An Exploration Study of Underrated Stackoverflow Answers: With Program Analysis Metrics

Hanhan Wu
Computing Science
Simon Fraser University
Burnaby, Canada
hanhanw@sfu.ca

1. Introduction

Developers are facing endless problem-solving challenges, such as learning a new programming language or fixing a bug, and they often look for solutions from many online resources. At the same time, Psychology indicates that worked examples help solve problems more efficiently than solving problems from scratch [1]. Therefore, solutions with code examples become a major resources for developers to learn. StackOverflow (SO) is one of the software development communities that allow people to share their developed code and discuss programming solutions.

Besides knowledge sharing, large amount of developers are interested in gaining reputations in SO. By answering questions and getting more votes, they could earn higher score. [2] is showing that, there are strategies to gain higher votes in a question without providing the best solution, such as answering the question between 4:00 to 8:00 GMT or answering more questions in a faster manner. This phenomenon is in fact providing barriers for other developers to learn [3], without seeing the right top voted answer in a question, it takes developers longer time to solve their problems.

Same as all the other collaboration networks, the major responsibilities for SO are to keep attracting new users and encouraging existing users to participate and contribute by providing incentives [4]. This incentive can be to make it easier to find the best solution for each answer, and to detect the better solution and adjust the answers' ranking automatically, that is to say, underrated answers should be reduced.

2. Problem Motivation & Background

2.1. Research Questions

The purpose of this study is to explore the underrated answers in SO, and to provide insights on the possibility of detecting the best solution of an SO question automatically, in order to make some contributions to ease developers experience in SO. The major research questions are:

1. Is there any underrated answer in SO? If so, is it easy to find them?
2. What could be the reason that makes an answer to be underrated?
3. Is that possible to detect underrated answers automatically with program analysis metrics?

2.2. Knowledge Background & Definitions

1. SO basic data structure:

- Each url is a Post
- A Post contains 1 Question, all the Answers and Comments for the Question or for each Answer
- A Question can be up-voted or down-voted, it also contains Favorite Count to record how many users like the question
- An Answer can be up-voted or down-voted
- A Comment can only be up-voted

2. Underrated Answer - The best solution at the time when I collected data, and it has lower votes than the top voted answer.

3. Tagging - SO uses tags on each question to make them more searchable.

4. Sensitivity - True positive rate.

5. Specificity - True negative rate.

6. Balanced Accuracy - A more robust accuracy measure that considers both sensitivity and specificity.

In order to answer the research questions, I have randomly extracted 107,588 post urls. Among them, 2223 posts with at least 3 answers have been selected, so that it is more likely to find underrated answers. After converting all the selected posts into JSON file, because of

the time and human resource limitation, I have only used 123 posts for this study. The experimenting methods are machine learning models in Clustering and Classification, and the features for data analysis contain 46 code & bug metrics, 12 sentiment metrics, as well as 3 other metrics.

Section 3 will talk about related work. Section 4 will provide detailed approach process from data collection to data analysis experiments and Section 5 will give the experiments evaluation as well as observations. At the end, Section 6 will briefly talk about conclusions and future work.

3. Related Work

To contextualize the insights about solutions that are trying to ease developers experience in SO, I draw up research in software engineering from different aspects.

1.1. Removing Programming Barriers

There are studies provide solutions to remove SO barriers for developers. [5] talks about current topic modeling problem in SO search, and provides suggestions to create more accurate topic modeling. [6] recommends to make SO tagging recommendation more accurate using text based machine learning technique. [7] has done big data analysis on SO text keywords in order to predict user activity trends and therefore provide recommendations to SO users. However, all these methods are focusing on text analysis without considering the code examples.

1.2. Manually Data Analysis

There are studies work on manually data analysis in order to improve SO user experience. [8] has explored what makes SO good code examples through manually inspecting into the code of 357 SO posts, and they summarized that concise code that is closer to question context with step-by-step solution is more likely to be good code examples. [9] did a survey among developers and manually analyzed the collected data to understand developers' learning barriers in SO, and they found that developers prefer well documented information. [10] has summarized code characteristics in SO by inviting 16 participants to do the summarization work. All these studies have provided meaningful insights to help improve SO user experience, and they are considering the SO code examples, but they are all doing manually analysis.

1.3. Code Quality Focused

Other studies tried to assume top voted answers as the best solutions and therefore majorly focus on analyzing the code quality of those highly voted SO answers [11, 12]. However they did not look into the existence of underrated answers.

To sum up, previous SO related software engineering studies majorly focus on text analysis, manually data analysis or code quality analysis with the assumption that top voted SO answers are the best answers. This study is trying to bridge some gap by studying the existence of SO underrated answers, reasons behind the underrating and the possibility of detecting underrated answers automatically, with program analysis metrics.

4. Approach

The whole approach contains 3 stages: Data Collection, Feature Generation, Data Analysis Experiments.

4.1. Data Collection

In data collection stage, I randomly extracted 107,588 SO post urls from StackExchange API, the time range is between 2013/01/10 and 2014/04/10. All the posts are using "Python" as the tag.

From all these posts, I have generated 2223 JSON files. Because data such as code, comments are invisible from StackExchange API, I had to collect the data from SO HTML of each post directly. Each JSON file represents an SO post that contains at least 3 answers. The basic format of each JSON file is as following:

- Each JSON file contains 1 question, the question data include question ID, question text, question code, question votes, favorite count
- All the answers of the question are also stored in this JSON file. For each answer, the data includes answer ID, answer text, answer code, answer total votes
- For each answer, the comments are stored in this JSON file too. For each comment, the data includes comment ID, comment text, comment code, comment votes

The data collection stage also includes data labeling. I had to label each post as "Y" or "N" to indicate the post has an underrated answer or not. When doing the data labeling for each post, I had to click the url, read the SO question and all the answers as well as comments, to judge whether there is any underrated answer. If there is, in order to do later data matching, I needed to record the first 5 words of the best underrated answer, this is because from SO website, there is no answer ID, other elements such as author name, time could be the same for different answers, meanwhile extracting the time from the website HTML adds more trouble, since there could be time zone issue. Therefore, the data labeling is very time consuming in this study.

Because of the time limitation, I have only labeled 123 posts among all the 2223 collected posts. There were 23 posts contained underrated answers at the time when I was doing data collection. These 123 labeled data are used for feature generation and data analysis in this study.

4.2. Feature Generation

In this stage, I have collected 46 code & bugs metrics, 12 sentiment metrics and 3 other metrics.

A. CODE & BUG METRICS

In order to collect code & bug metrics, I have tested 3 Python open source libraries: Pylint, Randon and Pyntch, and finally decided to use Pylint and Radon because their output is parsable. The collected metrics include 5 levels:

- Level 1 - Coding Style

The code analyzed in this study is all Python code, and Python is sensitive to code syntax. The metrics related to coding style contain 4 parts:

1. Module Metrics - Metrics about Python libraries. E.g. Module_Documented, Module_Badname, etc.
2. Class Metrics - Metrics about classes in the code. E.g. Class_Documented, Class_Badname, etc.
3. Function Metrics - Metrics about functions in the code. E.g. Function_Documented, Function_Badname, etc.
4. Line of Code Metrics - Metrics about lines of code. E.g. code_percentage, comment_percentage, etc.

- Level 2 - Raw Metrics

1. LOC - Total number of lines of code
2. LLOC - Total number of logical lines of code, each logical line contains exactly 1 statement
3. SLOC - Number of source of lines of code
4. Comments - Number of comments
5. multi - Number of lines which represent multi-line strings
6. Blanks - Number of blank lines

Meanwhile, SLOC - Comments - multi = LOC should always hold.

- Level 3 - Cyclomatic Complexity

Cyclomatic Complexity corresponds to the number of decisions a block of code contains plus 1. This number (also called McCabe number) is equal to the number of linearly independent paths through the code. This number can be used as a guide when testing conditional logic in blocks [13]. In this study, I used Cyclomatic Complexity to calculate the complexity of each function and all the functions.

- Level 4 - Halstead Metrics

By compiling the code and generating AST tree, Halstead Metrics identify measurable properties of software and the relationship between them. The generated metrics include: Program_Vocabulary, Program_Length, Volume, Difficulty, Effort, Time_Required_to_Program and Bugs. [14] contains more detailed metrics explanation.

- Level 5 - Maintainability Index

Maintainability Index measures how maintainable (easy to support and change) the source code, it is also one of the metrics used in Microsoft Visual Studio 2010.

B. SENTIMENT METRICS

When I was doing data labeling, I have realized that automatically generated code metrics may not be enough, some underrated answers and top voted answers could share very similar code metrics. However, comments under each answers could give clues about whether a solution really works. Therefore, sentiment analysis on all the comments of each answer became my first choice. I am using Stanford CodeNLP for sentiment analysis. There are 5 levels sentiment in Stanford CoreNLP: Very Positive, Positive, Neutral, Negative and Very Negative.

At the same time, I measured the sentiment on each sentence of a comment or an answer, and finally calculated the total counts of each sentiment level, as well as the overall sentiment score. For example, a comment with 3 sentences could generate the sentiment metrics as below:

{“score”:4, “Very Negative”: 2, “Negative”:1, “Neutral”: 0, “Positive”:0, “Very Positive”:0}

The example indicates that this comment contains 2 very negative sentences and 1 negative sentences, the overall sentiment score is 4.

Sentiment metrics are calculated for each answer and it sums up sentiment counts for all the comments of the answer. So, for each answer, there are 12 sentiment metrics.

C. OTHER METRICS

- Code Sequence Matching

According to [8], a good SO code example should be closer to the question context, however when I was doing data labeling, there were cases that an answer’s

code serves the question very well and it is similar to the question's code, however, it is not a good solution that serves for larger user population. Therefor, I calculated the Sequence_Match_Score to tell how close an answer's code and the question's code is, in order to study whether this feature would be important to predict underrated answers. The sequence match algorithm I am using [15] will ignore the junk items in code sequence, in order to maximize the matching length, and therefore it will provide more accurate code matching output.

- Compare With The Top Votes

I was wondering, if an answer's vote count is much lower than the highest answer's vote in a question, whether it is less likely to be an underrated answer. Meanwhile, if there are large amount of answers for a question, and most answers got up-votes, whether the percentage of vote count could help decide an answer is underrated or not. Therefore, I calculated the 2 metrics based on answer vote count.

1. vote_percentage = the answer's vote/all the answers' votes in a question
2. vote_difference = highest answer's vote in this question - the answer's vote

To sum up, all the 61 metrics are generated for each answer. After feature generation, I separated the hierarchical data structure of each JSON file into question-answer pair and saved all the data from 123 posts into one single CSV file. Each row of this CSV file has the following format:

QuestionID, AnswerID, …61 features…, IsUnderrated

Here, IsUnderrated is the data label, its value only has "Y" or "N". In order to match these labels to the right answers, I used the first 5 words from the underrated answers, and they all matched correctly.

Finally, there are 442 distinct rows of data for the data analysis in this study. 23 rows have "Y" as the label and others have "N" label.

4.3. Data Analysis Experiments

In the whole data analysis process, machine learning methods can be divided in to Clustering and Classification. With Clustering, I was planning to see whether data with label "Y" could be clustered together, and if so I should check the patterns in that cluster. With Classification, I was planning to find whether it is possible to predict underrated answers automatically, if so what can be influencing features.

Before Clustering and Classification, I used the same data preprocessing methods in order to make both Clustering and Classification models work better.

The major challenge of the data analysis in this study is the small imbalanced dataset, it has 442 data records and only 23 records have label "Y".

A. DATA PREPROCESSING

The data preprocessing before using Clustering or Classification has the following 5 steps:

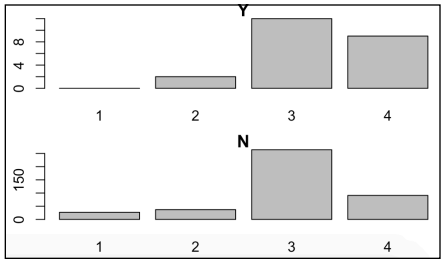
1. Remove Zero Variance Features - features with 0 variance are the noise of the data and can create negative influence on some machine learning models. In this step, 15 features have been removed.
2. Deal With Outliers - After checking the boxplot of the rest numerical data, 4 features have a few outliers, considering the meaning of these features, such as Module_Badname, the appearance of outliers is useful. Meanwhile, the dataset is small, I decided neither to remove outliers nor bucketing the values, but just leave the outliers there.
3. Remove Highly Correlated Features - Correlated features will add duplicated weights in multiple machine learning models, such as K-Means Clustering. So, in this step I removed those features with correlation score higher than 0.7, 17 features have been removed.
4. Data Scaling - The value ranges of the rest features have significant difference, machine learning models such as K-Means will be negatively influenced by the difference. So, in this step, I scaled all the data features into [0, 1] range. In fact, I tested to normalize all the data using KNN, which uses another data scaling method, but its model evaluation was not satisfying. So, in this paper, I will focus on talking about features in [0,1] scale.

B. CLUSTERING

In this study, the Clustering methods are K-Means and Cluster Ensembling, and labels have been removed when running the models.

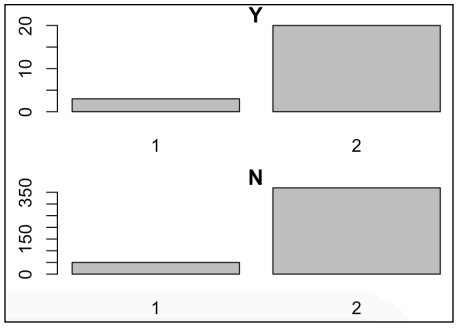
With elbow method, I have found 4 is the optimal cluster number for the dataset. After running K-Means with k=4 several times till the cluster membership became stable, I have checked the cluster distribution for both "Y" and "N" classes. However, as figure 1 shows, the cluster distribution are the same for both "Y" and "N" groups, which means K-Means cannot tell the difference between "Y" class and "N" class.

figure 1



Then, I experimented with Cluster Ensembling. It ensembles multiple clustering models in order to generate more robust results. I used Ward.D, Single Linkage, Complete Linkage, Average Linkage and Mcquitty to form this cluster ensemble, they are all hierarchical clustering methods. The output visualization indicated 2 is the optimal cluster number. With the generated cluster output, I have also checked the cluster distribution of both "Y" and "N" classes, but still they share the same distribution pattern as figure 2 shows.

figure 2



C. CLASSIFICATION

In classification experiments, there are majorly 3 rounds, which use preprocessed dataset, preprocessed dataset with ROSE data over-sampling, and preprocessed dataset with SMOTE data over-sampling. In each round, the machine learning models are SVM, Random Forests, GBM, XGBoost because these models are more robust when dealing with imbalanced dataset, even when the dataset is small. All the machine learning models have used the same cross-validation settings to avoid overfitting and to find the optimal parameters.

• Round 1 - Preprocessed Dataset

In the first round, I only used preprocessed dataset, which has 432 data records and 23 "Y" labels among them. 67% became the training data with 16 "Y", the rest 23% became the testing data with 7 "Y" labels. However, all the machine learning models made predictions all as "N". Therefore, the prediction results in this round became meaningless.

• Round 2 - ROSE + Preprocessed Dataset

ROSE is a random data over-sampling method, it randomly selects subset from the minority class and makes copies, in order to achieve the data balance. In this study, ROSE created a new dataset which contains 227 "N" and 215 "Y", this new dataset is called "ROSE dataset".

One thing need to mention is, this method used ROSE on all the 432 preprocessed dataset first, and later separated it into 67% training data and 23% testing data. Before using ROSE on the whole dataset, I only applied ROSE on the training data from the first round but kept the first round testing data, the output had balanced accuracy lower than 0.3. So, in this paper, Round 2 will only focus on talking about the experiments on ROSE dataset, which is generated after applying ROSE on the whole preprocessed data.

After running SVM, Random Forests, GBM, XGBoost on Rose dataset, all with the same cross-validation settings, SVM, Random Forests and XGBoost worked with ROSE better. Here are their evaluation output:

SVM got 0.86 Sensitivity, 0.88 Specificity, 0.87 Balanced Accuracy.

Random Forests and XGBoost all got 1.0 as their Sensitivity, Specificity and Balanced Accuracy. The reason they got this insane prediction output is because the dataset is small in this study. However, with proper training data and testing data separation, as well as cross-validation, the prediction here is still reliable.

• Round 3 - SMOTE + Preprocessed Dataset

SMOTE is another over-sampling method to deal with data imbalance. Instead of making duplications for the minority class, it randomly selects a subset minority class data and generates similar dataset, which may not appear in the original dataset. The advantage of SMOTE is, it is reducing the overfitting created by data duplicates, But using SMOTE can also be risky, since the generated minority class data could overlap with the majority class data, which will create more data noise.

In this study, in order to use SMOTE, I had to remove 10 features that are near 0 variance (less than 5% distinct values). The dataset is called "SMOTE dataset". The original preprocessed data been separated as 67% training data and 23% testing data, and SMOTE has been applied only on the training data. This is because SMOTE does not create duplications for over-sampling, applying it on testing data may not generate the ground truth.

After running SMOTE dataset with Random Forest, SVM, GBM and XGBoost, GBM got the best prediction output with 0.33 Sensitivity, 0.96 Specificity and 0.65 Balanced Accuracy. However, this result still indicates the data imbalance problem.

5. Evaluation & Observations

To sum up the evaluation output, Clustering methods in this study failed since they could not figure out the difference between "Y" class data and "N" class data, and therefore it is no longer necessary to look into the group patterns in each cluster. After 3 rounds classification experiments, using ROSE with preprocessed dataset, and ran experiments on Random Forests, XGBoost and SVM gave balanced and higher prediction results.

Started from Round 2 classification results, I have used 2 features selection methods and tested on SVM, in order to find whether there are influencing features for underrated answers.

A. FEATURE SELECTION 1 - RANDOM FOREST

Besides doing classification, Random Forests is also famous for feature selection. Meanwhile, in classification Round 2, the prediction results from Random Forests got 1.0 for Sensitivity, Specificity and Balanced Accuracy, which means it made the predictions all correct. So I extracted the top ranked features from the Random Forests in Round 2, it returned 7 features as very important features:

Module_Documented, Answer_VeryNegative, Function_Documented, multi, Class_Badname, DocumentString_Percentage and Class_Documented.

B. FEATURE SELECTION 2 - BORUTA

Boruta is an all-relevant feature selection method, it measures how relevant each feature to the prediction accuracy. It is a wrapper of Random Forests, but it is also an extension of Random Forests by comparing the relevance of features, therefore in theory, it should work better than Random Forests.

The important features returned by Boruta has more. However its top 7 features are the same as Random Forests selected features, but with different ranking. Here are the output important features from Boruta:

Module_Documented, multi, Answer_VeryNegative, Function_Documented, Class_Badname, Class_Documented, DocumentString_Percentage, vote_difference, Comment_VeryPOsitive, Comment_Positive, Comment_VeryNegative, Comment_Negative, Warning_Num, Function_Complexity, Function_Badname, vote_percentage.

C. FEATURE INSIGHTS

Based on ROSE dataset, I used both Random Forests selected features and Boruta selected features to run SVM, with the same cross-validation settings and there is 67% training data, 23% testing data.

SVM output for Boruta selected features are:

0.89 Sensitivity, 0.97 Specificity, 0.93 Balanced Accuracy

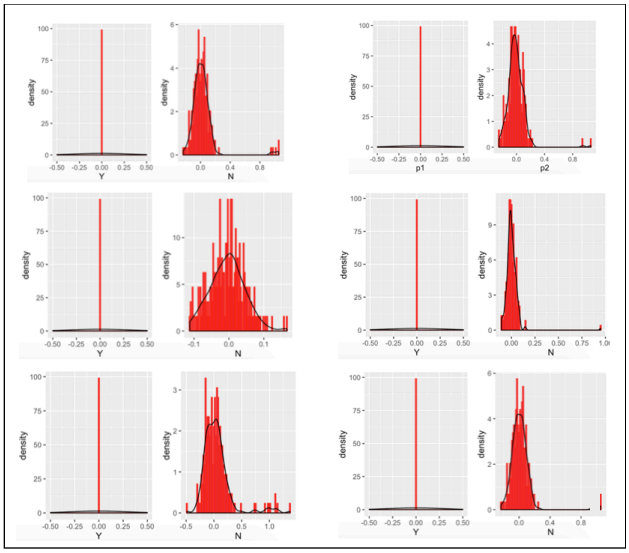
SVM output for Random Forests selected features are:

0.55 Sensitivity, 0.59 Specificity, 0. 64 Balanced Accuracy

The reason that Boruta selected features does better prediction can be because it contains more features and therefore more information for the prediction.

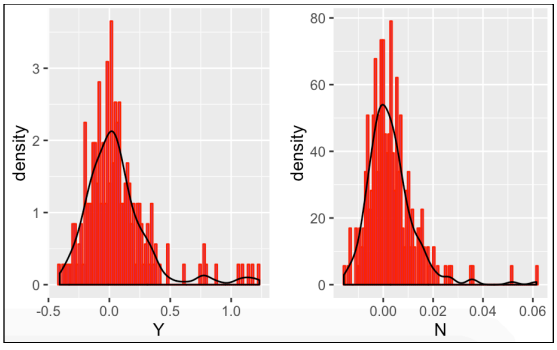
Since both Boruta and Random Forests returned top 7 features are the same. I looked into the distribution of each feature in both "Y" and "N" class and found an interesting pattern: for Module_Documented, multi, Function_Documented, Class_Badname, Class_Documented, DocumentString_Percentage, their values in "Y" class all appear to be 0, as figure 3 shows.

figure 3



By contract, Answer_VeryNegative has similar distribution in both "Y" and "N" class, as figure 4 shows.

figure 4



The observations are providing some insights about why an answer can be underrated:

- Even if an answer provides the best solution, without proper coding style, such as using documented module (library) or function or class, or the document string is 0, it can be underrated.
- Even if an answer has very negative sentiment or there is no bad class name, automatic detection can not determine whether the answer will get underrated.

5. Conclusion & Future Work

The purpose of this study is to explore the underrated answers in SO, and to provide insights on the possibility of detecting the best solution of an SO question automatically, in order to make some contributions to ease developers experience in SO.

After data collection and feature generations, I created 442 SO data records with 46 code & bug metrics, 12 sentiment metrics and 3 other metrics. The data has significant data imbalance problem by having only 23 underrated answers. After using Clustering and Classification methods, I have also gained insights to answer the research questions in this paper.

1. Is there any underrated answer in SO? If so, is it easy to find them?

Answer: Yes, there are. Among all the 123 random posts I have labeled, there are 23 posts with underrated answers.

2. What could be the reason that makes an answer to be underrated?

Answer: Based on the selected features from highly accurate predictions, it is showing that lack of proper coding style may lead to the underrate of an answer. These coding style can be, using documented library/ class/function, or having documentation string in the code example.

3. Is that possible to detect underrated answers automatically with program analysis metrics?

Answer: According to the experiments in this study, it is possible to detect underrated answers automatically. In this study, with ROSE to deal with data imbalance, SVM, Random Forests and XGBoost all returned balanced accuracy higher than 80%.

Besides, neither the similarity between an answer's code the question code nor the relationship between answer votes made a difference in automatic detection.

At the same time, there are limitations in this study and require more future work.

- The dataset is very small because of time and human resource limitation. In the future, a large amount of data needs to be collected and labeled by human experts.
- The SO code examples studied in this project are all Python code, it cannot represent other programming languages. In the future, multiple programming languages need to be investigated.
- Time series analysis should also be included in the future study, since current underrated answers are measured at the data collection time, without considering the changes along the time as well as the changing rate.

7. References

1. J. L. Plass, R. Moreno, and R. Brünken, Cognitive Load Theory, 1st ed. Cambridge University Press, 2010
2. Building reputation in StackOverflow: An empirical investigation, Amiangshu Bosu, Christopher S. Corley, Dustin Heaton, Debarshi Chatterji, Jeffrey C. Carver, Nicholas A. Kraft , 2013
3. D. Hou and L. Li, "Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions," in Proceedings of ICPC 2011, 2011, pp. 91–100.
4. Simon Walk, Denis Helic, Florian Geigl and Markus Strohmaier, Activity Dynamics in Collaboration Networks, 2015
5. What is Wrong with Topic Modeling? (and How to Fix it Using Search-based Software Engineering) Amritanshu Agrawal, Wei Fu, Tim Menzies, 2017
6. Tag Recommendations in StackOverflow, Logan Short, Christopher Wong, and David Zeng, 2014
7. A STUDY ON TRENDS IN INFORMATION TECHNOLOGIES USING BIG DATA ANALYTICS, Mahmut Ali ÖZKURAN, 2015
8. What makes a good code example?: A study of programming in StackOverflow Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, Chris Burns, 2012
9. M. P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers," IEEE Softw., vol. 26, pp. 27–34, Nov. 2009.
10. Selection and Presentation Practices for Code Example Summarization, Annie T. T. Ying and Martin P. Robillard, 2014
11. Subjective Evaluation of Software Quality Using Crowdsource Knowledge: An Exploratory Study,

Mohammad Masudur Rahman, Chanchal K. Roy, Iman Keivanloo, 2013

12. M. Bolton. What are the useful metrics for code quality?, URL http://www.linkedin.com/answers/technology/software-development/TCH_SFT/77088-3083818
13. Radon Documentation, http://radon.readthedocs.io/en/latest/intro.html
14. Radon Developer Manual, http://radon.readthedocs.io/en/latest/intro.html
15. Python SequenceMatcher, https://docs.python.org/2/library/difflib.html#difflib.SequenceMatcher