Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X
Volume 3, Issue 6, 2025

AI-POWERED TOOLS FOR FASTER AND BETTER SOFTWARE DEVELOPMENT

Usama Nasir[1], Hoor Fatima Yousaf*[2], Muhammad Abubakar Farooq[3], Misbah Maqbool[4], Haroon Ilyas[5], Dilawar Khan Sukhera[6], Rabia Abbas[7]

[1]Lecturer, Department of Computer Sciences, University of Central Punjab Lahore, Pakistan
*[*2]Lecturer, Department of Computer Sciences, Bahria University Lahore Campus, Lahore, Pakistan*
[3]Department of Information Technology, Bahria University Lahore Campus, Lahore, Pakistan
[4]Department of Artificial Intelligence, University of Management and Technology, Lahore, Pakistan
[5]Departmental Coordinator, Department of Computer Sciences Bahria University Lahore Campus Lahore, Pakistan
[6]MS (Population, Resource & Environmental Economics), Shanxi University of Finance and Economics China
[7]Lecturer, Department of Computer Sciences, Rashid Lateef Khan University Lahore Pakistan

[1]usama.nasir@ucp.edu.pk, *[2]hoorfatima.bulc@bahria.edu.pk, [3]abubakarfarooq123@gmail.com, [4]Mibba1996@gmail.com, [5]sacs.bulc@bahria.edu.pk, [6]dilawarkhansukhera7@gmail.com, [7]rabia.abbas@rlku.edu.pk

DOI: https://doi.org/10.5281/zenodo.15574570

Keywords

Article History
Received on 25 April 2025
Accepted on 25 May 2025
Published on 02 June 2025

Copyright @Author
Corresponding Author: *
Hoor Fatima Yousaf

Abstract
Artificial Intelligence (AI) has changed many fields, and making software is one of them. With more need for fast, good, and nice code, AI tools are being used a lot to help in different parts of building software, from making code and fixing it to finding mistakes and checking software. This paper looks at how AI tools affect the speed and quality of making software by conducting a test. We put AI tools into four main groups: code finishing, auto-checking, mistake finding, and project helping. A test was done with two groups of coders – one using old ways of working and the other using picked AI tools. Important measures, like code quality, time to make it, and mistake counts, were checked and compared. The results show that AI tools boost work speed and cut down on human mistakes, but problems like tool correctness and learning time are still there. These findings show AI can be a big deal in modern software building, giving useful ideas for coders and researchers.

INTRODUCTION

The use of Artificial Intelligence (AI) in making software is quickly changing how things work. AI tools are being used more and more to write code by themselves, find mistakes, make testing better, and handle projects more easily. These tools help cut down on building time, make code nicer, and reduce human slip-ups, which makes work better overall (Smith & Kumar, 2020). Coders now use AI not just for boring jobs but also for tricky stuff, like guessing what might happen and smart mistake fixing (Lee et al., 2021). New AI systems, especially those with machine learning and deep learning stuff, can get context and learn from big piles of code. Tools like code-finishing helpers and suggestion platforms make coders faster by giving correct and sensible tips right away (Nguyen & Zhang, 2019; Sharma et al., 2023). Because of this, companies see much shorter building times (Brown et al., 2020). Past studies show that AI testing setups can spot more mistakes in less time than old ways, making software tougher

(Gao & Li, 2019; Kaur & Malhotra, 2021). Also, making test cases by themselves has been shown to cover more and repeat less (Zhou et al., 2021). Likewise, AI debugging tools help coders by showing problems during running (Ali et al., 2019; Martin et al., 2022). Lots of writings also point out that AI is good for project handling tasks like picking what to do first, planning short work bursts, and estimating effort (Chaudhary & Patel, 2022; Iqbal et al., 2020). These uses help teams work together better and use stuff wisely. In particular, natural language processing (NLP) models are great for understanding user stories, managing papers, and making talking easier (Rahman & Zhou, 2020). Even with these steps forward, problems are still there. Things like explaining how AI works, how correct its tips are, keeping data private, and the hard time learning new tools are well-known (Thomas et al., 2022; Arora & Singh, 2023). Plus, worries about the quality of AI-made code and keeping it good for a long time have led to calls for more real-world checks (Jin et al., 2018; Niazi & Farooq, 2023).

Aims and objectives:

This research will empirically evaluate the performance of software development tools supported by AI. By classifying tools into four functional areas — code completion, automated testing, bug detection, and project management

This study explores their effect on software quality, development velocity, and user satisfaction. The outcomes are anticipated to offer empirical contributions to the mounting debate on the role of AI in software development.

Literature Review

AI in Making and Finishing Code New stuff in machine learning has made tools that help coders write correct and good code. Tools like GitHub Copilot and DeepTabNine use special models to guess and finish code bits (Chen et al., 2021). These tools cut down on typing and make thinking easier (Xu & Wang, 2022). Studies show coders using AI code helpers finish tasks 30–50% faster (Zhang et al., 2020). But, people still wonder if AI-made code is good for hard problems (Patel & Rana, 2023).

AI in Software Checking AI checking tools do unit, integration, and repeat tests by learning from old code changes and mistake patterns (Chakraborty & Bose, 2021). Smart learning and brain-like networks are put in systems to make test cases that cover rare cases and odd exceptions (Li & Zhao, 2019). Research shows a big jump in test coverage and fewer missed mistakes with AI test-making (Ahmed & Dar, 2023). Still, understanding and trusting these tools in real work is a worry (Tan & Noor, 2022).

Finding Mistakes and Code Quality AI-powered code checkers and mistake finders are better at spotting security holes and code typos (Singh et al., 2020). Smart mistake-predicting models can guess possible errors by looking at old code flops (Wang et al., 2022). Proof shows a 40% better early mistake catch with AI than hand-checking (Ali & Tariq, 2021). But, these tools have trouble finding logic mistakes that need special knowledge (Jiang & Han, 2020).

AI in Software Fixing and Tidying AI is changing software fixing with smart code tidying tools. These setups suggest changes to make code clearer, cut down on old problems, and boost speed (Yu & Cao, 2020). Brain-like models trained on code history can suggest tidying steps based on context (Mehmood & Shahid, 2021). Such tools reduce hand work, but people still need to watch because of wrong suggestions (Iqbal & Bashir, 2023).

AI for Handling Projects in Software Building In fast-paced building, AI helps with task giving, backlog sorting, and short-term guessing by looking at old data (Pereira & Costa, 2021). Guessing models help figure out delivery times, resource splitting, and work balancing (Huang & Lin, 2022). Project bosses use AI chatbots and helpers to deal with regular talking, leaving time for big choices (Amin & Siddiqui, 2020). Even with good results, trust in AI tips depends on how ready a company is (Naseer & Hussain, 2023).

Human-AI Teamwork and Coder Thoughts: How well AI tools work often depends on fitting with human work processes. Studies say that when coders see AI as a teammate, not a replacement, they use it more (Turner & Holmes, 2021). Coder happiness with AI tools ties to easy use, clarity, and fitting the situation (Rafiq & Zhang, 2019). But, worries about who owns code, who made it, and leaning too much on AI are big issues (Mohammed & Fraser, 2022).

Problems and Limits Even with quick use, problems like explaining AI, keeping data private, working everywhere, and setup costs still hold AI back (Choudhary & Fatima, 2023). Some say AI tools can stick too much to certain code patterns if trained on bad data (Kang & Lee, 2021). Others note that hard-to-get AI models might add quiet mistakes to systems (Shen & Yu, 2020). So, clear and understandable AI is a big study goal in software development.

Methodology

1. Research Design

This study used a mixed-methods plan, mixing number-based and word-based methods to fully check how AI tools affect software making speed and quality. The number part looked at work output measures, while the word part dug into coders' feelings about using AI tools.

2. Participants and Sampling

Participants were picked on purpose from software building companies in the USA, UK, and Pakistan. A total of 60 working coders with at least two years of know-how were asked to join. Of them, 30 coders used AI tools a lot (test group), and 30 stuck to old ways of coding (control group). Everyone agreed to share info before the data was gathered.

3. Tools Used

The AI tools checked in this study were:

- GitHub Copilot (for finishing code)
- TabNine (for smart code filling)
- DeepCode (for spotting mistakes and checking code)
- Test.AI (for auto-checking)
- Jira with AI add-ons (for helping with project tasks)

These tools were chosen because they are well-liked, useful, and work well with normal coding setups.

4. Data Collection Procedures

Data was gathered in two steps:

a) Quantitative Data:

Coders were given five standard coding jobs, like linking APIs, fixing mistakes, adding features, and writing notes.

These things were tracked:

- Time to finish each job (in minutes)
- Number of mistakes per 100 lines of code
- Lines of code (LOC) made
- Code quality score, checked with SonarQube.

Coders with AI tools used their helpers fully, while the control group did the jobs without AI help. All work was watched and saved for fairness.

b) Qualitative Data:

After the jobs, talks were held with 15 randomly picked coders from each group.

The talks tried to learn:

- How useful AI tools seemed
- Trust in AI-made code
- Hard parts of using AI

Changes in workflow or choices.

Each talk took about 30–45 minutes and was recorded with permission.

5. Data Analysis

a) Quantitative Data

Basic stats (average, spread) were figured out for each work measure. A special test (t-test) was used to compare the AI and non-AI groups for job finish time, code quality, and mistake counts. All number work was done with SPSS v26, with a cutoff for importance at $p < 0.05$. b) Qualitative Data Talk recordings were written out word-for-word and studied with theme sorting. The info was labeled by hand and grouped into main ideas using NVivo software. Trust in the findings was kept by checking with participants and talking with other researchers.

Results

1. Participant

A total of 60 coders took part in this study. The average age was 29.4 years (SD = 3.7), with 42 men and 18 women. All had at least 2 years of work know-how, and none had seen the test jobs before. The coders were split evenly into the AI-using group ($n = 30$) and the non-AI (control) group ($n = 30$).

Findings

a) Job Finish Time:

Coders with AI tools finished all five jobs much quicker than the control group. The average time for the AI group was 43.2 minutes (SD = 6.3), while the control group took 62.5 minutes (SD = 7.1). A special test showed a big difference between groups ($t(58) = 10.31$, $p < 0.001$). **b) Code Quality:**

Score Using SonarQube The AI group's average code score was better (mean = 8.7/10, SD = 0.9) than the control group's (mean = 7.1/10, SD = 1.1), with a clear difference found (t(58) = 6.48, p < 0.001).

c)Mistakes per 100 Lines of Code:
The AI group made fewer mistakes per 100 lines of code (mean = 3.2, SD = 1.5) than the control group (mean = 6.7, SD = 2.1), showing a big drop in errors (t(58) = 7.14, p < 0.001).

d)Lines of Code Made:
The AI group wrote more code lines (mean = 254 LOC, SD = 38) than the control group (mean = 201 LOC, SD = 34), hinting at better work output (t(58) = 5.33, p < 0.001).

2. Word-Based Findings
a) Seen Usefulness
All 15 AI group talkers said the tools boosted their work, especially for repeat coding and mistake fixing. GitHub Copilot and TabNine were liked for cutting down hand work.

b) Trust in AI-Made Code:
11 of 15 AI group coders were unsure at first, but grew to trust AI code a bit or a lot after checking it with tests. Most still checked AI code by hand.

c) Work Flow Betterment:
Coders noted smoother work with Test.AI for auto-tasks and Jira AI add-ons for tracking short work bursts. These tools let them focus on harder problems instead of boring tasks.

d) Problems mentioned were:
- Sometimes AI gave useless or wrong tips.
- Worry about leaning too much on AI, especially for new coders.
- Need to hand-fix AI code in tricky logic cases.

e) Non-AI Group Thoughts Control:
Group coders said they felt more brain strain and spent longer fixing mistakes. They showed interest in trying AI tools for later projects.

4. Summary of Statistical Outcomes

Metric	AI Group (Mean ± SD)	Control Group (Mean ± SD)	p-value
Task Completion Time (min)	43.2 ± 6.3	62.5 ± 7.1	< 0.001
Code Quality Score (10)	8.7 ± 0.9	7.1 ± 1.1	< 0.001
Bugs per 100 LOC	3.2 ± 1.5	6.7 ± 2.1	< 0.001
LOC Written	254 ± 38	201 ± 34	< 0.001

Significant at p < 0.001

5. Ethical Adherence
No ethical violations were reported. All participants voluntarily consented, and anonymity was maintained. Data were securely stored, and participant identifiers were coded.

Discussion
The study results showed a significant positive effect of AI tools on different parts of software development, like job speed, code niceness, and coder work output. These findings match the growing pile of writings that point out AI's game-changing power in modern software building.

First, the big drop in job finish time for coders using AI tools backs up what Vasilescu et al. (2019) found, saying AI helpers like code fillers speed up coding by cutting down on choice fatigue and typing work. Also, Chen et al. (2021) saw that GitHub Copilot users finished coding tasks quicker than those using old ways.

The better code quality in the AI group fits with past studies. Zhou et al. (2020) said auto code checkers and mistake finders, like DeepCode, catch small problems humans often miss. Plus, Ahmad et al. (2022) noted that AI tools can make code easier to keep up and cut down on old issues when used right in the software process.

The fewer mistakes per 100 lines of code in this study aligns with Bird et al. (2019), who said AI code checkers spot security holes and logic slip-ups better than just human checking. Also, Huang et al. (2021) pointed out that AI testing tools, like Test.AI, lower human mistakes in repeat testing.

Oddly, the AI group made more lines of code than the control group, a sign often tied to better work output. While lines of code can be a shaky measure (Fuggetta et al., 2020), it matters in set job settings like this study, as backed by Kalliamvakou et al. (2022), who said AI tools help coders focus on doing the work, not on code rules or extra bits.

Word-based findings in this study echoed earlier work by Rahman et al. (2020) and Zimmermann et al. (2021), where coders said AI helpers cut brain strain and made the workflow smoother. Most coders here trusted the tools after checking them, which matches Jiang et al. (2023), who said coder trust in AI grows when tools work well and are clear.

But, some problems came up, like useless tips and leaning too much on AI, which matches worries from Tabassum et al. (2021) and Wessel et al. (2022), who said coders need to keep checking AI outputs and stay in charge. Relying on AI without thinking hard could mess up code correctness and setup strength.

Also, AI tools worked best when tied to team project platforms, like Jira, which fits what Li et al. (2021) found, saying AI-boosted fast-paced work practices made job guessing and task sorting better.

Conclusion

This study shows that AI tools greatly boost the speed, correctness, and overall work output of software development. Through a planned test, it was seen that coders using AI tools like code fillers, mistake finders, and auto-checkers finished jobs faster, made more code lines, and had fewer mistakes than those using old ways. Also, word-based feedback showed coders found these tools helpful in cutting brain strain and letting them focus on logic and problem fixing.

The results match writings that see AI's power in making coders better and code nicer. But the findings also stress the need for careful use, where AI helps as a teammate, not a total swap for human thinking. As the software world keeps changing, smart use of AI tools, along with coder training and the right rules, will be key to using their full strength without losing creativity, control, or deep thinking.

In the end, this study gives solid proof that AI tools are not just add-ons but vital helpers for quicker, smarter, and better software making in today's digital world.

References:

Ali, M., Khan, M. U. G., & Rehman, S. U. (2019). Automated debugging using machine learning techniques. *IEEE Access*, 7, 107126–107137. https://doi.org/10.1109/ACCESS.2019.2932935

Arora, S., & Singh, P. (2023). Challenges in implementing AI in software development: A survey. *Journal of Software Engineering Research and Development*, *11*, 1–19. https://doi.org/10.1186/s40411-023-00153

Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://doi.org/10.48550/arXiv.2005.14165

Chaudhary, M., & Patel, R. (2022). AI in agile software project management. *International Journal of Advanced Computer Science and Applications*, *13*(4), 320–328. https://doi.org/10.14569/IJACSA.2022.0130439

Gao, C., & Li, X. (2019). Test automation using AI: A deep learning approach. *Procedia Computer Science*, *162*, 306–313. https://doi.org/10.1016/j.procs.2019.11.285

Iqbal, M. Z., Khan, A., & Khan, S. (2020). Estimating software effort using AI-based models. *Journal of Systems and Software*, *170*, 110739. https://doi.org/10.1016/j.jss.2020.110739

Jin, X., Li, Y., & Zhang, H. (2018). Empirical study on the quality of AI-generated code. *IEEE Transactions on Software Engineering*, *44*(11), 1085–1097. https://doi.org/10.1109/TSE.2017.2730885

Kaur, P., & Malhotra, R. (2021). Improving software quality using machine learning-based test prioritization. *Applied Soft Computing, 103*, 107168. https://doi.org/10.1016/j.asoc.2021.107168

Lee, H., Park, S., & Kim, Y. (2021). AI-based code review systems: Enhancing developer productivity. *Journal of Systems Architecture, 117*, 102080. https://doi.org/10.1016/j.sysarc.2021.102080

Martin, D., Ahmed, T., & Zhao, W. (2022). Real-time debugging assistant using reinforcement learning. *Software: Practice and Experience, 52*(1), 23–37. https://doi.org/10.1002/spe.2943

Nguyen, A., & Zhang, Y. (2019). Code suggestion using deep neural networks. *Empirical Software Engineering, 24*, 3555–3578. https://doi.org/10.1007/s10664-019-09744-w

Niazi, M., & Farooq, S. (2023). Evaluating maintainability of AI-generated software code. *Journal of Software: Evolution and Process, 35*(1), e2375. https://doi.org/10.1002/smr.2375

Rahman, M. M., & Zhou, Y. (2020). NLP-based tools for improving software documentation. *Information and Software Technology, 120*, 106251. https://doi.org/10.1016/j.infsof.2020.106251

Sharma, D., Verma, P., & Singh, A. (2023). Exploring deep learning-based code assistants. *Software Quality Journal, 31*, 219–240. https://doi.org/10.1007/s11219-022-09617-2

Smith, L., & Kumar, R. (2020). The future of AI in software engineering. *ACM Computing Surveys, 53*(4), 1–36. https://doi.org/10.1145/3391196

Thomas, R., Joseph, R., & Wilson, D. (2022). Explainability in AI tools for software development. *IEEE Software, 39*(6), 60–66. https://doi.org/10.1109/MS.2021.3109290

Zhou, L., Jin, X., & Qian, Z. (2021). Automatic test case generation using AI: A comparative study. *Information and Software Technology, 129*, 106399. https://doi.org/10.1016/j.infsof.2020.106399

Wang, L., Wang, M., & Zhang, T. (2019). AI in DevOps: Opportunities and threats. *Future Generation Computer Systems, 99*, 709–716. https://doi.org/10.1016/j.future.2019.04.023

Liu, F., Tang, H., & Chen, Y. (2021). Automated requirement analysis using NLP techniques. *Software: Practice and Experience, 51*(5), 1034–1048. https://doi.org/10.1002/spe.2904

Yadav, M., & Prakash, A. (2020). Role of AI in CI/CD pipelines. *International Journal of Computer Applications, 176*(29), 1–5. https://doi.org/10.5120/ijca2020920805

Gonzalez, J., & Rivera, A. (2021). Enhancing DevOps with intelligent agents. *Journal of Software Engineering and Applications, 14*, 457–470. https://doi.org/10.4236/jsea.2021.149027

Pereira, M., & Alves, N. (2020). Machine learning approaches for defect prediction. *Applied Sciences, 10*(5), 1838. https://doi.org/10.3390/app10051838

Sundaram, S., & Babu, K. (2018). Deep learning models for code clone detection. *Journal of Systems and Software, 144*, 70–82. https://doi.org/10.1016/j.jss.2018.06.028

Khan, A., & Raza, M. (2021). Software refactoring through AI-based tools. *IEEE Access, 9*, 129884–129897. https://doi.org/10.1109/ACCESS.2021.3113644

Noor, M., & Zahoor, S. (2022). Software defect localization using AI algorithms. *Applied Intelligence, 52*, 8463–8477. https://doi.org/10.1007/s10489-021-02643-6

Chakraborty, T., & Mahajan, S. (2019). Chatbots in software engineering. *Journal of Systems Architecture, 97*, 72–82. https://doi.org/10.1016/j.sysarc.2019.02.007

Singh, R., & Chauhan, R. (2021). NLP models in bug triaging. *Information Processing & Management, 58*(5), 102671. https://doi.org/10.1016/j.ipm.2021.102671

Mehta, D., & Shah, K. (2022). Predictive analysis of developer performance using ML. *Computers in Industry, 138*, 103631. https://doi.org/10.1016/j.compind.2022.103631

Abbas, H., & Qureshi, A. (2020). A framework for AI-based code smell detection. *Journal of Software: Evolution and Process*, *32*(2), e2225. https://doi.org/10.1002/smr.2225

Ahmed, I., & Zubair, M. (2023). Ethical implications of AI-generated code. *AI and Ethics*, *3*, 343–358. https://doi.org/10.1007/s43681-022-00244-1

Ahmed, T., & Dar, S. (2023). Smart test case generation using deep learning. *Software Testing, Verification & Reliability*, *33*(1), e2361. https://doi.org/10.1002/stvr.2361

Ali, S., & Tariq, M. (2021). Bug prediction using neural networks in open-source software. *IEEE Access*, *9*, 54312–54324. https://doi.org/10.1109/ACCESS.2021.3070624

Amin, R., & Siddiqui, H. (2020). Role of virtual assistants in agile software project management. *Journal of Systems and Software*, *168*, 110631. https://doi.org/10.1016/j.jss.2020.110631

Chakraborty, S., & Bose, R. (2021). Machine learning in automated regression testing. *Journal of Software Engineering Research and Development*, 9(1), 1–15. https://doi.org/10.1186/s40411-021-00133-3

Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. *arXiv preprint*. https://doi.org/10.48550/arXiv.2107.03374

Choudhary, S., & Fatima, S. (2023). A survey on limitations of AI in software engineering. *Artificial Intelligence Review*, *56*, 1321–1347. https://doi.org/10.1007/s10462-022-10231-8

Huang, Y., & Lin, Q. (2022). Predictive modeling for sprint planning using AI. *Journal of Software: Evolution and Process*, *34*(3), e2350. https://doi.org/10.1002/smr.2350

Iqbal, F., & Bashir, A. (2023). Intelligent refactoring in Python using deep reinforcement learning. *Software Quality Journal*, *31*(2), 385–400. https://doi.org/10.1007/s11219-022-09651-0

Jiang, L., & Han, Y. (2020). Machine learning for logic error detection. *Empirical Software Engineering*, *25*(3), 1714–1738. https://doi.org/10.1007/s10664-020-09808-0

Kang, H., & Lee, J. (2021). Bias and fairness in AI tools for code review. *AI and Ethics*, *2*, 177–186. https://doi.org/10.1007/s43681-021-00053-2

Li, R., & Zhao, H. (2019). Deep learning models for test automation. *Information and Software Technology*, *113*, 97–109. https://doi.org/10.1016/j.infsof.2019.05.004

Mehmood, K., & Shahid, A. (2021). Learning-based refactoring recommendation systems. *IEEE Transactions on Software Engineering*, *47*(4), 779–793. https://doi.org/10.1109/TSE.2019.2960523

Mohammed, A., & Fraser, M. (2022). Ethics in AI-assisted software development. *AI & Society*, *37*, 613–621. https://doi.org/10.1007/s00146-021-01177-9

Naseer, N., & Hussain, M. (2023). Trust and resistance in adopting AI for project planning. *Computers in Human Behavior Reports*, *8*, 100226. https://doi.org/10.1016/j.chbr.2023.100226

Patel, D., & Rana, K. (2023). Measuring performance of AI-based code completion tools. *Journal of Systems Architecture*, *137*, 102478. https://doi.org/10.1016/j.sysarc.2023.102478

Pereira, A., & Costa, R. (2021). AI-driven backlog prioritization techniques. *Software: Practice and Experience*, *51*(12), 2541–2556. https://doi.org/10.1002/spe.2975

Rafiq, A., & Zhang, W. (2019). User satisfaction of AI code assistants. *Journal of Computer Languages*, *53*, 100716. https://doi.org/10.1016/j.cola.2019.100716

Shen, J., & Yu, B. (2020). Silent failure risks in black-box AI development tools. *IEEE Software*, *37*(5), 92–98. https://doi.org/10.1109/MS.2020.2985874

Singh, V., Khan, R., & Verma, J. (2020). Static code analysis with AI integration. *International Journal of Advanced Computer Science and Applications*, *11*(9), 312–319. https://doi.org/10.14569/IJACSA.2020.0110942

Tan, Y., & Noor, Z. (2022). Testing frameworks in the AI era. *Procedia Computer Science, 199*, 127–136. https://doi.org/10.1016/j.procs.2022.01.015

Turner, J., & Holmes, K. (2021). Collaboration between human and AI programmers. *Empirical Software Engineering*, 26(6), 1–24. https://doi.org/10.1007/s10664-021-10015-y

Wang, S., Zhang, L., & Yuan, Q. (2022). Machine learning for fault localization in Java projects. *Software Testing, Verification & Reliability, 32*(4), e2321. https://doi.org/10.1002/stvr.2321

Xu, C., & Wang, D. (2022). DeepCodeSuggest: Transformer-based intelligent coding assistant. *IEEE Access, 10*, 19876–19890. https://doi.org/10.1109/ACCESS.2022.3149817

Yu, Y., & Cao, J. (2020). Automatic refactoring through AI modeling. *Information and Software Technology, 124*, 106290. https://doi.org/10.1016/j.infsof.2020.106290

Zhang, M., He, Y., & Luo, J. (2020). Evaluating the efficiency of AI auto-completion. *Empirical Software Engineering, 25*(2), 1243–1267. https://doi.org/10.1007/s10664-019-09769-1

Zhao, J., & Sun, L. (2019). Reinforcement learning in software testing. *AI Perspectives, 1*(1), 5–15. https://doi.org/10.1186/s42467-019-0010-4

Khan, B., & Zafar, A. (2022). Knowledge graphs for AI-based debugging. *Knowledge-Based Systems, 239*, 107943. https://doi.org/10.1016/j.knosys.2021.107943

Arif, M., & Haider, T. (2023). Dataset quality and overfitting risks in AI coding tools. *Applied Artificial Intelligence*, 37(1), 1–18. https://doi.org/10.1080/08839514.2022.2125178

Malik, S., & Arshad, I. (2021). Cost of AI integration in small software firms. *Technovation, 104*, 102267. https://doi.org/10.1016/j.technovation.2021.102267

Ahmed, I., & Junaid, M. (2020). NLP-based documentation generation in agile. *Journal of Computer Languages, 57*, 100935. https://doi.org/10.1016/j.cola.2020.100935

Ahmad, I., Ahmed, F., & Khan, M. S. (2022). AI-enhanced software maintenance and technical debt reduction. *Journal of Software: Evolution and Process*, 34(1), e2290. https://doi.org/10.1002/smr.2290

Bird, C., Zimmermann, T., & Murphy, B. (2019). The art of testing less without sacrificing quality. *Empirical Software Engineering*, 24(4), 2163–2188. https://doi.org/10.1007/s10664-018-9666-2

Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*. https://doi.org/10.48550/arXiv.2107.03374

Fuggetta, A., Lavazza, L., & Morasca, S. (2020). On the economics of software development with AI. *Information and Software Technology*, 120, 106241. https://doi.org/10.1016/j.infsof.2019.106241

Huang, Q., Liu, H., & Zhang, H. (2021). An empirical study on AI-based automated testing tools. *IEEE Transactions on Software Engineering*, 47(10), 2120–2135. https://doi.org/10.1109/TSE.2019.2957462

Jiang, H., Ren, J., & Zhang, D. (2023). Trust in AI tools: Developer perspectives and behavioral impact. *Empirical Software Engineering*, 28, 12. https://doi.org/10.1007/s10664-022-10109-3

Kalliamvakou, E., Gousios, G., & Spinellis, D. (2022). Understanding developer productivity with AI pair programming tools. *Journal of Systems and Software*, 192, 111367. https://doi.org/10.1016/j.jss.2022.111367

Li, W., Lin, C., & Chan, W. K. (2021). AI-driven agile project management. *Information and Software Technology*, 134, 106555. https://doi.org/10.1016/j.infsof.2021.106555

Rahman, M., Roy, C. K., & Schneider, K. A. (2020). Impact of code intelligence on developer productivity. *IEEE Transactions on Software Engineering*, 46(5), 514–528. https://doi.org/10.1109/TSE.2018.2889924

Tabassum, M., Densmore, M., & Ahmed, S. (2021). Ethics and human-AI collaboration in coding. *Proceedings of the ACM Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3411764.3445193

Vasilescu, B., Yu, Y., & Wang, X. (2019). The human and machine in software development: Co-evolution with AI. *Communications of the ACM*, 62(12), 78–87. https://doi.org/10.1145/3356727

Wessel, M., Anderson, J., & Herbsleb, J. D. (2022). Unpacking AI-supported code writing. *IEEE Transactions on Software Engineering*, 48(2), 345–359. https://doi.org/10.1109/TSE.2021.3069147

Zhou, Z., Yang, Y., & Wang, H. (2020). The rise of automated code review: What developers say. *Journal of Software: Evolution and Process*, 32(11), e2262. https://doi.org/10.1002/smr.2262

Zimmermann, T., Bird, C., & Nagappan, N. (2021). AI in software engineering: Challenges and future directions. *IEEE Software*, 38(1), 36–45. https://doi.org/10.1109/MS.2020.3026158

Alon, U., Barash, G., & Yahav, E. (2019). Code2vec: Learning distributed representations of code. *Proceedings of the ACM Programming Language Conference*, 3(POPL), 1–29. https://doi.org/10.1145/3290353

Jain, A., & Srivastava, A. (2020). Automating software refactoring using machine learning. *Software Quality Journal*, 28(3), 1137–1164. https://doi.org/10.1007/s11219-019-09467-7

Rajkumar, R., & Menon, V. (2021). AI-based bug prediction in large-scale systems. *Software: Practice and Experience*, 51(2), 325–340. https://doi.org/10.1002/spe.2805

Verma, R., & Arora, N. (2022). Impact of AI in agile development: Developer perspectives. *Journal of Systems and Software*, 190, 111347. https://doi.org/10.1016/j.jss.2022.111347

Kim, Y., & Lee, H. (2023). Software architecture optimization using AI. *Empirical Software Engineering*, 28, 34. https://doi.org/10.1007/s10664-023-10187-6

Bhatia, S., & Singh, K. (2020). Explainable AI in software tools: A framework. ACM *Transactions on Software Engineering and Methodology*, 29(4), 1–30. https://doi.org/10.1145/3409872.