GitHub Analysis : Bad smell feature extraction and detection

Mohit Sardar
msardar@ncsu.edu

Saravanan Balasubramanian
sbalasu7@ncsu.edu

Siddharth Heggadahalli
ssheggad@ncsu.edu

1. INTRODUCTION

Bad smell, a commonly used term in computer programming, refers to a symptom that could possibly point to a bigger problem. Different teams work differently when it comes to software development. The way they interact and collaborate could be completely different. But, a very good analysis on how they operate could reveal some very useful information. The information that deviate from normal terms, and could possibly lead to a bigger problem can be considered as a bad smell. Bad smells are not bugs in the code, but the practices that weaken the design and in turn could possibly lead to increased development time, risk for bugs and failure.

GitHub is a web based version control repository, widely used by the software developers and project managers. Besides version control feature, GitHub offers other collaboration features such as bug tracking, feature requests, task managements and wikis. Github also provides APIs to collect the project details in JSON format. Analysing the GitHub data will help detect the bad smells and it could be used to overcome the problems that we discussed in the future projects.

In this paper, we have analysed the GitHub data of 6 projects that were completed as part of the Software Engineering(CSC 510) course at North Carolina State University in the year 2017.

2. DATA COLLECTION

GitHub provides APIs to the users that can be used to retrieve all the data about a project in the form of JSON file. We were given a basic working script that collects data about the issues that were created during the time of the project. We modified the script in such a way that all the collected data were written into a CSV file to make it more organized, making it more easy to read and do our interpretation. Besides the issues data, we also collected the Comments, Milestones, and Commits data as more data could give better understanding of the project. The data collected from GitHub were as follows.

1. Issues
   (a) Issue ID
   (b) Time of the event
   (c) Action Performed
   (d) Labels Used
   (e) User performed/affected
   (f) Milestone under which the issue appears
   (g) Issue opened time
   (h) Issue closed time
2. Milestones
   (a) Milestone ID
   (b) Milestone Title
   (c) Milestone created time
   (d) Milestone due time
   (e) Milestone closed time
3. Comments
   (a) Comment ID
   (b) Issue ID
   (c) User who commented
   (d) Comment created time
   (e) Comment updated time
4. Commits
   (a) User ID
   (b) Commit time
   (c) Commit message

3. ANONYMIZATION

As the main intention of this project is to identify the bad smells and not to evaluate the teams or the teammates who complete the project, we decided to anonymize both the teams and teammates information.

3.1 Team Anonymization

We collected ten GitHub repository data to perform our analysis. All the ten GitHub names were stored as strings in a list. Then, we performed a random shuffle operation on the list. After shuffling the list, we iterated through the list items and assigned team numbers from one to ten. Collected data were stored in separate CSV files for all the teams. We will be analysing 6 of the 10 projects and thus the teams have been named Team1, Team2, Team3, Team4, Team5 and Team6.

```
random.shuffle(team_list)
for repo in team_list:
    team_id = team_id + 1
```

3.2 User Anonymization

In order to anonymize the users, we are using a dictionary to store the user information. While analysing the issue, commits and comments data, we will encounter user information. When a user is encountered for the first time, he will be given a name such as user0, user1, user2, etc., and saved in the dictionary. When we encounter the same user next time, we will user the assigned name from the dictionary. We use python dictionary lookup to decide whether the user has been encountered already.

```
def anonymize_user(user_dict, user):
    if user_dict.get(user,None) == None:
        count = len(user_dict)
        user_dict[user] = 'user'+ str(count)
    return user_dict[user]
```

4. DATA AND DATA SAMPLES

In this paper, we will be analysing data for 6 projects. The cumulative numbers can be found in the following table.

Milestones	Issues	Comments	Commits
35	174	317	799

The following table shows the data sample stored in the Issues CSV file. Each event in an issue is recorded as a separate row in the file for in depth analysis.

Issue ID	When	Action	Label	User	Milestone	Created At	Closed At
5	1485921949	labeled	bug	user1	Testing	1485911945	1485922500

The following table shows the data sample stored in the Milestone CSV file. Each row corresponds to a milestone created with respect to the project. Different files are used for storing different team milestones.

Milestone ID	Title	Created At	Due At	Closed At
2280839	Jan Milestone	1485459449	1485849600	1486420737

The following table shows the data sample stored in the commits CSV file. Each row corresponds to a commit performed by the user to GitHub repository. Six different files are used for storing different team commit details.

User ID	Time	Message
user1	1491622624	bug fix

The following table shows the data sample stored in the comments CSV file. Each row corresponds to a comment performed by the user on an issue. Six different files are used for storing different team commit details.

Comment ID	Issue ID	User ID	Created At	Updated At
278453132	5	user2	1486605110	1486605110

5. FEATURE DETECTION

5.1 Issues with Long Life Time

The duration of each issue was calculated by subtracting the closing time of the issue from the opening time. Issue open time and close time from the Issue CSV file was used to perform the analysis.

Issues that are open for an abnormally long time is a bad smell since it could be a result of team members delaying working on an important issue or poor collaboration among team members.

But, it is hard to give a uniform threshold for an abnormally long duration for all teams since each team have their own methods of planning, estimating and assigning issues. Taking more time on an average per issue is not necessarily a bad smell since that could just be a result of a team assigning more work per issue. So, it is a better idea to look at each team separately and find a good measure of a long issue for that team specifically.

The process we used was to plot the graph of the duration (sorted in ascending order) and look for a huge spike in the graph. Data before this spike is more representative of team than data after the spike. So after calculating mean and standard deviation for chunk of data before line we consider anything beyond *(mean+1.5*std dev)* as abnormally long.

5.2 Issues with Short Life Time

Sometimes, people tend to work alone and forget to add the issues to GitHub. They tend to assign the work to themselves without informing the team. This could create a lot of problems such as two people working on the same issue. This in turn could lead to increased development time and could potentially affect the project as a whole since development time is one of the important factors for the project success. In our analysis, if an issue was alive less than 5 mins, it is considered a bad smell.

5.3 Single Participant Issues

Some team members either might not have the skills or the intent to help the project succeed. This will lead to increased workload on other team mates. There are times when the same user will have to close the issue himself/herself after not receiving any kind of help from the team members. Participation in an issue by a team member is determined by his/her activity in the issue. For example, commenting,subscribing are considered to be participation. We iterated through all the events with respect to a single issue and checked if more than one participant is involved. If two or more participants involved, we don't consider it as a single participant issue. Also, We have neglected the case where the issue was assigned the single participant is same as the user to whom the issue was assigned. And, when only one participant involved with no one assigned to that issue, we consider it as single participant issue.

5.4 Number of Issues Assigned to each Team Member

The number of issues assigned to each team member was collected as it is a very good indication of the amount of responsibility held by each team member. We can also identify the sharing of workload between team members. And also assess the performance of the members.

The number of issues assigned to each team member is collected. Here the TA and Professors are excluded. And also a list of users is generated to see if there are users who have not been assigned any issues at all.

5.5 Number of Issues Closed by each Team Member

The number of issues closed by each team member is very important as this shows whether the team member closed equal number of issues, also useful in identifying share of responsibility of each team member. It is also an indicator of initiative taken by team members and also the sharing of workload among team members.

The number of issues closed by each team member is collected. Here the TA and Professors are excluded. And also a list of users is generated to check if there are users who have not closed any issues at all.

5.6 Issues Closed after Milestone Due Date

Milestones are the results of planning and effort estimation. When the issues cannot meet the deadline posted, there is a poor project planning system in place. When there is an estimation error, it could even result in project cancellation. After retrieving the milestone due date detail from the Milestone CSV file, each issue events are iterated and the issue close time are retrieved from the issue CSV file. If the issue close time is greater than the milestone due date, the team has failed to close the issue before milestone deadline.

5.7 Milestones without a Due Date

There are also cases where the team members could be working without setting a due date for a milestone. Milestones are used to break down a big project into small set of achievable tasks to keep the team members focused and motivated. This clearly portrays that they didn't use GitHub properly and collaboration between the team was very bad. Such poor communication will lead to project failure. When there is no due date, it is not clear what the teams were working towards. This shows that the team haven't made any planning or estimation. We iterated through the milestone CSV file to determine the number of milestones for which a due date was not set.

5.8 Milestones never closed

This feature deals with the scenario where people forgot to close the issue and the milestones. Since the project and the presentations are over, it is safe to assume that people have forgot to close the milestones rather than the considering it as milestones that missed the due date. This feature is particularly important as it clearly shows that people have been working outside GitHub. The consequences of this problem could lead to redundant work, increased development time. This also proofs to an extent that the communication between the team members is not great. To perform this analysis, we iterated through the milestones CSV files and checked if the milestones are still open.

5.9 Distribution of Comments among Team Members

Comparing number of comments of each user in a team is a very good measure of how much each user is contributing. Ideally, all users should be commenting equally. But in many cases, one or two users may be dominating the conversations and others users not participating as much which is a bad smell.

Here, we simply looked at distribution of the number of comments among the 3 or 4 users in a team to determine how imbalanced the contribution is.

5.10 Number of comments on each issue

Number of comments on an issue might be important to collect as it represents the participation of team members on an issue. Here, the correlation is pretty straightforward; more comments means more participation.

So, we can simply compare the boxplots for all 6 teams and infer which teams had good participation in solving an issue and which teams didn't

5.11 Commit Progression

The git commit history of projects were collected so that we could identify when the commits were made, were there more commits just before the deadline, or were there commits on a regular basis. This helps us identify the planning and execution of the project. The coordination among the team members.

6. FEATURE DETECTION : RESULTS AND DISCUSSION

6.1 Issues with Long Life Time

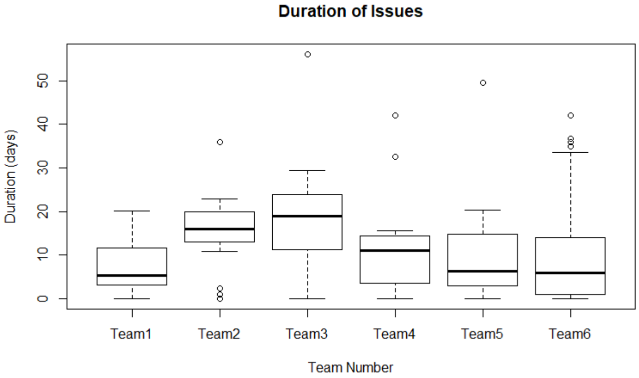


Figure 1: Boxplot of duration of issues for all 6 teams

Figure 1 shows boxplots of duration of issues for all 6 teams. This plot shows us that the central tendency and spread for this measure is not consistent among the 6 teams. Each team has their own methods of planning, estimating and assigning issues. So, it's better to look at this data team by team.

Team1:

Figure 2 is a plot for duration of issues for Team1. Here, everything before the red line is an approximately linear

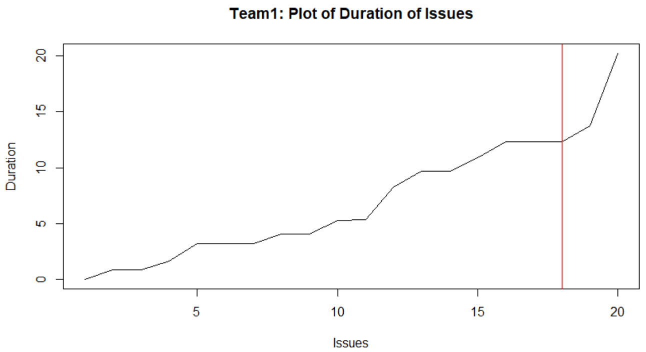


Figure 2: Team1: Duration sorted in ascending order

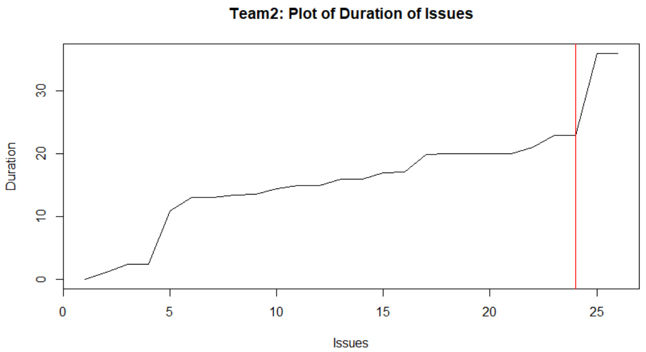


Figure 3: Team2: Duration sorted in ascending order

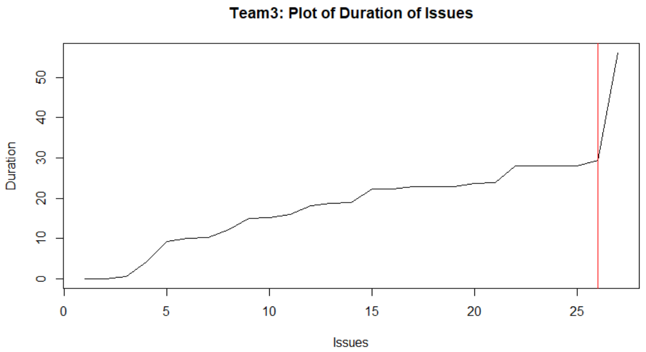


Figure 4: Team3: Duration sorted in ascending order

graph and then there is a sharp spike right after it. Hence, data before the line is more representative of team than the entire data. The mean and standard deviation of this chunk of data (before red line) is calculated and all points beyond (mean + 1.5*stdev) is considered abnormally long. There are 2 such points in this case.

Mean = 5.96 days
Std. Dev. = 4.29 days
No. of long issues = 2

Team2:
Mean = 14.46 days
Std. Dev. = 6.76 days
No. of long issues = 2

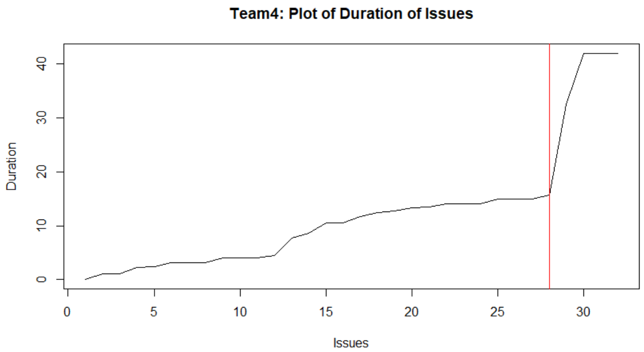


Figure 5: Team4: Duration sorted in ascending order

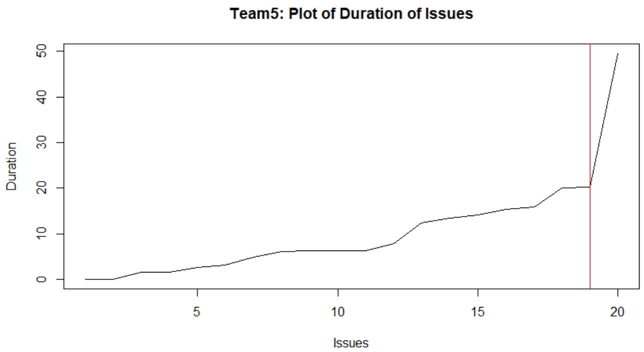


Figure 6: Team5: Duration sorted in ascending order

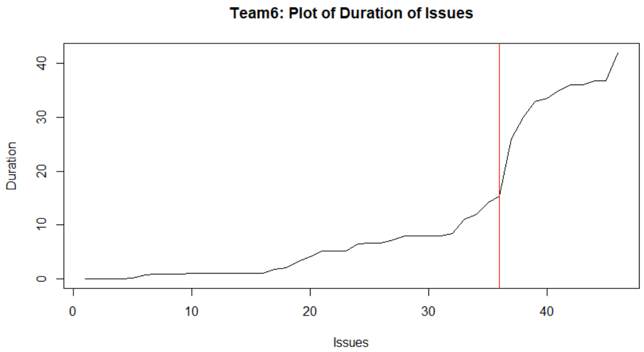


Figure 7: Team6: Duration sorted in ascending order

Team3:
Mean = 17.37 days
Std. Dev. = 9.15 days
No. of long issues = 1

Team4:
Mean = 8.43 days
Std. Dev. = 5.41 days
No. of long issues = 4

Team5:
Mean = 8.35 days
Std. Dev. = 6.6 days
No. of long issues = 3

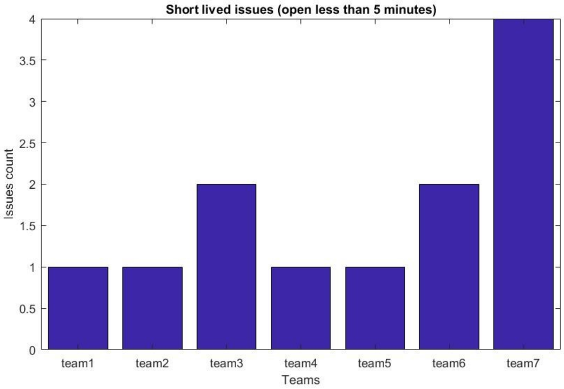


Figure 8: Bar graph showing number of short lived issues for each team

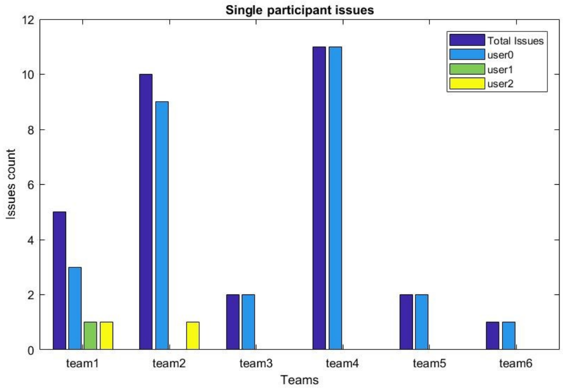


Figure 9: Bar graph showing distribution of single participant issues among all team members

Team6:
Mean = 4.39 days
Std. Dev. = 4.29 days
No. of long issues = 14

So for Teams 1,2,3,4,5 number of long issues are between 1 to 4. This means they were finishing most issues in a reasonable amount of time. But Team6 has 14 long issues which could be a potential bad smell.

6.2 Issues with Short Life Time

Figure 8 shows the issue counts that were open less than 5 minutes, making it issues with short life time. Team1, Team3, Team4 and Team5 have only one issue that was open less than 5 minutes, making their team very well coordinate though not perfect. Team 3 and 4 have two such issues making it more prone to problems compared to the first set discussed earlier. Though we did most of our analysis for only first 6 teams, low numbers on this particular bad smell category made us to analyse the 7th project data which showed 4 such issues making it more prone to communication problems and project failure.

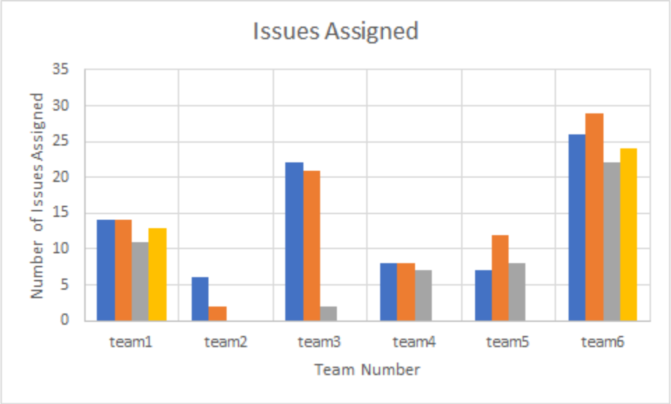


Figure 10: Bar graph showing distribution of assigned issues among team members

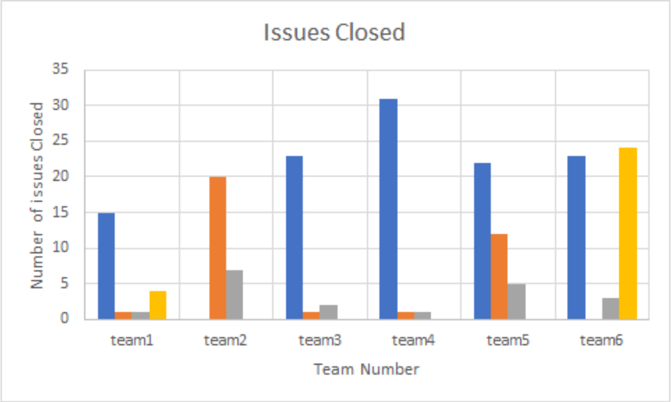


Figure 11: Bar graph showing distribution of closed issues among team members

6.3 Single Participant Issues

Figure 9 shows the bar graph plotting total identified single participant issues and the distribution among the team members who handled it. Team1, Team2, and team4 are the teams that have suffered highly compared to other teams. This portrays that the communication of issue is one way in these teams.

6.4 Number of Issues Assigned to each Team Member

Ideally the number of issues assigned to each team member must be equal, but that is not the reality among many teams which are represented in Figure 10. Each bar under a team represents the number of issues assigned to each team member. Here we can see that Team2 and Team3 have a very skewed distribution of issues assigned to each team member. This shows that the there is heavy workload on few members of the team whereas a lighter workload on other team members.

Team 1, Team 4, Team 5 and Team 6 have almost equal number of issues assigned to each team member. This is a good sign of workload sharing. Team 2 and Team 3 have unequal number of issues assigned to team members. This shows the lack of workload sharing.

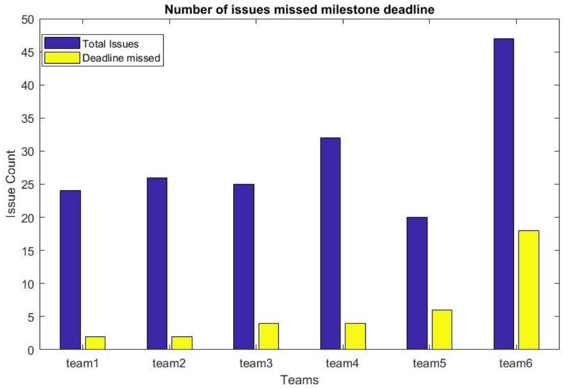


Figure 12: Bar graph of number of issues closed after due date

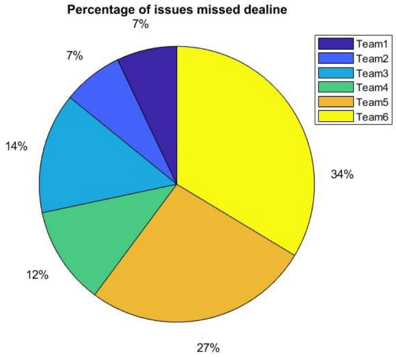


Figure 13: Pie chart showing distribution of issues closed after due date among 6 teams

6.5 Number of Issues Closed by each Team Member

Ideally the number of issues closed by each team member must be equal to the number of issues assigned to them. If not equal to the number of issues assigned, the number of issues closed must be about the same for all team members. But that is not the reality among many teams which are represented in Figure 11. In all the teams there is one user who was closed more number of issues than all other team members combined. This is a very bad trend. Which shows that one team member has taken up more responsibility than all other team members. Each bar under a team represents the number of issues closed by each team member.

6.6 Issues Closed after Milestone Due Date

Figure 12 shows the total number of issues and the number of issues that were not closed before the milestone deadline. The number of issues that weren't closed before milestone due date shows the level of poor planning and effort estimation error.

From Figure 12 and Figure 13, we can observe that both Team 1 and Team 2 haven't suffered much. Though they have a couple of issues that haven't met the deadline, the percentage of issues that didn't make the deadline is very small. Team3 and Team4 have suffered little more than Team1 and Team2. Team5 and Team6 are the teams with high percentage of issues that didn't make the deadline, making it unusual and prone to project failures.

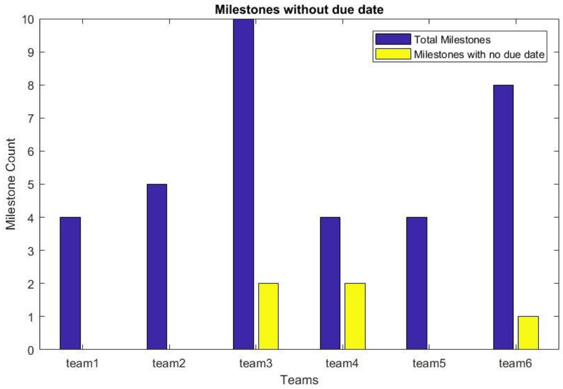


Figure 14: Bar graph showing number of milestones without due date

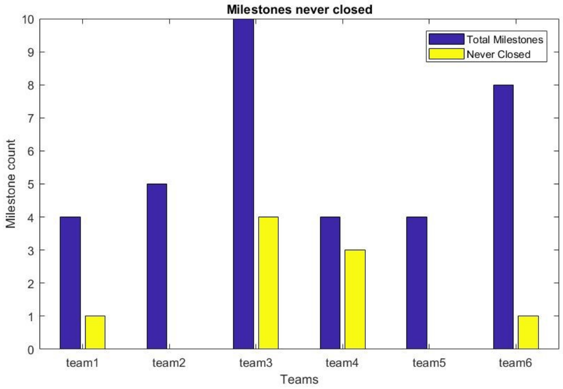


Figure 15: Bar graph showing number of milestones that were never closed

6.7 Milestones without a Due Date

Figure 14 shows the number of milestones with no due date and total milestones in each team. Clearly Team3, Team4 and Team6 haven't been using GitHub for collaboration efficiently. It looks like they have been working outside GitHub. Also, as the due date was not set, it is safe to assume that they haven't performed initial planning and effort estimation.

6.8 Milestones never closed

Figure 15 shows the number of milestones that were never closed. Though it is safe to consider that people have forgot to close the milestones, it is also worth believing that teams have not used GitHub exclusively and have worked outside the GitHub. Figure 4 shows team 4 and team5 are the teams that have high percentage of milestones never closed.

6.9 Distribution of Comments among Team Members

From the graph in Figure 16, we see that the volume of comments differs greatly among the six teams. Moreover, more comments does not necessarily mean more collaboration. The percentage of comments by each user in a team

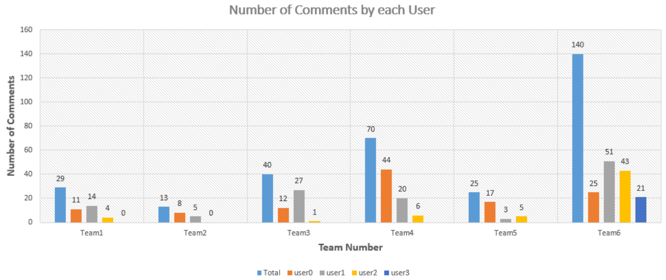


Figure 16: Bar graph showing number of comments by each user

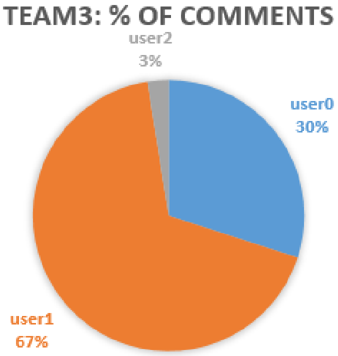


Figure 17: Pie Chart showing distribution of number of comments among users of Team3

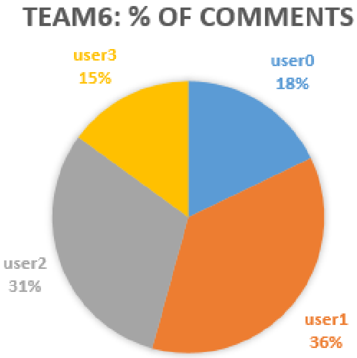


Figure 18: Pie Chart showing distribution of number of comments among users of Team3

might be more interesting to look at. This will give us an idea of how evenly or unevenly the comments are split among teammates.

In the pie chart in Figure 17, we can see that Team3 is an example of bad collaboration since the number of comments are unevenly divided among the users. It is mostly dominated by one user. Teams 1,2,4,5 also were similarly dominated by one user. Team6, depicted in Figure 18, shows relatively better collaboration than all other teams.

6.10 Number of comments on each issue

Figure 19 shows a boxplot of number of comments per issue for all six teams. Here, more comments per issue indicates more collaboration among teammates to solve issues.

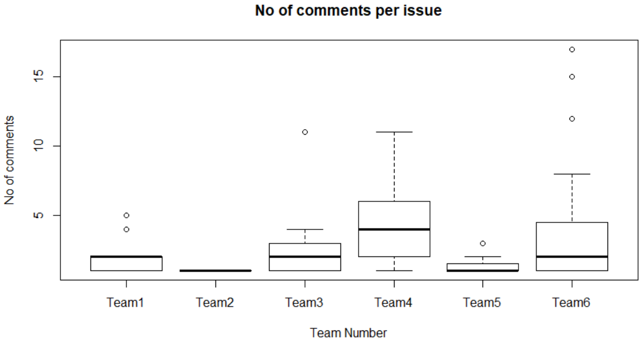


Figure 19: Boxplot of number of comments per issue for all 6 teams

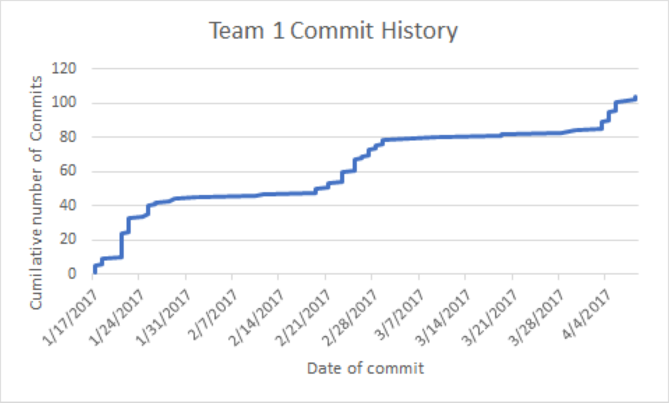


Figure 20: Team1: Commit Progression

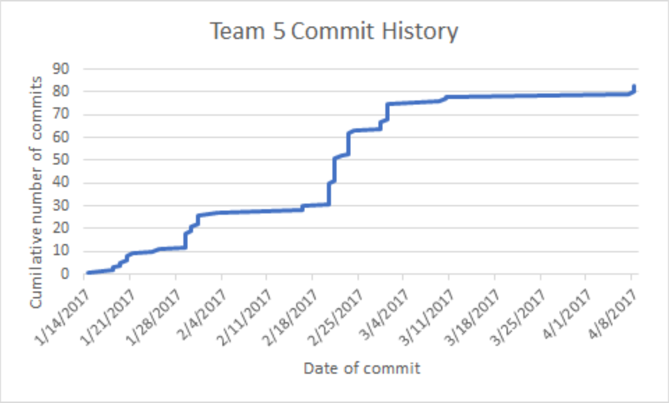


Figure 21: Team5: Commit Progression

Hence, Teams 4 and 6 are the best here while Team2 is really bad since it has only one comment per issue.

6.11 Commit Progression

Ideally, periodic commits represents good planning and balance of the work load. But in reality there are more commits just before the submission deadline, and fewer or no commits when the submission deadline is far away. This trend is an indicator of bad planning and execution.

Figure 20 and Figure 21 representing Team 1 and Team5 respectively have well spaced commits before each deadline. This shows they had planned and were able to successfully

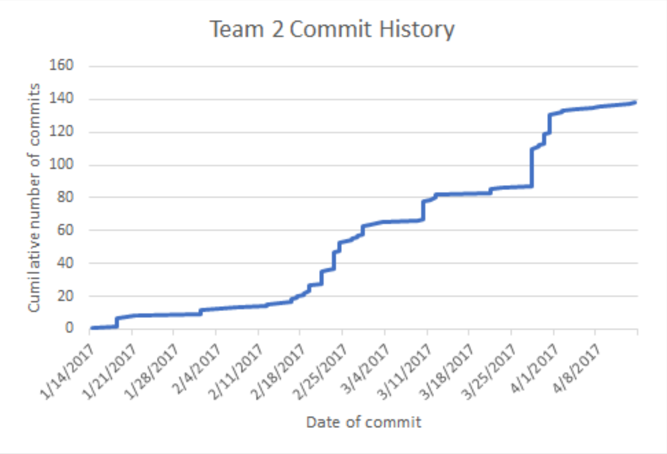


Figure 22: Team2: Commit Progression

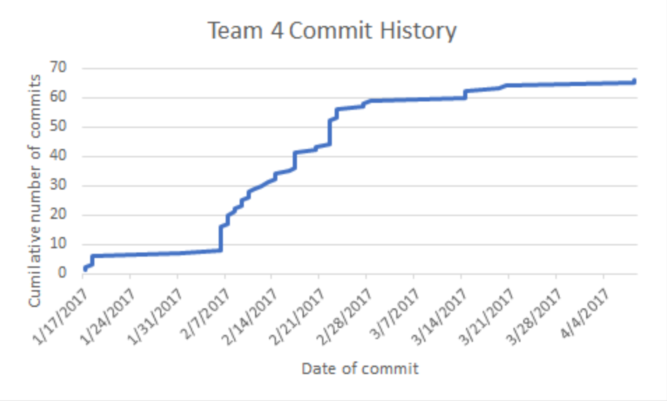


Figure 23: Team4: Commit Progression

complete before the deadline and then again had time to make changes and commit it before the deadline.

Figure 22 and Figure 23 representing Team 2 and Team 4 respectively had very few commits initially, this shows they had not started their work early, but later on as the project progressed they have made commits regularly before submission deadlines

Figure 24 representing Team 3 has a lot of activity initially. Seeing by the number of commits, it seems they had completed the project well before the all the deadlines. This either indicates they had worked under very high pressure initially and completed the tasks or their project was easy and were able to complete it faster than all other teams.

Figure 25 representing Team 6 have worked only for deadlines. Most of their commits are on the day of the submission deadline. This is a very bad practice. This shows lack of planning and execution. As the commits are on the day of the deadline, there is not sufficient time for other team members to evaluate the code of their peers.

7. BAD SMELL DETECTORS

Based on the features that we extracted, we manually assigned the importance of each feature with a numerical value ranging from 1 to 3, with 1 being less important and 3 being very important feature to detect bad smells. The assigned importance of each feature is shown in Figure 26. The cumulative score in each category of bad smell detectors will help to determine the extent to which project smells bad in the respective categories. It also can be used to measure the extent to which the project is poorly designed and managed.

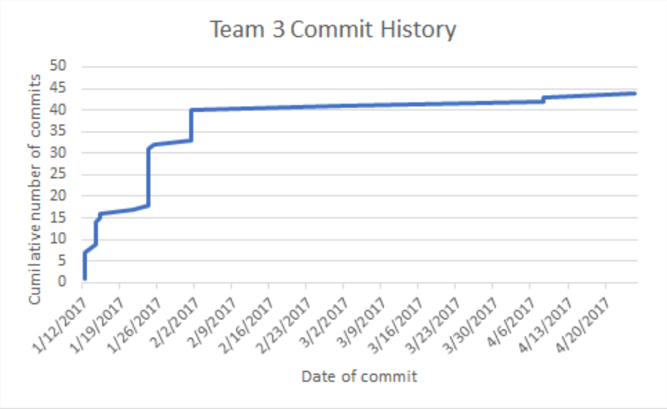


Figure 24: Team3: Commit Progression

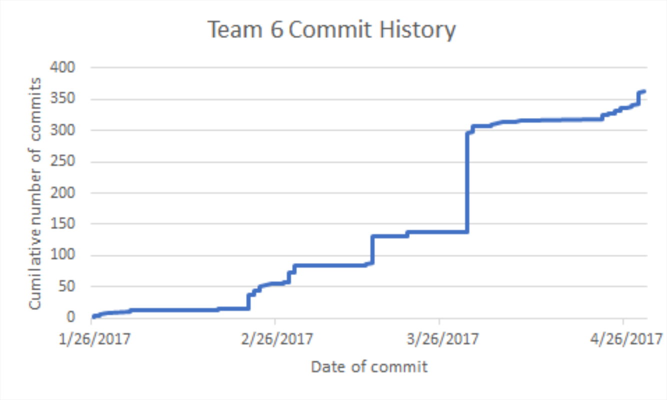


Figure 25: Team6: Commit Progression

All the tables in Figures 27, 28, 29, 30, 31 were populated based on the inference from the results section.

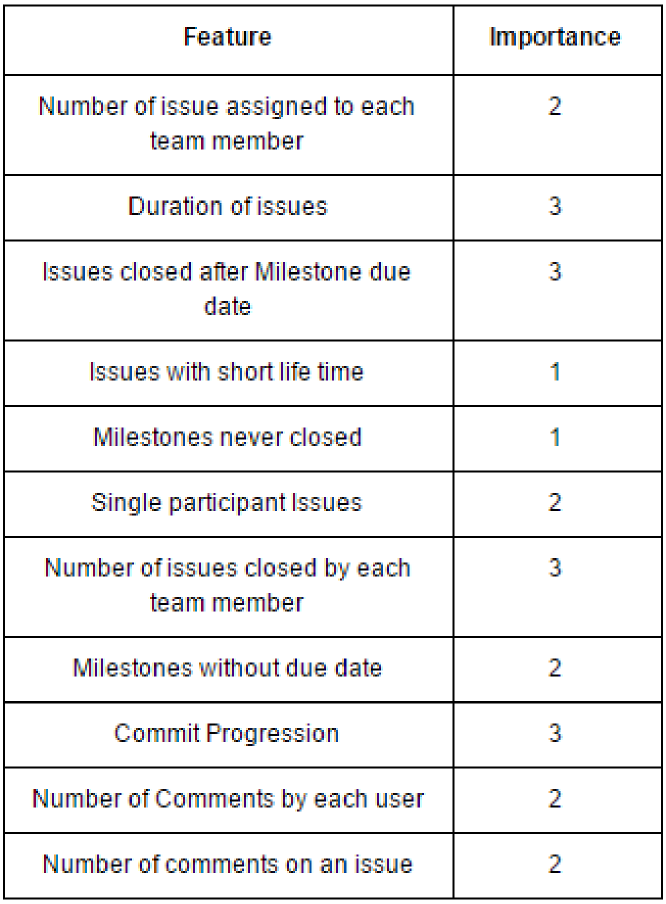
7.1 Poor Planning and Estimation

One of the main reasons for the project failure is poor planning and estimation. Milestones are used to divide the project into small sets of achievable targets. Thus milestones can be seen as the results of planning. The following features can be used to smell bad practices that are part of poor planning.

From the table in Figure 27, we can see that Team3 and Team6 haven't planned their project properly, earning 10 and 11 points respectively.

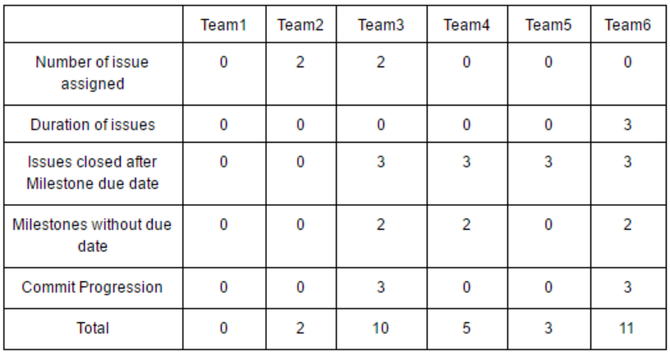
7.2 Imbalance in work division

Imbalance in work division could lead to lot of problems. One might have to handle a lot of work and some might be not doing any work. This in turn could lead to increased development time. As time is an important factor in performing estimation, the project could go off track and could possibly be scrapped.



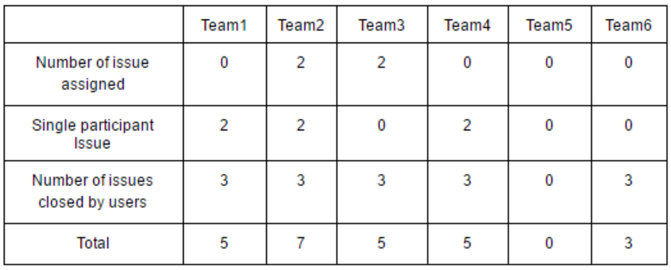
Feature	Importance
Number of issue assigned to each team member	2
Duration of issues	3
Issues closed after Milestone due date	3
Issues with short life time	1
Milestones never closed	1
Single participant Issues	2
Number of issues closed by each team member	3
Milestones without due date	2
Commit Progression	3
Number of Comments by each user	2
Number of comments on an issue	2

Figure 26: Table showing importance of each feature



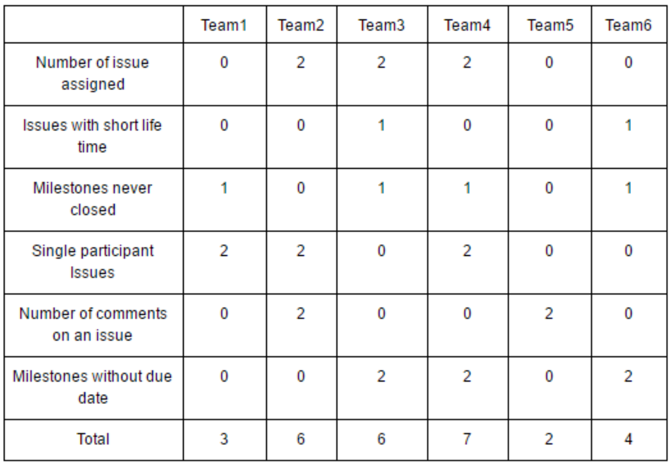
	Team1	Team2	Team3	Team4	Team5	Team6
Number of issue assigned	0	2	2	0	0	0
Duration of issues	0	0	0	0	0	3
Issues closed after Milestone due date	0	0	3	3	3	3
Milestones without due date	0	0	2	2	0	2
Commit Progression	0	0	3	0	0	3
Total	0	2	10	5	3	11

Figure 27: Table evaluating teams on Poor Planning and Estimation



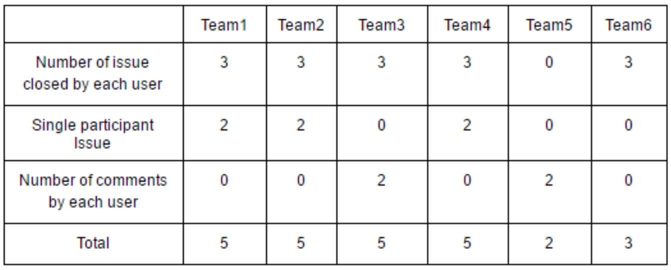
	Team1	Team2	Team3	Team4	Team5	Team6
Number of issue assigned	0	2	2	0	0	0
Single participant Issue	2	2	0	2	0	0
Number of issues closed by users	3	3	3	3	0	3
Total	5	7	5	5	0	3

Figure 28: Table evaluating teams on Imbalance in work division



	Team1	Team2	Team3	Team4	Team5	Team6
Number of issue assigned	0	2	2	2	0	0
Issues with short life time	0	0	1	0	0	1
Milestones never closed	1	0	1	1	0	1
Single participant Issues	2	2	0	2	0	0
Number of comments on an issue	0	2	0	0	2	0
Milestones without due date	0	0	2	2	0	2
Total	3	6	6	7	2	4

Figure 29: Table evaluating teams on Poor Communication



	Team1	Team2	Team3	Team4	Team5	Team6
Number of issue closed by each user	3	3	3	3	0	3
Single participant Issue	2	2	0	2	0	0
Number of comments by each user	0	0	2	0	2	0
Total	5	5	5	5	2	3

Figure 30: Table evaluating teams on Absent Team Members

From the table in Figure 28, we can come to the conclusion that Team1, Team2, Team3, and Team4 haven't divided the work equally compared to Team5 and Team6.

7.3 Poor Communication

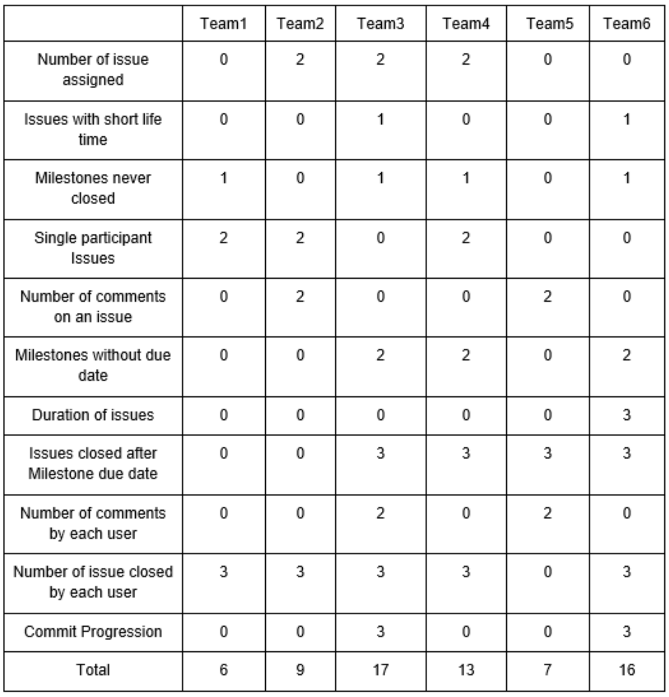
Communication between team members is very important for the success of the project. Poor communication could result in duplicate work, absence of ownership, increased development time, increased introduction of bugs and problems.

From the table in Figure 29, we can come to the conclusion that communication inside Team2, Team3 and Team4 were bad compared to other teams.

7.4 Absent Team Members

There are cases where certain team member will not take participate and will not contribute the success of the project. This can be identified by his/her inactivity in the GitHub logs. The problem associated with this are increased workload on other team members, missing deadlines, increased development time. When we consider a corporate environment, the absent team members can consume a lot of organization resources and not give back much to the organization. The inactivity could be the result of low skill levels or disinterested in contributing to the project success.

From the table in Figure 30, we can conclude that apart from Team5 and Team6, all other teams have suffered to an extent by absent group member or members



	Team1	Team2	Team3	Team4	Team5	Team6
Number of issue assigned	0	2	2	2	0	0
Issues with short life time	0	0	1	0	0	1
Milestones never closed	1	0	1	1	0	1
Single participant Issues	2	2	0	2	0	0
Number of comments on an issue	0	2	0	0	2	0
Milestones without due date	0	0	2	2	0	2
Duration of issues	0	0	0	0	0	3
Issues closed after Milestone due date	0	0	3	3	3	3
Number of comments by each user	0	0	2	0	2	0
Number of issue closed by each user	3	3	3	3	0	3
Commit Progression	0	0	3	0	0	3
Total	6	9	17	13	7	16

Figure 31: Table evaluating teams over all the features

7.5 Overall

The table Figure 31 shows the bad smell values with all the features together, portraying which teams have followed bad practices overall.

8. EARLY WARNING RESULTS

Detecting bad smells as early as possible is very important. Finding out the common early trends in projects that failed will help us correct those mistakes in future projects. Thus, improving the success rate of the future projects. Hence we have come up with a few early warning detectors which we think might be important in detecting the failure or "bad smell" of a project.

Parameter to identify the failure or "bad smell" of a project is percentage of issues missed by each team: This includes the issues that were closed after the due date or were not closed at all closed.

Parameter which is generated at an early stage of the project which might affect the above parameter is, Percentage of Milestones with no due date: Milestones are created at the early stage of the project. The 'measurable' mistake that can be done while creating milestones is not having due dates for the milestones.

We built a linear regression model to identify the relationship between Percentage of milestones with no due date (represents the activity in the early stage of the project) VS Percentage of issues missed by the team (represents the failure of the project) This is shown in Figure 32.

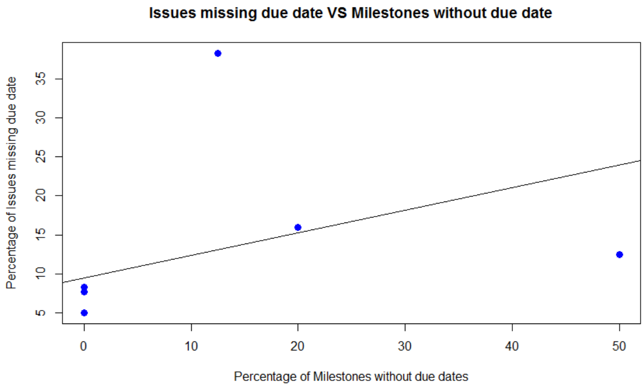


Figure 32: Linear Regression Model

8.1 Summary

Call:
lm(formula = x y)

Residuals:

1	-11.924
2	-11.379
3	-5.855
4	36.868
5	-10.96
6	-8.09

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	9.5124	14.5182	0.655	0.548
y	0.2896	0.7889	0.367	0.732

Residual standard error = 21.56 on 4 degrees of freedom
Multiple R-squared = 0.03258
Adjusted R-squared = -0.2093
F-statistic = 0.1347 on 1 and 4 DF
p-value = 0.7322

Due to lack of data we cannot prove much from the regression model, however we can say there is some correlation between the dependant and independant variable.

More data must be collected to arrive at a conclusion from this model because the rule of thumb for regression is there should be at least 30 observations for any conclusive results.