How to Refactor this Code? An Exploratory Study on Developer-ChatGPT Refactoring Conversations

Eman Abdullah AlOmar
Stevens Institute of Technology
Hoboken, New Jersey, USA
ealomar@stevens.edu

Anushkrishna Venkatakrishnan
Rochester Institute of Technology
Rochester, New York, USA
av3278@rit.edu

Mohamed Wiem Mkaouer
University of Michigan-Flint
Flint, Michigan, USA
mmkaouer@umich.edu

Christian D. Newman
Rochester Institute of Technology
Rochester, New York, USA
cnewman@se.rit.edu

Ali Ouni
ETS Montreal, University of Quebec
Montreal, Quebec, Canada
ali.ouni@etsmtl.ca

ABSTRACT

Large Language Models (LLMs), like ChatGPT, have gained widespread popularity and usage in various software engineering tasks, including refactoring, testing, code review, and program comprehension. Despite recent studies delving into refactoring documentation in commit messages, issues, and code review, little is known about how developers articulate their refactoring needs when interacting with ChatGPT. In this paper, our goal is to explore conversations between developers and ChatGPT related to refactoring to better understand how developers identify areas for improvement in code and how ChatGPT addresses developers' needs. Our approach relies on text mining refactoring-related conversations from 17,913 ChatGPT prompts and responses, and investigating developers' explicit refactoring intention. Our results reveal that (1) developer-ChatGPT conversations commonly involve generic and specific terms/phrases; (2) developers often make generic refactoring requests, while ChatGPT typically includes the refactoring intention; and (3) various learning settings when prompting ChatGPT in the context of refactoring. We envision that our findings contribute to a broader understanding of the collaboration between developers and AI models, in the context of code refactoring, with implications for model improvement, tool development, and best practices in software engineering.

CCS CONCEPTS

• **Software Engineering → Software Quality**; *Refactoring*.

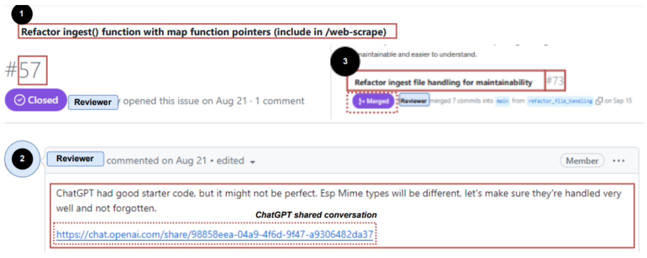
KEYWORDS

Refactoring documentation, ChatGPT, software quality, mining software repositories

ACM Reference Format:
Eman Abdullah AlOmar, Anushkrishna Venkatakrishnan, Mohamed Wiem Mkaouer, Christian D. Newman, and Ali Ouni. 2024. How to Refactor this Code? An Exploratory Study on Developer-ChatGPT Refactoring Conversations. In *Proceedings of MSR '24: Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/00.0000/0000000.0000000

1 INTRODUCTION

Artificial intelligence has been reshaping educational and industrial landscapes, and Large Language Models (LLMs) are emerging as the main driving force behind this revolution. The ability to harness massive amounts of information in multiple modalities allows LLMs to perform various tasks that would typically require human intervention [3, 4, 15, 20, 20, 22, 26, 27, 30, 34, 45].



Refactor ingest() function with map function pointers (include in /web-scrape)

#57

Reviewer opened this issue on Aug 21 · 1 comment

maintainable and easier to understand.

Refactor ingest file handling for maintainability #73

Reviewer merged 7 commits into main from refactor_file_handling on Sep 15

Reviewer commented on Aug 21 • edited

ChatGPT had good starter code, but it might not be perfect. Esp Mime types will be different, let's make sure they're handled very well and not forgotten.

ChatGPT shared conversation

https://chat.openai.com/share/98858eea-04a9-4f6d-9f47-a9306482da37

Figure 1: Example of a ChatGPT conversation in the context of GitHub issue about refactoring [1].

Learning from software repositories has opened up a myriad of LLM applications in software development tasks, such as code search [37], code quality [5, 38, 39], repair [19, 33, 40, 44], program comprehension [24], generation [14, 17], completion [28], and translation [21]. The impressive performance of these LLMs, in general, and ChatGPT, in particular, has rapidly increased its popularity within the development community. According to a recent GitHub survey with 500 US-based developers [2], 92% stated that their workflow has already integrated AI tools, and 70% found these tools to increase their productivity. When asked about the reason behind this fast adoption, respondents said that they believe these AI tools will improve the quality of their code while meeting existing performance standards with fewer production-level incidents.

Although developers can assess the performance of generated code from a functional perspective [17], little is known about how the model would react to refactoring requests. By definition, refactoring is the practice of improving the internal code structure, without altering its external behavior [18]. Given the subjective nature of refactoring, where there could be multiple equivalent solutions for a given input situation, it is interesting to see how LLMs would react to refactoring requests and what quality attributes they would target when optimizing the code.

The goal of this paper is to identify which of the various quality attributes, presented in the refactoring literature, are sought by developers when they request ChatGPT to refactor their code. Similarly, when developers request the refactoring of a given input code, we want to extract the quality attributes that ChatGPT reports optimizing when performing the refactoring. An illustrative example is shown in Figure 1: An issue was opened to update the `ingest()` method. When querying ChatGPT, the model has provided an updated version of the code, while explicitly stating that the refactoring was intended to improve *maintainability*. This can also be seen in the final title of the pull request that later integrated the modified code in production. Based on this example, we can see that *maintainability* was one of the quality attributes that ChatGPT considers for code optimization, and so we want to extract all other quality attributes to better understand the refactoring strategies adopted by the model.

2 STUDY DESIGN

Our study uses the DevGPT [42] dataset, which contains a wide range of information for open-source projects, such as code files, commits, issues, Hacker News, pull requests, and discussions. All, except Hacker News, utilize GitHub as their version control repository when demonstrating developer interactions with ChatGPT. Figure 2 depicts a general overview of our experimental setup. In the following subsections, we elaborate on the activities involved in the process.

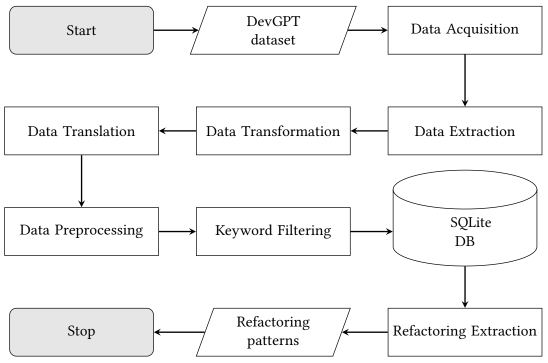


Figure 2: Overview of our experiment design.

2.1 Dataset Construction

To obtain data from various sources, we follow these steps:

Step #1: Data Acquisition: The data were initially gathered from the DevGPT dataset. The dataset consists of multiple JSON files organized into snapshots.

Step #2: Data Extraction: JSON files were extracted and consolidated into separate groups according to their source type.

Step #3: Data Transformation: The JSON files were processed to transform them into a structured relational tabular form and loaded into the database.

Step #4: Data Translation: This step involves leveraging the deep translator library's Google Translator[1] to facilitate the translation of non-English content into English. This multilingual analysis enhances the comprehensibility and accessibility of the dataset. Following the translation process, the database is systematically updated with the translated text.

Step #5: Data Preprocessing: Titles, bodies, prompts, and responses from various source types were cleaned, stop-words removed, and tokenized.

To execute all these steps, we built a pipeline that takes as input the JSON files, and outputs the needed subset in the form of a database. Our pipeline uses various technologies, such as *SQLite* for data management, *FastText* for identifying non-English conversations, *Google Translator* library for their translation, *NLTK & Spacy* for meticulous cleaning and tokenization, complemented by the efficiency of *Dask* for concurrent processing in both cleaning and tokenization tasks. The pipeline and the generated database are available for replication and extension purposes[2].

2.2 Refactoring Patterns Extraction

To identify refactoring documentation patterns, we perform a series of manual and automated activities:

Data sources associated with developer intention about refactoring. As our study focuses on refactorings, our analysis is limited to source types where refactorings were discussed as part of developer-ChatGPT conversations. Hence, we first extracted all the different conversations from the source dataset. To ensure that the selected software artifacts are about refactoring and to reduce the occurrence of false positives, we focused on prompts that reported developers' intention about the application of refactoring (*i.e.*, having the keyword '*refactor*'). The choice of '*refactor*', besides being used by all related studies, is intuitively the first term to identify ideal refactoring-related commits [6–8, 13, 25, 29]. This step resulted in the selection of three source types: commits, issues, and files. The procedure led to the examination of 470 commits, 69 issues, and 176 files.

Annotation of developer-ChatGPT conversations. When using ChatGPT, developers use natural language to describe the software development artifacts. Before running the experiment, a pilot experiment is conducted to ensure that the annotators reach a consensus on data annotation. Hence, given the diverse nature of developers who describe the problem, an automated approach to analyzing the prompt and answer text is not feasible. Therefore, we performed a thematic analysis approach of the prompt and response to identify refactoring documentation patterns based on guidelines provided by Cruzes *et al.* [11]. Thematic analysis is one of the most used methods in Software Engineering literature (*e.g.,* [10, 32]), which is a technique for identifying and recording patterns (or "themes") within a collection of descriptive labels. Next, we grouped this subset of conversations based on specific patterns. Further, to avoid redundancy of any pattern, we only considered one phrase if we found different patterns with the same meaning. For example, if we find patterns such as 'simplifying the code', 'code simplification', and 'simplify code', we add only one of these similar phrases to the

[1]https://deep-translator.readthedocs.io/en/latest/usage.html#google-translate
[2]https://smilevo.github.io/self-affirmed-refactoring/

Table 1: List of refactoring documentation in DevGPT ('*' captures the extension of the keyword).

Patterns				
Add*	Chang*	Chang* the name	Cleanup	Clean* up
Code clarity	Code clean*	Code organization	Code review	Clean code
Creat*	Customiz*	Easier to maintain	Encapsulat*	Enhanc*
Extend*	Extract*	Fix*	Inlin*	Improv*
Improv* code quality	Introduc*	Merg*	Modif*	Modulariz*
Migrat*	Mov*	Organiz*	Polish*	Reduc*
Refactor*	Refin*	Remov*	Remov* redundant code	Renam*
Remov* unused dependencies	Reorganiz*	Replac*	Restructur*	Rework*
Rewrit*	Simplif*	Split*		

list of patterns. This enables us to have a list of the most insightful and unique patterns, and it also helps to create more concise patterns that are usable to the readers. Furthermore, we manually analyzed conversation patterns (*i.e.,* learning settings) between developers and ChatGPT when seeking guidance on refactoring tasks. Two authors independently annotated the conversations, and any conflicts in their annotations were subsequently resolved through discussion.

3 EXPERIMENTAL RESULTS

3.1 RQ$_1$: What textual patterns do developers use to describe their refactoring needs using ChatGPT?

Approach. We manually inspect GitHub commits, issues, files to identify refactoring documentation patterns. These patterns are represented as a keyword or phrase frequently occurring in developer-ChatGPT conversations, as described in Section 2.

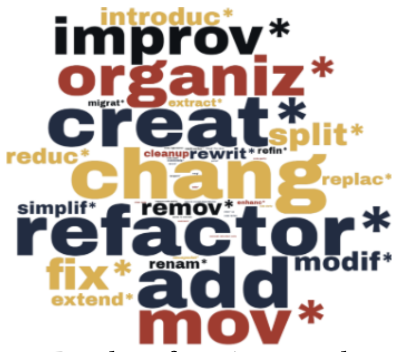


Figure 3: Popular refactoring textual patterns.

Results. Our in-depth inspection resulted in a list of 43 refactoring documentation patterns, as shown in Table 1 and Figure 3. Our findings show that the names of refactoring operations (*e.g.*, 'extract*', 'mov*', 'renam*') occur in the top frequently occurring patterns, and these patterns are mainly linked to code elements at different levels of granularity such as classes, methods, and variables. These specific terms are well-known software refactoring operations and indicate developers' knowledge of the catalog of refactoring operations. We also observe that the top-ranked refactoring operation-related keywords include 'mov*', 'renam*', and 'extract*'. Moreover, we observe the occurrences of refactoring specific terms such as 'cleanup', 'code quality', and 'restructur*'. The observation from this research question aligns with the work of Russo [31], who found that a significant portion of software professionals indicated their intention to use LLMs to improve and maintain their codebase.

Table 2: Summary of refactoring patterns, clustered by refactoring related categories.

Internal QA (%)	External QA (%)	Code Smell (%)
Dependency (25.33%)	Readability (22.13%)	Code smell (91%)
Inheritance (25%)	Usability (16.19%)	Long method (6%)
Composition (14.16%)	Performance (10.25%)	Duplicate code (3%)
Abstraction (12.16%)	Maintainability (8.32%)	
Coupling (10%)	Flexibility (8.32%)	
Encapsulation (8.83%)	Reusability (7.28%)	
Polymorphism (3%)	Accessibility (7.28%)	
Complexity (1.5%)	Modularity (4.6%)	
	Extensibility (3.86%)	
	Correctness (1.78%)	
	Manageability (1.48%)	
	Robustness (1.48%)	
	Compatibility (1.33%)	
	Scalability (1.18%)	
	Configurability (0.89%)	
	Simplicity (0.89%)	
	Reliability (0.89%)	
	Productivity (0.89%)	
	Adaptability (0.59%)	
	Understandability (0.29%)	

Summary for RQ$_1$. *Developer-ChatGPT conversations involve diverse textual patterns about refactoring activities. These patterns encompass generic descriptions like 'improve code readability' as well as specific refactoring operation names following Fowler's conventions, including 'extract' and 'rename'.*

3.2 RQ$_2$: What quality attributes does ChatGPT consider when describing refactoring?

Approach. After identifying the different refactoring documentation patterns, we categorize the patterns into three main categories (similar to previous studies [7, 9]): (1) internal quality attributes, (2) external quality attributes, and (3) code smells.

Results. Table 2 provides the list of refactoring documentation patterns, ranked based on their frequency, which we identify in ChatGPT responses. We observe that ChatGPT frequently mentions key internal quality attributes (such as 'inheritance', 'complexity', etc.), a wide range of external quality attributes (such as 'readability' and 'performance'), and code smells (such as 'duplicate code' and 'long method') that might impact code quality. To improve internal design, optimization of the structure of the system with respect to its dependency and inheritance appears to be the dominant focus that is consistently mentioned in the conversation (25.33% and 25%, respectively). Concerning external quality attribute, we observe the mention of refactorings to enhance nonfunctional attributes. Patterns such as 'readability', 'usability', and 'performance' represent the ChatGPT' main focus, with 22.13%, 16.19%, and 10.25%, respectively. The focus on code readability from the ChatGPT may be attributed to the fact that the model frequently tends to generate code that adheres to common naming and coding conventions, which was highlighted in a previous study when assessing the readability of the ChatGPT code [12]. Finally, for code smells, ChatGPT commonly used the generic term 'code smell' with 91%, the 'long method', and 'duplicate code' code smells represent the most popular anti-pattern ChatGPT intends to describe refactoring (6% and 3%, respectively). Furthermore, by analyzing the ChatGPT response, we notice that the model emphasizes certain coding practices, including dependency injection, naming conventions, exception handling,

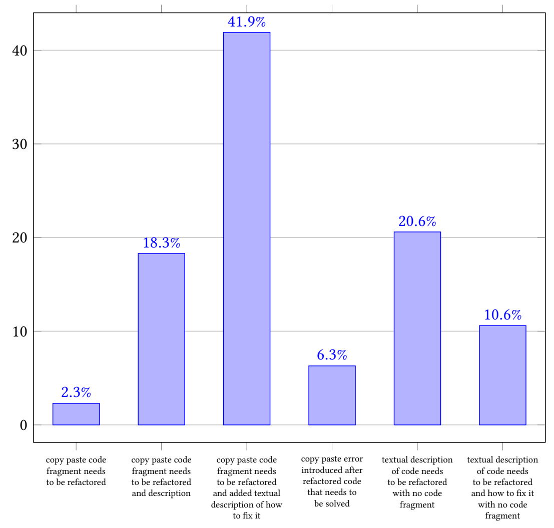


Figure 4: ChatGPT conversation patterns to refactor code.

single responsibility principle, unit tests, separation of concerns, design patterns, and behavior preservation.

Summary for RQ$_2$. *Our findings indicate that developers frequently make generic requests when seeking guidance on refactoring tasks, while ChatGPT typically includes the intention when applying refactorings, addressing quality issues related to internal attributes, external attributes, or code smells, along with applying well-known coding practices.*

3.3 RQ$_3$: How do developers typically initiate conversations with ChatGPT when seeking guidance on refactoring tasks?

Approach. To explore whether specific conversation patterns emerge as developers collaborate with ChatGPT for iterative refinement of refactoring solutions, we manually examine refactoring conversations and categorize the identified patterns.

Results. The effectiveness of any language model relies heavily on the ability of developers to craft appropriate prompts. Therefore, one of the outcomes of this work is to raise awareness among developers of the importance of prompt engineering, particularly in the context of refactoring. Analysis of developer prompts (as depicted in Figure 4) reveals that 2.3% of developers simply copy and paste code fragments that need to be refactored, assuming that ChatGPT can discern the intention behind the code. 18.3% of developers copy and paste code fragments that need to be refactored along with textual description/instruction to provide context. Most developers (41.9%) copy and paste code fragments that need to be refactored and add a textual description of how to fix it. About 6.3% of developers introduce copy and paste errors after the refactoring application suggested by ChatGPT. Some developers (20.6%) add textual description of the code that needs to be refactored, but without adding the associated code fragments. Others (10.6%) add a textual description of the code that needs to be refactored and how to fix it, but also without adding the associated code fragments.

Summary for RQ$_3$. *Many developers (41.9%) tend to copy and paste code fragments that require refactoring, along with providing a textual description of how they want them to be addressed.*

4 DISCUSSION & CONCLUSION

Takeaway #1: ***ChatGPT limited understanding of the broader context of the codebase.*** Although ChatGPT provides an informative context of refactorings, its comprehension of the entire codebase is restricted, leading to possible missing dependencies and codependencies. This limitation becomes apparent in instances where ChatGPT makes suggestions based on misunderstandings or false assumptions about the code. This exploratory study serves to unveil this practical limitation, encouraging developers to grasp the model's mechanics rather than treating it as a black box that consistently generates acceptable answers. The **RQ$_1$** and **RQ$_2$** results not only show instances where developers express their refactoring needs for certain queries, but also reveal potential issues with suggested code changes. Some developers encountered compiler errors introduced by ChatGPT-provided code, and in some cases, the suggested refactoring led to test failures, contrary to the expectation that refactoring should preserve the system's internal behavior.

Takeaway #2: ***The quality of ChatGPT responses is highly dependent on the quality of the prompts.*** The effectiveness of ChatGPT is closely related to the quality of its training dataset [15]. According to the developer-ChatGPT conversations (**RQ$_1$** and **RQ$_2$**), ChatGPT offers insightful suggestions on refactorings, which is particularly helpful for addressing non-urgent issues like code style, adding comments, and more. On the other hand, challenges arise when ChatGPT misunderstands the reported code fragments that need to be refactored or becomes confused when receiving a code excerpt instead of the entire code, especially when the code is too large to provide. Its corrections sometimes also misunderstood the purpose of an excerpt and would slightly alter the logic. Therefore, the more complex the given code input, the more likely that the code output will not work properly. By identifying common patterns and challenges in these refactoring conversations, researchers and developers can work toward improving ChatGPT's capabilities (*e.g.,* following software documentation quality framework [35]). This understanding can guide the refinement of the model to better address developers' needs.

Takeaway #3: ***Variability in refactoring conversation learning settings between developers and ChatGPT.*** From **RQ$_3$**, we observe that some of the developers' prompts were zero-shot, where they relied on the model's generative ability to understand an issue or propose a corresponding code fix. Zero-shot learning involves the model making decisions on presumably unseen data by approximating it with previously trained code [41]. For instance, the prompt asks the model to refactor the code to eliminate duplication of an input source code. Further, the prompt can be augmented by adding a label to the unseen data, *i.e.,* one-shot [16]. For example, mentioning how large file splitting can be performed can guide the model towards the decision to take (class extraction). Some developers opted for few-shot learning, entering code fragments and asking ChatGPT to propose fixes while checking for design antipatterns (*e.g.,* duplicated code, bad names, dead code) by providing an example. Few-shot learning involves providing context-specific examples to help the model make better decisions about complex tasks [36]. For instance, the input of code changes that address the complexity of a given class. This RQ shows the non-uniformity of developers' prompts, with a majority debating how to extract the necessary action from the models, while others overestimate the model's capabilities. This aligns with previous studies that have demonstrated the susceptibility of ChatGPT to hallucinations when it comes to coding semantic structures [20, 23, 24, 39, 43]. Insights from these conversations can inform the integration of ChatGPT or similar models into development tools, making them more user-friendly and aligned with developers' expectations during refactoring tasks.

Takeaway #4: ***Need for better refactoring descriptions.*** During our manual analysis, we observed various cases where developers ask the model to refactor a given input code by only mentioning the word refactoring (*e.g.,* refactor this...). While it allows for a high flexibility, it is better to constrain the model with what the developer expects. For instance, Adding the intent (why?) behind the refactoring, and potential instructions (how?) can help with ensuring specific coding practices and styles that are targeted by the developer's own decisions or their company policy.

REFERENCES

[1] [n. d.]. https://github.com/UIUC-Chatbot/ai-ta-backend/issues/57.

[2] [n. d.]. https://futurism.com/the-byte/github-92-percent-programmers-using-ai.

[3] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards human-bot collaborative software architecting with chatgpt. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 279–285.

[4] Ali Al-Kaswan and Maliheh Izadi. 2023. The (ab) use of Open Source Code to Train Large Language Models. *arXiv preprint arXiv:2302.13681* (2023).

[5] Ajmain Inqiad Alam, Palash Ranjan Roy, Farouq Al-omari, Chanchal Kumar Roy, Banani Roy, and Kevin Schneider. 2023. GPTCloneBench: A comprehensive benchmark of semantic clones and cross-language clones using GPT-3 model and SemanticCloneBench. *arXiv preprint arXiv:2308.13963* (2023).

[6] Eman Abdullah AlOmar, Hussein AlRubaye, Mohamed Wiem Mkaouer, Ali Ouni, and Marouane Kessentini. 2021. Refactoring Practices in the Context of Modern Code Review: An Industrial Case Study at Xerox. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 348–357.

[7] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni. 2019. Can refactoring be self-affirmed? an exploratory study on how developers document their refactoring activities in commit messages. In *International Workshop on Refactoring-accepted. IEEE.*

[8] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni. 2021. Toward the automatic classification of self-affirmed refactoring. *Journal of Systems and Software* 171 (2021), 110821.

[9] Eman Abdullah AlOmar, Anthony Peruma, Mohamed Wiem Mkaouer, Christian Newman, Ali Ouni, and Marouane Kessentini. 2021. How we refactor and how we document it? On the use of supervised machine learning algorithms to classify refactoring documentation. *Expert Systems with Applications* 167 (2021), 114176.

[10] Fabio Calefato, Luigi Quaranta, and Filippo Lanubile. 2023. A Lot of Talk and a Badge: An Empirical Analysis of Personal Achievements in GitHub. *arXiv preprint arXiv:2303.14702* (2023).

[11] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.

[12] Carlos Dantas, Adriano Rocha, and Marcelo Maia. 2023. Assessing the Readability of ChatGPT Code Snippet Recommendations: A Comparative Study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*. 283–292.

[13] Zhang Di, Bing Li, Zengyang Li, and Peng Liang. 2018. A preliminary investigation of self-admitted refactorings in open source software (S). In *International Conferences on Software Engineering and Knowledge Engineering*, Vol. 2018. KSI Research Inc. and Knowledge Systems Institute Graduate School, 165–168.

[14] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration Code Generation via ChatGPT. *arXiv preprint arXiv:2304.07590* (2023).

[15] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. *arXiv preprint arXiv:2310.03533* (2023).

[16] Li Fei-Fei, Robert Fergus, and Pietro Perona. 2006. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence* 28, 4 (2006), 594–611.

[17] Yunhe Feng, Sreecharan Vanam, Manasa Cherukupally, Weijian Zheng, Meikang Qiu, and Haihua Chen. 2023. Investigating Code Generation Performance of ChatGPT with Crowdsourcing Social Data. In *Proceedings of the 47th IEEE Computer Software and Applications Conference*. 1–10.

[18] Martin Fowler, Kent Beck, John Brant, William Opdyke, and don Roberts. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. http://dl.acm.org/citation.cfm?id=311424

[19] Md Asraful Haque and Shuai Li. 2023. The Potential Use of ChatGPT for Debugging and Bug Fixing. *EAI Endorsed Transactions on AI and Robotics* 2, 1 (2023), e4–e4.

[20] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620* (2023).

[21] Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. 2023. Is ChatGPT a good translator? A preliminary study. *arXiv preprint arXiv:2301.08745* (2023).

[22] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and individual differences* 103 (2023), 102274.

[23] Bonan Kou, Shengmai Chen, Zhijie Wang, Lei Ma, and Tianyi Zhang. 2023. Is Model Attention Aligned with Human Attention? An Empirical Study on Large Language Models for Code Generation. *arXiv preprint arXiv:2306.01220* (2023).

[24] Wei Ma, Shangqing Liu, Wenhan Wang, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, and Yang Liu. 2023. The Scope of ChatGPT in Software Engineering: A Thorough Investigation. *arXiv preprint arXiv:2305.12138* (2023).

[25] Emerson Murphy-Hill, Andrew P Black, Danny Dig, and Chris Parnin. 2008. Gathering refactoring data: a comparison of four methods. In *Proceedings of the 2nd Workshop on Refactoring Tools*. 1–5.

[26] Nascimento Nathalia, Alencar Paulo, and Cowan Donald. 2023. Artificial Intelligence vs. Software Engineers: An Empirical Study on Performance and Efficiency using ChatGPT. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*. 24–33.

[27] David N Palacio, Alejandro Velasco, Daniel Rodriguez-Cardenas, Kevin Moran, and Denys Poshyvanyk. 2023. Evaluating and Explaining Large Language Models for Code Using Syntactic Structures. *arXiv preprint arXiv:2308.03873* (2023).

[28] Rohith Pudari and Neil A Ernst. 2023. From Copilot to Pilot: Towards AI Supported Software Development. *arXiv preprint arXiv:2303.04142* (2023).

[29] Jacek Ratzinger, Thomas Sigmund, and Harald C. Gall. 2008. On the Relation of Refactorings and Software Defect Prediction. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories* (Leipzig, Germany) *(MSR '08)*. ACM, New York, NY, USA, 35–38. https://doi.org/10.1145/1370750.1370759

[30] Palash R Roy, Ajmain I Alam, Farouq Al-omari, Banani Roy, Chanchal K Roy, and Kevin A Schneider. 2023. Unveiling the potential of large language models in generating semantic and cross-language clones. *arXiv preprint arXiv:2309.06424* (2023).

[31] Daniel Russo. 2023. Navigating the complexity of generative ai adoption in software engineering. *arXiv preprint arXiv:2307.06081* (2023).

[32] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why We Refactor? Confessions of GitHub Contributors. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. ACM, New York, NY, USA, 858–870. https://doi.org/10.1145/2950290.2950305

[33] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653* (2023).

[34] Christoph Treude and Hideaki Hata. 2023. She Elicits Requirements and He Tests: Software Engineering Gender Bias in Large Language Models. *arXiv preprint arXiv:2303.10131* (2023).

[35] Christoph Treude, Justin Middleton, and Thushari Atapattu. 2020. Beyond accuracy: Assessing software documentation quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1509–1512.

[36] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[37] Yanlin Wanga, Lianghong Guob Ensheng Shic, Wenqing Chena Jiachi Chena, Wanjun Zhonga Menghan Wangd Hui Lie, and Hongyu Zhangf Ziyu Lyug Zibin Zhenga. [n. d.]. You Augment Me: Exploring ChatGPT-based Data Augmentation

for Semantic Code Search. ([n. d.]).

[38] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[39] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *arXiv preprint arXiv:2303.07839* (2023).

[40] Chunqiu Steven Xia and Lingming Zhang. 2023. Keep the Conversation Going: Fixing 162 out of 337 bugs for 0.42 each using ChatGPT. *arXiv preprint arXiv:2304.00385* (2023).

[41] Yongqin Xian, Bernt Schiele, and Zeynep Akata. 2017. Zero-shot learning-the good, the bad and the ugly. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4582–4591.

[42] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. (2024).

[43] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712* (2023).

[44] Quanjun Zhang, Tongke Zhang, Juan Zhai, Chunrong Fang, Bowen Yu, Weisong Sun, and Zhenyu Chen. 2023. A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair. *arXiv preprint arXiv:2310.08879* (2023).

[45] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).