Bad Smells and Early Detection Analysis

Ahmad Saad Khan
North Carolina State University
Raleigh NC
akhan7@ncsu.edu

Krishna Agarwala
North Carolina State University
Raleigh NC
kagarwa@ncsu.edu

Nikhil Raina
North Carolina State University
Raleigh NC
nraina@ncsu.edu

ABSTRACT

This reports shows a comprehensive overview of 3 teams by looking at their Github repository and tracking their milestones, issues opened and closed as well as the average time taken to close these issues. We also looked at their commit history, their shared responsibility in terms of issues opened, closed as well as commented on. We also look at overall team performance in areas like number of team members commenting on issues and average comments per issue by the teams. By considering the raw data that Github dumped for us in with such diverse outlooks, we can look for *bad smells* in their project - the way they managed the time and how they shared the responsibility amongst the team members, their communication and planning skills and the overall team chemistry.

Keywords

Github; Bad smells; Issues; Milestones; Commits; Github tags

1. DATA COLLECTION

The very first and probably the most important task in this project was collecting data from the different teams. The entire project hinged on us making meaningful observations from the raw data we received from Github. We used the gitable.py[1][2] to help retrieve the data from Github of 3 teams. We then analysed the data to look for mistakes which were made during the past three months by the teams and any other bad smells we could find.

1.1 Our data collection

Running the gitable.py script gave us a massive amount of raw data from Github, which has a very robust way of organising their data for data analysis. We didn't diversify from the data too much as we felt Github gave us enough data for the purpose of detecting bad smells.

This is the data we collected from Github for each of the three teams:

- Issues
  - Issue ID
  - Issue Name
  - Creation time
  - Action taken
  - User associated with the action
  - Milestone associated with action
- Milestones
  - Milestone ID
  - Milestone title
  - Milestone description
  - Milestone created at
  - Milestone due at
  - Milestone closed at
  - User associated with milestone
- Commits
  - Commit ID
  - Commit time
  - User performing the commit
  - Commit message
- Comments
  - Issue ID associated with comment
  - User writing the comment
  - Comment creation time
  - Comment update time
  - Comment text

1.2 Data collection technique

Executing the python script gave us an enormous amount of data which we needed to organise for sharing purposes as well as to make some observations. We used a mix of Microsoft Excel as well as SQL Lite 3 for organizing the data into easily readable tables. The steps of organizing the data were as follows:

1. Execute the gitable.py script to obtain the data from Github on our terminals.
2. Dump the data into SQL Lite 3 database and organize the tables.
3. Create a .csv formatted file from the database output.
   - Use the file for making pie-charts and histograms for a visually appealing representation of the data
   - Use the file to query for various rubrics and information gathering.
4. Take the help of both the charts and information obtained to identify bad smells in the project.

2. RAW DATA

The total data collected from 3 teams can be summed up in the table below. Using this data, we can draw conclusions and detect bad smells.

Issues	Milestones	Commits	Comments
224	42	208	189

3. FEATURE SET

We based our observations on a total of 9 different criteria. The criteria encompasses all the different data we collected from Github. A comparison of the 3 teams is done with the data and also a comparison between team members where data is related to them is performed. We have arranged the data in the form of pie-charts and histograms so the results can be easily viewed and conclusions can be easily drawn.

3.1 Average issue lifetime

Average issue lifetime refers to the average time difference between an issue being opened by a team and then being closed. A large average issue lifetime can be indicative of improper planning and trying to work on large parts of code instead of dividing it into small, easily manageable parts. The more agile a team is, the shorter should be the average issue lifetime.

The SQL query written for finding the average issue lifetime is as follows:

```
Select avg(z.a) from
(Select (b.y - a.x) "a", a.issueID from (SELECT min(time) "x", issueID FROM event where
issueID IN (select issueID from event where action = "closed") group by issueID) a,
(SELECT time "y", issueID FROM event where issueID IN (select issueID from event where
action = "closed") and action = "closed") b
where a.issueID = b.issueID ) z
```

3.2 Hanging issues

Hanging issues refer to issues which were created but not assigned to any particular milestone. Abundance of hanging issues would indicate that the team did not plan their work properly and did not think too much about possible problems creeping up in their projects or what issues each milestone could face. These issues seemed to be there *just for the heck of it.*

The SQL query written for finding the hanging issues is as follows:

```
Select count( distinct(a.id)) from issue a, event b where a.id = b.issueID and b.milestone is NULL
```

3.3 Milestones closed after due date

Milestones closed after due date indicates those milestones which got closed past their due date. Large amounts of these overdue milestones means teams underestimating the time required in finishing off their work.

The SQL query written for finding the milestones closed after due date is as follows:

```
Select count(*), (Select count(*) from milestone)b from milestone where closed_at > due_at
```

3.4 Issues closed after milestone due date

Issues closed after milestone due date are those issues which were tagged to a particular milestone but did not get closed by the time the milestone was supposed to be closed. This means either the milestone got delayed in closing or the issue was either delayed or not closed at all.

The SQL query written for finding the issues closed after milestone due date is as follows:

```
Select count(*) from event a, milestone b where a.milestone = b.id and a.action = "closed" and
a.time > b.due_at
```

3.5 Issues closed before milestone due date

Issues closed before milestone due date are those issues which were tagged to a particular milestone and also got finished before the milestone was due to be closed. This data indicates how well the team planned and executed their work and can be viewed as a percentage of actual closed issues vs. the total issues.

The SQL query written for finding the issues closed before milestone due date is as follows:

```
Select count(distinct(a.issueID)), (Select count(distinct(issueID)) from event where issueID IN
(Select issueID from event where action = "closed")) from event a, milestone b where
a.milestone = b.id and a.action = "closed" and a.time < b.due_at
```

3.6 Issue distribution

Issue distribution refers to the number of issues assigned to different team members. It can indicate how the total workload is shared amongst the team members and can be indicative of how active each team member was in tackling the issues.

The SQL query written for finding the issue distribution is as follows:

```
Select count(distinct(issueID)), user from event where  action = "assigned" group by user
```

3.7 Issue creation distribution

Issue creation distribution refers to the number of issues raised by each group member per team. This is indicative of how much each member is thinking and anticipating about their projects and what issues can crop up in the future. Though not directly linked, this parameter is important in that it can indicative of how "seriously" each team member is taking the project.

The SQL query written for finding the issue creation distribution is as follows:

```
Select count(*)  from (Select min(time) "mintime",issueID from event group by issueID )a, event
b where a.issueID = b.issueID and a.mintime = b.time
```

3.8 Issues not labeled

Issues not labeled refers to those issues which have been raised but have not been tagged with any label. These issues are difficult to understand in terms of why they were raised or which aspect of the project they address. Issues should generally have at least one label attached to them for better understanding of the label.

The SQL query written for finding the issues not labeled is as follows:

```
Select (Select count(distinct(issueID)) from event)count(distinct(issueID)), (Select
count(distinct(issueID)) from event) from event where action = "labeled"
```

3.9 Issues closing distribution

Issues closing distribution refers to how many issues were closed by each member of the team. This can be indicative of how much "work" each team member is doing but this is not the only criteria to judge the team effort distribution.

The SQL query written for finding the issues closing distribution is as follows:

```
Select count(distinct(issueID)), user from event where action = "closed" group by user"22"
```

3.10 User commit distribution

User commit distribution refers to the commit history of each user and can be indicative of how much effort each member is putting into the project. A balanced commit distribution implies a more balanced team as compared to a skewed distribution.

There is no SQL query for this as this was dumped by default as a part of the Github data.

3.11 Number of people commenting on issues

The number of people commenting on issues refers to how many team members commented on each of the issues which they raised. This is indicative of the communication skills of the team and the level of interaction which happened between team members during the project development phase. Higher numbers indicate good interaction and communication between team members whereas lower numbers would indicate poor interaction.

The SQL query written for finding the number of people commenting on issues is as follows:

```
Select count(*),a.num_users from (select issueID, count(distinct user) "num_users" from
comment group by issueID) a group by a.num_users
```

3.12 Average number of comments per issue

The average number of comments per issue issue is the average number of comments per team on a single issue. This was found by dividing the number of issues raised by the number of comments on the issues for each team. This is indicative of the level of interaction between team members on issues and also indicates the timeliness with which the members are trying to close down the issues.

The SQL query written for finding the average number of comments per issue per group is as follows:

```
Select avg(summ) from (Select sum(a) "summ", id from (Select id, (case issueID when id THEN
1 ELSE 0 END) a from issue Left OUTER JOIN comment on  issue.id = comment.issueid ) x
group by id)
```

4. RESULTS

The results of all the queries written above are displayed in the form of visual aids like graphs, pie-charts and histograms. The 3 teams are represented by team 1, team 2 and team 3, and their members will be mentioned as member 1 through member 4.

4.1 Average issue lifetime

We've shown the average issue lifetime in figure 1. It's in the form of a bar graph comparing all 3 teams and the average lifetime their issues remain open in hours.

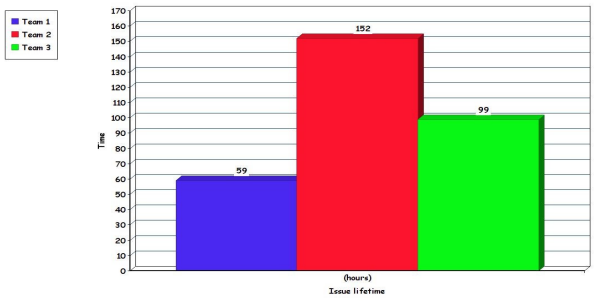

Figure 1: Average issue lifetime

4.2 Hanging issues

Figure 2 shows the number of issues created by each team without assigning any particular milestone to them. It's in the form of a bar graph comparing all 3 teams.

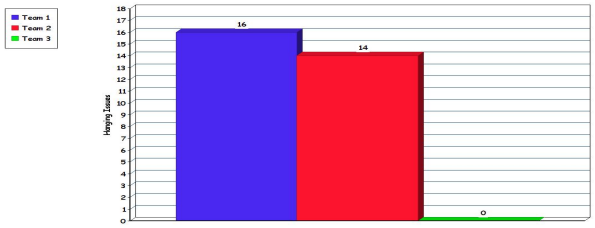

Figure 2: Hanging issues

4.3 Milestones closed after due date

Figure 3 features a stacked bar graph displaying the milestones which were closed by the teams after their due date had passed in comparison with the number of total milestones created by the teams.

Figure 3: Milestones closed after due date

4.4 Issues closed after milestone due date

Figure 4 features a stacked bar graph displaying the issues which were closed by the teams after their corresponding milestone's due date had passed in comparison with the number of total issues created by the teams.

Figure 4: Issues closed after milestone due date

4.5 Issues closed before milestone due date

Figure 5 features a stacked bar graph displaying the issues which were closed by the teams before their corresponding milestone's due date had passed in comparison with the number of total issues created by the teams.

Figure 5: Issues closed before milestone due date

4.6 Issue distribution

Figure 6 shows a pie chart comparing the issues which were assigned amongst the team members. The three teams are compared so we can see how the distribution of issues is done by each of the teams.

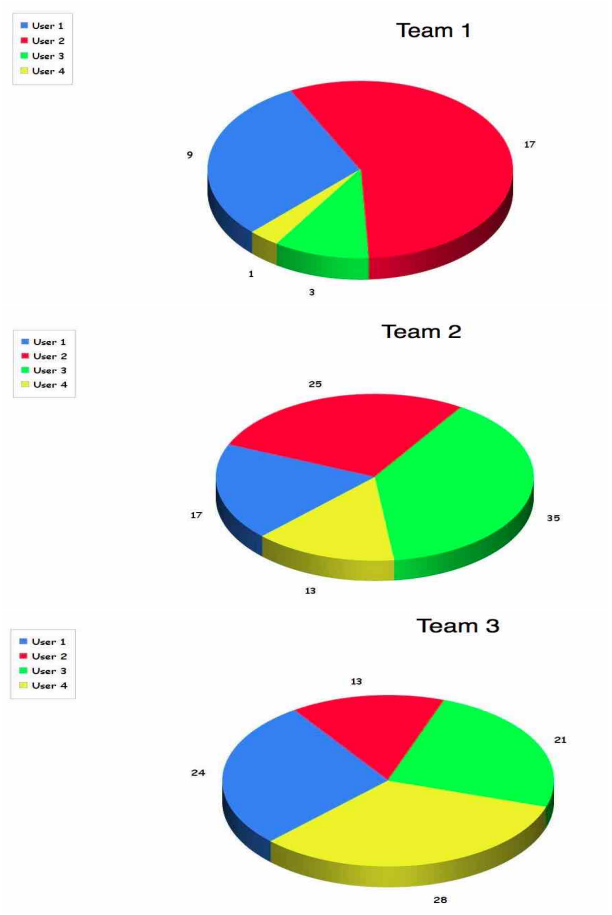

Figure 6: Issue distribution among team members

4.7 Issue creation distribution

Figure 7 shows a pie chart comparing the issues being created by team members. The three teams are compared so we can see how the creation distribution of issues is done by each of the teams.

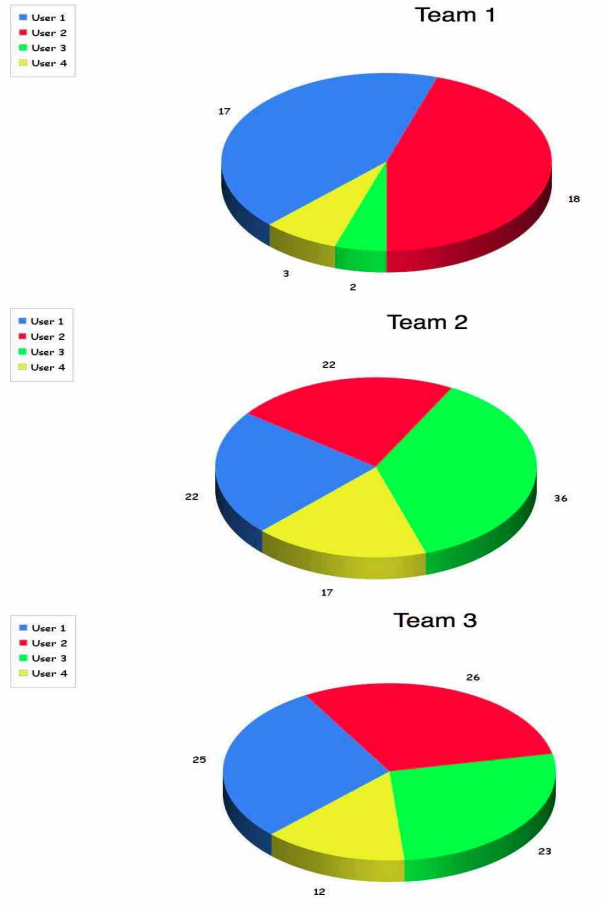

Figure 7: Issue creation distribution among team members

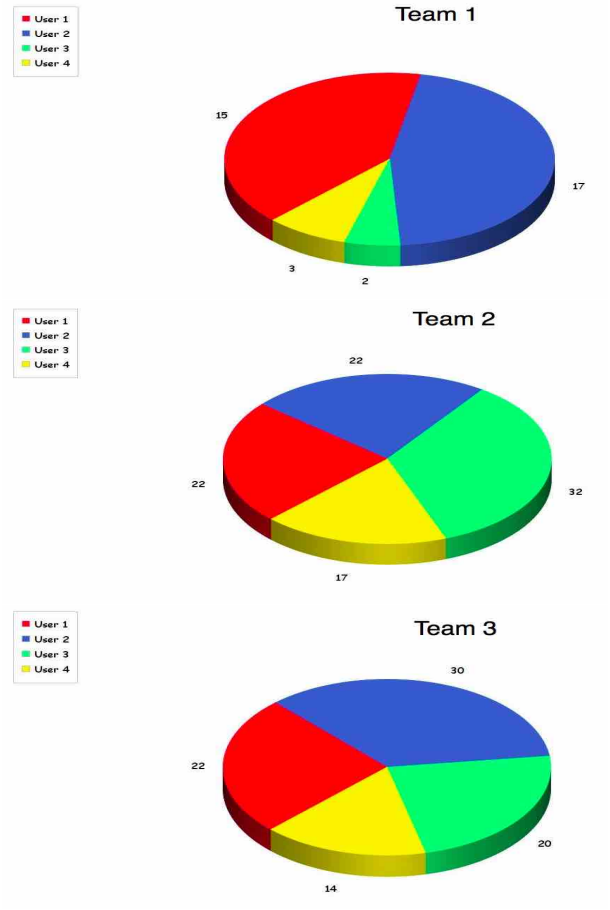

Figure 9: Issue closing distribution among team members

4.8 Issues not labeled

The issues which have been created by the teams but no labels have been assigned to them are shown in figure 8. It shows the number of issues without any labels compared with the total number of issues.

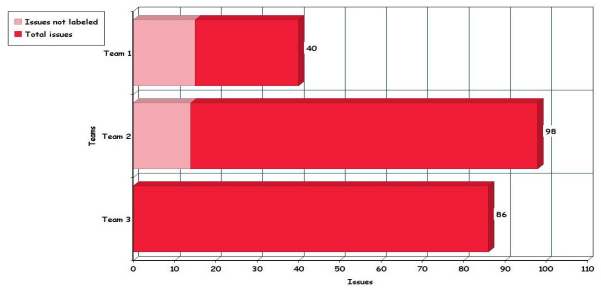

Figure 8: Issues not labeled

4.9 Issue closing distribution

Figure 9 shows a pie chart comparing the issues being closed by team members. The three teams are compared so we can see how the closing distribution of issues is done by each of the teams.

4.10 User commit distribution

Figure 10 shows a pie chart comparing the number of commits being performed by each team member. The 3 teams are compared so the total number of commits and the commit distribution can be seen.

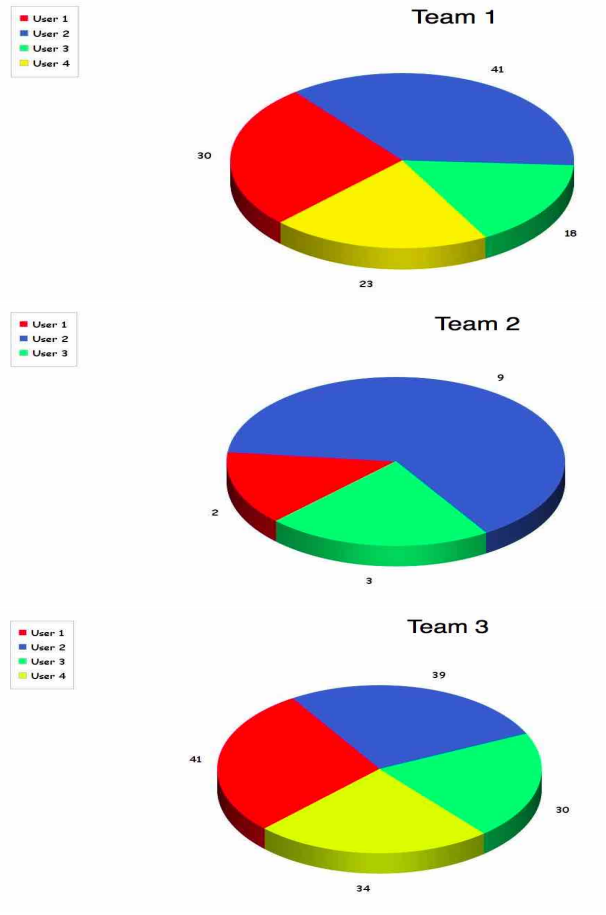

Figure 10: User commit distribution

4.11 Number of people commenting on issues

Figure 11 features a stacked bar graph displaying either 1, 2 or 3 people who commented on issues. The 3 teams can be compared in terms of how many members are commenting on issues.

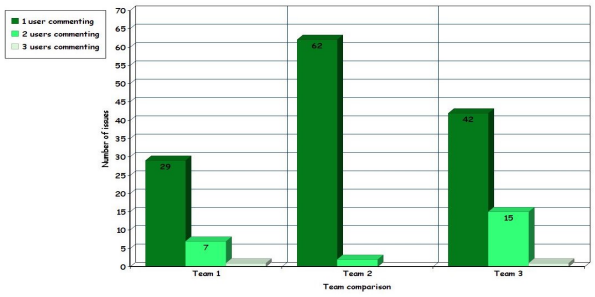

Figure 11: Number of people commenting on issues

4.12 Average number of comments per issue

Figure 12 features a bar graph displaying the average number of comments per single issue and the 3 teams are compared so we can see how many comments on an average the teams do for their issues.

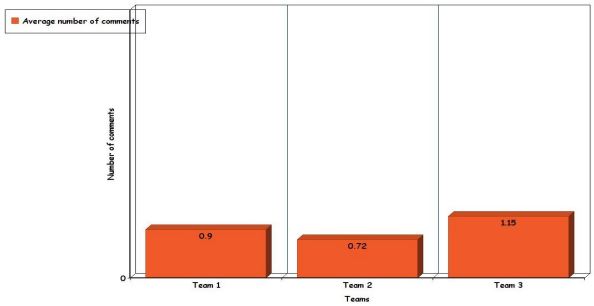

Figure 12: Average number of comments per issue

5. BAD SMELLS

We use the results from the above section to carefully identify a few bad smells. We categorize the bad smells into three major categories - teamwork, planning, execution.

5.1 Teamwork

Teamwork within the team is an extremely vital component. It enables the team to deliver the project on time, and also help a lot in conflict resolution as well. We measure teamwork in terms of communication within the team and the work load distribution. A few of the features we considered were:

- Issue Distribution
- Issue Creation Distribution
- User Commit Distribution
- Number of People Commenting on an Issue
- Average Number of Comments per Issue

As per our analysis the issue distribution has to be even so that all the members of team are equally responsible for the work. As we can see from Figure 6, Team 3 has done a better job as compared to others.

Issue creation distribution helps in pointing out the active members in the team who are involved in peer review and helps us identify the slackers. As seen from the Figure 7, Team 2 and Team 3 did a good job, but Team 1 had only 2 members contributing. This means that other 2 people were not as involved in the process.

User commit distribution gives us a sneak peak into the workload distribution in the team. If there is a huge difference in the number of commits between the team members it means that either the workload is not distributed properly or a few of the team members are compensating for others work. As seen in the Figure 10, Team 3 has a better spread as compared to others, indicating all the members equally pitching in.

Number of people commenting on a issue is probably a true reflection of the communication within a team. The higher the number the better. According to Figure 11, Team 3 has much balanced approach in this manner and it showcased the best communication amongst the three teams.

Average number of comments per issue, helps in determining the quality of the communication. It represents how willingly the team members are sharing the knowledge with each other. It also refers to how frequently an issue is updated. Figure 12 shows Team 3 has done a better job than the other two teams.

5.2 Planning

By planning we refer to how agile the teams were while doing their projects. We tracked how well they planned their tasks and if they were able to finish their tasks on time. We do this by measuring the following features:

- Hanging Issues
- Milestones Closed After Due Date
- Issue Closed After Milestone Due Date
- Issue Closed Before Milestone Due Date
- Issue Not Labelled
- User Commit Distribution

By taking the number of hanging issues into consideration we tried to figure out how the team members planned their tasks. If the issues were not assigned to any Milestones, it became difficult to track them and this was a problem. Hence, lesser this number the better. From Figure 2, we can say that Team 3 has 0 hanging issues, a perfect score.

We take the number of milestones and issues closed before and after the due date as an indicator of how well the team understood the task and how appropriately they planned for it. Delayed closing of issues and milestone can mean that team under planned for the task although it can also imply inefficiency of the team members. Early closing of issues can also lead to a problem since it means that the teams over planned their work and are now sitting idle drinking coffee. We can refer to Figures 3, 4, 5 to see the whole picture and it is clear that overall, Team 3 did a better job than others.

Issues not being labeled with appropriate labels also reflects the poor understanding of the task at hand by the team members. This can be a problem in huge projects and can lead to major issues. Figure 8 shows that Team 3 had almost zero issues that were not labelled from a total of 86 issues.

User commit distribution gives us the workload distribution. As already discussed this can also mean slacking off from work but can also be a result of poor planning. During planning phase a situation can occur that the team overburdens a single team member with a majority of the work while other are not doing their share. We can see from Figure 10, Team 3 has the most balanced approach.

5.3 Execution

Execution refers to how well the team executed the plan. We measure execution by referring to the following features:

- Average Issue Lifetime
- Milestone Closed After Due Date
- Issue Closed After Milestone Due Date
- User Commit Distribution
- Average Number of Comments per Issue

Average issue lifetime shows how quickly the team responded to reported issue and fixed them. The shorter the time the better. Figure 1 shows that Team 2 had the most long running issues. Team 1 had the best result according to this rubric with an average of only 59 hours per issue.

Milestones and issues being closed after the due date implies that the person responsible for the task was not able to process it in time and it stretched well past due date. Figures 3 4 show that Team 3 performed the best, with least number of issues/milestones open after due date.

By measuring the user commit distribution we can measure the work done by each team member. The more flattened the distribution the better the execution of the plan. According to Figure 10, Team 3 has the most balanced distribution and hence performed better than the other teams.

Average number of comments per issue indicates how much effort was spent by the team members to resolve an issue, lower the number the better. This is because a higher number means each time someone tried to fix an issue, they introduced newer bugs. This directly reflects on the code quality of the application. Figure 12 shows that both the Teams 1 2 performed less than one comment per issue implying they closed a few issues without a single comment. This indicates the lack of awareness of a few team members about a number of issues. Considering this factor, we can say that Team 3 performed better than the other teams.

6. EARLY DETECTOR

To detect bad smells ahead of time, we looked at two key factors namely - how users worked and behaved over time and how the project materialised over time.

For this we sorted the data in SQL tables chronologically as suggested by the professor. The time stamps in the SQL database has been saved as seconds beyond the epoch time '1/1/1970'. So we converted the time to the week number in 2016 in SQLite3 using the command strftime('%W', date(table.time, 'unixepoch')). We followed 2 rubrics to detect bad smells ahead of time.

1. Commit distribution over development time.
2. Issues and Milestones closed after due date distribution over time.

6.1 Commit distribution over development time

We checked both the group commit as whole over the period as well as user commits per group over the period to establish early bad smell.

As most of the development work was done in February and teams needed to be agile with respect to their issues

in March, we hypothesize that a team would finish their projects in a better fashion if the pattern of commits are evenly distributed over this period. If the pattern is close to even, it would tell us that the teams worked continuously and made a smooth progress. However if the pattern shows significant spikes, it would tell us that the respective project was not planned properly. Another point to note here is that the two major deadlines were March 1st and April 7th which correspond to Week 9 and 14 respectively and if the spikes are very close to these weeks, it can be understood that the teams worked in panic to meet the deadline which can in turn lead to the quality of work suffering.

To conclude, if the number of commits in the early part of February and early part of March are not significant enough, we can detect that the project may not be of supreme quality.

Note - If the team still pulled off a superb job, they need to be given a pat on the back.

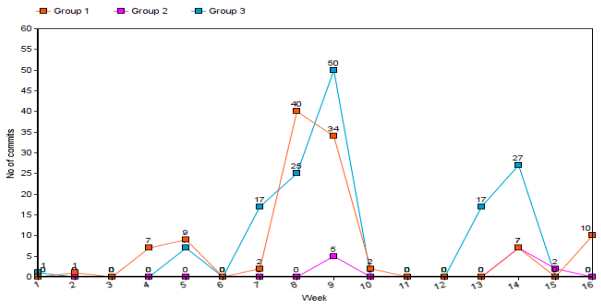

Figure 13: Commit distribution for 3 teams

The figure 13 shows that team 1 and 3 did majority of the work in Week 8, 9 and 14. The early part of February and March do not have a lot of commits which is an early sign of a bad smell.

Disclaimer - The commit data for Team 2 has an issue. It is sparse with respect to the data that we see at Github. However as we checked, most of the commits of this team were deletion commits directly from Github. We were not sure how to handle this situation and we have analyzed the data according to the result that we have.

The commits distribution per user per group over time can tell us if the entire team was productive throughout the development period. If these graphs are somewhat balanced, we can safely assume that the teams must have sailed smoothly and braced the storms together which helps in delivering a quality project. However if the graph is not balanced, we can say that the teams suffered from communication issues, poor planning and work distribution. Unequal team member contributions may also lead to loosing sight of a lot of goals that were discussed at the planning stage. So if the graph shows unbalanced team contributions, we can hypothesize that the teams would have suffered with the above mentioned issues.

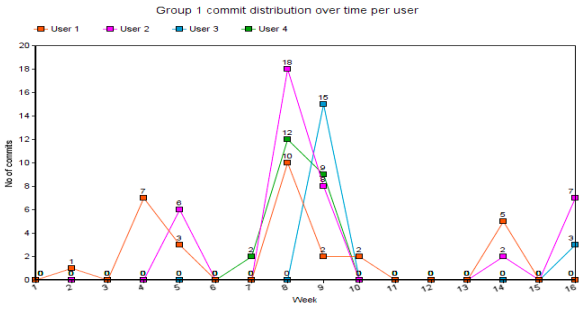

Figure 14: Commit distribution for Team 1

The figure 14 shows the distribution for the users of Team 1. It shows that user 2, 3 and 4 are somewhat balanced with their work in Week 8 and 9 and user 1 contribution was low.

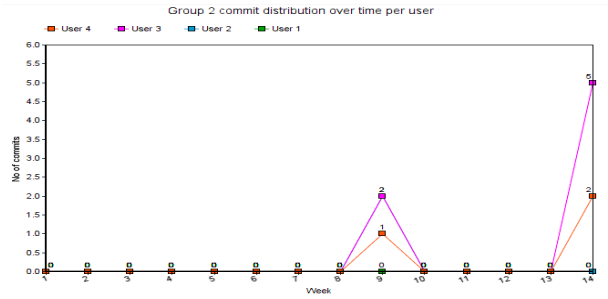

Figure 15: Commit distribution for Team 2

The figure 15 shows the distribution for the users of Team 2. It shows that user 3 has spiked while user 2 and 4 are balanced. User 1 has no contributions at all.

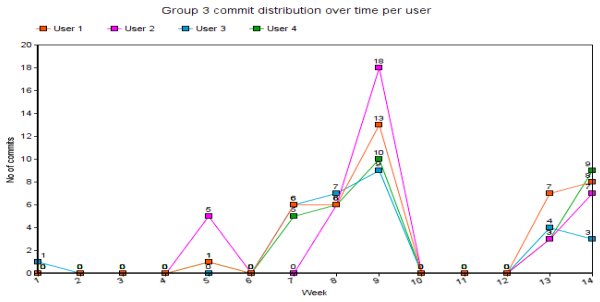

Figure 16: Commit distribution for Team 3

The figure 16 shows the distribution for the users of Team 3. Though user 2 has spiked in week 9, the user commits distributions are well balanced over the rest of the important periods of time.

6.2 Failed Targets distribution over time

In this rubric, we checked the distribution of the number of Issues and Milestones which were closed after the respective due date and have plotted a graph on this total value for the three teams over the weeks. What we want to say here is

that failed targets of a given period of time build up on top of the workload of the targets of the coming weeks. These continue to add up and have a cumulative effect and end up as a major point of worry. This was a pain point for our own project. Important weeks to note are 6, 7, 8, 9 and 12, 13, 14 as these were the major weeks where development and bug fixes were to be performed. The distribution can show us the count of failed targets over weeks and these count can tell us early on how badly will the project be affected as a result of these backlogs.

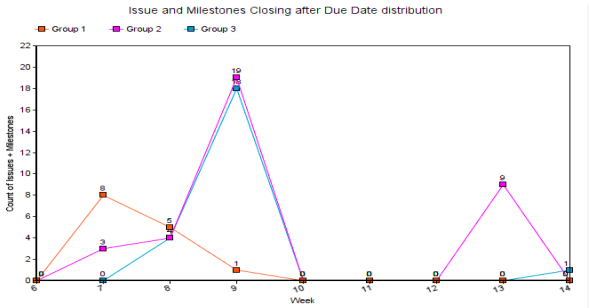

Figure 17: Failed Target Distribution over time for the 3 teams

The figure 17 shows that failed targets started at week 7 and 8 and ended up cumulating hugely for Team 2 and 3. They also started for Team 1 in Week 7 but the team was successful in managing the count and ended up in a reducing fashion which was commendable. As weeks 12-13-14 were mostly a bug-fixing period, so we don't see a lot of results there.

7. REFERENCES

[1] Code used to fetch data from GitHub: *https://github.com/CSC510-2015-Axitron/project2/blob/master/gitable-sql.py.*

[2] Code provided by professor to fetch the data from GitHub: *https://gist.github.com/timm/a87fff1d8f0210372f26.*