Is there a Correlation Between Code Comments and Issues? - An Exploratory Study

Vishal Misra, Jakku Sai Krupa Reddy, Sridhar Chimalakonda
Research in Intelligent Software & Human Analytics (RISHA) Lab
Department of Computer Science and Engineering
Indian Institute of Technology Tirupati
cs16b033@iittp.ac.in,cs16b012@iittp.ac.in,ch@iittp.ac.in

ABSTRACT

Comments in a software code base are one of the key artifacts that help developers in understanding the code with respect to development and maintenance. Comments provide us with the information that is used as a software metric to assess the code quality and which further can be applied to demonstrate its impact on the issues in the code. In this paper, we set out to understand the correlation between code comments and issues in Github. We conduct an empirical study on 625 repositories hosted on GitHub with Python as their primary language. We manually classify comments from a randomly selected sample of python repositories and then train and evaluate classifiers to automatically label comments as *Relevant* or *Auxiliary*. We extract the metadata of issues in each repository present in our dataset and perform various experiments to understand the correlation between code comments and issues. From our dataset of python repositories, we then plot a graph between the average time taken to resolve an issue and percentage of relevant comments in a repository to find if there is any relation or a pattern by which the latter affects the former. Our statistical approach of finding out the correlation between code comments and issues gives us the correlation factor by which code comments are related to issues. We conclude from our study that comments are indeed important and play an important role in solving issues of the project. We also found that increasing the percentage of *relevant comments* along with the source code can help in the reduction of the average number of days before an issue is resolved.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; • **Software and its engineering** → *Software libraries and repositories*;

KEYWORDS

Code Comments, Issues, Correlation, Python Repositories, Empirical Study, Machine Learning

SAC '20, March 30-April 3, 2020, Brno, Czech Republic
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3374009

ACM Reference Format:
Vishal Misra, Jakku Sai Krupa Reddy, Sridhar Chimalakonda. 2020. Is there a Correlation Between Code Comments and Issues? - An Exploratory Study. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341105.3374009

1 INTRODUCTION

Code comments are the second most used documentary artifact for code comprehension of a software project, the first being its source code [4]. Comments in a code can be used to explain the programmer's intent or to summarize the code [13]. Comments are used to express the program methods and specifications with the combination of predefined tags and natural language [2]. Code comments are widely used in various aspects of programming, such as debugging, algorithm description, where explanations may include diagrams and mathematical proofs. Code comments include resources such as logos and diagrams which can be used to illustrate the design or approach followed by the developer. Code comments include pseudo-code that explains the logic behind the code. Comments in the source code often store metadata about a program file such as URL of documentation, copyrights and install instructions. Hata et al. have found around 9.6 million links in source code comments [8]. Experiments conducted by Woodfield et al. [25] and T. Tenny. demonstrate that properly commented code is more readable and maintainable [23] .

It has been observed that developers spend more time resolving issues, debugging and maintaining the code, in comparison to the development of the software [20]. Researches have shown that poor documentation causes more inconsistency and misunderstandings for the developers which affect further modification of the code [6, 14]. Minelli et al.[16] state that developers spend most of their time in understanding the code. Code comments are also considered as an important software metric that helps in improving the quality of the source code [17]. Software systems and projects contain a significant amount of code comments which are observed to help developers in understanding the code, modifying the code, solving bugs and issues in the code [22]. Studies [18, 22] show that along with the external documentation, comments in the source code provide a convenient way for developers to keep documentation and code up-to-date and consistent.

GitHub is a widely-used open source code sharing platform, and a major source of software projects, consisting of over 600K Python repositories[1] with a lot of bugs and enhancements. We

[1] https://github.com/search?l=Python&q=python&type=Repositories

chose *Python* as the primary language as it is a language of continuous advancements, with increasing amounts of projects which makes it necessary to understand the quality of these projects. Code comments have been considered as an important factor to assess the quality of code[10, 19]. Earlier, researchers have used lines of comments as a software metric to analyze the quality of software projects [17] and have studied the co-evolution of code and code comments[5, 11]. Wen et al. also provide an empirical study that discusses techniques that causes code comment inconsistency[24]. There have been studies that investigate the area of issue trackers and the linking of issue reports with the changes in source code[26]. Bissyandé et al. provide a large scale empirical study on issue trackers on github[1].

However, to the best of our knowledge, there are no empirical studies that investigate the co-relation between quality of code comments and issues, and hence motivating our empirical research.

Code comments might belong to different categories such as comments that explain the functionality of various methods in the code, program flow, commented out code, code snippets that are used for debugging purposes and so on, of which, few comments might be less useful to developers than others. To support our study, we have created a machine learning model which categorizes comments into two different categories, as follows:

(1) **Relevant Comments:** Based on [22], header comments, method comments, inline comments, task comments, section comments and code comments are included in this category.
(2) **Auxiliary Comments:** All the copyright comments and noise comments are included in this category.

With the percentage of different kinds of comments present in each repository of our dataset along with the metadata of issues of that repository, we would like to structure our further study along the following two research questions:

- *RQ1: What is the correlation between the total number of relevant comments and issues across different repositories?*
- *RQ2: How does average time to resolve an issue in a repository varies with the percentage of relevant comments?*

The rest of the paper is organized as follows. Section 2 provides background details about different categories of code comments and situates this paper with respect to the related work. Section 3 introduces the machine learning model and other methodologies used in our empirical study followed by results and threats to validity in Sections 4 and 5. Section 6 summarizes our conclusions.

2 BACKGROUND & RELATED WORK

2.1 Background

Open source enthusiasts from the community can contribute to the repository in many ways including the opening of issues, submitting pull requests and performing code reviews. Open source contribution to any repository requires a proper understanding of the source code and other related artifacts. Various studies show that comments help in documentation and understanding the project and it's code better. [4, 18, 22].

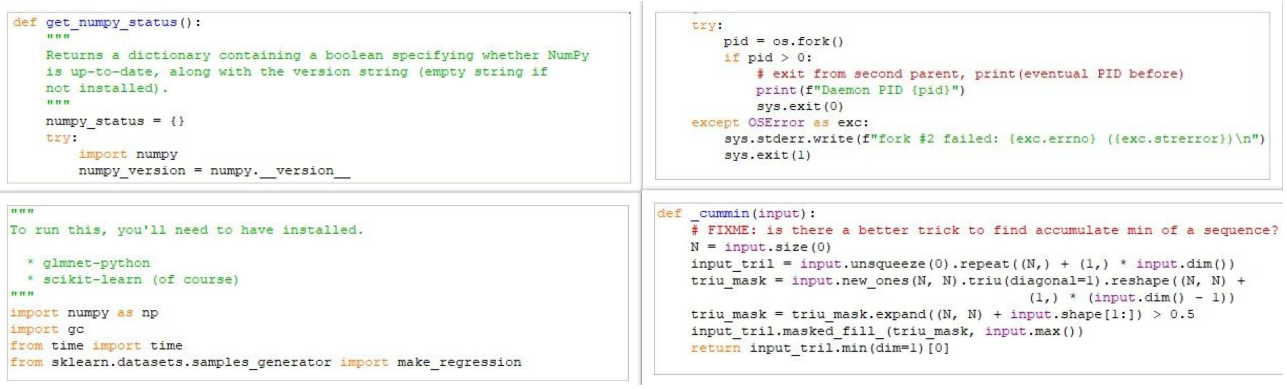
As discussed in the previous section, we have categorized comments in the source code into two categories i.e., Relevant and Auxiliary. Steidl et al. have automatically categorized comments into seven categories - *copyright*, *header*, *member*, *inline*, *section*, *task* and *code comments*, using machine learning techniques [22], which we have adopted in our study along with the addition of API and IDE Directives in relevant comments category. Steidl et al. have suggested that copyright comments should be excluded from the category of relevant comments as they do not provide any information that could be helpful in comprehending the code [22]. Auxiliary Comments category contains comments which do not provide any relevant information to developers in maintaining or comprehending the code. Noisy comments, Copyright comments come under this category. Relevant code comments are those that developers can look at to understand the source code. Relevant code comments include:

(1) **Header comments -** Gives an overview of the functionality of a class.
(2) **Member comments -** Gives an overview of the functionality of a method/field.
(3) **Inline comments -** Comments regarding implementation in method body.
(4) **Section comments -** Address several methods/fields together belonging to the same functional aspect.
(5) **Task comments -** Category of comments that tells about a bug that needs to be fixed or a todo or FixMe remark.
(6) **Code Comments -** Category of comments which consist of commented out code.
(7) **API or IDE comments -** Comments which contain all API calls and their meanings and IDE Directives.

```
import datetime
 #Single line comment --> Noisy Comment

'''Create a class named Person,use the __init__()
    function to assign values for name and age: -->
    Header comment'''
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
 '''Prints name of the person
 --> Member comment'''
  def myfunc(self):
    print("Hello my name is " + self.name)
  def length(self):
      name = self.name
      '''Prints number of characters in name
      -->Inline comment'''
      print(len(name))
  '''FIXME - write function to find number of
  vowels -->Task comment'''
  def vowels(self):
  '''Gets the current date from module datetime, then
     finds birth year by subtracting age --> Section
     comments'''
  def birthYear(self):
  '''datetime.now().year gives the current year --> API
     comment'''
      curryear = datetime.datetime.now().year
      birthYear = curryear - self.age
      print(birthYear)
p1 = Person("John", 36)
```

Listing 1: Basic examples of different types of code comments.



```
def get_numpy_status():
    """
    Returns a dictionary containing a boolean specifying whether NumPy
    is up-to-date, along with the version string (empty string if
    not installed).
    """
    numpy_status = {}
    try:
        import numpy
        numpy_version = numpy.__version__
```

```
    try:
        pid = os.fork()
        if pid > 0:
            # exit from second parent, print(eventual PID before)
            print(f"Daemon PID {pid}")
            sys.exit(0)
    except OSError as exc:
        sys.stderr.write(f"fork #2 failed: {exc.errno} ({exc.strerror})\n")
        sys.exit(1)
```

```
"""
To run this, you'll need to have installed.

  * glmnet-python
  * scikit-learn (of course)
"""
import numpy as np
import gc
from time import time
from sklearn.datasets.samples_generator import make_regression
```

```
def _cummin(input):
    # FIXME: is there a better trick to find accumulate min of a sequence?
    N = input.size(0)
    input_tril = input.unsqueeze(0).repeat((N,) + (1,) * input.dim())
    triu_mask = input.new_ones(N, N).triu(diagonal=1).reshape((N, N) +
                                              (1,) * (input.dim() - 1))
    triu_mask = triu_mask.expand((N, N) + input.shape[1:]) > 0.5
    input_tril.masked_fill_(triu_mask, input.max())
    return input_tril.min(dim=1)[0]
```

Figure 1: Different types of category comment examples from the code snippets in our dataset.

Figure 1 shows different types of category comment examples from the code snippets in our dataset.

- Quadrant 1 shows member comment example from *scikit-learn* repository.
- Quadrant 2 shows inline comment example from *pyload* repository.
- Quadrant 3 shows section comment example from *scikit-learn* repository.
- Quadrant 4 shows task comment example from *pyro* repository.

Table 1 shows examples of all kinds of relevant comments over different python repositories.

2.2 Related Work

Code comments are observed to be very much useful at various levels of building and maintaining a software project. Akash et al. have identified the usefulness of code comments in finding semantic code clones [7]. Stamelos et al. [21] presented a software metric that defines the relation between code comments and source code, which demonstrates the impact of code on the quality of software. Pascarella et al. [18] have observed that lines of code comments could be used as a metric to detect and analyze software projects. DeSouza et al. have conducted surveys which confirmed source code and comments to be the most important artifacts to maintain as a system, thus indicating the relevance of documentation in software maintenance [4]. They also concluded that qualitative code comments are one of the important parameters that contribute to proper documentation. In a similar work, Stamelos et al. [15] suggest that concise documentation and well-written code comments are important for experienced developers to maintain the project and to encourage collaborations and contributions from novice and intermediate developers. Steidl et al. have automatically categorized comments into seven categories - *copyright*, *header*, *member*, *inline*, *section*, *task* and *code comments* using machine learning techniques [22]. A further fine-grained classification of comments was provided by Pascarella et al. [18], which addresses limitations such as lack of different category of IDE Directives and improper handling of noise. As a part of our study, we have observed nine categories of comments, including the two categories- IDE and API call comments and noisy comments, in addition to the seven categories proposed by prior work from Steidl et al. [22]. As stated in the example code of Listing 1, noisy comments comprises of comments which do not provide any insight on either the description of code or on any matter regarding maintenance of code.

However, as mentioned, our primary focus is on understanding the correlation between code comments and issues. Comment types that deal with Copyright, Licensing and Authors are observed to be less useful in debugging a software project. Hence, in our study, we have grouped various categories of comments into two groups - Relevant and Auxiliary, where Auxiliary includes comments containing copyright and author information. The other categories of comments are grouped into Relevant Comments.

3 METHODOLOGY

We followed a two phase methodology to conduct our study, as shown in Figure 2.

3.1 Dataset Extraction

To perform our study, we have collected 625 Python repositories available on GitHub. We then extracted the code comments and metadata about issues (i.e both closed as well as open issues), for each of these repositories. These repositories comprise about 9000 closed issues and around 8000 open issues.

- Metadata of the issues included opening time of the issues, type of issues such as bugs and enhancements and closing times for closed issues.
- Code comments of Python files in the repositories have been extracted using *regular expressions* and *similarity techniques*.

We have extracted a summary of the total number of closed issues and open issues in each repository. By evaluating the difference between the opening and closing times of closed issues, we computed the average time taken to resolve issues in each repository.

We have used GitHub API v3 and its function calls to retrieve the dataset of issues and code comments. Repositories with Python as primary language and with more than 2,000 stars have been filtered out and extracted. For the purpose of our study, we wanted to focus more on the popular set of repositories present in Github. Further

Table 1: Examples of different types of comments from our dataset

RepoName	Header Comments	Member Comments	Inline Comments	Section Comments	Task Comments
django	This module collects helper functions and classes.	Return a *HttpResponse.*	The model's function will be called.	return *absolutemax* if it is lower than the actual total form...	Add an 'invalid' entry to default error message
scikit-learn	This example visualizes some training loss curves.	Returns a dictionary containing a boolean specifying...	Memoize the data extraction and memory map the resulting...	To run this, you'll need to have installed glmnet-python.	*FIXME*: the following snippet does not yield the same results on 32 bits.
schema	Pass your inherited Schema class to map all flags bits to a more...	Removes duplicates values in auto and error list.	for each key and value find a schema entry matching them.	Validate data using defined sub schema/expressions ensuring all values are valid...	*FIXME* given tests enough is expected to be converted to a datetime instance, but fails ...
pyro	Wrapper class for Markov Chain Monte Carlo algorithms.	Updates states of the scheme given a new statistic/subgradient...	If true, the current module name will be prepended to all description.	All configuration values have a default; values that are commented out...	*FIXME*: is there a better trick to find accumulate min of a sequence?...
catalyst	Lazy command class that defers operations requiring Cython and numpy.	Return iterator of all values in d except the values in k.	The sortino ratio is calculated by a empyrical function so testing of...	Verify that we checked up to the longest possible window.	*TODO*: the loop here could overwrite expected properties.
zulip	A helper class to turn a dictionary with ticket information into an object where ...	Returns a sorted list of unique dependencies specified by ...	In dev always include the currently active cache in order not to break ...	In the case that permissions got messed up and the deployment directory ...	*FIXME*: We should also test the Tornado URLs – this codepath can't ...
TensorFlowOnSpark	Base class for special marker objects in the data queue...	Distributed TF cluster w/ InputMode.SPARK and exception after feeding.	simulate post-feed actions that raise an exception.	define modeldir and exportdir for tests.	*TODO*: Remove once getvariable is able to colocate op.devices.
PyTorch-NLP	Class for invertibly encoding text using a limited vocabulary. Invertibly encodes a native...	Return True if check pass; if check fails and abort is True, raise an Exception, othereise return False.	Compute SRU with a GPU employing the use of CUDA.	initialize bias and re-scale weights in case there's dropout.	*TODO*: Use sklearn.metrics for a confusionmatrix implemented with ignoreindex.
pyload	Helper class that holds result of an initiated online check...	Saves a value persistently to the database.	exit from second parent, print(eventual PID before)	Download the translation catalog from the remote repository.	*TODO*: include addons that are activated by default.

studies can be investigated on the set of repositories with less than 2000 stars.

3.2 Classification of Code Comments

3.2.1 ***Manual Classification.*** We considered a random sample of 20 repositories from the retrieved 625 repositories and extracted 20,000 single-line and multi-line comments. A semi-automated approach has then been designed to classify 600 of these extracted code comments into relevant comments and auxiliary comments.

- We have inspected code comments for the presence of following words - LICENSE, COPYRIGHT, AUTHOR, and classified these comments as copyright comments. Hence, comments

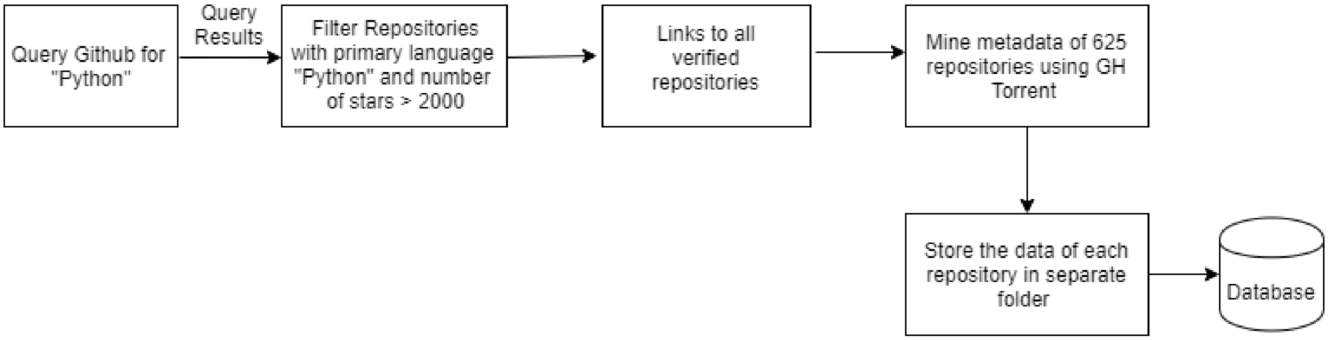

Query Github for "Python"
Query Results
Filter Repositories with primary language "Python" and number of stars > 2000
Links to all verified repositories
Mine metadata of 625 repositories using GH Torrent
Store the data of each repository in separate folder
Database

Figure 2: Dataset Collection Methodology

containing the previously mentioned words have automatically been classified as *Copyright Comments*.

- We then analyzed the code files present and manually categorized Header Comments, Section Comments, Member Comments, Inline Comments, IDE Directives and Task Comments as *Relevant Comments*.
- To classify code comments into *Auxiliary Comments*, we manually analyzed code files and categorized all the noise and commented out code into this category.

3.2.2 *Classification Technique*.

We tried several machine learning techniques such as SVM classifier, Decision Tree classifier, Logistic Regression, Random Forest Classifier, AdaBoost Classifier, Gradient Boost Classifier, to identify the best technique to classify comments. These classification techniques use different assumptions to fit the data and have their own advantages and drawbacks. We obtained different scores for different classifiers on the test data provided and observed that Random Forest Classifier provide better results than other classifiers. It provides better accuracy and F1-Score compared to other classifiers. The F1 score is the harmonic mean of precision and recall which gives equal weight to both precision and recall while calculating the scores. F1-score of the classifiers listed in Table 3 are 0.80, 0.86 and 0.79 for SVM, Random Forest and Gradient Boost Classifier respectively. Hence, we used Random Forest classifier for the classification of comments.

A four-step approach, as shown in Figure 3, has been used in building and training the machine learning model.

- **Step 1 - Data Preprocessing.** We used count vectorizer to preprocess and tokenize the text and to filter out stopwords. A dictionary of features has been built and the text is transformed into feature vectors.
- **Step 2 - Identify word frequencies.** We then applied tf-idf (Term Frequency times Inverse Document Frequency) to convert the occurrences of words into frequencies. Tf-idf also downscales the weights of more frequently occurring words, as such words are observed to be less informative than the words with less frequency in the corpus.
- **Step 3 - Test and Train sets.** Based on our analysis and experiments, we have settled on splitting our data into test and train sets, such that 30% of the data is test data. We then train our ML classifier on the training dataset.

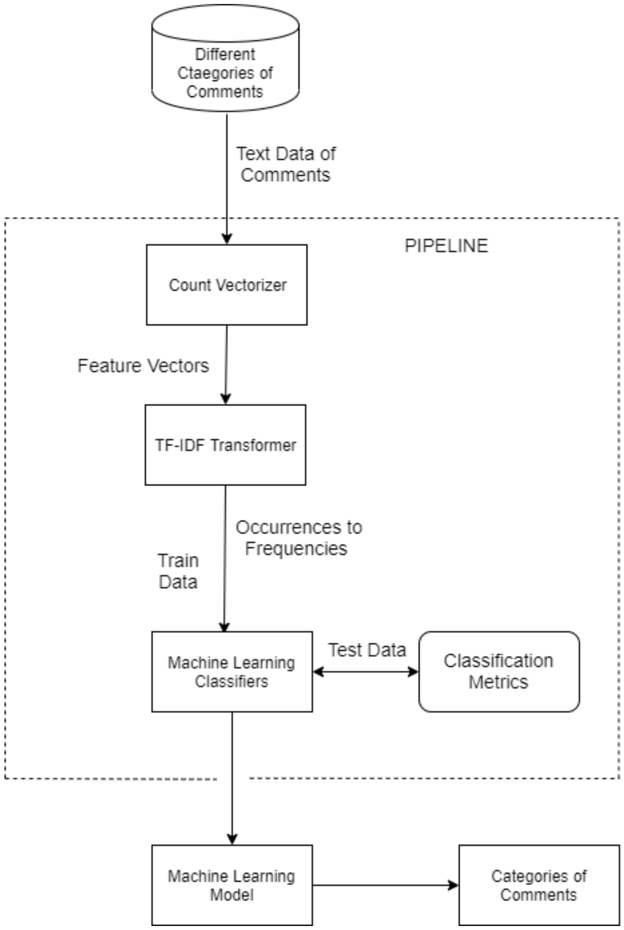


Figure 3: Pipeline showing the classification process

- **Step 4 - Pipeline.** To ensure easy and smooth performance of *vectorizer*, *transformer* and *classifier*, we used a compound classifier, Pipeline class, that integrates the functionalities of *vectorizer*, *transformer* and *classifier*.

3.2.3 *Classification Evaluation*.

We measured precision, recall and accuracy to evaluate our automated technique to classify code

Table 2: Accuracy, Precision and Recall of Classifiers

Classifier	Accuracy	Comments	Precision	Recall
SVM	80.67	Relevant Comments	0.85	0.75
		Auxiliary Comments	0.77	0.86
Random Forest	85.2	Relevant Comments	0.81	0.95
		Auxiliary Comments	0.94	0.76
Gradient Boost	71.667	Relevant Comments	0.98	0.90
		Auxiliary Comments	0.68	0.65

Table 3: Correlation test results

Correlation	Issues	correlation factor	p-value
Spearman Correlation	All Issues	0.56	9.43e-36
	Closed Issues	0.56	4.01e-35
	Open issues	0.49	3.91e-26
T-test	All Issues	6.40	4.13e-10
	Closed Issues	7.19	3.02e-12
	Open issues	9.25	1.27e-18
Kendall-tau Correlation	All Issues	0.41	2.23e-35
	Closed Issues	0.41	1.22e-34
	Open issues	0.35	1.33e-25

comments:

$$Precision = \frac{|TP|}{(|TP|+|FP|)}$$

$$Recall = \frac{|TP|}{|TP|+|FN|}$$

$$Accuracy = \frac{|TP|+|TN|}{|TP|+|TN|+|FP|+|FN|}$$

TP, TN, FP, FN are based on the following definitions:

(1) **TRUE POSITIVES (T P):** measures the proportion of actual positives that are correctly identified.
(2) **TRUE NEGATIVES (T N):** measures the proportion of actual negatives that are correctly identified.
(3) **FALSE POSITIVES (F P):** measures the proportion of predicted positives that are wrongly identified.
(4) **FALSE NEGATIVES (F N):** measures the proportion of predicted negatives that are wrongly identified.

We also computed the f1-score of each category of comments into consideration for every classifier used.

4 RESULTS AND ANALYSIS

In this section, we discuss the results of our research questions aimed at understanding the impact of code comments on issues and analyze the correlation between these comments and issues.

Our semi-automatic categorization of comments led to the development of a model that helped us in classifying comments into the mentioned categories.

We applied the machine learning model built on 20 python repositories of the extracted dataset, to help us answer the research questions listed below.

RQ1: What is the correlation between the total number of relevant comments and issues across different repositories?

We set out to find the correlation between total number of *relevant comments* and number of issues. To answer this question, we applied T-Test correlation, Spearman correlation and Kendalltau correlation tests between the number of relevant comments and number of issues. We have calculated the correlation factors for all the categories of issues i.e for open issues, closed issues and total issues (open issues + closed issues) separately. The following results show to what extent two datasets are related to each other using the correlation factor, statistic value and tau value for different methods used respectively.

The Spearman correlation is a non-parametric measure of the monotonicity of the relationship between two datasets. Spearman correlation does not assume that both the datasets should be normally distributed. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases. Dataset considered for our study is the number of relevant comments in different repository as an array in x and the corresponding number of issues in the array y.

If the correlation value is close to 1.0 or -1.0, then we can say that relation between two dataset is monotonic but as we know that number of issues is dependent on the number of factors among which quality of source code is the most important factor, along with source code comments. It is known that if Spearman correlation factor is between 0.75 to 1.0, then it is a strong correlation, if it is between 0.75 to 0.5, then it is moderate but significant, otherwise, its a weak correlation[9]. From our results we got Spearman correlation factor as 0.56 for closed issues which is significant, considerable and cannot be ignored. Hence, we can state that more number of relevant comments may imply a higher rate of solving issues in that repository.

T-Test is the two-sided test for the null hypothesis that 2 related or repeated samples have identical average (expected) values. A statistic value is the relative error difference in contrast to the null hypothesis. p-value, is the statistical significance of measurement in how correct a statistical evidence part is. If our regression is based on what statisticians call a “large” sample i.e a sample containing 30 or more than 30 observations, a t-statistic greater than 2 (or less than -2) indicates the coefficient is significant with greater than 95% confidence[2]. In our results, we got the statistic value for all types of issues greater than 2.0 with p-value less than 0.05. It is assumed that if p-value is less than 0.05 , then we can reject the null hypothesis, which states that there is no relation between two datasets. Hence we have a strong evidence against the null hypothesis and can state that there is a correlation between total number of relevant comments and number of issues.

Kendalltau (Kendall rank correlation coefficient) [12] is an another statistical approach used to define a correlation between total number of relevant comments and the total number of issues present in our dataset. Kendalltau doesn’t assume the data to be normally distributed. Moreover, it doesn’t assume the existence of a straight linear relationship between the analyzed pairs of metrics.

[2] https://onlinecourses.science.psu.edu/statprogram/reviews/statistical-concepts/hypothesis-testing/p-value-approach

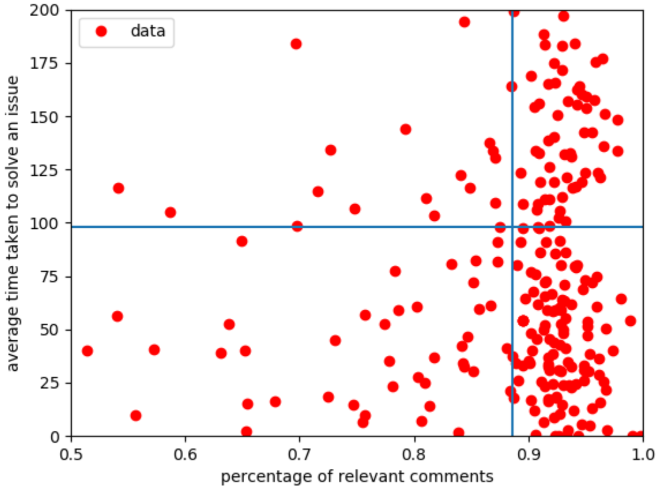


Figure 4: Graph of Percentage of Relevant Comments in each repository vs Average time to solve an issue for Random Forest Classifier

A certain set of guidelines for the understanding of the correlation coefficient is provided by Cohen [3]. We assume that if tau value is between 0 and 0.1, then there is absolutely no correlation at all. If its value is between 0.1 and 0.3, then there may exist a small correlation between the two datasets, or if the value is between 0.3 to 0.7, then a significant amount of correlation exists. Tau value greater than 0.7 implies a strong correlation.

Our results show that the tau value lies between 0.3 to 0.7 i.e 0.41,0.41 and 0,35 for total issues, closed issues and open issues respectively, hence we can say that a medium correlation does exist between both the datasets, especially in the case of closed issues. Table 3. shows the results of the correlation.

RQ2: How does average time to resolve an issue in a repository varies with percentage of relevant comments?

With the help of the model and total number of issues in each repository present in our dataset, we would present a correlation between the percentage of relevant comments and the average time taken to solve a closed issues across repositories. Percentage of relevant comments in the repository is calculated by calculating an average of percentage of relevant comments in each file present in the repository.

We applied Random Forest Classifier on the dataset and found the results which are shown in Fig4.

We divided the figure into four parts to check the density of repositories in each category which are defined as follows:

- Low percentage of Relevant Comments - Average time taken to solve an issue is less.
- Low percentage of Relevant Comments - Average time taken to solve an issue is more.
- High percentage of Relevant Comments - Average time taken to solve an issue is less.
- High percentage of Relevant Comments - Average time taken to solve an issue is more.

We have divided our graph into four categories or quadrants by drawing two perpendicular lines from x = 0.88, where x denotes the percentage of relevant comments and from y = 97, where y denotes average time taken to solve an issue in days. We chose these values by calculating **mean** of the percentage of relevant comments and the average time taken to solve issue respectively, across all repositories which we have covered in our study.

Our results indicate that there are a few set of repositories that fall under the category 1. Hence, we cannot provide strong evidence which can state that low percentage of relevant comments in a repository implies more time to solve an issue.

Major chunk of repository lies in category 4, which provides a strong evidence to our other hypothesis that more the percentage of relevant comments in the repository, less is the solving time of an issue.

The number of repositories that lie in the category 2 and 3 implies false positives and false negatives which may be arising due to fact that the machine learning model is not fully accurate and there are several other factors like the lines of code which contributes to the solving of an issue. If the file contains fewer lines of code, then even few relevant comments can result in a high percentage of Relevant Comments, which can affect our experiments. We restricted our percentage of relevant comments from 0.5 to 1.0 for results as 99% of repositories were lying in this range.

Our results show that if the percentage of relevant comments in a repository is more, then the probability that the average time taken to solve an issue is less than 100 days will be around 65%. Thus, we can provide an evidence that high percentage of relevant comments do make an impact on solving of issues, but there is a need for more future studies to prove the vice-versa, i.e. to prove that if there is a less percentage of relevant comments, then average time taken to solve an issue will be more.

5 THREATS TO VALIDITY

Internal Validity: One valid criticism of our work can be that our classification sample is made on a selected sample of repositories and is based on the classification metrics such as precision, recall, and accuracy. Thus, our classifier could not have delivered a very fine-grained classification of comments. Secondly, from each repository, we have taken at most 300 closed issues into consideration to calculate the average time taken to solve an issue. As stated in the work of Fluri et al. [5] code comments do change as the software evolves i.e. there is an impact of code modifications on code comments. Hence, we wanted to take the most recent number of issues from each repository. To maintain consistency and validation in our approach, we chose the first 300 closed issues from each repository to demonstrate our study.

External Validity: We have investigated our study for a sample of 625 python repositories, and hence, it may not generalize well on the data which is not included in our dataset. The proposed approach, however, shows distortions from our hypothesis. Our study is focused on only one programming language i.e. Python. Solving an issue depends on several factors. Of all, we chose to show the effect of code comments on solving an issue. So calculated average time to solve an issue may not go well as expected in case of all the repositories as there may be other factors playing a vital role there.

Construct Validity: Our empirical study suffers from construct validity as we use machine learning models and training data of code comments and issues manually.

6 CONCLUSION & FUTURE WORK

Code comments are an important artifact containing valuable information that helps in software development and maintenance. As all the comments are not the same, classification of comments into different categories for getting precise and accurate results have to be taken into account. Solving an issue is dependent on several factors among which code comments play an important role. We can conclude from our study that there is a correlation between code comments and issues as shown using two approaches.

Our first approach follows the statistical method in which we showed a correlation factor between the number of code comments and the number of issues. We found that the correlation between the number of closed issues and the total number of relevant comments is stronger compared to open issues and total number of issues in the repository. Hence, from this work, we can provide evidence that there exists a correlation between the total number of relevant comments and the number of closed issues in that repository.

Our second approach involves the use of classifier in which we demonstrated the correlation between code comments and average time of solving an issue and found that around 65% of results favour our hypothesis that more the proportion of relevant comments in a repository, lesser is the average time taken to solve an issue, but we could not find further evidence to conclude that whether issue-resolving time is high when relevant comments are low.

Further research in this area can be focused and organized along the line of our study, but including more programming languages from different paradigms. Also, including more factors and finding out the impact of each of them in solving an issue is also a possible area of research. Finally, we see that investigating value of code comments for developers during maintenance is largely unexplored in the literature and could help in improving program comprehension and software documentation.

REFERENCES

[1] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. 2013. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 188–197.

[2] Arianna Blasi, Alberto Goffi, Konstantin Kuznetsov, Alessandra Gorla, Michael D Ernst, Mauro Pezzè, and Sergio Delgado Castellanos. 2018. Translating code comments to procedure specifications. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 242–253.

[3] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.

[4] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2005. A Study of the Documentation Essential to Software Maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information (SIGDOC '05)*. ACM, New York, NY, USA, 68–75. https://doi.org/10.1145/1085313.1085331

[5] Beat Fluri, Michael Wursch, and Harald C Gall. 2007. Do code and comments co-evolve? on the relation between source code and comment changes. In *14th Working Conference on Reverse Engineering (WCRE 2007)*. IEEE, 70–79.

[6] Denae Ford and Chris Parnin. 2015. Exploring Causes of Frustration for Software Developers. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '15)*. IEEE Press, Piscataway, NJ, USA, 115–116. http://dl.acm.org/citation.cfm?id=2819321.2819346

[7] Akash Ghosh and Sandeep Kaur Kuttal. 2018. Semantic Clone Detection: Can Source Code Comments Help?. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 315–317.

[8] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 2019. 9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay. *arXiv preprint arXiv:1901.07440* (2019).

[9] Jan Hauke and Tomasz Kossowski. 2011. Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae* 30, 2 (2011), 87–93.

[10] Jane Huffman Hayes and Liming Zhao. 2005. Maintainability prediction: a regression analysis of measures of evolving systems. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 601–604.

[11] Walid M Ibrahim, Nicolas Bettenburg, Bram Adams, and Ahmed E Hassan. 2012. On the relationship between comment update practices and software bugs. *Journal of Systems and Software* 85, 10 (2012), 2293–2304.

[12] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[13] Ninus Khamis, René Witte, and Juergen Rilling. 2010. Automatic quality assessment of source code comments: the JavadocMiner. In *International Conference on Application of Natural Language to Information Systems*. Springer, 68–79.

[14] Ninus Khamis, René Witte, and Juergen Rilling. 2010. Automatic Quality Assessment of Source Code Comments: The JavadocMiner. In *Natural Language Processing and Information Systems*, Christina J. Hopfe, Yacine Rezgui, Elisabeth Métais, Alun Preece, and Haijiang Li (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 68–79.

[15] Wanwangying Ma, Lin Chen, Xiangyu Zhang, Yuming Zhou, and Baowen Xu. 2017. How do developers fix cross-project correlated bugs? a case study on the GitHub scientific Python ecosystem. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 381–392.

[16] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I know what you did last summer: an investigation of how developers spend their time. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. IEEE Press, 25–35.

[17] Alberto S Nuñez-Varela, Hector G Perez-Gonzalez, Francisco E Martínez-Perez, and Carlos Soubervielle-Montalvo. 2017. Source code metrics: A systematic mapping study. *Journal of Systems and Software* 128 (2017), 164–197.

[18] Luca Pascarella and Alberto Bacchelli. 2017. Classifying code comments in Java open-source software systems. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 227–237.

[19] Simone Scalabrino, Mario Linares-Vasquez, Denys Poshyvanyk, and Rocco Oliveto. 2016. Improving code readability models with textual features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 1–10.

[20] Zéphyrin Soh, Foutse Khomh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2013. Towards understanding how developers spend their effort during maintenance activities. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 152–161.

[21] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. 2002. Code quality analysis in open source software development. *Information Systems Journal* 12, 1 (2002), 43–60.

[22] Daniela Steidl, Benjamin Hummel, and Elmar Juergens. 2013. Quality analysis of source code comments. In *2013 21st International Conference on Program Comprehension (ICPC)*. Ieee, 83–92.

[23] Ted Tenny. 1988. Program readability: Procedures versus comments. *IEEE Transactions on Software Engineering* 14, 9 (1988), 1271–1279.

[24] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A large-scale empirical study on code-comment inconsistencies. In *Proceedings of the 27th International Conference on Program Comprehension*. IEEE Press, 53–64.

[25] Scott N Woodfield, Hubert E Dunsmore, and Vincent Yun Shen. 1981. The effect of modularization and comments on program comprehension. In *Proceedings of the 5th international conference on Software engineering*. IEEE Press, 215–223.

[26] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 15–25.