Bad Smells of Github Project

Zhe Yu, Di Chen
Amritanshu Agrawal, Shijie Li
Com Sci, NC State, USA
{zyu9, dchen20, aagrawa8, sli41}@ncsu.edu

ABSTRACT

Github is an excellent online project managing and communication platform. Nowadays, most of the open-scource software developments are conducted on github. As a result, there has been an increasing need for the evaluation of software projects according to github activities. In this project, we propose an analyser for github activities along with an early detector of bad smells based on the data of 14 group projects conducted on github from January 1st to April 7th, 2016.

1. INTRODUCTION

Github repository has become the latest star of data mining object. With more than 14 million users and 35 million repositories, lots of interesting information can be retrieved from mining the repository [1–3]. Among all the information extracted from github repositories, we are most interested in the bad smells of a software project. With the help of a bad smell detector, we will be able to evaluate a project after it is finished or to alert users that they are doing something wrong at early stage.

To build the bad smell detector, we extracted data from 14 group projects conducted on github from January 1st to April 7th, 2016. Then 12 carefully designed features are extracted from the original data for analysis. By analysing the extracted features, 4 rules for bad smells are formulated. At last, a machine learner is trained to predict the bad smells at early stage. All the work can be found in our repository [1].

2. DATA COLLECTION

The following data are extracted from the github repository of each group:

1. **Issue Activity**:
   (a) Issue ID
   (b) Issue Name
   (c) Issue Activity (labeled, milestoned, assigned, closed...)
   (d) Time for the activity
   (e) User ID for the activity
   (f) Issue Label
   (g) Issue Milestone
   (h) Issue Comment
2. **Milestone**:
   (a) Milestone ID
   (b) Milestone Name
   (c) Created Time
   (d) Due Time
   (e) Closed Time
   (f) Creator ID
3. **Commits**:
   (a) User ID
   (b) Time of commit
   (c) Message

The data collector is gitable-sql.py [2], which is a modified version of duh102's gitable-sql.py [3]. The original version is gitable.py [4].

The updates of gitable-sql.py are

1. Fix bugs associated with deleted milestones and issues.
2. Add a time bound for data collection. Data collected are all within the time bound.

With the help of time bound, two data sets are collected, one containing all the information from January 1st to March 3rd, the other containing all the information from January 1st to April 7th. Analyzer runs on the second data set while early bad smell detector runs on the first.

3. FEATURE EXTRACTION

A total number of 12 carefully designed features are extracted based on our engineering decisions. These 12 features cover all the four bad smells in Section 4.

3.1 Number of Issues

We first collect all the issue number for all the group. As you can see in Figure 1. Most of the groups have more than 50 issues posted. The groups with issue number smaller than 40 will be treated as bad smell.

3.2 Comments per Issue

[1] https://github.com/azhe825/CSC510/tree/master/BadSmell

[2] https://github.com/azhe825/CSC510/blob/master/BadSmell/gitable-sql.py

[3] https://github.com/CSC510-2015-Axitron/project2/blob/master/gitable-sql.py

[4] https://gist.github.com/timm/a87fff1d8f0210372f26file-gitable-py

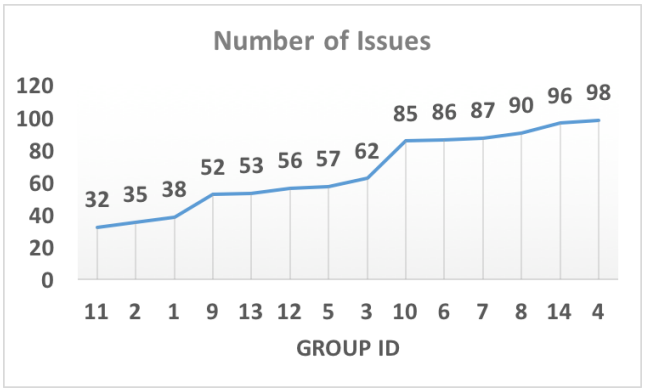


Figure 1: Issue Number

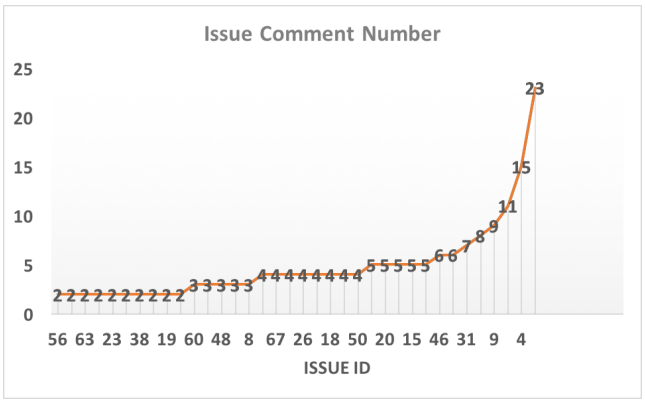


Figure 2: Issue Comments Number

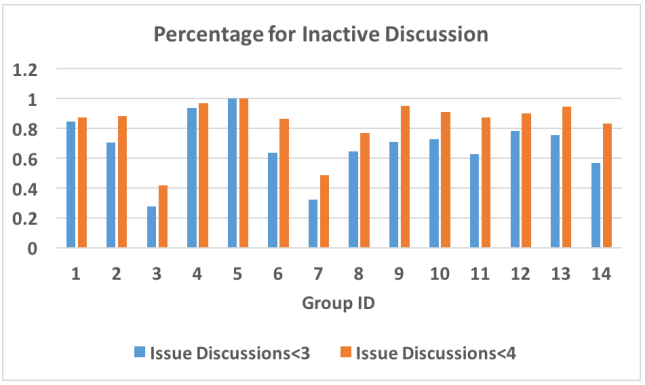


Figure 3: Comments Activeness for Each Group

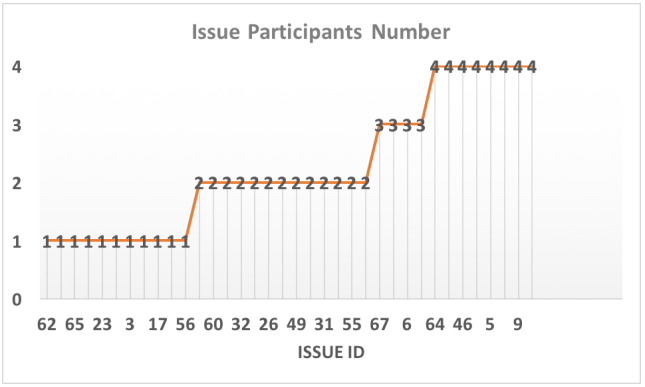


Figure 4: Issue Participants Number

The comments number for each issue is a good reflection for the activeness for the group communication. As you can see in Figure 2, it is the comments number for all the issues posted in one group. We rank the issue based on the issue comment number. From Figure 2, we can learn that the smaller the percentage of issues with less than 3 comments the better their communication is.

Figure 3 is the percentage of issues with less than 3 or 4 comments for all the groups. It is obvious that group 3, 7 communicate better than the other groups.

3.3 Participants per Issue

For this feature, we collected how many users participate in each issue. Figure 4 is a good example from one group. It is straightforward that a group has a bad team engagement if a group has a lot of issues with less than 2 or 3 participants involved.

After collecting all this feature for each group, we put the result into Figure 5. Consistent with our previous finding in Figure 3, group 3 and 7 have the best team communication, because most of their issues have more team members involved in the discussion.

3.4 Issue Durations

Issue Duration means the day since one issue is open to the day it is closed. If the duration for an issue is too long, it is probably because their bad management on the issues. We can see if one group manage their issues well, they also will make their progress quite well.

Figure 6 is the issue duration in day for one group. We can see that some issues last more than 30 days, which indicates the problems in their team management. While Figure 7 shows the percentage of issues with duration more than 20 and 30 days. The smaller the better.

3.5 Issue without Labels

Using different labels for issues is treated as a good practice for better co-operation and communication in Github. Taking this into consideration, we extract all the issues and calculate the percentage of issues without labels for each group. Figure 8 how the percentage of issues without labels for each group. The smaller the percentage, the better they use the Github.

3.6 Issue without Milestones

Similar with the previous one. Assigning each issue with a milestone indicates how well the group plan their work. Figure 9 shows the percentage of issues without milestones for each group.

3.7 Delayed Issues

We collect the percentage of issues that are closed after the due date of their milestone.

As shown in Figure 10, a good group should have all their issues closed before the due date. But that doesn't happen all the time. Some issues are closed long after the due date. The smaller such issues, the better the management. Figure

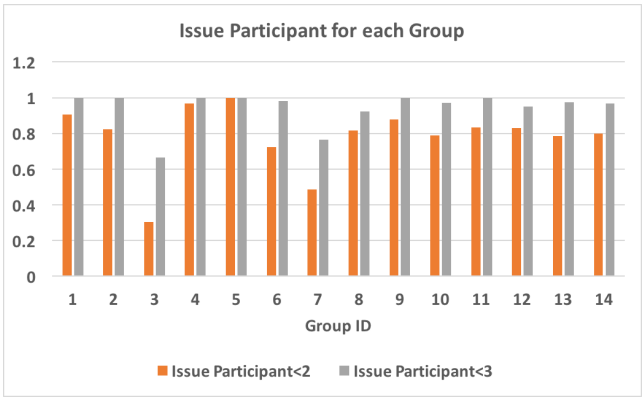


Figure 5: Issue Participants for Each Group

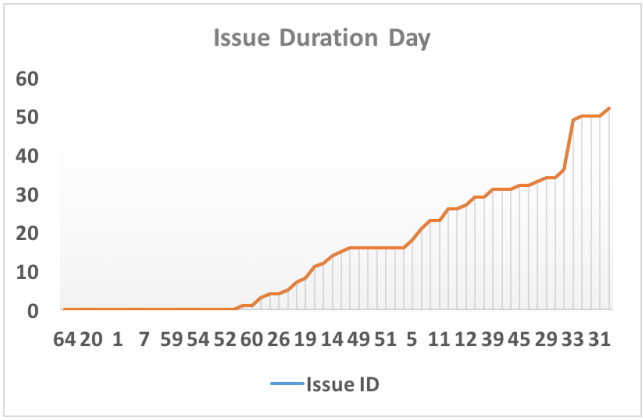


Figure 6: Issue Durations

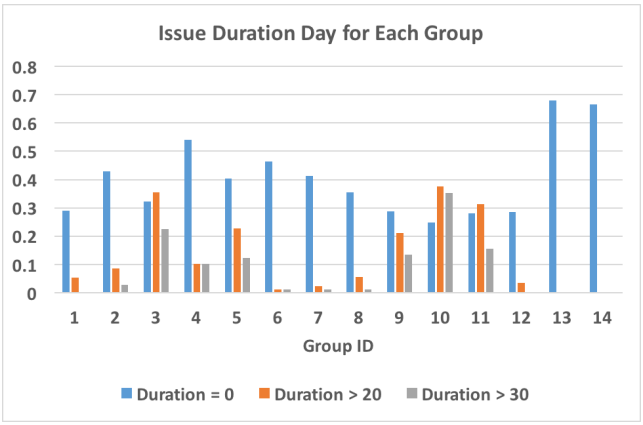


Figure 7: Issue Open Duration for Each Group

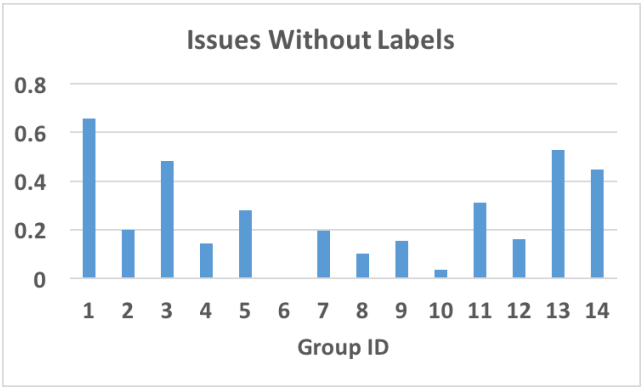


Figure 8: Issue without Labels

11 shows the overall performance among the groups.

3.8 Milestone without Due Date

Milestones are used to make the plan or schedule and track the progress of the project by associating with related issues. One of the important properties in a milestone is the due date setting. We checked the due date setting for milestones in each group in order to study their project planning. We found some group have milestones without due date, which really affect their effective planning or progress.

3.9 Unclosed Milestones

Besides due date, close date can reflect the implementation of the plan. After the problems in related issues are solve, the corresponding milestone is supposed to be closed. If milestones are found without closed date, we assume that they either delayed the implementation or abandoned the plan.

3.10 Silent User

A user is called "Silent" if he or she have less than 50% of their group's average comments. Average is calculated by (Total number of comments / number of group members)

3.11 Relaxed User

A user is called "Relaxed" if he or she have less than 50% of their group's average assigned issues. Average is calculated by (Total number of issues with assignees / number of group members)

3.12 Nonlinearity of Commits

The commits in Github are supposed to increase smoothly over the duration of the whole project. A good planned and implemented project should have a linearly increasing number of commits accumulated over time, which reflects that members in the group are concerned to the project and actively involved in the design and work. Therefore, we plot the accumulated number of commits for each group over time to study their planning and progress. Then the non-linearity for each group is calculated by the variances of commit number against the straight line that connects the initial number of commits and the final accumulative amount of commits when the project is due. Since each group made different total number of commits, we also normalized the variances by the amount of commits so that it can reflect the pace of the progress in that group.

4. BAD SMELLS

Bad smells are indicators that tell users there might be something wrong with the project. We carefully designed four bad smell detector by analysing the distribution of features in Section 3. The basic form of the bad smell detectors is a weighted sum of the feature score it is related to.

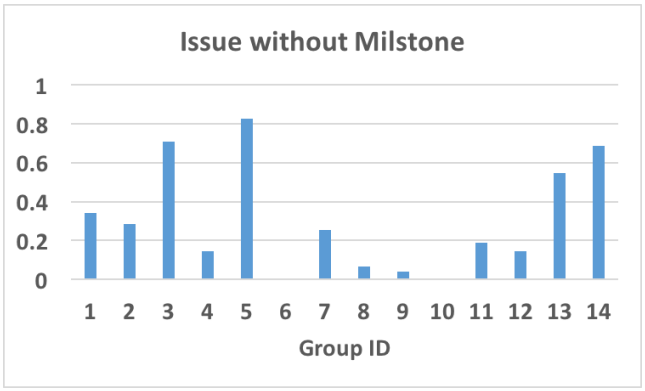


Figure 9: Issue without Milestones

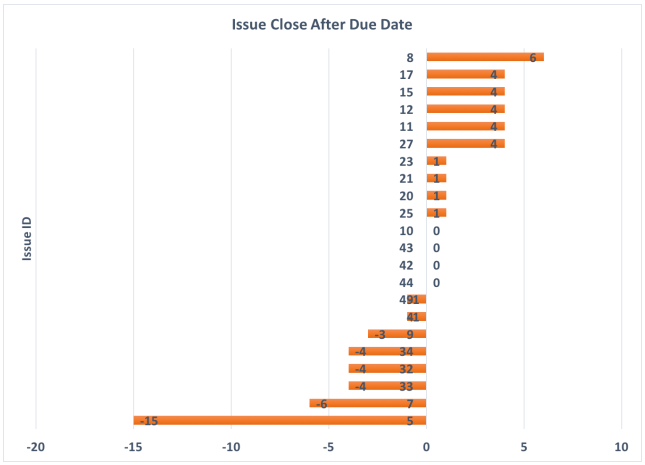


Figure 10: Issue Close Speed

$$BadSmell_j = Sign(\sum_{i \in F_j} Weight_{j,i} FeatureScore_i - Thres_j), \quad (1)$$

where F_j is the set of features related to $BadSmell_j$.

4.1 Lack of Communication

Lack of Communication is the bad smell indicating that the members do not communicate frequently through issues. This bad smell is related to features listed below:

1. Comments per Issue
2. Participants per Issue
3. Silent User

Figure 13 shows the feature values related to Lack of Communication bad smell. According to our rule, Group 1, 4, 5, 9, 10, 11, 12, 13 have the problem of Lack of Communication.

4.2 Delayed Delivery

Delayed Delivery is the bad smell indicating that some issues are finished after the due date of the milestone associated. This bad smell is related to features listed below:

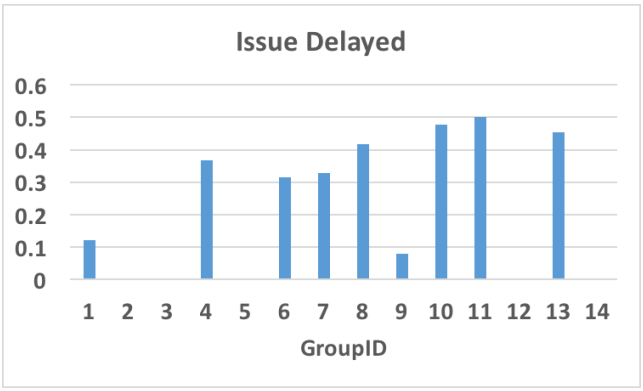


Figure 11: Issue Close Speed

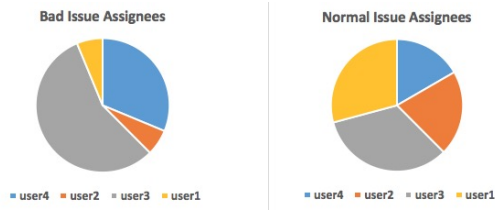


Figure 12: Issue Assignees

1. Delayed Issues
2. Unclosed Milestones

Figure 14 shows the feature values related to Delayed Delivery bad smell. According to our rule, Group 4, 5, 6, 8, 10, 11, 13 have the problem of Delayed Delivery.

4.3 Unbalanced Contribution

Unbalanced Contribution is the bad smell indicating that some users contribute much while others contribute very little. This bad smell is related to features listed below:

1. Silent User
2. Relaxed User
3. Comments per Issue
4. Participants per Issue

Figure 15 shows the feature values related to Unbalanced Contribution bad smell. According to our rule, Group 5, 10, 12, 13, 14 have the problem of Unbalanced Contribution.

4.4 Poor Planning

Poor Planning is a mixed bad smell indicating that the project is not well planned in the first place. This bad smell is related to features listed below:

1. Number of Issues
2. Issue Durations
3. Issue without Labels
4. Issue without Milestones
5. Milestone without Due Date

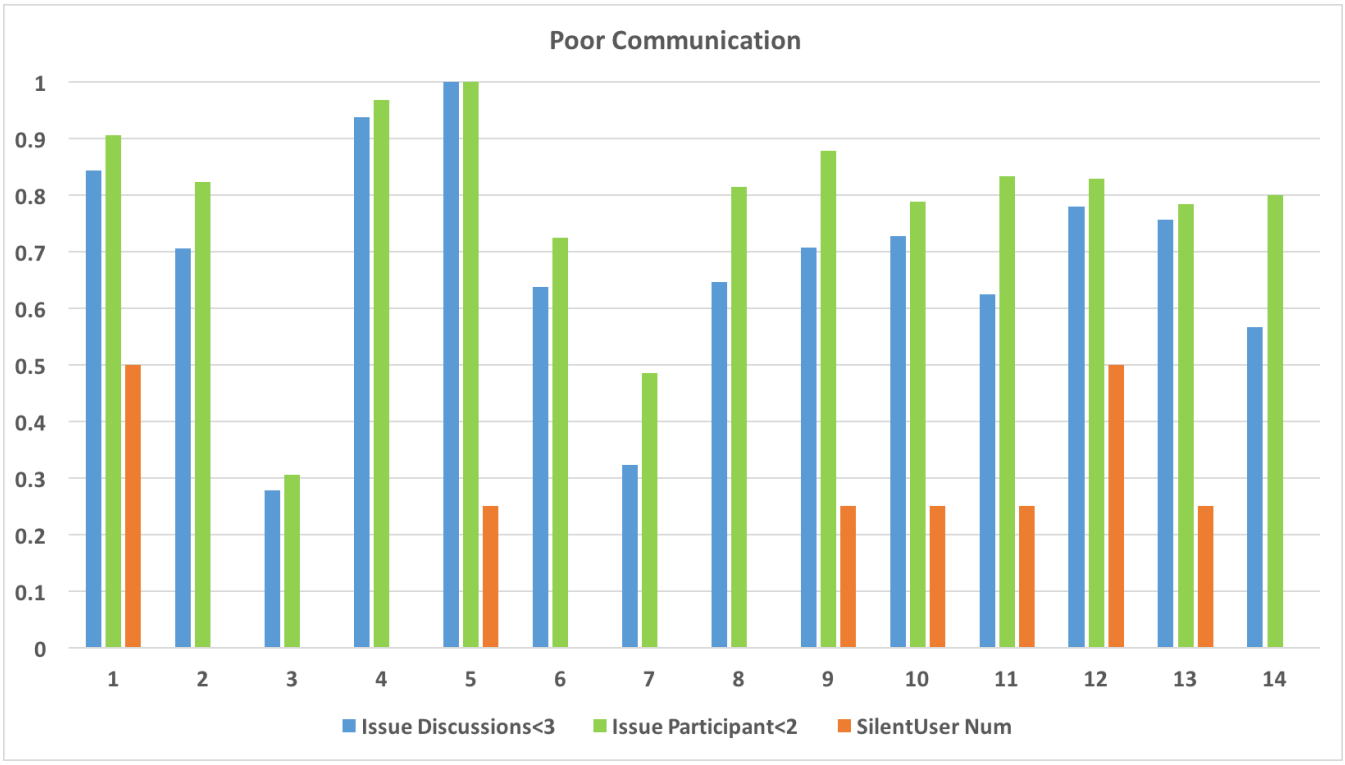


Figure 13: Lack of Communication

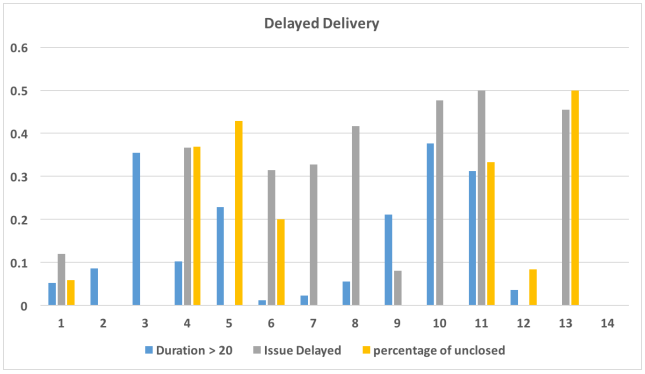


Figure 14: Delayed Delivery

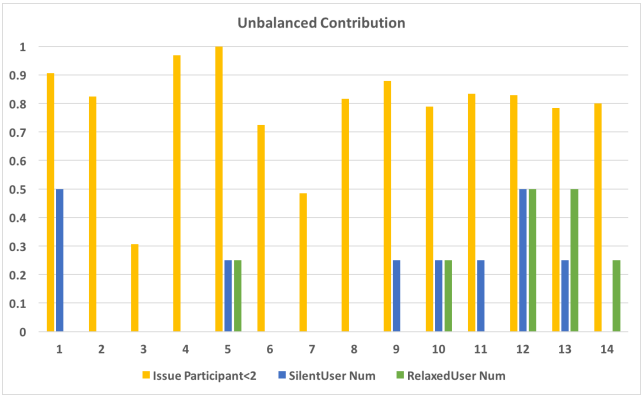


Figure 15: Unbalanced Contribution

6. Nonlinearity of Commits

Figure 16 shows the feature values related to Poor Planning bad smell. According to our rule, Group 4, 6, 7, 8, 10, 14 have the problem of Poor Planning.

4.5 Statistics of Bad Smells

Each bad smell appears in five to eight groups. Group 10 suffer from all four bad smells and Group 2 and 3 have no bad smells. Congratulations, Group 2 and 3! We probably can learn a lot from Group 10.

5. EARLY DETECTOR

We designed two different early bad smell detectors, one is based on rules generated from Section 4, the other is based on a support vector machine learner. As our experiment illustrated in Figure 17, the two detectors can be complementary.

5.1 Rule Based Early Detector

A rule based early detector applies the same rule from Section 4 on the March 3rd data for bad smell detection. As shown in Figure 17(a), this detector has a pretty good performance on predicting all four bad smells.

5.2 Learner Based Early Detector

A support vector machine classifier, with linear kernel, is trained to the early detection of bad smells. The outputs of the analyser for bad smells in Section 4 is treated as the

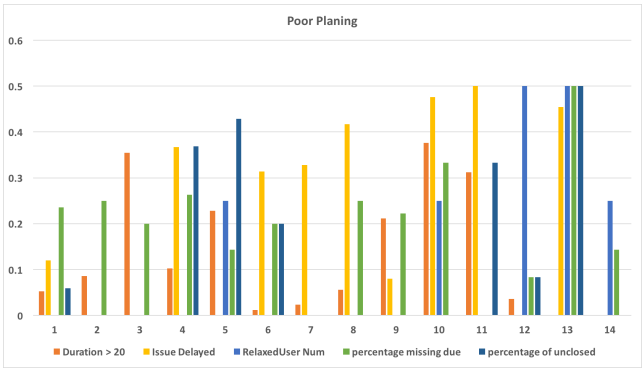


Figure 16: Poor Planning

desired labels for each group. The features extracted from the March 3rd data are utilized as the input for SVM. Half of the groups are kept as testing set and the other half with the true label given are used for training.

The result is shown in Figure 17(b) with 50 repeats. The Poor Planning bad smell can be nearly perfectly predicted from the March 3rd data set while others cannot. The reason might be that the Poor Planning bad smell has the most number of features related.

6. DISCUSSION

According to the features extracted, a set of rules of bad smells is designed along with an early detector for the bad smells. Due to the lack of training examples, the learner based early detector does not work well on the first three bad smells. However, the Poor Planning smell can be nearly perfectly predicted. On the other hand, the rule-based early detector has a satisfactory accuracy on the prediction of all four bad smells. Therefore our recommendation is the rule-based early detector, and it is also possible to use the learner based detector to predict the Poor Planning bad smell only.

Besides the satisfactory results, there are also several threads to the validity of this project.

6.1 Validity Threads

Firstly, for the analyser, is it really true for the project to have what was detected? If we find bad smells based on the rule, poor communication for example, does it really mean that the project members do not communicate enough? What if most of the communications between members are conducted off-line, face to face? Due to the lack of information, we can only get what we have.

Secondly, the rules of bad smells are formulated based purely on experience. Because of the lack of evaluation method, it is hard to tell if this set of rules is the best.

At last, since we only have data from 14 group projects, and the bad smell rules are designed strongly relying on the data, it is of high chance that our result cannot be generalized to other software projects.

7. CONCLUSION

By analysing the feature extracted from github activities, an evaluation method of bad smells is designed. Bad smells in projects indicate that the project may suffer from lack of communication, poor planning, unbalanced contribution, or delayed delivery. Having these information available would greatly benefit the group for the management of their next project. In addition, an early bad smell detector is designed for the prediction of bad smells before it is too late. If one of the alarm is triggered, it is better to rearrange everything to put things on the right track.

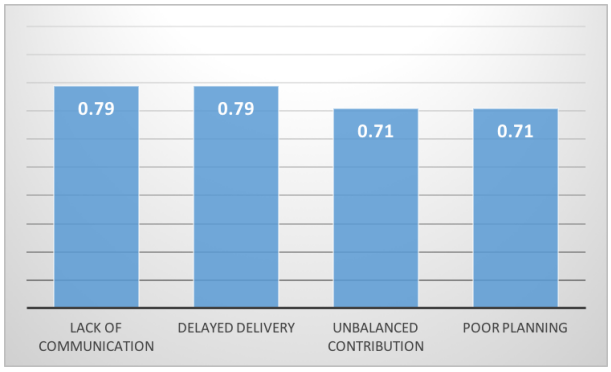
In our future work, we will focusing on the problems discussed in Section 6, explore more options and test more thoroughly on our early bad smell detector.

8. REFERENCES

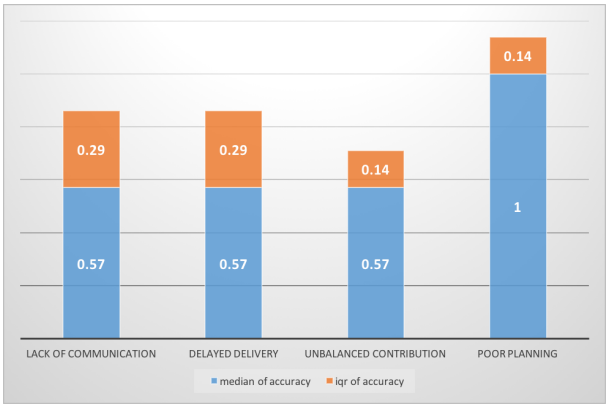
[1] M. Allamanis and C. Sutton. Mining source code repositories at massive scale using language modeling. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 207–216. IEEE Press, 2013.

[2] G. Gousios. The ghtorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 233–236. IEEE Press, 2013.

[3] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman. Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 384–387. ACM, 2014.



(a) Accuracy of rule based early detector



(b) Accuracy of learner based early detector

Figure 17: Performance of early bad smell detectors. Rule based detector has a better overall performance while learner based detector is better when predicting Poor Planning bad smell.

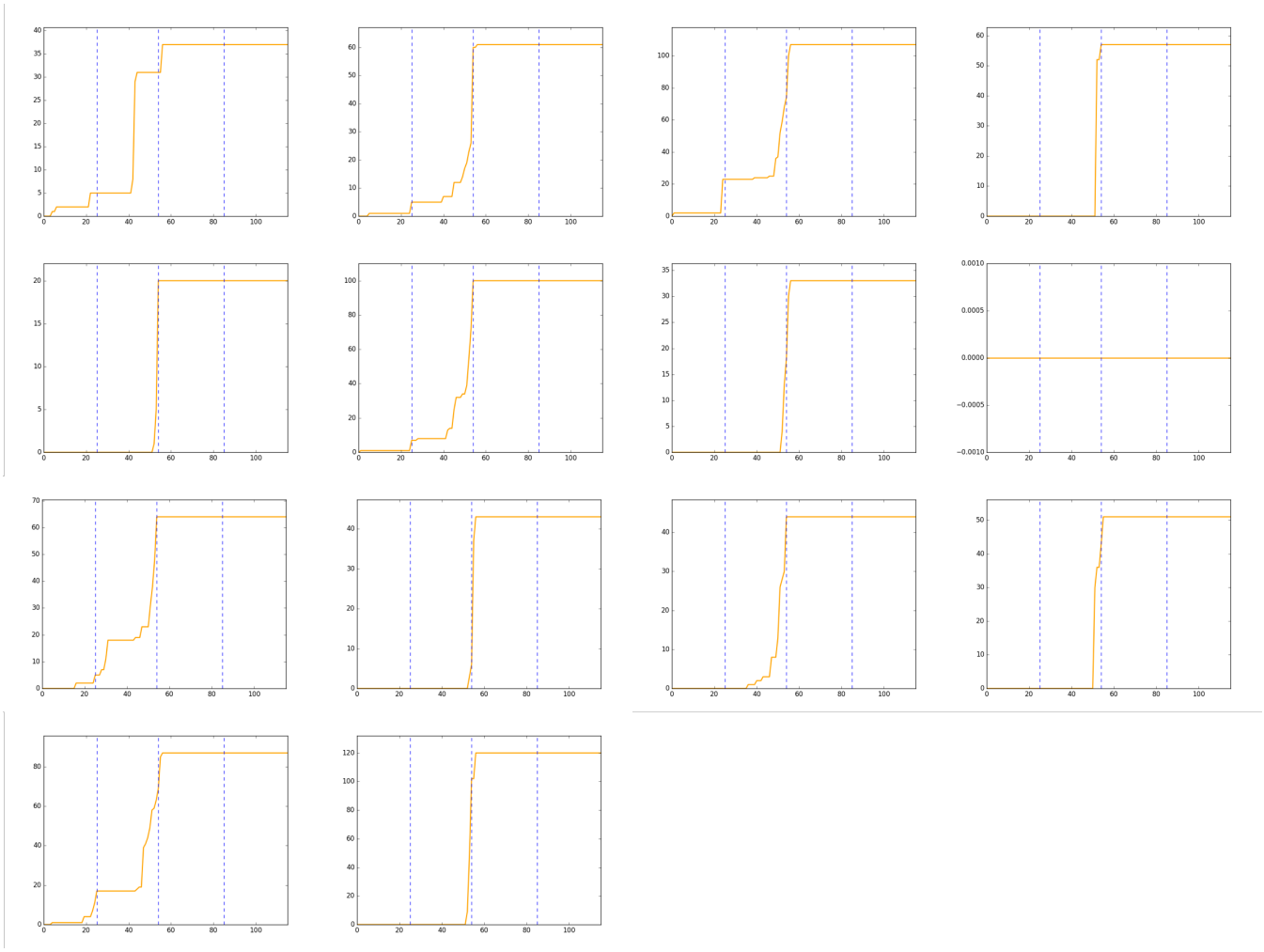


Figure 18: Commit Trend: Group 1 to 14 arranged from left to right and top to bottom, dashed line indicates Feb, March, and April dates from left to right. Group 8 never close before final due date.

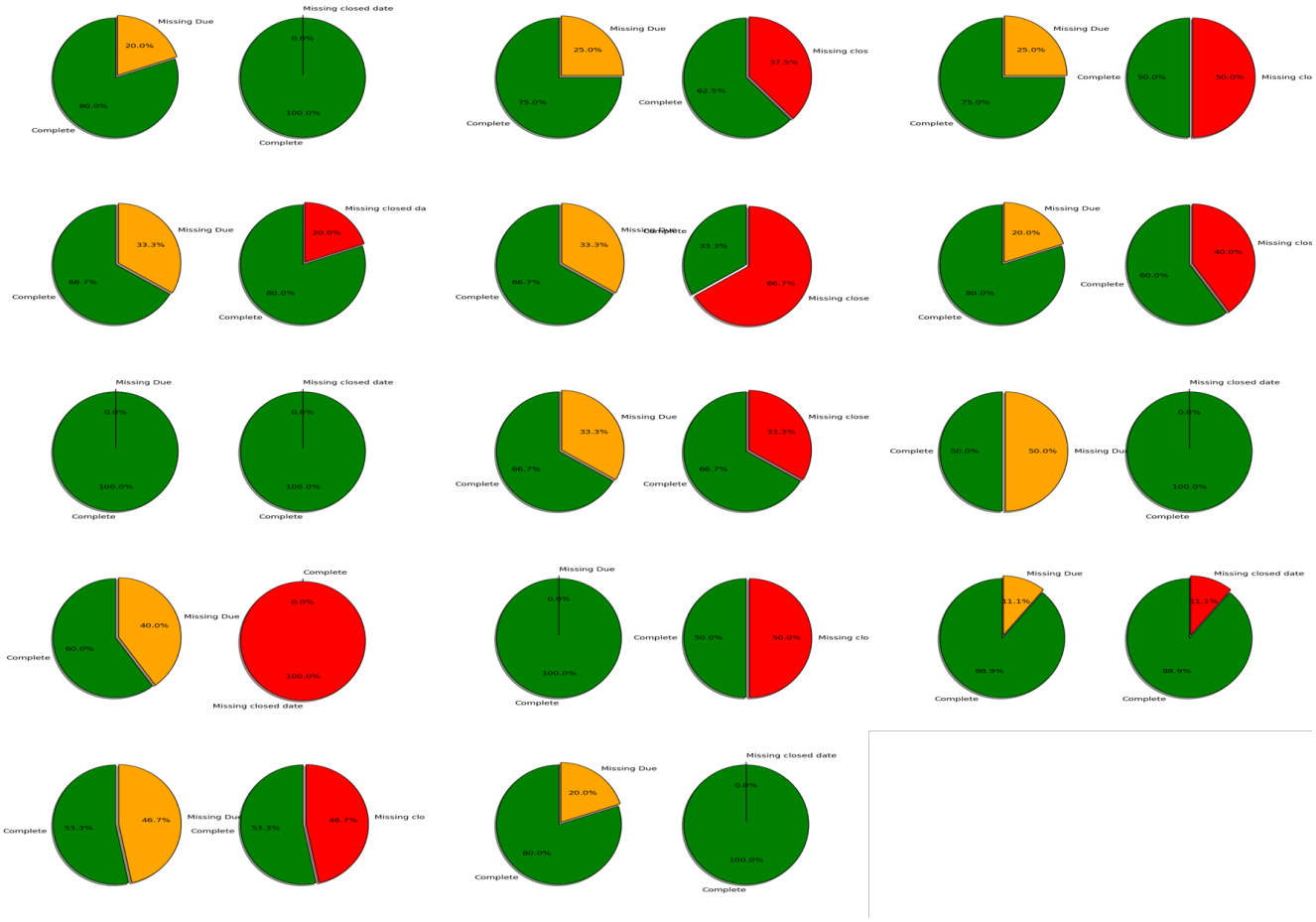


Figure 19: Milestone Usage: Group 1 to 14 arranged from left to right and top to bottom.

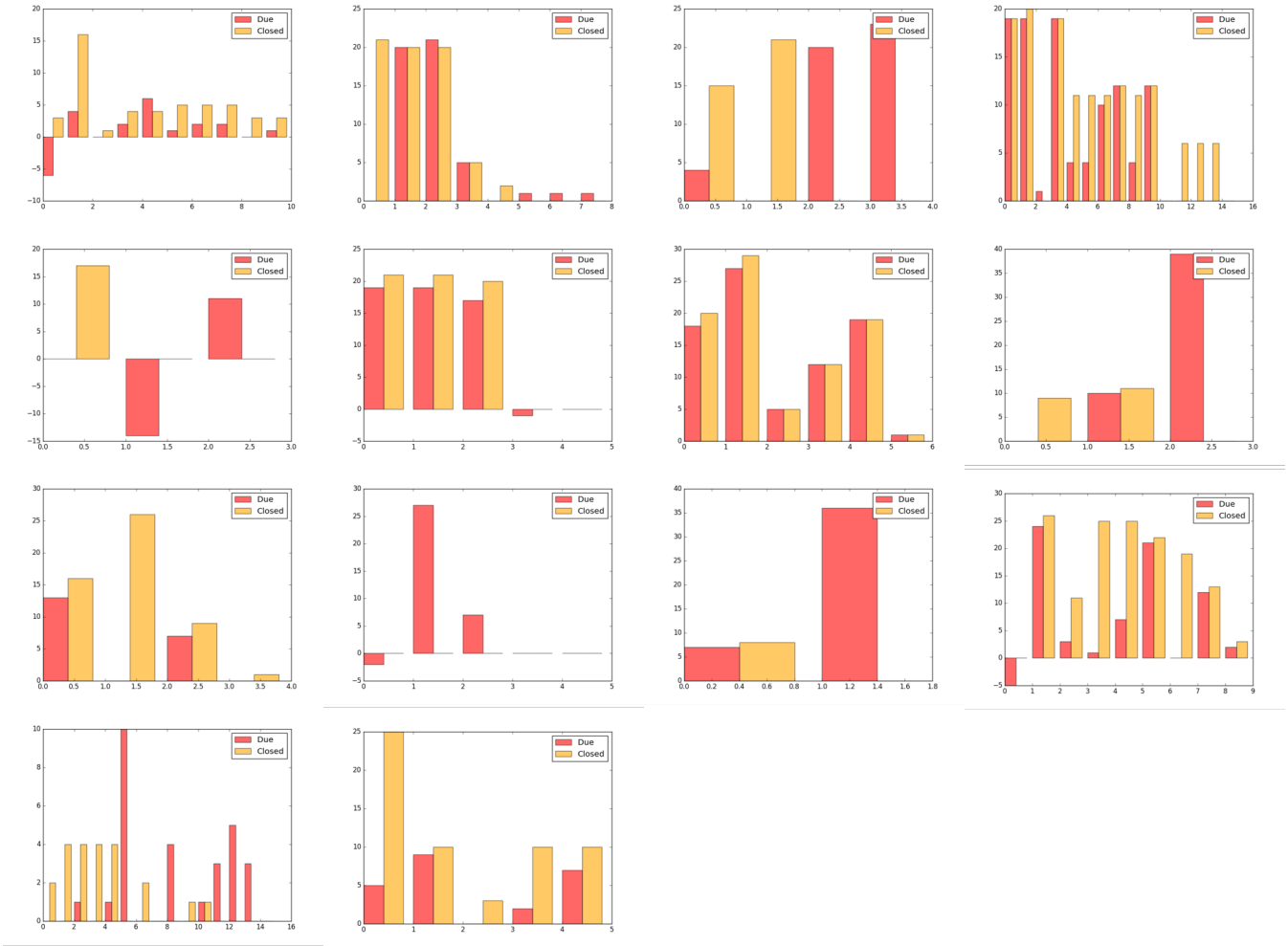


Figure 20: Milestone Plan and Progress: Group 1 to 14 arranged from left to right and top to bottom.