Toward Automated Rationale Extraction from Software Artifacts

Mehedi Sun
Department of Computer Science
William and Mary
Virginia, USA
msun12@wm.edu

***Abstract*—Developers spend approximately 60% of their time understanding the source code. Developers spend this much time because, when recording code changes, the emphasis is often on what has changed rather than why the change was made. While tools like Git[1] facilitate recording "what" has changed very easily, they fail to provide support capturing the rationale behind code changes. This research investigates the potential of automated techniques and large language models (LLMs) to systematically extract and manage the answer to those "whys" questions which we call *rationale* from software artifacts such as commit messages, issue logs, discussions, etc. This paper explores the capabilities of prompts and multi-model architectures and assesses the extent to which rationale can be effectively extracted from documentation linked to source code. The evaluation shows this project scored 63%, and 67% in precision, and recall scores respectively while extracting rationales texts from different sources. Additionally, Chain-of-Thought Prompting outperformed Zero-Shot and One-Shot prompts in constructing rationales.**

***Index Terms*—PROJECT AI4SE**

I. Introduction

In the dynamic field of software development, understanding the rationale behind code changes is as crucial. Developers frequently dedicate a significant around 60% of their time to comprehending existing codebases. [1], [2]. This process is essential for maintenance, debugging, and the implementation of new features, etc. However, the focus often skews towards understanding what changes were made, while the underlying why—the reasoning behind these changes—remains elusive.

Modern version control systems, such as Git, excel at recording the what, providing detailed logs of code modifications across time. While committing the code changes to Git, developers should write the changes along with the reason for that code change [3]. However, researchers state that about 40% of commit messages in certain scenario contain rationale for code changes [4].

Another important reason is manual capturing of rationales makes it harder to comply with industry practices which slows down developer and hinders their workflow. Like that the underlying reason for code change(CC) is eroded to unstructured artifacts such as commit messages, issue trackers, code reviews, and informal discussions. These artifacts are seldom consistent in their quality and completeness, creating a gap that hinders both human understanding and automated analysis.

The absence of structured rationale documentation can lead to reduced efficiency, increased risk of introducing bugs, and challenges in onboarding new team members. Safwan et. al.[5] identified 15 components of the rationale of code commits that are differently needed, found, and recorded. Although there are tools that support developers using manual annotation[6] as well as some retrival-based tools none of those can produce quality rationale.

The advent of large language models (LLMs) and advanced natural language processing (NLP) techniques presents an opportunity to bridge this gap. These technologies have demonstrated remarkable potential in extracting meaningful insights from unstructured data across domains. Software engineering offers a potential for systematically extracting and managing rationale information from diverse software artifacts.

This research explores the feasibility of employing automated techniques and LLMs to capture the why behind code changes. Specifically, it investigates the use of prompts and multi-model architectures to extract rationale from sources such as commit messages, issue logs, and associated discussions. The process is divided into smaller tasks i.e. code change(CC) summarizer, relevant text identifier based on different sources, and construction of rationale based on this related text. This paper aims to address key challenges in rationale extraction, including the lack of standardized benchmarks, variable artifact quality, and integration into existing development workflows. Overall, this project scored 63%, and 67% in precision, and recall respectively while extracting rationales texts from different sources.

This research aims to address the following research questions:

1) **Are commit messages sufficient to provide the rationale behind code changes?**
   - Understanding how often the commit message contains the rationale of code change and does it contributes to constructing the rationale of source code change.
2) **Which prompting techniques and data combinations yield the best results for rationale extraction?**
   - Comparing zero-shot, few-shot, and chain-of-thought prompting methods with varying inputs (e.g., diffs,

[1]https://git-scm.com/

commit messages, issue descriptions, and comments).

II. Related Work

For over 30 years, researchers have been exploring Design Rationale [7]. Early efforts looked at using code annotations to capture design rationales, which helped developers better understand the decisions made in the Software Development Life Cycle (SDLC) process. However, these approaches didn't significantly improve maintenance efficiency because they weren't practical or scalable during active software development. Tools like SEURAT [8] were developed to support maintenance tasks by presenting relevant design rationale and checking new rationale against existing documentation. Despite decades of work, there's still a lot of uncertainty about how effective and practical it is to capture rationale in real-world software development. While most of the research has centered on design or architectural rationales [9], this paper focuses specifically on rationale at the source code level.

Researchers have identified rationale information in various sources and found it to be highly valuable [5]. This information can be located in technical artifacts such as issue reports [10], [11], [12], [13], code commits [14], and code comments [15]. Additionally, the rationale can also be derived from non-technical sources like Python archive emails [16], IRC messages [17], and user reviews [18].

Over time, there has been a noticeable shift in how researchers approach capturing rationales. In earlier research, pattern-based techniques were commonly used [19], [20], [21], relying on vocabulary information, document features, and semantic analysis to identify rationales. Machine learning models were also introduced to support rationale capture [12], [22], with researchers developing diverse feature sets and focusing heavily on optimizing these features for better performance [23], [24].

Recently, language-based techniques have gained prominence in deriving rationales from various sources. Zhao et al. [11] utilized transformer-based models such as BERT [25] to extract polarity vectors of rationale-contributing comments from issue reports. They also employed LLama [26] to identify relationships between sentences and construct rationales. In this paper, I also leverage large language models to capture and construct rationales from artifacts.

III. Study Design

The firgure1 shows the overall design of this study. This study is composed of three distinct tasks:

A. Data Preparation

Given the lack of an existing dataset specific to my study, I created a customized dataset by analyzing the commit history of the Spring Framework[2]. The Spring Framework transitioned from using Jira[3] as its Issue Tracking System (ITS) to GitHub in 2019. To collect the necessary commit data, I utilized the GitHub REST API to fetch all commits from the *main* branch of the repository.

[2]https://github.com/spring-projects/spring-framework
[3]https://www.atlassian.com/software/jira

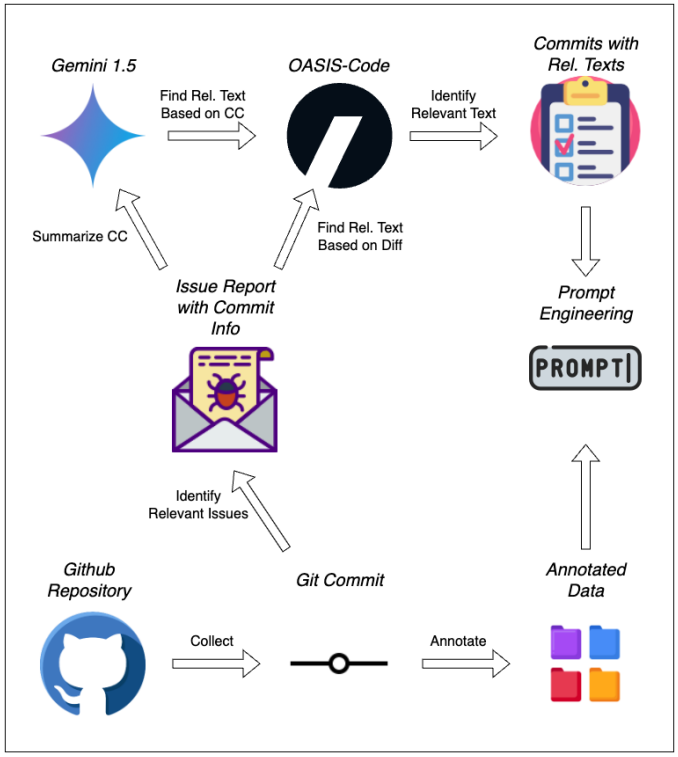

Fig. 1. Study Design

1) Code Commit Collection: Since this study is exploratory and focused on a specific aspect of code changes, I narrowed my attention to two types of changes: Condition and Iteration. This targeted approach helped focus on more relevant changes for the analysis. I disregarded those commits in which no condition(if, else, switch) or iteration (for, while) has been changed.

Despite the specific focus, I still had over 3,000 commits to process. To refine the dataset, I applied additional filtering steps. First, I excluded commits that involved changes to multiple Java files. From the remaining commits, I filtered out those where the change involved more than 10 lines of code. While counting the line changes, I excluded code comments, although I kept them in the dataset since they are an important source of rationale in code changes[15]. For each commit, I collected the commit message (both subject and body), author information, commit date, and the diff of the commit. To expedite the data collection process, I used multithreading, which significantly improved the efficiency and speed of the operation.

2) Linking Commit to Issue: The next step was to link each commit to its corresponding issue, which provided the context for the code changes. The paper hypothesizes that, before implementing code, developers often discuss various aspects of a problem in the issue report. This discussion can be leveraged to extract the rationale behind the code change. Initially, I used regular expressions (regex) to identify issue links in the commit messages. Although this approach seemed

straightforward, it presented two main challenges:

- **Irregular Patterns:** During the process, I encountered irregular patterns in many issue numbers that were referenced in commit messages. To address this, I reviewed a wide variety of commit messages to identify and account for all possible variations in the issue number format.
- **Migrated Issues:** Before 2019, the Spring Framework used Jira as its issue-tracking system. When they transitioned to GitHub's issue tracker, they migrated all of their issues from Jira to GitHub. This migration led to a mismatch between the Jira issue numbers and the corresponding GitHub issue numbers referenced in commit messages. To resolve this, I downloaded all 32,000 issues from the Spring Framework repository and established a mapping between the Jira and GitHub issue numbers.

Finally, I added all the comments from the linked issues to our dataset. To ensure the quality and relevance of the comments, I applied six types of filtering. First, I excluded any comments exceeding 2048 tokens in length. Second, I removed comments containing **Log Messages** or **Stack Traces**, as these do not contribute meaningful rationale for the code changes. Lastly, I parsed the comments into paragraphs to facilitate easier identification of the portions discussing the rationale behind the code changes. I also removed the auto-generated portions of comments, developer mentions, and replies since replies are already being considered in previous texts.

3) Data Annotation: To validate the retriever model and rationale construction model's performance, I took a small sample of the dataset and annotated it myself to achieve four main goals:

- Evaluation of Rationale Relevant Text Retriever model
- Evaluation of generated rationale quality
- Experiment with different and the needed number of related texts I have to identify
- Check if a code change summarizer is needed to construct or not

For data annotation, I categorized text from multiple software artifacts to identify whether they contain rationales for the implemented code. These artifacts include issue reports, pull requests, code comments, and commit messages. For each artifact, I labeled text as either containing rationale (e.g., IR_RATIONALE_PRES for issue report rationale presence) or not containing rationale (e.g., IR_RATIONALE_ABS for absence). Commit messages were further divided into titles (COM_T_RATIONALE_PRES_ABS) and bodies (COM_B_RATIONALE_PRES_ABS). Additionally, a separate label, *RATIONALE*, was assigned to sentences explicitly explaining the reasons behind code changes. This systematic labeling helps in accurately identifying rationale-containing text across different sources and works as a ground truth dataset.

B. Data Availability

The codebase utilized for conducting this research is publicly available in the GitHub repository at [4].

C. Rationale Relevant Text Identification

1) CC Summarizer: To identify which texts are relevant for extracting rationale, I used two data sources: the diff of the code commit, which highlights the added and deleted lines, and a code change summary generated by an LLM, *Gemini-1.5-Flash*. *Gemini-1.5-Flash* is the latest model released by Google, and I customized its system instructions to optimize its performance as a code change summarizer. I didn't find the training data of Gemini so I couldn't verify the data leakage here. However since I am using this model as a generative summarizer to work on rationale identification and no such dataset is available on rationale data the way I intended, it is safe to presume that this model doesn't suffer from data leakage.

2) Rationale Related Text Search: I used an additional language model from Hugging Face[5], named *OASIS-1.3B*. OASIS (*Optimized Augmentation Strategy for Improved Code Search*) is a cutting-edge code embedding model developed by Kwaipilot. It integrates advanced, proprietary techniques such as repository-level program analysis, the OASIS-instruct data synthesis algorithm, and a specialized fusion loss function, establishing new standards for efficiency and accuracy in code search tasks.

OASIS is particularly well-suited for scenarios that demand semantic understanding and retrieval of code snippets in diverse programming contexts. Since I required a model capable of interpreting code in the context of natural language, OASIS proved highly effective for this purpose. Upon comparing its performance against the ground truth data, the model exceeded expectations. Two types of experiments were conducted to evaluate its efficacy: Top-K Retriever and Source-Based Retriever. These experiments highlighted the model's capability to retrieve relevant code snippets and understand their context accurately.

- *Top-K Retriever*: In this experiment, I retrieved the top K relevant texts from issue reports and evaluated the model's performance. The value of K was varied from 1 to 10 to analyze its impact on the retrieval accuracy. This approach helped assess how effectively the model identified the most relevant rationale-containing texts for a given code change. It also can help identify the threshold value of relevant texts' similarity score.
- *Source-Based Retriever*: This retriever utilized either the diff of the code commit or the code change summary generated by *Gemini-1.5-Flash* to identify related texts. I observed minimal performance differences between using diffs or summaries, as the OASIS model effectively leveraged code semantics, whether presented in plain text

[4]https://github.com/mh-sun/code_rationale_extraction
[5]https://huggingface.co/

or source code, to understand the context and perform retrieval tasks accurately.

D. Rationale Construction

The process of prompt engineering in this study focused on generating high-quality input prompts for a language model to extract rationale for code changes effectively. To achieve this, I used manually annotated ground truth (GT) data and identified relevant texts associated with specific commit messages. The approach can be summarized as follows:

1) Prompt Settings: I used three types of prompt setting, Zero Shot, One Shot & Chain-of-Thought. For each setting, I used three experiments to identify the importance of the commit message and issue comments in extracting rationale. In the first experiment, I passed only diff to prompt LLM to construct rationale, in the second setting I used diff info + commit message (Commit title and commit body) and lastly, I used all information, i.e. diff, commit message, and all related texts. This leaves me with 9 Experiments. However, I had to remove the steps that the model uses in a chain-of-thought prompt response.

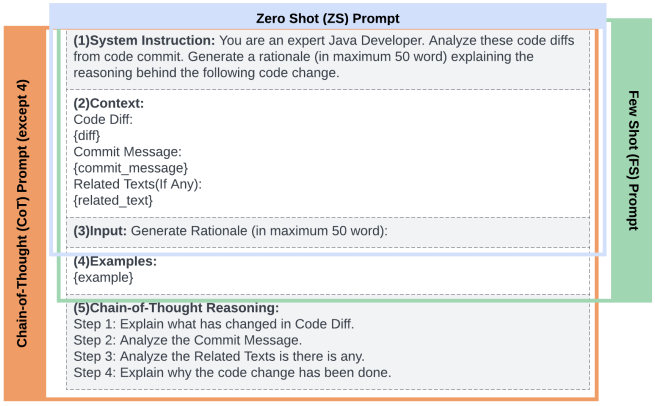

Fig. 2. Prompt Template for All Prompt Setting

2) Model Configuration: For rationale generation, the prompts were used as input to a state-of-the-art language model, *Meta-Llama-3-8B-Instruct*, fine-tuned for instruction-following tasks. The system instruction and tokenization process were optimized for code understanding, ensuring the model could interpret the structured inputs effectively.

3) Evaluation: The generated rationales were compared with manually annotated ground truth rationales to evaluate the model's performance. BERTScore has been used to measure the semantic similarity of the two rationales. The study experimented with variations in prompt structure to determine the optimal configuration for accurate rationale extraction. Since this paper wants to capture the original rationale of the code change, instead of the F1 Score, I will focus on the Recall value that we obtain from BERTScore.

IV. Results

This section presents and discusses the experimental results obtained thus far.

TABLE I
Rationale Related Text Retriever Evaluation

k	source	tp	fp	fn	precision	recall	f1	f2	f3	f4
1	diff	12	7	26	0.63	0.32	0.42	0.35	0.33	0.33
1	summary	12	7	26	0.63	0.32	0.42	0.35	0.33	0.33
2	diff	17	21	21	0.45	0.45	0.45	0.45	0.45	0.45
2	summary	16	22	22	0.42	0.42	0.42	0.42	0.42	0.42
3	diff	19	31	20	0.38	0.49	0.43	0.46	0.47	0.48
3	summary	19	31	20	0.38	0.49	0.43	0.46	0.47	0.48
4	diff	23	39	16	0.37	0.59	0.46	0.53	0.56	0.57
4	summary	21	41	18	0.34	0.54	0.42	0.48	0.51	0.52
5	diff	24	48	15	0.33	0.62	0.43	0.53	0.57	0.59
5	summary	23	49	16	0.32	0.59	0.41	0.50	0.54	0.56
6	diff	25	57	14	0.30	0.64	0.41	0.53	0.58	0.60
6	summary	25	57	14	0.30	0.64	0.41	0.53	0.58	0.60
7	diff	26	65	13	0.29	0.67	0.40	0.53	0.59	0.62
7	summary	26	65	13	0.29	0.67	0.40	0.53	0.59	0.62
8	diff	26	73	13	0.26	0.67	0.38	0.51	0.58	0.61
8	summary	26	73	13	0.26	0.67	0.38	0.51	0.58	0.61
9	diff	26	81	13	0.24	0.67	0.36	0.49	0.57	0.60
9	summary	26	81	13	0.24	0.67	0.36	0.49	0.57	0.60
10	diff	26	89	13	0.23	0.67	0.34	0.48	0.56	0.60
10	summary	26	89	13	0.23	0.67	0.34	0.48	0.56	0.60

A. Data Annotation

For data annotation, I utilized the Hypothesis tool[6]. A total of 55 annotations were made among which 35 were rationale-related text and 20 annotations were actual rationale. This is calculated across 20 commits, covering approximately 158 text paragraphs and 484 sentences. Among these commits, 54.5% contained rationale texts, while 36.4% lacked any rationale. The remaining 9.1% of commit messages included some rationale-related text but were not entirely self-explanatory. Overall, approximately 63.4% of the code commits contained rationale sentences in their commit messages.

> ***Finding 1***
>
> Around 63.4% code commits have rationales in their commit messages.

Among those annotated commits, there were 158 paragraphs and among them, 35 annotations were rationale-related which means 22% of the issue commit and commit log have rationales which proves how hard it can be to identify rationale sentences manually.

B. Retrival of Rationale Related

This paper includes two types of retriever experiments: Top-k-based retriever and Source-based retriever.

1) TopK Retrival: Top-K Retrieval experiments involve retrieving the top K most likely relevant texts from issue reports and commit messages, followed by an evaluation of the model's performance. I conducted experiments with K ranging from 1 to 10 and observed that for K=1, the model achieved the highest precision score. However, the model exhibited conservative behavior, resulting in a lower recall. Table I presents the performance of these experiments. Depending on the specific requirements, different K values may be chosen to best serve the intended purpose.

[6]https://hypothes.is/

Finding 2

Depending on our needs, we can select different K values for the Top-K Retriever. If high precision is required, K=1 is the optimal choice. Conversely, for better recall, K=7 is more suitable. For a balanced trade-off between recall and precision, K=4 offers a good compromise.

From the table, we can observe how different values of K produced varying results. For example, precision is highest when K=1, while recall reaches its maximum at K=7. When precision and recall are given equal weight, the highest F1 score is achieved at K=4.

However, there may be cases where developers prioritize recall over precision. To account for this, I calculated F2, F3, and F4 scores using the equation in 1, which places greater emphasis on recall. This approach is useful when the goal is to capture all rationale texts, even if it means accepting a slightly lower precision. Based on this analysis, K=7 produces the highest F3 and F4 scores, making it the most suitable choice in scenarios where recall is highly valued.

$$F_\beta = \frac{1+\beta^2}{\frac{\beta^2}{\text{Recall}} + \frac{1}{\text{Precision}}} \quad (1)$$

2) Source-Based Retrival: I also conducted experiments to determine whether rationale-related text can be extracted solely using diffs and whether summarizing diffs would aid in this process. The results showed very little difference between these approaches, with both yielding similar outcomes in most cases. However, slight variations were observed for K=2,4, and 5. This finding motivates the use of diffs to retrieve rationale texts, as it eliminates the need for a summarization model, which can be computationally expensive, time-consuming, and resource-intensive.

Finding 3

Code commit changes (diffs) can be effectively used to identify rationale-related text from various sources, eliminating the need to rely on code change summaries.

C. Rationale Construction

In this task, I experimented with how prompt engineering can be used to construct rationales that capture the contextual meaning of the original rationale. The effectiveness of these rationales can be measured by Recall. To compare the ground truth rationale with the generated rationale, I used BERTScore, which provides Recall, Precision, and F1 scores based on the reference and candidate texts. Since the goal is to maximize the capture of the original rationale, the focus is on optimizing the Recall value of BERTScore.

TABLE II
RATIONALE RELATED TEXT RETRIEVER EVALUATION

		MEAN	*MEDIAN*	*STD*
Zero Shot	**ZS_w_diff_0**	0.512	0.597	0.183
	ZS_w_diff_commit_0	0.548	0.615	0.190
	ZS_w_diff_commit_rel_text_0	0.541	0.605	0.185
Few-shot	**FS_w_diff_k**	0.493	0.572	0.176
	FS_w_diff_commit_k	0.500	0.535	0.185
	FS_w_diff_commit_rel_text_k	0.537	0.625	0.200
Chain-of-thought	**CoT_w_diff_0**	0.517	0.577	0.181
	CoT_w_diff_commit_0	0.549	0.598	0.169
	CoT_w_diff_commit_rel_text_0	0.572	0.645	0.189

Table II shows the average BERTScore(Recall) of different prompts. I experimented with three Prompt settings and three Prompt Versions:

- Prompt_setting - ZS: Zero Shot, FS: Few Shot, CoT: Chain-Of-Thought
- Prompt_version - w_diff: among diff(code change), commit message (commit title + commit body) & related texts I am only using diff in prompt, w_diff_commit: I am using diff as well as commit message, w_diff_commit_rel_text: I am passing diff, commit message & related texts
- ending_number - the ending number of each column indicates several examples used in prompt. For Few Shot I only experimented with one example.

The table shows how Chain-of-Thought was able to capture most of the original information than other prompts. Even though in one shot prompt we are providing an example.

Research Question 2

Chain-Of-Thought Performed better in capturing original rationale information than Zero shot and One shot Prompts.

In a local comparison between Zero-Shot, One-Shot, and Chain-of-Thought prompts, adding commit information increased BERTScore. Specifically, adding commit information led to improvements of 7%, 1.5%, and 6.2% in Zero-Shot, One-Shot, and Chain-of-Thought prompts, respectively. However, the same trend was not observed in the related texts, where BERTScore decreased in the Zero-Shot scenario. We also noticed from our annotated data, that 63% times commit message contains rationale information. So commit message plays a vital role while contrasting code rationale.

Research Question 1

Commit messages play a vital role in understanding and contrasting code rationale.

Although Related texts increased by 7.04% and 4.2% in One Shot Prompt and Chain of Thought, We can't say the same for Zero Shot Prompt. Besides Related texts also consist of

commit messages so we can't confirm about comments from the issue report's importance yet from these experiments.

V. Threats to Validity

- **External Validity:** The datasets of issues and commits were collected from the Spring Framework, which may not be representative of all software projects. Additionally, for experimental ease, I filtered out long comments and code commits that included multiple Java file changes, which limits the generalizability of the findings.
- **Internal Validity:** A primary threat to internal validity arises from the manual annotation and observation of rationales, which may introduce biases or inconsistencies.
- **Conclusion Validity:** A small evaluation dataset can undermine the statistical power of the analysis, leading to potential inaccuracies in drawing conclusions or identifying significant patterns.

Addressing these threats requires broader dataset coverage, improved annotation protocols, and the development of more holistic evaluation frameworks, which I plan to pursue in future work.

VI. Discussion

The findings of this study show the complexity and necessity of extracting rationale in software development. Developers invest significant time in understanding source code changes, yet the rationale behind these changes is often inadequately documented. This study demonstrates that large language models (LLMs) and natural language processing (NLP) techniques, can partially address this gap.

A. Human-in-the-Loop Systems

Automating the extraction of rationale introduces the risk of over-reliance on AI-generated outputs, which may not always capture the full context or nuances of a developer's decisions. To mitigate this, a hybrid approach that combines automation with human oversight could strike a balance between efficiency and accuracy. This is particularly important in critical or high-stakes software environments where errors in understanding could have significant consequences.

Future research could focus on developing systems that actively assist developers by identifying and prompting them to document important rationales. For this to work effectively, we would need metrics to assess the complexity and necessity of explaining certain code changes. These metrics could help determine when a rationale is particularly valuable to record. Automated tools could then suggest options for rationale documentation, allowing developers to review and select the most relevant explanations. This approach would streamline the documentation process, save time, and ensure that important rationale is preserved without burdening developers unnecessarily.

B. Integration with Development Tools

Embedding rationale extraction into integrated development environments (IDEs) or version control systems can significantly enhance adoption and usability for developers. By integrating these capabilities, the tools can actively suggest more informative commit messages and better-structured issue logs, encouraging developers to document their reasoning more easily and effectively. This seamless integration would not only reduce the cognitive load on developers but also improve the overall quality of project documentation, ensuring that rationales are preserved consistently throughout the development lifecycle.

VII. Conclusion

This research highlights the potential of LLMs and prompt engineering for addressing the long-standing challenge of capturing rationale in software development. By systematically evaluating different techniques and configurations, this study provides actionable insights for improving rationale extraction and management. While the proposed methods show promise, broader validation and refinement are necessary to integrate these techniques into real-world development workflows effectively. Future research should aim to develop standardized benchmarks and extend the approach to diverse programming languages and project types to further advance the field.

References

[1] D. Oliveira, "Recommending code understandability improvements based on code reviews," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2021, pp. 131–132.

[2] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 951–976, 2017.

[3] 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.

[4] M. Dhaouadi, "A data set of extracted rationale from linux kernel commit messages," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 2187–2188.

[5] K. A. Safwan and F. Servant, "Decomposing the rationale of code commits: the software developer's perspective," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 397–408.

[6] M. Adeli, N. Nelson, S. Chattopadhyay, H. Coffey, A. Henley, and A. Sarma, "Supporting code comprehension via annotations: Right information at the right time and place," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020, pp. 1–10.

[7] J. Lee and K.-Y. Lai, "What's in design rationale?" *Human-Computer Interaction*, vol. 6, no. 3, p. 251–280, Sep 1991.

[8] J. E. Burge and D. C. Brown, "Journal of software maintenance and evolution: Research and practice research rationale support for maintenance of large scale systems." [Online]. Available: https://api.semanticscholar.org/CorpusID:1225044

[9] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *Journal of systems and software*, vol. 79, no. 12, pp. 1792–1804, 2006.

[10] A. Kleebaum, B. Paech, J. O. Johanssen, and B. Bruegge, "Continuous rationale visualization," in *2021 Working Conference on Software Visualization (VISSOFT)*. IEEE, 2021, pp. 33–43.

[11] 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.

[12] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, "Automatic extraction of design decisions from issue management systems: a machine learning based approach," in *Software Architecture: 11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017, Proceedings 11*. Springer, 2017, pp. 138–154.

[13] 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.

[14] M. Dhaouadi, B. J. Oakes, and M. Famelis, "Towards understanding and analyzing rationale in commit messages using a knowledge graph approach," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2023, pp. 622–630.

[15] L. Pascarella and A. Bacchelli, "Classifying code comments in java open-source software systems," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 227–237.

[16] 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)*. IEEE, 2021, pp. 1008–1019.

[17] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, "How do developers discuss rationale?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 357–369.

[18] Z. Kurtanović and W. Maalej, "Mining user rationale from software reviews," in *2017 IEEE 25th international requirements engineering conference (RE)*. IEEE, 2017, pp. 61–70.

[19] C. López, V. Codocedo, H. Astudillo, and L. M. Cysneiros, "Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach," *Science of Computer Programming*, vol. 77, no. 1, pp. 66–80, 2012.

[20] Y. Liang, Y. Liu, C. K. Kwong, and W. B. Lee, "Learning the "whys": Discovering design rationale using text mining—an algorithm perspective," *Computer-Aided Design*, vol. 44, no. 10, pp. 916–930, 2012.

[21] R. McCall, "Using argumentative, semantic grammar for capture of design rationale," in *Design Computing and Cognition'18*. Springer, 2019, pp. 519–535.

[22] A. De Lucia, F. Fasano, C. Grieco, and G. Tortora, "Recovering design rationale from email repositories," in *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 543–546.

[23] M. Lester, M. Guerrero, and J. Burge, "Using evolutionary algorithms to select text features for mining design rationale," *Ai Edam*, vol. 34, no. 2, pp. 132–146, 2020.

[24] M. Lester and J. E. Burge, "Identifying design rationale using ant colony optimization," in *Design Computing and Cognition'18*. Springer, 2019, pp. 537–554.

[25] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of naacL-HLT*, vol. 1. Minneapolis, Minnesota, 2019, p. 2.

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.