JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

The Impact of AI-Generated Solutions on Software Architecture and Productivity.

Rohan Ramesh Jadhav[1]

Prof. Madhuri Chaudhari[2], Prof. Rajat Hedav[3], Dr. Madhulika Gupta[4], Prof. Pooja Ghadge[5]

1 Students, Dr. D. Y. Patil Centre for Management & Research, Chikhali, Pune, India
2 Assistant Professor , Department of Business Administration, Dr. D. Y. Patil Centre for Management & Research, Chikhali, Pune, India
3 Assistant Professor , Department of Computer Applications, Dr. D. Y. Patil Centre for Management & Research, Chikhali, Pune, India
4 Associate Professor , Department of Computer Applications, Dr. D. Y. Patil Centre for Management & Research, Chikhali, Pune, India
5 Assistant Professor , Department of Computer Applications, Dr. D. Y. Patil Centre for Management & Research, Chikhali, Pune, India

Abstract: **This research investigates the growing influence of Artificial Intelligence (AI) tools on developer productivity and software architecture quality across the Software Development Life Cycle (SDLC). With an increasing number of software practitioners incorporating AI into their daily workflows, questions arise regarding the short-term and long-term implications of these tools. While AI applications such as code generation, debugging automation, test case creation, requirement analysis, and architecture suggestion are becoming increasingly common, concerns remain regarding reliability, accuracy, complexity, and structural soundness of AI-generated outputs. A survey involving 40 software practitioners was conducted to explore these aspects in detail. The study examines how AI tools assist engineers, the frequency of use across SDLC stages, the perceived productivity improvements, and the architectural consequences of adopting AI solutions. Results show that AI offers substantial productivity gains, particularly in repetitive and early-stage development tasks, with many respondents reporting time savings exceeding 35%. Despite challenges such as integration overhead and inconsistencies in architectural decisions, evidence suggests that AI-generated outputs do not significantly degrade software architectural quality when properly reviewed by human experts. This research contributes important insights into the responsible use of AI tools and emphasizes how developers can leverage these tools effectively while maintaining high-quality architecture.**

KEYWORDS:

AI Tools; Software Engineering; Software Architecture; Productivity; SDLC; Code Generation; Multi-Agent AI; Software Quality

INTRODUCTION

Artificial Intelligence (AI) continues to reshape the landscape of software engineering by providing automated solutions that assist developers at multiple stages of the Software Development Life Cycle (SDLC). AI tools have rapidly transitioned from novelty applications to essential development companions. Modern systems such as GPT-4o, GitHub Copilot, ChatGPT, Cursor IDE, and multi-agent engineering frameworks offer real-time assistance in generating code, analyzing system requirements, creating documentation, improving architecture, and performing debugging tasks. According to the 2024 Stack Overflow Developer Survey, nearly 76.6% of software developers actively use or plan to use AI tools. This widespread adoption demonstrates the increasing reliance on AI to support development productivity. However, despite these benefits, AI-generated solutions may introduce risks, particularly in architecture-driven software design. Concerns revolve around issues such as oversimplified design patterns, poor modularity, increased coupling, lack of domain understanding, and long-term maintainability challenges. This study aims to bridge the gap between perceived benefits and potential drawbacks by exploring two primary research questions: (i) How do AI tools influence the productivity of software developers? and (ii) Do AI-generated solutions negatively impact software architecture quality? Through a detailed survey of 40 practitioners, this research sheds light on industry perspectives regarding the opportunities and limitations presented by AI in software engineering.

OBJECTIVE OF THE STUDY

- To examine how AI tools, influence the productivity of software developers across different stages of the Software Development Life Cycle (SDLC).
- To evaluate the quality, reliability, and correctness of AI-generated solutions used in software development tasks.
- To assess whether the adoption of AI-generated solutions has any negative impact on key software architecture attributes such as maintainability, modifiability, cohesion, coupling, and extendibility.
- To identify the common challenges developers, face when integrating AI-generated code into existing systems.
- To analyze the extent to which AI tools support or hinder decision-making in software architecture and design.

REVIEW OF LITERATURE

- *Sam Altman, GitHub & Microsoft Research — "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot" (2023)*

This controlled experiment evaluates how GitHub Copilot changes developer productivity and cognitive load. Using time-to-completion and correctness metrics across a set of programming tasks, the study finds that Copilot speeds up task completion, reduces mental effort, and increases developer satisfaction for many routine tasks. However, benefits vary by task type and developer experience; developers still must review and edit suggestions, and some complex design tasks receive little benefit. The paper concludes that Copilot-like tools are effective copilots for implementation-level work but must be integrated with code review and design practices to avoid architectural drift.

- *G. Amasanti et al. — "The Impact of AI-Generated Solutions on Software Architecture and Productivity: Results from a Survey Study" (2025)*

This recent survey of software practitioners examines how adopting AI-generated solutions affects architectural decisions and team productivity. The authors collected responses from engineers across domains and correlated perceived productivity gains with project complexity and AI usage patterns. Results show strong perceived productivity increases for routine coding and refactoring, but diminishing returns as architectural complexity rises; teams reported more time spent validating AI outputs on complex modules. The study highlights the need for architecture-aware prompting, stricter validation pipelines, and governance to manage long-term architectural quality.

- *Case Study — "The impact of GitHub Copilot on developer productivity: a software engineering body of knowledge perspective" (industrial case, 2024)*

This organizational case study (automotive sector) evaluates Copilot's effects on real-world development metrics: sprint velocity, defect rates, and review cycles. The study observed modest gains in short-term throughput and time-savings on boilerplate code, but also an increase in time spent on code review and integration for non-trivial modules. Authors caution that productivity metrics (velocity) can be misleading unless paired with measures of maintainability and architecture conformance. They recommend training, policy updates, and tooling that enforces architectural constraints when using AI suggestions.

- L. *Banh — "Copiloting the future: How generative AI transforms software engineering" (2025, review article)*

A grounded review synthesizing empirical studies, case reports, and conceptual work about GenAI in software engineering. The author maps benefits (faster prototyping, reduced routine work, improved developer experience) against risks (security vulnerabilities, brittle or inconsistent APIs, architectural erosion). The review emphasizes sociotechnical adoption factors — governance, upskilling, and workflow redesign — and proposes a framework for integrating GenAI into architecture-centric development: (1) tool qualification, (2) architecture-aware prompts, (3) automated validation and (4) human oversight. Practical recommendations target R&D and platform teams responsible for long-lived systems.

- Comprehensive *survey — "A Survey on Large Language Models for Code Generation" (ACM, 2025)*

This broad survey categorizes LLMs for code by capabilities (completion, synthesis, documentation, test generation) and reviews empirical work on performance, safety, and developer workflows. It discusses how LLMs change architecture-level tasks (design sketches, API recommendations, dependency selection) and notes that while LLMs help propose options, they often lack the global project context required for consistent architectural choices. The survey calls for research on model conditioning with architecture metadata, toolchains that embed architecture rules, and benchmarks that measure architectural correctness and long-term maintainability.

 Security/*quality analysis — Veracode industry study (reported 2025)*

An industry analysis found that a large share of AI-generated code samples contain security weaknesses (e.g., input-validation, injection vulnerabilities), especially in complex languages and security-sensitive contexts. The report warns that "vibe coding" (accepting AI outputs without security prompts or checks) can introduce vulnerabilities at scale. Practical implications: integrate SAST/DAST into AI workflows, add security-aware prompting templates, and require automated security gates for AI-produced patches to protect architectural integrity. This work underscores that productivity gains must be balanced with automated quality/security controls.

 Model Evaluation & Threat Research (METR) study reported in Business Insider — "Experienced engineers less productive with AI coding tools" (2025)

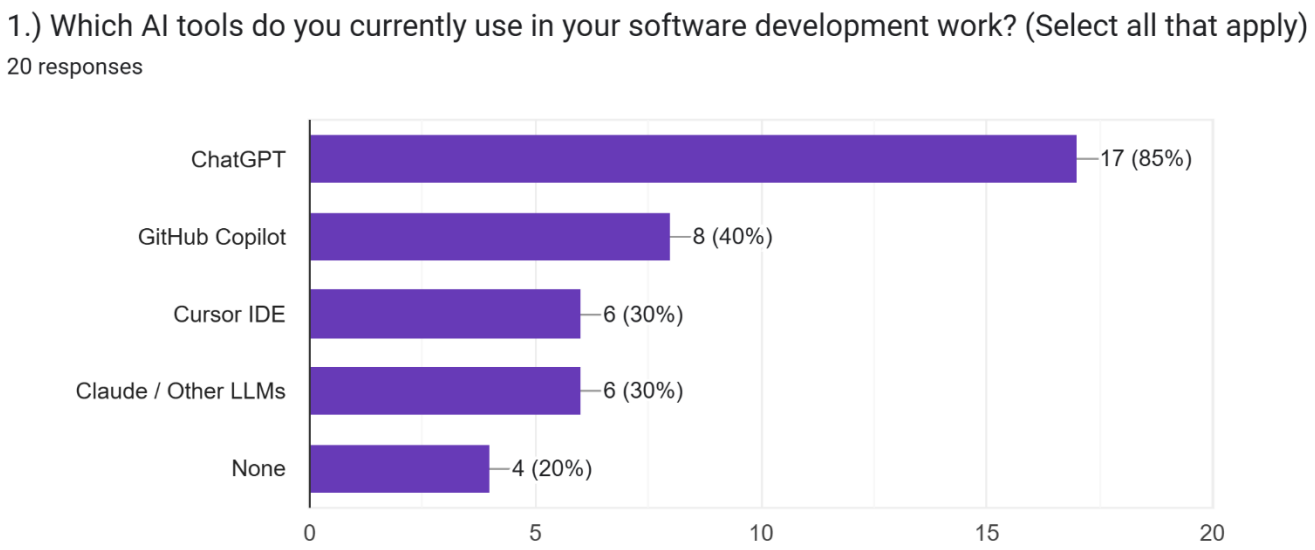
An empirical lab study reported that for experienced developers working on familiar, complex codebases, some AI tools actually reduced measured productivity because experts spent extra time prompting, reviewing, and correcting AI outputs. Interestingly, participants still *felt* more productive despite objective slowdowns — a caution about perceived vs. measured productivity. The study suggests that AI tools are most effective when they remove routine cognitive load, but they can impede experts on deep architectural tasks unless tightly integrated with project knowledge and expert workflows.

 ArXiv/systematic analysis — "The Impact of LLM-Assistants on Software Developer Productivity" (2025)
This paper reviews methodologies used to study LLM assistants and synthesizes results across controlled experiments, field studies, and surveys. Key takeaways: controlled trials (e.g., Copilot) often show time and mental-effort improvements on microtasks; field studies reveal mixed results influenced by team processes, onboarding, and architecture complexity; and surveys indicate widespread perceived productivity gains but also concerns about maintainability and skill atrophy. The authors recommend multi-method evaluation (lab + field + longitudinal) and new metrics that capture architecture-level outcomes (design consistency, coupling, long-term defect rates) rather than only short-term speed metrics.

RESEARCH METHODOLOGY

This research adopts a quantitative methodology using a structured survey distributed among 40 practitioners from different software engineering backgrounds. Participants included software developers, architects, testers, and engineering leads who have used AI tools in their workflows. The survey was designed to capture detailed insights into how AI is utilized across various SDLC phases, the perceived productivity improvements, and the architectural impact of AI-generated solutions.
The survey questionnaire contained 20 questions categorized into three primary dimensions:
1. AI Usage Patterns: Frequency of use, AI tools used, SDLC phases supported.
2. Productivity Impact: Time savings, reduction in effort, improvement in developer efficiency.
3. Architectural Effects: Maintainability, extendibility, modifiability, coupling, cohesion, and code quality.
To structure the analysis, the Goal-Question-Metric (GQM) model was applied. This model ensures each research goal is linked to targeted questions and measurable outputs. Data collected from the survey was analyzed using descriptive statistics, comparative assessment, and thematic analysis for open-ended responses.

DATA ANALYSIS AND INTERPRETATION

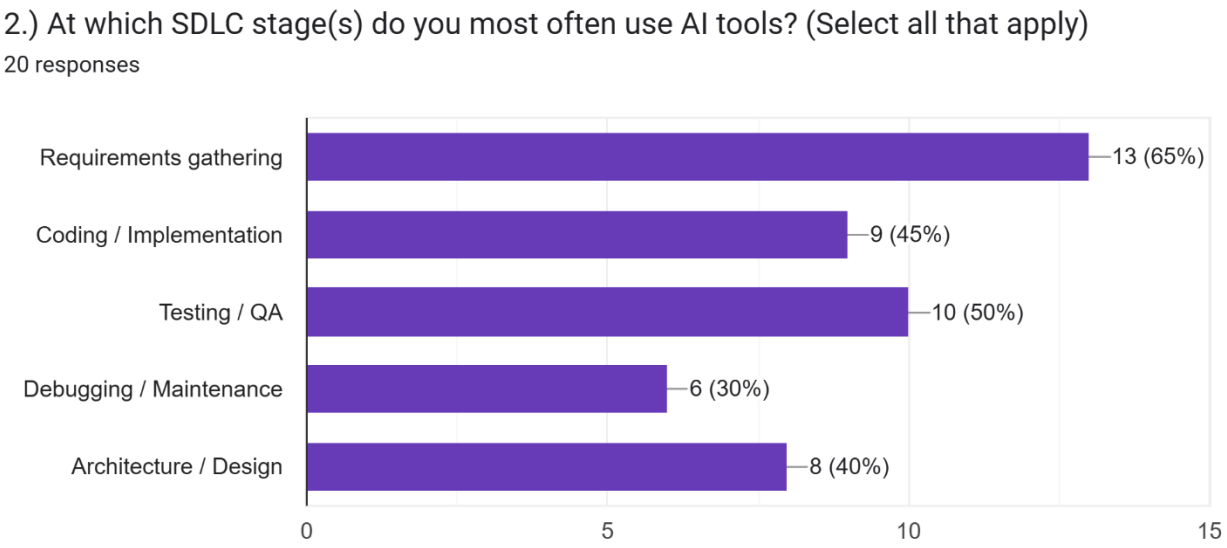


Graph 1

Options	Responds	Percentage
ChatGPT	17	85%
GitHub Copilot	8	40%
Cursor IDE	6	30%
Claude / Other LLms	6	30%
None	4	20%

Table 1

Conclusion: - Majority of software engineers prefer ChatGPT as there go to AI tool



Graph 2

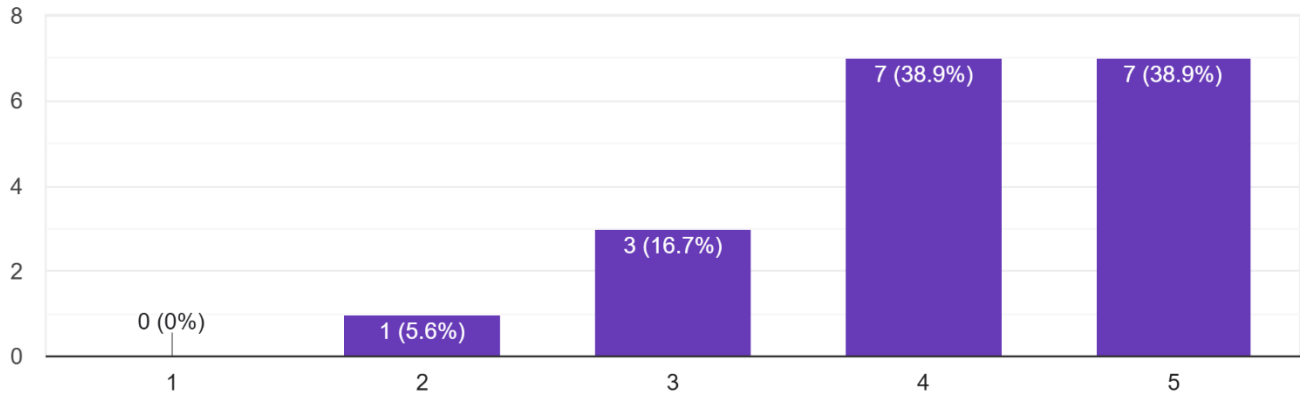
Options	Responds	Percentage
Requirements gathering	13	65%
Coding / Implementation	9	45%
Testing / QA	10	50%
Debugging / Maintenance	6	30%
Architecture / Design	8	40%

Table 2

Conclusion: - AI tools is most preferred to use in Requirements Gathering and Testing/QA phase in SDLC

3.) How would you rate the impact of AI tools on your overall productivity?

18 responses



Graph 3

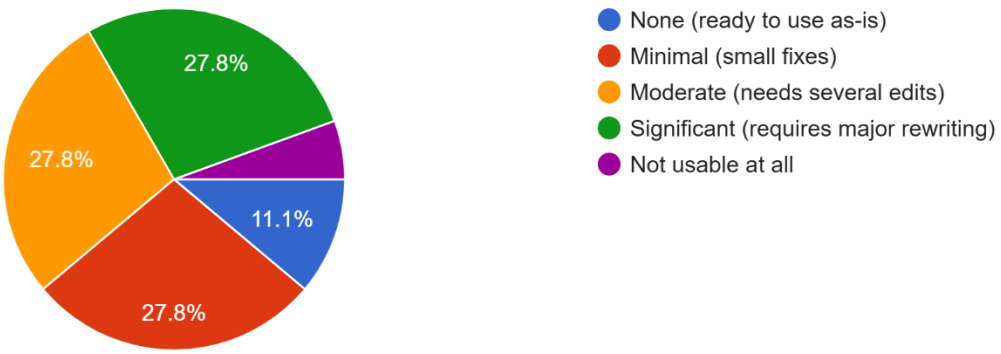
Rating	Responds	Percentage
1	0	0%
2	5	5.6%
3	16	16.7%
4	38	38.9%
5	38	38.9%
Total	100	100%

Table 3

Conclusion: - Majority of software engineers responded that AI tools has improved the productivity

4.) On average, how much rework or correction does AI-generated code require before integration? (Select one)

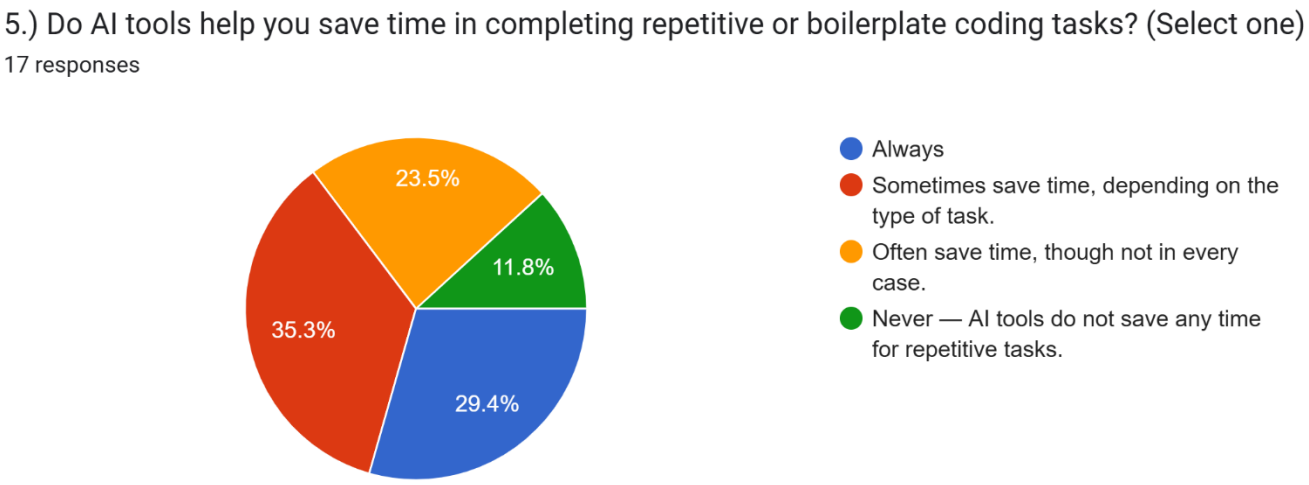
18 responses



Option	Responds	Percentage
None	11	11.1%
Minimal	27	27.8%
Moderate	27	27.8%
Significant	27	27.8%
Not Usable at all	0	0%
Total	100	100%

Table 4

Conclusion: - Most developers reported that AI-generated code generally requires moderate to significant rework, indicating that while AI speeds up development, its outputs are not ready for use without substantial human correction.



Graph 5

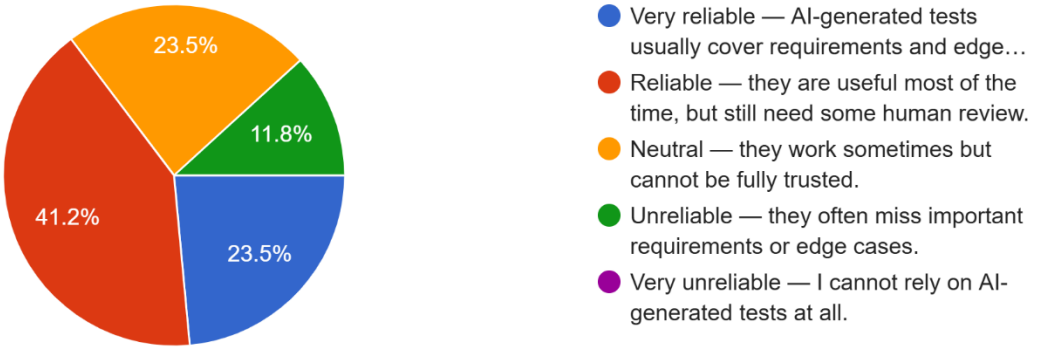
Option	Responds	Percentage
Always	29	29.4%
Sometime	35	35.3%
Often	23	23.5%
Never	11	11.8%
Total	100	100%

Table 5

Conclusion: - Majority of Software Engineers think that AI tools helps in saving time in completing repetitive or boilerplate coding tasks

6.) How reliable do you find AI-generated test cases compared to manually written tests? (Select one)

17 responses



Graph 6

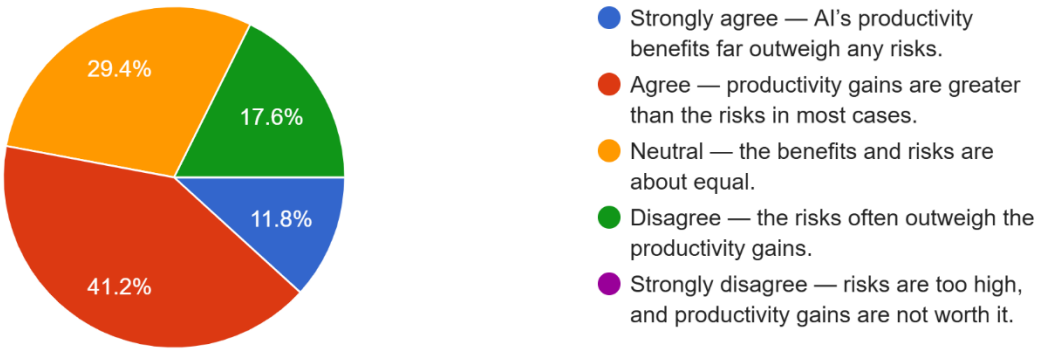
Option	Responds	Percentage
Very Reliable	23	23.5%
Reliable	41	41.2%
Neutral	23	23.5%
Unreliable	11	11.8%
Very Unreliable	0	0%
Total	100	100%

Table 6

Conclusion: - Majority of Engineers thinks that AI is reliable but need some human review

7.) Do the productivity benefits of AI outweigh the potential risks of architectural erosion and technical debt? (Select one)

17 responses

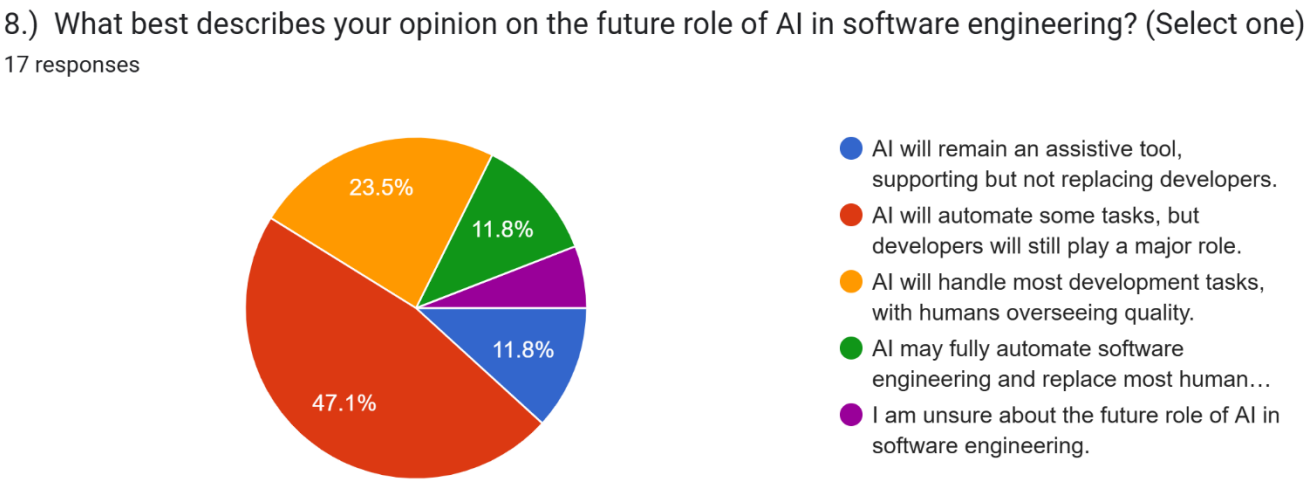


Graph 7

Option	Responds	Percentage
Strongly Agree	11	11.8%
Agree	41	41.2%
Neutral	29	29.4%
Disagree	17	17.6%
Total	100	100%

Table 7

Conclusion: - Majority of Engineers Agree — productivity gains are greater than the risks in most cases.



Graph 8

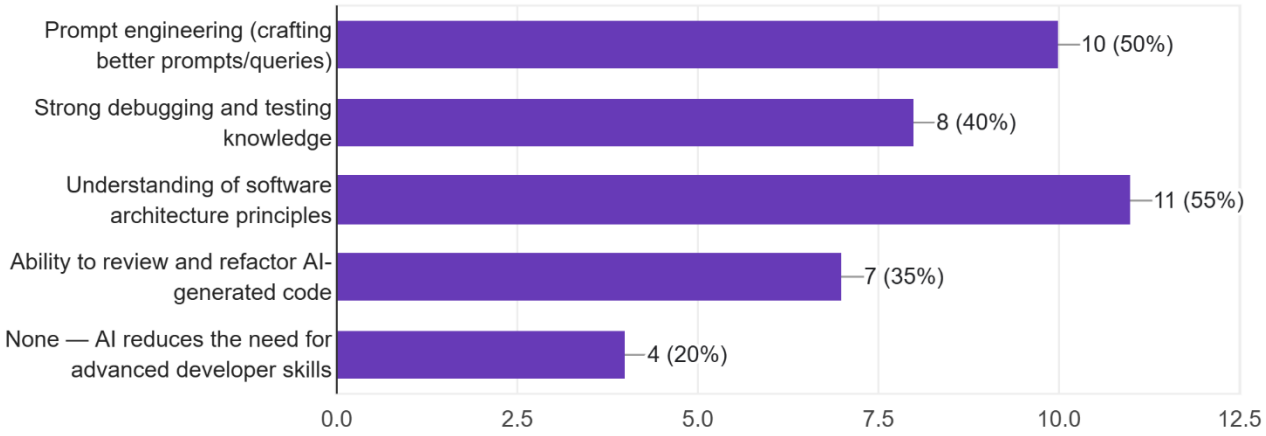
Option	Responds	Percentage
AI will remain an assistive tool	11	11.8%
AI will automate some tasks	47	47.1%
AI will handle most development tasks	23	23.5%
AI may fully automate	11	11.8%
Total	100	100%

Table 8

Conclusion: - Majority of Engineers think AI will automate some tasks, but developers will still play a major role.

9.) What skills do you think developers need to maximize productivity with AI tools? (Select all that apply)

20 responses



Graph 9

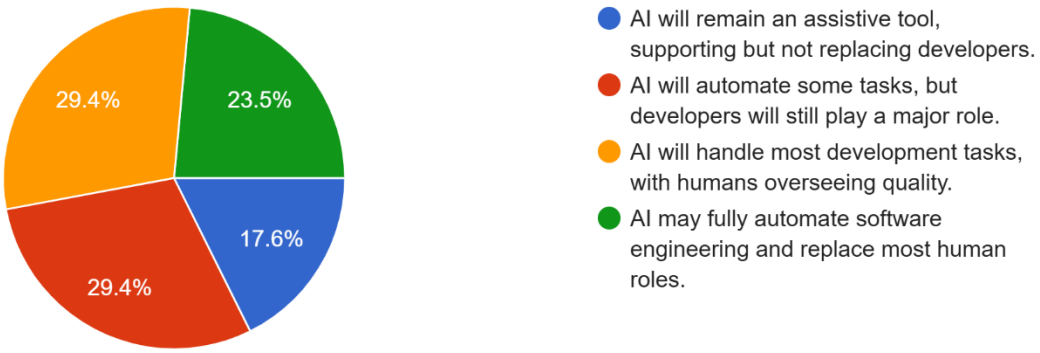
Options	Responds	Percentage
Prompt engineering	10	50%
Strong debugging and testing knowledge	8	40%
Understanding of software architecture principles	11	55%
Ability to review and refactor AI-generated code	7	35%
None	4	20%

Table 9

Conclusion: - Majority of Engineers think developers need to maximize Understanding of software architecture principles

10.) What best describes your opinion on the future role of AI in software engineering? (Select one)

17 responses



Graph 10

Option	Responds	Percentage
AI will remain an assistive tool	17	17.6%
AI will automate some tasks	29	29.4%
AI will handle most development tasks	29	29.4%
AI may fully automate	23	23.5%
Total	100	100%

Table 10

Conclusion: - Majority of Engineers think AI will handle most development tasks, with humans overseeing quality.

FINDINGS OF THE STUDY

Key Findings:
- Small AI-generated snippets maintain cohesion and coupling quality.
- Large-scale AI outputs reduce architectural clarity and maintainability.
- Productivity improvements are strongly influenced by prompt quality and developer experience.

Suggestions:
- Use AI primarily for small, scoped problems rather than full applications.
- Ensure manual review of AI-generated solutions to maintain architectural integrity.
- Train engineers in prompt engineering to maximize AI productivity benefits.

CONCLUSION

AI tools significantly improve developer productivity and accelerate several SDLC tasks. While integration challenges exist—particularly when working with established codebases—AI-generated solutions do not inherently degrade software architecture quality. When combined with proper human review, AI serves as a powerful assistant that enhances development efficiency without compromising structural integrity. As AI continues to evolve, its adoption will become increasingly essential for competitive software development. Future research should focus on improving AI's handling of large and complex codebases, reducing friction in integration, and enhancing AI's ability to support architectural decision-making. Additionally, domain-specific AI models could be developed for different SDLC stages, offering more reliable and context-aware outputs.

REFERENCES

[1] **OpenAI & Pilipiszyn, A.** (2023, January 8). *GPT-3 apps*. OpenAI. https://openai.com/blog/gpt-3-apps/

[2] **Basili, V. R., & Weiss, D. M.** (1984). A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering, SE-10*(6), 728–738. https://doi.org/10.1109/TSE.1984.5010301

[3] **Bencheikh, L., & Höglund, N.** (2023). *Exploring the efficacy of ChatGPT in generating requirements: An experimental study*.

[4] **Ciniselli, M., Cooper, N., Pascarella, L., Poshyvanyk, D., Di Penta, M., & Bavota, G.** (2021). An empirical study on the usage of BERT models for code completion. *arXiv*. https://arxiv.org/abs/2103.07115

[5] **Jain, N., Vaidyanath, S., Iyer, A., Natarajan, N., Parthasarathy, S., Rajamani, S., & Sharma, R.** (2022). Jigsaw: Large language models meet program synthesis. In *Proceedings of the 44th International Conference on Software Engineering* (pp. 1219–1231). ACM.