GitHub Bad Smells Observation and Detection

Harshal Gala
North Carolina State University
hgala2@ncsu.edu

Pooja Gosavi
North Carolina State University
pigosavi@ncsu.edu

Qiufeng Yu
North Carolina State University
qyu4@ncsu.edu

***Abstract*—Structural characteristics of a software that is a hint that some flaw in your code is making your software hard to evolve and maintain is a bad smell in your code. [1]This phenomenon may trigger some refactoring of your code. Some early warning or detection of such 'bad smells' will help the developers correct their process at an early stage. Thus, it is a good practice of software engineering to detect 'early smells' and reduce the wastage in development effort. This project extracts and analyzes the GitHub repository data to find possible bad smells. The data collection is done using a python script and organized using an SQL Database. We have used certain feature detectors and queries for the collected data in order to find and analyze the bad smells.**

***Keywords*—github, bad smells, commits, issues, milestones, comments**

I. DATA COLLECTION

1. The data we collected

We decided on three GitHub Repositories for this purpose - including our own repository. We collected some data from each of these repositories, enough for our analysis. It can be categorized as -

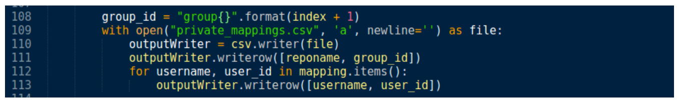
1) Issues
   - Issue ID
   - Issue creation time
   - The user to whom this issue was assigned
   - Milestones assigned
   - Number of Comments
   - Actions on this issue(Labeled/not labeled)
   - Date on which this issue was closed
   - Duration
   - What was the issue about
2) Commits
   - Time in Week
   - Number of Commits
   - Users who did the commit
   - No. of lines added
   - No. of lines added in a commit
3) Comments
   - Comment ID
   - Issue ID
   - The user who has commented
   - Comment body
   - Created at
   - Updated at
4) Milestones
   - Milestone ID
   - Created at
   - Closed at
   - Due at
   - Issue Closed
   - Issues Open

2. The way we collected data

We had access to gitpy.py [2] script and modified [3] the same in to extract the desired data from GitHub as per our needs. We desired a well structured data format, so the script output was directly stored into CSV files. To analyze the data thus obtained, we stored it into a MySQL database and imported the CSV files to create the tables for issues, milestones, commits and comments.

II. ANONYMIZING THE DATA

In order to anonymize the data for the groups and the repository members, the modified gitable.py contains a script which generates a private_mappings.csv which allowed us to hide the groups and their users from the actual analysis. Thus, the repositories were named as group1, group2 and so on while their members were renamed to user1, user2 etc. Thus, the groups and their users were anonymized.



```
        group_id = "group{}".format(index + 1)
        with open("private_mappings.csv", 'a', newline='') as file:
            outputWriter = csv.writer(file)
            outputWriter.writerow([reponame, group_id])
            for username, user_id in mapping.items():
                outputWriter.writerow([username, user_id])
```

Fig. 1: Anonymizing the data

III. DATA

Combining the data from all the repositories [4], we got the following numbers in Table 1:

TABLE I: Data collected

Issues	Comments	Commits	Milestones
172	384	533	30

IV. SAMPLE DATA COLLECTED

Below you will find a few data samples that we collected.

Sample data from the issues:

TABLE II: Sample from the issues

issue_id	33
when	2017-02-23T01:32:40Z
action	labeled
what	approach2
user	user2
milestone	Approach 2
comments	0
issuecreated_at	1487813560
issueclosed_at	1490983215
duration	36.68582176

Sample data from the comments:

TABLE III: Sample from the comments

recid	52
issueID	10
user	user2
createtime	1486433486
updatetime	1486433486
text	Does not that come under March milestone?
identifier	277879765

Sample from the milestones:

TABLE IV: Sample from the milestones

id	1
opened_at	2017-03-09T00:24:41Z
closed_at	2017-03-14T07:00:00Z
due_on	
closed_issues	2
open_issues	0

Sample from the commits:

TABLE V: Sample from the commits

week	21-Jan
commits	6
additions	6132
additions_per_commit	1022
user1	0
user2	0
user3	6
user4	0

V. BAD SMELLS

A. Creation of Issues by each team member

This feature depicts the fraction or the total number of issues the team member have created. This shows the involvement of the team member in the project. Group effort and proper work division are the important ingredients of successful project.

1) Feature Detection: We separated it on the basis on the way a team member is involved in the project. The different issues created by the members and it is shown by the pie-chart of each team the way the issues were created by which member showing the one who took all the initial.

2) Bad Smell Detection: Disproportionate creation of issues done by the user with respect to the team members shows that a sign of bad smell. As, we can notice that the work to be done in team effort is shown is handled by only a particular member as others have not involved themselves as much.

3) Results:

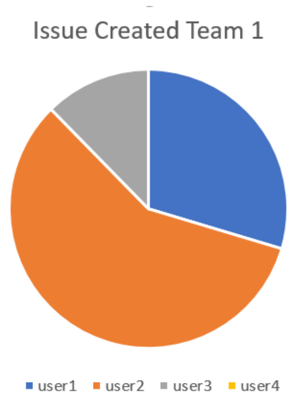


Fig. 2: Team 1 Issues created

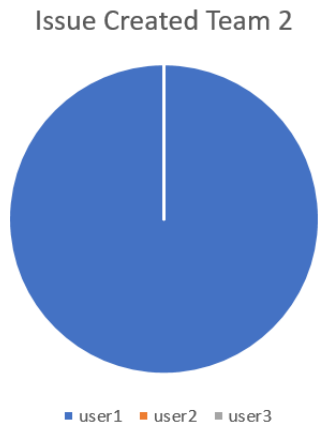


Fig. 3: Team 2 Issues created

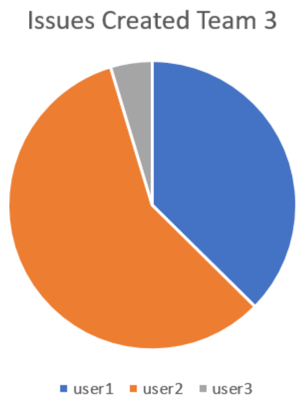


Fig. 4: Team 3 Issues created

From the diagram we can notice that of majority of the issues were created by user1 and user2 but user3 just made some contribution. This group then too stand above the other two teams in the work distribution point of view.

For the second team that we evaluated, we can see that, of all the member only one member i.e. user1 have been making issues and other two have not even created a single issue. This shows a bad smell feature where the user1 is been doing initiation work. It is seen a bad example of team work.

For team 3, we can conclude from the diagram that of all the users user2 have created more than 50% of the issue and user4 have given no contribution in making these issues. This shows that user4 making no issues shows a bad smell feature.

B. *Fraction of issues assigned to each team member*

This feature depicts the fraction or percent of total issues that each team member handled. Teamwork and proper planning are important ingredients for success of any project.

1) Feature Detection: We separated it on the basis of the issues assigned to each team member. It is represented in a graph as a fraction of the total number of issues assigned per team member.

2) Bad Smell Detection: Unequal distribution of work is a bad smell. If a repository has a high discrepancy in the issues assigned to a person, it qualifies for this bad smell.

3) Results:

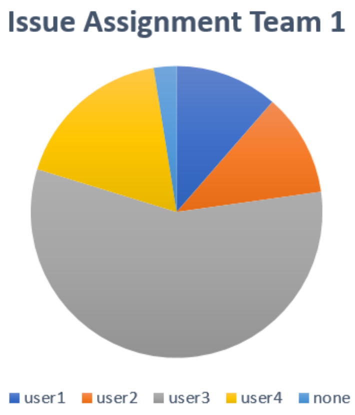


Fig. 5: Team 1 Issues created

For team 1, as you can see from the graph, the number of issues assigned to user3 exceed the others. This shows unequal distribution of work. This team the distribution of work is not near to equal then too there is some work distribution between the team members of them one is doing majority of the work.

The number of issues assigned to user1 exceed the others in team 2. This shows unequal distribution of work. This shows majority of the managing part and setting up the issues and it's assignment is been handled by only one member.

For team 3, the number of issues assigned to user3 also exceed the others. This shows unequal distribution of work. Same is the scenario that we see user1 have been assigned the major number of issues thereby showing the unequal division of the work.

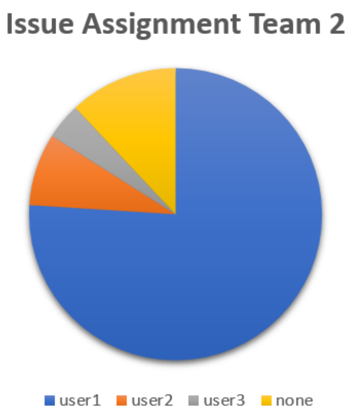


Fig. 6: Team 2 Issues created

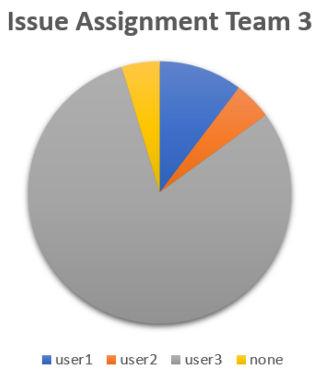


Fig. 7: Team 3 Issues created

C. *Duration of issues - short lived vs. long lived*

Issue that are short-lived identify as a bad smell since it shows that the issue was not needed. One of the reasons for such short lived issues is to present virtual progress by increasing the repository. Long lived issues indicate the efforts put in resolving issues and the complicated problems faced during the project. They usually show the kind of hacks and efforts involved in the code to overcome the problem.

1) Feature Detection: For this part, we have used the following issues data for comparison

- Issue_number
- created_at
- closed_at

2) Bad Smell Detection: The bad smell was determined by the graph of the issue_number against the duration of the issue in days calculated as per the created_at and closed_at time stamp.

3) Results:

Team 1 has an equal distribution of all the issues. From the different issues duration you can see how the different issue duration shows that all the issues were having a defined time period thereby showing the way issues were used to

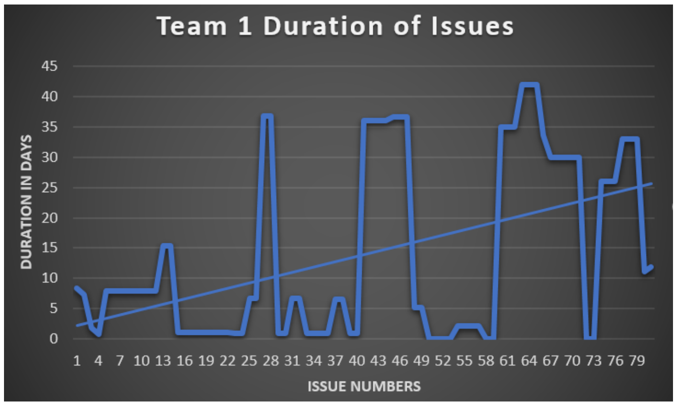


Fig. 8: Team 1 Duration of issues

communicate the problem and shows how it helped them in going and dividing the problem faced in the project.

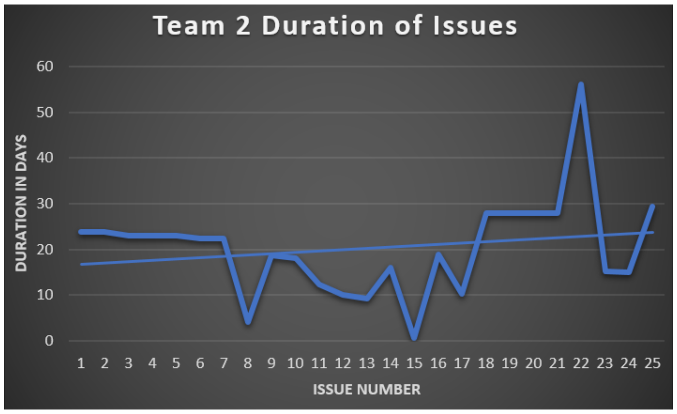


Fig. 9: Team 2 Duration of issues

Team 2 has more number of short lived issues than the long lived ones. They qualify for the bad smell. As the number issues were quite short we get to know that the team had not communicated the different problems related to the problem and have a very short usage of this feature of GitHub.

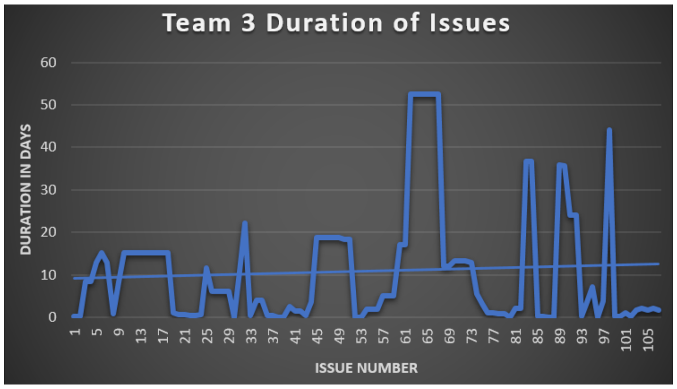


Fig. 10: Team 3 Duration of issues

Team 3 has an equal distribution of short lived and long lived issues. It shows the number of hacks they applied for this project. This distribution shows how initially they have not used issues while the project selection and report making process. In the end during implementation they have been combating the problems using the issues and assigning to each other.

D. Number of Issues without comments

This shows the different issues made by the team and of those the ones which were without comments i.e. without any discussion just the creation for issue for some task.

1) Feature Detection: This feature shows that there was either less communication between the team members and the issues were created which were tasks done individually without any discussion

2) Bad Smell Detection: The more number of ratio of issues without comment to the total number of comments shows the less communication in between the team members or else there might be some other way they were communicating.

3) Results:

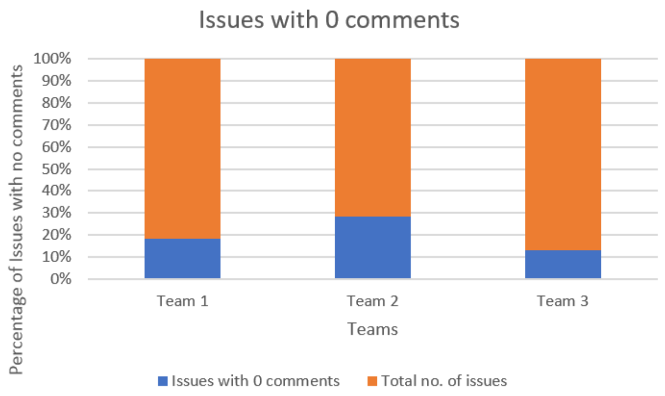


Fig. 11: Issues with 0 commits

Here we can see the different ratio of the issue with no comments to the total number of issues.

Team 1 - 82:18

Team 2 - 74:26

Team 3 - 88:12

Of these teams we see that all the teams had issues without comments. This we can say they were communicating from some other ways as well. But, the issues created by Team2 are quite less as seen in the complete analysis which points them as to feature in the bad smell property.

E. Uneven commits by the team members

Code commits are becoming to be considered as a measure of project development. This feature is to identify the person who has more number of commits - a dominant and the one who takes the team for granted - a freeloader.

1) Feature Detection: We identified the user and his corresponding commits by analyzing the group_commits.csv file.

2) Bad Smell Detection: The graphs shown in the results can help us identify clearly the dominants and the freeloaders from the corresponding teams.

3) Results:

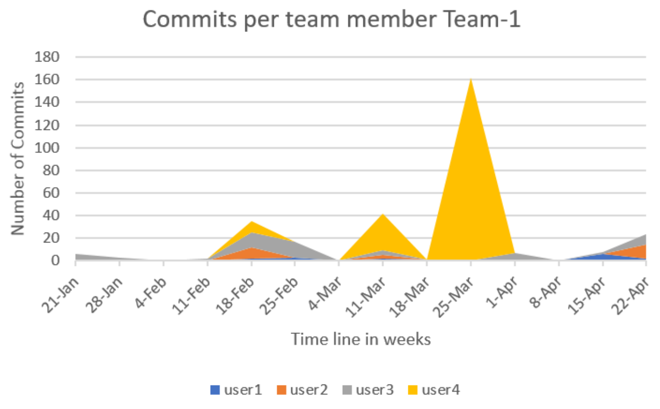


Fig. 12: Team 1 - Commits per team member

For team 1, the distribution seems quite consistent to early March for user1 and user2. The user3 has participated in the project contribution during January and a bit during April. We can see the distribution in such a way as we see whenever the submission deadline is near the number of commits has peaks during those period. In some of them there are all users some there is just a single user.

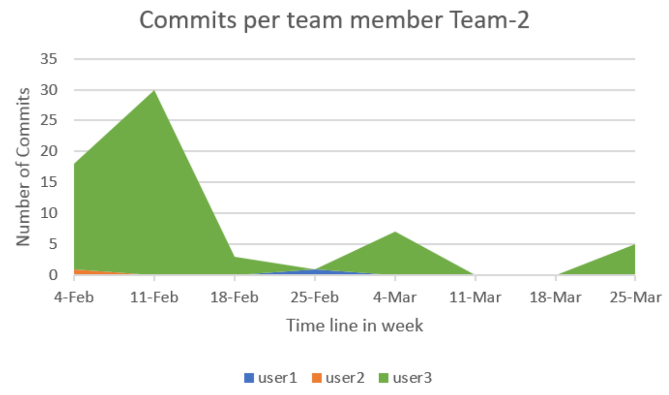


Fig. 13: Team 1 - Commits per team member

Looking at team 2, for the entire duration of the project, the user 3 is the dominant. We can see traces of user1 and user2 in some sections of the project however, they appear to be the freeloaders. Here also during the submission of the project all the work is done majoritively and here it is case of bad smell where no work distribution is shown and major work is done by just one member.

For team 3, we can see the equal distribution of commits during the months of February, however, after March, the user 4 is the clear dominant. User 3 seems to have shown consistent contribution to the project throughout. The contributions for users 1 and 2 are significant in February and April. User 4 have been doing majority of the implementation as the contribution

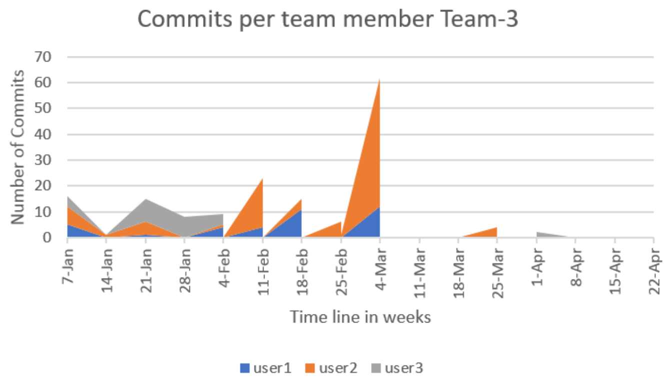


Fig. 14: Team 1 - Commits per team member

is maximum and others have been doing the remaining of work of report making and keeping daily updates.

F. *Issues exceeding milestone due dates*

The issues which exceed the milestone due date usually affects the overall pace of the project and its progress. More number of issues closed beyond the due date, the slower is the pace of the project.

1) Feature Detection: We identified each and every issue identified by issue number that was assigned to a milestone identified by milestone number and captured its milestones intended due date and actual closing date of the issue.

2) Bad Smell Detection: To qualify whether a team repository has a bad smell on the basis of number of issues exceeding milestone due date, we opted to select threshold of 30%, i.e. if a team repository has more than 30% issues that exceeded milestone due date, then it is qualified a bad smell.

3) Results: The consolidated graph of the issues closed to the milestones can be seen for the three teams in figure 15.

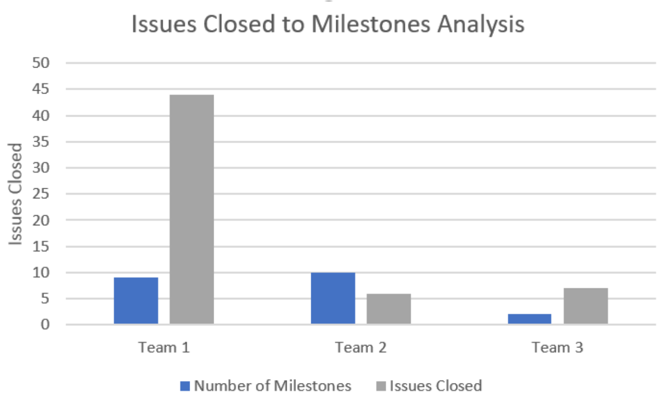


Fig. 15: Issues closed to milestone analysis

G. *Less number of comments on an issue*

1) Feature Detection: We attempt to determine if the issue status was updated by analyzing the number of comments associated with an issue. An updated issue should generally have comments explaining the different states. For example, in case of bugs, the tester should write the way to reproduce a bug. The developer trying to fix the bug should update his analysis on the bug which could include information about the root cause of the bug, whether it is a duplicate bug etc. It also makes sense to update the issue with the details of the check in that fixes it. Hence, the number of comments on an issue is a good indicator of a properly analyzed issue.

2) Bad Smell Detection:

For each repository we calculated

1) Maximum number of comments received by an issue
2) Minimum number of comments received by an issue
3) Mean number of comments

3) Results:

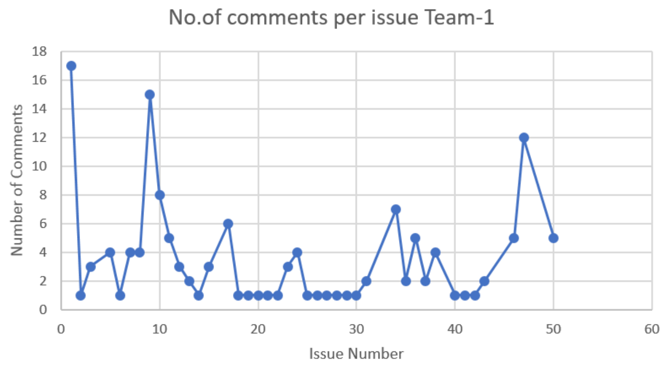


Fig. 16: Team 1 - Commits per issue

Maximum comments received by an issue = 17

Minimum Comments received by an issue = 1

Mean Number of Comments received by an issue = 8

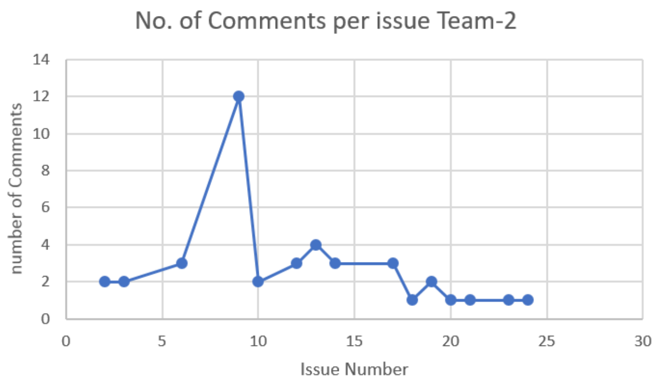


Fig. 17: Team 1 - Commits per issue

Maximum Comments received by an issue = 12

Minimum Comments received by an issue = 1

Mean Comments received by an issue = 4

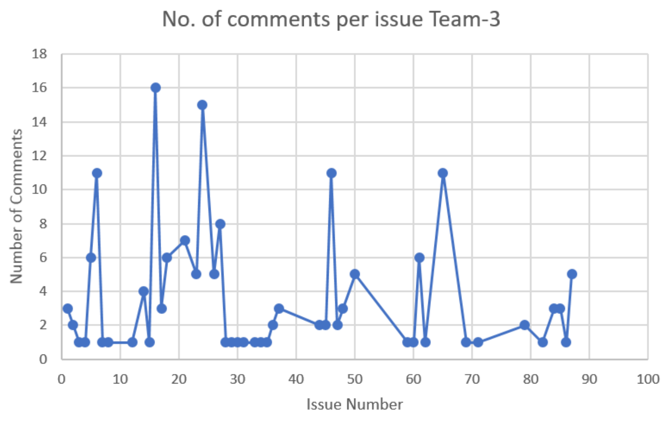


Fig. 18: Team 1 - Commits per issue

Maximum Comments received by an issue = 16

Minimum Comments received by an issue = 1

Mean Comments received by an issue = 6

H. *Uneven weekly commits by the team members*

This feature analyzes the total number of commits by the team members on a weekly basis. This determines whether the team has worked consistently throughout the project duration or worked only around the time of deadlines or submissions.

1) Feature Detection: In order to extract the features for this particular bad smell we extracted all the commits identified for the team along with their commit dates which we used to identify the week of commits.

2) Bad Smell Detection: To determine whether a team repository has a bad smell on the basis of weekly commits, we needed certain threshold to decide if group was regular or inconsistent during the project. It can be inconsistent if it under-worked or/and over-worked during some weeks. We decided these thresholds using the standard procedure, i.e. taking mean of all lines of commits.

3) Results:

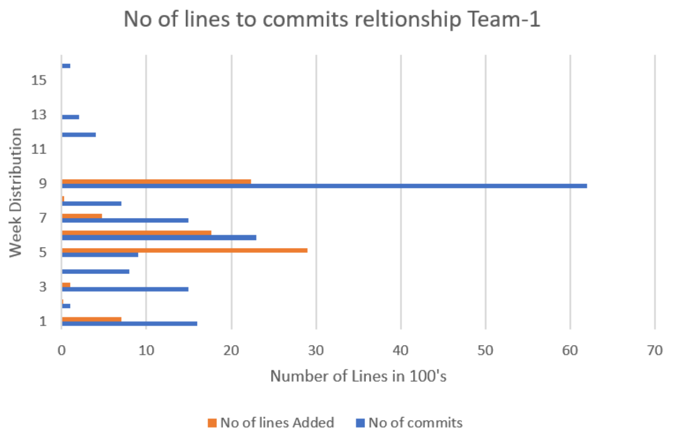


Fig. 19: Team 1

Team 1 has a total number of 143 commits over a duration of 15 weeks.

Mean = (143/15) = 9.53

Standard Deviation = 18.66

As it can be seen from the graph, we can say that the 5 weeks out of 15 are overworked and 4 are under-worked.

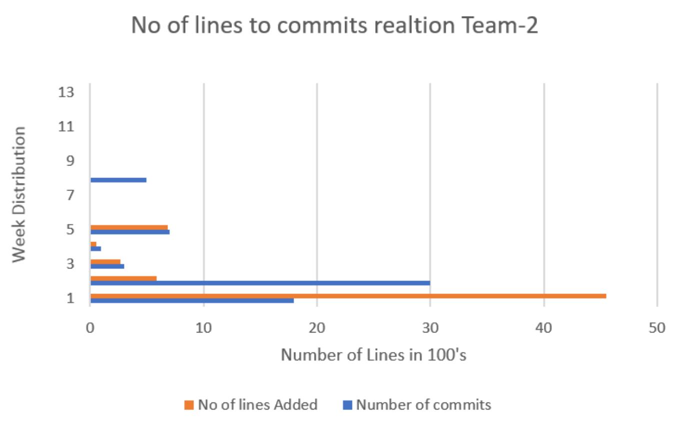


Fig. 20: Team 2

Team 2 has a total number of 65 commits over a duration of 13 weeks.

Mean = (65/13) = 5

Standard Deviation = 11.13

As it can be seen from the graph, we can say that the 3 weeks out of 13 are overworked and 7 are under-worked.

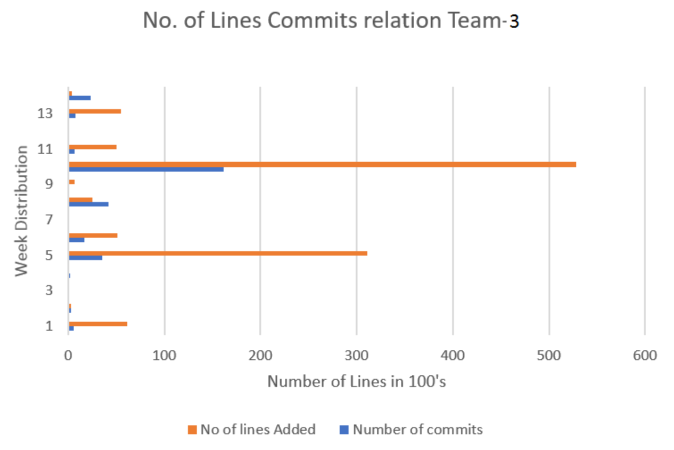


Fig. 21: Team 3

Team 3 has a total number of 325 commits over a duration of 13 weeks.

Mean = (325/13) = 25

Standard Deviation = 47.25

As it can be seen from the graph, we can say that their work is fairly distributed over all the weeks.

VI. A CONSOLIDATED OVERVIEW

We made a table to compare each type of bad smells among those three different teams:

TABLE VI: Bad smells among each team

Bad Smells	Team 1	Team 2	Team 3
Creation of issues by the team members	Yes	Yes	Yes
Fraction of issues assigned to each team member	Yes	Yes	Yes
Duration of issues short lived vs. long lived	No	Yes	No
Number of issues without comments	Yes	Yes	No
Uneven Commits by team members	Less in comparison to others	Yes	More than team repository 1
Issues exceeding milestone due dates	Yes	No	Yes
Less number of comments on an issue	Yes	Yes	No
Uneven weekly commits by the team members	Yes	No	No

VII. EARLY SMOKE DETECTORS

A. Early smoke detection on the basis of dominants and freeloaders

To identify the team members as dominants and freeloaders, we need to look at the individual member's contributions from somewhere in the middle of the project - week 7. If an individual is a freeloader with less than 15% of commits around middle of the project and some other person has more than 50% of the commits, it is most likely they will continue to be the same till the end of the project. Also other good indicator is week of their 1st commit in the project; it is most likely that freeloaders start committing late in the project.

B. Early smoke detection on the trend of missing the milestone due dates

We could see in the earlier graphs few of the teams miss their milestone due dates by a significant time. However, there were some early indicators that milestones in second half of the project will be missed based on the milestone completion trend in the first half of the project.

VIII. TIMELINE CHART FOR THE ENTIRE DURATION OF THE SE PROJECT

The repositories which we accessed belonged to various teams developing a software which would solve a real life problem using three different solutions. The main aim of the project was to help the teams understand how Software Development Life Cycle functions, experience its individual components and then at the last stage, analyze the work done in the project and identify the bad smells by analyzing the codes and the data collection residing in the repositories. We will briefly describe the various phases we encountered

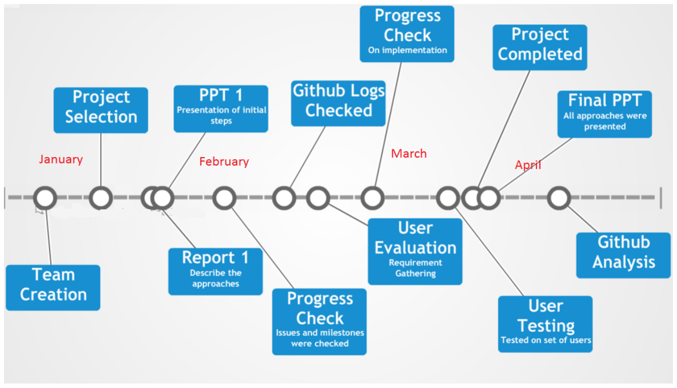


Fig. 22: Timeline

A. *The January phase*

It involved the process of identifying the problem in a real world scenario and trying to gather data about it. Various user surveys were conducted and enough data was gathered to jot down and present the requirements of this project.

B. *The February phase*

This phase was entirely spent in looking for three feasible solutions for the problem and once the design of the solutions was finalized, time was spent in implementing the solutions.

C. *The March phase*

The first quarter of this phase was spent in implementing the solutions, the remaining part was spent in testing the implemented solutions and obtaining results about them. User evaluations were conducted for the same and the users gave us their opinions regarding the application.

D. *The April phase*

Once the results were obtained and presented, it was time for the analysis of how much things that we learned so far regarding the ideal software development process have actually been implemented during our application development. It was during this phase that the repositories were anonymized and analyzed to obtain information regarding the techniques, procedures and standards followed during the entire software production phase.

IX. Conclusion

As we conclude this report, we would like to say that each of the team repositories we analyzed had some kind of a bad smell. Also, one of the aim of the course, the project and the bad smells analysis was to go through the software development lifecycle and learn about it “on the job”. Also, developing a software involves facing some issues and identifying ways to solve them. As software engineering practices are maturing and taking statistical procedures into account, they overcome some of the inherent problems of designing software. We have gained a significant insight into these github repositories and how these project may have been developed (including ours) using the above analysis. In future, we will avoid these bad smells while developing future applications.

References

[1] http://www.jot.fm/issues/issue_2012_08/article5.pdf

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

[3] https://github.com/SE-17-GroupQ/GitHub-Analysis/blob/master/gitpy.py

[4] https://github.com/SE-17-GroupQ/GitHub-Analysis