

CoMRAT: Commit Message Rationale Analysis Tool

Mouna Dhaouadi
DIRO, Université de Montréal
Montréal, Canada
0000-0001-9336-7714

Bentley James Oakes
GIGL, Polytechnique Montréal
Montréal, Canada
0000-0001-7558-1434

Michalis Famelis
DIRO, Université de Montréal
Montréal, Canada
0000-0003-3545-0274

Abstract—In collaborative open-source development, the rationale for code changes is often captured in commit messages, making them a rich source of valuable information. However, research on rationale in commit messages remains limited. In this paper, we present CoMRAT, a tool for analyzing decision and rationale sentences rationale in commit messages. CoMRAT enables a) researchers to produce metrics and analyses on rationale information in any Github module, and b) developers to check the amount of rationale in their commit messages. A preliminary evaluation suggests the tool’s usefulness and usability in both these research and development contexts.

Index Terms—rationale analysis, code commit messages, Github repository mining, Linux kernel analysis

I. INTRODUCTION

Reporting the *rationale* behind a proposed code change is a necessary practice in collaborative open-source projects. In modern software development, where developers rely on version control systems such as Git, rationale information is often documented in the commit messages [1]. Although researchers have previously attempted to develop an understanding of developers’ rationale in open source software by studying chat messages [2] or email archives [3], research about rationale characteristics in the code commit messages of open-source projects is rather limited [4]–[6]. Tian et al. [4] and Li et al. [5] reported analyses regarding the quality of commit messages in terms of *what* and *why* information based on only five open source projects, while our earlier work examined developers’ rationale specifically for the Out-Of-Memory Killer (OOM-Killer) module of the Linux Kernel [6].

We present CoMRAT, a **Commit Message Rationale Analysis Tool** for studying developer’s rationale in the commit messages of any open source Github project. CoMRAT is a web application that includes: a) a *Module Analyzer* for characterizing developer rationale in terms of its presence, impact factors, evolution and structure, and b) a *Commit Message Analyzer* for promoting rationale-providing commits. A preliminary evaluation suggests CoMRAT’s *usefulness* (RQs.1,3) and *usability* (RQs.2,4) for researchers and developers.

II. BACKGROUND

A. The OOM-Killer dataset

The Linux kernel is an extensive open-source project that has been developed collaboratively since 1991. Since 2005,

This research work is funded by the Fonds de Recherche du Québec (B2X).

Linux code patches have been organized as Git commits. Linux developers are encouraged to explain the motivation behind the commit and its impact on the kernel [7]. This community practice makes the kernel commits a valuable source of rationale information [1]. The Linux project has several sub-projects that focus on different areas of the kernel, e.g., the ‘mm’ folder contains code that focuses on memory management, the ‘fs’ folder code for filesystems, etc. Each sub-project contains several modules. For instance, the ‘mm’ sub-project contains the modules ‘oom_kill’, ‘slob’ and ‘migrate’. As these modules have different concerns and committers, we treat each one as their own individual project.

In our previous work, we created an annotated, high-quality rationale dataset for the OOM-Killer module [6]. We categorized sentences as *Decision* (an action or a change that has been made), *Rationale* (the reason for a decision or value judgment), and *Supporting Facts* (narration of facts used to support a decision), and then quantitatively analyzed the resulting dataset to characterize rationale in this subsystem. The next section details the analyses from [6] as implemented in CoMRAT. Section III illustrates them with an example.

B. Rationale Analyses

The first set of analyses report on the presence of rationale in commit messages. We consider that a commit contains rationale if at least one of its sentences is labelled as Rationale. Specifically, we answer “*How many commits contain rationale?*” by introducing the *rationale percentage* metric and “*How much of the commit contains rationale?*” by introducing the *rationale density* metric, as follows:

$$\text{rationale percentage} = \frac{\text{number of commits that contain rationale}}{\text{total number of commits}}$$

$$\text{rationale density} = \frac{\text{number of sentences labelled as Rationale}}{\text{total number of sentences in a commit}}$$

We also define the *average rationale density* for a specific module as follows:

$$\text{average rationale density} = \frac{\sum_{\text{commits}} \text{rationale density}}{\text{number of commits that contain rationale}}$$

The second set of analyses concern the possible factors impacting rationale, specifically the potential dependencies between the size of the commit and the developers experience,

and the rationale density. To answer “*Does the quantity of rationale reported depend on the commit message size?*”, we visualize the rationale density values versus the commit message size (i.e., number of sentences in a commit). To answer “*Does the quantity of rationale reported depend on the developer experience?*”, we visualize the average rationale density per author (i.e., we compute the mean of the *rationale density* of the commits of each author), along with the number of commits per author, as we consider the number of commits authored an indication of the developer’s experience.

For rationale evolution, we answer “*How does rationale evolve over time?*” by visualizing the evolution of the average *rationale density* and the average *decision density* (calculated based on the decision-containing sentences) per year.

The final set of analyses concerns the structure of a commit message. That is, what sentence category order developers prefer when elaborating their commit messages. To answer “*In what order do the categories mostly appear?*”, we visualize the distribution of the identified categories over the normalized positions of the sentences of the commit messages.

C. Rationale Extractors

In other work, we trained two binary BiLSTM classifiers on the OOM-Killer dataset (one for *Decision*-containing sentences and one for *Rationale*-containing sentences), and applied them on two different Linux modules: the *slob.c*¹ module of the *mm* component and the *button.c*² module of the *drivers/acpi* subproject, and five other open source projects. We then validated their generalizability to these new contexts.

III. CoMRAT DESIGN, IMPLEMENTATION AND USAGE

We combine our previous implementation of the rationale analyses of the OMM-Killer module with our Bi-LSTM binary classifiers to make the analyses available for any Github project. Having validated the generalizability of the extractors, we can apply them to other open source modules. In CoMRAT, we implement two analyzers that leverage the classifiers: a *Module Analyzer*, and a *Commit Message Analyzer*. The *Module Analyzer* allows the user to apply the analyses defined above for a specific module. The *Commit Message Analyzer* enables the user to enter a commit message and evaluate its quality. Our tool is packaged as a web application with two pages. The only prerequisites to its usage are a Github Username and a Github API token.

A. CoMRAT Module Analyzer

For the *Module Analyzer*, we implement the following workflow: 1) The user enters the Github API URL of a specific module, then clicks the *Start Module Analysis* button. 2) The analyzer downloads the commit messages through API calls. 3) The extracted messages are pre-processed similarly to the training dataset of the classifiers. 4) The tool applies the classifiers on the pre-processed commit messages.

¹<https://api.github.com/repos/torvalds/linux/commits?path=mm/slob.c>

²<https://api.github.com/repos/torvalds/linux/commits?path=drivers/acpi/button.c>

The output is a set of sentences labelled as *Decision* and/or *Rationale*. The tool also shows information about the distribution of the categories, and generates wordclouds from the most prominent words in each of the categories (without considering multi-labelled sentences). Note that when running the code locally, users can augment the built-in list of stop words of the wordcloud library³ with specific module-related keywords for better representations. Finally, the tool analyzes the labelled dataset: it computes the metrics and generates the figures. Optionally, the user can download the labelled dataset as a CSV file, or the generated figures as PDF files. Fig. 3 shows the output relating to the *slob.c* module.

B. CoMRAT Commit Message Analyzer

Here, the user inputs a commit message and clicks the “*Start Commit Analysis*” button. CoMRAT pre-processes the message, applies the classifiers, and outputs its sentences, labelled as *Decision* and/or *Rationale*. It also reports the *number_of_sentences*, the *rationale_density* and the *decision_density* of the commit message. If the *rationale_density* is below the threshold of 0.5, a warning message appears. If higher, a success message appears. Note that we are conducting further research to better determine this threshold value.

C. Limitations

CoMRAT is limited by Github’s API hourly rate limit (5000 authenticated requests per hour). This can be problematic for projects with longer commit histories, but can be overcome by upgrading to Github Enterprise. A second limitation is the non-optimal pre-processing we use [6]. In fact, due to improperly formatted commit messages, it is likely that the preprocessing may have been inaccurate or ineffective (e.g., not skipping blocks of raw source code from the message). A third limitation is that we only consider commit messages, and not additional sources for rationale, such as issues [8] or bug reports [9], [10]. This would require more downloaded data and further execution time.

D. Installation and local execution

CoMRAT is built with Python 3.11 and Streamlit⁴ and only needs specific libraries installable through `pip` in order to work. The source code is publicly available [11]. To install, users can download the source and, navigating to the *Rationale-Analyses-Tool* folder, create a virtual environment to install the required libraries and run the tool locally:

```
pip install -r requirements.txt
streamlit run tool.py
```

E. Potential Uses

The *Module Analyzer* could be used to create datasets of commits with rationale from open source Github projects, unearthing valuable rationale knowledge that could be reused in future development projects. In industrial settings, this

³https://amueller.github.io/word_cloud/

⁴<https://streamlit.io/>

The mm/slob.c module

Resulting dataset:

```
% preview of the labelled dataset
```

Number of commits: 146

Number of sentences: 833

Distribution

Decision only sentences: 233

Rationale only sentences: 162

Decision & Rationale sentences: 252

No Decision and No Rationale sentences: 186

Word Clouds



Rationale Presence

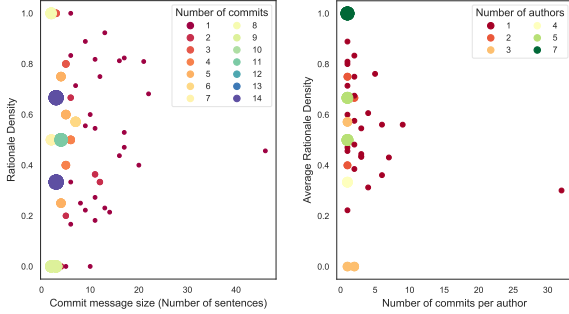
Total Number of commits: 146

Number of commits that contain rationale: 124

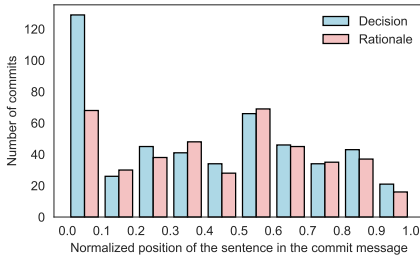
Rationale Percentage: 84.93%

Average Rationale Density: 0.56

Rationale Factors



Commit Message Structure



Rationale Evolution

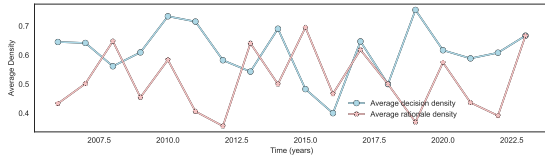


Fig. 3: CoMRAT analyses for the *slob.c* module

would enhance decision-making and productivity. Structural analysis may reveal structural patterns and identify common commit messages structures that could be used as guidelines or encouraged via automated tools. In development contexts, this would ensure well-structured commit messages, or suggest improvements before merging commits. For example, the *Commit Message Analyzer* allows the user to evaluate the quality of their commit message, e.g., by scoring their commit information against the module to which they are contributing to. This ensures that a desired level of rationale information is obtained, which is particularly valuable in open source projects for onboarding newcomers and retaining design information.

The *Module Analyzer* could also be used by researchers for in-depth investigation of the rationale in Github projects. Subsequent studies could reuse the research questions in Section II-B or include a comparison between modules in terms of rationale information. For instance, Table I shows the *rationale percentage* and the *average rationale density* for five Linux modules, suggesting stable rationale distribution and generalizability of prior findings from the OMM-Killer module [6]. The generated datasets could also support future research on how developers express their rationale, or regarding other factors influencing rationale in commit messages, e.g., the number of reviewers or the author's affiliation.

IV. PRELIMINARY EVALUATION

We conducted a preliminary evaluation of CoMRAT to assess usefulness (RQs.1,3) and usability (RQs.2,4). Complete materials / anonymized data are in a replication package [11].

A. Module Analyzer

a) *Usefulness (RQ.1)*: To get an expert evaluation, we asked a Software Engineering researcher with more than 10 years of experience to test the *Module Analyzer*. Then, we asked him to fill a form that contains a Likert scale about the usefulness and the usability of the tool overall, and six rating questions (with a scale of 1 to 5 stars) about the usefulness of each of the analyses. The researcher *Strongly Agreed* that the labelled dataset and the generated analyses are useful for researchers, with a minimum of four stars of five for the six analyses. He rated the Word Clouds analysis with 2/5 stars as he found the images were informative but could not be reused, and suggested frequency tables instead. He also described the kinds of research that might benefit from CoMRAT:

“Many types of MSR papers could benefit from such tools. I can think of using commit messages (with decisions and rationales) to enhance code review and bug localization. [...] Automatic bug fixes (or at least learning from past bug fixes) would also likely benefit.”

b) *Usability (RQ.2)*: The expert researcher *Agreed* that the tool is easy to use. We also report in the first three columns in Table I the execution time required to get the data through API calls, load and apply the classification models, execute the analyses, and generate the figures. The time was measured on a laptop with Intel Core i3 3000 GHz and 16 GB of memory. The results suggest that running CoMRAT locally is feasible.

Module	Number of Commits	Execution Time	Rationale Percentage (%)	Average Rationale Density (%)
mm	404	2m 24s	98.9	61.4
slob	146	34s	84.9	56.0
button	110	26s	90.9	61.9
FS	15	13s	86.7	46.5
migrate	666	8m 0s	94.3	63.0

TABLE I: Information about different Linux modules

B. Commit Message Analyzer

We conducted a preliminary user study to evaluate the usefulness and usability of the *Commit Message Analyzer*.

Study design. Our study includes a labelling task and a post-study questionnaire. The first author defined the Decision and Rationale categories and introduced the sentence-based labelling task using three commit examples. Then, the participants were asked to manually label the sentences of six other commit messages, three with rationale density higher than 0.5 and three lower. The labelling task was carried out to prompt participants to critically analyze and reflect on the messages. The first author then introduced the *Commit Message Analyzer* and asked the participants to apply it on the six commit messages. Participants then filled the questionnaire.

Our questionnaire comprises 29 English-language questions. It includes two multi-choice demographics questions about participants’ background (current position and experience with Git development), and four Likert scale questions per commit about the quantity and helpfulness of rationale information in the commit and about the tool’s ability to identify that information and provide helpful feedback messages. Finally, we include three Likert scale questions about the helpfulness of the labelling produced by the *Commit Message Analyzer* as well as its usability and impact overall.

Participants. To date, we have conducted the study with five participants with appropriate command of English to understand the commit messages. We use convenience sampling as we recruit participants from the university we are affiliated with. Our participants are all graduate students (40% Master’s and 60% PhD) with a minimum of two years with Git development. They frequently write commit messages as part of their academic activities and collaborative research projects.

Initial Study Results.

a) *Usefulness (RQ.3):* All users (100%) either *Agreed* or *Strongly Agreed* that the tool provides helpful labelling. For five out of six commit messages, at least 60% either *Agreed* or *Strongly Agreed* that the feedback message was helpful.

b) *Usability (RQ.4):* All users (100%) either *Agreed* or *Strongly Agreed* that the tool was easy to use, and that it encourages adding rationale to commits.

Evaluation Summary. From this initial study, we claim that CoMRAT is both *useful (RQs.1,3)* and *usable (RQs.2,4)* for researchers and developers.

V. RELATED WORK

Automatically mining developers rationale has recently attracted research interest. Alkadhi et al. experimented with machine learning models to extract rationale elements (decision, issue, alternative, pro-argument, con-argument) from developers’ chat messages [2]. Rogers et al. [10] used linguistic features while Lester et al. [9] experimented with evolutionary algorithms to optimize the feature sets to improve rationale extraction from Chrome bug reports. Sharma et al. tried to extract rationale from Python Enhancement Proposals using heuristics [3]. Kleebaum et al. proposed automatic rationale classification from Jira issues and commit messages using machine learning [12]. Zhao et al. mined design rationales from developers discussions in open-source issue logs using Large Language Models and heuristics. They also investigated the usefulness of the extracted information for automated program repair [8]. These prior works differ from us as they only focus on the extraction and do not analyze developer’s rationale. Also, none of this prior research has examined the rationale in open-source commit messages.

Researchers have also studied commit message quality [4], [5]. Tian et al. define what constitutes a “good” commit message [4] by studying five open-source projects. They found that it should summarize *what* was changed, and describe *why* those changes are needed, and proposed a good-message identification tool. Li et al. [5] continued this research by considering link contents in addition to the commit messages while training classifiers for the automatic identification of good commit messages. They also studied the commit quality evolution over time, and the correlation between defect proneness and the quality of the commit message.

Similar to us, these researchers considered rationale information in the commit messages of open source projects and proposed analyses that include the temporal evolution aspect and the factors that influence rationale. Different from us, their analyses only consider the evolution of the existence of *what* (decision) and *why* (rationale) information over time, while we study the evolution of their quantities. They also only focus on the correlation with defect proneness while we study factors that might influence rationale. Also, they do not consider the structure (order) in which the *what* and *why* information appear. Finally, previous work does not provide their classifiers as a tool applicable on any Github module as we do.

VI. CONCLUSION

We present CoMRAT, a tool that a) provides researchers with labelled datasets and insights about rationale information in any Github project and b) assists developers in writing rationale-aware commits. Preliminary evaluation indicates CoMRAT’s usefulness and usability for researchers and developers. In the future, we plan to package the *Commit Message Analyzer* as a Github Bot to be integrated directly into the development process, by scoring the rationale information in pull request commits.

REFERENCES

- [1] M. Dhaouadi, B. Oakes, and M. Famelis, “End-to-end rationale reconstruction,” in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.
- [2] R. Alkadhi, J. O. Johanssen, E. Guzman, and B. Bruegge, “React: An approach for capturing rationale in chat messages,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 175–180.
- [3] P. N. Sharma, B. T. R. Savarimuthu, and N. Stanger, “Extracting rationale for open source software development decisions — a study of Python email archives,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Madrid, ES: IEEE, May 2021, pp. 1008–1019.
- [4] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, “What makes a good commit message?” in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2389–2401. [Online]. Available: <https://doi.org/10.1145/3510003.3510205>
- [5] J. Li and I. Ahmed, “Commit message matters: Investigating impact and evolution of commit message quality,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 806–817.
- [6] M. Dhaouadi, B. Oakes, and M. Famelis, “Rationale dataset and analysis for the commit messages of the Linux kernel out-of-memory killer,” in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, 2024, pp. 415–425.
- [7] R. Love, *Linux kernel development*. Pearson Education, 2010.
- [8] J. Zhao, Z. Yang, L. Zhang, X. Lian, D. Yang, and X. Tan, “DRMiner: Extracting latent design rationale from Jira issue logs,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 468–480.
- [9] M. Lester and J. E. Burge, “Identifying design rationale using ant colony optimization,” in *Design Computing and Cognition’18*. Springer, 2019, pp. 537–554.
- [10] B. Rogers, Y. Qiao, J. Gung, T. Mathur, and J. E. Burge, “Using text mining techniques to extract rationale from existing documentation,” in *Design computing and cognition’14*. Springer, 2015, pp. 457–474.
- [11] Authors, “Replication package,” <https://zenodo.org/records/14176549>.
- [12] A. Kleebaum, B. Paech, J. O. Johanssen, and B. Bruegge, “Continuous rationale identification in issue tracking and version control systems,” 2021.