Bad Smells in Github Repositories

Ye Mao
Department of Computer Science
North Carolina State University
ymao4@ncsu.edu

Song Ju
Department of Computer Science
North Carolina State University
sju2@ncsu.edu

Arnab Saha
Department of Computer Science
North Carolina State University
asaha@ncsu.edu

Yiqiao Xu
Department of Computer Science
North Carolina State University
yxu35@ncsu.edu

ABSTRACT

For our software engineering project, all the teams used GitHub to manage their repositories. In this project, we tried to generate some features and extracted those data from three random repositories and combined the features into a second layer. Further we conducted some statistical analysis and used Multi-criteria decision making methods to show that one of the repository was emperically better managed than the other two. Finally we provided some bad smell indicators which can be used to identify bad smells which lies ahead.

Keywords

Usability studies, google maps, google search, web maps, multi-criteria decision making, ANOVA, t-test, Simple Additive Weighted Method.

1. INTRODUCTION

GitHub is one of the most popular web-based repository hosting service used for distributed revision control and source code management. Projects on GitHub can be accessed and manipulated using the standard Git command-line interface and all of the standard Git commands work with it. GitHub also allows registered and non-registered users to browse public repositories on the site [2].

Not only can GitHub help in tracking commits and issues, the GitHub Database API gives access to read and write raw Git objects to your Git database on GitHub and to list and update your references. For the first software engineering project, all the teams used GitHub for revision control and the user data could be retrieved using the GitHub Database API. The API access is over HTTPS, and accessed from the https://api.github.com. All data is sent and received as JSON [1].

For our analysis, we have used the data from the Issue, Event, Comment, Milestone, Commit and User table and stored them in a shared MySql database. Then we created 14 features and wrote queries to extract those features from the tables. Further we divided the results into 4 months to show how the results change over this span. This helped us to identify good features from our list which can provide useful information to detect bad smells which lies ahead.

This paper is structured as follows. Section 2 presents the data collection method. Section 3 talks about the anonymization technique. Section 4, 5 and 6 provides the data, tables and some sample data respectively. Section 7 presents the feature detection and results are mentioned in section 8. Section 9 and 10 talks about Bad Smells and the results. Section 11 and 12 presents the Early warning and the results. Finally section 13 and 14 presents the acknowledgement and conclusion respectively.

2. DATA COLLECTION

This section will state how to pull the data out of Github for the analysis in this project.

2.1 The way to collect data

During the data collection phase, we made use of gitable.py, which was recommended by Dr. Menzies. In order to get more data, we made some modification to the script, and we thus got all the data we needed with the help of GithubâĂŹ API. Then, we inserted the data into a SQL database on Cloud 9 after anonymization.

2.2 The data we extracted

After discussion, we agreed to focus on the following six types of data provided by Github' API:

1. ISSUE
   (a) ISSUE_URL

(b) ISSUE_ID
(c) ISSUE_TITLE
(d) ISSUE_USER_ID
(e) ISSUE_STATE
(f) ISSUE_CREATE_TIME

2. EVENT
   (a) EVENT_ID
   (b) EVENT_ACTOR_ID
   (c) EVENT_ACTION
   (d) EVENT_ISSUE_ID
   (e) EVENT_CREATE_TIME
3. COMMENT
   (a) COMMENT_ID
   (b) COMMENT_ISSUE_URL
   (c) COMMENT_USER_ID
   (d) COMMENT_CREATE_TIME
   (e) COMMENT_UPDATE_TIME
4. MILESTONE
   (a) MILESTONE_ID
   (b) MILESTONE_TITLE
   (c) MILESTONE_STATE
   (d) MILESTONE_CREATOR_ID
   (e) MILESTONE_CREATE_TIME
   (f) MILESTONE_UPDATE_TIME
   (g) MILESTONE_DUE_TIME
   (h) MILESTONE_CLOSE_TIME
5. COMMIT
   (a) COMMIT_ID
   (b) COMMIT_USER_ID
   (c) COMMIT_TIMESTAMP
6. USER
   (a) USER_ID
   (b) USER_GROUP_NUMBER

3. ANONYMIZATION

The main purpose of this project is to find the bad smells for the team work, which widely exist in our own development process. So precautions have been taken that any personal information has been hidden in the database. Groups have been labelled as group1, group2 and group3, and the users will be identified by these given names such as user1, user2 and etc. Also, there is no clue which will indicate the content of these projects in the features we have extracted.

4. TABLES

All tables for this project are in the May 1 section, one folder per group. After extracting data from github repositories using Github' API, we applied anonymization process, and store the anonymized data into csv files, along with the database. When analyzing data, we choose one or more parts in the data files, implemented with statistical methods like mean and standard deviation, and find characteristics of different projects. At last, we check the data and combine them to find if there is bad smell in projects.

5. DATA

The following table indicates the total numbers of data that we have collected from all the three groups, which comes from the select query in database.

Issues	Events	Comments	Milestones	Commits
220	1005	322	25	399

Table 1: Data

In SQL, this is expressed as: SELECT (SELECT COUNT(*) FROM ISSUE) AS Issues,
(SELECT COUNT(*) FROM EVENT) AS Events,
(SELECT COUNT(*) FROM COMMENT) AS Comments,
(SELECT COUNT(*) FROM MILESTONE) AS Milestones,
(SELECT COUNT(*) FROM COMMIT) AS Commits FROM dual

6. DATA SAMPLE

The following data samples are some of the raw data from our database. Table 2 shows a sample data from the ISSUE table. Similarly Table 3, 4, 5, 6 shows a sample from EVENT, COMMENT, MILESTONE, and COMMIT table respectively.

7. FEATURE DETECTION

Eventually, we came up with 15 different detectors to extract features from those group projects.

7.1 Long Open Issues

In the dataset, we found that different issues had different lifetime. Since each project lasted for approximately one month, so we considered those issues which was not marked as closed even after 15 days of creation as long open issues. In SQL, this feature could be extracted via:
SELECT ISSUE_ID, TIMESTAMPDIFF(DAY,
ISSUE_CREATE_TIME, ISSUE_CLOSE_TIME),
USER_GROUP_NUMBER from ISSUE
join USER on ISSUE_USER_ID = USER_ID where
TIMESTAMPDIFF(DAY,ISSUE_CREATE_TIME,
ISSUE_CLOSE_TIME)>15 order by USER_GROUP_NUMBER;

7.2 Short Open Issues

This feature is just as straightforward as the long open issues. In the tables, we found that some issues were marked as closed just after few minutes, so we considered those issues which were marked as closed within 30 minutes when they was first created as short open issues. In SQL, this feature could be extracted via:

id	title	user	state	assigner	milestone_id	created_at	updated_at	closed_at
149609587	Creating presentation	user1	closed	user1	1558878	2016-04-19	2016-04-22	2016-04-22

Table 2: Issue

id	actor	event	created_at	issue_id	label
639201033	user1	labeled	2016-04-24	132569582	enhancement

Table 3: Event

id	user	created_at	update_at	body
172395092	user3	2016-01-17	2016-01-17	1.Find problems 2.Count nubmer of clicks

Table 4: Comment

id	title	description	creator	open_issues	closed_issues	state	created_at	due_on	closed_at
1588944	Solution 2	All in solution2	user1	0	8	closed	2016-02-17	2016-04-24	2016-04-24

Table 5: Milestone

sha	commit_time	commit_message	committer
9684df02dc9c0f154b0723b5d758aa0767d6c319	2016-04-24	push our json file	user2

Table 6: Commit

SELECT ISSUE_ID, TIMESTAMPDIFF(MINUTE,
ISSUE_CREATE_TIME, ISSUE_CLOSE_TIME),
USER_GROUP_NUMBER
from ISSUE join USER on ISSUE_USER_ID = USER_ID
where TIMESTAMPDIFF(MINUTE,
ISSUE_CREATE_TIME,ISSUE_CLOSE_TIME)<30
order by USER_GROUP_NUMBER;

7.3 Issues Without Milestones

This feature is also rather straightforward. Since the function of milestone is to make the teamwork go more smoothly with better planning. And we felt that knowing the number of issues without milestone was important for the analysis. In SQL, this feature could be extracted via:
select sum(case when
ISSUE_MILESTONE_ID = “” then 1 else 0 end)
Milestone_Missing, count(ISSUE_ID) total,
USER_GROUP_NUMBER from ISSUE join USER
on USER_ID = ISSUE_USER_ID group by
USER_GROUP_NUMBER order by USER_GROUP_NUMBER;

7.4 Issues Without Assignees

This feature is not complicated, either. Since the reason to raise an issue is that there is something needed to be fixed so that the whole project can be done. Thus, those issues without assignees is another good way to find the bad smells. In SQL, this feature could be extracted via:
select sum(case when
ISSUE_ASSIGNEE_ID = “” then 1 else 0 end)
Assignee_Missing, count(ISSUE_ID) total,
USER_GROUP_NUMBER from ISSUE join USER
on USER_ID = ISSUE_USER_ID group by
USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;

7.5 Issues Missing Milestone Due Date

This feature is fairly straightforward. For those issues which have been milestoned would gain a due date, and we found that there were some issues which was marked as closed after the milestone due date. In SQL, this feature could be extracted via:
SELECT ISSUE_ID,ISSUE_NUMBER,
TIMESTAMPDIFF(DAY,MILESTONE_DUE_TIME,
ISSUE_CLOSE_TIME), ISSUE_USER_ID,
USER_GROUP_NUMBER from ISSUE join USER
on USER_ID = ISSUE_USER_ID join MILESTONE
on MILESTONE_ID = ISSUE_MILESTONE_ID
order by USER_GROUP_NUMBER;

7.6 Equal number of Issue Assignees

For those issues with assignees, we decided to get the number of issues assigned to each user in a project, which could be a helpful clue for the bad smells. In SQL, this feature could be extracted via:
SELECT count(ISSUE_ID),ISSUE_ASSIGNEE_ID,
USER_GROUP_NUMBER from ISSUE join USER
on USER_ID = ISSUE_USER_ID group by
ISSUE_ASSIGNEE_ID, USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;

7.7 Equal number of Issues posted by each user

As the same case with the assigned issues, we thought it was also important to know the spread of users who identified and created issues. In SQL, this feature could be extracted via:
SELECT count(ISSUE_ID),
USER_GROUP_NUMBER from ISSUE join USER
on USER_ID = ISSUE_USER_ID group by
USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;

7.8 Average number of comments per issue

We thought that team members' communication on the issues would be a reflection of the degree of the involvement and devotion for the team project. A relatively high or low number of people who commented on a particular issue could also give us some clues. In SQL, we can discover the average number of comments of each issue via the following query:

```
SELECT AVG(ISSUE_COMMENTS),
USER_GROUP_NUMBER from ISSUE
join USER on USER_ID = ISSUE_USER_ID
group by USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;
```

7.9 Number of comments per user

As with the number of comments per issue, we decided to get the the number of comments per person, which could also give us some clues for the involvement of particular user. In SQL, this feature could be extracted via:

```
SELECT COUNT(COMMENT_ISSUE_URL),
COMMENT_USER_ID, USER_GROUP_NUMBER
from COMMENT join USER on
USER_ID = COMMENT_USER_ID
group by COMMENT_USER_ID,
USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;
```

7.10 Time taken for fixing bugs

In the data, we noticed the difference between the time when the "bug" label was assigned to an issue and the time when the issue was marked as closed. And thus the time difference could give us the time taken for fixing bug, which somehow indicated the devotion of the participants in the project. In SQL, this feature could be extracted via:

```
SELECT ISSUE_ID, USER_ID,
TIMESTAMPDIFF(MINUTE,ISSUE_CREATE_TIME,
ISSUE_CLOSE_TIME),USER_GROUP_NUMBER
from EVENT join USER on
USER_ID = EVENT_ACTOR_ID join ISSUE
on ISSUE_ID = EVENT_ISSUE_ID
where EVENT_LABEL_NAME = "bug"
order by USER_GROUP_NUMBER;
```

7.11 Time taken for creating enhancements

As with the time taken for fixing bugs, we also decided to collect the data focusing on "enhancement" label, which could somehow give us some clue for the analysis phase. In SQL, this feature could be extracted via:

```
SELECT ISSUE_ID, USER_ID,
TIMESTAMPDIFF(MINUTE,ISSUE_CREATE_TIME,
ISSUE_CLOSE_TIME),USER_GROUP_NUMBER
from EVENT join USER on
USER_ID = EVENT_ACTOR_ID join ISSUE
on ISSUE_ID = EVENT_ISSUE_ID
where EVENT_LABEL_NAME = "enhancement"
order by USER_GROUP_NUMBER;
```

7.12 Number of commits per user

In the database, we found that each user had different number of commits in their repositories. And we decide to get the data which might give us some clue for the analysis. In SQL, this feature could be extracted via:

```
SELECT count(COMMIT_ID), COMMIT_USER_ID,
USER_GROUP_NUMBER from COMMIT join USER
on COMMIT_USER_ID = USER_ID
group by USER_GROUP_NUMBER,
COMMIT_USER_ID;
```

7.13 Active number of days per repository

Almost each group had one month for each of their project, but it seemed that their work for those projects were not even. So we decided to find the active number of days for each group. In SQL, this feature could be extracted via:

```
SELECT COUNT(DISTINCT(DATE(COMMIT_TIMESTAMP)))
as DistinctDays, USER_GROUP_NUMBER from COMMIT
join USER on COMMIT_USER_ID = USER_ID
group by USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;
```

7.14 Active number of days per user

As with the active number of day per repository, we also decided to find the active number of days for each user. In SQL, this feature could be extracted via:

```
SELECT COUNT(DISTINCT(DATE(COMMIT_TIMESTAMP)))
as DistinctDays,
USER_GROUP_NUMBER from COMMIT join USER
on COMMIT_USER_ID = USER_ID
group by USER_GROUP_NUMBER
order by USER_GROUP_NUMBER;
```

7.15 Spread of event history

From the event table, we found that the number of events for one single day was different. And we thought it could be an interesting part to illustrate the working frequency of one group. For a project which was perfectly planned and conducted, we expected the spread of its event history to be as even as possible. In order to find the spread of event history for each group, we could extract the related data from the database via the following SQL query:

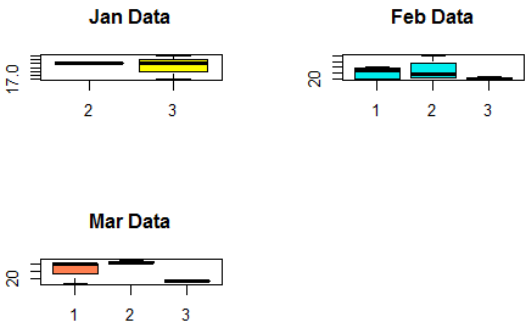
```
SELECT count(EVENT_ID), DATE(EVENT_CREATE_TIME),
USER_GROUP_NUMBER from EVENT join USER on
EVENT_ACTOR_ID = USER_ID group by
USER_GROUP_NUMBER,DATE(EVENT_CREATE_TIME);
```

8. FEATURE DETECTION RESULTS

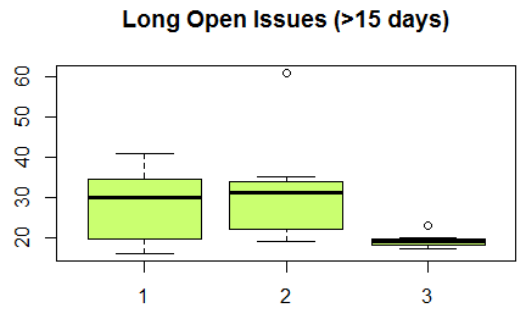
For each features that we have extracted for each group, we have separately evaluated the data in each month. Thus we could get more accurate information for the activities for each project, and we have used R studio to visualize those quantitative data we have got from section 7.

8.1 Long Open Issues

The boxplots in figure 1 shows the number of long open issues for three groups. As mentioned in section 7.1, taking the length of each project is one month into account, we consider those issues which has been open for more than 15 days as long open issues, or "mortal" issues. And there is no long open issue in April for all the three groups.



(a) Data in each month

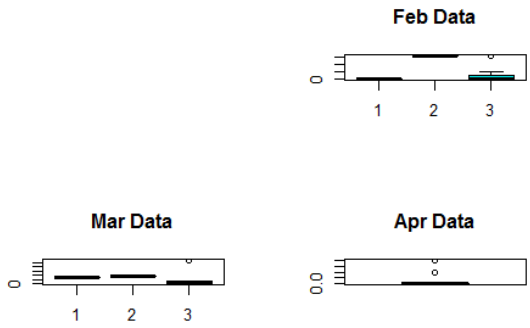


(b) All the data in four months

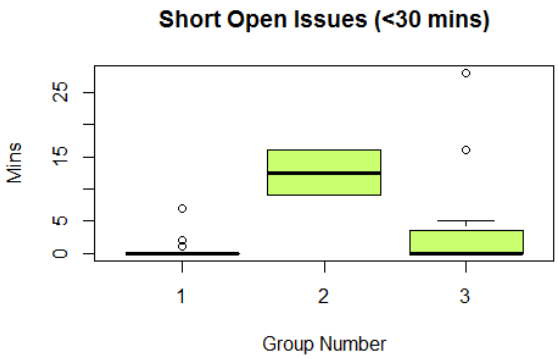
Figure 1: Number of long open issues

8.2 Short Open Issues

The boxplots in figure 2 shows the number of short open issues for three groups.As mentioned in section 7.2, those issues which was closed just in few minutes were not meaningful. According to the figure 1(a), we could easily notice that there is no "bad" issues in January, which is different from the case that no "mortal" issues in April.

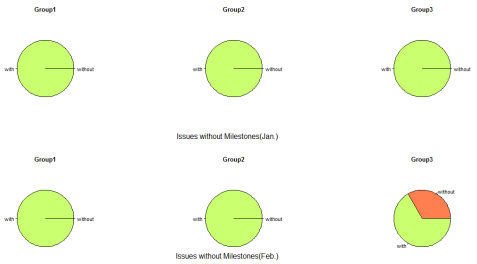


(a) Data in each month

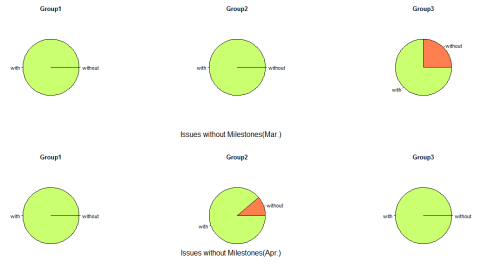


(b) All the data in four months

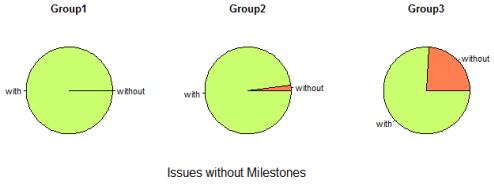
Figure 2: Number of short open issues



(a) Data in first two months



(b) Data in last two months



(c) All the data in four months

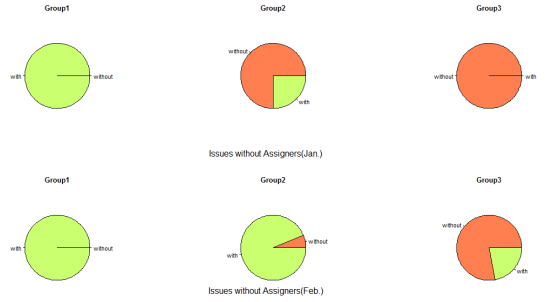
Figure 3: Percentage of Issues Without Milestones

8.3 Issues Without Milestones

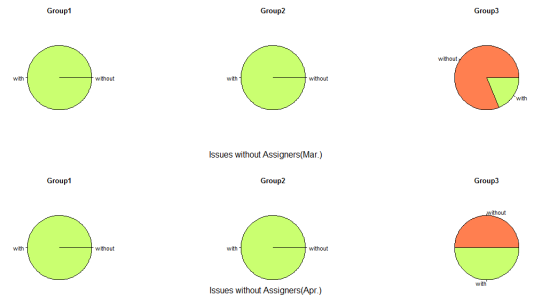
The pie charts in figure 4 shows the percentage of those issues which did not have any milestones, which indicates an improper use of issues. As can be seen below there were just few issues without milestone in the three groups.

8.4 Issues Without Assignees

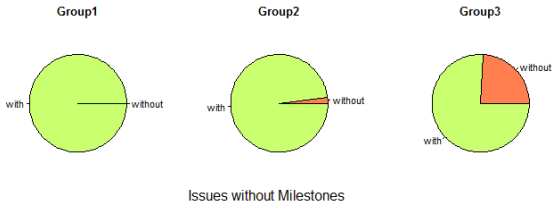
The pie charts in figure 3 shows the percentage of those issues which did not have any assignees, which indicates an improper use of issues. As can be seen below there were just few issues without assignees in the first two groups, while group 3 did have a lot of issues without assignees.



(a) Data in first two months

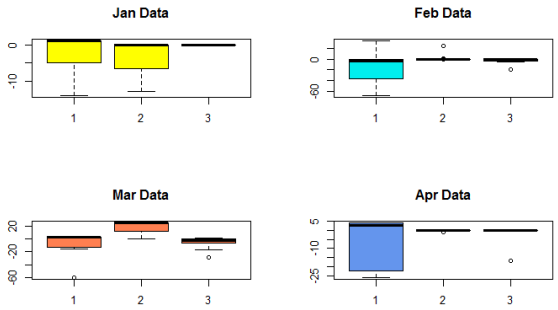


(b) Data in last two months



(c) All the data in four months

Figure 4: Percentage of Issues Without Assignees



(a) Data in each month

(b) All the data in four months

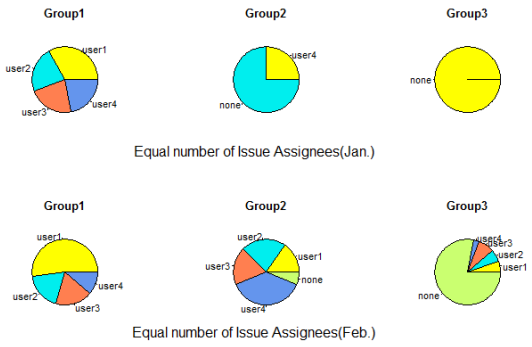
Figure 5: Number of issues missing milestone due date

8.5 Issues Missing Milestone Due Date

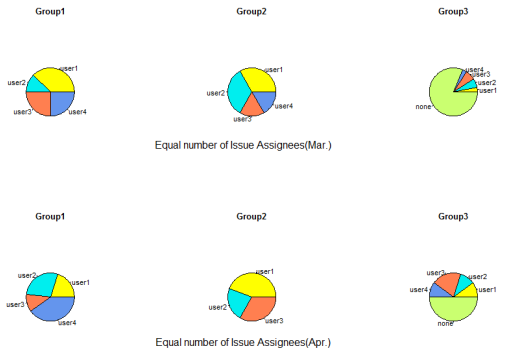
The boxplots in figure 5 shows the number of issues which had missed the milestone due date. As mentioned in section 7.5, when an issue was assigned a specific milestone, it automatically got a due date from the milestone. For those issues which were not marked as closed before the due date were ”expired” issues.

8.6 Equal number of Issue Assignees

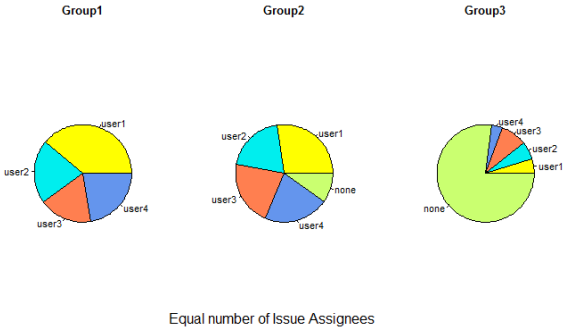
The pie charts in figure 6 shows the distribution of issues which had been assigned to each users. As mentioned in section 7.6, a group with good planing and cohesiveness should have even distributions of issue assignees.



(a) Data in first two months



(b) Data in last two months



(c) All the data in four months

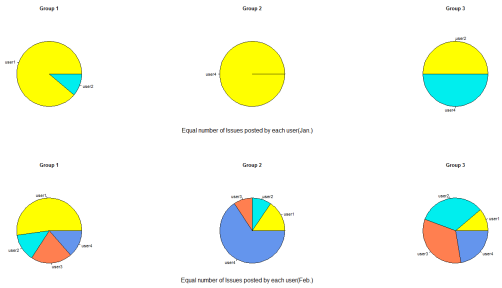
Figure 6: Distribution of Issue assignees

8.7 Equal number of Issues posted by each user

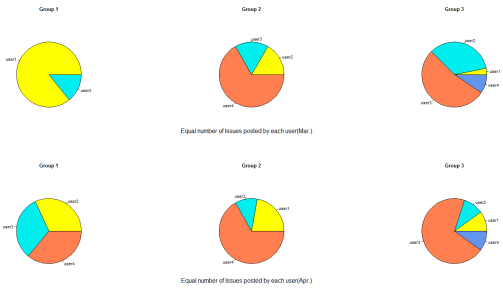
The pie charts in figure 7 shows the percentage of issues which were posted by each users. As mentioned in section 7.7, this data could indicate the involvement of one group member for the team project. And a good team should have the feature that all its members equally devote themselves to the work.

8.8 Average number of comments per issue

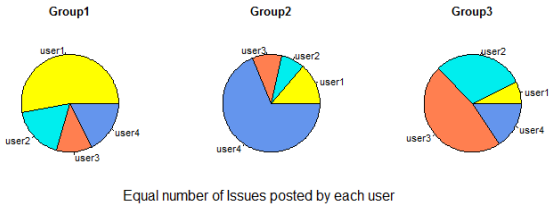
The histograms in figure 8 shows the average number of comments for each issue in each group. As mentioned in



(a) Data in first two months



(b) Data in last two months



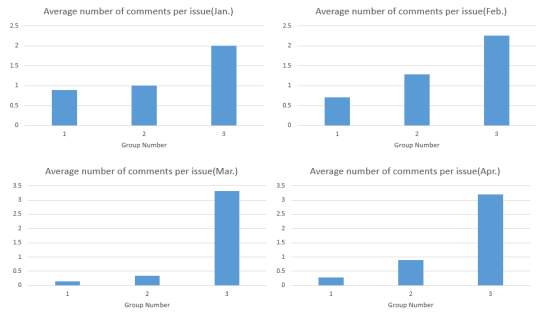
(c) All the data in four months

Figure 7: Percentage of Issues Posted by Each Users

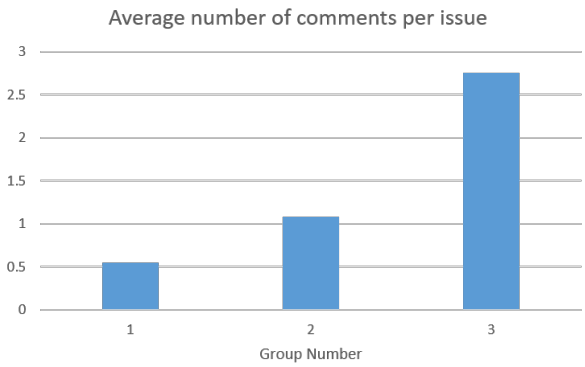
section 7.8, we considered the number of comments was a good reflection of the effective communication between the group members. And clearly, the better with the higher number.

8.9 Number of comments per user

The boxplots in figure 9 shows the number of comments for each user in each group. Certainly the idea situation would be a high number, which indicates that the group numbers communicated frequently during their work.

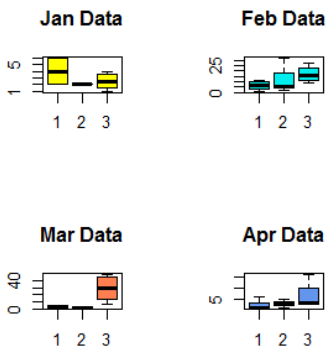


(a) Data in each month

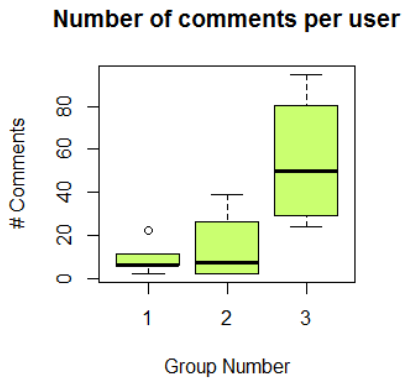


(b) All the data in four months

Figure 8: Average Number of Comments Per Issue



(a) Data in each month

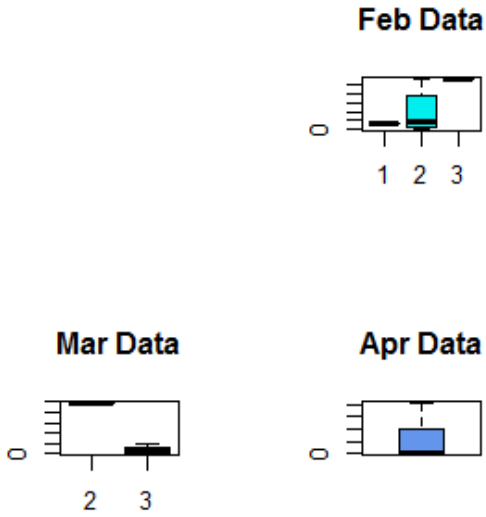


(b) All the data in four months

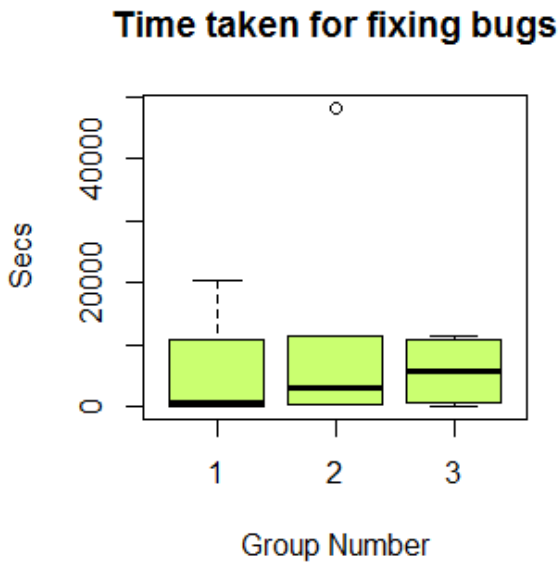
Figure 9: Number of Comments Per User

8.10 Time taken for fixing bugs

The boxplots in figure 10 shows that the time for each group spent on fixing bugs. It is such a common phenomenon that almost every one has to fix all kind of bugs during their development process. And from the point of view, we can say that shorter duration is better. While in figure 10(a), we have a blank graph and some missing data for some groups, and the reason is that those group did not have any issue labeled with bugs in a certain month.



(a) Data in each month

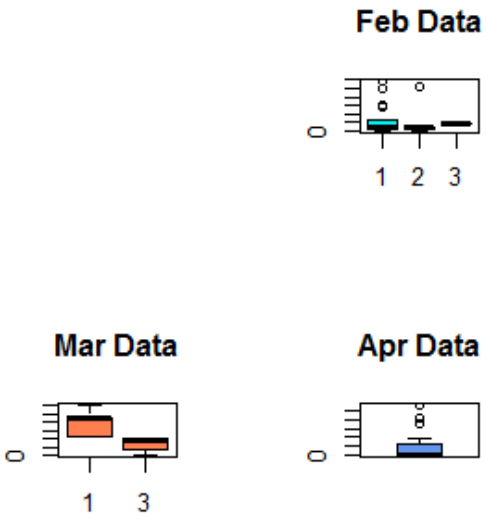


(b) All the data in four months

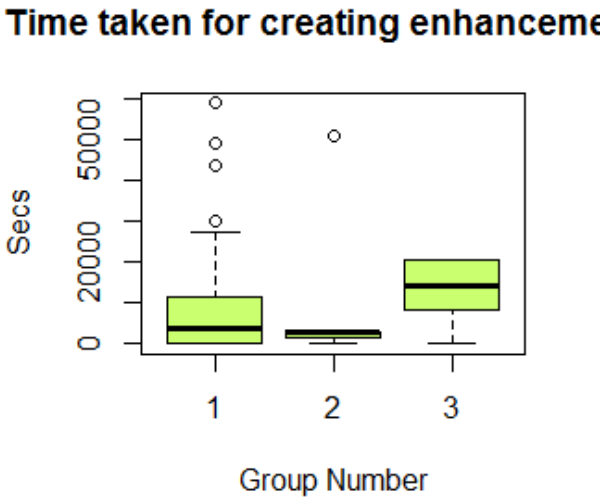
Figure 10: Time Taken for Fixing Bugs

8.11 Time taken for creating enhancements

The boxplots in figure 11 shows that the time for each group spent on gaining enhancements. Besides bugs, we also considered enhancements a good indicator of how hard the group worked for the projects. And certainly, shorter time is preferred. As the same case in figure 11(a), we have a blank graph and some missing data for some groups, and the reason is that those group did not have any issue labeled with enhancements in a certain month.



(a) Data in each month

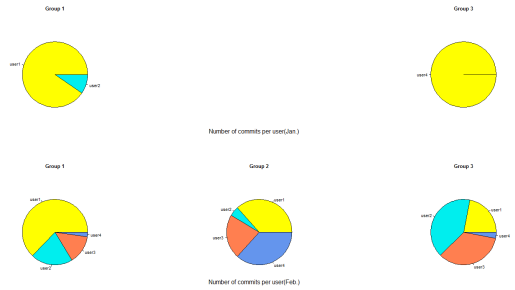


(b) All the data in four months

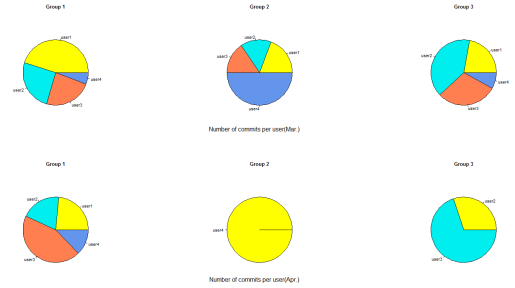
Figure 11: Time Taken for Creating Enhancements

8.12 Number of commits per user

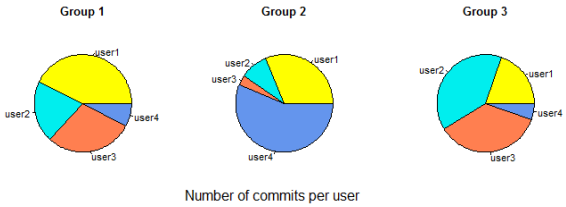
The pie charts in figure 12 shows that the percentage of commits for each user. Apparently, the idea case is that all the commits in one group are evenly partitioned among its members. And according to these graphs, the interesting thing is that there is a dictator at each group in some months.



(a) Data in first two months



(b) Data in last two months

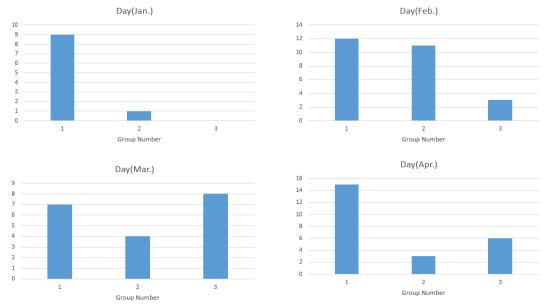


(c) All the data in four months

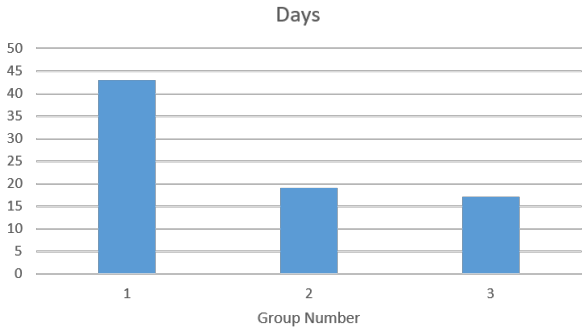
Figure 12: Percentage of Commits for Each User

8.13 Active number of days per repository

The histograms in figure 13 shows the number of active days for each group. For each group, we expected four-month work. We consider the day is an active day when there is at least one commit. So, when the number is higher, we would say that their behavior is better.

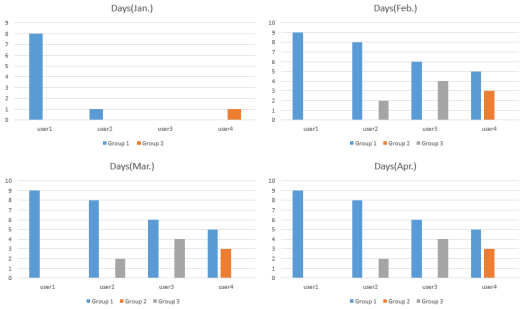


(a) Data in each month

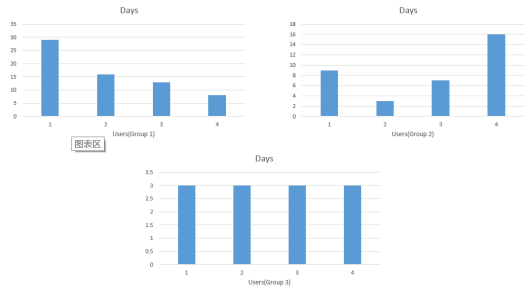


(b) All the data in four months

Figure 13: Active Number of Days Per Repository



(a) Data in each month



(b) All the data in four months

Figure 14: Active Number of Days Per User

8.14 Active number of days per user

The histograms in figure 14 shows the active days for each user in the last four months. As the same case with section 8.13, we expected the number to be as higher as possible.

8.15 Spread of event history

The curve chart in figure 15 shows the spread of events history over time. From the graph, we can see that almost each peak of the graph appeared around the end of month, which indicates that most of the group member would not try their best to work on the project until the due date. And we consider the slope of those curves, and if the slope for some point increase dramatically we would say that it is a bad one.

9. BAD SMELL DETECTOR

We create 5 different bad smell detectors that cover all our 15 feature extractors. In this section, we described the purpose of each detector and how they help us in detecting bad smells. In fact, the detectors are more about high-level bad smell in terms of design, management, team and time, but not GUI, coding or test.

9.1 Poor team cohesiveness

Team is the basic structure of how projects, activities and tasks are being organized and managed all over the world. The major advantages of teamwork are the diversity of knowledge, ideas and tools contributed by team members, and the camaraderie among members. A characteristic commonly seen in high-performance teams is cohesiveness. Those in highly cohesive teams will be more cooperative and effective in achieving the goals they set for themselves. Lack of cohesion within a team working environment is certain to affect team performance. It uses six feature extractors:

1. Issues Without Assignees
2. Average number of comments per issue
3. Number of commits per user
4. Number of comments per user
5. Equal number of Issue Assignees
6. Equal number of Issues posted by each user

For Issues Without Assignees, it means that some members may don't know which one to assign the issue or they don't know others well. If there is too many issues without assignee, we can induce that the team has low cohesion.

For the rest features, they all reflect the team member's participation. If a person has higher number of comments, we can see that he is positive and participate the project a lot. And also, if every member has similar number of issues, comments or commits, we can conclude that there is a good communication in the team and everyone doesnâĂŹt do too much or too little work.

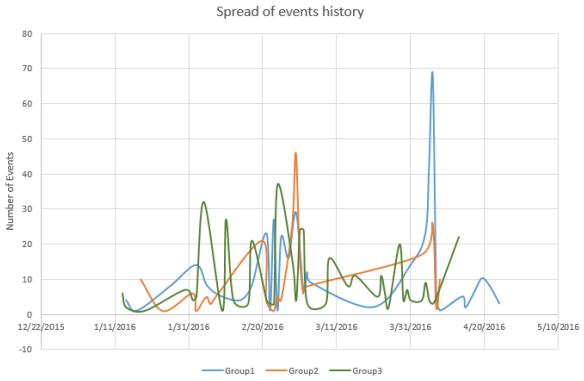


Figure 15: Spread of Event History

9.2 Poor budget

The budget contains many things and is very important to project. Here we are not talking about money, but about how we analyze the project cost at the very beginning. The budget is a detailed estimate of all the costs required to complete project tasks. It determines how to divide the project into separate parts and what cost of each part, which includes how many people is assigned to a particular part , how long will taken to implement and when the part must be done. This detector contains the following feature extractors:

1. Long Open Issue
2. Short Open Issues
3. Issues Missing Milestone Due Date
4. Time taken for fixing bugs
5. Time taken for creating enhancements
6. Active number of days per repo

For the Long Open Issue, if an issue is open and doesn't finished for a long time, it means that the design of each phases or parts may not have good boundary. There may be a heavy-work in one parts and cause overrun.
For Short Open Issues and Issues Missing Milestone Due Date, it reflects the same situation that at the beginning, or plan phase, our division of project is not suitable.
For Time taken for fixing bugs and Time taken for creating enhancements, we can see two things from these feature extractors, one is that the system performance target in planning may be too low and leaves us many time to enhance, the other one is that there may be a lot of mistakes when design the system, which result in more bugs.

9.3 Poor issue usage

Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. As github is the main tool to manage the project, it is essential for every team members to be familiar with github. Ont of the most frequently things we use is issue. It does not only record what we need to do and have done, but also a communication tool that let others understand what we have done and plan to do. In a Chinese ancient saying: if you want to do things well, first make every tools ready for you. Poor issue usage will delay and mislead project development. This detector combines the following metrics:

1. Issues Without Milestones
2. Issues Without Assignees

For both Issues Without Milestones and Issues Without Assignees, they are not a good way to use issues. This will make other members confused about what the issue is for and may cause important information missing.

9.4 Poor milestone usage

This bad smell detector is similar to poor issue usage. Milestones are great at helping everyone work towards a goal and splitting work into separate phase. They are usually used to track the progress of similar issues and pull requests as they're opened and closed over time. At a glance, people can easily see the progress of work in a milestone's lifetime. The feature extractors attached to this bad smell detector are:

1. Issues Without Milestones
2. Issues Missing Milestone Due Date
3. Active number of days per repo

For the first feature extractor, if too many issues missing milestone, it is hard for people to manage project based on timeline. It will make the process disordered and hard to track issues.
For Issues Missing Milestone Due Date, it may not because people didn't work hard, but for the reason that the setting of milestone is not suitable.
For Active number of days per repo, if the time is too little, it means that people didn't separate the work appropriately on the timeline and work seems too intensive.

9.5 Dictator

The idea with this bad smell detector is that if a dictator appears in team, it usually means that he did most of the work and other members rely on this person strongly. It is not a good status for a team and has many bad effect, such as cause other members lazy and loss better solution. In the team, people may feel they can get a good score without any hard work and then they will become careless about the project everything is well dealt with by the dictator. And also when people donâĂŹt pay much attention to the project, they won't have good meeting and discussion, so wonâĂŹt result in optimal solution when they confront problems.

1. Equal number of Issue Assignees
2. Equal number of Issues posted by each user
3. Number of comments per user
4. Number of commits per user

5. Active number of days per user

For all these 5 feature extractors, they reflect how people participate in the project. If the number of issues, comments and comments of a person is much more than other members, he may be the dictator in this team.

10. BAD SMELL RESULTS

In this section, we used Multi-criteria Decision making methods to compare the three groups. Our aim is to identify the group with the most bad smells, so we have used Multi-criteria decision making methods on the median values to compare the groups. We have employed Simple Weighted Additive Method which is the simplest and the most often used multi-criteria decision analysis method for evaluating a number of alternatives in terms of a number of decision criteria.

The method [3] is based on weighted average. If a MCDA [4] problem is defined on m alternatives and n decision criteria, such that W_j denotes the relative weight of importance of criterion C_j and a_{ij} is the performance value of alternative A_i when it is evaluated in terms of criterion C_j , the total importance of alternative A_i is defined as:

$$A_i^{SAWM-Score} = \sum_{j=1}^{n} (W_j a_{ij}) \, \forall i = 1, 2, 3, ..., m$$

SAWM works only with benefit criteria that is, the higher the values are, the better it is. As the user test criteria contains both benefit as well as cost criteria (that is, the lower the values are, the better it is) and so convert the cost criteria to benefit criteria and normalize the matrix using the equations: When the indicator is of the type more is better, we have:

$$\bar{R}_{ij} = 1 - \frac{R_j^* - R_{ij}}{R_j^* - R_{*j}} = \frac{R_{ij} - R_{*j}}{R_j^* - R_{*j}} \, \forall i, j$$

When the indicator is of the type less is better, we have:

$$\bar{R}_{ij} = 1 - \frac{R_{ij} - R_j^*}{R_{*j} - R_j^*} = \frac{R_{*j} - R_{ij}}{R_{*j} - R_j^*} \, \forall i, j$$

Here R_{ij} outcome achieved by the i^{th} system when is evaluated according to the j^{th} indicator; R_j^* = optimum value of the j^{th} indicator (ideal value); R_{*j} = worst value achieved by the j^{th} indicator; $\bar{R}_{ij}$ = normalised value achieved by the i^{th} system with respect to the j^{th} indicator.

We have considered equal relative weight of importance for all the criteria. The final results obtained from SAWM is shown in Table 7. When all the 15 criteria are considered, Group 1 has the least weighted bad smells whereas Group 3 has the most. All the results and images are available in this link. https://goo.gl/G1lbB4

Group	1	2	3
SAWM results	8.489	7.868	6.545

Table 7: SAWM Results

11. EARLY WARNING

We examined all 15 feature extractors and picked *Spread of event history* as our early warning detector. This detector looks at the slope of the number of events change and it is time sensitive. Our justification is that projects with a good design and high team cohesion will have a smooth line for the number of events each week. We chose events as an indicator as most of the activities in a project including planning (such as creating issues) or developing (such as commits) generates events. A dramatic increase in the number of events means that there is lack of planning and a lot of work is done before submission. An ideal scenario will be regular work at regular intervals with not high slope in the graph.

As the data is the number of events per week, we calculate the increasing and decreasing slope for the neighbouring two weeks. If the absolute value of the slope is double than the previous one, we will detect a warning. That is to say, the amount of work done is increasing or decreasing too fast.

12. EARLY WARNING RESULT

To evaluate the effectiveness of our early warning detector, the warning point is shown in the graph:

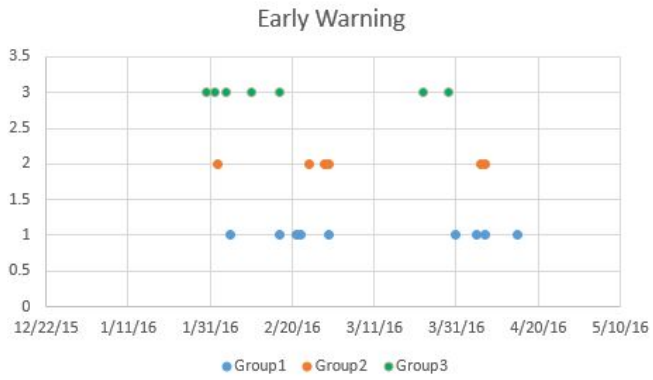


Figure 16: Early warning result

The Y-axis separates the three group. The numbers in the range of 0.5 does not mean anything. In the graph, we can see that Group 1 confront the first early warning at 2/5/2016, Group 2 confront the first early warning at 2/2/2016 and Group 3 confront the first early warning at 1/30/2016. It means that their workload increasing very fast at those time point. They are all around the project due time. We can see that just before the second due 3/1/2016, Group 1 has an early warning at 2/29/2016, Group 2 has an early warning at 2/28/2016, Group 3 doesn't have an early warning. Also from the whole graph, we can see that group 3 has less early warning and do better than the other two group. The finding is consistent with the belief that dramatically increasing of work is one of the bad thing that may cause project failure later.

13. ACKNOWLEDGEMENT

We would like to thank Dr. Tim Menzies and the teaching assistant Shaown Sarker for giving us valuable advice in implementing the project.

14. CONCLUSION

In this project, we learnt about using GitHub Database API and stored the data from three random repositories in a shared MySQL database. Then we extracted some features to identify which GitHub repositories were well maintained using Multi-criteria Decision Making. Further the early warning indicators provided in the paper can help GitHub users to make suitable changes early so as to avoid ending up with a poorly managed project.

15. REFERENCES

[1] API GitHub Wiki. `https://developer.github.com/v3/`.

[2] GitHub Wiki. `https://en.wikipedia.org/wiki/GitHub`.

[3] C. W. Churchman and R. L. Ackoff. An approximate measure of value. *Journal of the Operations Research Society of America*, 2(2):172–187, 1954.

[4] N. Sen, A. Ghosh, A. Saha, and B. R. Karmaker. Sustainability status of indian states: Application and assessment of mcdm frameworks. In *Computational Intelligence in Multi-Criteria Decision-Making (MCDM), 2014 IEEE Symposium on*, pages 78–85. IEEE, 2014.