What Really Changes When Developers Intend to Improve Their Source Code: A Commit-Level Study of Static Metric Value and Static Analysis Warning Changes

KEVIN COSGROVE*, Rochester Institute of Technology, USA
RAVEL LAROSE*, Rochester Institute of Technology, USA
SAMANTHA MORASH*, Rochester Institute of Technology, USA
NIKHIL SURYAWANSHI*, Rochester Institute of Technology, USA

---

* All authors contributed equally to this work.

1 INTRODUCTION

Software quality is a metric that is difficult to measure but is extremely important to satisfactory software engineering and development [1]. Software quality is a complex metric to understand because there are many factors that can impact software quality [2]. Apart from the difficulty of dealing with many factors for a clear and concise assessment; software quality itself is subjective. Static software metrics are frequently used to determine the quality of software. Static software metrics or static code analysis is testing that is done on non-running source code to determine possible vulnerabilities [3]. Previously it was understood that static software metrics can be used to determine if a code is complex; and if the code is complex, it is more likely to contain errors or be error-prone. Subjectivity continues to be a problem though because complexity is understood differently depending on the developer; there is no concrete standard of complexity to determine software quality. Samples of corrective and perfective changes in code can be used along with their respective commit messages to determine developer intent and software quality changes. Corrective changes in code aim to fix problems or "bugs" in the code, while perfective changes in code aim to improve the functionality of the system [4]. It is unrealistic to manually analyze each commit message in a software program to determine developer intent. To manage this problem, deep learning models can fill the gaps. An example of a deep learning model to use in understanding software quality is natural language processing. Natural language processing uses linguistic training on computers to give the computer a grasp of the meaning of the text in a similar way to human understanding [5]. Understanding developer intention in changes to code can provide clarity on what types of changes are made with the purpose of improving quality.

There has been previous research published on the classification of code changes, the relationship between quality enhancements and software metrics, and research focused on commit messages. Mockus' study focused on changes to a large system and the identified reasons for those changes [6]. Using a keyword-based approach, Mockus found that textual descriptions of the change can be used to identify the type of change with validation from a developer survey. The study's major finding was that corrective and perfective changes are smaller and delete more lines of code than other change types. Pantiuchina et al. analyzed commit messages to determine the aim of the developer to improve a selection of static source code metrics associated with software quality [7]. Tian et al. produced a study that examined commit messages in five open-source projects and found 44% of messages could be improved on average [8]. They utilized a classification model that determined the quality of commit messages. This model was built on a dataset of 1600 commits that were manually classified by researchers. Their study yielded a taxonomy of commit messages with expression categories. While there is

quite a breadth of published work on code changes, code quality, and commit-level studies, there still remain shortcomings and unanswered questions whose answers can provide much value to academia and industry alike.

Many papers utilize a keyword-based approach to building their classification models based on developer commit messages. This method has a major limitation in that researchers and practitioners have somewhat of a disconnect when it comes to the terminology of change classification. Additionally, the previous works on the classification of code changes are quite particular and how they classify these changes [9, 10]. This limits the ability to make general claims about code classifications and their effect on code quality metrics. There are also no published works on what types of files in terms of size and complexity are targets of corrective and perfective changes. Our study looks to replicate part of a published paper from Trautsch et al., whose intent was to address some of the limitations that persist in the research topic of code changes and whether developers' intent matches their effect on quality [11]. Trautsch et al. address these limitations by compiling a large training dataset of manually classified commit messages into categories associated with improving internal and external features. The commit messages come from 54 long-standing, open-source Java projects that make up a total of 125,452 commits. They also built a state-of-the-art deep learning model to automatically classify a test dataset of commit intents. The datasets and model were used to produce two main studies: a confirmatory study that measured size and complexity metrics and static analysis warning changes for improving code quality, and an exploratory study aimed at measuring size and complexity metrics as well as static analysis warnings of files that are targeted for quality enhancements. While this paper does provide confirmation of previous findings and some new insights to the research community, there are still ways that it can be improved.

Our paper looks to reproduce some of the findings that Trautsch et al. found [11]. Specifically, our objective is to reproduce the training dataset on a smaller scale, fine-tune our own deep learning model based on our training dataset, and use that model to determine if developer intent to improve internal or external code quality has a positive impact on quality metrics. We will achieve this objective by manually classifying a smaller training dataset of 250 commit messages (~0.2% of total commits). We will also fine-tune a deep learning model like that of Trautsch's model so that we can test the validity of the methodology and classify a subset of the remaining commits (12,500, ~ 10% of total commits). Lastly, we will look to measure the same 14 static source code metrics across commits to validate or refute the conclusion of Trautsch et al. on developers' intent and measured quality improvements.

2 STUDY DESIGN

In the original experiment conducted by the researchers of the paper, ‘What really changes when developers intend to improve their source code: a commit-level study of static metric value and static analysis warning changes’, the first step in building the Natural Language Processing model was the manual classification of commit messages from 54 open source Java projects from Mauczka [12] into the categories of perfective, corrective, and other to determine what the developers' intent was for changes made to the code. In the original experiment, the researchers who manually classified the commit messages had extensive software backgrounds. To replicate the solution to the problem, we assessed how well we could classify the commit messages. Each group member classified 10 commit messages, we then compared our results to one another and to the results of the original researchers. Group members correctly classified nine of ten commit messages. Our next step was to manually classify 50 commit messages to assess our competence in classifying the messages on a larger scale. We ultimately decided that our lack of experience with classifying manual commit messages could produce confusing or misleading results. We decided to use the manual classifications that were completed in the original experiment for our replication of the study. The researchers who originally completed the manual classification had strong software engineering backgrounds and experience with commit classification, which our group lacked, making replication of this section of the paper unproductive.

To replicate the experiment while also improving upon it, it was determined that we would use the manual classifications that were already completed for the training dataset, but with a different split than was previously done. All of these manual commit messages come from 54 open-source Java projects. The change in split between training and test data in the original dataset would allow us to confirm or refute the results produced in the paper to see if the results are reproducible. If the results we get are similar to that of the original experiment, it can be said that the original results were not an outlier, the results are reproducible, and the conclusions drawn are accurate. Another way in which we can confirm the results in the Trautsch et al. paper [13] is by performing another type of modeling on the dataset. We will use a clustering model to confirm the conclusions about perfective, corrective, and other commit message classifications.

As shown in the figure below, a brief description of our approach is as follows: 54 open-source Java projects from Mauczka were gathered [12]. Two percent of each of the projects was used to gather a sample of 2,533 commit messages. The sample of commit messages was manually classified into categories of perfective, corrective, or other by multiple software engineers with previous classification experience. A model was developed from the

manual classifications, and a 10-fold cross-validation was performed. For the cross-validation, 90% of the data was used for fine-tuning, and 10% was used for evaluating performance. Of the 90% of the data that was used for fine-tuning, an 80/20 split of training and testing data was used. The 20% of testing data was used to evaluate the performance of the model.

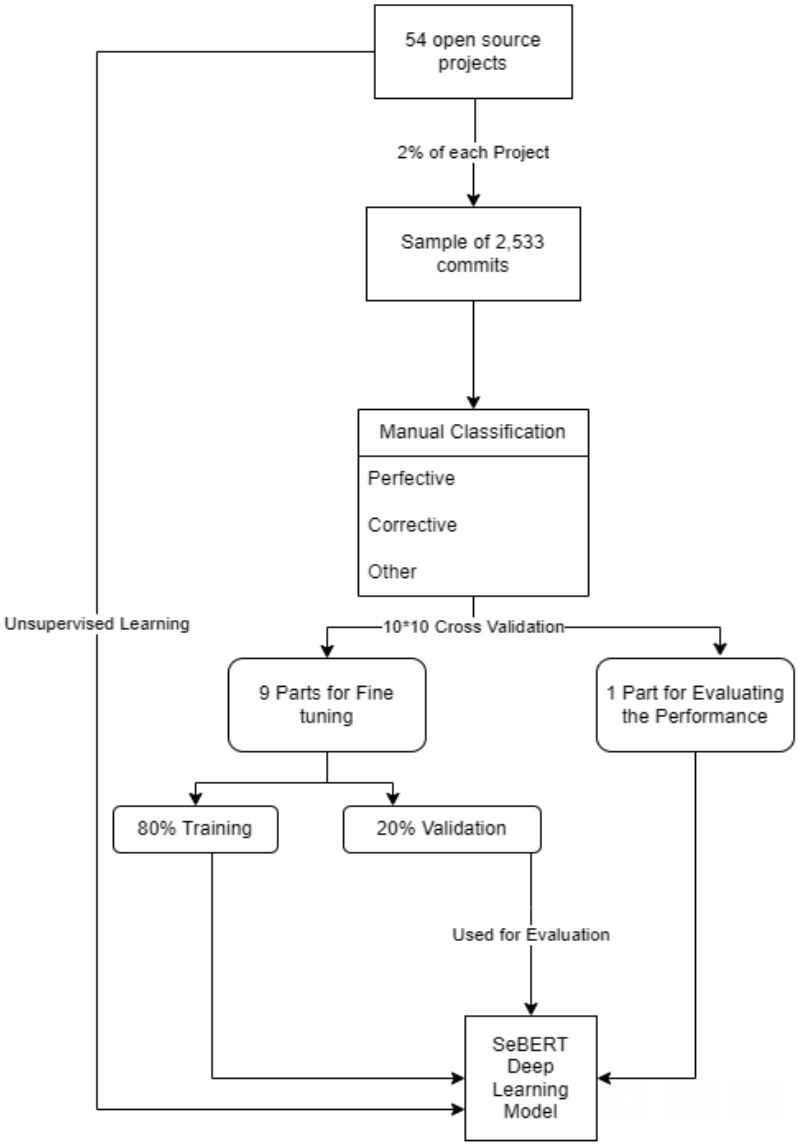

Fig. 1a: Black Box Approach Flowchart

The model is then fine-tuned and evaluated on the validation data for each epoch. In the end, the best epoch is chosen to classify the test data of the fold. This is repeated 10 times for every fold, which yields 100 performance measurements.

3 EXPERIMENTS

The experiment is based on the design of the study in the paper, and as such, will follow the research questions specified therein. This study focuses on the first two of the total of three research questions. The two questions are phrased as such. First, “Does developer intent to improve internal or external quality have a positive impact on software metric values?” Second, “What kind of files are the target of internal or external quality improvements?” The first research question appends the two following hypotheses: “Intended quality improvements are smaller than non-perfective, and non-corrective changes,” and “Intended quality improvements impact software quality metric values in a positive way.” In other words, the size of the commits that are categorized as perfective and corrective will be compared to the size of commits that are not classified as either of the two categories; and the software quality metrics will improve due to intended software quality enhancing commits.

The previous works of Mockus [14] and Purushothaman and Perry [15] found that corrective changes modify fewer lines than perfective changes, which tend to delete lines; and smaller changes overall for both perfective and corrective changes to code. Given the understanding of the previous works, the hypothesis is proposed that the observed pattern of perfective and corrective coding changes will persist in the open-source project the study observes.

In the paper, changes are categorized into one of the three classes: perfective, corrective, or other, based on the commit messages included with the changes. The size metrics used to classify the changes are described in Honel et al. [16] and Bakota et al. [17]. There are a total of 14 code metrics to track size, complexity, fallibility, reusability, and how well the code is documented. Relevant to the hypotheses at hand is the size metric Logical Lines of Code (LLOC), which measures the number of lines of code in the project that aren’t comments. This will allow the measurement of the number of lines added and removed per commit.

The paper itself provides some resources and replication kits to allow others to replicate their work and test it. The website provided for the paper, as well as the replication kit, were the main sources of information, code, and data for this experiment.

A GitHub repository, containing all of the code written and used by the original paper, was found in the paper. The GitHub of the project contained a folder dedicated to data, which included the batch of data manually labeled, as well as a few batches done via machine learning and by consensus. Other folders in the GitHub were dedicated to figures and images, but both were empty besides a .gitignore file. Most importantly, the folder named notebooks contained the code that created and collected the datasets, labeled the datasets, created the initial model, and then refined it into the ‘fine-tuned’ final model presented in the paper. This experiment uses the

model included in the FineTuneModel.ipynb file from the GitHub repository. Data for the project was originally located from MongoDB. For the experiment, the data was found in the all_changes_sebert.csv.gz file, which contains all of the data used in the paper as taken from MongoDB and labeled accordingly.

FineTuneModel.ipynb presents the final version of the fully built model from the paper. The model was built on the manually classified commit messages and then used to categorize the rest of the commits. These categorized commits became the dataset underlying the rest of the paper's commit and quality analysis; essentially, by running the commits through the model, the resultant dataset was output and categorized for proper testing. Only 14 of these total metrics were included with any level of detail in the body of the paper; the others are found in the dataset itself.

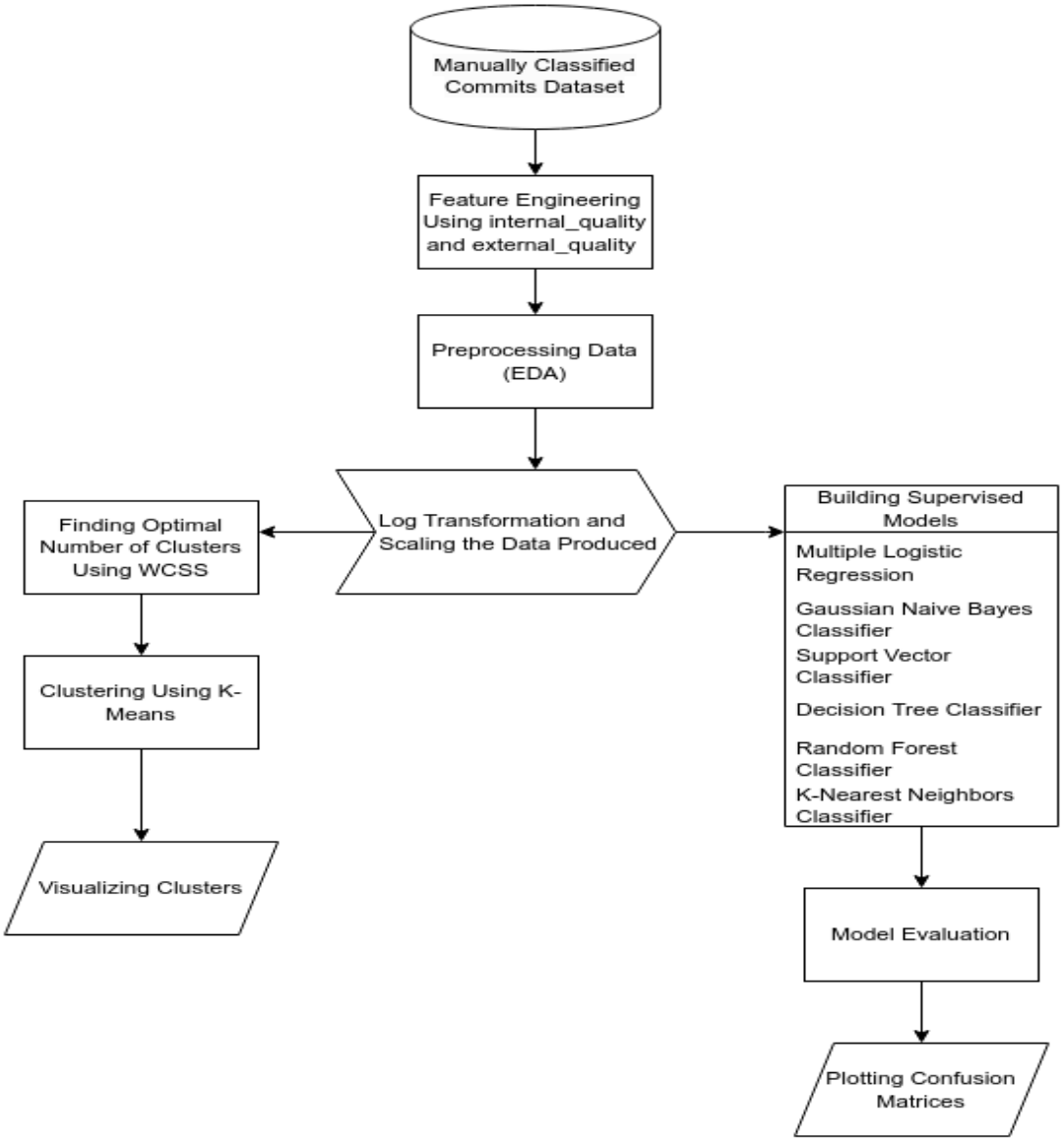

Fig. 1b: White Box Approach Flowchart

A few options for building on the work done in the paper, beyond what is done in the tables here, present themselves.

Clustering was performed on the datasets included in the paper. Specifically, both the manually classified data and the entire dataset were analyzed. By clustering the data based solely on the numerical evidence and not accounting for the commit messages left by developers, this would in theory bypass the need for a ground truth in the dataset while still allowing for analysis. It would also afford a chance to verify or question the accuracy of the classifications in the paper, both by hand and by model.

Preliminary clustering was performed on the manually classified data before the algorithms were introduced to the entire dataset, in order to see if the resultant cluster labels would be drastically different from those found in the paper. The tests run on the clustering algorithms can be divided into two overarching categories; one to investigate the claims of the first hypothesis of research question one, and the other to investigate the claims of the second hypothesis.

The first hypothesis of research question one, to reiterate, states that intended quality improvements are smaller than non-perfective and non-corrective changes. If this holds true, then the clusters are expected to be significantly different from each other when clustered by the size of their quality improvements. To this point, a k-means model was built to investigate the data in an unsupervised context.

The data, as collected from the paper itself, was not fit to be inputted into a k-means algorithm. The k-means clustering algorithm requires that the data it handles be normalized, and further, is not well equipped to handle data that is either highly skewed or uncentered around zero. Therefore, the data had to be transformed in order to be useable. All of the data had a heavy right skew due to many data points being equal to zero; it also did not fall along a normal distribution. Due to this, transforms were applied to the data, so that it fit the requirements to be usable in a k-means algorithm.

Provided that the paper classifies the data into three distinct categories, three distinct clusters are expected to be the ideal number for the data; this is explored in Figure 5.

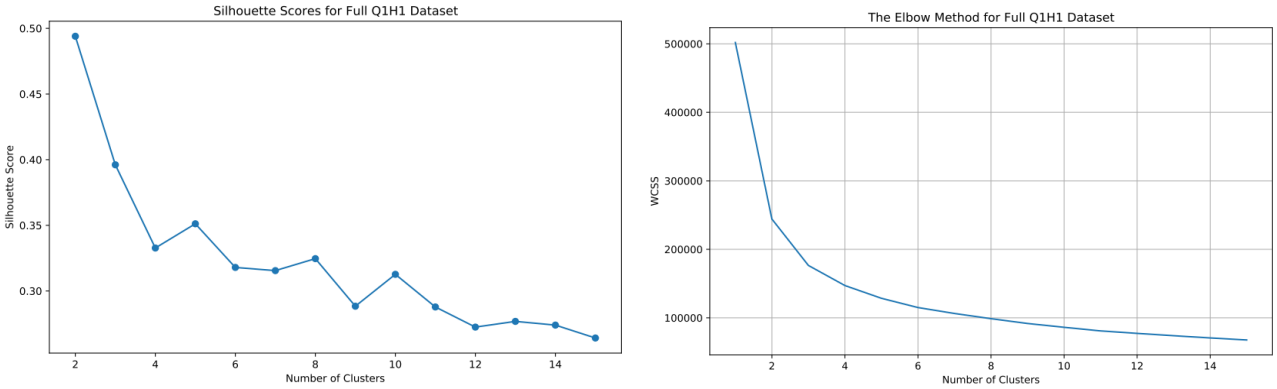

Fig. 2. Silhouette Scores (left) and the Elbow graph (right) for the data with respect to change size, in order to determine the appropriate amount of clusters without supervision.

The inflection point in the elbow graph presented in Figure 5 suggests that either two or three clusters would best fit the data based on its underlying numerical properties. This is further cemented by the calculated silhouette scores, which reach a local minima at three total clusters. Taken together, these two calculations back up the paper's decision to classify the data into three total categories.

Based on this confluence of evidence, the transformed dataset was passed through the clustering algorithm. The results of this pass are shown in Figure 6.

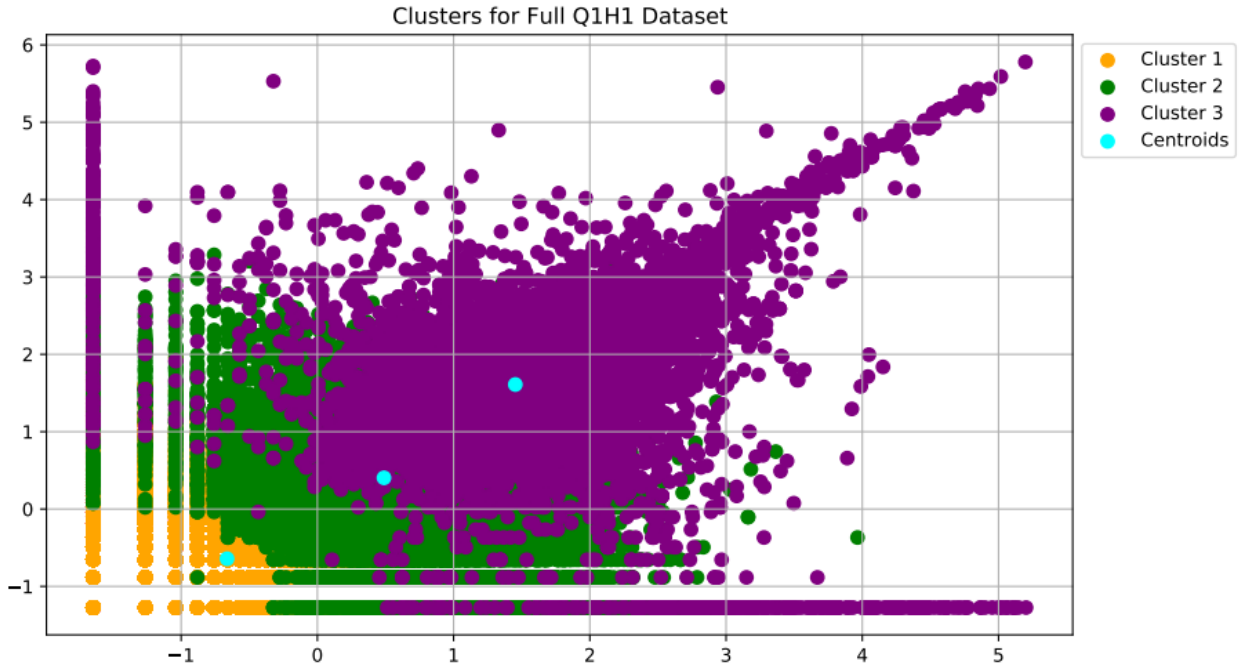

Fig. 3. K-means clustering performed on the normalized dataset relevant to the first hypothesis of the first research question.

The clustering results for the data are questionable, at best. Although three clusters seemed the optimal number for the dataset, the clusters don't appear to be well differentiated when plotted out.

The adjusted random score values for the clusters of this hypothesis were calculated by the program as 0.01478. The number's close proximity to zero indicated that applying the clustering model is nearly equivalent to assigning the data points to random clusters. In other words, the clusters don't appear to be significant.

This finding does not necessarily contradict the findings discovered in the paper and reinforced in the earlier parts of the experiment. Rather, this suggests that attempting to bypass the error introduced by human classification on the dataset is not viable. The lack of significant clusters and the lack of similarity between the ground truth established by the paper, and the data points themselves, shows this. The most likely cause of this is the nature of the dataset itself; the high number of variables, combined with the distribution of the variables and the datasets relative sparsity, makes it ill-fit to perform clustering on.

Due to the fact that the clustering was inconclusive, various types of supervised learning were attempted, which are shown in Figure 7 and Table 5.

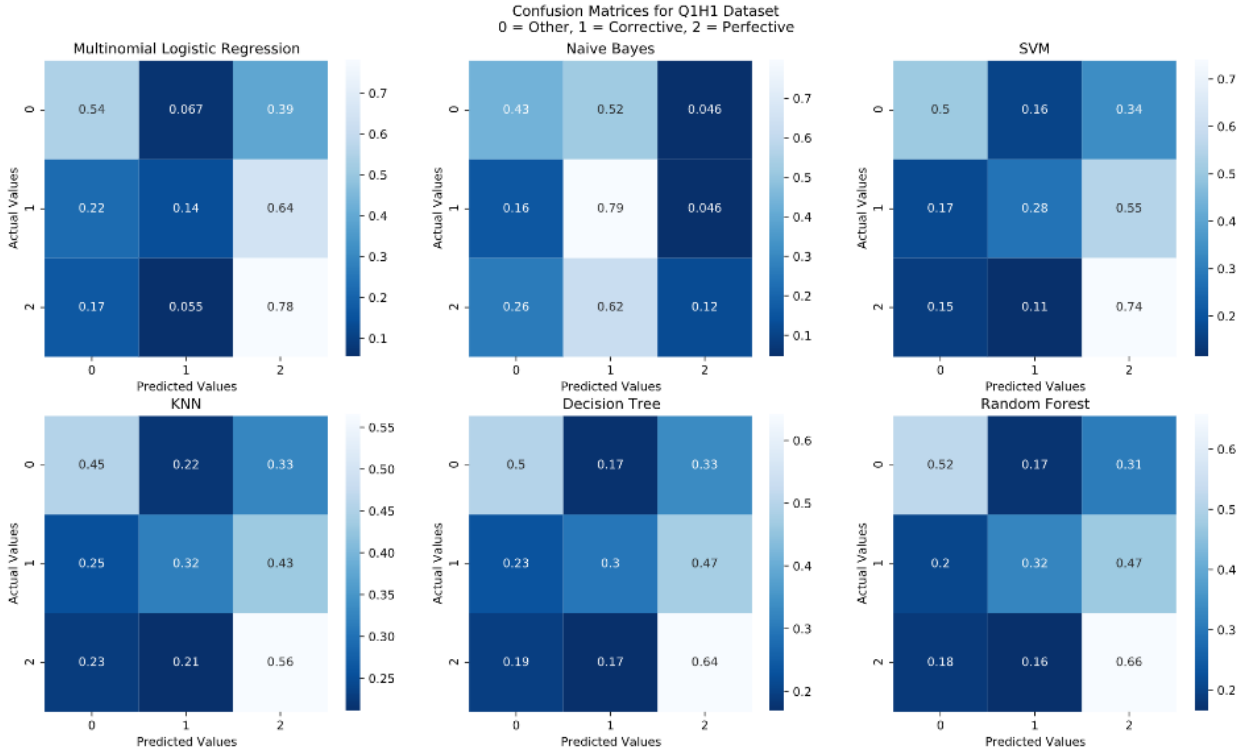

Fig. 4. Confusion matrices of various types of supervised models on the data relevant to hypothesis one.

Supervised Classifier Metrics for Q1H1 Dataset				
	Accuracy	Precision	Recall	F1 Score
Multinomial Logistic Regression	0.5198	0.5179	0.4868	0.4605
Naïve Bayes	0.4164	0.4926	0.4506	0.3864
SVM	**0.5295**	**0.5246**	**0.5062**	**0.5003**
KNN	0.5170	0.5088	0.5001	0.4986
Decistion Tree	0.4575	0.4473	0.4450	0.4442
Random Forest	0.4939	0.4830	0.4763	0.4736

Table 1. Quality metrics of the various types of supervised models on the data relevant to hypothesis one.

Looking through these confusion matrices suggests how other clustering models would handle sorting the dataset into three distinct classes. Broadly speaking, almost all of these unsupervised methods appear to be most confident in their classification of data points into class 2, corresponding to the perfective class. This is interesting as the perfective class was also suggested to be significantly different from both the corrective and other classes in terms of size changes. Therefore, the models largely being able to identify this class suggest that this class has the numerical significance suggested by the base paper and the earlier experiments.

The standout method for this dataset is clearly the SVM classifier, which scores highest in all quality metrics for correctly sorting the data from the values in Table 5. Other classifiers that consistently achieved higher quality metrics are the KNN and Multinomial Logistic Regression models. SVM models are an excellent method for handling sparse datasets, which leads to their widespread use in file retrieval; this fact also means that they are well suited for the current dataset, which is sparse due to the high number of zeros it contains. On the other hand, models such as the Naive Bayes struggle with sparse datasets, so it is expected that this model would consistently return the worst scores of the models tested. The same can be seen reflected in Figure 7; SVM and the Multinomial Logistic Regression models were the strongest at classifying perfective instances, while Naive Bayes was best at classifying corrective instances but struggled overall. Importantly, while none of the models were able to classify commits consistently into the ‘other’ class, they did perform better than picking at random, as the clusters did.

All told, however, the best performing classifier of all of the present classifiers may have gotten more meaningful results than the clustering algorithms, but the results are still not strong. This is most likely due to the high number of variables included in the dataset and the distribution of those variables not being Gaussian, and instead skewed.

The second hypothesis of the first research question states, “Intended quality improvements impact software quality metric values in a positive way.” In order to test this hypothesis in an unsupervised context, the clustering model was expanded to run on the dataset again, this time clustering the datapoints based on the same series of 14 distinct variables tested in the paper. To verify that three clusters was still the appropriate number for the dataset, the silhouette scores were calculated along with the values for the elbow method, as seen in Figure 8.

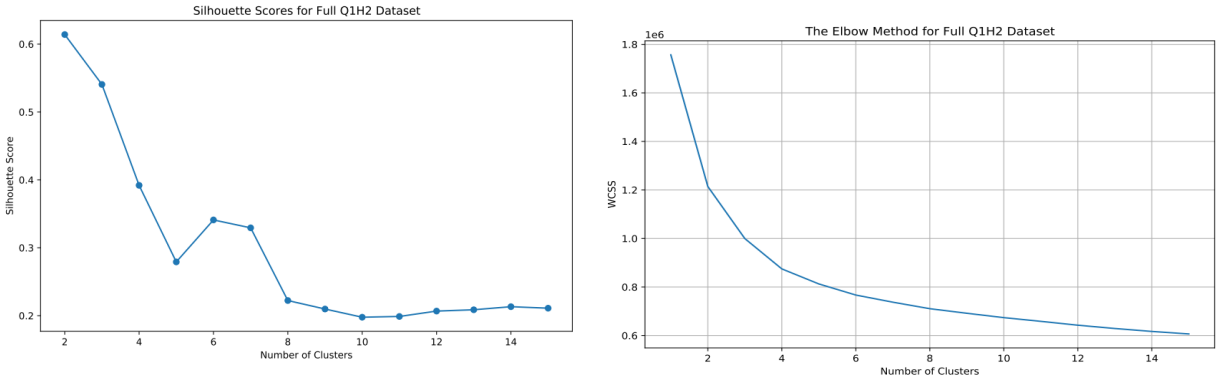

Fig. 5. Silhouette Scores (left) and the Elbow graph (right) for the data with respect to the 14 variables relevant to the paper, in order to determine the appropriate amount of clusters without supervision.

Unlike before, the silhouette scores for the second hypothesis data in Figure 8 seem to suggest that the optimal number of clusters for the underlying data is not three, but five. This would appear to be further supported by the elbow method, which sees the greatest change in its slope not around three, but around four or five. However, due to the paper’s use of only three distinct clusters for classification, the model was run using only three clusters.

The same transformations that were applied to the data previously were applied to the entire dataset, in order to make it feasible to run the clustering algorithm. The results of the k-means clustering for the second hypothesis data can be seen in Figure 9.

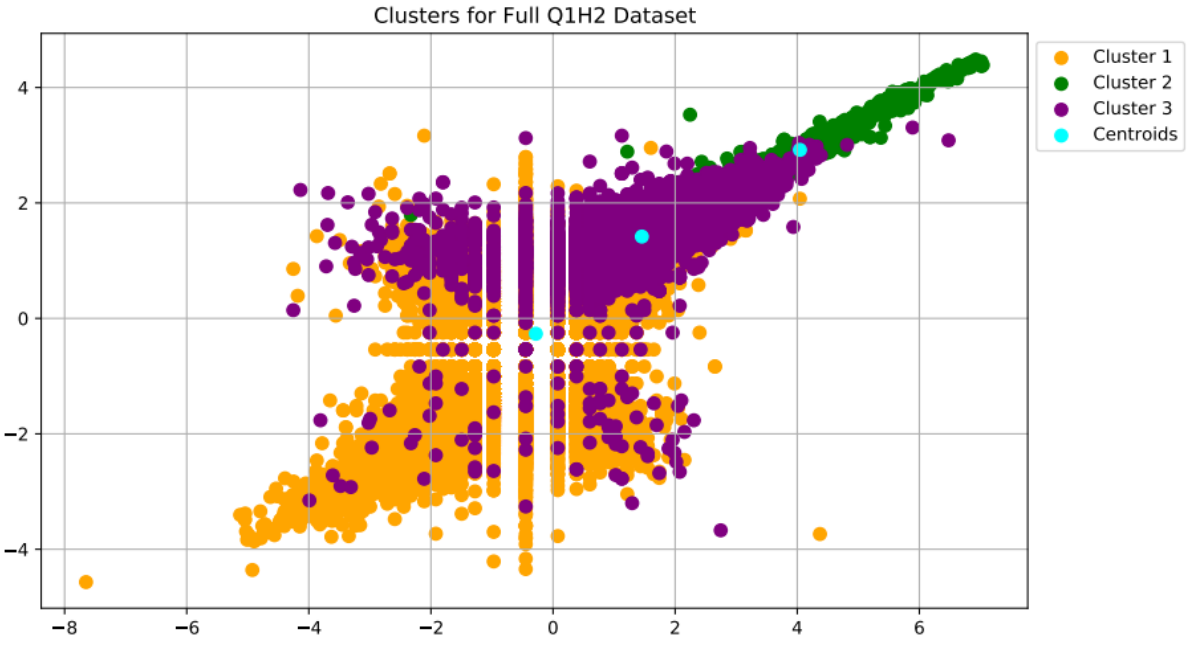

Fig. 6. K-means clustering performed on the normalized dataset relevant to the second hypothesis of the first research question.

The pattern of clusters in the second hypothesis data is unusual. As suggested by the elbow method and silhouette scores shown in Figure 8, the number of true clusters in the transformed data appears to be four rather than 3, with each cluster taking up a quadrant of the graph centered around 0. As before, this may indicate that the decision to create three distinct clusters to the dataset is not supported by the dataset's underlying values. However, this may also be due to the transformations required to pass the data through the clustering algorithm.

Due to the inconclusive nature of the clustering patterns, a series of supervised models were run on the same dataset.

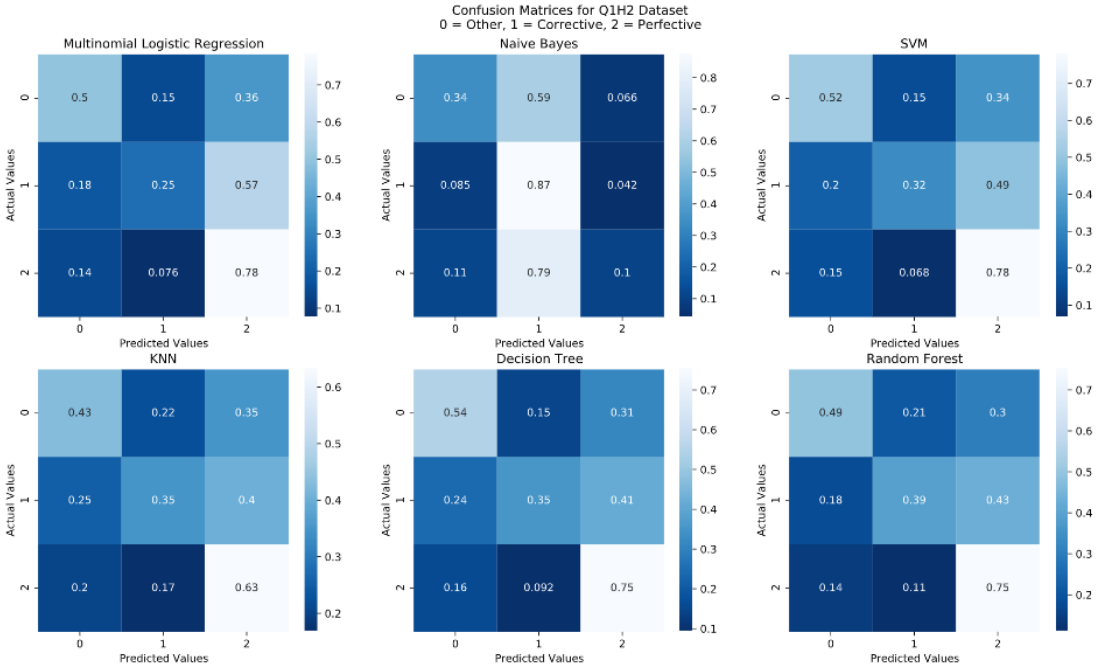

Fig. 8. Confusion matrices of various types of supervised models on the data relevant to hypothesis two.

Supervised Classifier Metrics for Q1H2 Dataset				
	Accuracy	Precision	Recall	F1 Score
Multinomial Logistic Regression	0.5355	0.5326	0.5084	0.4979
Naïve Bayes	0.3990	0.4987	0.4390	0.3641
SVM	0.5605	0.5633	0.5373	0.5324
KNN	0.5613	0.5570	0.5434	0.5413
Decistion Tree	0.4833	0.4730	0.4692	0.4673
Random Forest	**0.5691**	**0.5658**	**0.5488**	**0.5459**

Table 2. Quality metrics of the various types of supervised models on the data relevant to hypothesis one.

Previously, SVM was shown to perform the best of all of the introduced supervised models based on the metrics in Table 6; however, with all of the data being included, Random Forests have instead outperformed the other models. This could be due to the sheer size of the dataset relative to the previous one. SVM is well built for handling sparse datasets, but Random Forests is a more robust method when it comes to handling large amounts of data.

As before, the Naive Bayes method was the best at classifying corrective changes, as seen in Figure 10; and also consistent with the previous observations is the fact that SVM and Multinomial Logistic Regression were the best at classifying perfective changes.

Based on the values gathered in Table 6, it's clear that the addition of data may have improved the accuracy of the process, as this Random Forest has an accuracy of about 57% compared to SVM's 53% for the first hypothesis dataset. However, this increase in accuracy is still far from what would be considered a highly accurate model. Once again, this is most likely due to the fact that the dataset itself simply isn't well-fitted to analysis with these models, given its bias and sparsity.

Given these reasons, the attempts to apply other models to the dataset gathered in the base paper is inconclusive.

4 THREATS TO VALIDITY

There are four basic types of validity that should be considered in regard to this experiment, including conclusion validity, internal validity, construct validity, and external validity.

4.1 Conclusion Validity

A threat to conclusion validity is that when performing unsupervised clustering to compare to the results of a supervised algorithm, there is no guarantee that the clusters will be accurate. This is due to the potential disconnect between the author's intended input which is used in the paper, and the raw numbers, which are used in the clustering.

Another threat to conclusion validity comes from noise in clustering. Our clustering results showed significant noise from all 18 variables being used to cluster. Noise in our clustering model poses a problem to conclusion validity because it impacts the ability to draw accurate conclusions about the relationships demonstrated. The noise that exists in the clustering model reduces the overall cluster quality and degrades the algorithm, therefore calling the overall results into question [27].

4.2 Internal Validity

There are internal threats to validity in our experiment that consistently follow the threats to validity in the original research paper. While in our experiment, we did not manually inspect the commit messages; lack of information, incorrect information, or missing information could pose a problem to the validity of the experiment. The sample of commit messages that our

experiment and results were based on were part of a convenience sample, meaning that the projects were specifically selected and not chosen from a random sample of projects that fit the criteria for the experiment. A random sample would produce more reliable results [28], but since we used the previously classified commit messages, we were unable to change the sample.

4.3 Construct Validity

Issues also exist in this experiment under the category of construct validity. Some of the authors for the original experiment were new researchers who had recently received their PhDs only months before the research article was published. While they have valuable software engineering experience, they are not representative of professionals with significant experience in the field. The fact that the research team consisted of some less experienced professionals poses a problem to construct validity because this limits the ability to generalize from laboratory situations (the experiment) to real-world generalizations [29].

4.4 External Validity

External threats to validity include the fact that all of the projects selected for the original experiment were Java projects. While Java remains one of the most popular and widely used object-oriented programming languages, it would have been beneficial to include projects in other languages for a more diverse dataset or experimental group. Diversification of the projects used in the classification would limit the threats to external validity because it would be more representative of the diversity in real-world code. Another external threat to validity in our experiment is that all of the projects are open source. This is a field-wide problem, as defined by Wright in *Validity Concerns in Software Engineering* [30]. There is an issue with data source selection in many software experiments, but the ease, cost, and convenience of using open-source projects make them a popular choice for most research projects. Open-source projects can still be used for research, but the threat to external validity must be acknowledged, and the application of the conclusions to proprietary projects.

5 CONCLUSION

Software quality is a complex metric to try to understand [1]. Firstly, many different quality metrics exist for measuring the quality of software. Software quality is difficult to measure partly because there are many aspects that must be understood [2] but also because of the subjectivity of “quality”. Understanding the quality of software is vital for many reasons

including development of future software, learning from previous mistakes or errors, and improving the experience for both the developers and the users of the software that is being developed.

While software quality measurements remain a subjective matter, several conclusions were able to be drawn. Once it was understood what the developers intended to change in code, it was possible to find clarity about what types of changes were made with the purpose of improving quality. The utilized dataset originated from the experiment in 'What really changes when developers intend to improve their source code: a commit-level study of static metric value and static analysis warning changes' which contained 2,533 commits from 54 open source Java projects which were classified into perfective, corrective, or other. These commits were manually classified by the original researchers due to time and feasibility constraints. A model was built using the 2,533 commit messages which allowed us to classify the full dataset which contained 125,482 commit messages.

The full dataset was used to answer two of the three research questions that were in the original experiment. First, "Does developer intent to improve internal or external quality have a positive impact on software metric values?" as well as, "What kind of files are the target of internal or external quality improvements?" The original study was expanded upon by introducing four new variables to be evaluated. These 'new' metrics include the number of throw statements, the number of nodes, the number of method declarations, and the number of method invocations. Further expansion of the original study deployed clustering models and a selection of supervised classifiers.

After the replication of the original experiment and the expansion, it was found that for the first research question: 'Does developer intent to improve internal or external quality have a positive impact on software metric values?', the results do not contradict the findings in the original paper. The findings of the original study were that metric value changes of perfective commits were significantly different from non-perfective commits. The findings also included that most metric value changes of perfective commits have a positive impact on most metric values. The clustering methods to expand the original findings were largely inconclusive. The results for the supervised models relevant to hypothesis one proved to be better than that of the clustering models, but were still not very strong.

Following the replication of the original experiment and the expansion, it was found that for the second research question, 'What kind of files are the target of internal or external quality improvements' both the clustering model produced and supervised models produced inconclusive results. The results aside of the expansion do not contradict the findings of the original study which were that complex files are not necessarily the target for quality increasing work and it is more likely that perfective changes are applied to less complex files than in

non-perfective or corrective commits. Files that are contained in corrective changes are more complex than files contained in either perfective or non-corrective changes. Corrective changes are applied to files which are already complex and become more complex after the changes are applied.

REFERENCES

[1] J. A. Whittaker, "What is software testing? And why is it so hard?," in IEEE Software, vol. 17, no. 1, pp. 70-79, Jan.-Feb. 2000, doi: 10.1109/52.819971.

[2] Trautsch, A., Erbel, J., Herbold, S. et al. What really changes when developers intend to improve their source code: a commit-level study of static metric value and static analysis warning changes. Empir Software Eng 28, 30 (2023). https://doi.org/10.1007/s10664-022-10257-9

[3] *Static code analysis.* Static Code Analysis | OWASP Foundation. (n.d.). Retrieved February 10, 2023, from https://owasp.org/www-community/controls/Static_Code_Analysis

[4] M. J. C. Sousa and H. M. Moreira, "A survey on the Software Maintenance Process," Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272), Bethesda, MD, USA, 1998, pp. 265-274, doi: 10.1109/ICSM.1998.738518.

[5] *What is natural language processing?* IBM. Retrieved February 10, 2023, from https://www.ibm.com/topics/natural-language-processing#:~:text=Natural%20language%20processing%20(NLP)%20refers,same%20way%20human%20beings%20can.

[6] Mockus Votta (2000) Identifying reasons for software changes using historic databases. In: Proceedings 2000 international conference on software maintenance, pp 120-130.

[7] Pantiuchina J, Lanza M, Bavota G (2018) Improving code: the (mis) perception of quality metrics. In: 2018 IEEE International conference on software maintenance and evolution (ICSME), pp 80-91. https://doi.org/10.1109/ICSME.2018.00017

[8] Tian Y, Zhang Y, Stol KJ, Jiang L, Liu H (2022) What makes a good commit message? In: Proceedings of the 44th international conference on software engineering, ICSE ’22. Association for Computing Machinery, New York, pp 2389–2401. https://doi.org/10.1145/3510003.3510205

[9] Bavota G, De Lucia A, Di Penta M, Oliveto R, Palomba F (2015) An experimental investigation on the innate relationship between quality and refactoring. J Syst Softw 107:1–14. https://doi.org/10.1016/j.jss.2015.05.024. http://www.sciencedirect.com/science/article/pii/S0164121215001053

[10] Pantiuchina J, Zampetti F, Scalabrino S, Piantadosi V, Oliveto R, Bavota G, Penta MD (2020) Why developers refactor source code: a mining-based study. ACM Trans Softw Eng Methodol 29(4). https://doi.org/10.1145/3408302

[11] Mauczka A, Brosch F, Schanes C, Grechenig T (2015) Dataset of developer-labeled commit messages. In: Proceedings of the 12th working conference on mining software

repositories, MSR '15. http://dl.acm.org/citation.cfm?id=2820518.2820595. IEEE Press, Piscataway, pp 490–493

[12] Trautsch, A., Erbel, J., Herbold, S. et al. What really changes when developers intend to improve their source code: a commit-level study of static metric value and static analysis warning changes. Empir Software Eng 28, 30 (2023). https://doi.org/10.1007/s10664-022-10257-9

[13] Mockus Votta (2000) Identifying reasons for software changes using historic databases. In: Proceedings 2000 international conference on software maintenance, pp 120-130.

[14] Pantiuchina J, Lanza M, Bavota G (2018) Improving code: the (mis) perception of quality metrics. In: 2018 IEEE International conference on software maintenance and evolution (ICSME), pp 80-91. https://doi.org/10.1109/ICSME.2018.00017

[15] Honel S, Ericsson M, Lowe W, Wingkvist A (2019) Importance and aptitude of source code density for commit classification into maintenance activities. In: 2019 IEEE 19th international conference on software quality, reliability and security (QRS), pp 109–120. https://doi.org/10.1109/QRS.2019.00027

[16] Bakota T, Hegedus P, Kortvelyesi P, Ferenc R, Gyimothy T (2011) A probabilistic software quality model. In: 2011 27th IEEE international conference on software maintenance (ICSM), pp 243–252. https://doi.org/10.1109/ICSM.2011.6080791

[17] Bavota G, De Lucia A, Di Penta M, Oliveto R, Palomba F (2015) An experimental investigation on the innate relationship between quality and refactoring. J Syst Softw 107:1–14. https://doi.org/10.1016/j.jss.2015.05.024. http://www.sciencedirect.com/science/article/pii/S0164121215001053

[18] J. A. Whittaker, "What is software testing? And why is it so hard?," in IEEE Software, vol. 17, no. 1, pp. 70-79, Jan.-Feb. 2000, doi: 10.1109/52.819971.

[19] M. J. C. Sousa and H. M. Moreira, "A survey on the Software Maintenance Process," Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272), Bethesda, MD, USA, 1998, pp. 265-274, doi: 10.1109/ICSM.1998.738518.

[20] Mockus Votta (2000) Identifying reasons for software changes using historic databases. In: Proceedings 2000 international conference on software maintenance, pp 120-130.

[21] Pantiuchina J, Lanza M, Bavota G (2018) Improving code: the (mis) perception of quality metrics. In: 2018 IEEE International conference on software maintenance and evolution (ICSME), pp 80-91. https://doi.org/10.1109/ICSME.2018.00017

[22] Pantiuchina J, Zampetti F, Scalabrino S, Piantadosi V, Oliveto R, Bavota G, Penta MD (2020) Why developers refactor source code: a mining-based study. ACM Trans Softw Eng Methodol 29(4). https://doi.org/10.1145/3408302

[23] *Static code analysis.* Static Code Analysis | OWASP Foundation. (n.d.). Retrieved February 10, 2023, from https://owasp.org/www-community/controls/Static_Code_Analysis

[24] Tian Y, Zhang Y, Stol KJ, Jiang L, Liu H (2022) What makes a good commit message? In: Proceedings of the 44th international conference on software engineering, ICSE ’22. Association for Computing Machinery, New York, pp 2389–2401. https://doi.org/10.1145/3510003.3510205

[25] Trautsch, A., Erbel, J., Herbold, S. et al. What really changes when developers intend to improve their source code: a commit-level study of static metric value and static analysis warning changes. Empir Software Eng 28, 30 (2023). https://doi.org/10.1007/s10664-022-10257-9

[26] *What is natural language processing?* IBM. Retrieved February 10, 2023, from https://www.ibm.com/topics/natural-language-processing#:~:text=Natural%20language%20processing%20(NLP)%20refers,same%20way%20human%20beings%20can.

[27] A. Kaur, P. Kumar and P. Kumar, "Effect of noise on the performance of clustering techniques," 2010 International Conference on Networking and Information Technology, Manila, Philippines, 2010, pp. 504-506, doi: 10.1109/ICNIT.2010.5508461.

[28] Kriska, D., Sass, M. M., & Fulcomer, M. C. (n.d.). *Assessing limitations and uses of convenience samples: A guide for graduate students*. ScholarWorks. Retrieved April 7, 2023, from https://scholarworks.waldenu.edu/facpubs/783/

[29] Petursdottir, A. I., & Carr, J. E. (2018). Applying the taxonomy of validity threats from mainstream research design to single-case experiments in Applied Behavior Analysis. *Behavior Analysis in Practice*, *11*(3), 228–240. https://doi.org/10.1007/s40617-018-00294-6

[30] Wright, H. K., Kim, M., & Perry, D. E. (2010). Validity concerns in Software Engineering Research. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. https://doi.org/10.1145/1882362.1882446