Bad Smells

Fuxing Luan
Master of Computer Science
North Carolina State University
Raleigh, North Carolina

Fuzail Misarwala
Master of Computer Science
North Carolina State University
Raleigh, North Carolina

Zubin Samuel Thampi
Master of Computer Science
North Carolina State University
Raleigh, North Carolina

I. Data Collection and Anonymization

We collected data from GITHub for all teams, and chose to store them in CSV files. We did not want to use SQL, or any tabular structure, as we were analysing the data and plotting graphs completely on R, and we found R supports CSV files very well.

We created 4 different CSV files for 4 categories of data that we captured - Commits, Issues, Milestones and Events. For each category, we stored data for all teams together in the CSV files, so it is easier for us to draw comparisons. We also made the data anonymous, and assigned IDs to each team. For example, team1, team2 and so on. Within a team, we gave IDs to each user. For example, user1, user2, user3 etc. There were some rows in each Category from the Course Professor and TA, which we left intact. However, we excluded those rows using R code, prior to our data analysis and inference. We also focused on getting numerical data (For example, the number of comments instead of actual content of comments), to retain the anonymity of teams as much as possible.

II. Sample Data

As mentioned above, we stored data into 4 different files.
Issues.csv :

assignee	assignees	closed_at	created_at	created_by	state	body
user1	['user1', 'user2']	2017-03-01T00:58:25Z	2017-02-27T02:36:40Z	user3	closed	***
comments	labels	milestone_closed_at	milestone_created_at	milestone_due_on	title	
6	[u'help wanted']	2017-04-07T21:54:07Z	2017-02-07T21:54:07Z	2017-02-26T08:00:00Z	***	

Fig. 1. Sample data in Issue.csv

Milestone.csv:

created_by	title	description	closed_issues	created_at	closed_at
user1	***	***	5	2017-02-16T19:58:12Z	2017-03-15T17:54:08Z
team	number	open_issues	state	due_on	updated_at
team1	11	2	open	2017-02-24T08:00:00Z	2017-03-08T03:24:21Z

Fig. 2. Sample data in Milestone.csv

Event.csv:

actor	created_at	issue_number	issue_title	milestone_number	milestone_title	team	type
user2	2017-03-09T	21	***	9	***	team1	PushEvent

Fig. 3. Sample data in Event.csv

Commit.csv:

created_at	created_by	message	team
2017-04-28T	user1	***	team1

Fig. 4. Sample data in commit.csv

III. Features Analysis

1) Number of Commits

Our aim was to evaluate if there is a co-relation between number of commits and the general performance of a team on GITHub. Through our initial discussion, we came up with two different hypothesis.

a) Low number of commits indicate less involvement on GITHub, and overall lack of cohesion in the group.
b) All groups work differently. We would find high number of commits in teams which were working on all implementations collaboratively. We know several teams split up and took responsibility of one implementation each per user. They could possibly have lesser number of commits as users develop individually, and commit the code only when completed, or less frequently.

From Figure[5], we could not identify any patterns. The fact that the number of commits are different for each team is suggestive of our second hypothesis. It is indicative of their own characteristic method of going about work. We cannot say conclusively that the teams with lower number of commits performed bad. We decided that a better measure to evaluate would be to take the distribution of commits per user within a team, or the distribution of commits over time.

2) Number of Commits Per User

Figure[6] gives us a clearer picture of the distribution of commits within a team. Keeping 12% as the threshold for significant contribution, we can see that team3 and team6 has had some team members contributing very little of the total commits. We conclude that this is indicative of an absent team member in the group.

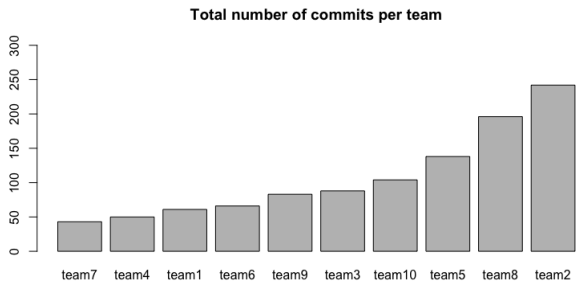

Fig. 5. Total number of commits per team

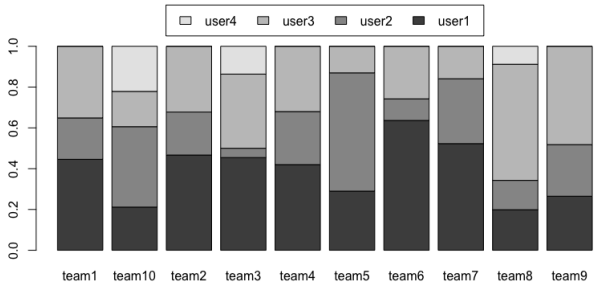

Fig. 6. % commits per user per team

3) Number of Commits Per Week

To evaluate this, we roughly split the project span into 16 weeks, and tracked the number of commits per user in a team. Figure[7] shows the scale that we used for plotting the number of commits. (The scale is based on the maximum number of commits per team, to avoid any bias based on the style of work of the team). Black spots indicate larger number of commits, and white indicate zero commits. Ideally, the line for a user should be majorly gray and black, suggesting consistent commits and activity. Our hypothesis is that lines which are majorly white indicate absent team members.

In the graph, the region in the middle should ideally have more activity, as those weeks align with the actual development phase of project. We understand that many teams might not have been active on GITHub during the Survey, or evaluation phases of the project. Therefore, in our analysis, we tried to overlook figures which have relatively less activity in the beginning, or towards the end.

Figure[8] shows a good amount of consistent activity for team5. We found the graph to be similar for majority of the teams. Most of the users have frequent gray spots in the line.

However, user2 in Figure[9] has very few gray spots in the line, and has been absent for the rest of the weeks. This is in line with the hypothesis that we drew, and is a valid indicator of a bad smell within a team.

Further, in Figure[10], we see an absence of activity for the entire team7 for a span of about 10 weeks in between. We identified this as a potetial bad smell, indicating lack of co-ordination and consistent effort in the team.

We found that the users had some commit on 32% of the weeks on average, with a standard deviation of

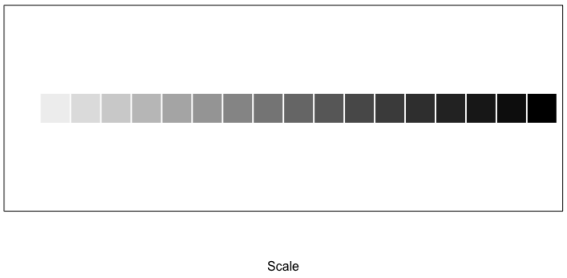

Fig. 7. Scale for plotting number of commits

12%. On median, users had commits on roughly 28% of the weeks. We framed our threshold on Table[I] based on this statistic.

4) Number of Issues

Issues are a great way to keep track of tasks, enhancements, and bugs for projects. Setting up issues frequently is a key for distribution and assignment of issues to persons in charge, Monitoring of handling, time spent and quality of work, and more importantly, making sure the development is proceeding in a good pace.

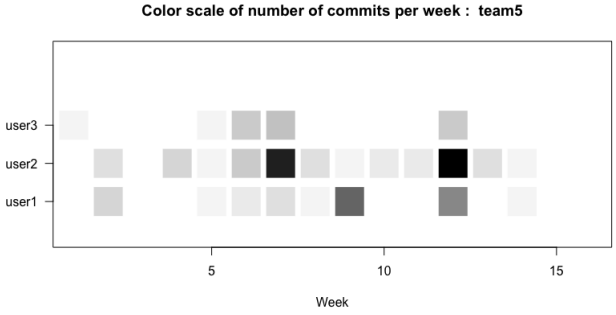

Fig. 8. Number of commits per week for team5

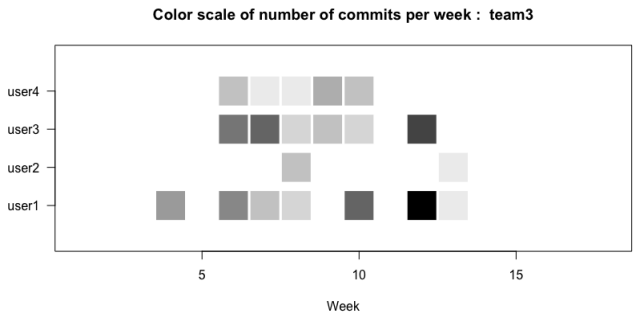

Fig. 9. Number of commits per week for team3

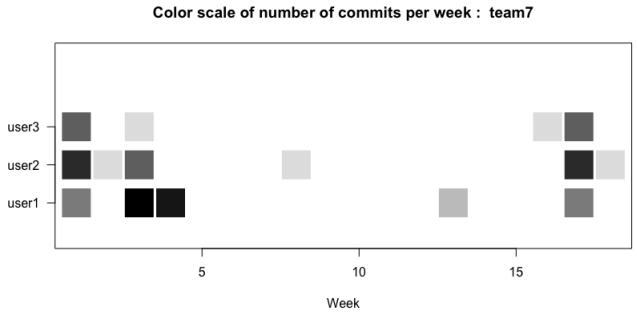

Fig. 10. Number of commits per week for team7

Our goal is to figure out whether the quality of a project is in direct proportion to the number of the issues. We have two hypothesis after initial discussions.

a) The teams that have high number of issues may be more enthusiastic in either development or debugging, also more likely to have a very specific work division since issues on Github can be assigned to team members. On the other side, teams that rarely use issues may have a lack of work motivation or communication, which may lead to a low number of commits or only one or two team members actually involved in the project.

b) The number of issues are reflective of the style of work of individual teams. While some prefer to create issues for even the smallest of bugs, others prefer to resolve them over other medium - like Slack, or Email. We agreed it is possible that some teams made issues on GITHub to track their progress in the long run.

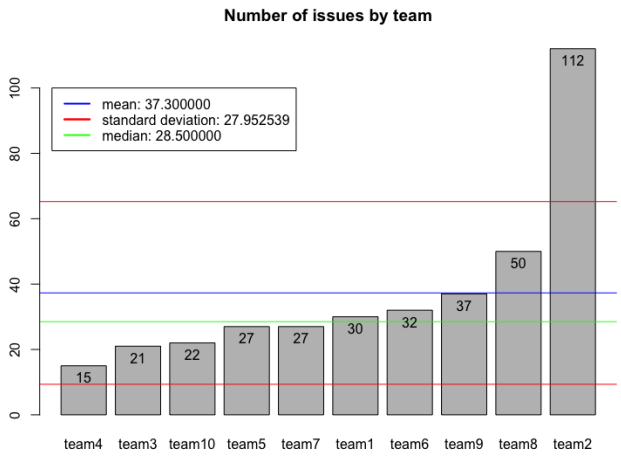

Fig. 11. number of issues per team

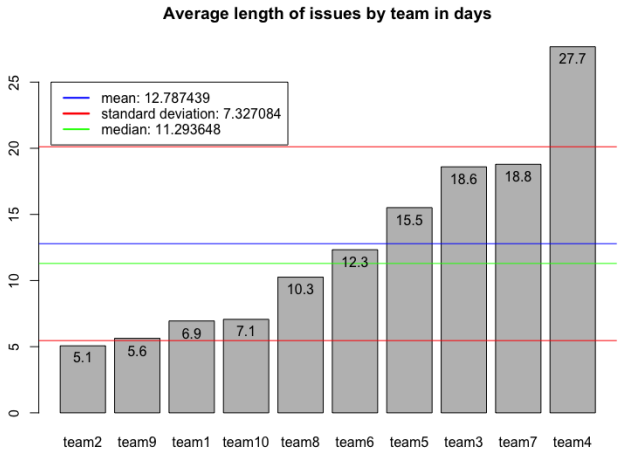

Fig. 12. Time taken per team to finish an issue

From Figure[11] alone we are not able to conclude whether or not there is a connection between the number of issues and others.

But if we take Figure[11] and Figure[12] together, we see that team4 and team2 are ironically, the leaders and trailers in the respective graphs. We found that teams with higher number of issues close off their issues faster. Whereas, teams with lower number of issues, are ones who plan on the long run. We cannot conclude that teams with lesser number of issues perform badly. This is reflective of our second hypothesis.

However, if we compare Figure[11] and Figure[13], there is no obvious pattern we can draw. Team4, which has the lowest number of issues, surprisingly, has the best proportion for each group member. And team2, which has the highest number of issues, its proportion for each member is not too bad either. We might have to say there is no connection between the number of issues and whether there is a lazy team member in a group. We found that in team7, there is an uneven

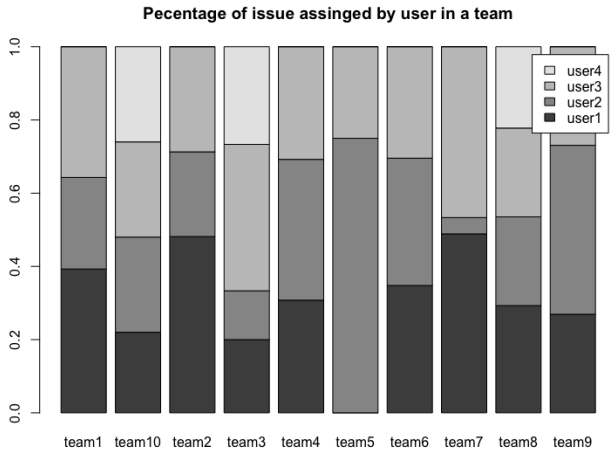

Fig. 13. Percentage of issue assigned by user in a team

ratio of issues assigned per user, which is a definite indication of bad smell. We concluded that it signifies lack of planning, and work division.

5) Time span of issues

Comparing Figure[11] with Figure[12], it seemed like teams with more issues close the issues faster. We came to this initial assumption because team4 and team2 were on extreme ends on these respective graphs. Our hypothesis was that teams with higher number of issues track their progress, including bugs entirely on GITHub. Whereas, teams with lower number of issues just used GITHub to track the progress of their project, resulting in longer closing times. In order to validate this, we did a plot of Mean and Median time span for issues within a team and the number of issues. From Figure[14], we see an almost inverse relation between the two. We also ran Pierson-Corelation test and Chi-Square test to confirm our hypothesis. Figure[15] concludes our hypothesis, since we got significant values in both the tests.

6) Time taken per user to complete an issue

From the data, we found that the mean time for closing an issue is around 10 days. The median value was approximately 5 days. Figure[16] reveals that the majority of the issue are closed in approximately the first 20 hours. Besides, we noticed that different teams create issues for various reasons. Some create issues to track their progress, and close them off within 5 minutes. Some even create issues to track Project Meetings, which stays open through the project span. Because of the high variability of the data, we decided not to draw any bad smell indicators from this data. We

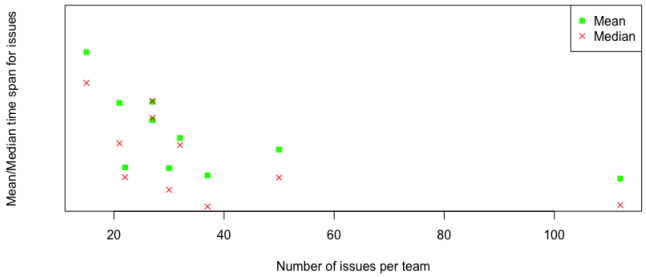

Fig. 14. Mean/Median closing time for issues vs Number of issues for the team

```
> cor(as.numeric(as.list(x)),as.numeric(as.list(y$x)))
[1] -0.5654863
> chisq.test(as.numeric(as.list(x)),as.numeric(as.list(y$x)))

        Pearson's Chi-squared test

data:  as.numeric(as.list(x)) and as.numeric(as.list(y$x))
X-squared = 80, df = 72, p-value = 0.2424
```

Fig. 15. Pierson and Chi-Square test results

decided a better measure would be the average closing issue closing time for a user within a team.

If users update their process of work correctly and timely, the time taken per user to finish an issue, no doubt, can represent either his work efficiency or workload. Therefore, our goal is trying to find out which team member make most contribution to the team and which one make least. From Figure[17] we saw that the average issue closing time is pretty much consistent for users within a team, indicating their own characteristic style of working.

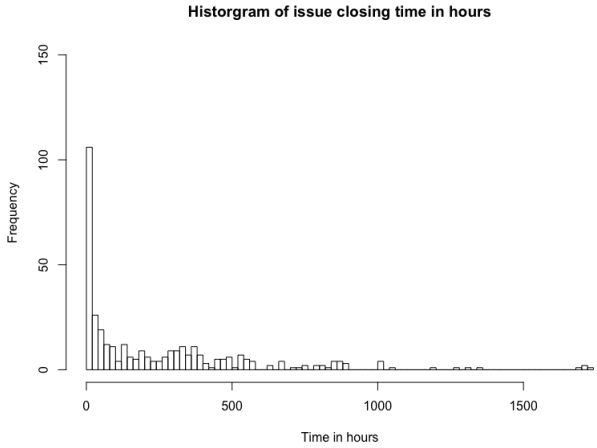

Fig. 16. Histogram of issue closing time in hours

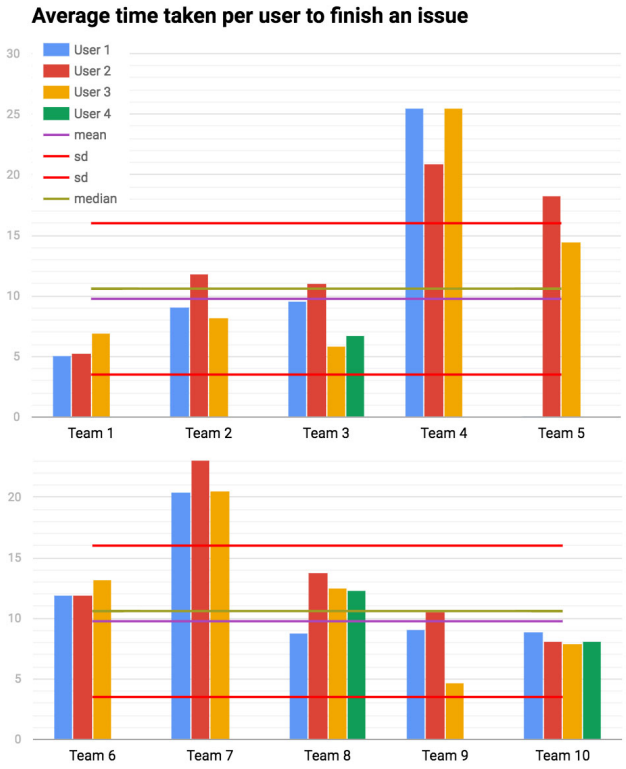

Fig. 17. Time taken per user to finish an issue

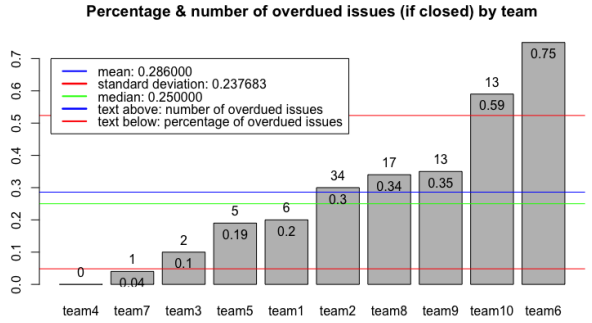

Fig. 18. Percentage of overdued issues

7) Issues that were closed after milestone deadline

Finishing issues on time is the key to a successful project, especially when there is due days for milestones. Our goal is to find out time management situation for each team and here is our hypothesis: If there are too many issues that closed after milestone deadline and if we assume the team update issue status on Github on time. Without doubt, the team either did not make a feasible plan of time line beforehand or they did not execute well, in the other word – procrastinated.

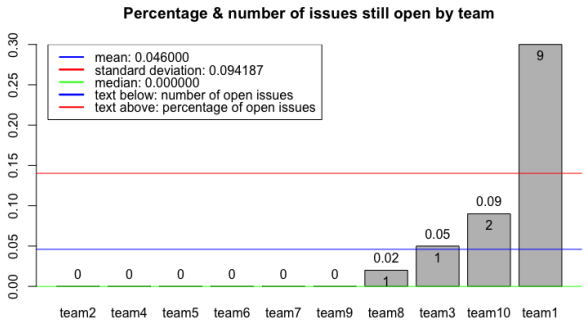

Fig. 19. Percentage of issues never closed

According to Figure[18], we can easily to see team6 and team10 did not have a good control of time (over half of their issues were overdued), whereas team4 finished all of their issues on time, despite the fact that they had the minimum number of issues.

8) Issues that were never closed

Since the project has been over for almost one month, there should not be any open issue.

Like the task above, we aim to figure out the feasibility of plans for every team. Here is our hypothesis: There are two possibility that some issues are still open.

a) Teams did not use Github well, becoming lazy in the finishing period of the project.
b) Teams made unrealistic plan and found out in the last that they could not finish.

Based on Figure [19], we can see most teams did a good job on closing issues (making feasible plans). However, we found that team1 has about 30% of issues that are still open. If we combine it with the results from Figure[18], it adds up to roughly half of the issues. We concluded this is a clear bad smell, and signifies lack of planning and co-ordination.

Also, team6 does not have any not closed issue but 75% of issues are overdue. The chance here is that they did not update GitHub very often, and in more likelihood, closed all the issues at the same time at the end of the project. We concluded that this again, signifies lack of planning.

9) Number of Issues created

Our hypothesis is that in a co-operative team, all users would be equally pro-active and raise roughly equal number of issues. Through graphical means, we wanted to find out users who have contributed less

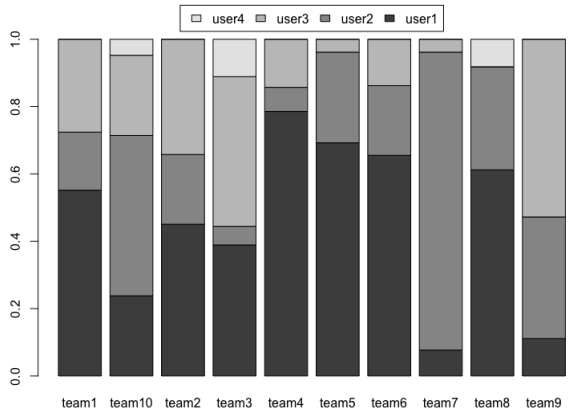

Fig. 20. % of issues created per user per team

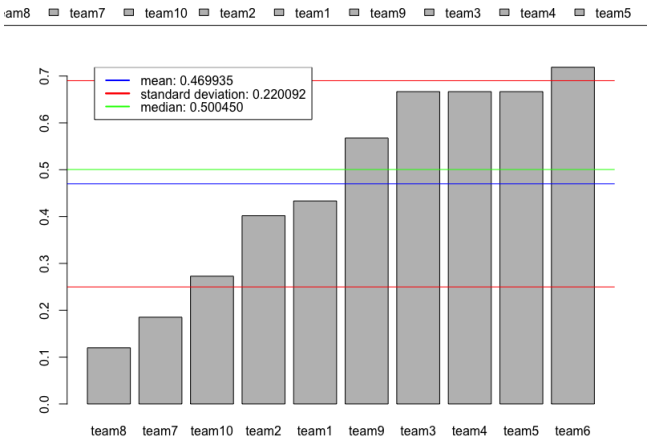

Fig. 21. % of unassigned issues per team

number of issues within a team, and users who have been dominating. Figure [20] revealed that this was a bad smells within most teams. Almost 5 teams have stark differences in the number of issues reported per user. We tried to reason that most teams might have a leader, who is usually the one pro-active on GITHub. That should explain the figures. But in any case, we concluded that this activity is a significant bad smell.

10) Number of Unassigned Issues

The number of issues per teams were starkly different. So, we tried to compare the percentage of issues that were unassigned per team. Figure [21] revealed no patterns in the data. The percentage of unassigned issues was different for different teams, signifying the style of work within the team.

11) Number of Events

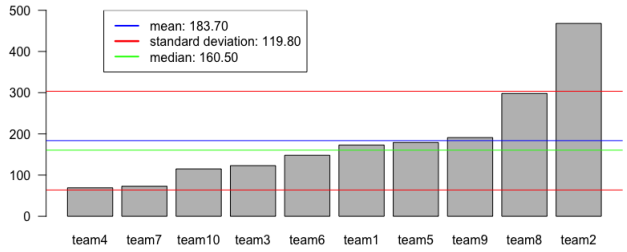

Fig. 22. Number of events per team

Events are a good signal of activity on a group. Events on GITHub are tracked while adding comments, pushes, forks, pull requests etc. First, we tried to find if there is any relation between the number of events registered on GITHub and the quality of the project. From Figure[22], we could not predict any patterns in the data. Similar to the number of commits, each team has varied level of activity on GITHub, which signifies the different style of work of each team.

12) Number of Events Per User

We thought it was better to compare the number of events for users within a team. We thought it was safe to assume that a well organized team would have set their own strategies, expectations and plans on GITHub. So ideally, within a team, team members should have similar level of activity of GITHub. Our hypothesis was that teams having irregular amounts of activity for different users should be considered a bad smell.

Setting a threshold of 10%, we saw from Figure[23] that team3, team5 and team7 has users within very less events on GITHub compared to their teammates. We concluded that these are definite signs of bad smells, and signifies absent team members in the group.

13) Number of Events Per Week

Having found that the distribution of events within a team is a significant sign of bad smells, we also wanted to check for consistent activity for users within a group. We believed that making a time line of events on GITHub could help us distinguish users who are consistent, from users who have sporadic activity. The latter could bring down the general motivation of the team.

Figure[24] shows the general pattern we saw in the data. We observed that most teams had uniform and

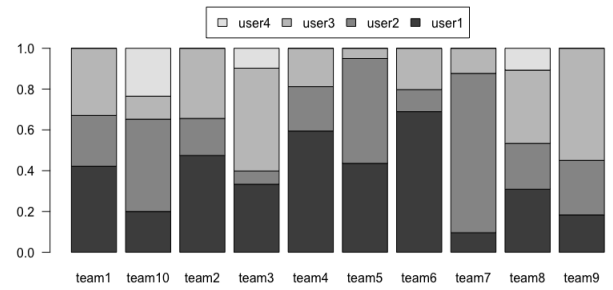

Fig. 23. % events per user per team

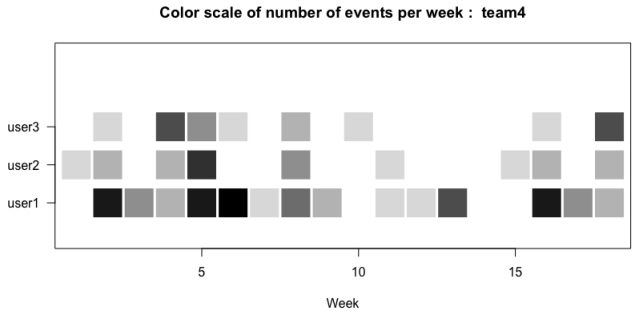

Fig. 24. Number of events per week: team4

consistent events counts for all users. We found an exception in Figure[25], where user1 for team7 had very less number of events. We concluded that this was a strong indicator of absent team member in the group.

We found that the users had some event on 37.5% of the weeks on average, with a standard deviation of 13.3%. On median, users had commits on roughly 39% of the weeks. We framed our threshold on Table[I] based on this statistic.

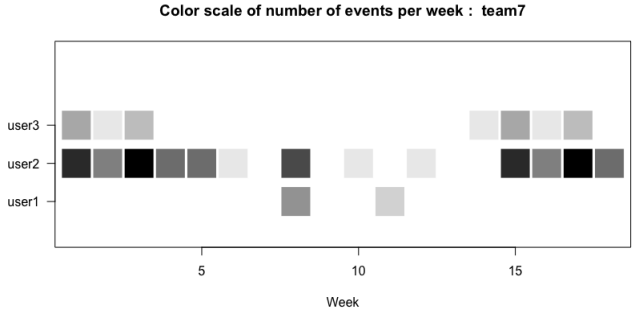

Fig. 25. Number of events per week: team7

TABLE I
BAD SMELL FEATURES AT A USER LEVEL

ID	Description	Bad Smell	Threshold : Score
bs1	less % commits within a team	Absent team member	<10% : 2 <15% : 1
bs2	0 commits for more than x weeks	Inconsistent Absent team member	>85% : 3 >80% : 2 >75% : 1
bs3	% of assigned issues within a team less than threshold	Low Team Cohesion Absent team member	<10% : 2 <15% : 1
bs4	% of issues created less than threshold	Low Team Cohesion Absent team member	<10% : 2 <15% : 1
bs5	% of issues created greater than threshold	Low Team Cohesion Dominating member	>70% : 2 >60% : 1
bs6	% of events less than threshold	Low Team Cohesion Absent team member	<10% : 2 <15% : 1
bs7	0 events for more than x weeks	Inconsistent Absent team member	>85% : 3 >80% : 2 >75% : 1

TABLE II
BAD SMELL FEATURES AT A TEAM LEVEL

ID	Description	Bad Smell	Threshold : Score
bs8	number of issue closed after milestone deadline	Lack of planning	>10 : 3 >5 : 2 >0 : 1
bs9	issues not closed	Lack of teamwork	>3 : 2 >0 : 1

IV. BAD SMELLS

After analysing the data in details, we categorized our bad smell indicators into two

1) Bad smells at a user level (Table[I])
2) Bad smells at a team level (Table[II])

For each of the bad smell features, we defined thresholds, and their respective Bad Smell scores. We intend to calculate a bad smell score for each team at the end, and display the results based on that.

V. RESULTS

Figure[26] and Figure[27] summarizes our results for bad smells at a User level. We found that no team is perfect. Every team had members scoring at least 1 point in the bad smell indicators that we chose. team1, team2 and team9 had a relatively smooth ride through the projects, and scored very less in the bad smell criteria. However, team3, team7 and team8 scored high on a lot of these bad smell indicators. Figure[27] gives us a better picture about what might have gone wrong. team3 had issues for 2 specific users, indicating they had absent team members. team7 had users scoring equally bad, indicating a combination of bad smells (absent team members, dominating team members, lack of

	team1	team10	team2	team3	team4	team5	team6	team7	team8	team9
user1	0	0	0	0	2	3	1	8	1	2
user2	1	0	1	12	2	0	3	5	1	0
user3	0	1	0	0	2	7	2	6	6	0
user4		6		5					6	

Fig. 26. Score for Bad Smell at a user level

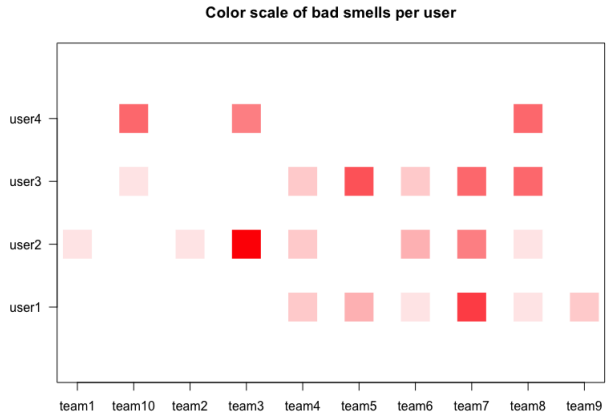

Fig. 27. Color scale for Bad Smell at a user level

consistency, lack of team cohesion, lack of planning etc).

Bad smells for users in a team directly co-relate to bad smells in a team. A team cannot be completely functional without all team members contributing equally. Hence, we decided to sum up the bad smell scores for users to get the bad smell score for a team. Further, we added the bad smells scores from the features we identified at a team level. Together, we thought it was an accurate representation of bad smells in a team.

Figure[28] shows the bad smells at the team level after we added up the bad smells at a user level, and the bad smell criteria from Table[II]. Even though all teams had some score in bad smell criteria, team1, team2 and team9 came out as the best 3. On the other hand, team3, team7 and team8 scored a lot on bad smell criteria, and ended up in the bottom 3.

VI. Early Smells

We found that the number of commits or events per week is a good early indicator of bad smells. We can draw graphs similar to Figure[29] for data like Commits, Events, Number of Comments etc. The expectation is that the data should be evenly distributed in these graphs, indicating good team work and collaboration. If you find irregularities in the data, it will be a signal to intervene and fix the issue. We can also configure the time span of the graphs (3 or 4 days instead of 1 week), based on the requirements of the projects.

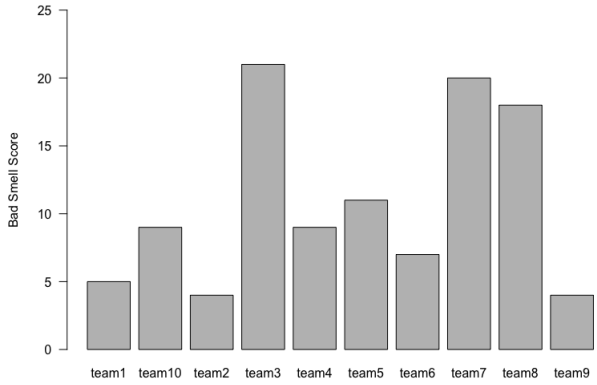

Fig. 28. Score for Bad Smell at a team level

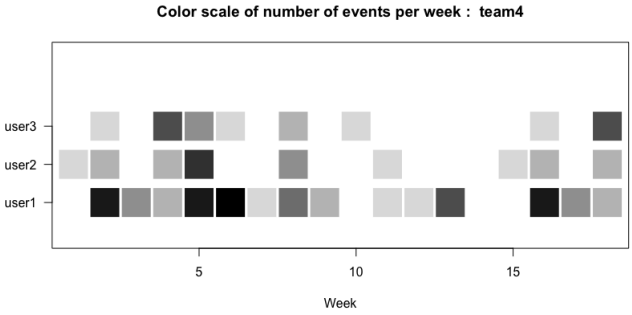

Fig. 29. Example of timescale representation of Commits/Events

VII. Conclusion

Through our analysis, we conclude that in Software Engineering projects, there is more than what meets the eye. Although we analyzed 10 teams working on a similar project, with similar deadlines and deliverables, we found stark contrast in the ways they went about work. In the data, we observed teams with 5x number of issues, commits etc, which gives an easy bias to someone just glancing over it. However, after the analysis, we found that even the teams with lower number of issues, and commits could have performed equally well. A lot of it depends on the architecture, the medium of communication, and the level of collective work that the teams agree on. We found that it makes more sense to compare data within teams, and to check whether team members work consistently and follow similar work pattern. That is more indicative of collaborative effort, and quality of Software Engineering practices. This further validates the fact that the majority of the bad smells we identified in our study, were at a User level within a team. That said, we also identified a couple of general bad smells which can be identified at a team level. Together, we found that it is a good representation to identify bad practices in Software Engineering projects.

VIII. Future Work

Following are the ways we could improve the Bad Smells Detector.

1) Due to limited time, our study was restricted to 4 categories mentioned in this report - Milestones, Issues, Commits and Events. We believe we can get even more useful information (and bad smell features) from stats like - Numbers of comments in issues/commits, Number of lines of code in commits, Percentage of total files contributed by each team member, Percentage of files contributed collectively etc.
2) We implemented our code in two stages now - 1. Capture the data and store it in CSV files 2. Get the static data from the CSV files and analyze it. However, having identified the bad smell detectors now, we feel we can easily build an end to end configurable solution. The solution will take the GITHub repository names as input, and output the summary of bad smells, as pictured earlier in the report.
3) In our study, we concentrated our efforts on some sections on graphical outputs, and comparison by visual means. However, we feel confident we can totally convert them into numerical means to come up with a COCOMO like model for detection of bad smells.