Detecting Bad Smells in Github

Dharmendra Vaghela
Masters in Computer Science,
North Carolina State University,
Raleigh, NC, USA, 27606
djvaghel@ncsu.edu

Aditya Mandhare
Masters in Computer Science,
North Carolina State University,
Raleigh, NC, USA, 27606
amandha3@ncsu.edu

Rohit Mandge
Masters in Computer Science,
North Carolina State University,
Raleigh, NC, USA, 27606
djvaghel@ncsu.edu

Rupaj Soni
Masters in Computer Science,
North Carolina State University,
Raleigh, NC, USA, 27606
rosoni@ncsu.edu

ABSTRACT

Bad smells are signs of potential problems in code that may adversely affect the maintenance of software. They make a system difficult to change, which may in turn introduce bugs. Bad Smell detection is a time consuming work for software engineers despite proposals on bad smell detection and refactoring tools. They are detected, in general, using quality metrics that formalize some symptoms based on detection rules. In this project, we have extracted the data from Github repositories and analyzed it. Potential bad smells have been mined from the data and an infographic representation of it has been done.

1. INTRODUCTION

Good design quality of software will ease the maintenance and reusing, and reusability, flexibility, understandability, functionality and extendibility will be improved. On the other hand, if too many bad smells exist in programs, the quality of the software would be very low. Bad smells are signs of potential problems in codes. It causes difficulty for understanding and modifying of programs. Bad smells are not mistakes or defects in codes, but may cause them. Indeed bad smells are alerting of codes. So bad smells should be removed if found.

Nowadays there is an increasing number of software analysis tools available for detecting bad programming practices, highlighting anomalies, and in general increasing the awareness of the software engineer about the structural features of the program under development. In this paper we focus our attention on code smells and on automated scripts developed for their detection. The main purpose for this report is to find the bad smells across various repos, which widely existed in our own development process as well. Thus, it is the conclusion that matters. This project enabled us to evaluate our performance during the semester long project which will prove to be highly useful in all our future software engineering careers.

2. COLLECTION

In this section, we explain how we have collected data/records during the development process. We have collected following types of data- issues, commits, milestones and labels.

We have used gitable.py file to extract the data from the github repos. In order to extract the data, we made use of an authorization token.

2.1 Collecting Issues

Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. They're kind of like email—except they can be shared and discussed with the rest of your team. Most software projects have a bug tracker of some kind. GitHub's tracker is called Issues, and has its own section in every repository.

A record of issues can be a good resource to learn how collaboratively and expressively the team members are coordinating and how closely have they followed the road map. We get all the information about issues tagged as "id", "url", "html_url", "state", "title", "body", "user", "label", "milestone", "comments", etc.

2.2 Collection Milestones

Milestones are used to track the progress of similar issues and pull requests as they're opened and closed over time. At a glance, you can easily see the progress of work in a milestone's lifetime.

It provides a good reference to determine if the team has closely followed the benchmarks that they had set. It can be used to gauge if they are underperforming or over performing. In former case, it would prove to be helpful to re-evaluate the team's potential and set practical goals. This will in turn result in better individual and overall performance. As for the latter case, the milestones may have to be restructured as it is probably not challenging enough for the team members to explore their full potential. The information for a single milestone include "id", "url", "creator", "title", "description", "create_at", "closed_at", etc.

2.3 Collecting Commits

There are several commits in a project. They help individual team members to save their work and update the existing version of the code.

The commit history can help track when and by whom was the code updated. It also represents the amount of contribution done by each team member in the project. Thus analyzing the commit history can help

determine the distribution of workload amongst the team members. Skewed distribution of workload means that the work has largely been dominated by a few team members and thus there is a high probability that collective decisions have not been made. This can be a potential bad smell and tracking commits can thereby help detect them. Common information for a commit includes "url", "author", "committer", "parents", etc.

2.4 Collecting Labels

Labels are the tags for the issues, through which the developer can classify their issues. Labels are used to sort and describe issues and pull requests. Some labels are usually reserved for one or the other, though most labels may be applied to both. Tracking these labels help determine which types of issues were faced by the team during the project. It can help the team determine which areas of work to focus upon.

If the team does not assign labels to the issues, or if the issues are repetitive, then there is a chance of a potential bad smell. If there are no labels the team may not be able to identify weak areas. Also repetition of labels may mean that the labels are not correctly assigned and they are not problem specific being generic ones. By tracking the labels in the repository we are able to track these bad smells. By extracting the data from GitHub we are able to obtain following attributes related to the labels: name, url, color.

3. DATA

The following table shows how much data we collected for analysis:

No.	Data set	Project1	Project2	Project3
1	Commit Record	123	198	164
2	Issue Record	57	90	87
3	Milestone Record	7	9	10
4	Labels	7	16	17

Table 1: Group Statistics

4. DATA SAMPLES

4.1 Commit Record

Committer	email_id	Commit time(epoch secs)	Comment
C0	E0	1457920414	Windows telemetry added
C1	E1	1455249434	get the base clipboard to my branch

Table 2: Commit Record

The commit record mainly focuses on the commit history during the development of software. It contains the Committer name, email id, commit time and comment which describes what the commit is about.

4.2 Issue Record

Issue Id	Description	State	Creator	Create time	Labels	Milestone due	Last update
45	Auto-Logging in the cloud	closed	C4	1455848038	['enhancement']	-1	1456374035
37	Create a log file	closed	C3	1455847983	['help wanted']	-1	1456887060

Table 3: Issue Record

The issue records are used to trace the issue publication in the development of the software. All Milestone due with values -1 indicate that no due date is set for them.

4.3 Milestone Record

Milestone id	Title	Create time	Due
3	March 1 Coding Coding Coding	1453431471	1456808400
13	Design and Conduct Experiment	1456799831	1458446400

Table 4: Milestone Record

The above table shows the attributes like milestone Id, title, create time, due. The milestone Id uniquely identifies each milestone and is usually on auto increment mode. The title provides a brief idea about the milestone. The create time helps determine when the milestone has been created and the due time tells by what time the work is to be completed. The difference between due time and create time helps determine the time duration assigned to each task.

4.4 Label Record

Name	URL	Color
bug	https://api.github.com/repos/nikign/Git-Helper/labels/enhancement	fc2929
duplicate	https://api.github.com/repos/nikign/Git-Helper/labels/help%20wanted	cccccc

Table 5: Label Record

The above table shows attributes name, url and color. The name helps to determine what the issue precisely was. Again the url helps to navigate to the specific issue. This facilitates ease of identification of these issues with one color assigned to each label.

5. ANONYMIZATION

The project is about analyzing the potential problems in development and not reveal and critique group's capability or performance.

In this report, all of the names will be hidden to protect the privacy. That is, developers are called "User1", "User2", "User3"; project groups are called "Project1", "Project2", "Project3" and so on. We defined a mapping method to substitute the real names and email ids.

6. FEATURE DETECTION AND RESULTS

6.1 Weekly commit distribution

The commit distribution was fetched through the commit record for project 1. It represents how many commits the team members made collectively for every week. At this point, we ignore individual committers. Since the commit time is given by epoch, we calculated the time elapsed since a particular commit time and returned time value in seconds.

Ideally, the commits should be uniformly distributed across all the weeks. However, it is interesting to note that the number of commits are comparatively larger around the delivery dates.

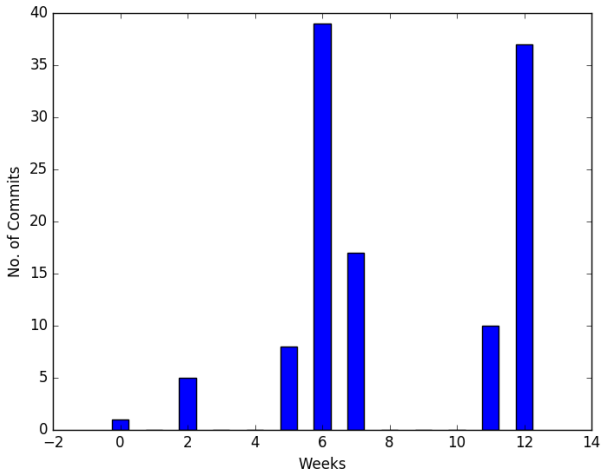

Figure 1. Project 1 weekly commit distribution

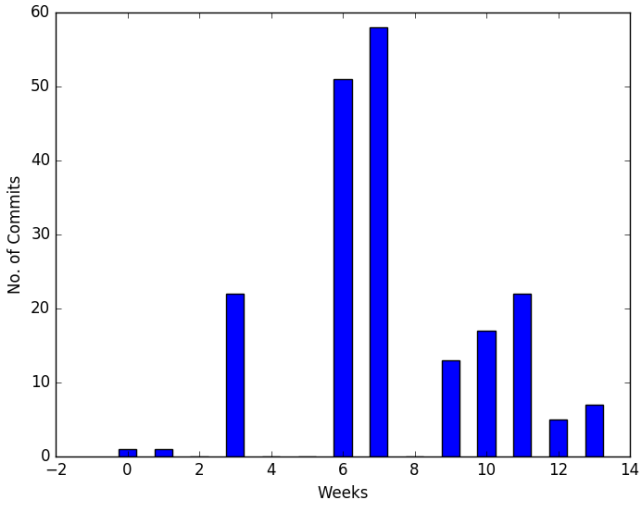

Figure 2. Project 2 weekly commit distribution

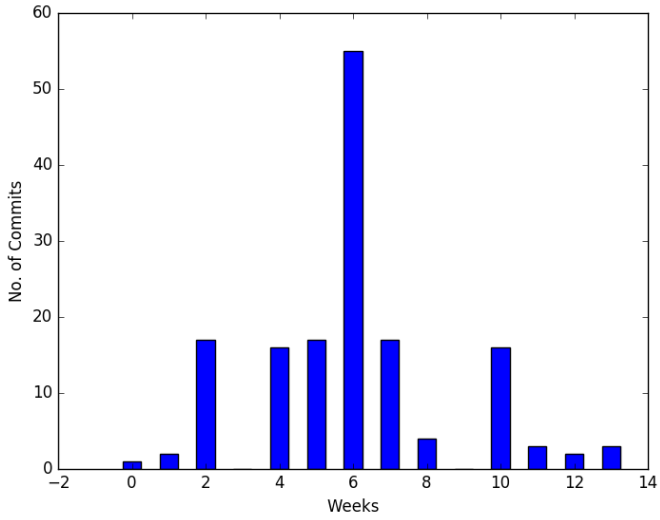

Figure 3. Project 3 weekly commit distribution

6.2 Personal commit distribution

Another important feature from the commit history is the total number of commits for one single person.

Under ideal conditions, each member should have equal contribution. This means that each team member has equally contributed to project development. However, practically this is not always the case as can be observed from the graphs below. There is a difference in each individual's technical skills and learning curve. Additionally, people use GitHub differently i.e. some developers use it to for daily backups. Some users make commit after completing one individual functionality. There are also a few people who have not much expertise with GitHub and do not use it to its full potential, but are actually contributing to the project equally. Thus, the results obtained through this metric are not always true representative of each member's efforts.

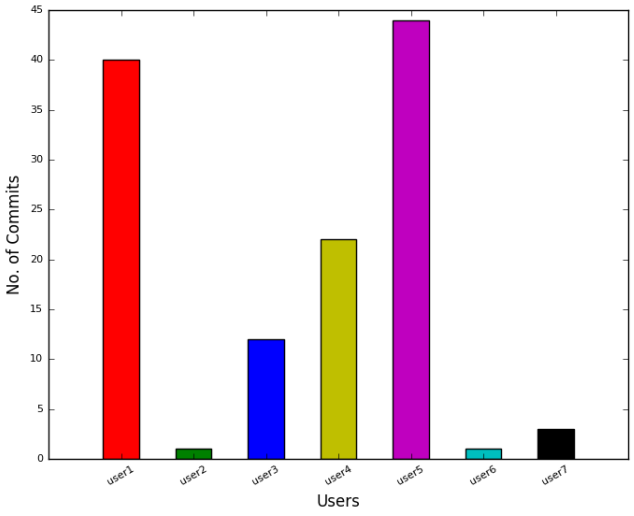

Figure 4. Project1 personal commit distribution

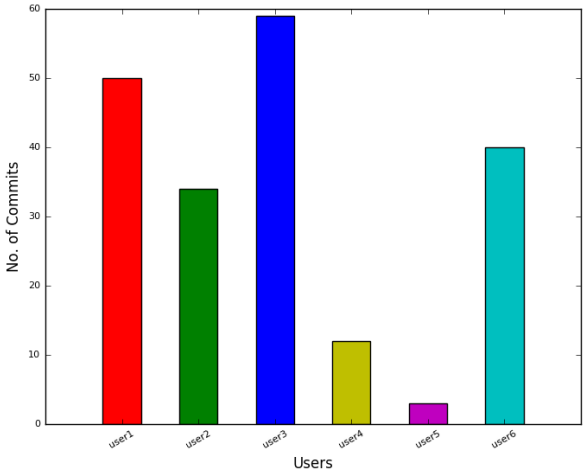

Figure 5. Project 2 personal commit distribution

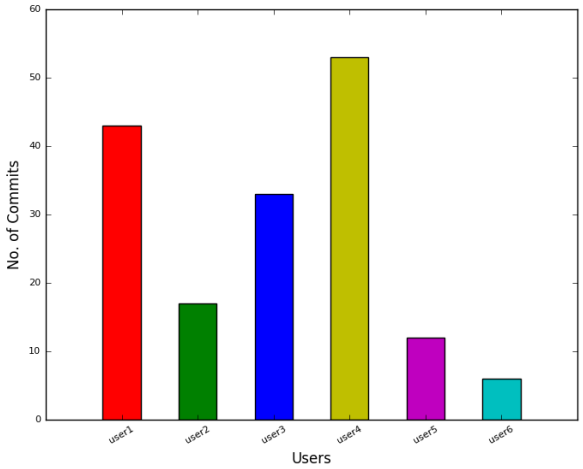

Figure 6. Project 3 personal commit distribution

6.3 Issues not closed

The open issues represent the tasks which are currently under progress for any project. Thus, it is logical to assume that for a completed project, all the issues should be closed. An open issue for any project that is not currently under development indicates that the project faced some untoward situations or change of requirements at a later stage which made the issue obsolete. In latter case, these issues should be marked as "invalid".

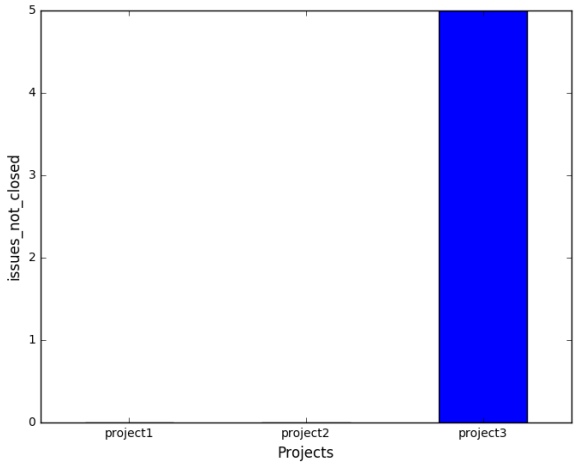

Figure 7. Issues not closed

6.4 Issues Overdue

The issues need to be handled in time. Timely redressal of issues ensures smooth flow of project activities. However, if the issues are left unresolved, then they may interfere with other activities and create problems in future. Thus if we identify that a team has not closed it's issues in time, it is indicative that the team is not following proper schedule or in worst case "not scheduling properly". It might also be the case that the team has carried out some activities which might have been affected by open issue and have gone unnoticed. Thus a potential bad smell. From the below graph, it can be seen that the project 3 has done maximum issues overdue and there is a high possibility of their code containing bugs due to this.

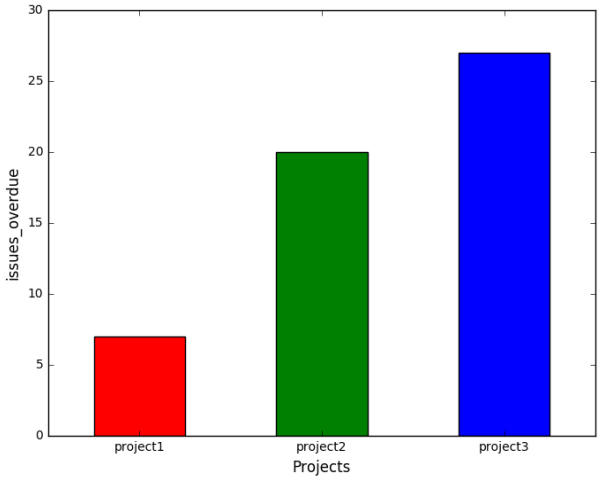

Figure 8. Issues overdue

6.5 Issues without description

Issue description is the summary of what a particular commit is about. It is used as a technical documentation which will provide some insight about a particular functionality which is being implemented.

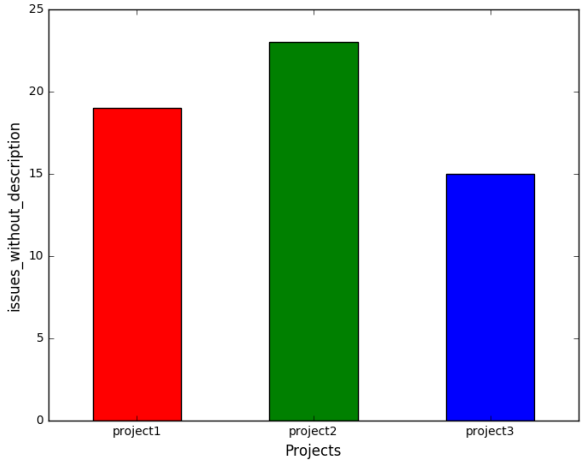

Figure 9. Issues without description

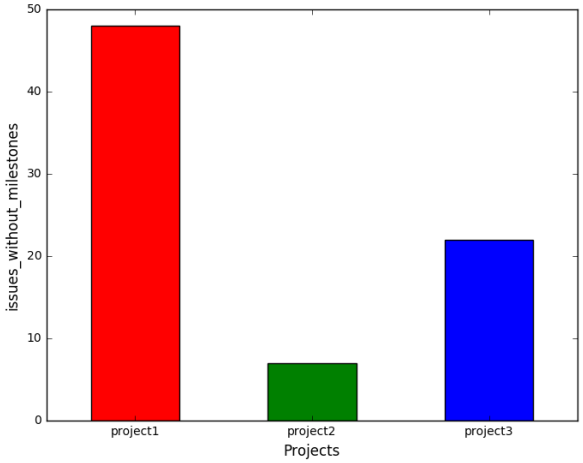

Figure 11. Issues without milestones

6.6 Issues without Label

Labels help identify the type of the issue. This helps in categorizing the issues under different labels. Thus while resolving them, the ownership of different type of issues can be assigned to different members of the team. This way there is no confusion as to who will handle which issues. However, if the issues are unlabeled, then the person to whom the issue is assigned might not know exactly what type of problem is associated with the issue and may find it difficult to solve. It may be possible that the issue is closed without solving all the problems associated with it thus creating problems in the future. This is thus a potential bad smell. The figure below shows that project 1 and project 3 have created many issues without labels. There is a high chance that some of their issues have been closed without completely resolving them.

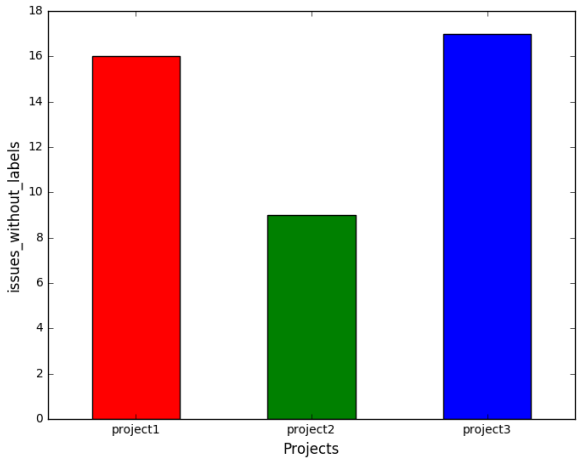

Figure 10. Issues without labels

6.7 Issues without milestones

After recognizing an issue, it is desirable to have an estimate of when it should be completed to achieve project milestones. Otherwise, developers might spend more time than they would have otherwise. Also, for reference at a later point of time, these can be used to indicate how much time was spent on a particular issue vs how much time should that issue ideally consume.

6.8 Issue Creator

Issues created by different team members shows that all team members are actively involved in the project activity. This is because issues arise only when you start working on a topic. If only one member creates issues, it is indicative of dominance by that team member in the project. If all team members contribute evenly, then more issues can be uncovered. However if only a few team members are involved in issue creation, there is high possibility that some of the issues stay undiscovered. This is a potential bad smell and can create magnified problems in the future. Project 2 and Project 3 graphs show that only one person has created most of the issues. Project 1 shows a relatively even distribution.

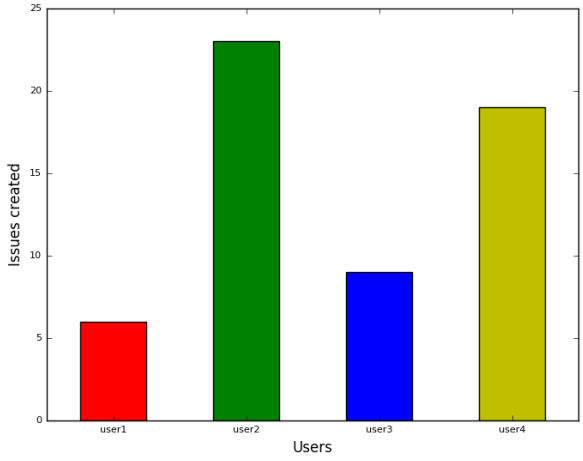

Figure 12. Project 1 issue creator distribution

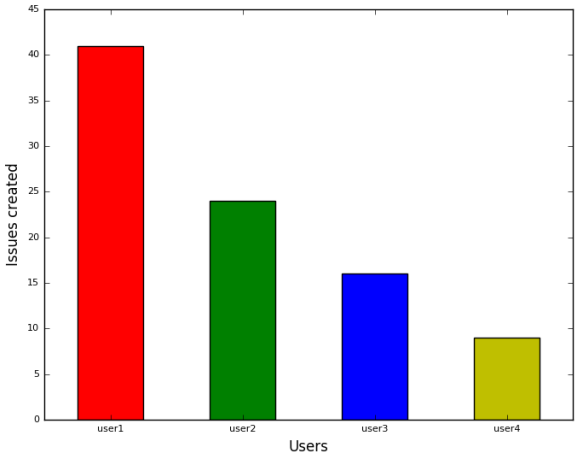

Figure 13. Project 2 issue creator distribution

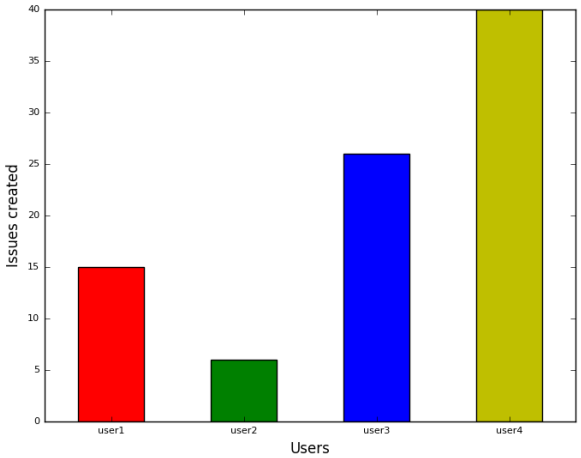

Figure 14. Project 3 issue creator distribution

6.9 Issue Label distribution

It is always a good practice to have as many labels as possible to represent more precisely a type of particular issue. Generally speaking, the number of bugs and duplicate issues should be low at every phase. However, many labels have some ratio which can be said to be good for that phase of the development cycle.

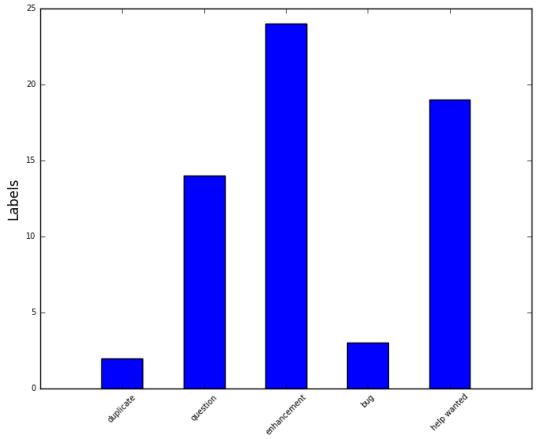

Figure 15. Project 1 label distribution

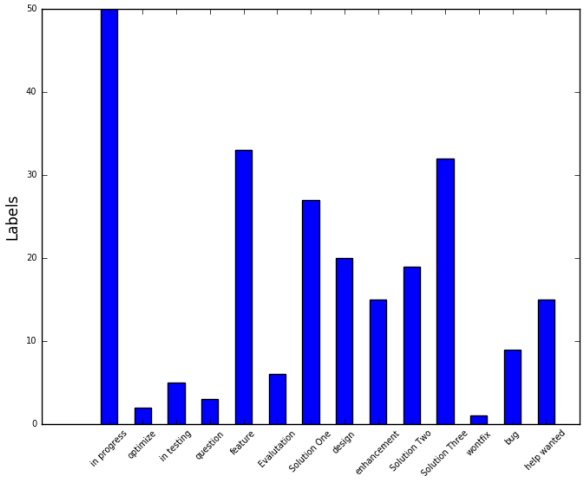

Figure 16. Project 2 label distribution

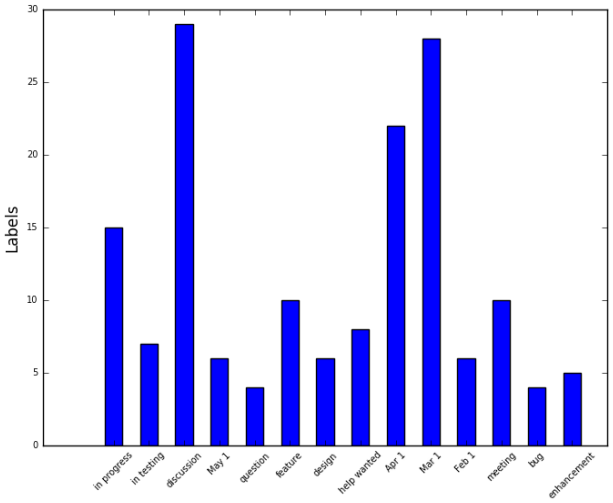

Figure 17. Project 3 label distribution

6.10 Issues Closed Without Comments

Issues are often created to describe a particular blocker, a bug or anything that is hampering the progress of the project. When the developer resolves the issue, it is closed by the person responsible for the resolution. Putting the comment while closing or re-opening the issue helps other users or future stake holders to understand what exactly was wrong with the code. So, the issues which are closed without putting an appropriate comment can be misleading, hence considered as a bad smell.

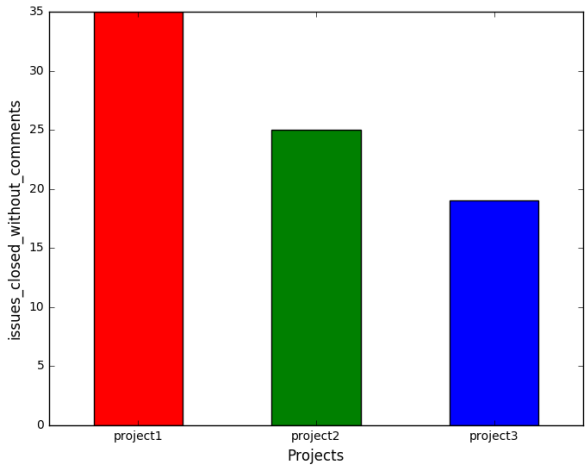

Figure 18. Issues closed without comments

6.11 Weekly Issue Distribution

Issues can be encountered at different stages of the project. However, if more issues are created towards the end of the project, it means that the team has not planned work properly. They could not do issue identification in the early phases of the project. Issues in the final phases have a high likelihood of not being dealt in a proper way. This may leave some unsolved issues in the project. As seen from the figures, project 1 has some issues in the last week. Other than that, other projects seem to have managed issues well.

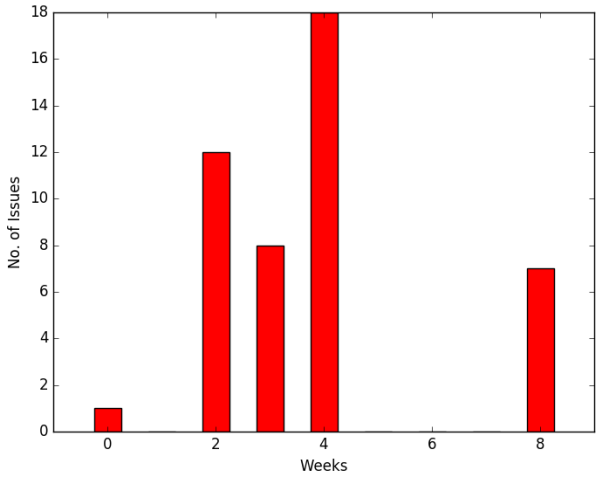

Figure 19. Project 1 weekly issue distribution

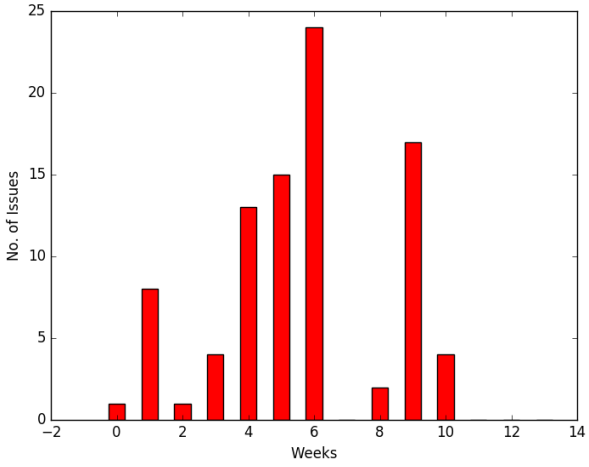

Figure 20. Project 2 weekly issue distribution

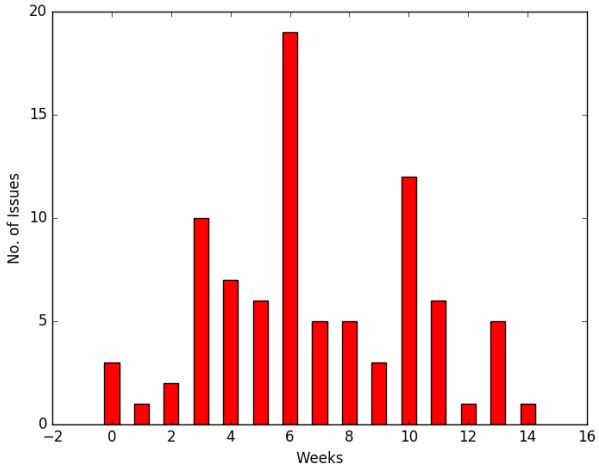

Figure 21. Project 3 weekly issue distribution

7. BAD SMELLS DETECTOR AND RESULT

7.1 Sporadic activity

As it can be seen from figure 4, 5 and 6 that the groups have not been consistently committing in their GitHub repositories. It is a clear indication that the teams were not consistently working on the project. It is indicative of the poor project management policies used by the teams. This results into project tasks done in a pressure situation to meet intermediate deadlines which may cause errors to creep in and flaws in the system functionality.

7.2 Poor issue management

The teams have failed to manage issues properly. There can be seen that there are still 5 open issues for project 3 which indicates that either the team has failed to meet the required objective and under-delivered or the issues have become irrelevant due to change in requirements. As it can be seen the project 3, since the project 3 has many unclosed issues, there is a high probability that the team might have faced problems in the later stage of the project.

Figure 22

7.3 Failed Targets

Some of the project groups could not solve the issues in time. As seen from the pie charts in fig. 21, project 3 has highest number of overdue issues. These issues which remained unsolved create problems in later part of the project. They may interfere with other parts of the project and may cause malfunctioning which may perhaps trigger other issues in the future. Some of the previously unsolved issues covertly degrade the project quality and the act as a slow killing knife.

Figure 23

7.4 Poor subject Interpretation

Issues are used to communicate the tasks that the developers have to perform in order to develop certain functionalities or achieve some targets. We recognize that almost all the teams have failed to recognize this. The problem is that the issue remains open ended and blurs the understanding of the requirement. If a member misinterprets some requirement while the creator of the issue has certain assumptions about the implementation, it may cause many problems in the projects. Thus, it is very important to provide granular details of the task and state assumptions and expectations clearly.

Figure 24

7.5 Unclassified Tasks

Providing proper labels to the issues help classify them. This way they can be dealt in a better way. Also the tasks can be better assigned to the team members. People in the team can take up ownership. Also the owner of the issue knows what actions are expected in this issue. The fig. 23 shows that project 1 and project 3 have maximum unclassified tasks. Hey have not allotted proper labels to the issues. Hence it is possible that they might have faced the problem of solving issues in later stage of project.

Figure 25

7.6 Poor Time Management

It is always good to assign milestones and provide a deadline to tasks. If the issues do not have a proper milestone, then they may not be dealt in a serious way. There may be a problem that they exceed the expected deadline. Figure 24, shows that project 1 has maximum number of issues which do not have any milestones. This group is most likely to miss the deadlines and might have ended up mishandling the issues.

Figure 26

8. EARLY DETECTION

It would be good to understand the problems which can arise in the project during the early stages rather than crying over spilt milk later. The team members can mend their ways beforehand so that they can strategize their roadmap in a much better way.

8.1 Identify the under-performer

We extracted the team member history for the first 4 weeks of the software development cycle. We made a few observations which can increase the productivity of the project and its team members.
More often than not, teams have members which under-perform throughout the project. We aim to identify those "under-performers" and take actions during early weeks to empower them.

The said team member may be deficient in technical skills or functional know-how. This identification can prove to be very useful to train them through knowledge sharing and internal training sessions so that every team member is on the same page. This can even be used to identify the lazy members who just don't care enough to perform.

The following table represents the commit numbers for project 1:

User	week 1	week 2	week 3	week 4
user 1	2	5	4	7
user 2	0	0	1	3
user 3	0	1	2	5
user 4	3	2	10	4

Table 6 : Statistics for first 4 weeks

The line graph below is a clear representation which helps to compare the team members commits for the initial weeks. It is quite evident that the user 2 had not made any contribution until the third week.

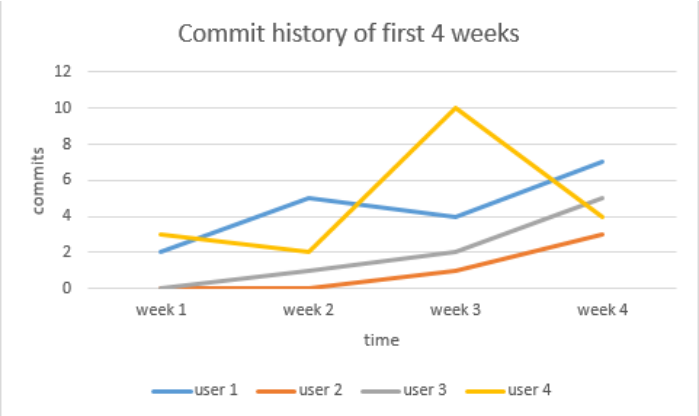

Figure 187. Graph for first 4 weeks commit data

9. CONCLUSION

In this project we have we have successfully mined the data abstracted from Github repository. The raw data about the team activity available on the Github is useful only if some useful inferences can be drawn out of it. In this project we have developed reusable scripts which when applied on the Github repositories can help the team identify the loopholes in their project planning. This will act as a good project management tool and help in early detection of bad smells thereby preventing future loss.