GitHub Bad Smells And Early Smoke Analysis By Group D

Angelyn Arputha Babu John
Department of Electrical and
Computer Engineering
North Carolina State University
Email: ababujo@ncsu.edu

Jordy Jose
Department of Computer Science
and Engineering
North Carolina State University
Email: jjose@ncsu.edu

Sameer Sharma
Department of Electrical and
Computer Engineering
North Carolina State University
Email: ssharm20@ncsu.edu

Stuti Nanda
Department of Electrical and
Computer Engineering
North Carolina State University
Email: snanda@ncsu.edu

I. Introduction

In this report we have evaluated three git repositories of the semester long Software Engineering project. Our aim is to discover trends and hints in the data collected from git that could be reasoned as a bad smell. We have preprocessed the data and parsed the data into relevant features. We also tried to predict the future trends or the usual pitfalls projects face by analyzing the data trends of three repositories.

II. Data Collection

We have collected lot of insightful data for the three repositories in question. This data is parsed to enumerate the relevant features for the task in hand. Below is the complete list of tables and columns generated from the collected data.

A. Tables and Columns

Issue

- Issue Number
- Milestone Number
- Creation Time
- Closing Time
- Label
- Title
- Text

Milestone

- Milestone Number
- Creation Date
- Due Date
- Close Date
- Title
- Creator

Pull Request

- Number

Commit

- Commit Sha
- User
- Timestamp
- Message

B. Techniques Used

For data collection we have used following two techniques:

- For data collection in Python some of the scripts make use of github3 API. The collected data was anonymized using technique defined below and processed to extract useful information.
- Second technique we used to collect data is with the help of python script gitable.py We have used the script with different urls to collect data for issues, milestones, events and commits. We parsed the data to fetch relevant fields.

C. Anonymization of data

For anonymization of repo name and contributor to those repositories a set of custom scripts have been written that reads repo_details.csv and person_map.csv. Repo_details.csv contains anonymized repo name to actual repo name mapping, whereas person_map.csv contains person's username and actual name to anonymized name mapping. These are matched when data is read using github3 API and incoming data is anonymized based on the mappings.

Please note that these csv are not present in our github repos for complete anonymization.

Repo Anonymization code

```
def read_anonymized_repos():
  w = 2
  h = 3
  repo_details = [[0 for x in range(w)] for y in range(h)]
  f = open('../repo_details.csv', 'rt')
  reader = None
  try:
    reader = csv.reader(f)
    i = 0
    for row in reader:
      repo_details[i][0] = row[0]
      repo_details[i][1] = row[1]
      i += 1
  finally:
    f.close()
  return repo_details
```

User Anonymization Code

```
def anonymize_persons():
  person_map = {}
  f = open('../person_map.csv', 'rt')
  reader = None
  try:
    reader = csv.reader(f)
    for row in reader:
     person_map[row[0]] = row[1]
  finally:
    f.close()
  return person_map
```

III. Data Samples

Below tuples are the sample data that is collected using above two techniques:

Issue

Issue No.	Miles tone	Creation Time	Closing Time	Label	Title
14	2	14560 86281	14561 93343	data-collection	Sample DB
14	2	14560 86179	14564 64028	data-collection	DB Ready!

Milestone

Milestone Number	Creation Date	Due Date	Closed Date
2	Jan 21, 2016	Feb 2, 2016	Feb 8, 2016
5	Feb 6, 2016	Mar 4, 2016	Mar 1, 2016

Commit

Commit Sha	User	Time stamp	Message
659b81802 77cd3402a 3a3ff175e	Person 1	Feb 2, 2016	Added Data Collection Instructions
3100570b71 2d148a0d fb7b6964	Person 2	Mar 4, 2016	Talk Updated

Pull Request

Number
2
8

Comments

Issue	User	Created at	Updated it	Text	Identifier
21	Person 3	14564 64218	14564 64218	Cool.. lets get on to it and finish it ASAP	18911 8579

IV. Bad Smells

A. *Fraction of Issues Handled by each team member*

This feature depicts the fraction or percent of total issues that each team member handled. Teamwork and proper planning are important ingredients for success of any project. This feature captures the effort put in by each team member by calculating the number of issues handled by him or her. Here we have considered issues as basic units of work.

Feature Detection:
For extracting this feature we have used issues data for each of the repository. We segregated the issues on the basis of assignee and calculated the total count of issues for each team member. We have represented issues handled by a person as the fraction of total number of issues created in the project by the team.

Feature Detection Result:
The below sample data column from the csv file enumerates the columns used in the calculation of this feature.

Group_id	Issue_id	State	User	Title
Repo 1	35	open	Person 1	Which groups…

Bad Smell Detector:
The uneven or not equal distribution of work is qualified as bad smell. The above feature is helpful in detecting this kind of unequal distribution of issues or work. Any repository that contains high discrepancies in the percentage value of the issues assigned per person is a candidate for this bad smell.

Bad Smell Results:
The results obtained corresponding to this feature is explained below with help of pie charts generated for each of the repositories.

Repo 1 – The division of issues among group members is not uniform for group 1. Person 1 handled majority of issues where as rest of the team members combined were assigned fewer number of issues. The pie chart signifies that all team members not contributed equally to the project and project was more of an individual effort by Person 1.

Repo 2 – As evident by the pie chart below work distribution is uniform across all team members of group 2. Person 2 has been assigned majority of issues but the difference is not significant to raise any alarms.

Fig. 1: **Repo1**

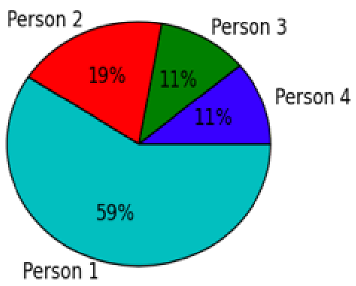
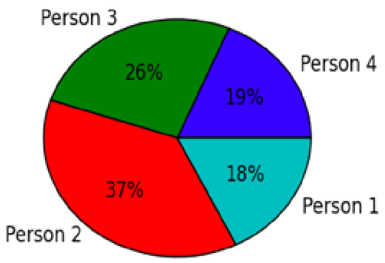


Fig. 2: **Repo2**



Repo 3 – The pie chart below brings to our attention one very important discrepancy about the work distribution in group 3 that is none of the issues were assigned to Person 1. This shows that Person 1 has very less or no contribution towards the project. Among rest of the team members also the distribution of work is not uniform and Person 2 does high percentage of work.

B. Number of Issues per Milestone

This feature accounts for total number of issues assig ned to a milestone created. Milestones act as important stages in the lifecycle of project. Each milestone consists of features that are relevant to the project success. Milestones with large number of issues or with very few issues signify bad smell or a drawback in the planning or design phase of project.

Feature Detection:

For extracting this feature we have used issues and milestones data for each of the repository. We have calculated the total number of issues under each created milestone in a repository.

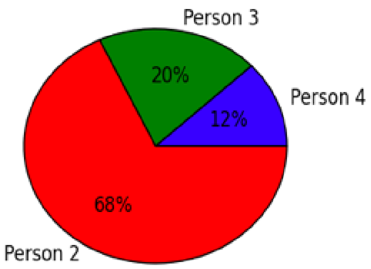
Feature Detection Result:

Columns in the csv file used for this feature are shown below:

Group_id	Milestone id	Milestone state	Issue id	Milestone title
Repo 1	10	open	35	Project2:...

Bad Smell Detector:

Fig. 3: **Repo3**

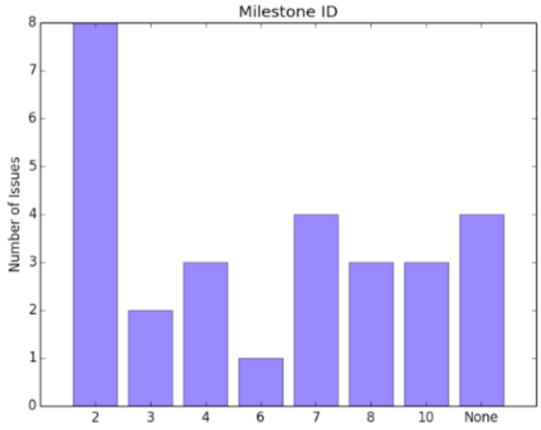


The planning or design phase of the project is equally important as the implementation phase for the success of the project. Milestones with unusually large or small number of issues signify loopholes in the planning process of the projects. Apart from that very large or small number of milestones in project itself can be designated as a bad smell.

Bad Smell Results:

Below we have shown bar charts corresponding to this feature that depicts the distribution of issues across all the milestones in a project.

Fig. 4: **Repo1**



Repo 1 – As we can see in the bar chart below four of the created issues are never assigned to a milestone, which is not a good practice. Apart from that total number of milestones in the repository are satisfactory. Milestone 2 has unusually large number of issues assigned to it whereas there very few issues in Milestone 3 and 6 that indicates that Milestone 2 could have been further divided and 3 and 6 could be created as issues rather than milestones.

Repo 2 – Repository have good number of milestones generated but the distribution of issues across milestones is not satisfactory. Milestone 4 contains majority of the issues and should have been further divided in other milestones. The repository has one issue that was never assigned to any milestone.

Fig. 5: **Repo2**

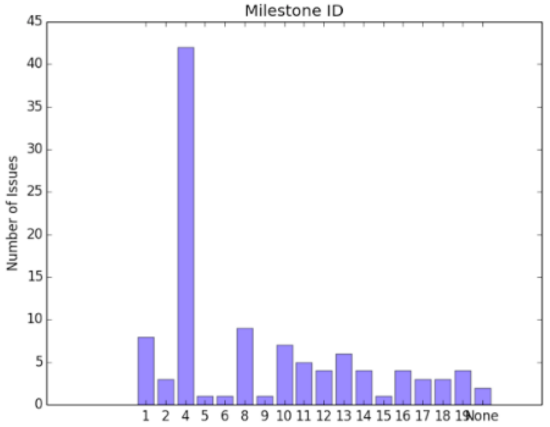
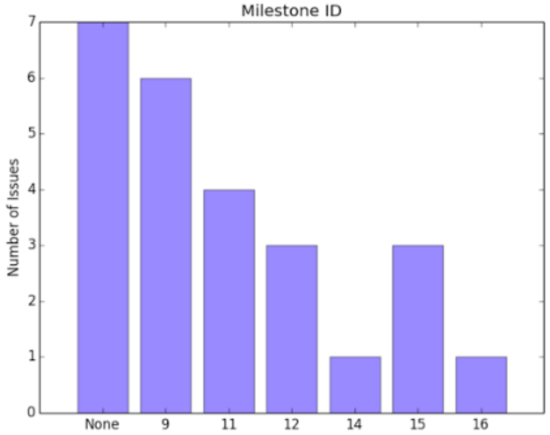


Fig. 6: **Repo3**



Repo 3 – Group 3 does not have enough number of milestones that tells us that work was not planned in a modular manner and the planning phase was not comprehensive. Apart from that maximum numbers of issues were never assigned to the milestone as the count od None is highest as evident in graph. This is a clear indication of bad planning and hence bad smell.

C. Short-Lived or Long-Lived Issues

Issue that is short-lived or an issue that is closed within few days of creation signifies bad smell as it tells that issue was not needed in the first place. The reason for creating such an issue could be just increase data in repo to show virtual progress.

On the other hand the long-lived issues are those that remained open for much larger phase. These issues are indicative of complicated problems or hacks enforced in the project. These kinds of issues are generally indicative of hacks put in the code to surpass a problem.

Feature Detection:

For extracting this feature we have used issues data of the repository that have created_at and closed_at timestamp. We have considered all the issues for one repository at a time while doing the calculations.

Feature Detection Result:

Columns in the csv file used for this feature are shown below:

Group_id	Issue id	Created at	Closed at	Issue state
Repo 1	25	14567 19089	14568 55681	closed

Bad Smell Detector:

For extracting this feature we have used calculated the average lifetime of an issue in a repository. Any issue that lasted for a time period longer than 1.5 times the average lifetime whereas the issue that got closed before 0.5 times the average lifetime is counted in short-lived issue.

Formulae used: Short-lived Issue <= 0.5 * Avg. lifetime of an issue

Long-lived Issue >= 1.5 * Avg. lifetime of an issue

Bad Smell Results:

The results obtained corresponding to this feature is explained below with help of pie charts generated for each of the repositories.

Fig. 7: **Repo1**

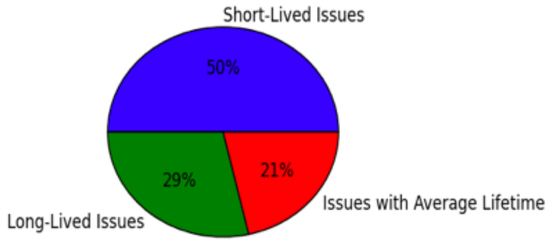
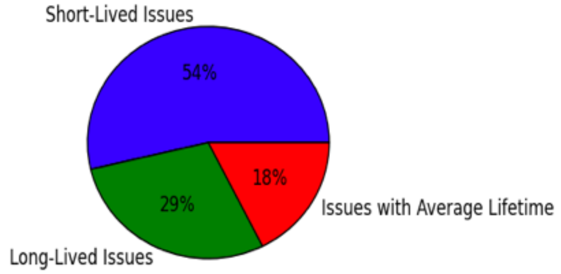
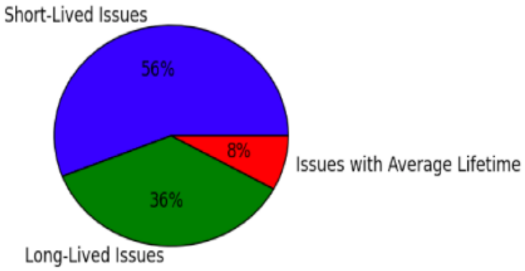


Fig. 8: **Repo2**



Repo 1 – As evident by the pie chart below half of the issues created in the repository are short-lived and around one-fourth were long-lived issues. These are indication of loopholes in the planning phase and hacks put in to make the code work in later stages.

Fig. 9: **Repo3**



Repo 2 – More than half of the issues created in the repository falls under the category of short-lived issues and there are plenty of long-lived issues present as well. Group 2 also does not performed well in this metric.

Repo 3 – In the pie chart below 92 percent of the issues fall under the category of either short-lived or long-lived issues. The number of issues with optimum lifetime is very less and hence these trends indicate that quality of code is not maintained throughout the life cycle of project.

D. *Uneven Commits By Contributors in Project Group*

In this feature our goal was to identify freeloader and dominator in project repos. In order to accomplish the same we need analyzed the required features to extract total commit counts during the course of project by each individual.

Feature Detection:

We identified each unique user using their user ID and username as we specified in person_map.csv and their corresponding commits using commit sha.

Feature Detection Result:

Below are the results of the feature detection which were further summarized for plotting the graph.

Commit Sha	User
659b8180277cd3402a3a3ff175e	Person 1
3100570b712d148a0dfb7b6964	Person 2

Further these features were further summarized as shown below for each repo:

Commiter	Commit Count
Person 1	12
Person 2	92

Bad Smell Detector:

To qualify whether a repo has a bad smell on the basis of uneven person's commit, we needed certain threshold to decide if any person is playing role of a freeloader or dominator. We decided these thresholds to be 10% and 50%. If a person's commits <10% of total then person is qualified as freeloader, of if the person's commit >50% then its qualified as dominator.

Bad Smell Results:

Fig. 10: **Repo1**

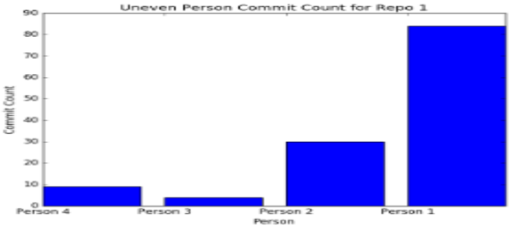
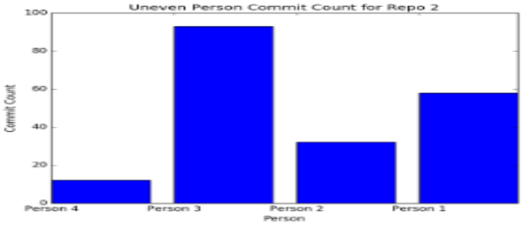


Fig. 11: **Repo2**

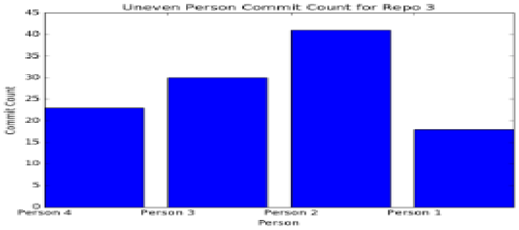


Repo 1: From the below graph it is pretty evident that person 1 did most of the work (67%) thus a dominator, whereas person 3 and person 4 did least work (3.1% and 7.1% resp.) thus are freeloaders.

Repo 2: From the below graph we can easily infer that person 3 did most of the work (47%) thus almost like a dominator whereas person 4 is a freeloader with 7% commits.

Repo 3: The contributor's in this repo managed to contribute equally and thus have no bad smell.

Fig. 12: **Repo3**



E. *Issues exceeding Milestone Due Date*

In this feature our goal was to identify the issues that surpasses their respective milestone due dates, which directly affects the overall pace of the project work.

Feature Detection:

We identified each and every issue identified by issue number that was assigned to a milestone identified by milestone number and captured it's milestones intended due date and actual closing date of the issue.

Feature Detection Result:

Below are the results of the feature detection which were further summarized for plotting the graph.

Issue Number	Milestone Number	Due Date	Closed Date
14	2	Feb 2, 2016	Feb 8, 2016
23	5	Mar 4, 2016	Mar 1, 2016

Further these features were further filtered as shown below only for issues that exceeded milestone due date for each repo:

Issue Number	Milestone Number
14	2
29	7

Bad Smell Detector:

To qualify whether a repo has a bad smell on the basis of number of issues exceeding milestone due date, we opted to select threshold of 25%, i.e. if a repo has more than 25% issues that exceeded milestone due date, then it is qualified a bad smell.

Bad Smell Results:

Below are the results of the bad smell detection on the basis of our threshold assumed for bad smell issues.

Fig. 13: **Repo1**

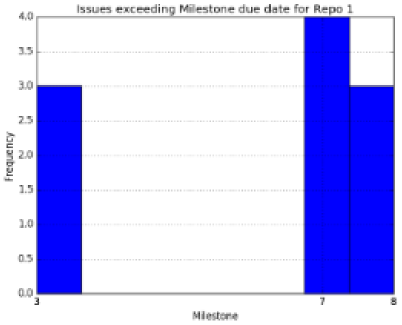
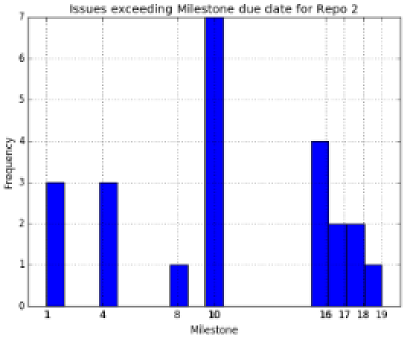


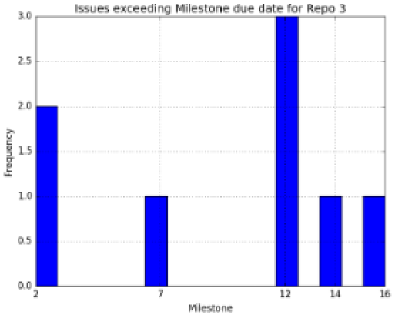
Fig. 14: **Repo2**



Repo 1: Given that the repo one had 31 issues out of which 10 missed the milestone due date (33%), this repo is qualified to be having a bad smell.

Repo 2: It had a total of 96 issues. And from below graph we can count that it missed 23 (24%) it is qualified as not to have bad smell, yet it is very close to being one.

Fig. 15: **Repo3**



Repo 3: This repo had a total of 37 issues out of which 8 (26%) missed their deadline, thus this repo is also qualified to have bad smell.

F. Issues without milestones

In this feature our goal was to identify the issues were not assigned to the issues at all, rationale behind finding this bad smell was to identify improper planning of milestones and creation of issues without having a set goal/due-date of completing it.

Feature Detection:

This feature extraction was pretty straight forward and simple we looked only for issues that had milestone number not assigned. However, there was a caveat in this approach as pull requests are also identified as issues in github, which were also getting picked up as issues and this was not our intended goal. Thus we first picked all the pull request numbers and then only picked those issues that were missing milestone number and didn't match any pull request number.

Feature Detection Result:

Below are the results of the feature detection which were further summarized for plotting the graph.

Pull Request Numbers
5
18

Issue Number	Milestone Number
14	None
18	None

Repo Number	Issues without milestones
Repo 1	18
Repo 2	5

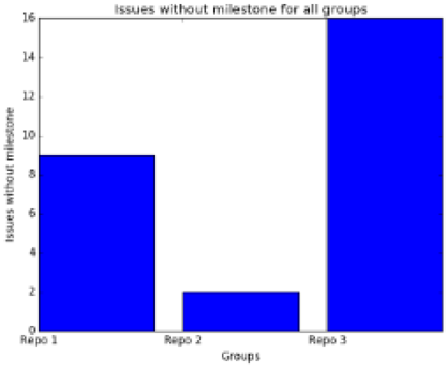
Bad Smell Detector:

To qualify whether a repo has a bad smell on the basis of number of issues without, we opted to select threshold of 25%, i.e. if a repo has more than 25% issues that were not assigned to milestones, then it is qualified a bad smell.

Bad Smell Results:

TBelow are the results of the bad smell detection on the basis of our threshold assumed for bad smell issues.

Fig. 16: **Repo1**



Repo 1, 2 and 3: Given that the repo one had 31 issues out of which 9 (29%) were not assigned to any milestone, this repo is qualified to be having a bad smell. Whereas repo 2 had only 2 (2%) issues out of 96 issues not assigned to any milestone (1 was a comment by Dr. Tim Menzies) thus not a bad smell and repo 3 had 16 (43%) not assigned to any milestone thus very high on this particular bad smell.

G. *Uneven weekly commits by Project Group*

In this feature our goal is to analyze weekly commits by project group to identify if they worked consistently throughout the project or worked only around the deadlines or submissions.

Feature Detection:

In order to extract the features for this particular bad smell we extracted all the commits identified by commit sha for the repo along with their commit dates which we used to identify the week of commits.

Feature Detection Result:

Below are the results of the feature detection which were further summarized for plotting the graph.

Commit sha	Date of Commit	Week of Commit
659b8180 277cd34 02a3a3 ff175e	Feb 8, 2016	6
3100570 b712d1 48a0df b7b6964	Mar 1, 2016	9

Week Number	Commit Count
4	12
8	92

Bad Smell Detector:

To qualify whether a repo has a bad smell on the basis of weekly commits, we needed certain threshold to decide if group was regular or inconsistent during the project. It can be inconsistent if it underworked or/and over-worked during some weeks. We decided these thresholds using the standard statistical procedure, i.e. taking mean of all commits and then if commits in a week<mean + 2*standard deviation then it over-worked in that week, otherwise if its <mean – 2*standard deviation then it is qualified as under-worked week. Also, we are taking into consideration commits beginning week 2.

Over-Worked Week commits have ¿ mean + 2*standard deviation

Under-Worked Week commits have ¡ mean - 2*standard deviation

Bad Smell Results:

Below are the results of the bad smell detection on the basis of our threshold assumed for freeloader and dominator.

Fig. 17: **Repo1**

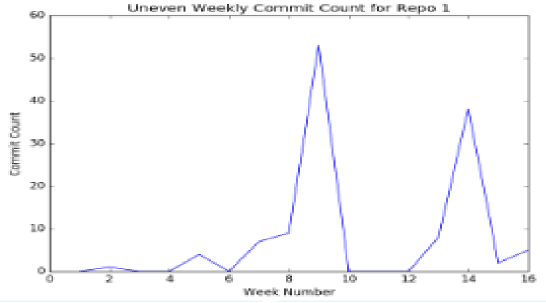
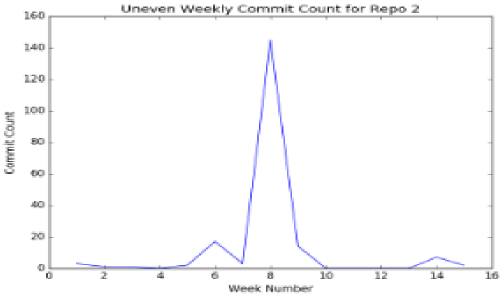


Fig. 18: **Repo2**



Repo 1: It had a total of 127 commits over the period of 15 weeks.

Mean = 127/15 = 8.46

Standard Deviation = 15.62

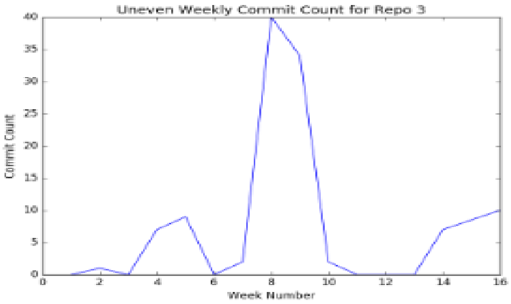
Thus from the below graph it can easily be inferred that 5 weeks out of 15 are under-worked (zero work) and 2 are over-worked.

Repo 2: It had a total of 195 commits over the period of 15 weeks.

Mean = 195/15 = 13

Standard Deviation = 36.8

Fig. 19: **Repo3**



Thus from the below graph it can easily be inferred that 5 weeks out of 15 are under-worked (zero work) and 1 is over-worked.

Repo 3: It had a total of 112 commits over the period of 15 weeks.

Mean = 112/15 = 8

Standard Deviation = 12.85

Thus from the below graph it can easily be inferred that 5 weeks out of 15 are under-worked (zero work) and 1 is over-worked.

H. *Less number of comments on an issue*

Here we aim to find if the issue status were properly updated by analyzing the number of comments associated with an issue. A properly updated issue should generally have comments explaining the different states. For instance, in case of a bug the originator of the issue should explain what the bug is and how it can be reproduced. The developer trying to fix the bug should update his analysis on the bug which could include information about the root cause of the bug, whether it is a duplicate of another bug etc. It also makes sense to update the issue with the details of the check in that fixes it. Hence, the number of comments on an issue is a good indicator of a properly analyzed issue.

Feature Detection:

For detecting this bad smell we extracted the issue comments data from all the three repository.

For each repository we calculated

1) Maximum number of comments ever received by an issue
2) Minimum number of comments ever received by an issue
3) Mean number of comments
4) Standard deviation

Feature Detection Result:

The extracted data was saved to a csv file in the following format: (add table)

Issue	User	Created at	Updated it	Text	ID
3	Person 1	14548 94217	14548 94217	As we only need one node.	18115 1440

Bad Smell Detector:

Less number of comments on an issue can be an indication of a bad smell. We assumed that an issue would at least have two comments - one when it was created which explains what the issue is and another when the issue was closed which explains what closed the issue.

Bad Smell Results:

Repo 1

Minimum number of comments ever received by an issue = 1

Maximum number of comments ever received by an issue = 4

Mean number of comments = 1.47058823529
VARIANCE = 0.719723183391
STANDARD DEVIATION = 0.848365005992

Fig. 20: **Repo1**

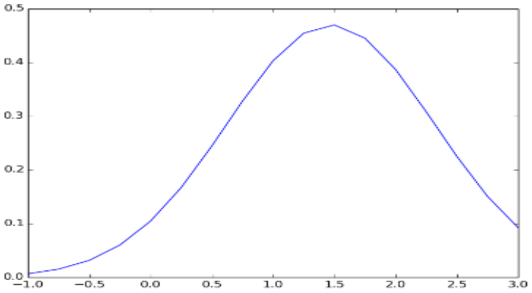
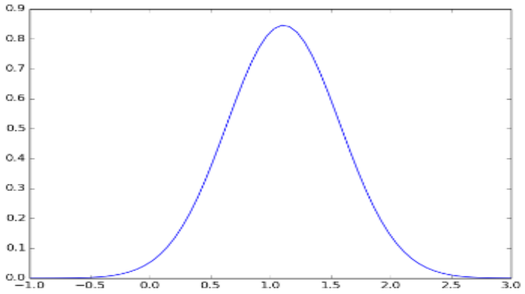


Fig. 21: **Repo2**



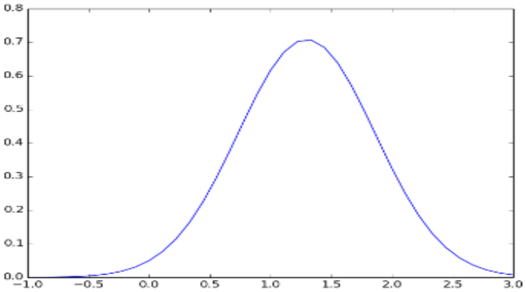
Repo 1

Minimum number of comments ever received by an issue = 1

Maximum number of comments ever received by an issue = 4

Mean number of comments = 1.47058823529
VARIANCE = 0.719723183391

Fig. 22: **Repo3**



STANDARD DEVIATION = 0.848365005992

Repo 2

Minimum number of comments ever received by an issue = 1

Maximum number of comments ever received by an issue = 4

Mean number of comments = 1.109375

VARIANCE = 0.222412109375

STANDARD DEVIATION = 0.471605883525

Repo 3

Minimum number of comments ever received by an issue = 1

Maximum number of comments ever received by an issue = 3

Mean number of comments = 1.2972972973

VARIANCE = 0.317019722425

STANDARD DEVIATION = 0.563045044757

I. Number of users commenting on an issue

For team projects it is common to have multiple users work on an issue at the same time or at different stages of an issue. There is a high probability that the users working on the issue updates their work on the issue through comments. Hence, the number of users commenting in an issue is a good indicator of the level of collaboration in a team project.

Feature Detection:

For detecting this bad smell we extracted the issue comments data from all the three repository.

For each repository we calculated

1) Maximum number of users ever commented on an issue
2) Minimum number of users ever commented on an issue
3) Mean number of users
4) Standard deviation

Feature Detection Result:

The extracted data was saved to a csv file in the following format:

Issue	User	Created at	Updated it	Text	ID
3	Person1	1454894217	1454894217	As we only need one node.	181151440

Bad Smell Detector:

Less number of users commenting on an issue can be an indication of a bad smell for team projects. We assumed that an issue would have more than one person looking into it in a team project.

Bad Smell Results:

Repo 1

Minimum number of comments ever received by an issue = 1

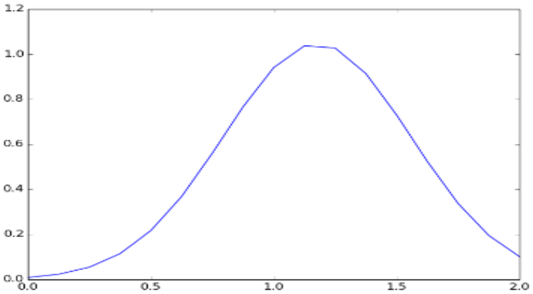
Maximum number of comments ever received by an issue = 2

Mean number of comments = 1.17647058824

VARIANCE = 0.145328719723

STANDARD DEVIATION = 0.381220041083

Fig. 23: **Repo1**



Repo 2

Minimum number of comments ever received by an issue = 1

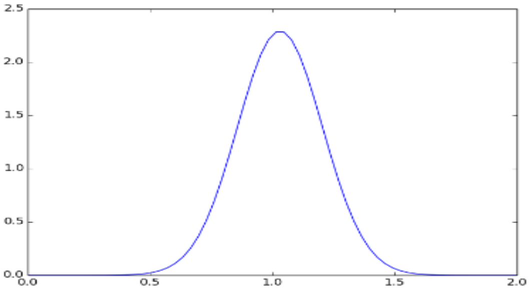
Maximum number of comments ever received by an issue = 2

Mean number of comments = 1.03125

VARIANCE = 0.0302734375

STANDARD DEVIATION = 0.173992636338

Fig. 24: **Repo2**



Repo 3

Minimum number of comments ever received by an issue = 1

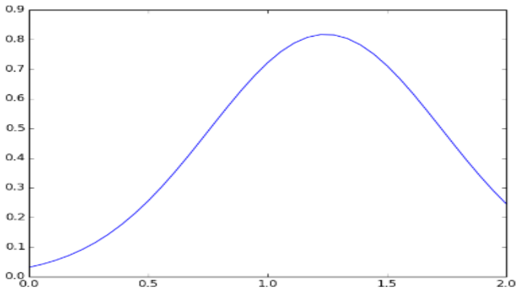
Maximum number of comments ever received by an issue = 3

Mean number of comments = 1.24324324324

VARIANCE = 0.238130021914

STANDARD DEVIATION = 0.48798567798

Fig. 25: **Repo3**



J. *Commits linked with issues*

While collaborating on projects through version control systems like git, it is a good practise to keep track of all the tasks through issues and link the check-ins made to the associated issue. While checking in to github this can be done by using the syntax #XYZ or gh-XYZ (XYZ is the issue ID which is a number) in the commit comment. The commits missing the associated issue information would be difficult to track in future.

Feature Detection:

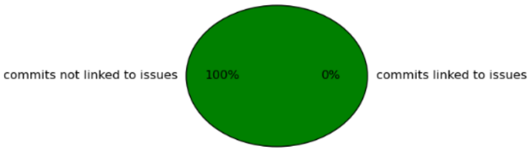
For detecting this bad smell we extracted the commit data from all the three repository. For each repository we analyzed the commit comment text to find patterns matching reference to an issue. The regex pattern we used for this was: (#|[gG][hH]-)\d+

Feature Detection Result:

The extracted data was saved to a csv file in the following format:

Time	Sha	User	Message
1456718215	fd0bc2 c17365 392a7c 5df8b5b e6d89ba 149f7d7	Person 2	creating log file

Fig. 26: **Repo1**



Bad Smell Detector:

It is recommended to link all the commits with an issue. Anything less than 100% could make it difficult to track in future what the commit was about.

Fig. 27: **Repo2**

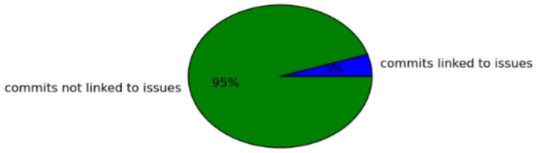
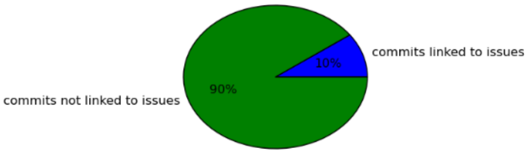


Fig. 28: **Repo3**



V. Early Smoke

A. *Individual Commit Trend*

In one of the above bad smell we found out that 2 repos had freeloader and dominators, however there were some early indicators that contributors were on path of becoming dominator or freeloader. In this analysis that we show below we aim to identify those early indicators, using which foretell if a contributor will be freeloader, dominator or not.

Feature Detection:

In order to extract the features for this particular early smoke we identified each unique user using their user ID and username as we specified in person_map.csv, their corresponding commits using commit sha and also extracted each user's date of commit beginning year 2016. **Feature Detection:**

Below are the results of the feature detection which were further summarized for plotting the graph.

User	Commit Sha	Date of Commit	Week of Commit
Person 1	659b8 180277 cda3f f175e	Feb 8, 2016	6
Person 2	31005 70b71 2d148a 0dfb7 b6964	Mar 1, 2016	9

Further these features were further summarized as shown below for each user in each repo:

Week Number	Commit Count
4	12
8	92

Early Smoke Detector

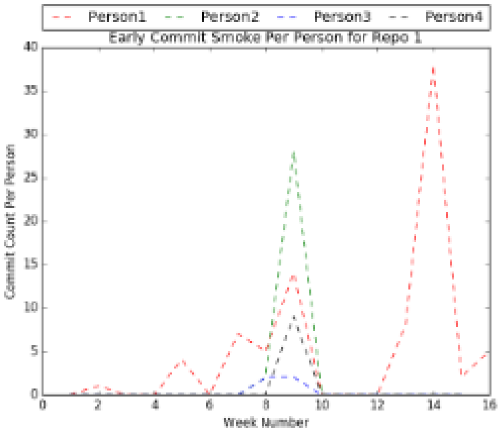
To qualify a contributor well in advance as freeloader or dominator we look at each individual's percentage of contributions around middle of the project i.e. week 8. If an individual is a freeloader with less than 10% of commits around middle of the project and some other person has more than 50% of the commits, it is most likely they will continue to be the same till the end of the project. Also other good indicator is week of their 1st commit in the project; it is most likely that free loaders start committing late in the project.

Early Smell Results:

Below are the results of the early smoke detection on the basis of our threshold assumed for freeloader and dominator around week 8.

Repo 1: By week 8 it had a total commits of 16, with Person 1 having 12 commits (75%), thus a potential of being a dominator, whereas Person 2 and 3 had 2 commits each (12.5%) closer to being a freeloader and person 4 had no commits therefore a freeloader. Person 1's first commit was in week 2, week 2 for person 2 and person 3 and week 9 for person 4. And, these results completely agree with the bad smell that we detected for the repo 1 above where person 4 is a dominator, person 3 and 4 are free loaders.

Fig. 29: **Repo1**



Repo 2: By week 8 it had a total commits of 167, with Person 1 having 52 commits (31%), whereas Person 2 had 19 commits (11.3%) closer to being a freeloader, person 3 had 91 commits (54.4%) thus a dominator and person 5 commits (3%) therefore a freeloader. Person 1's first commit was in week 5, week 1 for person, week 6 for person 3 and person 4. And, these results are in consistence with the bad smell that we detected for the repo 2 above where person 3 is a dominator, person 4 is a free loaders. Repo 3: By week 8 it had a total commits of 62, with Person 1 having 0 commits thus a freeloader, whereas Person 2 had 24 commits (39 %), Person 3 had 18 commits (29%) and Person 4 had 20 commits (32%). Person 1's first commit was in week 9 (later than most others), week 5 for person, week 2 for person 3 and week 7 for person 4. And, these results are nearly consistent with the bad smell that we detected for the repo 3 above except person 1 made significant contribution in second half of the project.

Fig. 30: **Repo2**

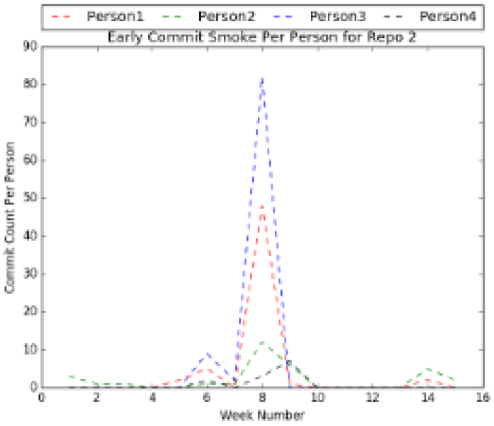
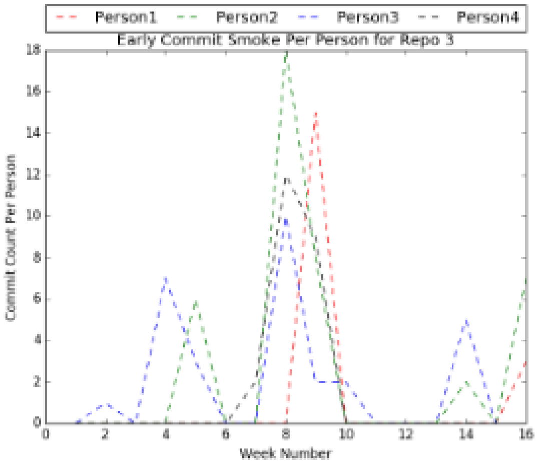


Fig. 31: **Repo3**



B. Milestone Completion Trend

In one of the above bad smell we found out that some repos significantly miss their milestone due dates, however there were some early indicators that milestones in second half of the project will miss their due dates based on the milestone completion trend in the first half of the project. Possible reasons can be any, team may not be concerned about meeting the deadlines, a team may be a little maid back or later milestones are dependent on earlier milestones. In this analysis that we show below we aim to identify those early indicators, using which foretell if future milestones in second half of project would meet their deadlines or not.

Feature Detection:

In order to extract the features for this particular early smoke we identified each milestone's (identified by milestone number) due date, closing date, and creation date. However we didn’t take into account the milestones without any due date.

Below are the results of the feature detection which were further summarized for plotting the graph

Early Smoke Detector

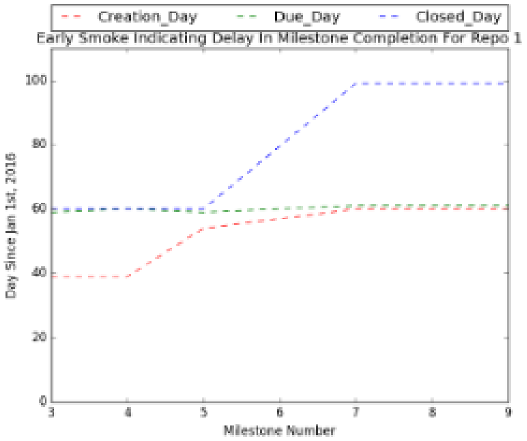
To foretell well in advance if a project will miss most of the deadlines in later half of the project we weigh them on the basis of their milestone completion trend in first half. If a repo misses half (50%) of its milestone due dates it is more probable their issues will miss milestone due dates in later half of the project.

Early Smell Results:

Below are the results of the early smoke detection on the basis of our threshold assumed first half of the project.

Repo 1: By milestone 6 it had missed milestone due date for all of its milestones. And, we can see that it also tends to miss its milestone due date significantly in second half of the project.

Fig. 32: **Repo1**



Repo 2: By milestone 5 it has missed 2 milestones due dates (1 by 1 day) and we see a similar trend in the second half of the project as well. It continued to deliver and achieve results in coherence to its milestone deadline.

Repo 3: By milestone 6 it missed all of its milestone due dates but not by significant number of days and we see a similar trend in the second half of the project as well. They were not affected by the missed deadline in first half of the project.

VI. Conclusion

In the conclusion we would like to state that in the above analysis each of the repo had some kind of bad smell. We believe that is the reason we have software engineering process like other engineering domains to develop software with least/no problems and avoid these bad smells.

Fig. 33: **Repo2**

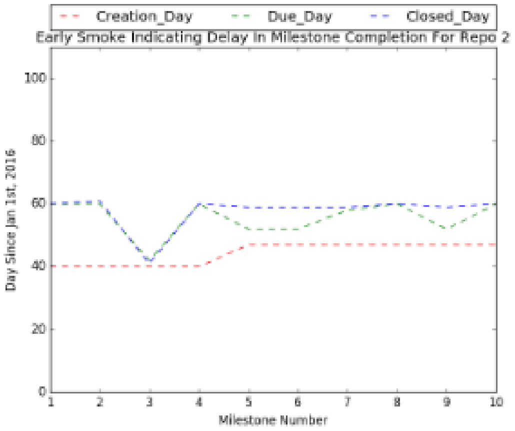
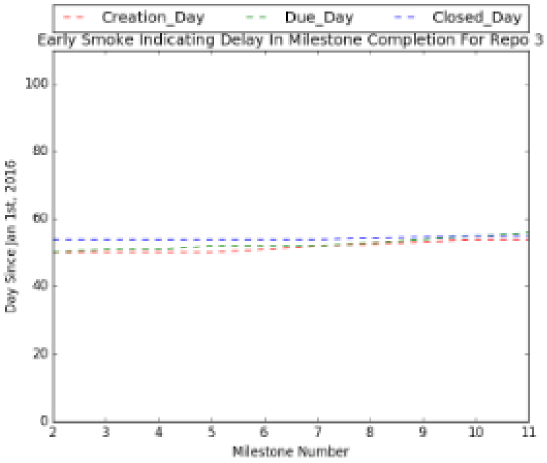


Fig. 34: **Repo3**



As more software engineering practices are maturing and taking statistical procedures also into account to overcome some of the inherent problems of designing software. We too gained a significant insight into these github repositories and how these project may have been executed (including ours) using the above analysis and we strongly feel that such measurements periodically can reduce bad smells in project and bring it back onto the track.

Below we have summarised our findings (whether a repo had a bad smell or not) on the basis of above ten bad smell comparisons.

Bad smell	Repo 1	Repo 2	Repo 3
Fraction of Issues Handled by each team member	Yes	No	No
Number of Issues per Milestone	No	No	Yes
Short-Lived or Long-Lived Issues	Yes	Yes	Yes
Uneven Commits By Contributors in Project Group	Yes	Yes	No
Issues exceeding Milestone Due Date	Yes	No	Yes
Issues without milestones	Yes	No	Yes
Issues exceeding Milestone Due Date Uneven weekly commits by Project Group	Yes	Yes	Yes
Issues without Milestones	Yes	No	Yes
Uneven weekly commits by Project Group	Yes	Yes	Yes
Commits linked with issues	Yes	Yes	Yes
Less number of comments on an issue	Yes	Yes	Yes
No. of users commenting on an issue	Yes	Yes	Yes

References

[1] Report reference , https://github.com/CSC510-2015-Axitron/project2, *Example from the Class of 2015*

[2] Reference for GitHub API, https://github.com/sigmavirus24/github3.py, *github3.py*

[3] wrapper for the GitHub API , http://github3py.readthedocs.io/en/latest/index.html#more-examples, *Reference Python code*