Using Large Language Models to Support Software Engineering Documentation in Waterfall Life Cycles: Are We There Yet?

Antonio Della Porta[1,*,†], Vincenzo De Martino[1,†], Gilberto Recupito[1,†], Carmine Iemmino[1,†], Gemma Catolino[1,†], Dario Di Nucci[1,†] and Fabio Palomba[1,†]

[1]SeSa Lab - Università Degli Studi di Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano, Salerno, Italy

Abstract

Software documentation is key for producing high-quality projects and ensuring their smooth evolution. Nonetheless, the activity of writing software artifacts is time-consuming and effort-prone. Looking at the existing body of knowledge, we outline limited evidence of how automated approaches may support practitioners when documenting the artifacts produced throughout the software lifecycle. In particular, there is still a lack of investigations into the capabilities of Large Language Models (LLMs), which are indeed supposed to be highly beneficial in this respect. In this paper, we propose a preliminary case study to understand how LLMs can support the development of the documentation of projects developed through a Waterfall lifecycle. Using ChatGPT, we engineered specific prompts to generate and validate the artifacts produced, taking an existing, documented software engineering project as an oracle. The main findings of the study show the ability of ChatGPT to produce most artifacts correctly. In addition, we find that software engineers would require a relatively low effort to adapt the outputs provided by ChatGPT to their own context, especially for textual artifacts.

Keywords

Large Language Model, Artificial Intelligence for Software Engineering, ChatGPT,

1. Introduction

Integrating Large Language Models (LLMs) into various domains has recently garnered significant attention. Recent statistics indicate that ChatGPT, a prominent example of LLM, has gathered over 180 million users, underscoring the widespread adoption of such models [1]. LLMs showcase a remarkable versatility, particularly in software engineering [2], thus leading practitioners to wonder how these models can effectively replicate their tasks. From here, there is a need to explore their potential within the Software Development Lifecycle (SDLC). In particular, the literature showed how LLMs can simulate team members in a development environment, perform code analysis, generate code, and predict bugs [3]. These AI-powered systems can analyze large amounts of code and data quickly and accurately, enabling automation of repetitive tasks and allowing developers to focus on more complex issues [4].

These benefits allowed us to resolve key issues in software engineering tasks, especially considering software development and maintenance activities [5]. However, other software engineering tasks, especially those related to documentation, are still defined as key challenges [6]. Since there is a lack of studies in this specific field, we aim to provide preliminary results to show the capabilities of an LLM to tackle the challenge of crafting software documentation. We selected a Waterfall Life Cycle project to explore LLMs' documentation abilities across development phases, from requirements to technical details. Through this preliminary case study, we employed ChatGPT [1] to generate documentation artifacts.

We aim to evaluate ChatGPT's real-world efficiency by comparing it to a benchmark project and gauging the effort to produce similarly high-quality artifacts. Preliminary findings suggest ChatGPT eases documentation and speeds up design replication but requires human input for response refinement and query tuning. Initial integration efforts are moderate, but some artifacts necessitated revised prompts and external software for satisfactory outcomes.

2. Related Work

Artificial Intelligence for Software Engineering (AI4SE) is a well-known research area that aims to develop AI

Ital-IA 2024: 4th National Conference on Artificial Intelligence, organized by CINI, May 29-30, 2024, Naples, Italy

*Corresponding author.

†These authors contributed equally.

adellaporta@unisa.it (A. Della Porta); vdemartino@unisa.it (V. De Martino); grecupito@unisa.it (G. Recupito); c.iemmino@studenti.unisa.it (C. Iemmino); gcatolino@unisa.it (G. Catolino); ddinucci@unisa.it (D. Di Nucci); fpalomba@unisa.it (F. Palomba)

0000-0003-1860-8404 (A. Della Porta); 0000-0003-1485-4560 (V. De Martino); 0000-0001-8088-1001 (G. Recupito); 0000-0002-4689-3401 (G. Catolino); 0000-0003-4927-9324 (D. Di Nucci); 0000-0001-9337-5116 (F. Palomba)

[1]https://chat.openai.com

solutions and SE practices to improve software development processes and tools [7, 8]. With the emergence and proliferation of LLMs, this field has encountered new opportunities to support and streamline the labors of software engineers and researchers [5].

In the vein of such advancements, De Vito et al. [9] introduced ECHO, an innovative method utilizing LLMs to aid software engineers in improving the quality of UML use cases. Further extending the utility of AI in SE, De Vito et al. [10] a chatbot designed for software engineering, streamlining tasks like code review, testing, and criteria evaluation.

Ahmad et al. [11] explore the role of ChatGPT as a bot in collaborative software architecting to support the analysis, synthesis, and evaluation of microservices-based software. A study by Liang et al. [12] surveyed developers' perceptions, noting issues like code not meeting requirements. Despite these advancements, the domain of AI-assisted documentation in SE remains underexplored, especially the comprehensive support for the entire documentation lifecycle.

As Robillard et al. [13] highlighted, traditional documentation practices are inefficient because of the manual nature of its creation and the gap between creators and consumers. Aghajani et al. [14] reported that documentation suffers numerous shortcomings and problems, including insufficient and inadequate content and outdated and ambiguous information. Recent investigations have further explored the extent to which LLMs can assist in tasks like writing code [15], conducting code reviews [16], providing code explanations [17], and teaching programming concepts [18]. These studies suggest the potentiality of LLMs to create significant support in the activities involved in the SDLC and focus the human effort on the quality and relevance of the results.

White et al. [19] emphasized the importance of prompt engineering to guide LLMs by presenting a catalog of patterns to dialogue with LLMs to achieve satisfactory outputs. A well-written prompt enables correct answers by minimizing prompts [20, 21, 22]. Our work builds on these studies, exploiting how to use prompts to support documentation artifacts.

Our research is motivated by the goal of comprehensively understanding how ChatGPT can support both students and practitioners during the software development lifecycle, focusing on creating improved documentation of software systems. We aim to shed light on the role of ChatGPT and LLMs in simplifying the development process and assess the complexities involved in using ChatGPT to produce high-quality results.

3. Research Method

The *goal* of the study was to determine to what extent LLMs can support the activities of a software engineer when writing documentation in a software project employing the Waterfall Life Cycle model, with the *purpose* of providing software engineers elements that can be leveraged to support and improve the design process of software projects. The *perspective* is of both researchers and practitioners. The former are interested in understanding the current potential and limitations of using LLMs for documentation tasks, possibly identifying opportunities for further research and improvement. The latter are interested in assessing how LLMs can act as documentation assistants in practice, verifying whether these models may be employed in real-world contexts and potentially integrating them into their workflow.

3.1. Research Question

Our *research question* aimed to understand whether LMMs can substantially support the software documentation activities developed using a Waterfall Life model. Understanding how documentation writing activities using LLM can improve artifacts and possibly reduce effort would be crucial. We chose ChatGPT because of its popularity and availability, in line with similar studies [23, 11].

In this context, we formulated the following research question.

> **RQ$_1$.** *To what extent can ChatGPT support software engineering documentation tasks in a Waterfall Life Cycle model?*

To address our research question, we conducted a preliminary case study [24] using an oracle project and comparing it to the output of the LLM to provide insights into understanding its usefulness for documentation tasks. We followed the guidelines by Wohlin et al. [25] and the *ACM/SIGSOFT Empirical Standards* for the report.[2]

3.2. Context of the Study

To address the goal of our work and provide preliminary insights into the capabilities of ChatGPT for documentation tasks, we selected a project named Rojina Review, a web-based platform for news and reviews of video games. This project has 100k lines of code and was initially developed by a team of three software engineering students at our university using a Waterfall lifecycle. On the one hand, we selected a fully developed project, *i.e.*, with the full set of artifacts already developed to have a *ground truth* against which to assess the capabilities of ChatGPT.

[2]Available at https://github.com/acmsigsoft/EmpiricalStandards. We leveraged the guidelines available for *"General Standard"* and *"Case Study"*.

Table 1
Generated Artifacts

Document	Description	Artifact Considered
Requirements Analysis Document	Gathers and analyzes the system requirements.	Scenarios
		Functional Requirements
		Non Functional Requirements
		Use Cases
		Class Diagram
		Sequence Diagram
		Statechart Diagram
System Design Document	Outlines the overall system architecture.	Design Goals
		Subsystems Division
		Software/Hardware Mapping
		Boundary Conditions
Object Design Document	Defines the component design.	Class Interfaces
		Design Pattern
Test Plan & Test Case Specification	Describes how to test the system.	Test Case Specifications
		Category Partition

On the other hand, this project was closely supervised by the paper's authors. We were *familiar* with the business case and the artifacts that should have been developed, but also *confident* of the quality of the project. We are aware of potential threats to internal and external validity related to this choice. However, we believe the project was good enough to ensure a satisfactory preliminary assessment. Following Bruegge and Dutoit [26], we briefly explain the documents created for this project in Table 1.

3.3. Formulating the Waterfall Story

Before starting our study, we gathered a working group to determine a suitable prompt for ChatGPT. We adopted a specific prompting process when interacting with ChatGPT for all artifacts to be created. This method allows the conduction of the activities to produce documentation artifacts, simulating the phases of the Waterfall lifecycle Model. In detail, the process includes three steps:

#1-Initial interaction: We set up the environment in ChatGPT. Specifically, we adopted a single chat to interact and prevent the LLM from losing the project context. Subsequently, we provided ChatGPT with an initial prompt containing the preliminary information of the project. We asked ChatGPT to provide information concerning the problem statement.

#2-Artifact generation: to maintain the context of the output generated in the previous phase, we asked ChatGPT to provide the previous artifact at each development phase.

#3-Inter-rater assessment: following the extraction of answers provided by ChatGPT, an inter-rater assessment process was initiated to evaluate the generated output by the three first authors. The artifact produced by ChatGPT was compared with the same artifact in Rojina Review. The three first authors of the paper had to agree to make an artifact acceptable. In case of disagreement, a collaborative discussion was facilitated to address and resolve assessment disparities. Afterward, the feedback was re-submitted to improve the quality of the artifact. In this case, the discussion about creating the artifact continued, and the feedback from this phase was provided to ChatGPT until the output was evaluated compliant for the evaluators or the LLM could not respond better than the previous phase.

When the third step of the process was completed, the second step was repeated to create the next artifact. Additionally, we noted that the language seemed more accurate when we asked ChatGPT to impersonate a software engineer. For this reason, we used a generic prompt that guided our research:

Prompt of Requirement Tasks

You have to impersonate a **software engineer** who has to produce the project documentation of a software project. Consider the following problem statement to generate the output:
<problem statement content>
#**Optional**: Given that you have *<additional info>* (e.g., the non-functional requirements in the RAD)
Generate *<name of the artifact>* for the scope of the software project that we defined
#**Optional**(only for UML artifacts) using the *PlantUML syntax*.

We then started to generate the documentation in an iterative and incremental process. The set of the documentation artifacts, according to the Waterfall Model, the five main documents, and related tasks, are specified in Table 1.

3.4. Data Extraction

From the documentation of the project selected, we extracted the document produced for each phase of the Waterfall Model, a set of the most important artifacts as listed in Table 1.

We produced a prompt for each artifact that ChatGPT could use to generate the artifact. For the generation of the diagrams, we have used PlantUML[3]. This open-source tool allows users to create Unified Modeling Lan-

[3]Source code available at https://github.com/plantuml/plantuml

guage (UML) diagrams using a plain text language. The tool follows the findings of Cámara et al. [27], stating that ChatGPT produces fewer syntactic mistakes and gets significantly better results when using PlantUML compared to other tools, such as USE[4] tool.

Table 2
Effort Mapping

Effort	Description
Low Effort	The desired answer is obtained with a maximum of *two* prompts, does not need to be too much articulated, and does not require corrections, so it can easily used.
Medium Effort	The desired answer is produced with several prompts ranging from *three* to *five*; the response may require manual modification where it is more complicated to have the bot adjust the response.
High Effort	The desired answer is obtained with a minimum of *six* very detailed prompts, and the response requires manual corrections that the bot cannot implement.

3.5. Data Analysis

To analyze the result obtained using ChatGPT, the first three authors of the paper, who have significant experience in software engineering both from an academic and enterprise perspective, had defined a set of criteria to evaluate the effort needed by a software engineer who has to be supported in creating the artifacts of the documentation. Those criteria, listed in Table 2, consider the number of prompts needed and the level of adjustment of the prompt to reach an optimal result from ChatGPT. The final acceptance of each artifact produced by ChatGPT was given by comparing it with the same artifact in Rojina Review to assess the quality.

4. Preliminary Results

We submitted the prompts to ChatGPT for each selected artifact to address our research question and obtained the results detailed in Table 3. We started with the extraction of *scenarios*. During the interaction, we noted that ChatGPT finds difficulties in identifying key elements in the context. For instance, actors involved in a specific functionality are switched compared to the context of the system given in input. Therefore, we added additional prompts to address these issues. Subsequently, we extracted *functional requirements*; ChatGPT produced well-structured and formatted requirements after the first interaction. On the same line, the results for the *non-functional requirements*; by defining the functional ones, ChatGPT has been able to extract directly the related *non-functional requirements* with a single prompt. *Use Cases* need specific prompts for each system's functionality defined previously. Moreover, additional prompts were required to get the alternative flows. For the *class diagram*, ChatGPT failed to produce a correct result with the right hierarchies, relationships, and cardinality. We observed the need to write the specific string "system class diagram" to obtain results, allowing ChatGPT to report associations among classes. For these reasons, the LLM fail to give a correct result.

Table 3
Results

Artifact	Effort
Scenarios	Medium Effort
Functional Requirements	Low Effort
Non Functional Requirements	Low Effort
Use Cases	Medium Effort
Class Diagram	High Effort
Sequence Diagram	High Effort
Statechart Diagram	Medium Effort
Design Goals	Medium Effort
Subsystems Division	High Effort
Software/Hardware Mapping	Low Effort
Boundary Conditions	Low Effort
Class Interfaces	Low Effort
Design Pattern	Low Effort
Test case specification	Medium Effort
Category Partition	High Effort

On the one hand, in the *statechart*, a restricted number of prompts were needed to generate artifacts comparable to Rojina Review. On the other hand, the Sequence Diagrams needed more prompts with additional specifications to achieve a good result.

We needed a few prompts to generate the *design goals*; assigning and ordering using priority needed more prompts. The *subsystems division* needed many prompts and corrections to get a result comparable with the artifact of Rojina Review because initially, ChatGPT produced a semantically incorrect division, so we needed to provide more details and required the PlantUML code. There were no issues for *software/hardware mapping*, *boundary conditions*, *class interfaces*, and *design patterns*: ChatGPT has been able to generate a good result without effort.

For the testing artifacts of the project, the *category partition* required many prompts and was very specific

[4]Source code available at https://github.com/useocl/use

for each functionality to test. Otherwise, the *test case specifications* was easier, as is using the category partition as input to build each test.

5. Threats to Validity

Construct Validity. The main concern for construct validity in our study concerns subject selection, particularly the version of the AI model. For evaluation, we used the GPT-3.5 model, the most advanced and available version during the research. Even if the GPT-4 version has been released, the use is currently limited by strict speed limits, and early feedback from the user community suggests potential stability and accuracy issues.

Internal Validity. To ensure robust internal validity, we carefully considered factors that could influence the outcomes derived from the LLM. Recognizing that LLMs' responses are susceptible to prompt formulation, we conducted preliminary tests to identify the most effective prompt structures [19, 22]. This step was crucial to minimize variations in the model's responses that could arise from prompt-related biases, thereby ensuring that our findings more accurately reflect the capabilities of the LLM rather than the nuances of our prompt phrasing. Additionally, each interaction with the LLM was assessed iteratively by more authors through inter-rater assessment, allowing the reduction of the subjectivity of the results. We evaluated the accuracy of documents generated by ChatGPT using a high-quality project from an undergraduate software engineering course as an oracle. This comparison was critical to verify that the observed results were indeed attributable to ChatGPT's capabilities.

External Validity. The external validity threat examines whether the results of a study can be generalized to other contexts. We experienced only one case study of moderate complexity, which may limit the generalizability of the study. Scenarios with greater development complexity, different types of development (*e.g.*, agile instead of waterfall), and human writing prompt skills may affect the external validity of this research. Future work may involve validating the process with project managers and a more significant number of software projects to minimize this external threat to validity.

6. Conclusion and Future Work

In our study, to what extent ChatGPT can support software engineers in documenting waterfall projects. We compared its use with a high-level university project, focusing on response variability, design impact, and the balance between AI support and human oversight. Our preliminary findings suggest ChatGPT reduces time and effort. Future work will involve a longitudinal study with professional feedback, exploring how prompt generation expertise enhances real-world outputs.

Acknowledgments

This work has been partially supported by the European Union - NextGenerationEU through the Italian Ministry of University and Research, Projects PRIN 2022 "QualAI: Continuous Quality Improvement of AI-based Systems" (grant n. 2022B3BP5S , CUP: H53D23003510006) and PRIN 2022 PNRR "FRINGE: context-aware FaiRness engineerING in complex software systEms" (grant n. P2022553SL, CUP: D53D23017340001). The opinions presented in this article solely belong to the author(s) and do not necessarily reflect those of the European Union or The European Research Executive Agency. The European Union and the granting authority cannot be held accountable for these views.

References

[1] DemandSage, Chatgpt statistics for 2024 (users demographics and facts), 2024. URL: https://www.demandsage.com/chatgpt-statistics/, accessed: January 13, 2024.

[2] S. Wang, L. Huang, A. Gao, J. Ge, T. Zhang, H. Feng, I. Satyarth, M. Li, H. Zhang, V. Ng, Machine/deep learning for software engineering: A systematic literature review, IEEE Transactions on Software Engineering 49 (2023) 1188–1231. doi:10.1109/TSE.2022.3173346.

[3] L. Belzner, T. Gabor, M. Wirsing, Large language model assisted software engineering: prospects, challenges, and a case study, in: International Conference on Bridging the Gap between AI and Reality, Springer, 2023, pp. 355–374.

[4] Y. K. Dwivedi, N. Kshetri, L. Hughes, E. L. Slade, A. Jeyaraj, A. K. Kar, A. M. Baabdullah, A. Koohang, V. Raghavan, M. Ahuja, et al., "so what if chatgpt wrote it?" multidisciplinary perspectives on opportunities, challenges and implications of generative conversational ai for research, practice and policy, International Journal of Information Management 71 (2023) 102642.

[5] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, arXiv preprint arXiv:2308.10620 (2023).

[6] I. Ozkaya, Application of large language models to software engineering tasks: Opportunities, risks,

and implications, IEEE Software 40 (2023) 4–8. doi:10.1109/MS.2023.3248401.

[7] M. Barenkamp, J. Rebstadt, O. Thomas, Applications of ai in classical software engineering, AI Perspectives 2 (2020) 1.

[8] T. Xie, Intelligent software engineering: Synergy between ai and software engineering, in: Proceedings of the 11th Innovations in Software Engineering Conference, 2018, pp. 1–1.

[9] G. De Vito, F. Palomba, C. Gravino, S. Di Martino, F. Ferrucci, Echo: An approach to enhance use case quality exploiting large language models, in: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2023, pp. 53–60. doi:10.1109/SEAA60479.2023.00017.

[10] G. De Vito, S. Lambiase, F. Palomba, F. Ferrucci, Meet c4se: Your new collaborator for software engineering tasks, in: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2023, pp. 235–238. doi:10.1109/SEAA60479.2023.00044.

[11] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, T. Mikkonen, Towards human-bot collaborative software architecting with chatgpt, in: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, 2023, pp. 279–285.

[12] J. T. Liang, C. Yang, B. A. Myers, Understanding the usability of ai programming assistants, arXiv preprint arXiv:2303.17125 (2023).

[13] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, E. Wong, On-demand developer documentation, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 479–483. doi:10.1109/ICSME.2017.17.

[14] E. Aghajani, C. Nagy, M. Linares-Vásquez, L. Moreno, G. Bavota, M. Lanza, D. C. Shepherd, Software documentation: The practitioners' perspective, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 590–601. URL: https://doi.org/10.1145/3377811.3380405. doi:10.1145/3377811.3380405.

[15] P. Vaithilingam, T. Zhang, E. L. Glassman, Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, in: Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, CHI EA '22, Association for Computing Machinery, New York, NY, USA, 2022. URL: https://doi.org/10.1145/3491101.3519665. doi:10.1145/3491101.3519665.

[16] Q. Guo, J. Cao, X. Xie, S. Liu, X. Li, B. Chen, X. Peng, Exploring the potential of chatgpt in automated code refinement: An empirical study, arXiv preprint arXiv:2309.08221 (2023).

[17] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, A. Hellas, Comparing code explanations created by students and large language models, arXiv preprint arXiv:2304.03938 (2023).

[18] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, J. Sorva, Exploring the responses of large language models to beginner programmers' help requests, arXiv preprint arXiv:2306.05715 (2023).

[19] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D. C. Schmidt, A prompt pattern catalog to enhance prompt engineering with chatgpt, arXiv preprint arXiv:2302.11382 (2023).

[20] E. A. Van Dis, J. Bollen, W. Zuidema, R. van Rooij, C. L. Bockting, Chatgpt: five priorities for research, Nature 614 (2023) 224–226.

[21] S. Arora, A. Narayan, M. F. Chen, L. Orr, N. Guha, K. Bhatia, I. Chami, F. Sala, C. Ré, Ask me anything: A simple strategy for prompting language models, arXiv preprint arXiv:2210.02441 (2022).

[22] U. Lee, H. Jung, Y. Jeon, Y. Sohn, W. Hwang, J. Moon, H. Kim, Few-shot is enough: exploring chatgpt prompt engineering method for automatic question generation in english education, Education and Information Technologies (2023) 1–33.

[23] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, W. Lam, Chatgpt and software testing education: Promises and perils, in: 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2023, pp. 4130–4137. doi:10.1109/ICSTW58534.2023.00078.

[24] R. K. Yin, Case study research and applications, volume 6, Sage Thousand Oaks, CA, 2018.

[25] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[26] B. Bruegge, A. H. Dutoit, Object–oriented software engineering. using uml, patterns, and java, Learning 5 (2009) 7.

[27] J. Cámara, J. Troya, L. Burgueño, A. Vallecillo, On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml., Softw Syst Model 22 (2023) 781–793. doi:https://doi.org/10.1007/s10270-023-01105-5.