Software Engineering in the Era of Large Language Models: A Review

Ng Jie Sheng 243UC247D5 25%
Pan Han Cheng 243UC247BY 25%
Tee Wai Hong 243UC247C6 25%
Tey Hui Sze 243UC247CY 25%

September 5, 2025

1 Introduction and Problem Statement

The advent of Large Language Models (LLMs) has initiated a transformative phrase in software engineering (SE), significantly altering how technology is engaged by enhancing natural language processing, automating complex tasks, and supporting decision-making. (Vieira, 2025 ; Lin et al., 2024). These models, which leverage deep learning and massive datasets, demonstrate an unprecedented capacity to understand, generate, and operating programming languages, thereby assisting developers across various stages of the software development lifecycle (SDLC), including software design, automated programming, and maintenance (Gao et al.,2025 ; Vieria, 2025). The integration of LLMs within the SE landscape, often referred to as LLM4SE, represents a burgeoning trend with potential for end-to-end software development, moving beyond traditional multi-stage processes (Ga0 et al.,2025).A pivotal evolution within this landscape is the emergence of LLM-Based Multi-Agents(LMA) systems, which harness the collaborative and specialized abilities of multiple agents to address complex SE challenges, improve robustness, and provide scalable solutions for managing real-world software projects (He et al., 2025).Existing LMA systems, such as ChatDev, MetaGPT, and AgileCoder, effectively simulate human roles like generic software developers and product managers, demonstrating progress in automating software engineering tasks (He et al., 2025 ; Lin et al., 2024).
Despite the promising applications and transformative potential of LLMs and LMA systems in automating various software development activities, significant challenges remain in ensuring the quality, stability, reliability, and trustworthiness of LLM-generated software artifacts, especially when integrated holistically across entire SDLC (Gao et al., 2025 ; Vieria, 2025 ; Lin et al., 2024). These models frequently encounter limitations such as the inherent semantic gap between code and natural languages, the instability of generated results, and the constant need of updated data and models, which compromise their reliability in critical applications (Gao et al., 2025). Furthermore, current LLM applications for software development often focus on isolated tasks, lacking a comprehensive vision that prioritizes broader trustworthiness objectives (Vieria, 2025). Ensuring trustworthiness is dufficult due to the diversity, scale, and complexity of modern software systems, as well as the subjective nature of trust itself, often requiring a delicate balance of quality attributes like security, scalability, fairness, and maintainability (Vieria, 2025). While multi-agent systems have been explored, existing research often diverges form a fine-grained analysis of how different software process models and individual development activities impact various code quality metrics beyond mere functional correctness, such as code smells or exception handling (Lin et al., 2024). This gap highlights the need for a more structured approach to integrate LLM agents within established software process models to systematically improve generated code quality and stability.
The objective of this report aims to gain a comprehensive understanding and insights into chosen research area of LLM-based multi-agent systems for software engineering and the progress made within this field.The another objective is to identify and critically review existing research papers, outlining their respective problem statements, research methodologies or algorithms, results, discussions, advantages, and limitations.The third objective is to provide critical comments and suggest future enhancements to address the identified challenges and research gaps within the domain, contributing to the development of more trustworthy and effective LLM-driven software engineering solutions.
Citation example: (Vieira, 2025), (Lin, Kim, & Chen, 2025)(He, Treude, & Lo, 2025) and (Gao, Hu, Gao, Xia, & Jin, 2025).

2 Methodology

The four core research papers presented in this section employ diverse research methods to examine the contribution of large language models (LLMs) to software engineering. Together, the diverse and complementary method provides a strong evidence base for the applications, challenges, and future of software engineering LLMs. All of these approaches will be analyzed below.

2.1 Expert Workshop and Qualitative Analysis

Representative Articles: (Gao et al., 2025) The Current Challenges of Software Engineering in the Era of Large Language Models
https://dl.acm.org/doi/abs/10.1145/3712005

This study used a qualitative method based on an expert workshop. 24 academic and industry experts participated in the "9th CCF Beautiful Lake Seminars" to discuss the challenges faced in LLM4SE. The researchers used qualitative methods such as NVivo to perform open coding of the discussion and card sorting to simplify and categorize the original concepts. They eventually derived 26 key challenges from seven key areas. This method effectively brought together the consensus of domain experts and revealed the most cutting-edge and pressing real-world issues.

2.2 Systematic Literature Review

Representative Articles: (He et al., 2025) LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead
https://dl.acm.org/doi/10.1145/3712003

This article describes a systematic planning, searching, screening, and synthesis process to identify, evaluate, and interpret all available evidence relevant to a specific research question. The research team mainly uses the DBLP database for keyword-based searches, one set covering terms such as "Agent" and "LLM", and another set targeting specific SE activities such as requirements, coding, testing, and maintenance. To ensure the quality of the literature, the research team established strict inclusion and exclusion criteria, and ultimately 41 articles were obtained from the initial search, and 30 more articles were added through the snowball method, for a total of 71 articles.
https://dl.acm.org/doi/10.1145/3712003

2.3 Conceptual and Framework Research

Representative Articles: (Vieira, 2025) Leveraging LLMs for Trustworthy Software Engineering: Insights and Challenges
https://ieeexplore.ieee.org/document/11052849

This study adopted a research approach of theory construction and conceptual analysis. Its core is to deconstruct and reintegrate the two key concepts of "LLM" and "software credibility". (Vieira, 2025) study constructed a systematic conceptual framework to explain how LLM can be used to embed and ensure the trustworthiness properties of software in various stages of the software development life cycle (SDLC),from requirements, design, development, deployment, to assessment. The final output is a detailed, structured conceptual framework that depicts a comprehensive, trustworthy software development blueprint driven by LLM, rather than a set of testable hypotheses or empirical data. Its main purpose is to inspire new research ideas and guide the development of future tools and practices.

2.4 Empirical Evaluation & Multi-Agent Framework Construction

Representative Articles: (Lin et al., 2025) SOEN-101: Code Generation by Emulating Software Process Models Using Large Language Model Agent
https://ieeexplore.ieee.org/document/11029771

The study adopted an approach that combined design science research with empirical evaluation. FlowGen is a code generation framework based on multi-agent large language models. In FlowGen, different LLM agents such as requirement engineers, software architects, developers, and testers are created to simulate three software process models such as FlowGenWaterfall, FlowGenTDD, and FlowGenScrum. The agent then goes through a self-improvement cycle (three times) until it produces better results. They then conducted experiments on four standard code generation benchmark datasets (HumanEval, HumanEval-ET, MBPP, MBPP-ET), using Pass@1 as the core to evaluate the impact of different process models on the functional correctness of the code. In addition, the study also used a static code analysis tool (Pylint) to detect code smells and count exception handling.

3 Result

3.1

The article “The Current Challenges of Software Engineering in the Era of Large Language Models” presented findings from a qualitative study involving 24 participants, identifying 26 key challenges for LLM-based SE (LLM4SE) across seven aspects, including requirement and design, coding assistance, testing code generation, code review, software maintenance, software vulnerability management, and data training and evaluation. Specific challenges include inaccurate generated code due to LLM hallucination, the introduction of security vulnerabilities from training on open source repositories, limited performance for new programming languages, insufficient evaluation methods for machine-generated code, difficulties in integrating generated code into projects, and syntactical and semantic challenges in testing code generation. Challenges also include the lack of high-quality code review data, the gap between industry and open source code review practices, and the absence of end-to-end automation in code review. In software maintenance, issues arise from the complexity of service dependencies in microservice architectures, the lack of high-quality private operational data, and insufficient interpretability and reliability of operational information. For vulnerability management, key challenges include LLMs’ limited understanding of vulnerability information due to scarce training data, the scarcity of high-quality analysis and explanation data, and the complexity of vulnerability context exceeding LLM window size.

3.2

The study of “LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead” conducted a systematic review of 71 recent primary studies on LLM-Based Multi-Agent (LMA) systems in Software Engineering (SE) across the software software development lifecycle. Its case studies using ChatDev, powered by GPT-3.5-turbo, showed that LMA systems performed strongly in moderately complex tasks, successfully producing a Snake game meeting all requirements in the second attempt, taking an average of 76 seconds and costing $0.019 per attempt. However, for more complex challenges like a Tetris game, ChatDev struggled, only producing a playable vision on the tenth attempt that lacked core functionality, such as removing completed rows,despite remaining efficient and cost-effective at 70 seconds and $0.020 per attempt. This highlighted that current LMA systems are suitable for moderately complex tasks but have limitations in handling highly complex tasks requiring deeper logical reasoning and abstraction.

3.3

Furthermore, “Leveraging LLMs for Trustworthy Software Engineering Insights and Challenges” identified crucial challenges for integrating LLMs in trustworthy software engineering, such as integration with established practices, accuracy and reliability (due to probabilistic outputs), bias migration, explainability and interpretability, scalability and integration with large-scale/legacy systems, adherence to standards and regulations, real-time adaptability in CI/CD environments and ethics and privacy concerns.

3.4

Lastly, "Code Generation by Emulating Software Process Models Using Large Language Model Agents" evaluated FlowGen, a multi-agent LLM-based code generation framework. It found that FlowGenScrum consistently outperformed RawGPT (direct ChatGPT use) by 5.2% to 31.5% in Pass@1 accuracy across benchmarks (HumanEval, HumanEval-ET, MBPP, MBPP-ET) and showed the most stable results with the lowest standard deviation (average 1.2%). The study revealed that testing had the most significant impact on functional correctness, with its removal causing a densities. Design and code review activities were found to significantly improve reliability by increasing the density of handled exceptions and reducing refractor and warning code smell. FlowGen models maintained stable Pass@1 results across GPT-3.5 versions and temperature values, unlike RawGPT which showed large fluctuations. When compared to other techniques, FlowGenScrum and Code T achieved similar, statistically significant higher Pass@1 results than Reflexion, and integrating CodeT into FlowGenScrum further improved Pass@1 by up to 5%, achieving the highest scores.

Model	HumanEval Pass@1 (Avg ± StdDev)	HumanEval-ET Pass@1 (Avg ± StdDev)	MBPP Pass@1 (Avg ± StdDev)	MBPP-ET Pass@1 (Avg ± StdDev)	Key Findings
RawGPT	64.4 ± 3.7	49.8 ± 3.0	77.5 ± 0.8	53.9 ± 0.7	Baseline for comparison. Highest standard deviation (average 2%)
FlowGenWaterfall	69.5 ± 2.3 (+7.9%)*	59.4 ± 2.5 (+19.2%)*	76.3 ± 0.9 (-1.5%)	51.1 ± 1.7 (-5.2%)	Shows statistically significant improvement over RawGPT for HumanEval and HumanEval-ET.
FlowGenTDD	69.8 ± 2.2 (+8.4%)*	60.0 ± 2.1 (+20.5%)*	76.8 ± 0.9 (-1.0%)	52.8 ± 0.7 (-2.1%)	Shows statistically significant improvement over RawGPT for HumanEval and HumanEval-ET.
FlowGenScrum	75.2 ± 1.1 (+16.8%)*	65.5 ± 1.9 (+31.5%)*	82.5 ± 0.6 (+6.5%)*	56.7 ± 1.4 (+5.2%)*	Consistently outperforms RawGPT by 5.2% to 31.5% in Pass@1, with statistically significant improvements across all benchmarks. Achieves the most stable results with the lowest standard deviation (average 1.2%).

4 Discussions

Based on the results above, the integration of Large Language Models (LLMs) into Software Engineering (SE) presents a landscape of significant potential tempered by substantial, multifaceted challenges. While LLMs demonstrate efficiency and capability in certain tasks, their current limitations in complex reasoning, reliability, and integration necessitate structured, process-oriented approaches for effective and trustworthy implementation.

4.1 Current Capabilities and Performance

Current research indicates that LLM-based systems are effective for tasks of moderate complexity. A study on the LLM-Based Multi-Agent (LMA) system ChatDev found it could successfully generate a functional Snake game quickly and cost-effectively. However, its performance faltered when tasked with a more complex Tetris game, failing to implement core logic even after multiple attempts. This highlights a critical limitation: current LLMs struggle with tasks requiring deeper logical reasoning and abstraction.

Despite this, the way LLMs are utilized significantly impacts their performance. The FlowGen framework, which uses LLM agents to emulate established software process models like Scrum, demonstrates a clear path to improving results. FlowGenScrum consistently outperformed direct LLM use (RawGPT) by a significant margin (5.2

4.2 Pervasive Challenges Across the Software Lifecycle

The challenges of applying LLMs in software engineering (LLM4SE) are widespread, impacting nearly every phase of the development lifecycle. A primary concern is the accuracy and reliability of generated code, which is susceptible to LLM "hallucination" and can introduce subtle security vulnerabilities learned from open-source training data.

Beyond simple code generation, specific challenges emerge in key areas:

- Testing and Code Review: LLMs face syntactical and semantic difficulties in generating effective test cases. Furthermore, the lack of high-quality code review data for training and the absence of end-to-end automation hinder their utility in ensuring code quality.
- Software Maintenance: The complexity of modern microservice architectures and the lack of high-quality private operational data for training limit the ability of LLMs to provide reliable and interpretable maintenance support.
- Vulnerability Management: LLMs often have a limited understanding of vulnerability context, a problem exacerbated by scarce training data and the constraints of model context windows.
- Integration and Trust: Broader, systemic challenges include integrating LLMs with established SE practices and legacy systems, ensuring their outputs are explainable, mitigating bias, adhering to regulatory standards, and addressing significant ethics and privacy concerns.

4.3 The Road Ahead: Process Emulation and Synergistic Integration

The most promising direction for overcoming these challenges appears to be the development of sophisticated, multi-agent systems that structure LLM interactions. The success of FlowGen reveals that emulating proven software process models is key. Within these frameworks, specific SE activities have a profound impact. The FlowGen study identified that the testing agent had the most significant impact on functional correctness, while design and code review agents were crucial for improving reliability by handling exceptions and reducing code smells.

This approach not only boosts accuracy but also enhances stability and reliability while mitigating the unpredictable nature of LLMs. Moreover, the finding shows that integrating specialized techniques (like CodeT) into the FlowGenScrum framework further improved performance which suggests that synergistic, hybrid models will likely achieve the best results. Ultimately, successfully leveraging LLMs in software engineering requires moving beyond simple prompting and toward creating holistic, process-aware systems that guide LLM agents through structured workflows, mirroring the discipline and rigor of traditional software development.

5 Conclusion

This report explores how the emergence and dominance of Large Language Models (LLMs) has changed the world of software engineering. First, it has been shown that large language models are capable of handling moderately complex tasks within minutes at very low costs. Despite that, it has also been proven that LLMs can produce faulty software plagued with securities vulnerability due to it not recognizing vulnerability context and low training data, which can cause a lack of trustworthiness and permeate the entire software development lifecycle.

The most crucial insight from the research is that the key to overcoming these limitations are to apply LLMs in a structured and disciplined manner. The success of frameworks like FlowGen, which employ multi-agent systems to emulate established software process models like Scrum, provides a clear path forward. By assigning specialized roles to different agents and guiding them through a structured workflow, this approach systematically enhances functional correctness, improves code quality by reducing smells and increasing exception handling and ensures greater stability and reliability.

In essence, the future of effective LLM-driven software engineering depends on shifting the focus from isolated, prompt-based interactions to the creation of holistic, process-aware systems. This paradigm of embedding LLMs within the proven rigor of traditional software development methodologies is essential for mitigating their inherent weaknesses, harnessing their full potential, and ultimately building the next generation of trustworthy, high-quality software.

6 Future work

Based on the four articles, the various directions that we can take for the future Large Language Models (LLMs) research are apparent.

Four directions that can do:

1. **Trustworthiness And Security**
   In future research, we can further study how to ensure the trustworthiness and security of LLM-generated code, including whether the LLM-generated code contains bias, security vulnerabilities, and intellectual property issues (e.g., SQL injection, memory leaks), as well as whether it complies with coding standards. This includes developing new methods to verify these quality attributes of LLM output.

2. **Evaluation And Benchmarking**
   In future research we need to establish a new, comprehensive, and standardized benchmark that accurately reflects real-world software engineering tasks.These new benchmarks should move beyond simple code generation problems (like those in HumanEval) to include tasks that require multi-step reasoning, logical complexity, and an understanding of system-level architecture. They should also evaluate the non-functional qualities of the generated code, such as efficiency, maintainability, and security. Research is also needed to develop robust metrics for evaluating multi-agent systems, including metrics for agent collaboration effectiveness, communication overhead, and robustness to failure.

3. **Improving LLM Agent Capabilities And Optimizing Multi-Agent Systems**
   In future research, we can focus on improving the capabilities of individual LLM agents, especially in logical reasoning, to better cope with the complexity of large software projects. This could involve exploring new hinting techniques, model architectures, or fine-tuning strategies to help models handle more complex, multi-layered problems beyond simple task automation. Furthermore, we could investigate the effectiveness of LLM multi-agent systems (LMAs), which hinge on their collaborative capabilities. For example, we could investigate optimal communication protocols, role allocation strategies, and cross-examination methods to minimize hallucinations and improve overall output. The goal is to enable LMAs to autonomously and reliably handle entire software projects from start to finish.

4. **Explore More Research Directions**
   In the future, we need to conduct more research to explore how to utilize LLM agents to simulate other traditional and agile software process models to incorporate enhancements into each development activity. Examples include Lean, Kanban, and even specific DevOps pipelines. This research will explore how these different processes affect the quality, efficiency, and maintenance of software produced by LLM.

References

Gao, C., Hu, X., Gao, S., Xia, X., & Jin, Z. (2025, May). The current challenges of software engineering in the era of large language models. *ACM Trans. Softw. Eng. Methodol.*, *34*(5). Retrieved from `https://doi.org/10.1145/3712005` doi: 10.1145/3712005

He, J., Treude, C., & Lo, D. (2025, May). Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Trans. Softw. Eng. Methodol.*, *34*(5). Retrieved from `https://doi.org/10.1145/3712003` doi: 10.1145/3712003

Lin, F., Kim, D. J., & Chen, T.-H. (2025). Soen-101: Code generation by emulating software process models using large language model agents. In *2025 ieee/acm 47th international conference on software engineering (icse)* (p. 1527-1539). doi: 10.1109/ICSE55347.2025.00140

Vieira, M. (2025). Leveraging llms for trustworthy software engineering: Insights and challenges. *Computer*, *58*(7), 79-90. doi: 10.1109/MC.2025.3546204