Proceedings of CONF-SEML 2026 Symposium: Importance of Machine Learning Methods and Analysis in Engineering
DOI: 10.54254/2755-2721/2026.AS31339

The Application of Artificial Intelligence in Software Engineering: An Exploration of the Research Trajectory from Requirements Analysis to Test Optimization

Weiyi Ren

School of Software, Shanxi Agricultural University, Jinzhong, China
rwy336611@163.com

Abstract. In the digital economy, traditional software engineering faces complex needs and fast change, and it often meets risks such as cost overruns and low process efficiency. AI methods, including machine learning and natural language processing, provide strong support for a more intelligent form of development. This paper reviews the path of AI use in software work through literature study and real cases, and it covers tasks from requirement analysis to testing and operation. The findings show a shift from a simple helper to a driver of the whole process, and cases from many fields report higher efficiency and better quality. Yet several problems remain, such as weak model transparency, unstable data, and a lack of skilled staff. It presents a framework and guides firms toward high-quality growth globally.

Keywords: Artificial Intelligence(AI), Software Engineering, Machine learning

1. Introduction

In the digital economy, software acts as a central driver of social change and industrial renewal. However, the traditional software engineering model is difficult to cope with complex requirements and rapid iteration of the market, and a large number of software projects are faced with risks such as budget overruns and schedule delays, which are caused by subjective requirements analysis, low development efficiency and insufficient test coverage [1]. The breakthrough of Artificial Intelligence(AI) technologies, such as deep learning and natural language processing, has brought revolutionary opportunities for software engineering, which can not only automate repetitive tasks, but also optimize decisions and improve software quality and efficiency through data-driven [2]. For this reason, a systematic review of its application path is essential. From an academic view, this work can close existing gaps and help form a clear and complete framework. From an industry view, it can guide firms in practice and offer technical support for developers [3]. The main aim of this study is to trace how AI is used across the full software engineering process and to build a usable framework. It centers on two main ideas: common use scenes of AI in different fields, and clear gaps in its performance, plus the key barriers and possible ways toward deeper links with software engineering. This paper has a layout. Section 2 gives the basic theory. It describes the software lifecycle, AI tools, and their relations. Section 3 shows AI use in analysis, design, coding, testing, and upkeep, with cases. Section 4 reviews integration issues. These cover technical limits, practical

DOI: 10.54254/2755-2721/2026.AS31339

concerns, skill gaps, and ethical risks, and it suggests future research paths. Section 5 sums up results and practical value.

2. Theoretical background

2.1 Traditional software engineering process

Traditional software engineering follows five main stages: needs review, system planning, coding, testing, and maintenance. It works step by step, from clear goals and structure to program writing, bug finding, and stable service after release in real practice today [4].

2.2 Core AI technologies related to software engineering

The link between artificial intelligence and software engineering depends on three main tools: machine learning, natural language processing, and large language models.First, machine learning works as the main engine. It appears in supervised tasks, such as defect prediction and need grouping, using past project data and methods like support vector machines and random forests. It also appears in unsupervised tasks. These include code grouping and similarity checks, which help reduce repeated code. The next area is natural language processing. It gives meaning support for software work. It applies tools like word splitting and entity tagging. These methods turn free text into clear data and improve requirement study accuracy [5]. Models such as BERT find conflicts in files. This lowers manual effort and raises overall work efficiency for teams in real projects today often.

Large language models mark a new stage of generative intelligence in software engineering. With training on large datasets, they can perform tasks such as code writing and error repair [6].

2.3 Technical complementarity and paradigm evolution of AI and software engineering

The synergy between AI and software engineering comes from their technical fit. Software engineering offers a clear structure, and AI adds automation and smart functions. This reduces manual work and improves efficiency levels.The integration of the two realizes "1+1>2"—the structured data of the former provides samples for AI model training, and AI technology promotes software engineering from "artificial driven" to "data and intelligence dual driven". From the perspective of paradigm evolution, AI-driven software engineering has gone through three stages: the "auxiliary tool" stage (AI acts on a single link with an independent tool), the "process embedding" stage (AI is deeply integrated into each process, such as the rapid transformation from requirements to code), and the "whole process intelligence" stage (LLM runs through the whole process to realize end-to-end intelligence and closed-loop optimization) [7]. At present, the industry is moving towards the third stage, and the whole process intelligence is the core direction in the future [8].

3. The core scenarios and practical applications of AI in software engineering

3.1 Application of AI in requirements analysis

Requirements analysis is crucial in software engineering, as its quality directly impacts project success. Traditional methods often suffer from inefficiency and high error rates. AI, with its automation and intelligent analysis capabilities, addresses these pain points effectively. The core

DOI: 10.54254/2755-2721/2026.AS31339

applications are concentrated in three aspects: For requirements acquisition, the NLP system can automatically extract and classify requirements from multi-source data such as app store reviews to solve the problem of time-consuming information leakage by manual methods [9]. The related modules of Microsoft Dynamics 365 have reflected the efficiency value. In requirements conflict resolution, AI identifies contradictory requirements through semantic graphs, and SAP's AI system can combine historical data to quantify risks and output solutions. In terms of requirements tracking, AI uniquely identifies requirements, establishes the whole process association, automatically locates the affected links when requirements change, and reduces rework and shortens the delivery cycle after the application of the head bank [10].

3.2 Application of AI in software design and development

Software design and development stay at the heart of software engineering. AI tools try to lift design quality and speed up coding. Their main uses fall into three fields: smart architecture planning, code creation and tuning, and help for component reuse. In architecture work, AI studies many past projects and links them to current needs. This avoids plans that are too complex or too basic [11]. Microsoft Azure tools are now used in many firms. They shorten the full design cycle. GitHub Copilot and similar systems can write code and remove repeated parts [12]. The CodeLlama model supports language transfer and cuts platform change costs [13]. Reuse tools match and adapt modules by themselves. IBM systems raise reuse levels and lower repeated development work. This supports faster delivery in real projects.

3.3 AI applications in software testing and operation and maintenance

Testing and maintenance are central to software quality. AI tools move these stages forward with automation and prediction. They lower labor needs and raise work speed. In testing, AI builds wide test sets. In operations, AWS CloudWatch AIOps finds faults early and speeds checks, while IBM systems adjust resources and reduce costs [14]. This change supports stable, large-scale services for modern firms in real practice today worldwide across many sectors.

3.4 Typical industry application cases

Software needs in engineering vary by industry, so AI tools receive different types of attention. Manufacturing and auto fields focus on stable systems. Volkswagen uses Google Gemini in its myVW car service. The smart check unit finds software and hardware mismatch issues early. This cuts repair demand and speeds fault response. Finance puts weight on risk control and rule compliance. Standard Chartered Bank and PWC built a generative AI test tool for customer services. The system found 17 defect types with NLP checks [15]. Review speed increased by 65 percent. Output accuracy rose from 78 percent to 99.2 percent [15]. The Internet field prefers fast change. The Snap social app in the United States upgraded “My AI” with Gemini. This cut the need review cycle from five days to one day. Automatic coding reduced release time by 40 percent. Maintenance tools kept delays under 50 ms, and daily use grew by 18 percent [16]. These cases show AI goals change with context, scale, and work needs in practice.

DOI: 10.54254/2755-2721/2026.AS31339

4. Challenges and future in the integration of AI and software engineering

4.1 Technical challenges

In software engineering, AI still faces three main technical limits: low transparency, weak data quality, and poor cross-scenario use. Many AI systems work like black boxes. Their inner rules are hard to see, so their decisions are hard to explain. This lack of clarity restricts use in areas like finance and healthcare, where clear logic is needed. Labeled data for software tasks costs a lot to make. Data samples stay small, and most sets come from mature Internet projects [17]. This narrow source lowers model strength. Missing data in special fields also hurts learning and slows real use. At the same time, cross-scenario transfer remains weak. Projects differ by field, size, and form. Models trained in one case often fail in others, so wide reuse is still hard [18]. This problem slows stable adoption and limits large-scale industrial use in real practice today.

4.2 Challenges at the engineering practice and talent level

The core obstacles for enterprises to implement the integration of AI and software engineering focus on three aspects. The first is the lack of compatibility between tools and processes. The existing tool chains such as Jira and Git are disconnected from the data and processes of new AI tools, which makes it difficult to realize automatic closed-loop. Second, the threshold for team adaptation is high, developers will not use advanced functions of AI tools, and the training cost is high, which is difficult for small and medium-sized enterprises to afford. Third, interdisciplinary talents are scarce. Talents with both AI and software engineering capabilities are in short supply, difficult to recruit, high cost, and significant salary premium of relevant positions are still vacant [19].

4.3 Assessing management and ethical compliance challenges

The use of artificial intelligence (AI) in software engineering faces many barriers that slow its progress. A major issue is how to measure AI value. The costs of adoption are clear, but the expected gains, such as lower failure risk, lack solid financial proof, and this weakens business interest. Security and intellectual property also raise concern. Data may leak during model training, and AI tools may reuse protected code. These risks require careful control. Ethical rules remain another problem. AI systems can produce biased code, and responsibility for related errors is hard to assign [20, 21]. These unresolved issues continue to limit trust and restrict large-scale deployment in real projects in practice today.

4.4 Future research directions

To address the technical, practical, and ethical problems described above, future research on AI and software engineering should seek clear progress and workable answers, with a few main directions to guide work. First, develop AI that is transparent and dependable for software tasks. This can be achieved by mixing knowledge graphs with causal tools to improve clarity, and by building open systems for high-risk fields like finance and healthcare. And, create a shared evaluation framework. Universities, companies, and professional bodies should cooperate to set multi-level standards that measure efficiency, quality, and safety in both design and real use. What's more, promote new styles of human–AI teamwork. Roles should be clear, so AI handles routine duties and people focus on creative work, supported by AI-based decision tools for smoother cooperation in practice.

DOI: 10.54254/2755-2721/2026.AS31339

5. Conclusion

This study explores how artificial intelligence, or AI, enters software engineering. It uses a "technology–scenario–challenge–direction" frame to review AI use across many stages of software work. The results show strong links between AI and this field. They also show a clear change. AI is not only a helper now. It has become a main driver of smart processes. With real cases from several sectors, the study shows how AI helps with requirement analysis, system design, and code building. It also explains how good adaptation can improve daily efficiency.

The paper looks at common limits in AI use. It then gives advice for both scholars and practitioners. For academic research, it calls for more work on interpretable models and few-shot learning. It also asks for shared datasets and common evaluation rules. For small and medium firms, the advice is to use mature tools first. Large companies should build integrated platforms and strengthen data protection.

A shortage of skilled staff remains a serious problem today. To close this gap, the study backs shared training plans. These cover school links and in-house courses. It highlights cross-field learning and long-term skill growth programs for workers today. With these steps, the sector can build broader skills and reduce the lack of trained experts here over time. Such actions support steady and responsible development in practice today.

References

[1] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv: 2107.03374. https: //arxiv.org/abs/2107.03374

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901. https: //proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[3] Willems, M. (2025). How Standard Chartered is applying software testing discipline to GenAI. QA Financial. https: //qa-financial.com/how-standard-chartered-is-applying-software-testing-discipline-to-genai/

[4] Alessio Ferrari, Liping Zhao, and Waad Alhoshan. 2021. NLP for requirements engineering: tasks, techniques, tools, and technologies. In Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings (ICSE '21). IEEE Press, 322–323. https: //doi.org/10.1109/ICSE-Companion52605.2021.00137

[5] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., ... & Liu, S. (2021). CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv: 2102.04664. https: //arxiv.org/abs/2102.04664

[6] Nijkamp, E., Xie, B., Hayashi, H., Pang, B., Xia, C., Xiong, C., ... & Zhou, Y. (2023). CodeGen2: Lessons for training LLMs on programming languages. arXiv preprint arXiv: 2305.02309. https: //arxiv.org/abs/2305.02309

[7] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., ... & Zimmermann, T. (2019). Software engineering for machine learning: A case study. Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, 291-300. https: //ieeexplore.ieee.org/document/8804457

[8] Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. 2025. SWE-GPT: A Process-Centric Language Model for Automated Software Improvement. Proc. ACM Softw. Eng. 2, ISSTA, Article ISSTA104 (July 2025), 22 pages. https: //doi.org/10.1145/3728981

[9] Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, 610-623. https: //dl.acm.org/doi/10.1145/3442188.3445922

[10] Lozhkov, A., Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Mou, C., ... & De Vries, H. (2024). StarCoder 2 and The Stack v2: The Next Generation. arXiv preprint arXiv: 2402.19173. https: //arxiv.org/abs/2402.19173

[11] Chen, Z., Kang, Y., Li, L., Zhang, X., Zhang, H., Xu, H., ... & Wu, R. (2021). Towards intelligent incident management: Why we need it and how we make it. Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 1487-1497. https: //dl.acm.org/doi/10.1145/3368089.3417055

DOI: 10.54254/2755-2721/2026.AS31339

[12] Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021). Program synthesis with large language models. arXiv preprint arXiv: 2108.07732. https: //arxiv.org/abs/2108.07732
[13] Xu, F. F., Alon, U., Neubig, G., & Hellendoorn, V. J. (2022). A systematic evaluation of large language models of code. Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, 1-10. https: //dl.acm.org/doi/10.1145/3520312.3534862
[14] Patel, J. S. B. (2025). AI-driven test automation: transforming software quality engineering. Journal of Computer Science and Technology Studies, 7(2), 339–347. https: //doi.org/10.32996/jcsts.2025.7.2.35
[15] Crudu, V., & Mold Stud Research Team. (2025, August 12). Exploring the impact of AI on financial software testing - revolutionizing the future of fintech. MoldStud. https: //moldstud.com/articles/p-exploring-the-impact-of-ai-on-financial-software-testing-revolutionizing-the-future-of-fintech
[16] Najmi, A., & El-dosuky, M. A. (2025). Intelligent software testing for test case analysis framework using ChatGPT with natural language processing and deep learning integration. Journal of Computer Science, https: //doi.org/10.3844/jcssp.2025.1140.1155
[17] Allamanis, M. (2019). The adverse effects of code duplication in machine learning models of code. Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, 143-153.https: //dl.acm.org/doi/10.1145/3359591.3359735
[18] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.https: //ieeexplore.ieee.org/document/5288526
[19] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23), Vol. 1. Association for Computing Machinery, New York, NY, USA, 106–121. https: //doi.org/10.1145/3568813.3600138
[20] Mittelstadt, B. (2019). Principles alone cannot guarantee ethical AI. Nature Machine Intelligence, 1(11), 501-507.https: //www.nature.com/articles/s42256-019-0114-4
[21] Jobin, A., Ienca, M., & Vayena, E. (2019). The global landscape of AI ethics guidelines. Nature Machine Intelligence, 1(9), 389-399.https: //www.nature.com/articles/s42256-019-0088-2