190	203	334

Table 1: Summary of data collected

3.1 Data Sample

Sample row from ISSUES table:

TEAM	1
ISSUENO	ISSUE 33
LABEL	Null Label
WHEN	1456779702
CREATED_AT	1456775599
ASSIGNEE	user4
COMMENTS	1
MILESTONE	42430
DURATION	4103
ISSUECLOSEDAT	1456779702
ACTION	Closed
CREATOR	user4

Table 2a: Sample data from issues table

Sample row from COMMITS table:

TEAM	1
CREATOR	user4
TIME	1454379150
MESSAGE	Corrected references

Table 2a: Sample data from commits table

Sample row from MILESTONES table:

TEAM	1
ID	1
M_TITLE	Feb 1 submission
DESCRIPTION	Identify problem and analysis
CREATED_AT	1453158230
DUE_AT	1454302800
CLOSED_AT	1454614905
CREATOR	user4

Table 2c: Sample data from milestones table

Sample row from COMMENTS table:

TEAM	1
CREATOR	user4
TEXT	Completed
CREATED_AT	1454321099
UPDATED_AT	1454321099

Table 2a: Sample data from comments table

4. FEATURE EXTRACTORS

We came up with the below feature detectors form the data we collected:

4.1 Trying to push to the next sprint!

We wanted to check the issues of all the three which had been opened for too long than the threshold. This is an indication that either the issue is tough or more often that the work is being kept on hold and delayed.

4.2 Individual or a team effort?

We wanted to check if a lot of issues are being assigned to specific users or the team is actually working as a team and not relying on specific users to resolve issues.

4.3 Work without deadline, not good!

When there is no due date, the work is often neglected. For the precise reason, we looked into the issues which were created but not assigned any milestone for completion.

4.4 Did you tell after you already worked on it?

One thing we noticed in our group was how we started working on the something and when about to finish it, we realized there wasn't an issue entry for the same. Such issues would often have a quick creation and end time and although it would have been completed, is not good from team's engagement and information point of view. These were the issues which were closed too soon after they were created.

4.5 Not telling how you are doing it

We all learn from every member in the team, be the technical aspect or the managerial one. We noticed sometimes that an issue was created and closed without any comments and information on it. It is good for a team to interact on an issue, discuss the problem and resolution in comments. If there is no comment activity on something, this is something we wanted to check.

4.6 Someone's pushing most work

In a group projects, it does happen that someone takes a center seat and does a majority of the work. In this scenario, a specific team member will have often larger number of GitHub commits than the other members of the team.

4.7 Not closing your own goals

A good proactive team is the one which sets up a deadline and completes the tasks and closes it. For the same reason, we searched for the milestones which were closed past their due date.

4.8 Taking long breaks

Did the team work a lot initially? Did the team work a lot towards the end of the deadline? Or, did the team was proactive throughout, which is a good sign? We checked into how the teams were working and committing the same to GitHub on a weekly basis.

4.9 Are you a good tester?

Any product needs good testing, there are always bugs, some you might expect, others you might not. More bugs is an indication of how the team kept testing and digging the issues in their solutions. We checked the

usage of 'bugs' label from GitHub. Less usage of the same is often an indication of not proper level of testing.

4.10 Overusing a label

If a team is using a label excessively, it implies that it could be a sign that the category is big or complex enough and should have been broken down into sub-labels. If a label was overused more than a threshold value, this is an indication of that.

4.11 Someone's doing most coding

A major indication of a user's activity is the additions and deletions, he/she is doing. We looked into the effective additions ratio (additions/deletions) per user to see how the users in the teams were doing that. A value of more than 1.5 as a threshold times indicates that a user added the additions effectively.

4.12 Who is the manager here?

Is a specific team member planning more of the work? If so, he would be the one creating most of the issues in the team's repository. A somewhat similar contribution from the team indicates a good team activity.

4.13 A lot of 'unassigned' work

We noticed in our project that often when an issue was created, it was unassigned. This led to a confusion as to who would be working on the same and is not a good practice. For the same reason, we looked into the issues and to see if they were unassigned.

4.14 Closing tasks towards the end

We wanted to find if majority of issues are being closed towards a particular week which would be a sign of improper task management and delays.

5. FEATURE DETECTION RESULTS

From the data extracted and the querying on the feature extracted we defined earlier, we got the below results for the three teams. We also created visuals on the same for better interpretation.

5.1 Trying to push to the next sprint!

The mean and standard deviation of duration of issues is calculated from the issues table in our SQL database and then the following query is run on the table. Benchmark is calculated from mean+ 2*standard deviation. We checked if there were issued opened for long.

SQL Query: select AVG (Duration/ (24*60)) as MeanDuration,stddev(Duration/(24*60))asStandardDeviationDuration , team from issues group by team;

Data:

Hours/Team	Mean	Standard Deviation	Benchmark
Team 1	673.3308	909.7405	2492.812
Team 2	289.3816	490.5221	1270.426
Team 3	139.9921	259.9236	659.8393

Table 3: Issues duration values

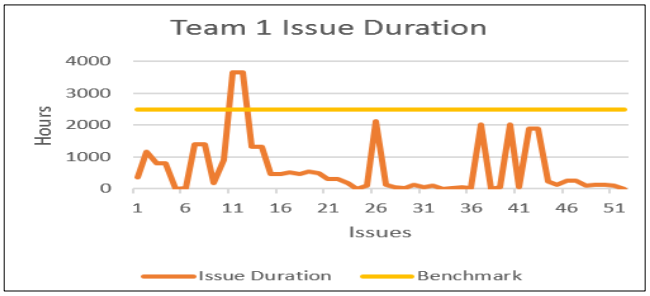

Figure 1a: Issues duration of team1

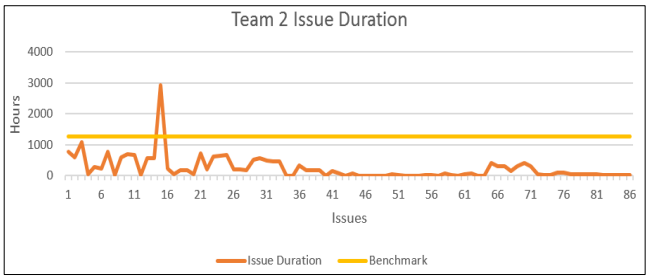

Figure 2b: Issues duration of team2

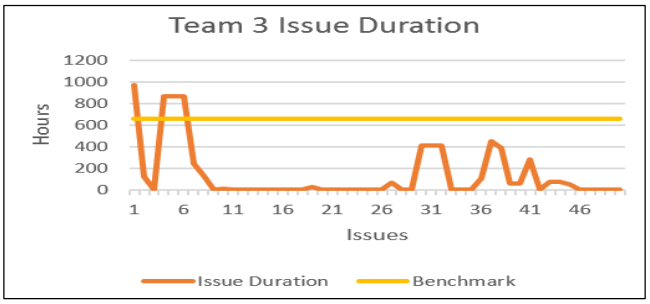

Figure 3c: Issues duration of team3

5.2 Individual or a team effort?

This distribution is calculated from Issues table in our SQL database and then the following query is run on the table.

SQL Query: select count(distinct issueno), assignee, team from issues group by assignee, team order by team;

Data:

	Team1	Team2	Team3
None	11.53	9.30	48.21
User 1	19.23	27.90	1.78
User 2	21.15	22.09	28.57
User 3	21.15	15.11	16.07
User 4	26.92	25.58	0

Table 4: Issue assignment values

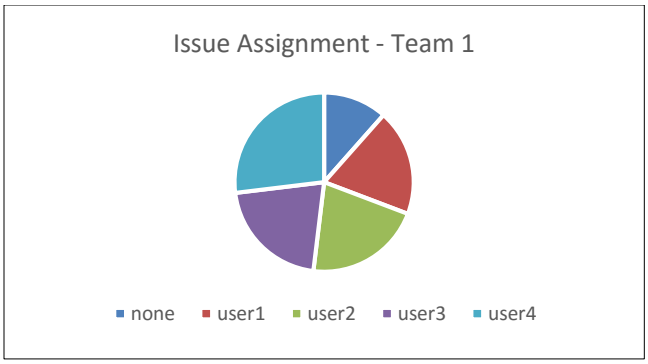

Figure 2a: Issue assignment distribution of team 1

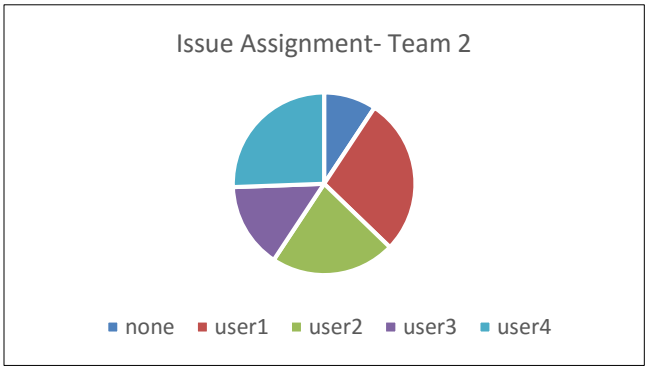

Figure 2b: Issue assignment distribution of team 2

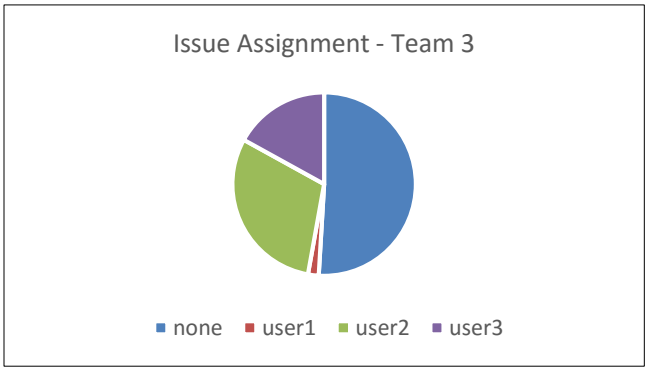

Figure 2c: Issue assignment distribution of team 3

5.3 Work without deadline, not good!

The milestone set up a deadline for solving the issue. This is a very helpful tool. This feature counts the number of issues without milestones. This count is calculated from the issue table in our SQL database and then the following query is run on the table.

SQL Query: select count (distinct issueno), milestone, team from issues where milestone like 'none' group by milestone, team;

Data:

Team	Percentage of issues without milestone
1	1.92
2	0
3	51.78

Table 5: Issues unassigned to milestones

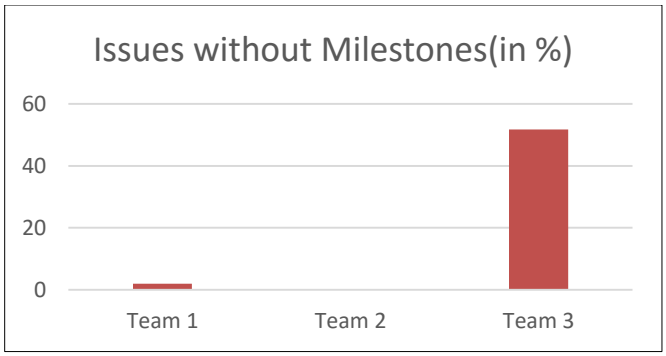

Figure 3: Percentage of issues without milestones

5.4 Did you tell after you already worked on it?

This count is calculated from the issue table in our SQL database and then the following query is run on the table.

SQL Query: select count(distinct issueno), team from issues where duration<15*60 group by team;

Data:

Team	Percentage of Issues closed too soon
1	1.78
2	6.97
3	38.46

Table 6: Issues closed too soon

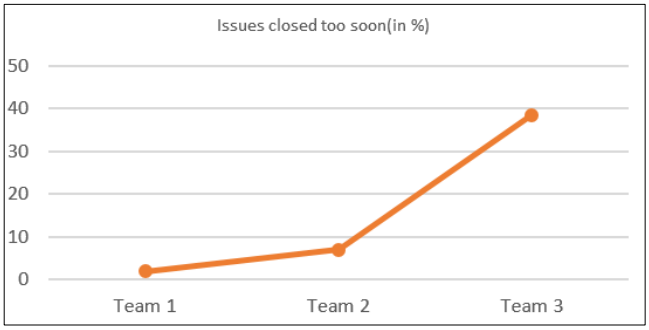

Figure 4: Percentage of issues closed too soon

5.5 Not telling how you are doing it

This count is calculated from the issue table in our SQL database and then the following query is run on the table.

SQL Query: select count(ISSUENO),team from ISSUES where COMMENTS < 1 group by team;

Data:

Team	Issues% with no comments
1	19.64
2	32.55
3	30.76

Table 7: Issue with no comments values

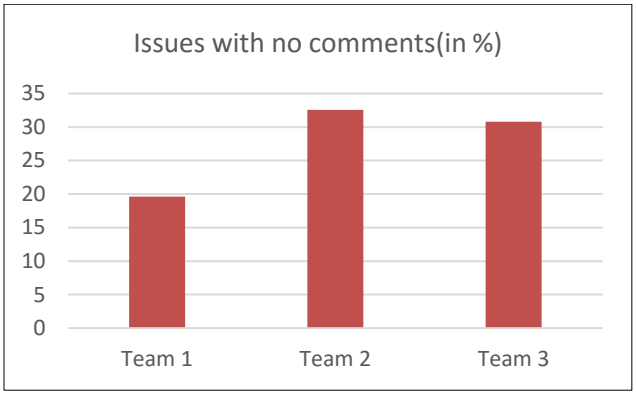

Figure 5: Percentage of issues with no comments

5.6 Someone's pushing most work

We calculated commit rate for each user of a team. This distribution is calculated from commit table in our SQL database and then the following query is run on the table.

SQL Query: select count(*), creator, team from commits group by creator,team order by team;

Data:

	Team1 %	Team2 %	Team3 %
User 1	16.66	23.61	16.07
User 2	25.64	20.83	36.60
User 3	7.69	27.083	26.78
User 4	50	28.47	20.53

Table 8: User contribution to commits

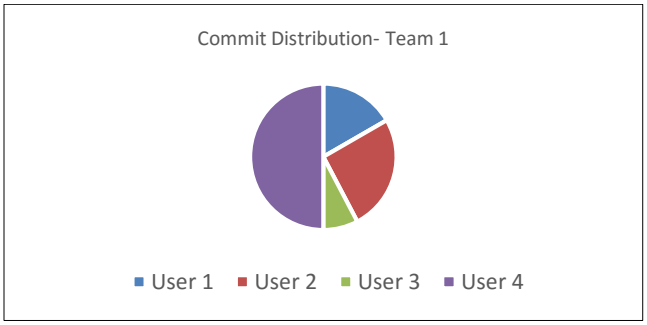

Figure 6a: Commit distribution of Team1

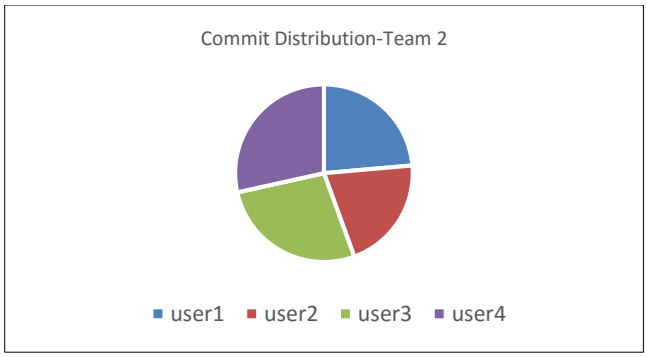

Figure 6b: Commit distribution of Team2

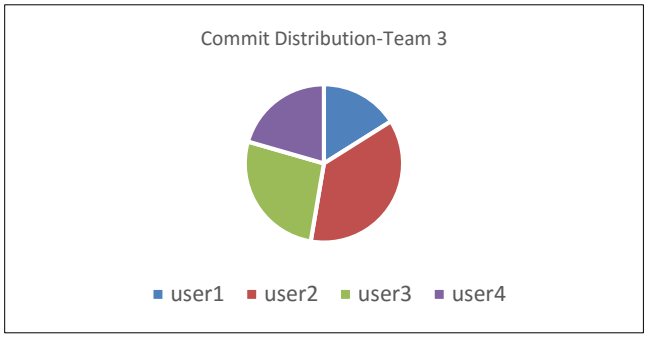

Figure 6c: Commit distribution of Team3

5.7 Not closing your own goals

This count is calculated from the Milestone table in our SQL database and then the following query is run on the table.

SQL Query: select count(*),team from mile where due_at<closed_at group by team;

Data:

Team	No. of milestones closed after due date
1	8
2	5
3	6

Table 9: Milestones not closed in time

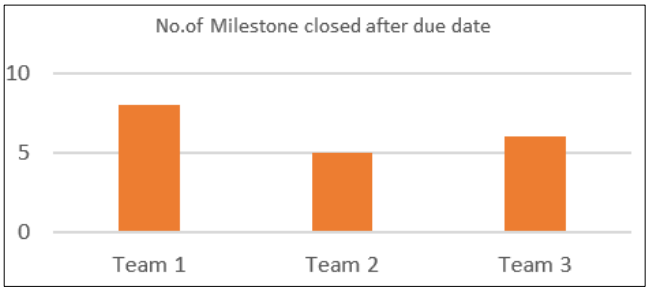

Figure 7: Milestones count after due date

5.8 Taking long breaks

This percentage is calculated from the commits table in our SQL database(mainly focusing the development phase) and

then the following query is run on the table. We did notice that during some weeks, no teams had any commits to their repository.

SQL Query: select count(*), team, to_char(to_date('1970-01-01', 'YYYY-MM-DD') + numtodsinterval(time, 'second'), 'ww')-1 as week1 from commits where to_char(to_date('1970-01-01', 'YYYY-MM-DD') + numtodsinterval(time, 'second'), 'ww')-1 in (5,6,7,8,9) group by to_char(to_date('1970-01-01', 'YYYY-MM-DD') + numtodsinterval(time, 'second'), 'ww')-1, team order by team,count(*);

Data:

	Team 1	Team 2	Team 3
Week 5	20.63492	0	0
Week 6	0	4.347826	0
Week 7	7.936508	23.91304	34.61538
Week 8	65.07937	71.73913	55.12821
Week 9	6.349206	0	10.25641

Table 10: Work distribution in weeks

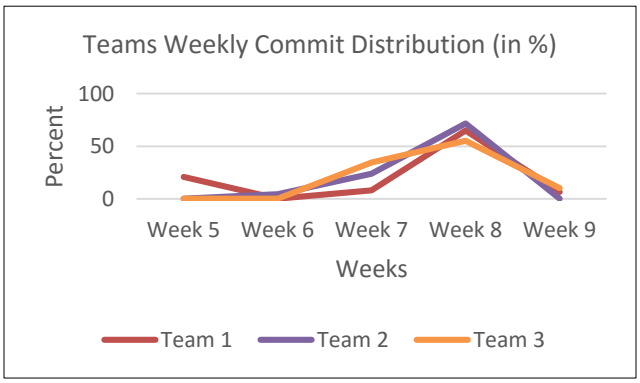

Figure 8: Commit distribution in the development phase

5.9 Are you a good tester?

This count is calculated from the Issue table in our SQL database and then the following query is run on the table.

SQL Query: select count(distinct issueno),label,team from issues where label ='bug' group by label,team order by team;

Data:

Team	% usage of 'Bugs' label
1	16.07
2	4.65
3	11.5

Table 11: 'Bug' label usage

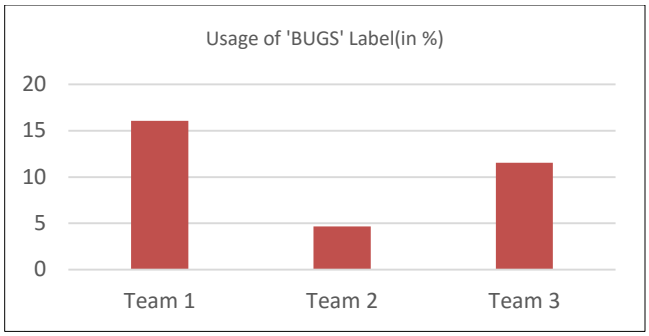

Figure 9: Bugs label usage

5.10 Overusing a label

This count is calculated from the issues table in our SQL database and then the following query is run on the table.

Benchmark is created on the basis of 1.5*(mean of counts for each label)

SQL Query: select count(DISTINCT issueno), label, team from issues where not label = 'Null Label' group by label,team order by team;

Data:

	Mean	Benchmark	Count
Team 1	8	12	2
Team 2	14.8	22.2	2
Team 3	10.6	15.9	1

Table 12: Overusage of labels

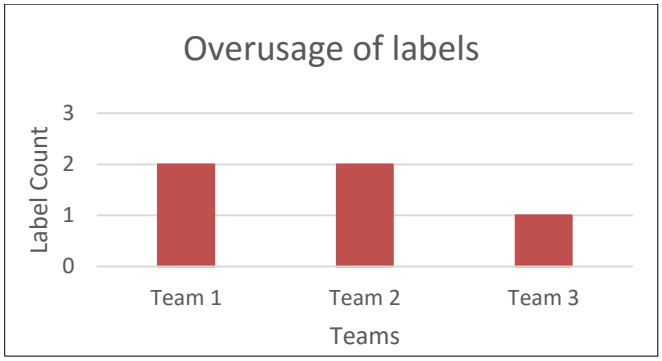

Figure 10: Count of the labels being overused for all teams

5.11 Someone's doing most coding

This was calculated from the lines of coded added/deleted by the users. Effective addition value = (no. of added LOC) / (no. of deleted LOC). If the effective addition value < 1.5, the user is not an effective contributor.

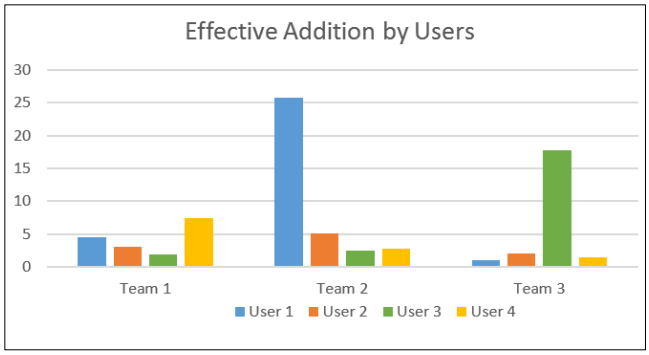

Figure 11: Ratio of additions to deletions

5.12 Who is the manager here?

The issue creation distribution per user in a group is calculated from the Issues table in our SQL database and then the following query is run on the table.

SQL Query: select count(*), creator, team from commits group by creator,team order by team;

Data:

	Team1 %	Team2 %	Team3 %
User 1	19.64	29.06	3.84
User 2	21.42	27.90	59.61
User 3	21.42	22.09	32.69
User 4	33.92	27.90	5.76

Table 13: Issue creation distribution

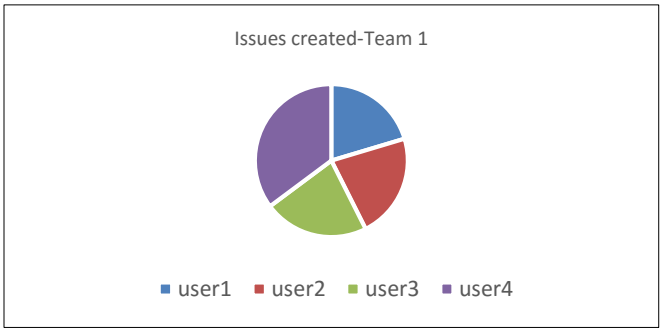

Figure 12a: Issue creation distribution of Team1

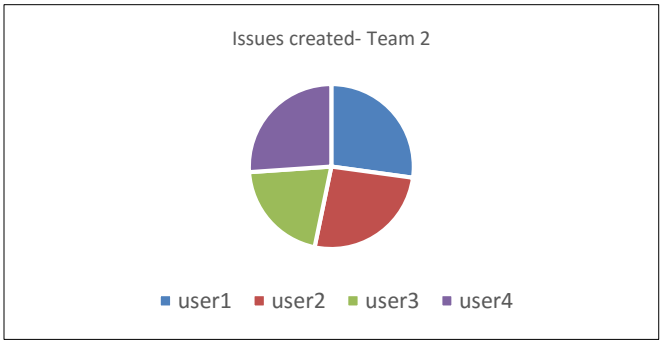

Figure 12b: Issue creation distribution of Team2

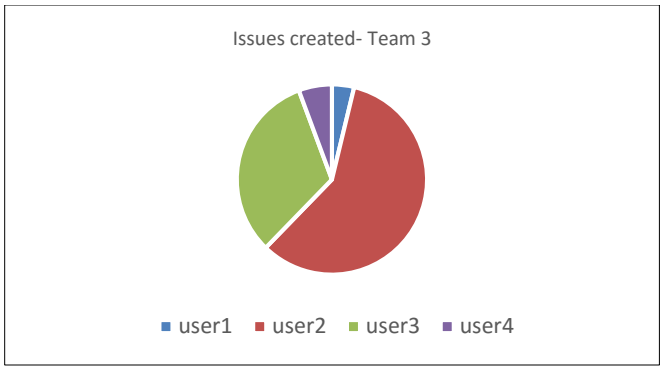

Figure 12c: Issue creation distribution of Team3

5.13 A lot of 'unassigned' work

This count is calculated from the issue table in our SQL database and then the following query is run on the table.

SQL Query: select count (distinct issueno),team from issues where assignee = 'none' group by team;

Data:

Team	% of Issues unassigned
1	11.53
2	9.30
3	48.21

Table 14: Issues unassigned percentage

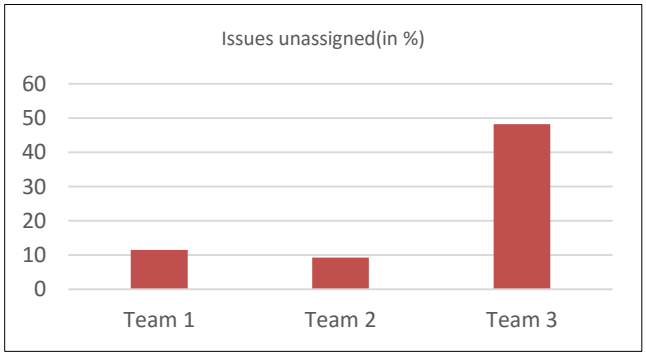

Figure 13 Percentage of issues unassigned in all teams

5.14 Closing tasks towards the end

This feature is calculated from the commits table in our SQL database and then the following query is run on the table.

SQL Query: select count(distinct issueno), team, to_char(to_date('1970-01-01', 'YYYY-MM-DD') + numtodsinterval(issueclosedat, 'second'), 'ww')-1 as weeknumber from issues group by to_char(to_date('1970-01-01', 'YYYY-MM-DD') + numtodsinterval(issueclosedat, 'second'), 'ww')-1, team order by team,to_char(to_date('1970-01-01', 'YYYY-MM-DD') + numtodsinterval(issueclosedat, 'second'), 'ww')-1 ;

Data:

Week no.	Team 1	Team 2	Team 3
1	0	0	0
2	1	0	0
3	0	0	0
4	5	0	2
5	1	0	0
6	0	3	0
7	1	19	27
8	28	36	13
9	0	0	8
10	0	0	0
11	0	0	0
12	0	3	0
13	10	24	0
14	16	0	3

Table 15: Work distribution in weeks

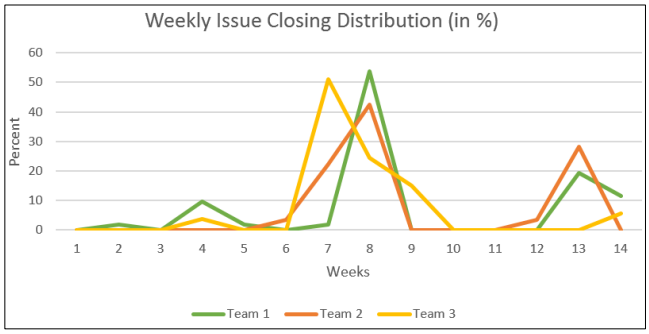

Figure 14: Issue closing distribution throughout weeks

6. BAD SMELLS DETECTOR

We combined our extractors into the categories depending upon how they would result in a bad smell. We list the below detectors we decided on, what it means and what feature extractors contributed to that. We have denoted feature extractors as F<<Number>> further for easy usage at some places.

6.1 Poor time management

We wanted to check the issues of all the three which had been opened for too long than the threshold. This is an indication that either the issue is tough or more often that the work is being kept on hold and delayed. This involved the following feature extractors:

- F3- Work without deadline, not good!
- F7- Not closing your own goals
- F8- Taking long breaks

6.2 Poor communication

Poor communication is a sign when people are not informing properly what they are doing, there are not comments on how they are doing it, or they take up work and close it fast without effective communication. This results in an unhealthy team environment with team members not being on the same sometimes. Hence, we looked to see how this panned out for the teams.

- F4- Did you tell after you already worked on it?
- F5- Not telling how you are doing it

6.3 Poor team work

An effective team is the one planning their tasks well, closing it when done, and engaging with the members well. There are scenarios when a lot of work is being pushed or done by a particular team member or someone particular who is managing most of the project. We wanted to highlight the same, so this can be a learning on effective team work for future.

- F2- Individual or a team effort?
- F6- Someone's pushing most work
- F11- Someone's doing most coding
- F12- Who is the manager here?

6.4 Poor task management

A good thing when managing a project on GitHub is effective and good utilization of GitHub features, managing creation and closing of tasks, using labels effectively, managing tasks assignment well.

- F1- Trying to push to next sprint
- F9- Are you a good tester?
- F10- Overusing a label
- F13- A lot of 'unassigned' work
- F14- Closing tasks towards the end

7. BAD SMELLS RESULTS

If a team had a bad smell on a particular feature i.e. they exceeded the threshold on the particular feature, we assigned a score of 1 to it as an indication of bad smell. If they did well as a team in the feature category, they were assigned a score 0. We assigned this score to each of the teams depending upon if they had a bad smell on a feature or not. The total score for the team would be the Bad Smell Score (BSS).

7.1 Poor time management

Feature	Team1	Team2	Team3
F3	0	0	1
F7	1	1	1
F8	1	1	1
BSS	2	2	3

Table 16: Poor time management BSS

This gives an indication on how the team performed from the time management point of view. All the teams lagged in this aspect, especially from the point of view of dividing the work properly and doing it in time within the deadlines. All three teams also lagged in the aspect of closing up the goals which they had originally set. However, teams 1 and 2 did well from the point of view of defining the deadlines for their work.

7.2 Poor communication

Feature	Team1	Team2	Team3
F4	0	0	1
F5	1	1	1
BSS	1	1	2

Table 17: Poor communication BSS

We noticed that all the three teams were not effectively communicating and interacting on the platform explaining things the way they were doing it. Interaction using comments is a good way to keep everyone in loop and informed. We also noticed that team 3 were defining and closing some issues too soon, which is a sign of improper division of work. It is a good practice to divide the work equally depending upon the complexity of tasks.

7.3 Poor team work

Feature	Team1	Team2	Team3
F2	0	0	1
F6	1	0	0
F11	0	0	1
F12	0	0	1
BSS	1	0	3

Table 18: Poor team work BSS

Here, team 2 did exceptionally well and had all their tracks covered. From the work assignments, to dividing the work properly and effective contribution from all the team members, they were good.

Team 1 did fairly good as well except the fact that a single team member was committing a major proportion of the work. They had the other activities sorted out. Team 3 again lagged here, since they didn't have proper assignment towards different team members, major proportion of the work was being inclined towards a single team member.

7.4 Poor task management

Feature	Team1	Team2	Team3
F1	1	1	1
F9	0	0	0
F10	1	1	1
F13	1	0	1
F14	1	1	0
BSS	4	3	3

Table 19: Poor task management BSS

This is one factor where all teams were behind. The task management wasn't good for any of the three. All the teams had some issues opened for a very long time indicating the delay it would case in the work. Team 3 were good at closing the issues in time and not waiting for it till the approaching deadlines

In the below figure (Figure 16), we illustrate the summary of our Bad Smell Scores for the various bad smells.

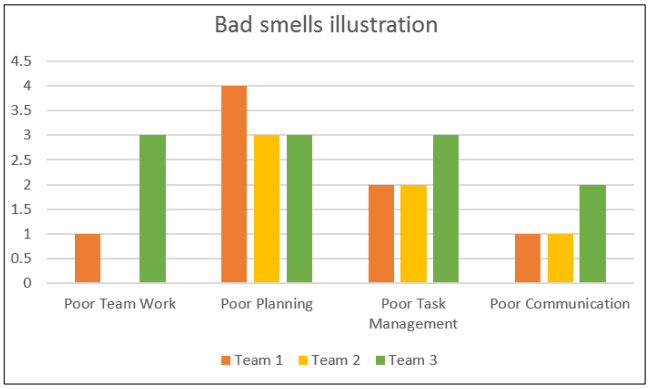

Figure 16: Bad smells scores for all teams

8. EARLY WARNING

For early warning detection, we focused on the creation time of issues by the team. The underlying assumption was that if a team has a proper project structure and a stable development plan, there will be a set pattern in the issue creation. If a team creates many issues at a short span of time, it symbolizes that the team is rushing without a plan. On the other hand, if the team does not create any issue for a long time, it depicts that team is not working properly and is not on schedule.

9. EARLY WARNING RESULTS

We plotted the interval between creation-time of adjacent issues in seconds. Peaks depict long gaps between issue creation and flat line depicts rush period. An ideal graph will be a constant line with few fluctuations.

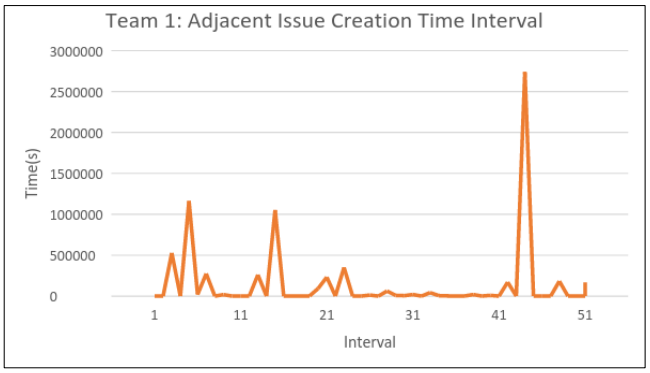

Figure 17a: Issues creation time intervals for team 1

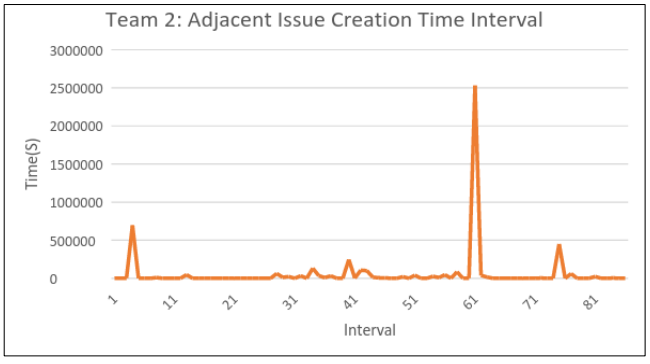

Figure 17b: Issues creation time intervals for team 2

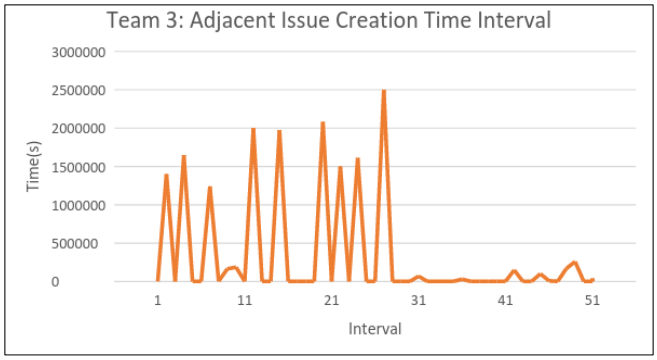

Figure 17c: Issues creation time intervals for team 3

From the above graph, it can be seen that the team 1 and 2 have an almost stable pattern of issue creation with only a few rush periods (fluctuations). Whereas, team 3 has long intervals in the beginning of the development and then a rush period for the rest of the development phase.

Feature	Team1	Team2	Team3
F1	1	1	1
F2	0	0	1
F3	0	0	1
F4	0	0	1
F5	1	1	1
F6	1	0	0
F7	1	1	1
F8	1	1	1
F9	0	0	0
F10	1	1	1
F11	0	0	1
F12	0	0	1
F13	1	0	1
F14	1	1	0
BSS	8	6	11

Table 20: Overall Bad Smell Score

From the Table 20, we noticed that out of all the three teams, team 3 has the highest bad smell score (BSS) which was the result of analysis of our feature extraction. We had similar insights from our early warning detector which is in-line with our overall bad smell analysis.

10. CONCLUSION

From our above results, it can be concluded that proper project planning, even distribution of tasks throughout the Software Development Life Cycle and active team engagement are the key factors for an efficient project planning and delivery.

11. REFERENCES

[1] https://gist.github.com/timm/a87fff1d8f0210372f26

[2] https://github.com/shivamgulati1991/CSC510-SE-group-i/blob/master/May1/gitable_new.py

[3] https://github.com/shivamgulati1991/CSC510-SE-group-i/tree/master/May1/db_csv_exports_data