University of Passau

Lab Report

Mining Software Repositories Lab

Realized by Group 10 :
Waleed Abunafiseh
Eya Baklouti
Ameer Anqawe

Academic Advisors:
Prof. Dr. Steffen Herbold
Alexander Trautsch
Lukas Schulte

Winter Semester 2022-2023

Contents

List of Figures

List of Tables

Chapter 1: Self-Admitted Technical Debt

1.1 Motivation

Static file analysis is one of the techniques that allow developers to analyze code before executing it. This helps finding bugs, errors, issues, that might occur. Such techniques are used in developer tools, which can lead to higher efficiency in the software development process by supplying the developers with immediate feedback and insights on their code. Furthermore, finding bugs as early as possible is critical to the quality of the software. This Experiment will focus on studying therelation between Self Admited technical Dept in software repositories and commit messeges to the general complexity of the code.

1.2 Data Extraction

This experiment will focus on the commit messages and comments within the repositories. To collect the data, a script has been developed whichclones the java repositories to be analyzed. Shown in figure 1.1 (appendix) are the steps of the script to collect and clean the data. The script has used different libraries such as PyDriller to traverse the commits and filter the modified java files to fetch the needed methods before and after the requested changes in a commit. The experiments will calculate the **cyclomatic complexity CCN** and **Self-Admitted Technical Dept SATD** of each file in every commit, comparing between the value before and after the commit.
The Repository used were: TareqK/Jesse ,javalin/javalin, apache/commons-net ,apache/commons-vfs

1.3 RQ1: Does SATD in code comments correlate with high code complexity?

1.3.1 Experiment Design

RQ1 focus on studying the affect of the complexity and the SATD. To study this corelation, complexity and SATD value should be calculated before and after each change and compared. Hence for this experiment, there will be two samples used, a sample which has changes in SATD and a sample with stable SATD value.

- Sample 1 : SATD changed
- Sample 2: SATD didn't change

In the process of the filtering and splitting, two features has been added: **diff_satd** (difference of SATD between two versions) and **diff_ccn** (difference of SATD between two versions).

1.3.2 Comparing samples

1.3.2.1 Hypothesis testing

Hypothesis of the first research question is as follows:

- H0: SATD in comments changes has no effect on the CCN
- H1/Ha: SATD in comments changes have effect on the CCN
- Significance level:0.05

Figure 1.2 bellow visualize the distribution of the code complexity variation between the two samples. The Plot box show highly unbalanced sample sizes. Sample 1 epresents 0.2% of sample 2 size.

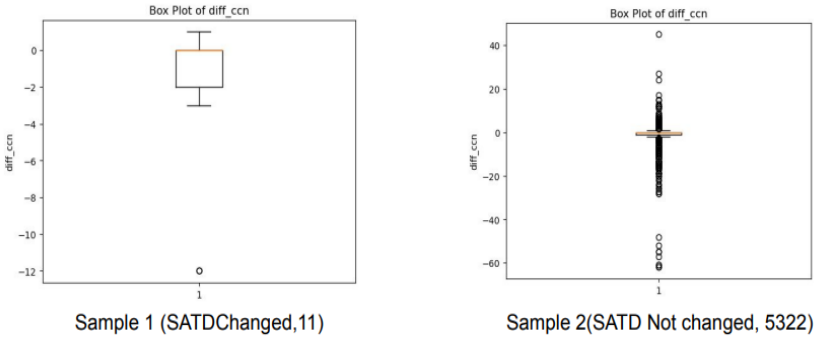

Figure 1.2: Box plots of Sample 1 and 2 code complexity variation distribution (comments)

Sample 1 (SATDChanged,11)

Sample 2(SATD Not changed, 5322)

1.3.2.2 Testing

Despite the previous results tests to compare the two samples have been carried. During this testing, a normally distributed data is assumed for the experiment. The tests and their results are as follows:

- **T-test** : t-statistic:1.6, p-value: 0.08
- **Mann-Whitn**ey U test : U-statistic:1.3 p-value:0.16

The results are not significant the samples are extremely unbalanced and we also assumed a normal distribution during the testing which is not the case.

1.4 RQ2: Does SATD in commit messages correlate with high code complexity?

1.4.1 Experiment Design

For this question we did a similar filtering as for the RQ1 but this time we filtered the data that we collected before and kept only commits where the CCN of the method changed then splitted the data into two samples:

- Sample 1 : SATD changed
- Sample 2: SATD didn't changed

1.4.2 Comparing samples

1.4.2.1 Hypothesis testing

The hypotheses for research question 2 is as follows:

- H0: SATD in commit messages has no effect on the CCN
- H1/Ha: SATD in commit messages has an effect on the CCN
- Significance level:0.05

Figure1.3 is a box plot which visualize the code complexity variation distribution between the two samples.The data is facing a similar issue due to the size of the data similarly to RQ1. Hence, the data is unbalanced meking it not possible to conclude any significant affect.

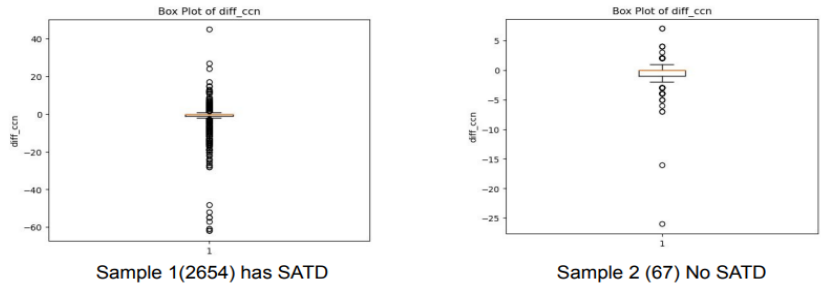

Figure 1.3 : Box plots of Sample 1 and 2 code complexity variation distribution (commits)

1.4.2.2 Testing

Despite the previous results again we wanted to try some tests to compare the two samples and we also assumed that we are having normally distributed data. The tests and their results are as follows:

- **T-test : t-statistic**: -1.1 , p-value: 0.3
- **Mann-Whitney U test** : U-statistic: -0.9, p-value: 0.35

Unfortunately the results are significant. We can't interpret it since the data is extremely unbalanced and the test assumption that the data distribution is normal is not true.

1.5 Conclusion and discussion

To answer RQ1: According to the results of the experiment, the data is unbalanced, and the conducted test and plots were not significant. we couldn't come up with a clear interpretation so our answer for RQ1 is we can't tell whether the SATD in code comments correlates with high code complexity.

To answer RQ2: Same as RQ1 we faced the exact same issues that made our analysis and tests uninterpretable. The answer would be that we can't tell whether the SATD in commit messages correlates with high code complexity.

For both RQ1 and RQ2: We couldn't come up with a defined answer because of the data quality, the repositories that we used do not have a significant number of methods where the SATD is changing this can be fixed if find repositories where in a lot of methods have SATD changed . Moreover the algorithm for SATD patterns detection works with pattern matching, but usually, they tend to have a good precision and a bad recall and in the tool that we have used there patterns that it is matching with is so short even after adding more patterns the improvement wasn't really significant . So to further improve the results we can try other advanced techniques like Natural Language Processing and Machine Learning .

Chapter 2: Pull Request Analysis

2.1 Motivation

Open source contributions depend on pull requests submitted by the maintainers of the projects. Pull requests are a request to submit changes in a code Repository. They contain the changes per line level and more information about the developer and changes in other fields organized within commits. Studying these Pull requests allows supporting developers who wish to contribute to different projects and could eventually help maintainers to automate processes and evaluations of pull requests. Understanding these pull requests will aid the automation process of testing and maintaining a level of maintainability to the project. Hence, this chapter focuses on studying different properties of pull requests as well as the reason behind these requests.

2.2 Data extraction

We started by extracting the necessary data as it is shown in Figure 2.1. The process takes a repository URL as an input and gives as an output two datasets one contains comments and review comments from the repository, we need this dataset for RQ1, and the other dataset contains the following features:

- Num_commits: number of commits per Pull Request.
- Age: a number of days since the creation of the repository (This feature was created by us)
- Added_lines: lines added in Pull request, from all files.
- Deleted_lines: lines added in Pull request, from all files.
- Changed_lines: lines changed in Pull request, from all files.
- Num_files: number of lines edited.
- Reviews_num: a number of reviews on the pull request.
- Comments_num: number of comments on the Pull request.
- Commits_word_count: number of words in the commit (This feature was created by us)

2.3 Approach RQ1: What are the reasons for changes in pull requests?

2.3.1 Inductive Coding:

We performed the inductive coding on a small sample of data from commons-configuration and commons-lang repositories the sample contains 49 pull requests we did two iterations through which we created the following set of labels: *Code Quality, Styling, Versioning, Typo, Test, Bug, Documentation, Requirements, Enhancement*

2.3.2 Deductive Coding

The next step was labeling the same sample by two different labelers independently. Figure 2.2 shows the distribution of the labels from the point of view of each labeller. we can notice a disagreement regarding labels *Typo, Code Quality* and *Styling* but they almost agreed on all the other labels.

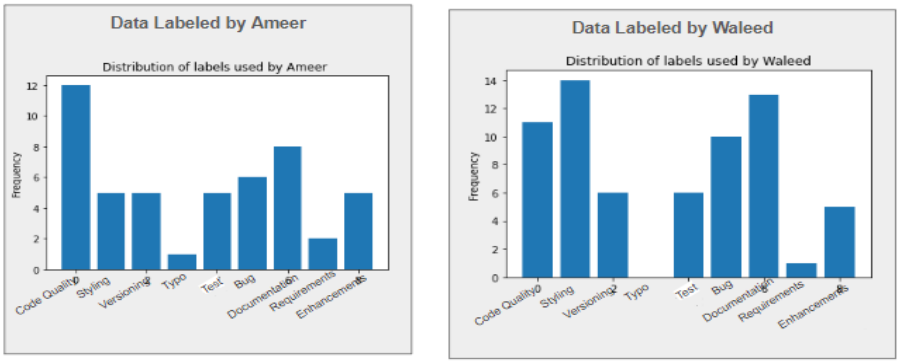

Figure 2.2 – Deductive Coding

2.3.3 Inter-rater agreement

To measure the agreement between the two labelers we used **Cohen Kappa Score**. The results showed a Score of **0.36** which indicates a fair agreement between the two labelers.

2.4 Approach RQ2: Can we predict whether a pull request will be merged?

2.4.1 Data characteristics

For RQ2 we tried to collect data from different types of repositories, Table 2.1 summarizes the collected data we tried to select repositories with different sizes, programming languages, team sizes, etc.
In total the size of the gathered data is 3321 and it is distributed as follows:

- merged (True): 2015
- non-merged (False): 1306

Repository	Number of PR	Language	Starts	Forks	Contributors
alibaba/arthas	700	Java	31.9k	6.9k	169
vuejs/router	955	JavaScript	2.7k	896	179
apache/commons-math	222	Java	492	339	52
nodejs/node-gyp	679	JavaScript	8.9k	1.7k	211
adamchainz/django-mysql	765	Python	516	106	28

Table 2.1 – Summary of repositories characteristics

2.4.2 Feature Correlation and importance

We measured the correlation between all the features that we have and we found that the correlation between *Num_files* and *Added_lines* is **0.97** which means that they are highly correlated. So we decided to remove the feature *Num_files.* Another experiment we did on the features is that we tested the model performance before and after adding the *Age* and we noticed that there is an improvement in the performance which means that the *Age* is a relevant feature. We did the same with commits_word_count but we found out that it had no effect on the model which means that it is not a relevant feature .

2.4.3 Modeling

To select the most suitable model for our use case we tested 5 models using the **10-Fold Cross-validation** technique and to score them we used **Mean accuracy** as a metric. The results the performance of the model was very close but K Neighbors Classifier gave the best result with 0.63 mean accuracy. After choosing the model we trained it using 80% of the dataset that we have and we kept the rest for testing.

2.4.4 Model Evaluation

2.4.4.1 Model Performance

To evaluate the model performance following metrics were used:

- Precision:0.64
- Recall: 0.79
- Accuracy: 0.63
- F1-score:0.71

The results show an acceptable but not good enough precision and accuracy but the recall and the F1-score are so good this means the model may make a lot of mistakes when predicting whether the repository may be merged or not but the model would always identify almost all the cases where the repository is merged. This can be confirmed and visualized by the confusion matrix in Figure 2.3 by the high rate of True Positives, and low rate of True Negatives.

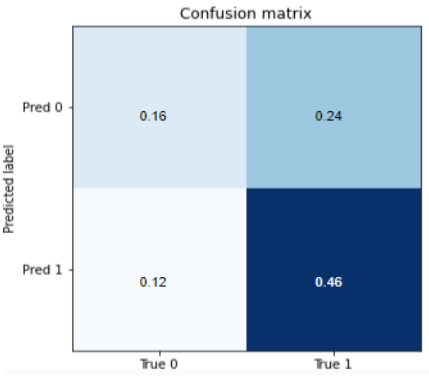

Figure 2.3 – Confusion matrix

2.4.4.2 Comparison with a baseline

We also tried to compare our model performance with a baseline model performance. As a baseline we choose a random classifier with a distribution: 40% non_merged, 60% merged. The table below shows the difference using the previous performance metrics. Our model clearly outperforms the baseline when it comes to Recall, F1-score, and accuracy.

	Precision	Recall	Accuracy	F1-score
Baseline	60%	60%	51%	60%
Our model	64%	79%	63%	71%

Table 2.2 – Performance difference between our model and the baseline

2.5 Conclusion and discussion

To answer RQ1: There are multiple reasons for changes in pull requests and by observing the Pull Requested comments and review comments we were able to extract a preliminary set of those reasons which are: *Code Quality, Styling, Versioning, Typo, Test, Bug, Documentation, Requirements, Enhancement*.This list can be extended by studying more Pull requests from different repositories. Moreover Labeling the Pull Requests with their associated reasons from the list can be automated using NLP and machine learning techniques.

To answer RQ2: Yes, We can predict whether a pull request can be merged or not. The model that we trained already does the job and the performance can be improved by extracting more features that are relevant, or maybe using the labels of the reasons that we mentioned them in RQ1 as additional features too and we should also extract more data from a lot and different repositories.

Chapter 3: Readability Analysis

3.1 Motivation

In light of the growing number of software projects that are being written and maintained. The Importance of writing well-written and documented software is high. Readability analysis is important for the Maintainability of code, Collaboration, Code quality, Time and cost savings, User satisfaction Automation, and CI/CD pipelines.

3.2 Experiment design

Focus of this experiment is to investigate how the readability metric of source code could be utilized and used in prediction models. Hence, this experiment will study 4 different Java open-source projects. These repositories will be extracted using a script that will fetch all the commits within a repository and study the readability of each changed file. Then, this data will be further manipulated and used to answer the two proposed research questions by comparing two different samples of increased and decreased readability in commits and their relation to specific words in commit messages.

3.3 Data Extraction

The data extraction process of the experiment focused on retrieving all the needed information for the research questions. It is crucially important to gather the commit message, commit hash, commit date as well as the number of changed files within each commit.
Furthermore, for each changed file, in each different commit, the file must be analyzed before the change and after the change to retrieve the complexity value of the file and the number of lines of code changed.

3.4 RQ1: Do developers deliberately increase readability?

This experiment will examine the commit message as an indicator of ‘deliberately’ increasing the readability. As it is one of the only possible ways to study the intention of the developers. Hence, despite that in each commit, there could be more than one file changed, which could be both increasing and decreasing. In order to avoid this issue, the changed files were aggregated per commit by taking the sum of the readability of the changed files. Hence subtracting the new (after a commit) from the old value of a file results in a positive value for increased readability.
The data sets will be split into two different samples in order to compare the samples of increased readability with the decreased readability sample.

3.4.1 Text Processing and exploration

In order to correctly use the commit messages as an indicator of the deliberate declaration of improving the readability of a file, the bag of words model was used to collect the frequencies of words. Hence, further text processing is needed. The two samples were processed by removing the stop words, tokenization, stemming, and finally removal of unwanted repeated words like: *'git', 'apache', 'svn', 'id', 'method etc*
The initial Word clouds and the frequencies of the words have shown half of the top terms repeated as shown in figures 3.0 and 3.1.
After the initial exploration of the data, it is noticeable that there is a crossover of the two populations and there needs to be further filtering in order to find a correlation. Hence, the mean of the rate of change of readability of the two different samples is calculated.
The mean of the readability of the increased group was 0.01107 and -0.01203 for the decreasing readability. Finally, the two samples were adapted to include values above and below the mean values respectively. This change has resulted in cleaner data that could be compared and includes less common words between the two samples as shown in figures 3.2 and 3.3.

3.4.2 Word Clouds:

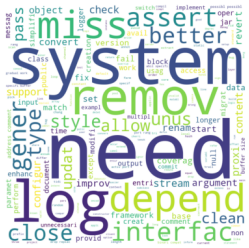

Figure 3.3 – Decrease Readability Word Cloud (using mean)

Figure 3.2 – Increase Readability Word Cloud initial (using Mean)

3.4.3 Suggested keywords lists

The keywords which have been suggested to filter the data include the following two sets:

- Increased readability:'support', 'set', base, allow, 'assert', 'implement', 'local', 'line', ‘configur’, 'miss',’ refactor’, ‘enhanc’, ‘clean’,’simplifi’,’duplicat’
- Decreased readability:'implement', 'lock', 'need', 'miss', 'base', 'system', 'better', 'close', 'check', 'remov', 'log', 'simplifi','featur',’ conflict’, ‘configur’,’paramet’

3.4.4 Distribution of commit message readability variation based on keywords extracted

The following graphs show the variation in readability for both samples with and without the list of the extracted and suggested words.

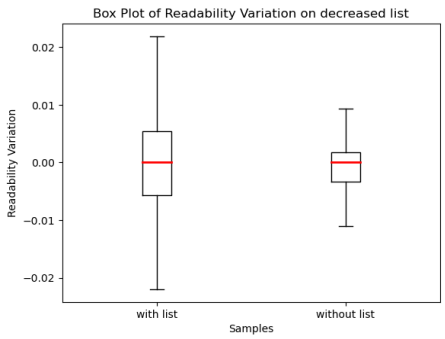

Figure 3.4 Box Plot variationwith list on decreased sample

Figure 3.5 Box Plot variation with the list on increased sample.

3.5 RQ2:Can we predict whether the readability of a file will be improved?

3.5.1 Feature importance

In order to know the importance of the words within a commit to the prediction if file readability has been improved. The data has been split into two groups a group where the readibility has increased and a group where the readability decreased. Then Analyzed each commit to retrieve how many word from the lists of keywords created in RQ1 it contains. We notice that we were able to find some of the words from our created list some of them even twice in one commit but in most cases we didn’t find any of them this may be because our lists are short somehow , and the fact here we have a diffrent repositories that the ones we extracted the keyword list from which leads us to the conclusion that our lists of keywords are not relevant in this case.

3.5.2 Model Evaluation

For modeling we used a Random Forest Classifier with Tf idf Vectorizer and 80% of the data for the training and 20% for testing.

3.5.2.1 Evaluation within-project

Repository	Commons-vfs (1708)	Commons-codec (1122)	commons-bcel(1181)	javalin(469)
Performance	Accuracy: 59% precision: 61% recall: 22% f1: 33%	Accuracy: 59% precision: 52% recall: 31% f1: 39%	Accuracy: 56% precision: 54% recall: 39% f1: 45%	Accuracy: 56% precision: 45% recall: 28% f1: 34%

Table3.1: Model evaluation across within-projects

3.5.2.2 Evaluation Cross-projects

Train data repository	**Test data repository**	**Performance**
commons-codec , Commons-bcel (2285)	commons-vfs (1708)	Accuracy: 58%, precision: 46% recall: 39% ,f1: 42%
Commons-vfs commons-bcel (2866)	commons-codec (1122)	Accuracy: 58%, precision: 47% recall: 32%, f1: 38%
Commons-vfs, commons-codec, (2811)	Commons-bcel (1181)	Accuracy: 60%, precision: 51% recall: 29%, f1: 37%

Table 3.2: Model evaluation a cross-projects

3.6 Conclusion and Discussion

To answer RQ1: Based on the initial word clouds fig 3.2 and 3.3 and the top frequencies of the words result in the conclusion that there is no correlation between the commit message and the readability of the files. This is concluded after reviewing 4 different repositories with 1000+ commits each. The top words using the bag-of-words model do not reflect any correlation with the deliberate improvement of readability on either sample.

To answer RQ2: Based on the model evaluation in Table 3.1 and 3.2 the model performance is not excellent but we can notice that it indeed able whether the readability of a file will be improved or not .The performance can be explained by the fact that the task of classifying a text is not suitable for Random Forest Classifier and with a data size that is considered small , in order to improve the performance we can use Transfromers like BERT and collect more data . In addition to that the readability scoring tool is not perfectly accurate so as an improvement we use more accurate one or improve it .

3.7 Limitations

It is crucial to note a few of the limitatooons faced during this experiment. Using the bag of words could have affected the results. There are other alternatives than bacg of words that could eventually understand the context of the messsege or what is written. Furthermore, all teh repositories used for teh study were using the same programming language, which takes into consideration best practices and how mature is a community with its guidelines which affects how a repository is used and structures. Lastly, the extracted features do not fully reflect the complexity of the data, it is important to include experience of a developer and type of pull request when analyzing the data.

Appendix:

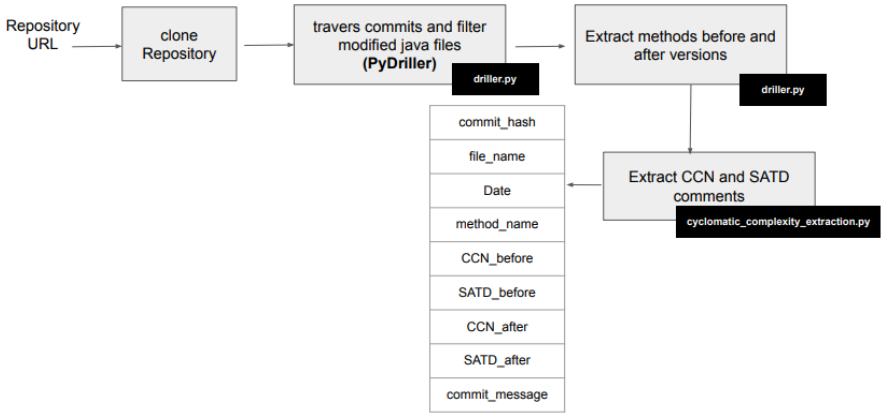

Figure 1.1 : Data extraction workflow project 1

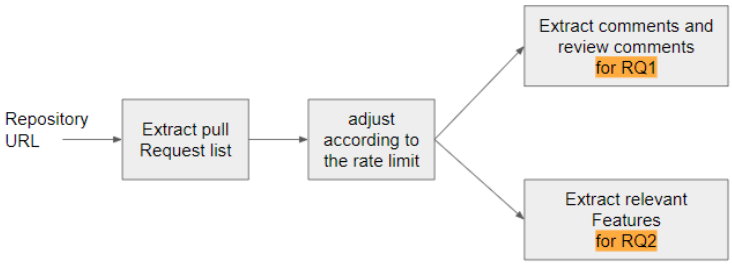

Figure 2.1 – Data extraction workflow project 2

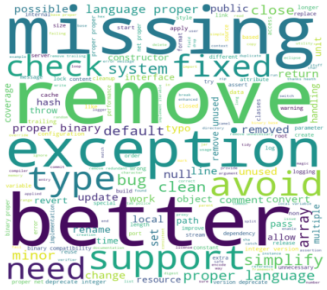

Figure 3.0 – Increase Readability Word Cloud initial

Figure 3.1 – Decrease Readability Word Cloud

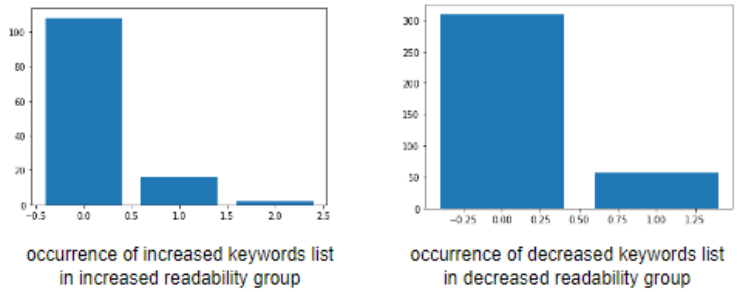

Figure 3.6 : Bar plots of the occurrence of increased and decreased keywords lists in increased and decreased readability groups

References

1. Scalabrino, Simone, et al. "A comprehensive model for code readability." Journal of Software: Evolution and Process 30.6 (2018): e1958.
2. Lee, Taek, Jung Been Lee, and Hoh Peter In. "A study of different coding styles affecting code readability." International Journal of Software Engineering and Its Applications 7.5 (2013): 413-422.
3. Zhang, Xunhui & Yu, Yue & Wang, Tao & Rastogi, Ayushi & Wang, Huaimin. (2022). Pull request latency explained: an empirical overview. Empirical Software Engineering. 27. 10.1007/s10664-022-10143-4.
4. Sophilabs. (2018, January 15). PR Prediction with Machine Learning. Sophilabs. https://sophilabs.com/blog/pr-prediction-machine-learning