Essay Topic: How will Large Language Models change how we develop software?

ABSTRACT

Large Language Models (LLMs) like GPT, CodeBERT, and Codex are changing the way software is developed, moving away from older, rule-based methods. This essay explores at how LLMs are being used in different parts of software development, such as gathering requirements, writing code, testing, and maintaining software. LLMs help automate tasks like code generation and debugging, making the process faster and more efficient. Currently, 98.25% of LLM applications focus on these areas, improving workflows and teamwork. However, issues like bias and incorrect outputs mean that human oversight is still necessary. As LLMs continue to improve, they will play an even bigger role in software development, but ethical concerns remain important.

Keywords: large language models, gpt, software development, natural language processing

I. Introduction

Recent advances in Natural Language Processing (NLP) are revolutionizing software development, shifting away from traditional rule-based systems and basic machine learning models that required extensive manual setup and had limited flexibility. A key driver of this transformation is the emergence of Large Language Models (LLMs) built on transformer architectures. These pre-trained models offer automation and enhanced intelligence by being adaptable to various tasks without the need for task-specific setups (Brown, 2020). For example, GPT-4, the fourth-generation model from OpenAI, with a larger size for better accuracy, multilingual understanding, enhanced contextual awareness, and improved reasoning abilities, making it more capable of handling complex tasks (Baktash & Dawodi, 2023).

These models are no longer limited to just natural language tasks; they can assist with writing code, debugging, generating documentation, and facilitating collaboration between developers (Ozkaya, 2023). As these capabilities continue to evolve, LLMs are poised to transform how we approach various phases of software development, streamlining workflows and reducing the manual effort traditionally required.

This essay will explore how LLMs are transforming software development and outlines the specific changes involved.

II. Discussion

Software engineers should view Large Language Models (LLMs) as powerful collaborators, enhancing their efficiency rather than replacing them. LLMs can assist with numerous tasks, from generating code to debugging and documentation, making them invaluable in many stages of the Software Development Life Cycle (SDLC). In fact, recent study found that LLMs are applied in 98.25% of use cases across key stages such as software development, maintenance, quality assurance, and requirements engineering (Hou, Zhao, Liu, & Yang, 2024). Focusing on these four areas allows us to explore how LLMs are reshaping modern software engineering workflows and increasing productivity.

A. Software Requirement Engineering

Software requirements engineering is an important step in software development to ensure products meet stakeholders' needs. Traditionally, this phase involves manually gathering, analyzing, and validating requirements, which can introduce

human errors, misunderstandings, or delays. Large Language Models (LLMs), such as GPT-4, can significantly change this process by automating and improving many of these tasks (Marques, Silva, & Bernardino, 2024).

LLMs **accelerate requirement gathering by quickly interpreting user stories, specifications, and feedback, generating clear, comprehensive lists of functional and non-functional requirements**. They synthesize insights from multiple sources, offering broader perspectives on stakeholder needs, and can produce clearer documentation, reducing misinterpretation (White, 2023). LLMs also **enhance brainstorming, suggesting innovative features and solutions** that might be overlooked, fostering creativity (Bencheikh & Höglund, 2023). However, LLMs carry risks, such as introducing biases from their training data, which may skew requirements, or generating incorrect information. These errors can have significant consequences throughout the development process (Luitel, Dipeeka, Hassani, & Sabetzadeh, 2024). Therefore, human oversight remains crucial to ensure accuracy, minimize bias, and align outputs with stakeholder goals (Ronanki, Krishna, Berger, & Horkoff, 2023). Combining LLM outputs with human expertise helps create innovative yet reliable requirements while reducing time and effort.

B. Software Development

LLMs like CodeBERT and *Codex* are transforming software development by **automating tasks** that once required significant time and expertise (Feng, Guo, Tang, & Duan, 2020). Traditionally, developers manually wrote, debugged, and optimized code, which could be slow. Now, LLMs trained on large programming datasets assist with **code generation, completion, and debugging**, simplifying the process.

One key innovation is the ability to **generate code from natural language prompts**. Developers can describe their needs in human language, and models like *Codex* translate these descriptions into working code, speeding up development and allowing teams to focus on design and problem-solving. **Real-time code completion** is another significant improvement, which boosts efficiency and reduces errors by suggesting code, allowing smoother workflows (Hou, Zhao, Liu, & Yang, 2024).

In addition to these features, LLMs **automate repetitive tasks** like writing test cases, setting up environments, and creating file structures. They also help with **code summarization and analysis**, explaining functionality and even generating flowcharts, making it easier to understand and update legacy codebases.

LLMs excel at automating debugging by identifying bugs, suggesting fixes, and refactoring code for readability and performance, which accelerates development and improves software quality.

However, LLMs may generate incorrect or inefficient code in complex scenarios, and over-reliance on them can weaken developers' critical thinking skills. So, human oversight is necessary to review and test LLM-generated outputs, ensuring accuracy and reliability.

C. Software Quality Assurance

LLMs are transforming the field of software quality assurance (QA) **by automating key testing processes, improving test coverage, and assisting in bug detection**. Traditionally, QA has been a time-intensive process, requiring manual test creation,

execution, and analysis to ensure that software functions as intended. However, LLMs, with their ability to understand both natural language and code, are significantly reducing the time and effort required for comprehensive testing (Santos, Santos, Magalhaes, & Santos, 2024).

LLMs greatly enhance software quality assurance **by automating test generation**, **increasing test coverage**, and **aiding in bug detection**. They can quickly create a wide range of test cases, from common to edge scenarios, ensuring thorough and consistent testing without manual effort. LLMs also **identify gaps in test suites and detect bugs**, suggesting fixes based on learned patterns. However, challenges like false positives and over-reliance on automation highlight the need for human oversight to ensure accuracy and relevance in testing.

D. Software Maintenance

In software maintenance, LLMs have been used for tasks like **repairing programs, reviewing code, debugging, and other related activities**.

LLM aim to automatically detect and fix bugs or defects in software by using techniques to analyse faulty code and create correct patches. For instance, ESBMC-AI, one of the best model in automating the detection and repairing issues of software, offers the potential for integration into the continuous integration and deployment (CI/CD) process within the software development lifecycle (Tihanyi, et al., 2024).

LLMs can also identify bugs by analysing execution results and explaining the generated code in natural language (Chen, Lin, Schärli, & Zhou, 2023).

III. Conclusion

In summary, Large Language Models (LLMs) are transforming the software development process by improving efficiency throughout the Software Development Life Cycle (SDLC). They automate tasks like code generation, debugging, and testing, speeding up everything from requirement gathering to development and maintenance. While LLMs boost productivity and reduce manual effort, they still have limitations, such as bias and the potential for errors, making human oversight essential. As LLMs evolve, they will play a crucial role in software engineering, balancing innovation with the need to carefully manage risks.

References

Baktash, J. A., & Dawodi, M. (2023). Gpt-4: A Review on Advancements and Opportunities in Natural Language Processing. *arXiv*.

Bencheikh, L., & Höglund, N. (2023). Exploring the Efficacy of ChatGPT in Generating Requirements: An Experimental Study. *Bachelor's Thesis, Chalmers University of Technology, Göteborg, Sweden*.

Brown, T. B. (2020). Language models are few-shot learners. *arXiv*.

Chen, M., Tworek, J., & Jun, H. (2021). Evaluating Large Language Models Trained on Code. *arXiv*.

Chen, X., Lin, M., Schärli, N., & Zhou, D. (2023). Teaching Large Language Models to Self-Debug. *arXiv*.

Feng, Z., Guo, D., Tang, D., & Duan, N. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv*.

Hou, X., Zhao, Y., Liu, Y., & Yang, Z. (2024). Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv*.

Luitel, Dipeeka, Hassani, S., & Sabetzadeh, M. (2024). Improving requirements completeness: Automated assistance through large language models. *Requirements Engineering*, 73-95.

Marques, N., Silva, R., & Bernardino, J. (2024). Using ChatGPT in Software Requirements Engineering: A Comprehensive Review. *Future Internet 2024*, 180.

Ozkaya, I. (2023). Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software*, 4-8.

Ronanki, Krishna, Berger, C., & Horkoff, J. (2023). Investigating ChatGPT's potential to assist in requirements elicitation processes. *49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 354-361). 354-361: IEEE.

Santos, R., Santos, I., Magalhaes, C., & Santos, R. d. (2024). Are We Testing or Being Tested? Exploring the Practical Applications of Large Language Models in Software Testing. *IEEE Conference on Software Testing, Verification and Validation (ICST).* Toronto, Canada: IEEE.

Thapa, C., Jang, S. I., Ahmed, M. E., Camtepe, S., Pieprzyk, J., & Nepal, S. (2022). Transformer-Based Language Models for Software Vulnerability Detection. *arXiv*.

Tihanyi, N., Jain, R., Charalambous, Y., Ferrag, M. A., Sun, Y., & Cordeiro, L. (2024). A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. *arVix*.

White, J. F. (2023). A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *arXiv*.