# Bad Smell Detection

**Rushi Bhatt**
North Carolina State Univeristy
`rbhatt@ncsu.edu`

**Devika Desai**
North Carolina State Univeristy
`dndesai@ncsu.edu`

**Li Shi**
North Carolina State Univeristy
`lshi7@ncsu.edu`

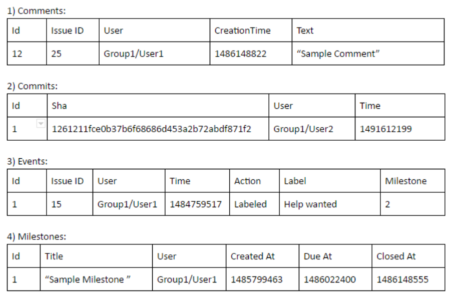
**Chen Zhao**
North Carolina State Univeristy
`czhao13@ncsu.edu`

May 3, 2017

## 1. Data Collection & Anonymization

### 1.1 Data Collection

We used the script gitable-sql.py to grab data of three repositories (two other repositories with lots of issues along with ours). As initial gitable-sql.py was not sufficient to get all the required data, we made some changes to fetch more relevant data that will be used in analysis. The script used github API to fetch four tables containing comment, commits, event and milestone which are stored in a SQLite database. We used SQL as well as Excel functions for grouping the data of each feature. Some of the complicated data has been gathered manually from individual github repositories. Below is the sample data stored in each table:



1) Comments:

| Id | Issue ID | User | CreationTime | Text |
|---|---|---|---|---|
| 12 | 25 | Group1/User1 | 1486148822 | “Sample Comment” |

2) Commits:

| Id | Sha | User | Time |
|---|---|---|---|
| 1 | 1261211fce0b37b6f68686d453a2b72abdf871f2 | Group1/User2 | 1491612199 |

3) Events:

| Id | Issue ID | User | Time | Action | Label | Milestone |
|---|---|---|---|---|---|---|
| 1 | 15 | Group1/User1 | 1484759517 | Labeled | Help wanted | 2 |

4) Milestones:

| Id | Title | User | Created At | Due At | Closed At |
|---|---|---|---|---|---|
| 1 | “Sample Milestone ” | Group1/User1 | 1485799463 | 1486022400 | 1486148555 |

### 1.2 Data Anonymization

first thing we did after getting the data is that we looked for all identifying symbols and then replaced them with randomly assigned group and person number. So all the groups are renamed to Group1,Group2,Group3 and every person of each individual group is renamed to User1, User2..and so on. So Person1 of group 1 is recognized as Group1/User1.

## 2.Feature Detection

### 2.1 Commit Timeline

Commit timeline shows how often and regularly (or irregularly) the group has made various commits on the github. Generally groups have different start and end dates of their commit history and therefore to get the clear timeline, we divided their commit duration in equal number of timeslabs along with using the fixed timeline. Once we get the interval period for each group, we made time slabs and counted the number of commits done by each group in each particular time slab.

Below is the equation to find time slabs:
Interval duration: (TimeStamp of Last commit - TimeStamp of First commit) / Number of weeks TimeSlab[i] = TimeStamp of First Commit + ( i * Interval duration )

We took into consideration both the actual number of commits for the fixed timeline and commits made within the timeslab. Both mostly follow the same trend for each individual group.

### 2.2 User Activity

User activity shows how active an individual group member has been in the project duration. We gathered these data in terms of events (Milestones and issues), comments and commits.

Commits:

Select user,COUNT(user) from commits group by user;

Events:

Select user,COUNT(user) from event group by user;

Comments:

Select user,COUNT(user) from comment group by user;

## 2.3 Participating Users per Issue

We want to explore the degree of user's' activity. We find out the number of users involved in each issue. We define "involved" by at least one action from the user related with that issue.

Select count(distinct user) from event group by issueID;

## 2.4 Number of Assigned Issues per User

With this feature, we want to know the workload distribution among users. As a good balanced team, all the members will share similar amount of workload. In the other words, we expect each user has equal number of issues assigned.

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

## 2.5 Issues without Milestones

As a well-structured project, people should set many milestones to divide their work. The issues should be assigned a "milestone" label to better locate the problem. We find out the percentage of issues without a milestone to display this feature.

Select count(distinct issueID) from event where milestone is NULL;

## 2.6 Long Time of Issues

When people create issues, it is a good thing that issues could be solved as soon as possible. In other words, long opening issues will bring uncertainty to the project. As the time goes on, it grows more difficult for users to identify their initial problems. We try to find out those issues that closed in a very short time as well as issues closed after a long time to determine the efficiency of the teamwork. We used SQL to fetch the time interval of each issue in seconds. Then we transformed the unit to hours and calculate the median and standard deviation. At last we draw the normal distribution of the data.

Select max(time)-min(time) from event group by issueID;

## 2.7 Number of Comments per Issue

This is the total number of comments of each issue, we want to know how team members react to a issue posted by others by this attribute. And by knowing this we want to know the communication frequency between teammates further, which is a key factor of a group.

Select issueID, count(*) from comment group by issueID order by issueID

## 2.8 Average Time between Comments

This is the average value of period lengths between two consecutive comments per issue. It is important that all team members keep an eye on what their teammates have posted and response quickly. We want to know whether this is true in real by this attribute.

## 2.9 Number of Milestones(Total/Incomplete)

This is the total number of milestones and total number of incomplete milestones for each group. Milestone is a common goal for all team members, and completing milestones is crucial for a group. So we want to know whether a team can complete most of their milestones. We also want to know whether they have detailed milestones, that is, whether they just have simple, rough milestones.

## 2.10 Issues Created per User

As with users being assigned issues, we felt that it will be beneficial to know spread of who created issues. In SQL:

select user, count(*) from event e1 where time ¡= (select min(time) from event where issueID = e1.issueID) group by user;

## 2.11 Issue Completion Time

Issue completion time indicates the average time the users of a group spend to work in collaboration to resolve something.

## 2.12 Help Wanted Issue

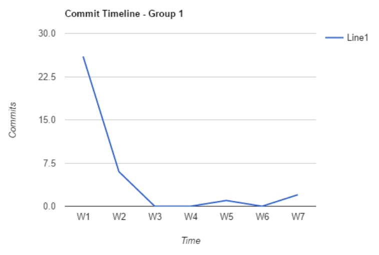
This feature shows us the proportion of issues that had labels that indicated there was a help needed in a particular issue.

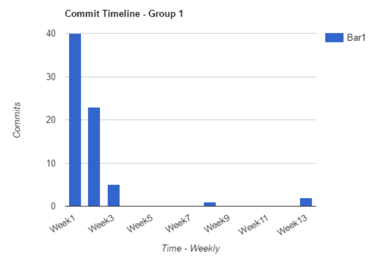
Select count(issue_id) from event where label = "help wanted"

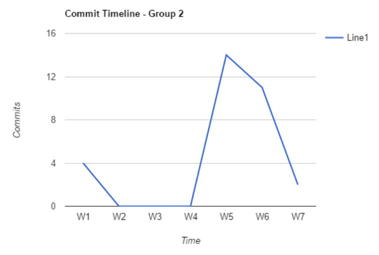
# 3. Analysis

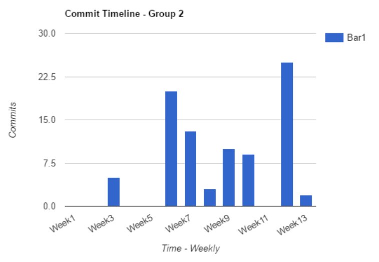
## 3.1 Commit Timeline

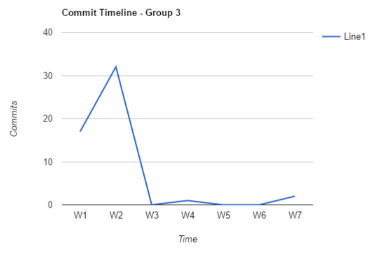
Ideal trend should be a constantly increasing linear graph with slope of 1.0 assuming that with each going week, the understanding, workload and feature extension are increased.But we have noticed different trends for different groups. Most of the work has been done in the initial 3 weeks and then number of commits constantly plummeted till the end where it spiked up a bit.

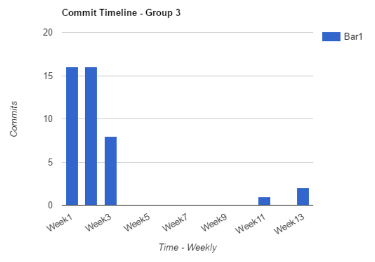






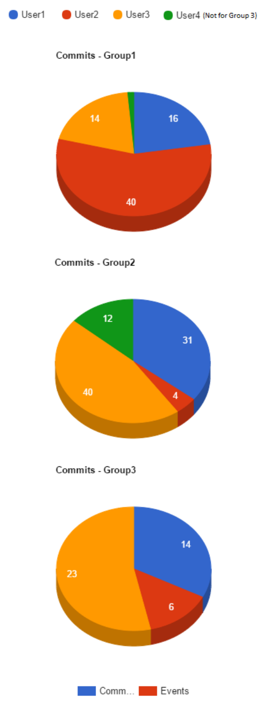


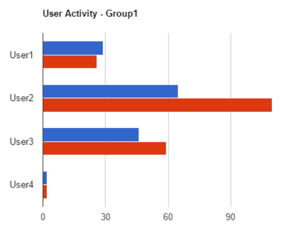


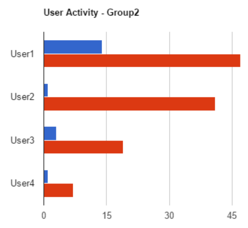


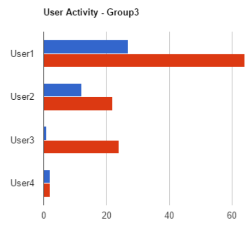
## 3.2 User Activity

We grouped together all the github related activities in the single chart where total number of comments and all the milestone & issues related events are represented by blue and red bars respectively. Individual contributions of all the users are also represented in terms of commits. Ideally, all the users should have almost equal number of user activity and code contribution to the project, but that is not the case with any particular group here. For each group, we have a highly active(eg.User 2 in group 1) and highly passive/inactive members(eg. User 4 in group 1)



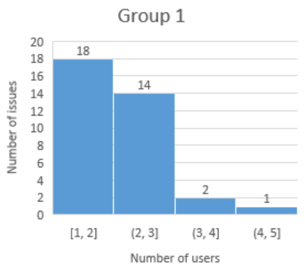


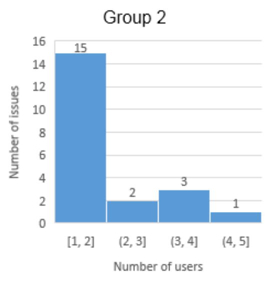


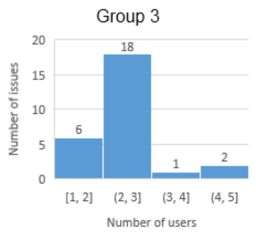


## 3.3 Participating Users per Issue

Since every group has added instructors as contributor, most groups have 4 "real" users. This is a very small group. Hence we expect that, in most cases, good activity means the issues will related with most of the members. Meanwhile, a few issues will only be related with one or two members considering the distribution of work. Per our hypothesis, group 1 has a good activity with considerably equal number issues that involve 1-2 users and 3 users. Group 2 has most issues related with only 1-2 users which indicate that there may be users that didn't fully carry their responsibility. Group 3's data looks closer to our expectation. They have most issues related 3 members and some issues specified for 1 or 2 users.

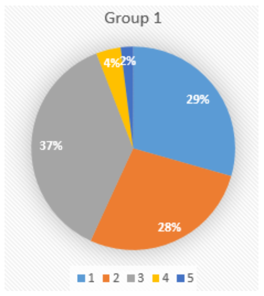


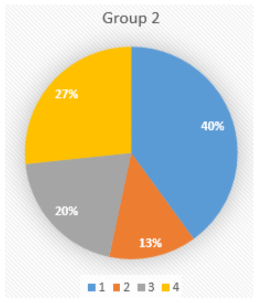


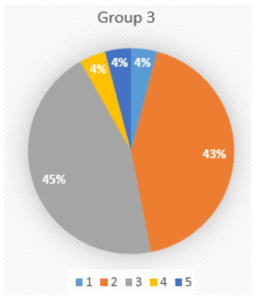


## 3.4 Number of Assigned Issues per User

As is shown below, group 1 (4 members) has some degree of balanced workload. We can see user 1,2,3 shares similar number of issues, while user 4 has a low rate of activities of issues. Group 2 did a little better. All of them have numbers of assigned issues. However, one of them took nearly half of the issues which means they put too much work on that member. Group 3's data is another version of group 2. They have two key members who undertook more than 85 percent issues which cannot reflect their distribution of work.

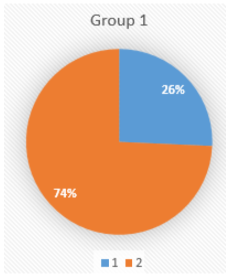


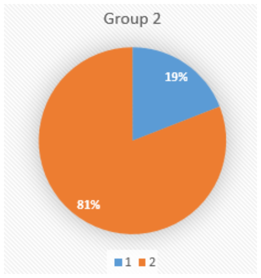


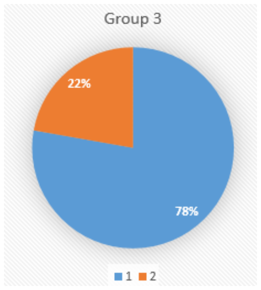


## 3.5 Issues without Milestones

We expect all the issues during specific stage of milestone should have been labeled a milestone. Meanwhile, there are cases that some issues are not related with milestones. For example, the issue about a general design bug or issues cannot be classified by milestones. Hence we do allow some percentage of issues without a milestone. As is shown below, group 1 and group 2 has 26% and 19% issues without milestones which looks reasonable. However, group 3 has a rate of 78% which looks very high among this data set.

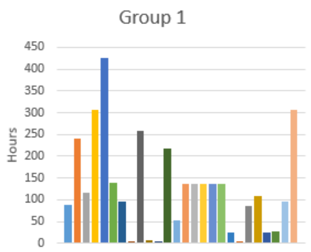


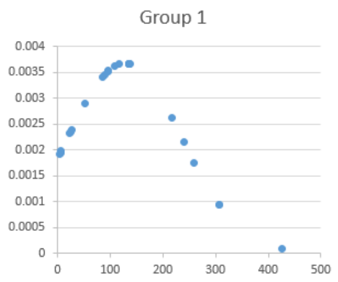


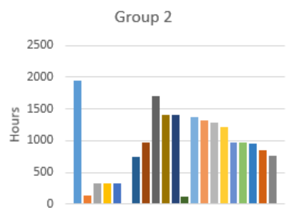


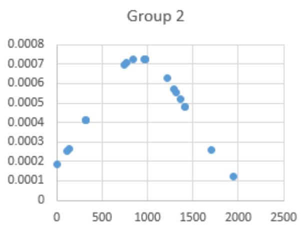
## 3.6 Long Time of Issues

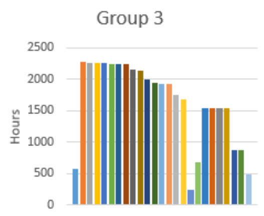
As is shown below, Group 1 has a mean of 127.59 which is a relatively short time to close an issue. The standard deviation is 108.36 which is larger compared with their mean. That is not a bad thing from our view. It means they have a wide range of time interval to solve different kinds of issues. Also, they have a reasonable number of issues that took "unusually" long time (between 300 and 500 hours). Group 2 has a mean of 910.85 and standard deviation of 546.48. For a project lasting about 4 months, 910 hours (about 37 days) to solve an issue looks not so good. The good thing is that group 2's data do not contain any "unusual" values. Group 3's mean is 1650.14 and standard deviation is 650.63. The data does not look like suitable for normal distribution because most of the data is between 1500 and 2500 and the standard deviation of this part is relatively small. Moreover, 1650 hours to solve an issue is not a good practice while working on the project.

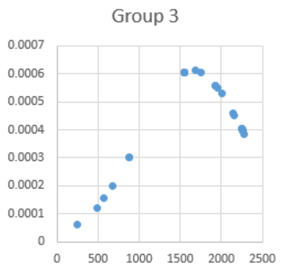






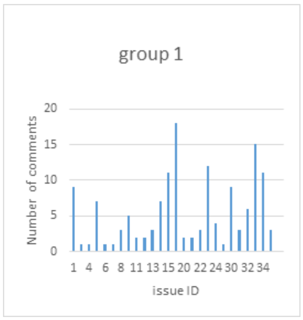


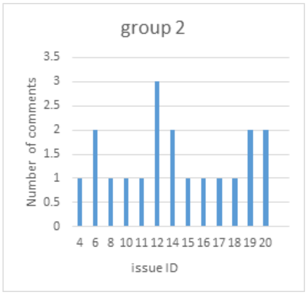


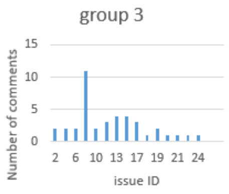


## 3.7 Number of Comments per Issue

As we can see, there are several issues whose comments are more than the number of team members in each group. And this can prove that there will be common problems in a team. In other words, since all team members are working together to finish a common target or finish similar targets, it is very possible that they will face similar problems, similar issues. But on the other hand, there are about 50 percent of issues whose comments are less than the number of team members, that means about half of issues are not responded by all the team members. And this is an indication of lack of communication between team members.

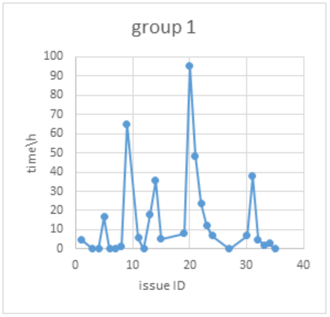


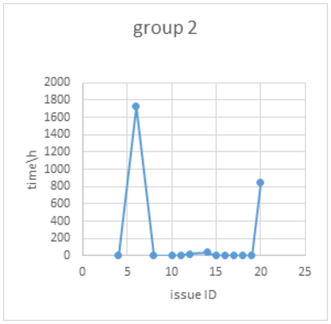


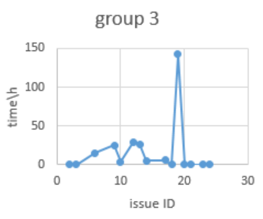


## 3.8 Average Time between Comments

As we can see from the graph, average time between comments for most issues in each group are less than 40 hours, that indicates group members have taken notice of what their team mates have posted. But in each group, there are issues whose average time between comments is large, for example, in group 3, issue 19's average time between comments is over 140 hour. That indicates for some issues communication between team members are really slow.

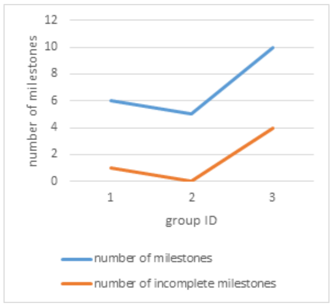






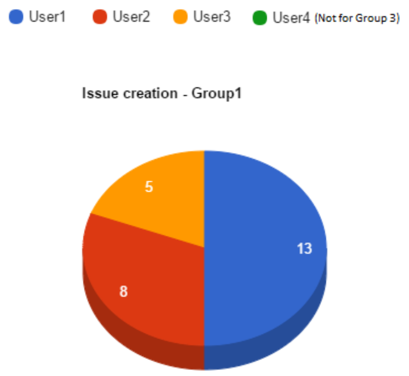
## 3.9 Number of Milestones(Total/Incomplete)

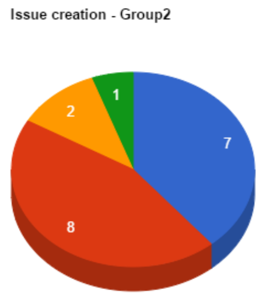
As we can see, group 1 and group 2 has finished most of their milestones, while group 3 still have 4 milestones to complete. But totally, all the groups finished half of their milestones.

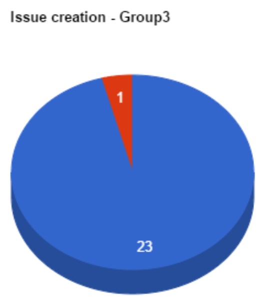


## 3.10 Issues Created per User

This feature shows us the total number of issues created by each user. GitHub defines an issue as something that stops the progress of a project and should be resolved. The issue includes labelling, referencing, mentioning, closing, merging, subscribing or unsubscribing to, and milestoning the issue. We have taken into consideration only the creation of the issue.
We were expect all the users to have approximately an even share of issues they've created. From the data, we see that this is the case for Group 1. Group 2 seemed to have user 1 and take part in the issue creation about 95% of the events, while Group 3 had user 1 and 2 take part in creation for more than three-fourths of the events alone. Group 1 also had user 1, take part in approximately creating half issues, indicating that these groups had one member that did most of the work pertaining to organizing issues.

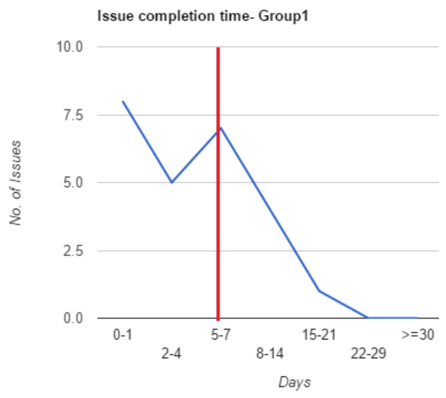


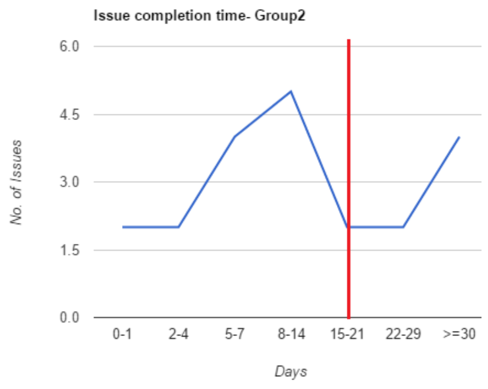


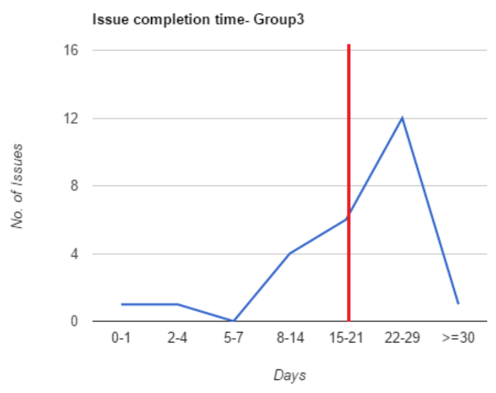


## 3.11 Issue Completion Time

This data shows the length of time that issues had been open. It was a calculation of inception of an issue until when it was closed. This is a bit difficult to categorize as better or worse either way, but issues over 20 days old are not usually the best in our experience. Ideally, we'd expect issues closed every week since people should be continuously working on the project, with peaks closer to the deadlines when most work is completed. From the data we see that this is the case with Group 1. Group 2 was also pretty regular with their work except for a three week period towards the end where they did not close any issues. Group 3 had issues closed only over a 6 to 7 week period and this shows us that they didn't start work with the rest of the groups or they didn't use issues properly to log their work.







## 3.12 Help Wanted Issues

| Group 1 | Group 2 | Group 3 |
|---|---|---|
| 1 | 4 | 6 |

This feature shows us the proportion of issues that had labels that indicated there was a help needed in a particular issue. Having no help wanted issues, or having too many issues labelled with terms pertaining to help raises questions with a project's collaboration and effectiveness. If there were no issues that had a label indicating there is a help wanted, one possibility is that the group didn't do group work enough or views of other team members are not taken into consideration much. Another possibility is that the group didn't use issues to communicate on the presence of difficulties in the code.

On the other hand, having too many help wanted issues by a particular user, also indicates that something isn't quite right with the project, members are struggling to accomplish a task on their own. Both these cases lead to a drop in effectiveness of the development of the project. The data shows us that Group 1 reported a single help through issues, while Group 2 and Group 3 had an unusually four to five of issues that were related to help. Group 1 also had the least number of issues, indicating they did not really use them correctly. This is one possible reason for the lack of any issues being related to help. Groups 2 and 3 had approximately 5% of their issues related to help, and this seems to be a reasonable number to indicate their projects were progressing smoothly.

## 4. Bad Smell Detector

The bad smell upon further inspection should most of time lead to interesting problems and should be quick to spot in a program. The symptoms of poor design and implementation choices can be the bad smells in programming. A bad smell in programming can be an indication that usually corresponds to a deeper problem in software.

The statistics and data analysis on small features we have discussed in above section, in this section we will discuss bad smells with combinations of our feature extractors and the result analysis from them. Here we adopted a different approach by studying GitHub data generated throughout development process and focused on its implication of problems with design and development which is different from convention where refactoring is done and bad smells are detected by analyzing the source code.

### 4.1 Bad Collaboration

The major bad smell we identified was having teams where just one person is contributing. When one or two persons from the team does the major part of the work both by making changes in their commits and events contributed to the project it reflects a bad smell showing lack of collaboration.

Explaining this further, figure 8.1 displays the changes made proportionally by user to the code base through commits, and figure 8.2 and 8.3 displays the events proportionally more by one user. It can be seen from these two figures that a trend with groups 2 and 3 identifying them as examples of one person teams.

The figure 11.3, shows that an overwhelming majority of the issues made for group 3 were by a single person. Group 1 also demonstrate a single person creating most of the issues. To affirm these findings, we cross the reference figure 12.1 and we can see that for groups 1 and 3 the majority of events are also done by the same user. It was essential to check this reference, otherwise if we just look at changes made it would appear more groups classify as one person groups, when in reality the entire group is working together, rather a single user happens to have more number of commits.

We identified a sub bad smell, the two person group, when inspecting with this bad smell. An example of for this smell is group 2, which as shown in figure 2.2 exhibit two users doing the majority of changes and events.

Along with this bad smell of one and two person groups there is also another bad smell that can be identified within these figures that we call, communication without collaboration. An example of this bad smell can be seen with group 2. In figure 11.2 we can see that group 2 has excellent collaboration in terms of acting on issues, and if we back reference to figure 6.2 they also have excellent communication and closing the issue in less time. However, when observing figure 2.2 it is apparent that a single user did most of the changes through commits. Thus group 2 is an example of communicating without collaborating.

## 4.2 Procrastination

On procrastination we define a group's pattern of taking time to accomplish certain tasks to be effective when they commit frequent small changes to the code base. To this end, we have defined two bad smells on procrastination.

The first bad smell is the long lasting issues. This occurs when a group spends way too much time working on resolving a single issue, with many people commenting but not reaching on any conclusion. Furthermore we coined this term as long lasting issues because even though they resolved everything in short time,, they also spent a lot of time discussing but not even closing the issue after that.

A group that demonstrates this bad long lasting issue smell is group 3. If we look at figure 12.3 we can see that group 3 had the majority of their issues lasting for more than a month; however, they also had one absurdly large addition that was not closed for two months. Further, if we cross reference this with figure 5.3 we can see that this group had large numbers of issues without milestones.

On the other hand, our second bad smell related to procrastination is from commit timeline. This occurs when a group semi-frequently adds larger commits in a particular timespan and does not efficiently contribute regularly. Examples of this bad smell can be seen by looking at groups 3, we can see in figure 1.3 that both of this group have no commits at all in a time range of approximately 30 days. As well, when we then look at figure 3.3 we see that group 3 had just one user who regularly commented on issues. Thus, both group 3 fall victim to the procrastination bad smell.

## 4.3 Bad Planning

Bad planning was a bit of a hodgepodge of different feature extractors. The motivation here is that if a group had poor planning there could be many different failings that emerge over a semester. If a group has planned poorly it would have likely limited the amount of success they could achieve over the life of their project. The main way that takes place is in the lack of thinking about potential roadblocks in a project, if there is no plan in place then projects could get much of the way through development and hit an issue that makes the entire project non-functional. Also a group may find that the scale of their project is either way above or way below the amount of time available for the project. The metrics for the smell are:

- Long time between issues opening and closing
- Issues missing milestones
- Equal number of assignees
- Short issues
- Commit history

For long time between issues opening and closing, we considered twenty days a poor performance. At twenty days an issue had been outstanding for quite a long time and represents a bad smell. We used the same calculations mentioned above for issues missing milestones, equal number of assignees, and non-linear progress momentum. Finally with the linearity of commit history, if the error in linearity was greater than 20%, we considered that a bad smell.

With short time between issues opening and closing, we considered that if greater than 20% of the issues for a group were open less than an hour, that was a bad smell.

## 4.4 Poor Communication

The last bad smell detector is for poor communication. If a group cannot communicate well, then their productivity over the long term may be affected negatively which is the motivation behind the bad smell. People in such situations may work discordantly which results into issues such as commit collisions. It uses three feature extractors, the number of assigned issues per user, the number of comments in issues and user activity.

For the number of assigned issues per user, we were looking if the issues are assigned equally to all the users. The idea is that if only a small percentage of a group actually discussed issues then there is very poor communication overall. We set the limit at maximum 50% of the issues, can be assigned to a single user, more than that had a bad smell.

Number of comments in issues was gauged a bit differently. We came to the conclusion that if an issue was important enough to be posted, then there should be some kind of discussion of that issue. We decided that less than 2 comments per issue was a bad smell. At that point there is little to no discussion of individual issues.

Percentage of comments by each user is another metric for communication which is taken from the user activity. If only a few members post comments in issues, then there are only a few members that are actually communicating.

## 5. Conclusion

Based on our analysis of the data we collected, we think that although teams can finish their application and finish other processes necessary for an application, there could be many problems during the completion process and there are still much space for the teams to improve themselves.