CHALMERS
UNIVERSITY OF TECHNOLOGY

Enhancing Software Design and Developer Experience Via LLMs

Citation for the original published paper (version of record):
Sun, S. (2024). Enhancing Software Design and Developer Experience Via LLMs. Proceedings - 2024 39th ACM/IEEE International Conference on Automated Software Engineering, ASE 2024: 2498-2501. http://dx.doi.org/10.1145/3691620.3695606

(article starts on next page)

Enhancing Software Design and Developer Experience Via LLMs

Simin Sun
simin.sun@gu.se
Chalmers | University of Gothenburg
Gothenburg, Sweden

ABSTRACT

This research explores the transformative potential of generative AI in software development. Generative AI is revolutionizing the field by offering capabilities to automatically generate, refactor, and test code. Through the use of action research, new methods and tools based on generative AI models are studied and developed. The initial focus is on the models' ability to comprehend high-level design concepts. Subsequently, the research moves into the augmented generation of software artifacts. Finally, organization-specific or task-specific methods are introduced to enhance software developers' productivity and experience.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; **Software development methods**;

KEYWORDS

Generative AI, LLM, CI/CD, Log Analysis

ACM Reference Format:
Simin Sun. 2024. Enhancing Software Design and Developer Experience Via LLMs. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3691620.3695606

1 INTRODUCTION

The integration of Artificial Intelligence (AI) in software development is ushering in a new era of automation and efficiency. Recent advancements in generative AI have demonstrated significant potential in transforming various aspects of software development, from code generation and refactoring to automated testing.

Recent surveys [4, 6] explore various applications of Large Language Models (LLMs) in software engineering activities and identify current stages and remaining challenges by analyzing separate software development activities without focusing on processes covering multiple activities. Most research concentrates on specific stages, such as requirement generation [11], code generation [18], and testing [19]. This research aims to focus on the entire software design and development process rather than treating development activities as isolated events. Current Machine Learning (ML) solutions are typically applied after code compilation or submission, which delays feedback for developers. According to developers' feedback, there is a preference for receiving insights and advice at the earliest possible stage [7]. Therefore, this study also intends to examine the current state of development suggestions powered by generative AI and LLMs, with the goal of providing early-stage or even runtime suggestions to developers.

The initial step involves thoroughly examining the capabilities of current LLMs to identify relevant procedures, encompassing a range of activities. Continuous Integration and Continuous Deployment (CI/CD) pipelines automate various stages of the software development lifecycle, including code building, testing, deployment, and delivery [21]. Despite their widespread use, there has been limited research on leveraging LLMs to enhance CI/CD pipelines. Additionally, by studying CI jobs, we find it can be time-consuming, and developers often seek expedited results from these pipelines [7]. The second phase of this project addresses this gap by focusing on the CI/CD pipeline itself. This phase involves analyzing current CI/CD processes and investigating how LLMs can be utilized to optimize and improve these workflows.

The procedures can be divided into identifying challenges, finding solutions and generalization. We first conduct a case study following established guidelines [14] to investigate the current stage in several companies and identify common problems and areas where LLMs can be beneficial. We then provide a proposal and evaluate it to find an efficient solution. The current research focuses on finding possible errors during the whole development cycle. One major challenge we aim to address during Phase 1 was the lack of detailed analysis or suggestions after code submission, as developers only received results from the CI jobs. This made debugging a time-consuming task for developers. To resolve this, we intend to provide a solution that includes both log analysis and potential fixes.

Another challenge to be tackled during Phase 2 is that the current CI/CD pipeline contains numerous jobs, causing minor code changes to take hours, especially if failures occur in the final stages. To address this, we plan to build on our previous work and provide a more efficient solution to predict problems before submission. Following this, during Phase 3, instead of offering pre-commit predictions, we aim to provide real-time suggestions to developers.

This research begins with analyzing logs and identifying related issues to locate root causes and corresponding code. By shifting from post-commit to pre-commit analysis, we aim to shorten the development cycle by predicting potential failure points and suggesting solutions before code submission.

As illustrated in Figure 1, the ultimate goal is to develop a runtime plugin for development tools or CI/CD software that can offer real-time machine learning suggestions during the development process. By using this framework, companies can utilize and update customized models, and developers can use and evaluate the ML

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10.
https://doi.org/10.1145/3691620.3695606

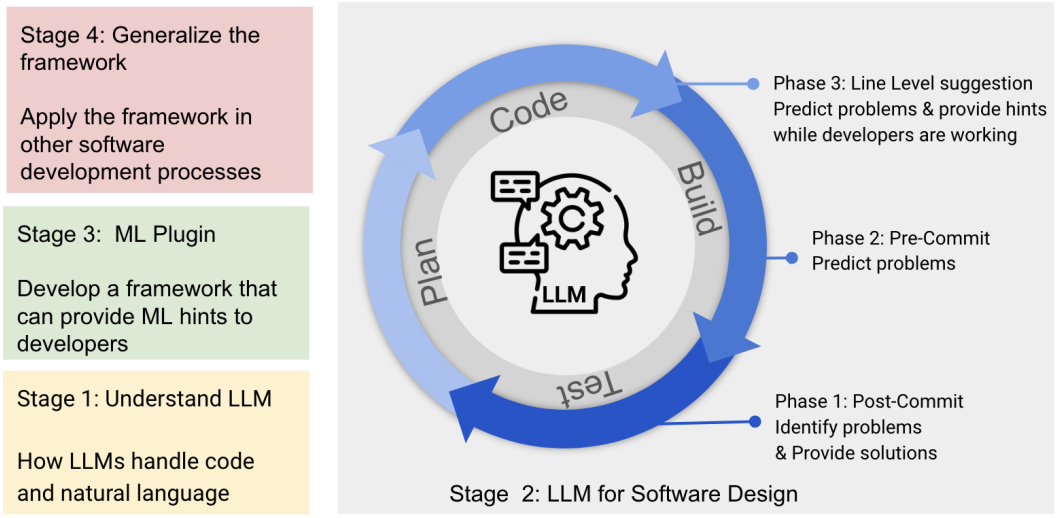

Figure 1: Research Plan: LLMs for Improving Software Design and Developer Experience

suggestions. Consequently, researchers can use this feedback as part of the evaluation data.

The fourth stage focuses on generalizing this framework to other processes. Specifically, we aim to enhance the post-test phase by improving test log analysis and re-usage. This includes auto-suggesting relevant logs when writing test cases as well as simulating new maintenance scenarios. Hundreds of logs are generated daily, including test-related and maintenance-related logs. These logs are valuable for future testing, simulation, and maintenance. We aim to provide runtime suggestions for log analysis, classification, and future usage. Our goal is to identify commonalities and differences in log analysis to meet various expectations and address related problems.

By the end of this project, the contributions are expected to be as follows:

- **Enhanced Log Analysis and Reusability**: This project aims to advance the analysis and utilization of logs generated during software development and testing. By implementing real-time log analysis and auto-suggestion features, the research seeks to improve the post-test phase and provide actionable insights for future testing, simulation, and maintenance activities.
- **Development of a Real-Time Feedback Framework**: A primary outcome will be the development of a runtime plugin for development tools or CI/CD software that provides real-time machine learning suggestions. This framework is designed to deliver prompt feedback and solutions based on log analysis, thereby facilitating more efficient problem resolution and optimizing the development workflow.
- **Refinement of User Experience**: The project aims to integrate LLMs throughout the software development lifecycle to offer early-stage or real-time insights. This integration seeks to address limitations of traditional machine learning approaches that are typically applied after code submission, thereby potentially improving the efficiency and experience of developers.

2 RELATED WORKS

As LLMs have evolved in recent years, both the technologies and the sizes of the models have advanced significantly. Transitioning from neural networks to transformers, and from millions of parameters to billions, LLMs have demonstrated their capabilities across many fields. Various aspects of software engineering have been transformed by the rapid development of LLMs and generative models.

2.1 Large Language Models (LLMs)

LLMs refer to models trained on vast amounts of data, enabling them to understand and classify or generate human-like text. The introduction of the transformer [17] mechanism and pretrain technology have significantly enhanced the capabilities of LLMs [8], prompting more researchers to explore their potential applications in the field of software engineering.

Based on their architecture, LLMs can be categorized into three general types: encoder-only, decoder-only, and encoder-decoder models. Each category has distinct characteristics and applications:

Encoder-only models: These models, such as BERT [3] and its variations, consist solely of the encoder component, which maps input sequences to low-dimensional representations. They excel in understanding the context of the text by encoding input sequences into fixed representations.

Decoder-only models: Models like GPT [1] and LLaMA [16] fall into this category. They comprise only the decoder component and are designed to generate coherent text token by token by predicting the next word in a sequence.

Encoder-decoder models: Examples include T5 [12] and BART [10]. These models integrate both encoder and decoder architectures,

combining their strengths to make them versatile for a range of tasks.

2.2 LLMs for Software Engineering

After 2017, there has been a surge in papers on LLMs, and since 2019, extensive research has focused on LLMs and their applications in software engineering [4]. In 2021, GitHub Copilot, powered by OpenAI's CodeX [2], based on the GPT-3 [1] model and fine-tuned for programming languages, significantly advanced the use of LLMs in programming. Subsequently, more research has aimed to improve code-based language models.

Researchers have discovered that LLMs can be utilized not only for programming but also in various other aspects of software development, such as requirement engineering, design, comment generation, and software testing.

Code generation is a major area in software engineering that can benefit from LLMs. These models can generate corresponding code from a natural language description. Models like Codex [2], CodeBERT [5], GPT-3 [1], Code LLaMA [13], and CodeT5 [20] have demonstrated their abilities in this domain. These models can also be fine-tuned for tasks like translating programming languages, classifying programming languages, detecting code clones, and generating test cases.

Additionally, research has explored models' abilities to understand code, performing tasks such as code summarization and code comment generation. These models can comprehend the given code and produce corresponding text.

Another significant application is generating software-related text, such as requirements, which is traditionally a time-consuming task. Using LLMs can streamline and improve this process.

3 METHODOLOGY

To integrate LLMs into the company's software development process, we propose to utilize the Model-Driven Architecture (MDA) methodology. MDA is software development method that grounded in established standards, such as the Unified Modeling Language (UML), and uses models as primary artifacts throughout the development lifecycle, which can be highly beneficial for designing, developing, and maintaining ML systems [9].

The MDA development process comprises three major steps that includes, creating a Platform-Independent Model (PIM), transforming it into a Platform-Specific Model (PSM), and finally generating executable code [9]. This methodology emphasizes the transformation from PIM to PSM, facilitating a clear delineation between system design and implementation details.

In our research, we aim to provide organization-specific or task-specific solutions tailored to different company contexts, as well as a framework capable of generalizing these solutions for broader application in the software industry. To achieve this, as depicted in Figure 2, we plan to first design a PIM based on the domain, followed by PSMs that reflect the specific settings and requirements of different companies. Subsequently, we intend to refine the initial PIM based on feedback from the PSMs, thus establishing a standardized architectural framework that can be used in future projects. This iterative loop not only bridges all PSMs but also allows for

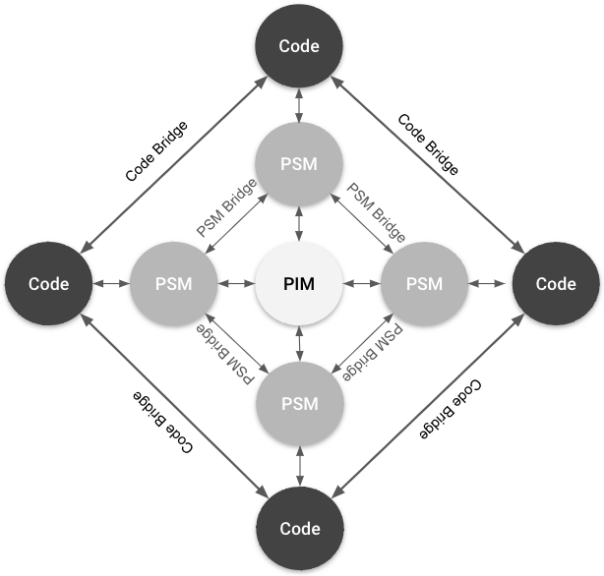

Figure 2: MDA Workflow(Image Adapted From [9])

continuous improvement and refinement of the PIM, ensuring it remains relevant and adaptable.

For each stage, the procedures can be divided into identifying challenges, finding solutions, and generalization. As mentioned earlier, we will first conduct case study research following the guidelines proposed by Runeson et al. [14] to identify common problems and challenges worth noting. Each time, we will start with a specific challenge identified within one company. To provide organization-specific or team-specific methods, we will employ action research [15], a collaborative approach involving both researchers and practitioners. This approach is known for its iterative process and one iteration includes five steps. The corresponding activities are outlined in Table 1. For each topic, we will iterate until we reach a conclusive solution. We will then attempt to generalize the solution to validate its applicability across different company settings.

4 EVALUATION

To evaluate the PSMs, computational experiments will be conducted to pre-process the data and assess the model design and training process. This includes fine-tuning the models and comparing their performance against other proposed models, thereby providing a quantitative evaluation of the proposed architecture. These evaluations will be performed during the action research iterations, facilitating iterative refinement and validation.

For the PIMs, a combination of qualitative methods will be employed, such as surveys and interviews with developers and users from the collaborating companies. This approach aims to gather feedback on the system's usability and effectiveness.

The MDA methodology will be adapted to the development of a ML system, and the action research method will be employed to iteratively refine the solution within an industrial setting. Both qualitative and quantitative methods will be utilized to evaluate

Table 1: Action Research Activities

Diagnosing	Identifying the problem in the industry. For each topic, it should be more than one cycles to revise the problem and address new problem.
Action Planning	Define an action plan outlining research goals, deliverables, and necessary actions.
Action Taking	Conducting the experiments based on the action plan and making necessary changes.
Evaluation	Evaluating the methodology proposed. Concluding if the evaluation is enough, otherwise make new diagnosis.
Learning	By generalizing the framework across different companies, we learn the commodities and differences of the task and corresponding AI-based solutions and their efficiencies.

the models and system architectures. Computational experiments will be conducted to design, develop, and quantitatively assess the performance of the models. Experimental validation will be performed to test the effectiveness of the proposed solutions and collect relevant data for analysis. Additionally, questionnaires and surveys will be administered to gather qualitative feedback from users and developers, ensuring a comprehensive evaluation of the system from multiple perspectives.

5 PROGRESS AND FUTURE WORK

Currently, we have completed the first stage, which involves evaluating the extent to which the naturalness hypothesis holds when using LLMs. Our results indicate that the capability of these models to recognize programming languages and tasks improves with their size.

The next phase of our research will focus on investigating the CI/CD workflows in collaborating companies to identify common patterns and differences. We aim to discover more areas where generative AI can enhance efficiency. Additionally, we are addressing a known challenge from our partner companies: predicting CI/CD processes to reduce potential failures and shorten the waiting time when developers submit code.

In parallel, we will explore ways to improve the use of test data, including test log data and test environment simulation. Our goal is to provide generative AI-based solutions to collaborating companies, helping them mitigate their current problems.

ACKNOWLEDGMENTS

This work is partially funded by Software Center, a collaboration between University of Gothenburg, Chalmers and 18 universities and companies – www.software-center.se.

REFERENCES

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[4] 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. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.

[5] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[6] 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).

[7] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 672–681.

[8] 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.

[9] Anneke G Kleppe, Jos B Warmer, and Wim Bast. 2003. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.

[10] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[11] Nuno Marques, Rodrigo Rocha Silva, and Jorge Bernardino. 2024. Using ChatGPT in Software Requirements Engineering: A Comprehensive Review. *Future Internet* 16, 6 (2024), 180.

[12] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[13] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[14] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.

[15] Miroslaw Staron. 2020. *Action research in software engineering*. Springer.

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[18] Jianxun Wang and Yixiang Chen. 2023. A Review on Code Generation with LLMs: Application and Evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*. IEEE, 284–289.

[19] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* (2024).

[20] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[21] Fiorella Zampetti, Salvatore Geremia, Gabriele Bavota, and Massimiliano Di Penta. 2021. CI/CD pipelines evolution and restructuring: A qualitative and quantitative study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 471–482.