Commit Message Matters: Investigating Impact and Evolution of Commit Message Quality

Jiawei Li
University of California, Irvine
Irvine, CA, USA
jiawl28@uci.edu

Iftekhar Ahmed
University of California, Irvine
Irvine, CA, USA
iftekha@uci.edu

2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00076

***Abstract*—Commit messages play an important role in communication among developers. To measure the quality of commit messages, researchers have defined what semantically constitutes a *Good* commit message: it should have both the summary of the code change (*What*) and the motivation/reason behind it (*Why*). The presence of the issue report/pull request links referenced in a commit message has been treated as a way of providing *Why* information. In this study, we found several quality issues that could hamper the links' ability to provide *Why* information. Based on this observation, we developed a machine learning classifier for automatically identifying whether a commit message has *What* and *Why* information by considering both the commit messages and the link contents. This classifier outperforms state-of-the-art machine learning classifiers by 12 percentage points improvement in the F1 score. With the improved classifier, we conducted a mixed method empirical analysis and found that: (1) Commit message quality has an impact on software defect proneness, and (2) the overall quality of the commit messages decreases over time, while developers believe they are writing better commit messages. All the research artifacts (i.e., tools, scripts, and data) of this study are available on the accompanying website [2].**

***Index Terms*—Commit message quality, software defect proneness, empirical analysis**

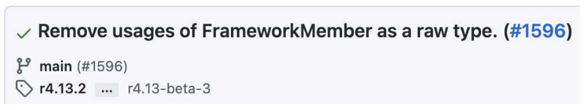
I. Introduction

Software developers commit code changes during the course of software development and maintenance using version control systems [34]. These code changes are usually accompanied by commit messages written in natural language, which is a free-form textual description of its corresponding change. The message summarizes what happened in the change and/or explains why the change was made [11], [44], [50].

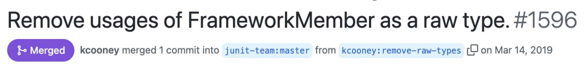
These commit messages play a vital role in modern software development, especially in Open-Source Software (OSS), by serving as one of the communication channels through which developers communicate the context of a change to collaborators [47]. Commit messages are also regarded as the documentation of software changes which can help program comprehension and software maintenance [44] in long-lived projects where commit messages might be the only source of information left for future developers to understand what changes were made and why those were made [65].

While it is evident that commit messages play an important role in communication among developers, till recently there was no approach for measuring the commit message quality that considers the semantics of a commit message. Researchers had defined various syntactic rules to measure the quality of commit messages [4], [13], [15], but these approaches did not capture the semantic meaning of a commit message. Tian et al. [65] first defined what semantically constitutes a *Good* commit message where they proposed that a *Good* commit message should include both a summary of the changes in the commit (noted as *What*) and a description of the reasons or justifications for the changes (noted as *Why*). As long as a commit message contained a link to the issue reports or pull requests, the authors considered the commit message to have provided *Why* information without examining the link contents.

However, the presence of a link does not guarantee the quality of the link content and the presence of *Why* information. Researchers have shown that such links attached to commit messages might become outdated, resulting in commit messages that are difficult to understand [41]. In addition, the contents of those links might be badly written or do not provide any additional information. For example, in Figure 1, the content of the pull request link attached to this commit message only repeated what the commit message had stated. [1]. There is no additional information in the link content for getting the *Why* information.


✓ Remove usages of FrameworkMember as a raw type. (#1596)
main (#1596)
r4.13.2 ... r4.13-beta-3

(a) Commit Message


Remove usages of FrameworkMember as a raw type. #1596

kcooney merged 1 commit into junit-team:master from kcooney:remove-raw-types on Mar 14, 2019

(b) Referenced Pull Request

Fig. 1: A Commit Message and its Referenced Pull Request Link Content

Based on such observations we posit that it is necessary to consider the contents of the links in order to decide whether commit messages include *Why* information instead of simply relying on the presence of a link. This prompted us to ask our first research question:

[1]shorturl.at/jrtvG

RQ1: Does the issue report/pull request link referenced in a commit message always provide *Why* information?

Since commit messages are heavily used for communication among OSS developers, in the absence of other forms of documentation (which is very common in the development of some OSS projects [3]), low-quality commit messages can have a negative impact on the overall quality of the software. Therefore, following the state-of-the-art semantic definition of a *Good* commit message [65], we take the first step to investigate the impacts of commit message quality on software quality, at a large scale.

In this work, we are specifically investigating the impact of commit message quality on software defect proneness [63]. We posit that low-quality commit messages can severely impede developers' comprehension of the existing code, which may lead to software defects in the future. While prior research investigated the relationships between commit message length and defect proneness [8], they only relied on syntactic rules to capture the quality of a commit message. They did not consider the semantic quality of commit messages and their association with defect proneness. This prompted us to ask our second research question:

RQ2: How does commit message quality impact software defect proneness?

Similar to any other artifacts in an OSS project, commits evolve over time in terms of size, the number of unique commit messages, etc. [4] While only one prior research investigated the evolution of commit message quality [4], they studied the evolution of commit message quality in 5 projects by measuring (among others) the number of unique commit messages and found that the quality of commits declined over time. They did not consider either the syntactic or semantic quality of the commit message. As a result, there is still a lack of understanding regarding the evolution of commit message quality over time. In this study, we aim to fill this gap by analyzing the evolution of commit message quality at a large scale, considering the semantic quality of commit messages to answer the following research question:

RQ3: How does the quality of the commit messages in OSS projects evolve over time?

To answer our RQs, we conducted formative interviews with 13 OSS developers and performed empirical analyses on 32 Apache projects. We opted to investigate Apache projects because these projects have well-documented discussion procedures and long histories (i.e., 16k commits on average). We manually checked commit messages with issue report/pull request links to investigate if these links have quality issues that hamper their ability to provide *Why* information (RQ1). Then, we manually labeled commit messages with links from the dataset curated by Tian et al. [65] and used Machine Learning (ML) classification to identify *Good* commit messages by considering both commit messages and link contents (*What* and *Why*). Through the interviews, we found developers agreed that commit message quality has an impact on software defect proneness (RQ2). We also asked our interviewees about the evolution of their own commit message quality over time (RQ3). To triangulate our results, we surveyed 93 developers who had contributed to 32 Apache projects. Finally, using our ML classifier, we conducted empirical analyses to further understand the impact of commit message quality (RQ2) and its evolution (RQ3).

The significance of our contributions are following:

(1) We show that issue/pull request link content should be considered when measuring commit message quality.

(2) We analyze the impact of commit message quality on OSS software defect proneness.

(3) We analyze the evolution of commit message quality.

The remainder of this paper is organized as follows: In Section II, we review related research on commit message quality. We outline our data labeling, collection, and analysis pipeline in Section III. Next, we present our observations in Section IV. Then, we discuss the implications for our study in Section V. Section VI shows potential threats to the validity of our reported findings. Finally, we conclude with a summary of the key findings in Section VII.

II. Related Works

A. Commit Message Quality Measurement and Evolution

Various syntactic rules for measuring commit message quality have been proposed. Chahal et al. [13] composed ten rules and 11 relevant attributes for measuring commit message quality and built a model that could determine commit message quality. Chen et al. [15] focused on measuring the quality of commit messages based on their expressiveness measured by commit message length, uniqueness of commit message title, and word frequency. However, these rules and attributes only considered the syntactic characteristics of a commit message to evaluate its quality while ignoring the semantics.

More recently, Tian et al. [65] defined what semantically constitutes a high-quality commit message. They showed that a *Good* commit message should include both a summary of the changes in the commit (noted as *What*) and a description of the motivations for the changes (noted as *Why*). However, they assumed that *Why* information is present as long as the issue report/pull request links are mentioned in the commit message without considering the content of those links. In our study, we measure commit message quality by considering the information from both the commit message and the content of the issue report/pull request links associated with it.

Researchers have studied commit message quality evolution. Agrawal et al. [4] used the size of commit comments, the ratio of the total number of unique commit messages to the total number of commits, and the number of unique commit messages as a proxy for measuring commit message quality. Their results showed that the quality of commits declines over time. However, their proposed metrics did not evaluate the quality of commit message content. In our study, we measured commit message quality by considering the semantics of both commit message content and the content of the associated link.

B. Evaluation of Generated Commit Message from Commit Message Generation (CMG) Techniques

To improve the overall quality of commit messages and standardize the writing style, Commit Message Generation (CMG) techniques have been widely studied by researchers [12], [19], [34], [36], [42], [44], [64], [68]. All of these approaches rely on reference messages that are written by software developers to evaluate their techniques. However, prior works showed that developer-written messages may not always be of high quality [24], [43], [65]. So automated approach for measuring the quality of the commit message using its content is needed. In this work, we focus on developing an ML classifier that measures the quality of the commit message using a semantic aware commit message quality measurement criterion [65].

C. Impact of Commit Message Quality

Very few research works have investigated the impact of commit message quality. Barnett et al. [8] analyzed the relationship between the defect proneness and commit message detail (i.e., the length of the commit message). They found that including commit message length as a feature could add explanatory power to Just-In-Time defect prediction models [51]. However, they did not consider the impact of including *What* and *Why* information on software defect proneness. Santo et al. [59] pointed out that software build failure is marginally related to the "unusualness" of commit message measured by large language models. In addition, Lu et al. [46] explored the association between the topics extracted from commit messages and technical debt. Their results showed that detailed commit messages are negatively associated with technical debt, while empty commit messages may have a positive association. All of these studies only considered syntactic rules for measuring the commit message quality. However, to have an accurate measure of a commit message quality, it is necessary to consider the semantics [65]. In this study, we take the first step to analyze the impact of commit message quality on software defect proneness by using the most state-of-the-art quality definition that considers the semantic information of not only the commit message but also the link content associated with the commit.

III. Methodology

Our goal was to investigate whether the issue report/pull request link referenced in a commit message always helps to provide *Why* information. Then, we aimed to analyze the impact and evolution of commit message quality in OSS projects. In the following subsections, we detail the applied methodology. Figure 2 shows an overview of our methodology.

A. Formative Interviews

We start our investigation by exploring how developers perceive the use of issue report/pull request links in commit messages, their commit message writing quality evolution, and the impact of commit message quality on software defect proneness. We conducted semi-structured interviews with 13

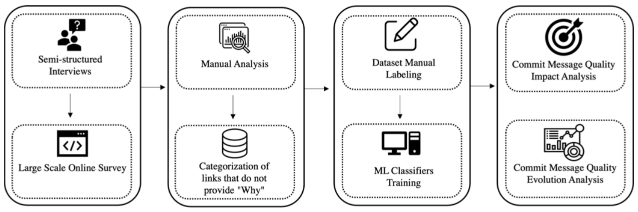

Semi-structured Interviews
Large Scale Online Survey
Manual Analysis
Categorization of links that do not provide "Why"
Dataset Manual Labeling
ML Classifiers Training
Commit Message Quality Impact Analysis
Commit Message Quality Evolution Analysis

Fig. 2: Overview of Research Method

OSS contributors. We recruited the participants by using social network sites. On average, they have 3 years of OSS development experience and are currently working as software engineers in the industry.

The interviews were done remotely using Zoom, lasting around 20 minutes each. Most questions we asked during the interviews were about the impacts of not including *What* or *Why* information on software defect proneness. We also asked the participants about their own commit message writing style and its evolution in terms of *What* and *Why*. See supplementary [2] for interview questions.

B. Large Scale Online Survey

To further validate our findings from the interviews and acquire a more generalized perception, we surveyed a larger population of OSS developers who had contributed to Apache projects. We decided to use Apache since these projects have a well-defined and adhered process for code contribution, and they are established OSS projects that were studied by a number of previous works [5], [47].

Survey design: Our survey comprised 20 questions, a mix of multiple-choice, Likert scale, and open-ended questions (see supplemental for the survey questions [2]). The survey included demographics questions (Q2-Q3), questions about participants' writing styles for commit messages (Q4-Q5), questions about participants' opinions and expectations for the links in terms of providing *Why* information for commit messages (Q13-Q14), question about their awareness of commit message writing guidelines (Q15), questions about the impacts of commit message quality on software defect proneness (Q11-Q12), and finally questions about participants' perceptions of their own commit message writing quality evolution (Q17-Q18). In addition, we also include "Text entry" options to allow developers to provide additional commit message quality impacts on software development activities (Q6-Q10), the reasons for their commit message writing change trends (Q19), and other possible metrics that can also be considered to estimate commit message quality (Q16). We conducted five pilot studies with two graduate students and three professionals with OSS experience. The participants for the pilot were identified using snowball sampling [29]. After each pilot study, we collected feedback and refined the survey based on the feedback. The pilot survey responses were used solely to improve the questions and these responses were not included in the final results.

Survey participants: In order to identify participants, we relied on 32 Apache projects in the list curated by Mannan et al. [47], where referencing to issue report/pull request links in commit messages is a common practice. The details of the project selection are explained in Section III-D. We used the GitHub API [52] to mine contributor emails from these 32 projects. After removing emails of accounts that were deleted or private, we were left with 2,771 developer email addresses in total.

Survey responses: We used Qualtrics [55] as a distribution platform to deploy our survey. We emailed the survey to 2,771 developers (following university-approved IRB protocol), and 114 emails bounced (giving 2,657 valid emails). To increase survey participation, we followed the best practices described by Smith et al. [62], such as sending personalized invitations and allowing participants to remain anonymous. We also sent a reminder email after the first week. Thus, the survey was open for two weeks in total, during which we received 93 responses or a response rate of 3.50%. The response rate is consistent with other studies in software engineering [65].

Survey data analysis: In our survey, over half (55.26%) of our respondents had 2 to 10 years of open-source experience, and 31.58% were senior contributors with over 20 years of programming experience. We quantitatively analyzed the closed-ended questions to understand developers' expectations related to including links, commit message quality's impact on software defect proneness, and evolution of their commit message writing quality.

C. Manual Analysis

We did a manual inspection to check whether issue report/pull request links always provide *Why* information. We used the same commit message dataset used by Tian et al. [65], which includes 1,597 commit messages from 5 large and popular Java OSS projects. Each message was labeled based on the information it included (i.e., *Why*, *What*). The detailed dataset construction process can be found in [65]. Since we are interested in commit messages that have issue report/pull request links, we identified 611 commit messages meeting this criterion. Then, we manually analyzed them to check if there exist issues that hamper the links' ability to include *Why* information. To do so, we followed an open coding protocol [27] to find and categorize the potential link quality issues from the 611 commit messages. Specifically, two authors of this study independently checked all 611 commit messages by examining both the link contents and the commit messages for *Why* information. In case *Why* could not be found in both of them, we coded the issues into types. During the analysis, each emerging category was compared with existing ones to determine if it is a new category through multiple comparison sessions. Finally, the two authors exchanged ideas for the categorization and reached a consensus through negotiated agreement [25].

D. Commit Message Quality Analysis

In this section, we detailed the approach we used to analyze commit message quality's impact on software defect proneness, and its evolution in OSS projects.

Project Selection: First, we decided to focus on projects using Java since it is one of the most widely-used programming languages [66]. Second, we selected 32 Apache projects written in Java on Github. We decided to focus on Apache projects since numerous prior research papers have used Apache projects as the subject of analysis [14], [26], [39]. Our rationale is that since Apache projects are more systematic and mature with structured guidelines for project contribution [1], contributors are more likely to write relatively high-quality commit messages in these projects. So any effect we see of commit message quality on defect proneness in these projects is going to be many-fold worse in projects that lack structured guidelines for project contribution. We collected project duration, developer profile, and project size by analyzing the git repositories. In total, we gathered 520,306 commits. Table I provides a summary of our selected Apache projects.

TABLE I: Project Statistics

Dimension	Max	Min	Average	Median
Line count	2,593,845	58,656	592,070.2	430,215.5
Duration (weeks)	669	228	437.7	415.5
# Developer	1,044	12	171.2	79.5
# Commits	35,471	4,242	16,259.6	14,247.5

Training Data Refinement: Since it is impractical to manually evaluate the quality of thousands of commit messages, we decided to use ML technique. While Tian et al. [65] had already trained ML classifiers to identify what type of information (i.e., *Why*, *What*) is contained in a commit message, they only considered the textual part of a commit message and treated the presence of links as providing *Why* without examining the link contents during their manual labeling of the training dataset. We improve upon the training dataset from Tian et al. [65] by considering information from both link content and commit message.

We used the definition of *Why* and *What* provided by Tian et al. [65] and looked for information that should be in a commit message, and manually checked 611 commit messages that contained issue report/pull request links. The two authors of this study labeled those 611 commit messages independently to determine if they contain *Why* information while retaining labels about *What*. To be more specific, we first checked the commit message's textual content to see whether it contains *Why* information. If *Why* information was not found in the commit message, we examined the content of the link referenced in it to check if the link content provided *Why* information. After labeling the 611 messages, Cohen's kappa coefficient of agreement between the two authors was 0.95, which is a perfect agreement [40]. As for the messages the two authors did not agree on, we held meetings to resolve 7 (approx. 1.15%) disagreements.

ML Classifier Training: With our refined version of the training dataset, we built three separate binary classifiers for identifying *What*, *Why*, and *Good* (which considered both *What*, and *Why*).

Since we considered the referenced link content along with the message body to determine a commit message's quality, we extracted link content titles as the titles of issue reports/pull requests usually summarize all valuable information [70], [71]. We then replaced the links in commit messages with the extracted titles to prepare the enhanced commit messages.

We selected BERT [21] to tokenize and embed the enhanced commit messages into numeric vectors since using such embedding has shown outstanding performance in text classification tasks [28]. The vectors were then input into various ML classification architectures. In this study, we considered the most widely-used ML classification techniques, including Bidirectional Long Short-Term Memory (Bi-LSTM) [60], Bidirectional Gated Recurrent Units (Bi-GRU) [17], [60], XGBoost [16], and Support Vector Machine (SVM) [69]. Bi-LSTM was used by Tian et al. [65], so we selected it for comparison purposes.

To ensure the best performance, we applied hyper-parameter tuning to all classifiers. By applying GridSearch [35] and BayesSearch [9] on a wide range of hyper-parameters, we found the optimal hyper-parameters for each classifier. Due to space limitations, we provide the selected hyper-parameters in our replication package [2]. The models were trained and evaluated using 10-fold cross-validation. That is, the data was randomly divided into ten equal splits, and nine of them were used for training and one for evaluating performance. We trained our models using this method ten times and reported the mean scores. To further boost our classifier's performance, we used majority/hard voting [56] to ensemble the three top classifiers among the compared classifiers, namely, Bi-LSTM, Bi-GRU, and XGBoost.

In addition, we also tested the trained Bi-LSTM classifier from Tian et al. [65] on our newly labeled dataset to see if their model's performance was negatively affected by their limitation in dataset labeling. Table II shows the precision, recall, F1, and accuracy scores of all classifiers, where we highlighted the best scores (i.e. F1, Precision, Recall, and Accuracy) in identifying a given type of information (i.e. *Why*). Compared with the Bi-LSTM classifier that was trained on the original dataset from Tian et al., our ensemble ML (ensemble of Bi-LSTM, Bi-GRU, and XGBoost trained on our refined dataset) classifier increased 12 percentage points and 9 percentage points in F1 for identifying *Why* and *Good*. This indicates that Tian et al.s' model was negatively affected by their limitation in dataset labeling. Since the label for *What* did not get changed, its performance for *What* classifier remained almost the same with a difference of 0.02. We also present the confusion matrix of our ensemble ML model that classifies *Why* information in Figure 3. We can infer from the confusion matrix that neither class (*Why*, No *Why*) is disproportionately impacted by the misclassifications. For the rest of the paper, we use the classification results by the ensemble ML classifier.

Analysis of Commit Message Quality's Impact on Defect Proneness: To answer RQ2, we aimed to investigate if there is any difference in the commit message quality of the commits related to a defect-introducing commit and a non defect-introducing commit. We deployed a widely-used defect-introducing commit detection tool SZZUnleashed [10] to identify defect-introducing commits across the collected commit histories. Due to the fact that all Apache projects in our analysis have long commit histories (average 16k commits), we could not manage to finish running SZZUnleashed on all our selected projects even after running the analysis for two months. We ended up analyzing 238k commits from 17 projects.

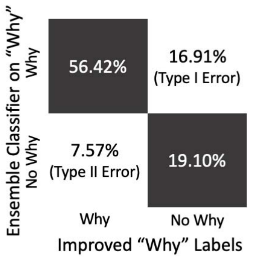


Fig. 3: Confusion Matrix for *"Why"* Predictions

Then, we used RefMiner [67] to detect refactoring commits and then excluded them from our analysis because such commits did not change the behavior of the code regardless of the quality of its preceding commits. This left us with 185,026 commits. We also filtered out bot commit messages using the same approach as in [22], [65]. That is, we analyzed the variability of all contributors' commit message writing patterns, and we identified bots if the variability of the messages generated is lower than a threshold proposed by [22]. After filtering, we have 91,926 commits for our commit message quality impact analysis. Out of these 91,926 commits, SZZUnleashed identified 15,174 (16.5%) commits as defect-introducing ones.

Next, for every remaining commit, we set a window in the commit history that contains a number of commits preceding every commit in terms of commit date and calculated *Window Quality Score* using 1. The *Window Quality Score* represents an estimate of the commit message quality where *Positive labels* in the equation stands for the fact that a commit message has *Why*, *What*, or both. We should note that we did not calculate the *Window Quality Score* for those initial commits that have fewer preceding commits than the window size because we believed they did not have enough history for us to analyze the commit message quality impact.

Choosing a proper window size can be challenging. For example, if the window size is small, we would face difficulties in capturing the long-term effect of commit message quality. On the other hand, if the window size is large, old commits far back in the commit history would be analyzed that might not actually have an impact on the current commit at all. So, we conducted our experiments on a wide range of window sizes. Specifically, we set 5, 10, 20, 50, 100, 200, 400, 800, and 1000 as our window sizes. Finally, we used Welch's t-test [58] and

TABLE II: ML classification techniques evaluation

Techniques	**F1**			**Precision**			**Recall**			**Accuracy**		
	Why	**What**	**Good**	**Why**	**What**	**Good**	**Why**	**What**	**Good**	**Why**	**What**	**Good**
Bi-LSTM from [65]	0.685	***0.951***	0.653	***0.860***	***0.970***	***0.820***	0.570	0.932	0.543	0.671	***0.913***	***0.700***
Bi-LSTM	0.787	0.926	0.718	0.711	0.922	0.634	0.880	0.929	0.827	0.699	0.875	0.665
Bi-GRU	0.781	0.927	0.722	0.723	0.918	0.648	0.848	0.937	0.815	0.703	0.875	0.679
XGBoost	0.803	0.925	0.699	0.751	0.871	0.704	0.860	0.987	0.697	0.729	0.867	0.687
SVM	0.781	0.921	0.652	0.751	0.864	0.682	0.823	***0.989***	0.648	0.711	0.859	0.661
Ensemble ML	***0.808***	0.935	***0.744***	0.740	0.916	0.657	***0.890***	0.956	***0.857***	***0.730***	0.888	0.691

Cohen's D [23] to analyze the difference in *Window Quality Score* between defect-introducing commits and non defect-introducing commits. We decided to use Welch's t-test because it does not assume equal variance between groups but only assumes normality of the data. Since we are performing multiple tests, we have to adjust the significance value accordingly to account for multiple hypothesis corrections. We use the Bonferroni correction [30], which gives us an adjusted p-value of 0.006.

$$Window\ Quality\ Score = \frac{\#\ Commits\ with\ positive\ labels\ in\ the\ window}{Window\ size} \quad (1)$$

One of the limitations of the above-mentioned commit-level analysis is that unrelated commits also get analyzed. For example, not all preceding commits in the analysis window changed the files that are being modified by the current commit. To mitigate this, we conducted a *file-level* analysis. We followed a similar procedure as commit-level analysis. The only difference was that only those preceding commits were considered that modify the files that were also being modified by the current commit. If the total number of preceding commits that changed the same files that were also changed by the current commit is less than the set window size, we use this total number as the denominator rather than the window size in Equation 1. We also excluded those commits that changed more than 100 files since such commits might introduce noise that is caused by routine maintenance (i.e., typo fixes, copyright updates.) [48]. Finally, we analyzed the score difference by using Welch's t-test and effect size using Cohen's D.

Barnett et al. [8] have found commit message length in terms of the word count (*Commit Message Volume*) has an association with defect proneness. However, they did not consider measuring commit message quality in terms of *What* and *Why*. In this study, we wanted to check whether our *Window Quality Score* in terms of *What* and *Why* has a stronger association with defect proneness compared to *Commit Message Volume*. In order to do so, we built a Generalized Linear Regression model (GLM) [32]. The dependent variable (whether the commit is a defect-introducing commit or not) follows a Poisson distribution. Therefore, we used a Poisson regression model with a log linking function. The independent variables were *What* and *Why*, and *Commit Message Volume*. In order to calculate *Commit Message Volume* for each commit message, we used NLTK [45] to remove common stop words and then tokenize the message to get the word count.

After collecting these metrics, we checked for multi-collinearity using the Variance Inflation Factor (VIF) of each predictor in our model [18]. VIF describes the level of multi-collinearity (correlation between predictors). A VIF score between 1 and 5 indicates a moderate correlation with other factors. We found that all three independent variables had a VIF score greater than five. So instead of using all factors at once, we built three separate models using each of the independent variables. This step was necessary since the presence of highly correlated factors forces the estimated regression coefficient of one variable to depend on other predictor variables that are included in the model. Finally, we conducted Welch's t-test [58] and Cohen's D analysis [23] on the model's coefficients to understand if one is statistically more associated than another with defect proneness across all window sizes.

One important point to note is that our goal was not to build a state-of-the-art defect prediction model. Our goal was to check the level of association of *What*, *Why* in comparison to *Commit Message Volume*. Hence, we did not build a full-placed defect prediction model using the state-of-art features identified in defect prediction literature and focused only on comparing between *What*, *Why*, and *Commit Message Volume*.

Commit Message Quality Evolution Analysis: To investigate how software artifacts evolve over time, researchers have used releases, individual commits, and discrete-time units (years, months, weeks, days) [5], [33]. Individual commits would be too fine-grained for our purpose. Therefore, following previous works [5], [47], we selected "week" as our unit of analysis because it provides us with enough information for analyzing the evolution of commit message quality. We cut off our analysis at 415 weeks (median duration of all selected projects) to prevent extremely long-lived projects skewing our results.

Using the ML classifier's predictions on all human-written commit messages (246,735 commits in total after filtering out bots from 32 projects), we had an estimate of the quality for each commit message in terms of (*Why*, *What*, *Good*). To compare the overall commit message quality across weeks, we normalized the labeled data by calculating the ratio between the number of positive labeled commits (i.e., the commit's corresponding message contains *Why*, *What*, or *Good*) to the number of all the commits made in the particular week (*Weekly Quality Score* in Equation 2), which gave a score between 0 and 1. To get a bigger picture, we averaged the *Weekly Quality Score* of all projects resulting in a quality score for each week across all projects. This overall quality score was then used in our analysis since our goal is to investigate the general evolution trends of commit message quality across projects. Finally, we performed a Spearman correlation [53] analysis on the ratio scores and week numbers to estimate the overall commit message quality evolution trend over time.

It's widely accepted that a relatively small number of core developers are responsible for more than 80% of the contributions in any OSS projects [47]. We used this principle to classify a developer as a *core* developer if they are among the top 20% of the developers in terms of the number of commits authored, and a *non-core* developer otherwise [5], [47], [49]. In this study, we used emails as developers' identifiers. We collected their emails and contributions by using git commands (i.e., git log) in the downloaded git repositories. Then, similar to the overall commit message quality evolution analysis, we calculated *Weekly Quality Score* (Equation 2) for each group of developers across weeks (i.e., out of all the commit messages written by a group of developers in a week, how many of them have *What*, *Why*, or *Good*). We then averaged the *Weekly Quality Score* across all projects for analyzing a general evolution trend.

$$Weekly\ Quality\ Score = \frac{\#\ Commits\ with\ positive\ labels\ in\ a\ week}{\#\ Commits\ made\ in\ a\ week} \quad (2)$$

IV. Results

We organized the results of this study based on our research questions in Section I.

A. Inclusion of Why information by attaching links

Our manual inspection of the commit messages found several commits with different types of issues that negatively affected the links' ability of providing *Why* information. We list the identified issues below:

Broken Link URL: The links in the commit messages are broken with 404 error message returned. We found 1 such commit. The information contained in the links would be lost if the link is broken [37]

Content Repetition: The link contents simply repeat what the commit messages have stated. No additional information such as *Why* could be acquired. [38] We found 59 such commits.

Badly Written Texts: The link contents are either written in poor English making the meaning elusive (i.e. "I18N effort for dubbo code base - dubbo-plugin", "duplicate decrease for ExecuteLimitFilter onError #4380") or the link content only contains several words that make pinpointing *Why* information difficult (i.e. "DER encoder"). [7] We found 9 such commits.

Additional Information on What: The link contents only provide additional *What* information for the commit messages (i.e. Commit message: add @SPI annotation (#6436), Link content: add @SPI annotation to ExtensionFactory #6436) [6]. We found 21 such commits.

From the problem categories listed above, we could see that the most common one is *Content Repetition*. This contradicts what the developers expect from a link since majority of our survey respondents (89.77%) expected additional information in the links when the commit message itself does not provide *Why*.

Observation 1: Contrary to developers' expectations, 15% of issue reports/pull requests do not provide *Why* information when referenced in a commit message.

Since there exist quality issues that hamper issue report/pull request links' ability to provide *Why* information, considering both commit messages and link contents becomes important to determine if commit messages include *Why* information. Following this principle, we found 89 (14.57%) commit messages that were incorrectly labeled as having *Why* in Tian et al.'s original dataset after our training data refinement. Then, we trained ML models using our refined training dataset, and the performance of the models for classifying *Why* gets improved by up to 12 percentage points in the F1 score (Table II), which also indicates that it is necessary to consider referenced link contents in addition to commit message to estimate its quality.

Observation 2: It is necessary to consider both commit messages and link contents while estimating a commit message's quality.

B. Impact of Commit Message Quality

Commit-level analysis: In this study, we used *Window Quality Score* (Equation 1) to estimate the commit message quality of commit history for every commit (See Section for III details). Next, we used Welch's t-test [58] to check if the *Window Quality Score* between defect-introducing commits and non defect-introducing commits are statistically significantly different or not. We also used Cohen's D [23] to measure the effect size.

We list our results in Table III. Results show that the difference is statistically significant across all window sizes for *What*, *Why*, and *Good*, indicating that the commit message quality of prior commits does have an impact on the current commit's defect-proneness regardless of how far back we go in the history. However, we note that the effect size is very small across all window sizes.

TABLE III: Commit-level Analysis: Difference in commit message quality of commit history between defect-introducing commits and non defect-introducing commits

Window Size	**Welch's t-test p-value**			**Cohen's D**		
	Why	**What**	**Good**	**Why**	**What**	**Good**
5	***0.00045***	***9.39883e-06***	***0.00095***	0.03168	0.03751	0.02911
10	***7.21172e-07***	***1.22335e-09***	***2.59436e-06***	0.04566	0.05053	0.04136
20	***1.41449e-06***	***2.97021e-17***	***2.26556e-09***	0.04514	0.06841	0.05272
50	***3.89681e-07***	***4.90136e-21***	***5.14493e-10***	0.04740	0.07620	0.05478
100	***0.00155***	***8.71832e-18***	***1.41634e-06***	0.02972	0.07131	0.04280
200	***0.00544***	***1.35567e-10***	***2.87051e-06***	0.02565	0.05459	0.04142
500	***7.90503e-09***	***0.00282***	***0.00510***	0.05599	0.02812	0.02603
800	***1.68103e-32***	***1.32784e-20***	***1.66446e-19***	0.11966	0.09197	0.08711
1000	***5.11631e-47***	***5.25024e-33***	***9.56609e-33***	0.14742	0.12043	0.11686

File-level analysis: Similar to commit-level analysis, we calculated the *Window Quality Scores* for each commit window size. The only difference was that only those preceding commits were considered that modified the files that were also being modified by the current commit. The results are in Table IV. Although the effect size is small, there is statistically significant difference in the *Window Quality Score* between defect-introducing commits and and non defect-introducing commits.

TABLE IV: File-level Analysis: Difference in commit message quality of commit history between defect-introducing commits and non defect-introducing commits

Window Size	**Welch's t-test p-value**			**Cohen's D**		
	Why	**What**	**Good**	**Why**	**What**	**Good**
5	***2.35592e-19***	***3.72234e-11***	***1.77846e-22***	0.09121	0.06700	0.10012
10	***9.42900e-19***	***0.00079***	***1.13363e-14***	0.09191	0.03523	0.08057
20	***6.40050e-21***	***3.91683e-08***	***2.34095e-18***	0.09644	0.05554	0.08948
50	***1.60110e-22***	***4.66404e-10***	***2.16190e-17***	0.10138	0.06378	0.08901
100	***2.19140e-10***	***0.01863***	***3.63092e-08***	0.07830	0.02917	0.13972
200	***1.44070e-08***	0.48416	***0.00224***	0.06268	0.00765	0.03380
500	***2.32601e-19***	***0.02141***	***6.43257e-12***	0.10113	0.02620	0.07755
800	***1.24497e-08***	0.84772	***0.00049***	0.06302	0.00210	0.03869
1000	***1.51677e-05***	***0.01347***	0.45646	0.04867	0.02819	0.00839

Observation 3: Preceding commit message quality in terms of *What* and *Why* has a statistically significant impact on the defect proneness of future commits with a very small effect size.

Feature Importance Analysis: In this study, we wanted to check whether our *Window Quality Score* in terms of *What*, and *Why* has stronger association with defect proneness compared to *Commit Message Volume* which was shown by Barnett et al. [8] to have association with defect proneness. The dependent variable was whether the commit is a defect-introducing commit or not and the independent variables were *Window Quality Score* in terms of *What* and *Why*, and *Commit Message Volume*. The details of the regression model building for this purpose is in Section III.

We calculated McFadden's Adjusted R^2 as a quality indicator of the model because there is no direct equivalent of R^2 metric for Poisson regression. The ordinary least square (OLS) regression approach to goodness-of-fit does not apply to Poisson regression. Moreover, adjusted R^2 values like McFadden's can not be interpreted as one would interpret OLS R^2 values. McFadden's Adjusted R^2 values tend to be considerably lower than those of the R^2. Values of 0.2 to 0.4 represent an excellent fit [31]. The McFadden Adjusted R^2 [31] of these models were smaller than 0.002. In our case, this was expected since we used only one factor at a time instead of using multiple factors for model building. Since we care about the association of the factors with defect proneness, not about the model's capability to explain overall variability, we focus on regression coefficients instead of McFadden's Adjusted R^2.

Table V shows the coefficients of regression model built for commit level analysis for different independent variables across all window sizes. The coefficient values of our *Window Quality Scores* for both *What* and *Why* are larger than that of *Commit Message Volume*. We also found significant difference between the coefficients of *Commit Message Volume* and *What* (Welch's t-test, p-val<6.660e-05, Cohen's D(3.55, large)), and between *What* and *Why* (Welch's t-test, p-val<0.005, Cohen's D(1.732, large)) across all window sizes.

TABLE V: Commit level: Coefficients of GLM model

Window Size	**Coefficient of Volume**	**Coefficient of What**	**Coefficient of Why**
5	0.00373	0.11740	0.08787
10	0.00367	0.19327	0.17434
20	0.00363	0.34908	0.14494
50	0.00361	0.44871	0.11038
100	0.00364	0.48256	0.09678
200	0.00370	0.37022	0.13113
500	0.00383	0.22888	0.50962
800	0.00390	0.25426	0.73425
1000	0.00391	0.29600	0.83342

For file-level analysis, we saw similar results shown in Table VI . The coefficient values of our *Window Quality Scores* for file level analysis in terms of both *What* and *Why* are larger than that of *Commit Message Volume*. A significant difference was found between the coefficient of *Commit Message Volume* and *Why* (Welch's t-test, p-val<1.153e-06, Cohen's D(6.134, large)), and between *What* and *Why* (Welch's t-test, p-val<3.606e-05, Cohen's D(3.095, large)).

TABLE VI: File level: Coefficients of GLM model

Window Size	**Coefficient of Volume**	**Coefficient of What**	**Coefficient of Why**
5	0.00393	0.24889	0.43222
10	0.00388	0.00295	0.55609
20	0.00421	0.34908	0.59306
50	0.00485	0.12108	0.64131
100	0.00366	0.16979	0.77763
200	0.00439	0.26760	0.45286
500	0.00452	0.25065	0.44335
800	0.00416	0.28544	0.45567
1000	0.00437	0.56136	0.42511

Observation 4: *What* and *Why* information in a commit message has a significantly higher association with defect proneness compared to *Commit Message Volume*.

Impact of Commit Message Quality-interview: When asked during the interview, majority (84.61%) of our interviewees agreed that the quality of existing commit messages has an impact on the code changes that they commit and has the potential to introduce or prevent defects from getting into the code repository. We also asked our participants if *Good* commit message in general is important in preventing software defects from being introduced into the code base. 61.53% of them gave us a positive answer. One of the interviewees explicitly commented that *"if developers don't write Good commit messages, other developers may not fully understand the code change made in that commit, which may cause software issues"*[I-1][2].

Impact of Commit Message Quality-survey: Our survey respondents also agreed that the quality of existing commit messages impacts the code changes they commit. 37.36%

[2]S-N refers to survey participant number and I-N to interview participant number.

answered “Definitely yes”, while 34.07% answered “Probably yes”. We also asked them if *Good* commit messages help ensure software quality, almost all of them (96.67%) agreed. The results corroborated the feedback from the interviews.

Then, we asked our survey participants to rank the importance of each type of information (*Why*, *What*) in a commit message in preventing software defects. Figure 4 shows that both types of information (*What* and *Why*) in a commit message have the potential to affect software defect proneness.

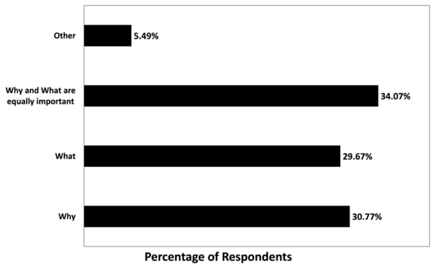


Fig. 4: Survey respondents’ perspectives on the importance of each type of information in preventing software defects

Observation 5: Commit message quality has an impact on software defect proneness, and *Good* commit messages can help to prevent software defects.

C. Evolution of Commit Message Quality

Commit Message Quality Evolution: We conducted Spearman correlation analysis [53] between the week numbers and the corresponding *Weekly Quality Scores* (Equation 2) to analyze the evolution of commit message quality. The overall quality of commit messages decreases significantly over time in terms of *What* (Spearman correlation coefficient=-0.79953, p-value=1.78787e-93), *Why* (Spearman correlation coefficient=-0.42674, p-value=8.51816e-20), and *Good* (Spearman correlation coefficient=-0.80203, p-value=1.78787e-94). From the correlation coefficients, we can see that commit message quality in terms of *What* (-0.79953) and *Good* (-0.80203) decreased more rapidly than *Why* (-0.42674). To have an understanding of the overall evolution trend, we looked at the *Weekly Quality Scores* across all projects and found that commit message quality degrades over time. Figure 5 shows the trends.

Observation 6: The overall commit message quality degrades over time.

Developers’ Writing Quality Evolution: We conducted Spearman correlation analysis [53] between the week numbers and the corresponding *Weekly Quality Scores* to analyze the evolution of commit message quality of *core* and *non-core* developers. The results of the Spearman correlation are shown in Table VII. For *core* developers, their commit message quality in terms of containing *Why* information became better over time. Meanwhile, *What* and the overall quality (i.e. *Good*)

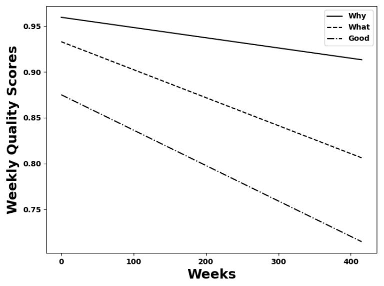


Fig. 5: Week-wise average commit message quality

got worse. As for *non-core* developers, their commit message quality in terms of both *Why* and *What* gradually became worse over time.

TABLE VII: Spearman correlation between week numbers and developers’ commit message writing quality

	Why		**What**		**Good**	
	Core	**Non-core**	**Core**	**Non-core**	**Core**	**Non-core**
Correlation	0.11534	-0.28654	-0.64504	-0.65961	-0.48316	-0.63566
p-value	***0.01874***	***2.76536e-09***	***3.41329e-50***	***3.59693e-53***	***1.15959e-25***	***2.32192e-48***

Developers’ Writing Quality Evolution-interview: 46.15% of our interviewees mentioned that they tended to write more *Good* commit messages over time, with the goal of *“saving other developers’ time in reading commit messages”*[I-2], *“making code base cleaner”*[I-3], or*“following companies’ guidelines”*[I-4]. 38.46% only paid more attention to writing *What* while *Why* was sometimes ignored. One developer commented that *“Writing Why information would make a commit message unnecessarily long so that I don’t write Why”*[I-5]. Thus, there are developers whose commit message writing quality degrades over time, while half of our interviewees believed they wrote better commit messages over time.

Developers’ Writing Quality Evolution-survey: In terms of writing *Why* information in commit messages over time, 61.90% of respondents tended to write *Why*, while 28.57% believed they tended to ignore writing *Why*. For writing *What*, 77.38% paid more attention to it over time while only 10.71% tended to ignore it. Moreover, eight respondents followed other writing styles, such as *“Only writing Why or What for complicated code changes”*[S-1], *“Implying What by writing Why, including Why in code comments instead of commit messages”*[S-2] etc. The remaining respondents either did not have a clear idea of how their commit message writing quality changed or tended not to write commit messages at all. From the results of our survey, we could see that more than half of our target developers believed that they tended to write better commit messages over time.

Observation 7: Overall commit message quality degrades over time, while more than half of the participating developers believed they wrote better quality commit messages over time.

V. Discussion

Our analysis showed that *What* and *Why* are more associated with defect proneness compared to *Commit Message Volume*. This might be happening because previous commits without *What* and *Why* information could confuse the developer who is making the current code change. This was also mentioned by one of the interviewees: *"if developers don't write Good commit messages, other developers may not fully understand the code change made in that commit, which may cause software issues."* [I-1]. Since *What* and *Why* are associated with defect proneness, one interesting future research would be to construct defect prediction models using *What* and *Why* as features to further boost prediction models' performance.

In addition to having an impact on defect proneness, our interviewees and survey respondents also shared a variety of potential impacts of commit message quality on software. *"Not including Why or What information in commit messages would make other developers hard to understand the code change in the commits."* [I-6] *"The understandability/readability of the entire code base would be reduced if no Why/What"*[I-7] *"The software's handover in the future will be affected by the commit message quality."*[I-8]. All these responses suggest the need for future research on commit message quality's impact on other facets of software quality, including maintainability, code understandability, and project success.

Most survey participants believed their written commit message quality improves over time. However, we found the opposite trend through our analysis (Figure 5). This is a case where reality is different from developers' beliefs. Devanbu et al. [20] showed that developers' beliefs are primarily formed based on personal experience and do not necessarily match the real situation. Our results provide developers with evidence that they might have an erroneous view of their commit messages quality, and necessary steps should be taken to improve this situation since commit message quality has a negative impact on software defect proneness and probably other quality aspects as well.

In terms of writing *What* and *Why*, *core* developers are doing better than *non-core* developers (Table VII). And surprisingly, both core and non-core developers are writing commit messages with a degrading quality of *What* even though developers believe that writing *Why* is more difficult compared to writing *What*. As one of the survey participants mentioned, *"My commit messages are focusing on the what. The why would usually require more context and is linked in the PR/issue, probably because creating a good Why summary is more difficult"*[S-3]. One probable reason behind this could be that developers believe that *What* information can be easily deduced from the patch, and that's why developers are not paying attention to ensure the *What* information. As one of the participants mentioned *"The Why usually cannot be derived from the code changes. The What is obvious from the changes in the commit"*[S-4]. Further investigation is required to understand the exact reason behind this.

Although the overall quality of commit messages in terms of including *Why* remains relatively high (i.e., around 0.95 of *Weekly Quality Score*), *non-core* developers may need to pay more attention to including *Why* information when writing commit messages. As they become *core* developers in the future, most commit messages will be composed by them, so it's really important for them to include *Why* as one survey respondent commented *"There will be more bugs when future developers change this code. Future developers will be more afraid to change this code if it's tricky code since they don't understand the author's intent"*[S-5]. Developers also indicated that writing *Why* information requires the context of the change, and it is not easy to convey that concisely in commit messages. This highlights the need for future research to build tools that can automatically generate summarized *Why* by collecting information from the context (i.e., commit history) of the project.

VI. Threats To Validity

We have taken all reasonable steps to mitigate potential threats that could hamper the validity of this study, it is still possible that our mitigation strategies might not have been effective.

Construct validity It is possible that our survey participants misunderstood the questions or were led to give a specific answer based on how the questions were phrased and how the answer options were provided. To mitigate this threat, we conducted pilot studies with developers of different backgrounds and experiences from the OSS community. We updated the survey based on the feedback from these pilot studies to ensure there was no bias or difficulties in understanding the questions. For example, we tried our best to make our survey questions neutrally worded. Moreover, we increased the variety of the available response options for multiple-choice questions by providing a wide range of choices or "Others" option. Since the responses of multiple-choice questions are distributed instead of one choice dominating all responses, it indicates that our effort was successful.

We categorized the developers into *core* and *non-core* groups based on the number of commits they contributed. Some of the developers could have been categorized as *non-core*, but in fact, they were *core* developers who might focus on high-level architecture design that does not require a large number of total commits.

Internal validity It is possible that our manual labeling process could have introduced unintentional bias. To address this, two authors inspected and labeled independently. A Cohen kappa of 0.95 indicates a high reliability of our labeling.

For identifying defect-introducing commits, we used SZZUnleashed [10]. For identifying refactorings, we used RefMiner [67]. Just like any tool, there are inherent limitations of these tools that could lead to missing defect-introducing commits and refactoring commits. However, these tools have been validated and used in other studies [54], [57], [61], making them reliable.

External validity Our conclusions may not be generalizable to projects that do not use Java or are hosted on other version control systems except GitHub. Moreover, we conducted our impact and evolution analysis only on Apache projects. It is possible that the conclusions from these analyses may not apply to other OSS projects.

VII. Conclusion and Future Work

Our result shows that along with the commit message, the content of the associated issue report/pull request link needs to be considered for evaluating the quality of a commit message. Based on this observation, we developed a ML classifier for automatically identifying whether a commit message has *What* and *Why* information by considering both the commit messages and the link contents. This classifier outperforms state-of-the-art classifiers by 12 percentage points improvement in F1 score.

Our results also highlighted that prior commit messages quality in terms of *What* and/or *Why* information is associated with the current commit's defect proneness. We also found that the overall quality of the commit messages decreases over time, while developers believe the opposite.

The results reported in this paper lay the foundation for our future work. In addition to the future directions already presented in the discussion section (Section V), future research entails an investigation of commit message quality's impact on software design quality, analysis of the root causes for commit message quality degradation, and the creation of a more comprehensive commit message quality dataset that covers projects written in multiple programming languages.

Data Availability: All the research artifacts (i.e., tools, scripts, and data) of this study are available in [2].

References

[1] "Apache developer guidelines." [Online]. Available: https://www.apache.org/dev/

[2] "Our replication package." [Online]. Available: https://zenodo.org/record/7042943#.YxG_ROzMLdo

[3] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1199–1210.

[4] K. Agrawal, S. Amreen, and A. Mockus, "Commit quality in five high performance computing projects," in *2015 IEEE/ACM 1st International Workshop on Software Engineering for High Performance Computing in Science*. IEEE, 2015, pp. 24–29.

[5] I. Ahmed, U. A. Mannan, R. Gopinath, and C. Jensen, "An empirical study of design degradation: How software projects get worse over time," in *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*. IEEE, 2015, pp. 1–10.

[6] Apache, "add @spi annotation (6436)." [Online]. Available: https://github.com/apache/dubbo/commit/40d03b081e8a8f74fb5092163e51bdd17842cb5a

[7] ——, "Der encoder (6139)." [Online]. Available: https://github.com/square/okhttp/commit/e736f927f82bcde9490b0e195f89d0a8884ba68b

[8] J. G. Barnett, C. K. Gathuru, L. S. Soldano, and S. McIntosh, "The relationship between commit message detail and defect proneness in java projects on github," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 496–499.

[9] "https://scikit-optimize.github.io/stable/ modules/generated/skopt.bayessearchcv.html."

[10] M. Borg, O. Svensson, K. Berg, and D. Hansson, "Szz unleashed: an open implementation of the szz algorithm-featuring example usage in a study of just-in-time bug prediction for the jenkins project," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 7–12.

[11] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 33–42.

[12] ——, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*. Antwerp, Belgium: ACM Press, 2010, p. 33. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1858996.1859005

[13] K. K. Chahal and M. Saini, "Developer dynamics and syntactic quality of commit messages in oss projects," in *IFIP International Conference on Open Source Systems*. Springer, 2018, pp. 61–76.

[14] B. Chen *et al.*, "Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation," *Empirical Software Engineering*, vol. 22, no. 1, pp. 330–374, 2017.

[15] D. Chen and S. E. Goldin, "A project-level investigation of software commit comments and code quality," in *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*. IEEE, 2020, pp. 240–245.

[16] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[18] P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/-correlation analysis for the behavioral sciences*. Psychology press, 2014.

[19] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2014, pp. 275–284.

[20] P. Devanbu, T. Zimmermann, and C. Bird, "Belief & evidence in empirical software engineering," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 108–119.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[22] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, and A. Mockus, "Detecting and characterizing bots that commit code," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 209–219.

[23] M. J. Diener, "Cohen's d," *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.

[24] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 422–431.

[25] J. Forman and L. Damschroder, "Qualitative content analysis," in *Empirical methods for bioethics: A primer*. Emerald Group Publishing Limited, 2007.

[26] M. Gharehyazie, D. Posnett, B. Vasilescu, and V. Filkov, "Developer initiation and social interactions in oss: A case study of the apache software foundation," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1318–1353, 2015.

[27] B. G. Glaser, "Open coding descriptions," *Grounded theory review*, vol. 15, no. 2, pp. 108–110, 2016.

[28] S. González-Carvajal and E. C. Garrido-Merchán, "Comparing bert against traditional machine learning text classification," *arXiv preprint arXiv:2005.13012*, 2020.

[29] L. A. Goodman, "Snowball sampling," *The annals of mathematical statistics*, pp. 148–170, 1961.

[30] W. Haynes, "Bonferroni correction," in *Encyclopedia of Systems Biology*. Springer, 2013, pp. 154–154.

[31] D. A. Hensher and P. R. Stopher, "Behavioural travel modelling," in *Behavioural travel modelling*. Routledge, 2021, pp. 11–52.

[32] J. M. Hilbe, *Logistic regression models*. Chapman and hall/CRC, 2009.

[33] C. Izurieta and J. M. Bieman, "Testing consequences of grime buildup in object oriented design patterns," in *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 2008, pp. 171–179.

[34] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in

2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017, pp. 135–146.

[35] Á. B. Jiménez, J. L. Lázaro, and J. R. Dorronsoro, "Finding optimal model parameters by discrete grid search," in *Innovations in Hybrid Intelligent Systems*. Springer, 2007, pp. 120–127.

[36] T.-H. Jung, "Commitbert: Commit message generation using pre-trained programming language model," *arXiv preprint arXiv:2105.14242*, 2021.

[37] Junit-Team, "Making super and sub member classes as suggested. · junit-team/junit4@3920f3f." [Online]. Available: https://github.com/junit-team/junit4/commit/3920f3fe11dffc904e1ab41a9ff4fc9d36b1c25b

[38] ——, "Remove usages of frameworkmember as a raw type. (1596)." [Online]. Available: https://github.com/junit-team/junit4/commit/6d0fad48ce3a05b32d903d2016c24d276b6e1eb8

[39] S. Kabinna, C.-P. Bezemer, W. Shang, and A. E. Hassan, "Logging library migrations: A case study for the apache software foundation projects," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 154–164.

[40] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, pp. 363–374, 1977.

[41] T.-D. B. Le, M. Linares-Vásquez, D. Lo, and D. Poshyvanyk, "Rclinker: Automated linking of issue reports and commits leveraging rich contextual information," in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 36–47.

[42] S. Liu, C. Gao, S. Chen, N. Lun Yiu, and Y. Liu, "ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9261989/

[43] S. Liu, C. Gao, S. Chen, N. L. Yiu, and Y. Liu, "Atom: Commit message generation based on abstract syntax tree and hybrid ranking," *IEEE Transactions on Software Engineering*, 2020.

[44] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.

[45] E. Loper and S. Bird, "Nltk: The natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[46] C. Lu, "But do commit messages matter? an empirical association analysis with technical debt," *Joint Proceedings of the Summer School on Software Maintenance and Evolution*, 2019.

[47] U. A. Mannan, I. Ahmed, C. Jensen, and A. Sarma, "On the relationship between design discussions and design quality: a case study of apache projects," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 543–555.

[48] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, 2017.

[49] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[50] A. Mockus and L. G. Votta, "Identifying reasons for software changes using historic databases." in *icsm*, 2000, pp. 120–130.

[57] G. Rosa, L. Pascarella, S. Scalabrino, R. Tufano, G. Bavota, M. Lanza, and R. Oliveto, "Evaluating szz implementations through a developer-informed oracle," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 436–447.

[51] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.

[52] T. Mombach and M. T. Valente, "Github rest api vs github archive: A comparative study," 2018.

[53] L. Myers and M. J. Sirois, "Spearman correlation coefficients, differences between," *Encyclopedia of statistical sciences*, vol. 12, 2004.

[54] R. Paul, A. K. Turzo, and A. Bosu, "Why security defects go unnoticed during code reviews? a case-control study of the chromium os project," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1373–1385.

[55] Qualtrics, "Qualtrics XM - Experience Management Software," sep 13 2015, [Online; accessed 2022-03-14].

[56] A. Rojarath, W. Songpan, and C. Pong-inwong, "Improved ensemble learning for classification techniques based on majority voting," in *2016 7th IEEE international conference on software engineering and service science (ICSESS)*. IEEE, 2016, pp. 107–110.

[58] G. D. Ruxton, "The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test," *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, 2006.

[59] E. A. Santos and A. Hindle, "Judging a commit by its cover," in *Proceedings of the 13th International Workshop on Mining Software Repositories-MSR*, vol. 16, 2016, pp. 504–507.

[60] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[61] G. Sellitto, E. Iannone, Z. Codabux, V. Lenarduzzi, A. De Lucia, F. Palomba, and F. Ferrucci, "Toward understanding the impact of refactoring on program comprehension," in *29th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2022, pp. 1–12.

[62] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International workshop on cooperative and human aspects of software engineering (CHASE)*. IEEE, 2013, pp. 89–92.

[63] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE transactions on software engineering*, vol. 37, no. 3, pp. 356–370, 2010.

[64] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "On the evaluation of commit message generation models: an experimental study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 126–136.

[65] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" *arXiv preprint arXiv:2202.02974*, 2022.

[66] TIOBE, "Tiobe index," http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[67] N. Tsantalis, A. Ketkar, and D. Dig, "Refactoringminer 2.0," *IEEE Transactions on Software Engineering*, 2020.

[68] B. Wang, M. Yan, Z. Liu, L. Xu, X. Xia, X. Zhang, and D. Yang, "Quality assurance for automated commit message generation," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 260–271.

[69] L. Wang, *Support vector machines: theory and applications*. Springer Science & Business Media, 2005, vol. 177.

[70] T. Zhang, I. C. Irsan, F. Thung, D. Han, D. Lo, and L. Jiang, "Automatic pull request title generation," *arXiv preprint arXiv:2206.10430*, 2022.

[71] ——, "itiger: An automatic issue title generation tool," *arXiv preprint arXiv:2206.10811*, 2022.