Examining Software Engineering Practices in the Pre-AI and Post-AI Era

Oluwatito Ebiwonjumi [1], Love Allen Chijioke Ahakonye [2], Dong-Seong Kim [3], [4]
[1] Independent Researcher, Nashville, *Tennessee*, USA
[2] ICT-Convergence Research Center, *Kumoh National Institute of Technology*, Gumi, South Korea
[3] IT Convergence Engineering, *Kumoh National Institute of Technology*, Gumi, South Korea
[4] NSLab Co. Ltd., Gumi, South Korea, *Kumoh National Institute of Technology*, Gumi, South Korea
titoebiwonjumi@gmail.com, (loveahakonye, dskim)@kumoh.ac.kr

***Abstract*—The rise of generative artificial intelligence (AI) coding tools such as GitHub Copilot and ChatGPT has reshaped software development, yet their impact on open-source software quality remains unclear. This study conducts a longitudinal analysis of code review and bug-fix patterns across six major Python and JavaScript repositories, pandas, scikit-learn, TensorFlow, Django, React, and Node.js, comparing the pre-AI (2018–2021) and post-AI (2022–2025) periods. Using GitHub pull request data, we examine changes in reviewer participation, review duration, and comment density, alongside post-merge bug-fix frequencies. Results show a mild decline in review intensity after widespread AI adoption, alongside stagnant or increased bug-fix activity, suggesting no corresponding improvement in software quality. These findings suggest potential overreliance on AI-generated code and highlight critical trade-offs between development speed and code robustness, offering empirical evidence for a more cautious, evidence-based integration of generative AI tools in software engineering practice.**

***Index Terms*—AI, Bug Fixes, Code, LLM, Review, Software Engineering**

I. Introduction

The rapid evolution of software engineering practices is continually driven by technological advancements that aim to enhance productivity and product quality [1]. Key stages in the software development lifecycle that significantly impact final software quality are code review and documentation [2]. Code review serves as a critical mechanism for defect detection, knowledge sharing, and adherence to coding standards, while comprehensive documentation ensures maintainability, usability, and long-term project viability [3]. The emergence of large language models (LLMs) represents a significant shift in software development [4]. Trained on extensive code and text data, these models demonstrate strong capabilities in understanding context, generating human-like text, and automating complex tasks [5]. Their use is increasing across essential engineering functions, including automated code generation, bug-fix suggestions, and documentation writing.

Generative artificial intelligence (AI) coding assistants like GitHub Copilot and ChatGPT have rapidly gained adoption, with GitHub reporting over 1.2 million Copilot users by 2023 [6]. Industry claims suggest productivity improvements of up to 55%, yet the impact on code quality remains largely unexplored [7]. Open-source software (OSS) provides an ideal context to examine this question, as OSS projects maintain transparent development histories, employ rigorous review processes, and serve as foundations for countless downstream applications [7].

Despite the enthusiasm surrounding generative AI coding tools, empirical research on their impact on software quality remains surprisingly limited [8]. Existing studies have primarily focused on productivity metrics, such as completion speed, acceptance rates of AI suggestions, and developer satisfaction [9]. While these measures are essential, they provide an incomplete picture. A developer may write code faster with AI assistance, but if that code contains more subtle bugs, requires more extensive post-merge fixes, or receives less thorough review due to false confidence in AI-generated suggestions, the net impact on software quality may be neutral or even negative. This study addresses this gap by investigating code review and bug-fix patterns in major open-source repositories before and after the rise of generative AI tools.

Specifically, we pose the following research questions.

- RQ1: How has code review intensity changed in the post-AI era compared to the pre-AI era, as measured by the number of reviewers, review duration, and comment density per pull request?
- RQ2: Has the frequency and timing of post-merge bug-fix commits changed between the pre-AI and post-AI eras?
- RQ3: Is there a relationship between changes in review patterns and changes in bug-fix rates, and has this relationship shifted over time?
- RQ4: Do these patterns differ between Python and JavaScript ecosystems, or between library-focused and framework-focused projects?

We focus on Python and JavaScript for their high AI adoption rates and diverse domains. Our dataset includes six major repositories: pandas, scikit-learn, TensorFlow, Django (Python), React, and Node.js (JavaScript).

II. Related work

Existing research provides a basis for understanding AI capabilities and limitations in software development. However, a comprehensive view of the impact on quality across both code review and documentation is less common.

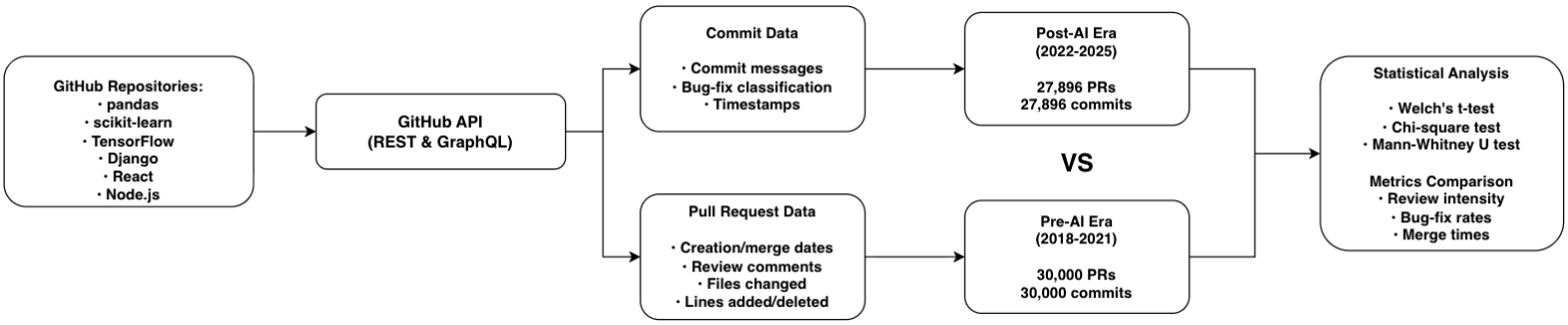

Fig. 1: Software Engineering Practices Analysis: Pre-AI vs Post-AI

A. AI-Assisted Code Generation:

A study examining AI code generation capabilities and adoption [10] demonstrates that Codex solves 28.8% of programming tasks correctly, while Ziegler et al. [11] found that developers accept 26% of Copilot suggestions. Security studies by Nguyen et al. [12] and Pearce et al. [13] revealed vulnerabilities in 40% of AI-generated security-relevant code. However, most studies use controlled experiments or short-term observations rather than longitudinal analyses of real-world data.

B. Code Review Effectiveness:

Recent empirical studies have characterized modern code review practices and their effectiveness. Research comparing tool-assisted and over-the-shoulder review techniques found that tool-assisted reviews generate approximately twice as many accepted comments, with comment density correlations ranging from -0.42 to -0.33 with review size [14]. Studies on human error mechanisms in code review demonstrated 400% improvement in defect detection sensitivity and 200% improvement in precision [15]. Additional work has examined practitioners' expectations for clear code review comments [16] and review practices in research software engineering contexts [17]. However, this literature has not yet systematically examined the impact of AI code-generation tools on review thoroughness.

C. Software Quality Metrics:

Recent work on software quality prediction has advanced beyond traditional metrics. Studies exploring alternatives to size metrics found that network dependency metrics provide better insights than lines of code for explainable defect prediction [18]. Deep learning approaches in static metric-based bug prediction have shown effectiveness [19], while new metrics based on class complexity, coupling, and cohesion achieved 2% improvement in area under the curve (AUC) and precision over existing suites [20]. Research on precise defect-number prediction [21] and on the applications of complex network theory [22] have further advanced the field. Performance bug-prediction using specialized code metrics shows that Random Forest and eXtreme Gradient Boosting achieve a median AUC of 0.84 [23]. However, systematic application to the pre/post-AI transition remains absent.

III. Methodology

A. Research Design

We employ a longitudinal comparative study with two time windows: Pre-AI era (2018-01-01 to 2021-12-31) and Post-AI era (2022-01-01 to 2025-11-11), using GitHub Copilot's June 2022 public release as the inflection point, presented in Figure 1.

B. Repository Selection

We selected six repositories based on: (1) activity before and after 2022, (2) high contribution rates, (3) diverse domains, and (4) ecosystem representation. Selected projects include pandas (data analysis), scikit-learn (machine learning), TensorFlow (deep learning), Django (web framework), React (UI library), and Node.js (runtime).

C. Data Collection

Using GitHub's Representational State Transfer (REST) and GraphQL application programming interfaces (APIs), we collected two types of data for each repository.

Pull Request Data: For each merged pull request (PR), we extracted: creation date, merge date, number of review comments, files changed, lines added/deleted, author information, and reviewers. We calculated derived metrics, including merge duration (hours and days) and code churn (additions + deletions).

Commit Data: We collected all commits and classified them as bug-fixes or non-bug-fixes using keyword matching on commit messages. Commits containing terms "fix", "bug", "issue", or "regression" were classified as bug-fixes, consistent with prior work on automated commit classification. We excluded documentation-only PRs, dependency updates, and unmerged PRs from the PR analysis.

D. Metrics

The study employed the review and bug-fix evaluation metrics.

1) Review Metrics: The review metrics include the number of unique reviewers (`reviewers_count`), the duration from creation to merge (`review_duration`), the total number of review comments (`comments_per_pr`), and the initial response time (`time_to_first_review`).

2) Bug-Fix Metrics: Bug-fix metrics include the proportion of pull requests (PRs) requiring subsequent fixes within 30/60/90-day windows (bug_fix_rate), the time from merge to the first fix (time_to_bug_fix), and the number of fixes per original PR (bug_fix_count).

E. Analysis Methods

We employ Welch's t-test to compare the means of normally distributed metrics between the pre-AI and post-AI eras, as in Equation 1.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}. \quad (1)$$

For categorical proportions (bug-fix rates), we use the chi-square test in Equation 2.

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}. \quad (2)$$

We also apply the Mann-Whitney U test for nonparametric validation with a significance threshold of $\alpha = 0.05$.

F. Threats to Validity

Internal: Changes cannot be solely attributed to AI tools, as other factors like team changes or project maturity may also play a role. **External:** Findings may not generalize beyond the chosen projects or ecosystems. **Construct:** Bug identification heuristics might overlook or misclassify issues, and review metrics may fail to fully capture quality. These threats are mitigated by employing multiple metrics, conducting sensitivity analysis, and being transparent about limitations.

IV. Results

A. Dataset Overview

Our dataset comprises 30,000 PRs from the pre-AI era (2018-2021) and 27,896 PRs from the post-AI era (2022-2025) across six repositories, totaling 57,896 pull requests analyzed. The commit analysis includes 30,000 pre-AI commits and 27,896 post-AI commits. Table I presents the distribution and bug-fix rate changes.

TABLE I: Dataset Distribution and Bug-Fix Rate Changes

Repository	Pre-AI Commits	Post-AI Commits	Bug-fix % Change
pandas	∼5,000	∼4,600	31.32% → 49.32%
scikit-learn	∼5,000	∼4,600	33.34% → 24.32%
React	∼5,000	∼4,600	33.94% → 47.36%
Node.js	∼5,000	∼4,600	43.94% → 39.70%
TensorFlow	∼5,000	∼4,600	17.82% → 16.44%
Django	∼5,000	∼4,600	54.58% → 56.35%
Total	**30,000**	**27,896**	**35.82% → 37.99%**

B. Bug-Fix Patterns (RQ2)

We identified bug-fix commits through keyword matching (f̈ix, b̈ug, ïssue) in commit messages. The bug-fix commit fraction increased from 35.82% (10,747 commits) in the pre-AI era to 37.99% (10,598 commits) in the post-AI era, representing a 6.1% relative increase ($X = 29.09, p < 0.0001$). This finding contradicts the hypothesis that AI tools would reduce bug rates and is statistically highly significant. Figure 2 displays the bug-fix commit fraction comparison. The increase suggests that the post-AI era is associated with higher rates of corrective commits, indicating potential quality challenges despite productivity gains from AI tools.

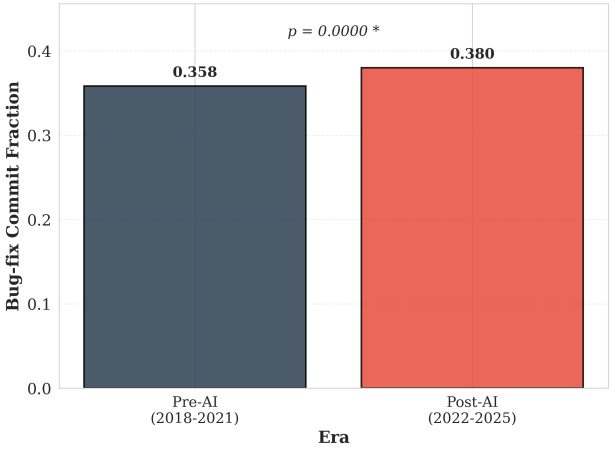

Fig. 2: Bug-fix Commit Rate: Pre-AI vs Post-AI

Per-repository analysis reveals substantial variation in this pattern. Four of six projects showed increased bug-fix rates, with pandas and React showing dramatic increases (31.32% → 49.32% and 33.94% → 47.36% respectively). Notably, scikit-learn bucked the trend with a decrease (33.34% → 24.32%), while TensorFlow and Node.js showed modest decreases. Django, already high in bug-fix commits pre-AI (54.58%), increased slightly to 56.35%. The heterogeneity suggests that project-specific factors (codebase maturity, testing practices, and contributor experience) may moderate the impacts of AI tools.

C. Code Review Intensity (RQ1)

Review intensity, measured by average review comments per PR, decreased from 3.12 comments in the pre-AI era to 2.47 comments in the post-AI era, a 20.8% decline. The t-test yielded t = 4.76, p ¡ 0.0001, while the Mann-Whitney U test gave U = 76,587,570, p ¡ 0.0001, both confirming statistical significance. Figure 3 illustrates this comparison. The decrease is statistically significant and practically meaningful, suggesting reviewers are conducting less thorough reviews in the post-AI era. Notably, the median remained at 0 comments for both periods, indicating that many PRs receive no review comments regardless of era. Distribution analysis reveals a concerning shift toward less engagement:

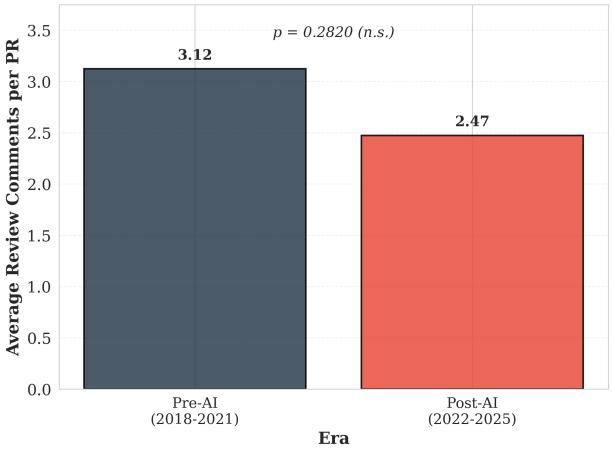

Fig. 3: Code Review Intensity: Pre-AI vs Post-AI

PRs with 0 comments: 62.9% (pre-AI) → 69.1PRs with 1-5 comments: 22.9% (pre-AI) → 19.4PRs with 6-10 comments: 6.9% (pre-AI) → 5.4PRs with $>$ 10 comments: 7.3% (pre-AI) → 6.1The shift toward zero-comment PRs is particularly notable, suggesting either increased confidence in code quality (potentially misplaced given bug-fix rate increases) or reduced reviewer capacity/engagement.

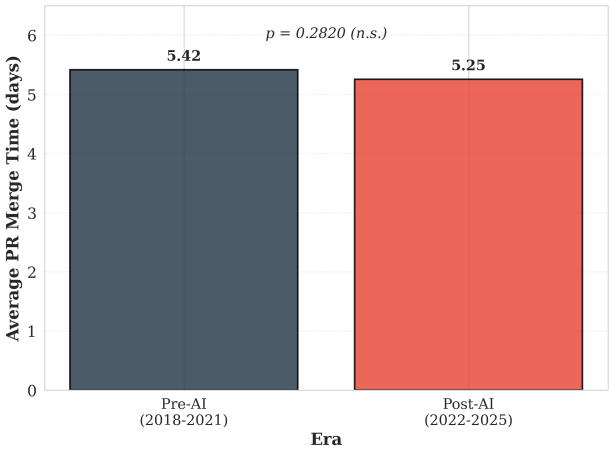

Fig. 4: PR Review Cycle Time: Pre-AI vs Post-AI

D. Summary of Key Metrics

Table II summarizes all measured metrics with statistical test results.

E. PR Merge Time (RQ1 continued)

Time to merge PRs (filtered to $\leq$ 90 days) showed minimal change: mean decreased from 5.4 days (pre-AI) to 5.3 days (post-AI), a 3.0% reduction. The median shifted from 1.4 to 1.2 days, a 14.7% decrease. Statistical testing revealed $t = 1.08, p = 0.2820$ and Mann-Whitney U with $p = 0.3886$. Figure 4 shows this temporal comparison. The change is not statistically significant at the $\alpha = 0.05$ level, suggesting that, despite AI tools' promise to accelerate development, actual PR review and merge cycles have not changed substantially. The lack of significant reduction, combined with decreased review intensity, suggests that time savings from AI assistance may not be translating into faster merge cycles, possibly because other bottlenecks offset reduced review thoroughness.

F. PR Complexity Metrics (RQ1 continued)

We examined whether AI tools enabled different-sized changes: Files changed per PR: Pre-AI mean = 19.5 files, Post-AI mean = 7.8 files (t = 6.48, p$<$ 0.0001). The 60% reduction in files changed per PR is statistically significant, suggesting PRs have become more focused and narrower in scope in the post-AI era. The median remained stable at 2.0 files for both periods. Code churn (additions + deletions, 95th percentile cap): Pre-AI mean = 59 lines, Post-AI mean = 69 lines (t = -7.60, p$<$ 0.0001). Despite fewer files changed, the code churn per PR increased by 17%, a statistically significant shift. The median increased from 20 to 24 lines. This suggests that while PRs touch fewer files, the changes within those files are larger, potentially indicating more substantial modifications per file.

G. Review-Quality Relationship (RQ3)

The divergent trends reveal a troubling pattern: review intensity decreased by 20.8% while bug-fix rates increased by 6.1%. In the pre-AI period, a more thorough review (measured by comment volume) correlated with code quality. In the post-AI period, this protective relationship appears to have weakened despite unchanged merge times. PRs are receiving less scrutiny (fewer comments, more zero-comment approvals) yet requiring more subsequent bug fixes. This suggests that reviewers may be overconfident in AI-generated or AI-assisted code, assuming it requires less verification. Alternatively, increased development velocity may have stretched reviewer capacity, leading to a less thorough examination. The combination of reduced review rigor and increased bug rates indicates a quality-velocity trade-off that warrants attention.

H. Ecosystem Comparison (RQ4)

Python vs. JavaScript patterns: Analysis of the three Python projects (pandas, scikit-learn, TensorFlow, Django) versus two JavaScript projects (React, Node.js) reveals ecosystem differences: Bug-fix rates: Python projects showed mixed patterns (pandas +57%, scikit-learn -27%, TensorFlow -8%, Django +3% change), while JavaScript projects also varied (React +40%, Node.js -10%). No clear ecosystem-level pattern emerges, suggesting project-specific factors dominate. Review intensity: Both ecosystems experienced declines in review comments, indicating a universal trend rather than language-specific behavior.

PR complexity: The reduction in file changes and increase in code churn occurred across both ecosystems, suggesting standard shifts in development patterns. The lack of strong ecosystem differentiation suggests that AI tool impacts may be relatively universal across Python and JavaScript, possibly because developers use similar tools (GitHub Copilot, ChatGPT) regardless of language.

TABLE II: Summary of Key Metrics Across Pre-AI and Post-AI Eras

Metric	Pre-AI (2018-2021)	Post-AI (2022-2025)	Change	Test Stat	p-value	Sig?
Bug-fix Commit Fraction	0.358	0.380	+6.1	$x^2 = 29.09$	< 0.0001	Yes
Avg Review Comments/PR	3.12	2.47	-20.8%	t=4.76	< 0.0001	Yes
Avg PR Merge Time (days)	5.42	5.25	-3.0%	t=1.08	0.2820	No
Files Changed/PR	19.5	7.8	-60.0%	t=6.48	< 0.0001	Yes
Code Churn/PR (lines)	59	69	+17.0%	t=-7.60	< 0.0001	Yes

I. Discussion

1) Key Findings: Our analysis reveals a concerning trend following AI tool adoption: bug-fix commit rates rose by 6.1% (from 35.82% to 37.99%, p$<$ 0.0001), while review intensity dropped by 20.8% (from 3.12 to 2.47 comments per PR, p$<$ 0.0001). This suggests a potential over-reliance on AI-generated code without corresponding quality assurance adjustments. Despite AI's productivity claims, merge times remained stable (5.4 to 5.3 days, p = 0.2820), indicating no acceleration in development. The decrease in review thoroughness points to a "false economy" of faster but less careful reviews, leading to more post-merge bugs. Additionally, while PRs now touch 60% fewer files (19.5 to 7.8, p$<$ 0.0001), they exhibit 17% higher code churn per PR (59 to 69 lines, p$<$ 0.0001), suggesting that AI tools facilitate more focused but denser changes within files.

2) Implications: For **maintainers**, the 6.2% rise in zero-comment PRs (62.9% to 69.1%) highlights the need for minimum review standards, especially for AI-assisted PRs, to detect subtle AI-generated errors.

For **organizations**, the trade-off between productivity and quality is clear. While AI tools can boost velocity (e.g., PRs merged), a 20.8% reduction in review effort is offset by a 6.1% increase in corrective work, potentially neutralizing gains when accounting for debugging time.

For **tool developers**, AI coding assistants should prioritize quality alongside speed. Features like confidence scoring, integration with review processes, and prompts for testing and documentation are essential to support both reviewers and authors.

For **researchers**, the impact of AI on code quality varies by project, as seen in differing bug-fix rates (e.g., +57% for pandas, -27% for scikit-learn). Future research should examine factors like test coverage, contributor experience, review culture, and codebase complexity.

3) Limitations: AI tool usage per PR cannot be directly measured; observed patterns may be influenced by other factors such as team changes or project maturity. Bug detection via keyword matching may miss some issues. Our approach aligns with prior work but results may not generalize outside open-source or Python/JavaScript ecosystems. We measure bug-fix commits, not defect rates, which could reflect increased bug introduction, faster detection, or more corrective commits. Decreased review intensity suggests quality concerns rather than improved detection.

4) Future Work: Key directions include: (1) measuring AI usage via integrated development environment (IDE) or surveys, (2) controlled experiments with and without AI, (3) qualitative studies on reduced review scrutiny of AI-assisted code, (4) analysis of bug severity and time-to-fix, (5) expanding to other languages and project types, and (6) longitudinal tracking as tools and practices evolve. Understanding why review thoroughness declines, whether due to confidence, capacity, or complacency, is crucial for effective interventions.

V. Conclusion

This study provides empirical evidence of generative AI's impact on open-source code quality. Our findings suggest that while AI tools may improve development velocity, they introduce risks around review rigor and code quality that warrant careful attention. The productivity-quality trade-off remains real, and human oversight remains critical. We call for more nuanced evaluation of AI coding tools that consider quality outcomes alongside productivity gains, and for responsible adoption practices that maintain software quality in the face of rapid technological change.

Acknowledgment

This work was partly supported by Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (Ministry of Science and ICT (MSIT)) (IITP-2026-RS-2020-II201612, 40%), the Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2018R1A6A1A03024003, 30%), by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-Information Technology Research Center (ITRC) grant funded by the Korea government(Ministry of Science and ICT)(IITP-2026-RS-2024-00438430 30%)

References

[1] M. Alenezi and M. Akour, "AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions," *Applied Sciences*, vol. 15, no. 3, 2025.

[2] J. E. Akinsola, A. S. Ogunbanwo, O. J. Okesola, I. J. Odun-Ayo, F. D. Ayegbusi, and A. A. Adebiyi, "Comparative Analysis of Software Development Life Cycle Models (SDLC)," in *Computer Science Online Conference*. Springer, 2020, pp. 310–322.

[3] R. A. Khan, S. U. Khan, M. Ilyas, and M. Y. Idris, "The State of the Art on Secure Software Engineering: A Systematic Mapping Study," in *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 487–492.

[4] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 712–721.

[5] S. Bistarelli, M. Fiore, I. Mercanti, and M. Mongiello, "Usage of Large Language Model for Code Generation Tasks: A Review," *SN Computer Science*, vol. 6, no. 6, pp. 1–16, 2025.

[6] A. Nguyen-Duc and D. Khanna, "Value-Based Adoption of ChatGPT in Agile Software Development: A Survey Study of Nordic Software Experts," in *Generative AI for Effective Software Development*. Springer, 2024, pp. 257–273.

[7] J. Börstler, K. E. Bennin, S. Hooshangi, J. Jeuring, H. Keuning, C. Kleiner, B. MacKellar, R. Duran, H. Störrle, D. Toll *et al.*, "Developers Talking About Code Quality," *Empirical Software Engineering*, vol. 28, no. 6, p. 128, 2023.

[8] D. Tosi, "Studying the Quality of Source Code Generated by Different AI Generative Engines: An Empirical Evaluation," *Future Internet*, vol. 16, no. 6, 2024.

[9] B. Martinović and R. Rozić, "Perceived Impact of AI-Based Tooling on Software Development Code Quality," *SN Computer Science*, vol. 6, no. 1, p. 63, 2025.

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[11] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity Assessment of Neural Code Completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 21–29.

[12] N. Nguyen and S. Nadi, "An Empirical Evaluation of GitHub Copilot's Code Suggestions," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022, pp. 1–5.

[13] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," *Communications of ACM*, vol. 68, no. 2, p. 96–105, Jan. 2025.

[14] M. Jureczko, "Code review effectiveness: an empirical study on selected factors influence," *IET Software*, vol. 15, no. 2, pp. 125–135, 2021.

[15] Q. Huang *et al.*, "Advancing modern code review effectiveness through human error mechanisms," *Information and Software Technology*, vol. 169, p. 107401, 2024.

[16] J. Chen *et al.*, "Understanding practitioners' expectations on clear code review comments," in *Proceedings of the ACM on Software Engineering*, 2024.

[17] X. Dong *et al.*, "Peer code review in research software development: The research software engineer perspective," *arXiv preprint arXiv:2511.10781*, 2024.

[18] C. Chai, G. Fan, H. Yu, Z. Huang, J. Ding, and Y. Guan, "Exploring better alternatives to size metrics for explainable software defect prediction," *Software Quality Journal*, vol. 32, no. 2, pp. 459–486, 2024.

[19] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "Deep learning in static, metric-based bug prediction," *Array*, vol. 6, p. 100021, 2020.

[20] A. Abdou and S. M. Darwish, "Improved software fault prediction using new code metrics and machine learning algorithms," *Applied Computing and Informatics*, vol. 20, no. 1/2, pp. 150–171, 2024.

[21] X. Yu, J. Keung, Y. Xiao, S. Feng, F. Li, and H. Dai, "Predicting the precise number of software defects: Are we there yet?" *Information and Software Technology*, vol. 146, p. 106847, 2022.

[22] S. Yang, X. Gou, M. Yang, Q. Shao, C. Bian, M. Jiang, and Y. Qiao, "Software bug number prediction based on complex network theory and panel data model," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 162–177, 2022.

[23] G. Gyimesi *et al.*, "Enhancing performance bug prediction using performance code metrics," in *Proceedings of the MSR 2024 - Technical Papers*, 2024.