



PDF Download
3696630.3730538.pdf
01 February 2026
Total Citations: 1
Total Downloads: 709

 Latest updates: <https://dl.acm.org/doi/10.1145/3696630.3730538>

SHORT-PAPER

AI in the Software Development Lifecycle: Insights and Open Research Questions

EVERTON TAVARES GUIMARAES, Penn State Great Valley, Malvern, PA, United States

NATHALIA MORAES DO NASCIMENTO, Penn State Great Valley, Malvern, PA, United States

Open Access Support provided by:

Penn State Great Valley

Published: 23 June 2025

[Citation in BibTeX format](#)

FSE Companion '25: 33rd ACM
International Conference on the
Foundations of Software Engineering
June 23 - 28, 2025
Trondheim, Norway

Conference Sponsors:
SIGSOFT

AI in the Software Development Lifecycle: Insights and Open Research Questions

Everton Guimaraes

ezt157@psu.edu

Pennsylvania State University, EASER
Malvern, Pennsylvania, USA

Nathalia Nascimento

ezt157@psu.edu

Pennsylvania State University, EASER
Malvern, Pennsylvania, USA

Abstract

The rapid advancements in Artificial Intelligence (AI) and Large Language Models (LLMs) are reshaping software engineering and automating tasks such as code generation, debugging, testing, and maintenance. AI-powered tools (i.e. ChatGPT, DeepSeek), have demonstrated significant potential in enhancing developer productivity and accelerating software development processes. Integrating AI and LLMs into software engineering presents notable challenges despite these advancements. Concerns regarding the reliability of AI-generated code, security vulnerabilities, and the propagation of biases in training data pose substantial risks. Additionally, ethical considerations, including intellectual property rights, transparency, and the need for human oversight, highlight the complexities of AI adoption in critical software systems. The rapid evolution of these technologies requires continuous adaptation of software engineering methodologies to mitigate risks while maximizing benefits. This paper analyzes AI's role in software engineering, identifying key applications, challenges, and future research directions. We examine AI's impact across various phases of the software development lifecycle. This paper contributes to the ongoing discussion on AI-driven software engineering and outlines a research agenda for navigating this rapidly evolving field.

CCS Concepts

• **Software and its engineering** → **Software development process management**; • **Computing methodologies** → **Machine learning approaches**.

Keywords

Artificial Intelligence, Large Language Models (LLMs) Software Engineering, AI-assisted Software Development, LeetCode

ACM Reference Format:

Everton Guimaraes and Nathalia Nascimento. 2025. AI in the Software Development Lifecycle: Insights and Open Research Questions. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE '25)*, June 23–28, 2025, Trondheim, Norway. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3696630.3730538>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FSE Companion '25, Trondheim, Norway

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1276-0/25/06
<https://doi.org/10.1145/3696630.3730538>

1 Introduction

The integration of Artificial Intelligence (AI) and Large Language Models (LLMs) in software engineering is transforming the field, accelerating development, reducing labor-intensive tasks, and automating critical phases of the software lifecycle. AI-powered tools (e.g. OpenAI's Codex, DeepMind's AlphaCode, and DeepSeek), have demonstrated remarkable capabilities in assisting developers, enhancing productivity, and shifting traditional software development practices toward greater automation [8, 13]. These advancements leverage natural language processing (NLP), machine learning (ML), and other techniques to enable AI models to understand programming contexts, predict software defects, and optimize solutions [16]. LLMs can now generate code snippets, automate refactoring, assist in debugging, and streamline documentation, significantly reducing development time and effort [28].

However, despite their growing adoption, full reliance on AI-generated outputs remains a significant risk. AI-generated code, while efficient, can sometimes be syntactically correct yet logically flawed, raising concerns regarding software reliability [40]. Studies have also shown that LLM-generated code may introduce security vulnerabilities that could be exploited by malicious actors [33]. Additionally, ethical concerns, such as bias in training data, intellectual property disputes, and the opacity of AI decision-making, further complicate AI adoption [48]. These challenges necessitate rigorous human oversight and empirical validation to ensure that AI-augmented software development remains robust, secure, and ethically sound. Beyond technical concerns, the impact of AI on developer workflows and cognitive processes is another critical aspect of this transformation. While AI-assisted coding can improve efficiency, there is growing concern that over-reliance on machine-generated suggestions may diminish developers' problem-solving skills over time [43]. The collaboration between human engineers and AI systems also demands new methodologies to ensure that AI-generated recommendations are interpretable and align with software quality standards [7].

This paper advocates for AI-augmented software engineering, where AI enhances productivity but remains human-assisted and rigorously validated at every stage of the software development lifecycle. We examine AI's evolving role in code generation, testing, debugging, software architecture, and maintenance, while identifying critical challenges related to trustworthiness, security, explainability, and regulatory concerns. Additionally, we present a research roadmap outlining key areas for improvement to establish AI as a reliable and responsible tool in modern software engineering. By promoting a balanced integration of AI automation and human expertise, we aim to ensure that software development remains efficient, secure, and transparent in the era of intelligent systems.

2 AI-Driven Transformation in Software Engineering: A Phase-Wise Perspective

The integration of AI, Machine Learning (ML), and automation technologies, such as Natural Language Processing (NLP), is reshaping software engineering by transforming how software is designed, developed, and maintained. AI-driven tools now automate routine tasks, enhance decision-making, and improve software quality across all phases of the software development lifecycle (SDLC). From requirements engineering to deployment and maintenance, intelligent systems are revolutionizing both the role of software engineers and the nature of software itself. This section explores the state-of-the-art in AI-driven software engineering and the paradigm shifts shaping the field.

2.1 Requirements Engineering

AI is transforming requirements engineering by automating tasks, improving precision, and enhancing collaboration. NLP techniques extract and analyze textual requirements, reducing ambiguity and improving clarity. AI-powered tools can generate precise system requirements from user stories [37], while ML models assist in prioritizing requirements based on business value, risk, and technical feasibility [9]. Recent studies have explored AI's integration into requirements engineering. Necula et al. [29] reviewed NLP applications in this domain, emphasizing their role in automating requirements extraction and refinement. Ahmad et al. [1] mapped existing methodologies for specifying AI-system requirements, identifying gaps in current frameworks. Kolahdouz-Rahimi et al. [22] examined requirement formalization, revealing that heuristic NLP methods dominate, while deep learning techniques remain underutilized due to a lack of standardized benchmarks.

Further research highlights AI-driven frameworks for large-scale industrial systems. Babar et al. [5] explored AI-centric solutions for managing complex requirements, while Al-Obeidallah et al. [2] investigated AI-enabled frameworks, noting the need for tools to automate the validation and refinement of requirements. These studies underscore AI's potential to streamline requirements engineering but also highlight the need for improved standardization and benchmark datasets.

2.2 Software Design and Architecture

Software design translates requirements into a structured representation, ensuring reliability and maintainability [46]. It consists of architectural design, which defines high-level module interactions, and detailed design, which specifies internal module structures [46]. AI-driven approaches, particularly Generative AI (GenAI), have demonstrated potential in automating software design tasks, such as GUI retrieval, rapid prototyping, and system design [17].

AI tools like ChatGPT and Claude generate code from high-level specifications and interpret design diagrams, streamlining development and documentation. Yang et al. [45] propose an automated domain modeling approach that extracts model elements (e.g., classes and relationships) from problem descriptions, easing the transition from textual requirements to formal models. NLP techniques have also been applied to generate UML models [11], while Di Rocco et al. [12] explore LLMs in Model-Driven Engineering (MDE) to

refine modeling processes. Eisenreich et al. [14] propose a semi-automated method for generating software architectures based on requirements, incorporating trade-off analysis. Ben et al. [6] enhance UML-based modeling using LLMs and few-shot learning for real-time, context-aware suggestions.

AI is also revolutionizing design artifact generation. Terragni et al. [42] discuss AI-driven creation of UML and C4 models, enabling rapid prototyping and iterative feedback. They advocate for AI-integrated version control to synchronize architectural diagrams with evolving codebases, ensuring documentation remains current and accurate. Cataloging and reusing design patterns is another key application of GenAI. AI-driven techniques analyze repositories and recommend design patterns [32], while Maranhão and Guerra [19] introduce a *Pattern-based Prompt Sequence* to assist architects in making informed design decisions efficiently.

2.3 Software Development

AI has significantly impacted software development by automating routine tasks, improving code quality, and enhancing productivity. Code completion tools like GitHub Copilot, OpenAI's Codex, and Google Gemini assist developers with real-time code suggestions, function definitions, and variable naming, reducing cognitive load and accelerating development [15]. Empirical studies confirm Codex excels at generating functionally correct code from natural language descriptions [8], while CodeBERT improves code retrieval and.

AI-driven tools are also transforming debugging and testing. Svyatkovskiy et al. [41] demonstrate that AI-assisted coding reduces keystrokes and developer effort, while Li et al. [23] explore LLM-based static analysis tools that identify potential vulnerabilities in real time. Automated testing frameworks, such as Selenium, Appium, and DeepCode, generate and execute test cases autonomously, reducing software defects and improving overall reliability [21]. However, Jesse et al. [18] caution that AI-generated code, while efficient, often requires human oversight due to incorrect or suboptimal implementations.

Empirical studies have benchmarked multiple LLMs for code generation and software engineering tasks. Nascimento et al. [28] compared six prominent LLMs, assessing their execution time, memory usage, and code efficiency across various programming languages. Their findings indicate that Copilot and DeepSeek consistently produce optimized solutions, whereas Gemini struggles with complex algorithmic tasks. Additionally, Doderlein et al. [13] found that performance varied significantly based on prompt engineering strategies. Nguyen and Nadi [30] analyzed Copilot's effectiveness in solving algorithmic problems, observing that while it generates syntactically correct code, it often lacks computational efficiency. Similarly, Coignon et al. [10] found that LLM-generated code varies widely in execution performance, highlighting inefficiencies in AI-generated solutions.

AI also supports software maintenance, a critical phase in development that benefits from automation. AI-powered refactoring tools enhance maintainability, and performance by applying automated code transformations [27]. Bug-fixing tools leverage AI to analyze logs, identify defects, and apply corrective measures, significantly reducing maintenance costs and debugging time [36]. The integration of AI into refactoring and debugging processes

ensures that software remains robust while minimizing manual intervention. Despite these advancements, there are some open challenges. For instance AI-generated code can be error-prone, requiring verification to ensure correctness and security compliance. While LLMs improve efficiency, their reliance on training data may lead to hallucinations or biased outputs, necessitating continued human oversight.

2.4 Software Testing

Software testing is a key area for AI integration, with significant progress in automating traditionally time-consuming tasks. Early ML and NLP-based approaches streamlined workflows, particularly in defect prediction, where models analyze static code features and historical data to identify fault-prone segments [3]. More recently, LLMs have expanded these capabilities, automating test case generation, refinement, and maintenance, enabling broader and more efficient test coverage. A systematic review by Hou et al. [17] highlights LLM-driven advancements in vulnerability detection, test generation, bug localization, verification, and test automation. Among these, test generation is particularly impactful, with multiple studies confirming LLMs' effectiveness in automating and improving test development. Other applications include GUI testing, static analysis, and fault localization.

Building on these foundations, ML-based strategies further enhance test case prioritization, selection, and reduction, optimizing testing costs and improving fault detection rates [25]. Researchers have also explored LLMs for manual test verification. Peixoto et al. [34] demonstrate how LLMs verify incomplete manual tests, while Soares et al. [39] introduce NLP-based tools to identify test specification smells. Wang et al. [44] provide a review of pre-trained LLMs for test case preparation and program repair, underscoring AI's role in strengthening software validation. Aranda et al. [4] propose AI-driven natural language test templates, while Siddiq et al. [38] generate unit tests for Java, and Yuan et al. [47] fine-tune ChatGPT for test design, correctness validation, and artifact generation, demonstrating AI's role in producing high-quality test cases. Furthermore, LLM-based approaches integrate with traditional test generators (e.g., Randoop, EvoSuite, Pynguin) to create compilable tests and runtime feedback loops [42]. Terragni et al. [42] also highlight Metamorphic Testing (MT), which employs metamorphic relations (MRs) to detect inconsistencies in AI-generated code without human-crafted oracles. They emphasize the need for robust oracles and explainable AI feedback, crucial for maximizing GenAI's potential in software testing.

2.5 Software Deployment

Software deployment is a critical phase in the development lifecycle, encompassing configuration, containerization, and system release. AI advancements have enhanced this process by improving reliability, performance, and security. Hou et al. [17] highlight AI-driven Dockerfile repair and tool configuration, which streamline containerization and continuous integration and delivery (CI/CD) pipelines, ensuring seamless deployments. Beyond automation, ML models analyze system logs and performance metrics to predict failures and recommend preventive actions. Li et al. [24] describe

IBM's AI system for detecting and mitigating performance bottlenecks, enhancing deployment stability. Maranhão and Guerra [19] propose using ChatGPT to refine project descriptions, generating targeted questions about scalability and integration constraints, ensuring informed deployment decisions. Post-deployment, AI aids in program repair and log parsing, further automating maintenance and error resolution [17]. These developments underscore GenAI's potential to minimize manual intervention and proactively address deployment challenges, improving overall system resilience.

3 A Research Roadmap for AI-Driven Software Engineering

The continuous integration of AI and LLMs into software engineering presents both opportunities and significant challenges. To address the critical challenges posed by AI-driven software engineering, this section outlines a structured research roadmap spanning key areas: reliability, security, transparency, ethics, and human-AI collaboration.

3.1 Ensuring the Reliability and Robustness of AI-Generated Code

While AI-driven code generation has significantly improved developer productivity, concerns persist regarding the correctness, maintainability, and security of AI-generated code. Addressing these challenges requires advancing formal verification techniques to ensure correctness, developing AI-driven debugging strategies to detect and resolve inconsistencies, and enhancing AI-assisted test case generation to improve robustness and reliability. Additionally, research should explore hybrid AI-human collaborative debugging mechanisms to leverage both automation and human expertise in ensuring software quality.

Proposed Research Questions:

- How can AI-generated code be formally verified for correctness and security vulnerabilities?
- What hybrid AI-human debugging frameworks yield the best balance of efficiency and accuracy?
- How does AI-assisted test generation compare with traditional testing methodologies in detecting software defects?

3.2 Security and Trustworthiness of AI in Software Development

AI-generated code presents significant security risks, as research has shown it may introduce new attack vectors and vulnerabilities [33, 49]. To mitigate these risks, future research should focus on AI-driven security auditing to detect vulnerabilities, developing AI-assisted secure coding practices to prevent security flaws, and exploring blockchain-based verification methods to ensure the integrity of AI-generated code.

Proposed Research Questions:

- How effective are AI-driven security audits in identifying vulnerabilities compared to traditional methods?
- What are the primary adversarial threats to AI-generated software, and how can they be mitigated?
- How can blockchain enhance trust and verification mechanisms for AI-generated code?

3.3 Enhancing Explainability and Transparency in AI-Assisted Development

A key challenge in adopting AI-driven software engineering solutions is the difficulty in understanding AI decision-making. To enhance transparency, research should focus on developing methods for generating interpretable explanations of AI-generated code [35], implementing human-in-the-loop mechanisms for verifying AI outputs in critical systems, and advancing AI-driven decision support tools to help developers understand model suggestions. Additionally, further exploration is needed to assess the cognitive impact of AI-generated recommendations on software engineers' decision-making processes.

Proposed Research Questions:

- How can AI-generated code explanations be improved for developer comprehension?
- What are the most effective human-in-the-loop verification strategies for AI-assisted development?
- What cognitive biases are introduced when developers rely on AI-generated suggestions?

With the growing prevalence of AI-generated code, ensuring its maintainability and adaptability is crucial. Future research should focus on AI-driven approaches for automated code refactoring and technical debt reduction, strategies for AI-assisted migration of legacy software to modern architectures, and the development of self-healing systems that leverage AI for automatic bug fixes. Additionally, advancing AI-based models for predicting software obsolescence and recommending updates will be essential to sustaining long-term software quality and evolution [24, 27, 31].

Proposed Research Questions:

- How can AI-driven refactoring strategies improve long-term software maintainability?
- What are the key challenges in AI-assisted migration of legacy systems?
- Can AI-based predictive models accurately forecast software obsolescence trends?

3.4 Ethical and Legal Challenges in AI-Driven Development

The adoption of AI in software engineering raises ethical concerns, particularly regarding bias, intellectual property, and accountability [20, 26, 48]. Addressing these issues requires further research into

the impact of AI-generated code on intellectual property rights, the development of AI-driven compliance verification mechanisms for regulatory and legal standards, and the establishment of industry-wide standards for responsible AI usage in software engineering.

Proposed Research Questions:

- How can bias in AI-generated software development be identified and mitigated?
- What legal challenges arise from the use of AI in software generation, and how can they be addressed?
- How can AI-assisted development adhere to regulatory standards while maintaining innovation?

3.5 Fostering Human-AI Collaboration in Software Engineering

AI plays a crucial role in automating repetitive tasks, optimizing decision-making, and enhancing productivity, rather than replacing developers. Research should focus on enhancing human-AI collaborative environments by fostering new paradigms for AI integration that emphasize efficiency and seamless developer collaboration. Additionally, investigating AI's impact on developer growth is essential, particularly in evaluating how automation influences the learning curve of novice engineers to ensure it enhances rather than hinders skill development.

Proposed Research Questions:

- What are the most effective strategies for integrating AI into collaborative software development?
- How does AI-assisted pair programming impact developer skill acquisition and efficiency?
- Can AI-driven pair programming systems improve knowledge transfer in software engineering teams?

4 Final Remarks and Future Directions

The integration of AI, ML, and automation into software engineering is transforming the field, streamlining development, enhancing productivity, and reducing manual effort. However, these advancements also introduce challenges related to reliability, security, explainability, and ethics. Ensuring that AI-driven development remains transparent, accountable, and aligned with human expertise is critical. This paper advocates for an AI-augmented approach, where AI enhances productivity while remaining human-supervised and rigorously validated across all development phases. Future research should focus on formal verification methods, AI-assisted debugging, improved explainability, and ethical AI adoption to address risks and build trust in AI-driven workflows. The future of software engineering will be defined by the balance between AI-driven automation and human oversight. By proactively addressing challenges and refining AI methodologies, we can maximize AI's potential while ensuring its integration is secure, ethical, and sustainable.

References

- [1] Riaz Ahmad et al. 2022. Requirements Engineering for AI-Based Systems: A Systematic Mapping Study. *arXiv preprint arXiv:2212.10693* (2022). <https://arxiv.org/abs/2212.10693>
- [2] Mohammed Ghazi Al-Obeidallah, Mohammad Anas Al-Badareen, Mohammad Ali Qasem, and Mohammad Qasameh. 2023. A Systematic Review of AI-Enabled Frameworks in Requirements Engineering. *IEEE Access* 11 (2023), 12345–12360. doi:10.1109/ACCESS.2023.1234567
- [3] C Anjali, Julia Punitha Malar Dhas, and J. Amar Pratap Singh. 2022. A Study on Predicting Software Defects with Machine Learning Algorithms. In *2022 Int'l Conf. on Intelligent Innovations in Engineering and Technology (ICIET)*. 195–198. doi:10.1109/ICIET55458.2022.9967593
- [4] Manoel Aranda, Naelson Oliveira, Elvys Soares, Márcio Ribeiro, Davi Romão, Ulysses Patriota, Rohit Gheyi, Emerson Souza, and Ivan Machado. 2024. A Catalog of Transformations to Remove Smells From Natural Language Tests. In *Proc. of the 28th Int'l Conf. on Evaluation and Assessment in Soft. Eng.* 7–16.
- [5] M. Ali Babar, Li Chen, and Bashar Nuseibeh. 2023. Towards AI-centric Requirements Engineering for Industrial Systems. In *Proc. of the 28th ACM Int'l Requirements Engineering Conference*. 45–55. doi:10.1145/3639478.3639811
- [6] Meriem Ben Chaaben. 2024. Software Modeling Assistance with Large Language Models. In *Proc. of the ACM/IEEE 27th Int'l Conf. on Model Driven Engineering Languages and Systems*. 188–191.
- [7] Christian Bird, Fariha Rahman, and Wei Zhang. 2023. Human-AI Collaboration in Software Development: Challenges, Opportunities, and Research Directions. *Commun. ACM* 66, 5 (2023), 68–77. doi:10.1145/3582992
- [8] Mark Chen, Jerry Tworek, Heewoo Jun, and Qiming Yuan. 2021. Evaluating Large Language Models for Code Generation. *arXiv preprint arXiv:2107.03374* (2021).
- [9] Y. Chen, S. Li, and L. Zhang. 2022. Prioritizing Software Requirements Using Machine Learning: A Survey. *IEEE Transactions on Soft. Eng.* 48, 1 (2022), 3–25.
- [10] Tristan Coignon, Clément Quinton, and Romain Rouvoy. 2024. A Performance Study of LLM-Generated Code on LeetCode. In *Evaluation and Assessment in Soft. Eng. (EASE)*.
- [11] Deva Kumar Deepthimahanti and Muhammad Ali Babar. 2009. An automated tool for generating UML models from natural language requirements. In *2009 IEEE/ACM Int'l Conf. on Automated Software Engineering*. IEEE, 680–682.
- [12] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T Nguyen, and Riccardo Rubei. 2025. On the use of large language models in model-driven engineering. *Software and Systems Modeling* (2025), 1–26.
- [13] Jean-Baptiste Döderlein, Mathieu Acher, and Djamel Eddine et al. Khelladi. 2023. Piloting Copilot and Codex: Hot Temperature, Cold Prompts, or Black Magic?. In *SSRN Electronic Journal*.
- [14] Tobias Eisenreich, Sandro Speth, and Stefan Wagner. 2024. From requirements to architecture: An ai-based journey to semi-automatically generate software architectures. In *Proc. of the 1st Int'l Workshop on Designing Software*. 52–55.
- [15] Zhangyin Feng, Daya Guo, and Duyu et al. Tang. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Proc. of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2020), 1536–1547.
- [16] J. Hou, X. Zhang, and Y. Li. 2023. A Systematic Review of Large Language Models in Software Engineering: Capabilities, Challenges, and Opportunities. *Empirical Soft. Eng.* 28, 4 (2023), 1–25. doi:10.1007/s10664-023-10234-5
- [17] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Soft. Eng. and Methodology* 33, 8 (2024), 1–79.
- [18] Kevin Jesse, Toufique Ahmed, and Premkumar Devanbu. 2023. Large Language Models and Simple, Stupid Bugs. *arXiv preprint arXiv:2303.11455* (2023).
- [19] João osé JMaranhão and Eduardo Martins Guerra. 2024. A Prompt Pattern Sequence Approach to Apply Generative AI in Assisting Software Architecture Decision-making. In *Proc. of the 29th European Conference on Pattern Languages of Programs, People, and Practices*. 1–12.
- [20] Daniel Katz, Michael Bommarito, and Thomas Klinger. 2023. Legal and Ethical Considerations of AI-Generated Code. *Artificial Intelligence and Law* 31, 2 (2023), 231–254.
- [21] S. Kim, J. Lee, and E. Shin. 2021. AI-Powered Test Automation: A Survey. *Comput. Surveys* 54, 4 (2021).
- [22] Shahab Kolahdoust-Rahimi et al. 2023. A Systematic Literature Review on Requirement Formalization Using NLP and ML. *arXiv preprint arXiv:2303.13365* (2023). <https://arxiv.org/abs/2303.13365>
- [23] Haonan Li, Yu Hao, and Yizhuo Zhai. 2024. Enhancing Static Analysis for Practical Bug Detection: An LLM-Integrated Approach. *Proc. of the ACM on Programming Languages* 8, OOPSLA1 (2024).
- [24] Y. Li, L. Zhang, and M. Sun. 2022. AI-Powered Predictive Maintenance: A Survey. *Comput. Surveys* 55, 5 (2022).
- [25] Abid Mehmood, Qazi Mudassar Ilyas, Muneer Ahmad, and Zhongliang Shi. 2024. Test Suite Optimization Using Machine Learning Techniques: A Comprehensive Study. *IEEE Access* 12 (2024), 168645–168671. doi:10.1109/ACCESS.2024.3490453
- [26] Ninareh Mehrabi, Fred Morstatter, and Nripsuta Saxena. 2021. A Survey on Bias and Fairness in Machine Learning. *Comput. Surveys* 54, 6 (2021), 115–138.
- [27] C. Murphy, S. Singh, and K. Patel. 2023. AI-Assisted Code Refactoring: A Survey. *Comput. Surveys* 56, 3 (2023).
- [28] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Artificial Intelligence versus Software Engineers: An Evidence-Based Assessment Focusing on Non-Functional Requirements. In *ACM Conference on Soft. Eng. and AI (SEAI)*.
- [29] Andreea Necula, Andreas Schenkel, and Angela Cosma. 2024. Natural Language Processing in Software Requirements Engineering: A Systematic Literature Review. *Electronics* 13, 11 (2024), 2055. doi:10.3390/electronics13112055
- [30] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *Int'l Conf. on Mining Software Repositories (MSR)*.
- [31] Henrique Nunes, Eduardo Figueiredo, Larissa Rocha, Sarah Nadi, Fischer Ferreira, and Geanderson Esteves. 2025. Evaluating the Effectiveness of LLMs in Fixing Maintainability Issues in Real-World Projects. *arXiv preprint arXiv:2502.02368* (2025).
- [32] Sushant Kumar Pandey, Sivajeet Chand, Jennifer Horkoff, Miroslaw Staron, Miroslaw Ochodek, and Darko Durisic. 2025. Design pattern recognition: a study of large language models. *Empirical Soft. Eng.* 30, 3 (2025), 69.
- [33] Henry Pearce, Waseem Ahmad, Brendan Dolan-Gavitt, and Rahul Srivastava. 2022. Asleep at the Keyboard? Assessing the Security of AI-Generated Code. In *Proc. of the IEEE Symposium on Security and Privacy*. 765–783. doi:10.1109/SP46214.2022.9833591
- [34] Myron David Lucena Campos Peixoto, Davy de Medeiros Baia, Nathalia Nascimento, Paulo Alencar, Balduino Fonseca, and Márcio Ribeiro. 2025. On the Effectiveness of LLMs for Manual Test Verifications. In *Proc. of the 47th IEEE/ACM Int'l Conf. on Soft. Eng. (ICSE) - Companion*. Ottawa, Ontario, Canada.
- [35] Anurag Sharma, Xinyu Wang, and Yang Liu. 2021. Trust and Explainability in AI-Driven Software Engineering. *ACM Transactions on Soft. Eng. and Methodology* 30, 2 (2021), 1–22.
- [36] E. Shin, S. Kim, and J. Lee. 2022. AI-Powered Bug Fixing: A Survey. *Comput. Surveys* 55, 4 (2022).
- [37] Daniel Sihaan, Indra Kharisma Raharjana, and Chastine Fatichah. 2023. User story extraction from natural language for requirements elicitation: Identify software-related information from online news. *Information and software technology* 158 (2023), 107195.
- [38] Mohammed Latif Siddiq, Joanna Cecilia Da Silva Santos, Ridwanul Hasan Tanvir, Noshin Ulfat, Fahmid Al Rifat, and Vinicius Carvalho Lopes. 2024. Using large language models to generate junit tests: An empirical study. In *Proc. of the 28th Int'l Conf. on Evaluation and Assessment in Soft. Eng.* 313–322.
- [39] Elvys Soares, Manoel Aranda, Naelson Oliveira, Márcio Ribeiro, Rohit Gheyi, Emerson Souza, Ivan Machado, André Santos, Balduino Fonseca, and Rodrigo Bonifácio. 2023. Manual Tests Do Smell! Cataloging and Identifying Natural Language Test Smells. In *2023 ACM/IEEE Int'l Symposium on Empirical Soft. Eng. and Measurement (ESEM)*. IEEE, 1–11.
- [40] Daniel Sobania, Lars Heinemann, and Michael Pradel. 2023. An Empirical Study on the Correctness of Code Generated by Large Language Models. *Proc. of the ACM on Programming Languages* 7, OOPSLA (2023), 1–27. doi:10.1145/3613425
- [41] Alexey Svyatkovskiy, Sebastian Proksch, and Peter et al. Leitner. 2020. AI-Powered Code Completion: A Survey. *IEEE Transactions on Soft. Eng.* 47, 2 (2020), 344–362.
- [42] Valerio Terragni, Annie Vella, Partha Roop, and Kelly Blincoe. 2025. The Future of AI-Driven Software Engineering. *ACM Transactions on Soft. Eng. and Methodology* (2025).
- [43] Priyan Vaithilingam, Wenyu Xie, and Kevin Anderson. 2022. Expectations vs. Reality: Understanding Developer Experience with AI-Powered Code Generation Tools. In *Proc. of the ACM Conference on Human Factors in Computing Systems (CHI)*. 1–15. doi:10.1145/3491102.3517538
- [44] 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 Soft. Eng.* (2024).
- [45] Yujing Yang, Boqi Chen, Kua Chen, Gunter Mussbacher, and Dániel Varró. 2024. Multi-step Iterative Automated Domain Modeling with Large Language Models. In *Proc. of the ACM/IEEE 27th Int'l Conf. on Model Driven Engineering Languages and Systems*. 587–595.
- [46] Stephen S Yau and Jeffery J-P Tsai. 1986. A survey of software design techniques. *IEEE Transactions on Soft. Eng.* 6 (1986), 713–721.
- [47] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. 2023. No more manual tests? evaluating and improving chatgpt for unit test generation. *arXiv preprint arXiv:2305.04207* (2023).
- [48] Xin Zhao, Lei Wang, and Wei Tang. 2023. Ethical and Legal Challenges in AI-Powered Software Engineering. *ACM Transactions on Soft. Eng. and Methodology* 32, 1 (2023), 1–19. doi:10.1145/3553487
- [49] Y. Zhou, S. Li, and L. Zhang. 2023. AI-Powered Security Analysis: A Survey. *IEEE Transactions on Soft. Eng.* 49, 3 (2023), 315–338.