International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2025: 7.089

AI and the Future of Software Engineering: Expertise, Employment, and Policy

Shruti[1], Jayanthi M[2]

[1]School of Science and Computer Studies, CMR University, #5, Bhuvanagiri, OMBR Layout, Bangalore- 560 043, Karnataka, India
Email: *shrutijujar16549[at]gmail.com*

[2]School of Science and Computer Studies, CMR University, #5, Bhuvanagiri, OMBR Layout, Bangalore- 560 043, Karnataka, India
Email: *jayanthi.m[at]cmr.edu.in*

Abstract: ***Artificial intelligence has moved rapidly from an experimental add-on to a routine part of software development, with AI coding assistants now writing functions, generating tests, and, in agentic form, completing multi-step engineering tasks with limited human oversight. This raises a familiar question: does automating part of a skilled trade erode the expertise around it, and who bears the cost? Drawing on randomized trials, labour-market studies, industry surveys, and code-security audits, this report examines how AI adoption is reshaping human expertise, employment, and code quality in software engineering. Entry-level employment is contracting even as overall technical demand rises, and AI-written code merged without scrutiny carries more security flaws and technical debt. Software engineering is being reorganized rather than elimin0ated, with value shifting toward architecture, verification, and accountability. The report closes with recommendations for employers, institutions, and regulators.***

Keywords: Artificial intelligence; software engineering; human expertise; labour market; employment; AI coding assistants; skill formation; code security; technical debt; policy

1. Introduction

Few technologies have reshaped knowledge work as quickly as artificial intelligence, and software engineering has felt this more than most professions. Language models trained on large code corpora can write functions from short descriptions, translate code between languages, generate tests, summarise unfamiliar codebases, and, in agentic form, carry out multi-step engineering tasks with only occasional human check-ins. Adoption has been fast: within roughly three years of general availability, generative AI tools reached about half of surveyed workplaces, outpacing the diffusion of the personal computer or the internet [13], and most professional developers now use an AI coding assistant for part of their daily work [14].

This raises a familiar question whenever automation meets a skilled trade: does the tool cheapen the expertise around it, or does that expertise move elsewhere? The question is sharper in software engineering because the profession has long been defined by the visible act of writing working code. Once a system can produce plausible code within seconds, fast implementation stops being a reliable marker of skill- which motivates this report's focus on whether AI is redefining engineering skill itself, who enters and stays in the profession, and what happens to quality and security once code is produced faster than it can be reviewed.

The stakes are concrete: payroll research covering millions of workers documents a marked drop in employment among the youngest developers since generative AI became widely available, even as hiring for experienced engineers keeps increasing [2]; productivity experiments swing considerably by tool generation and task type [1]; and a substantial share of AI-generated code contains security vulnerabilities [34]. This report synthesises this evidence to examine changing expertise, employment effects, and code-quality risk, and closes with policy recommendations for employers, universities, and regulators.

2. Literature Review

a) *AI and the Transformation of Development Practice*

Early qualitative research found developers use AI coding assistants mainly to cut repetitive typing, look up syntax, and speed through boilerplate while remaining selective about which suggestions they accept [12]. Studies inside large technology companies found continued use makes developers view AI tools as more useful over time but does little to increase trust in their correctness; output is still treated as a first draft requiring review [15]. Recent surveys describe a shift from single-line autocomplete toward agentic tools that plan changes across multiple files, letting junior contributors attempt more ambitious work while adding to review burden [3].

b) Productivity Effects: A Contested and Moving Target

Productivity research looks contradictory until tool generation and task type are accounted for. Peng et al. found developers using GitHub Copilot on a tightly scoped task finished about 56% faster than controls [8]. A 2025 METR trial with experienced developers in large, unfamiliar codebases instead found early-2025 AI tools slowed developers by about 19%, despite developers predicting a 24% speed-up beforehand and still believing afterward that the tools had helped [1]; repeating the trial with newer models flipped the result to roughly an 18% speed-up [17]. Industry surveys fall between these extremes: McKinsey reports about 46% time savings on routine tasks but under 10% on complex work [9], and Google's DORA team found individual effectiveness up 17% even as delivery stability fell nearly 10% [29]. AI appears to help most with routine work and least with complex, unfamiliar systems, and perceived speed often diverges from measured speed.

c) Code Quality, Security, and Technical Debt

Early evaluation of GitHub Copilot found about 40% of generated code contained vulnerabilities, worse in C than Python, and that developers judged their own insecure AI-assisted code as safe [35]. Testing across more than a hundred large language models found only about 55% of AI-generated samples free of known security issues, a figure that has barely improved as syntactic correctness has [34]. Production-repository studies link AI-assisted development to higher code churn and more architectural debt when AI contributions are merged without adequate review [28][30], moving engineering cost from the point code is written to when it must later be maintained [28]. Cognitive-load research adds that AI shifts programming effort from writing to evaluating code, which is often the more demanding, error-prone task [31].

d) Labor Market and Workforce Effects

Payroll data covering millions of workers shows employment among developers aged 22–25 fell nearly 20% between late 2022 and mid-2025, while employment among more experienced developers at the same firms grew [2][4]. Job postings stayed roughly a third below pre-pandemic levels through late 2025, with junior postings down more sharply [2]- consistent with automation displacing routine, lower-complexity tasks first [4]. Yet overall technology employment keeps growing: the WEF projects a net gain of 78 million jobs by 2030, with software and AI-related roles among the fastest-growing, even as 92 million more automatable jobs are lost [18][19][20]. Near-term contraction is concentrated among the least experienced doing the most automatable work; medium-term growth favours roles needing deep judgment or new hybrid AI-oversight skills.

Existing research largely treats productivity, code quality, and employment as separate literatures that rarely reference each other, obscuring a compound risk: if AI is simultaneously narrowing entry-level hiring, shifting effort toward evaluation, and increasing code needing expert review, the pool of engineers available to do that review may itself be shrinking.

3. Methodology

This report uses a structured narrative synthesis, suited to combining randomized trials, large surveys, payroll-based labour research, and code-security audits whose designs differ too much for formal meta-analysis. Sources were drawn from peer-reviewed venues, preprints, and reports from established research organisations, prioritising randomized/ quasi-experimental designs and large administrative datasets, with industry commentary used mainly for context. Evidence was organised around AI's effect on productivity by experience level, on employment by career stage, and the risks revealed by code-quality research. The evidence base is young and fast-changing; labour-market findings are correlational, not causal, and should be read as the best current synthesis rather than a final word [16].

AI and the Changing Nature of Human Expertise

Fast, correct code has long signalled engineering skill, but AI assistants let developers at almost any level produce plausible, often functional code quickly, making that signal harder to read. Practitioners describe this as an amplifier rather than an equalizer: AI tends to make both strong and struggling engineers faster without supplying the judgment to recognise when a solution does not fit the system it is added to [3]. As one senior engineer put it, agentic AI does not turn a junior engineer into a senior one, because output quality still depends on the judgment of whoever directs the tool [3].

Three forms of expertise resist automation: architectural judgment, which fits a design to constraints rarely spelled out in a prompt; requirement interpretation, turning ambiguous business needs into a coherent specification using organisational knowledge no code-focused model can observe; and accountability, the need for someone who can answer for a decision when software fails.

This affects how expertise gets built. As routine implementation is automated, the apprenticeship model in which junior engineers build judgment under supervision comes under strain. Evaluating code is often more demanding than writing it, and strong evaluative judgment usually depends on prior hands-on generation experience [31] — so automation that lightens coding may make it harder to build the judgment needed to supervise it safely.

Employment Effects and the Restructuring of Software Careers

Employment outcomes diverge sharply by career stage. Payroll research covering 2021–2025 found employment among developers aged 22–25 fell nearly 20% from its late-2022 peak- timed with wider AI adoption- while employment among more experienced developers grew 6–12% [2][4]. Job postings through late 2025 stayed about a third below 2020 levels, with junior postings down further than senior ones [2]. This matches a selective-automation pattern: routine, clearly defined junior work is what AI tools currently handle most reliably, while judgment-heavy senior work stays comparatively hard to automate [4], so organisations under budget pressure may substitute AI for junior hires rather than using it to make them more productive.

This coincides with continued growth in overall technical demand: the WEF projects a net gain of 78 million jobs by 2030, driven substantially by AI and software-development roles, even while projecting 92 million losses in more automatable categories [18][19]. The contraction and growth describe different groups over different timeframes- the former concentrated among the least experienced doing automatable work, the latter in roles needing deep judgment or new hybrid AI-oversight skills.

Compensation data reinforces this: entry-level roles built around evaluating and directing AI pay meaningfully more than roles that do not require AI fluency [7]. There is also an individual skill-formation risk- reaching for AI beyond current understanding can crowd out learning, while heavy reliance on already-mastered tasks can erode hands-on skill [27], compounding the pipeline risk unless AI use is balanced against independent practice.

Table I Summarises the main productivity studies discussed above.

Table I: Comparative Summary of Empirical Findings on AI Coding Productivity

Study / Source	Method	Key Finding
METR (Becker et al., 2025) [1]	RCT; 16 experienced developers, 246 tasks	Early-2025 AI slowed developers ~19%, vs. a predicted 24% speed-up
METR follow-up (2026) [17]	Same RCT design, newer model generation	Effect reversed to roughly an 18% speed-up
Peng et al. [8]	RCT; GitHub Copilot on a bounded task	Assisted group finished ~56% faster than controls
McKinsey survey [9]	Industry survey, ~4,500 developers	~46% time savings on routine tasks; <10% on complex tasks
Google DORA [29]	Industry-wide developer survey	Individual effectiveness +17%; delivery stability – 10%

Risks: Code Quality, Security, and Skill Atrophy

Beyond employment, the growing volume of AI-generated code carries risk. Testing across more than a hundred large language models found only about 55% of samples free of common security flaws, a figure that has stayed roughly flat even as syntactic correctness has improved [34]. Independent evaluation finds a similar pattern of architecturally significant flaws, including a rise in privilege-escalation vulnerabilities that need contextual reasoning to spot and that automated static analysis catches poorly [33]. Disclosed vulnerabilities traceable to AI-generated code are rising quickly, and researchers believe documented cases understate the problem across the open-source ecosystem [32][33]. Production-repository analysis links AI-assisted development to higher code churn and to mismatched dependencies and design patterns that surface when contributions are merged without adequate review [28][30], moving risk from the point code is written to when it must be maintained [28].

Emerging Roles and Opportunities

These risks should not overshadow real opportunities. AI tooling is lowering the barrier to building software, letting domain experts with no formal programming background prototype applications directly ("vibe coding"), bringing subject-matter expertise into development even as it raises the security concerns above [32][34]. New specialist roles are emerging around responsible AI use: engineers who validate AI-generated code before production, those who build review pipelines for agentic tools, and staff responsible for AI governance. Compensation data suggests these roles already command a pay premium [7]. For experienced engineers, the opportunity is spending more time on architecture and product judgment rather than routine implementation; surveys report 30–60% time savings on coding, testing, and documentation when AI is used well, time that could shift toward higher-value work if organisations deliberately make room for it [7].

Synthesis of Challenges and Opportunities

Table II summarises the challenges and opportunities identified above, leading into the policy recommendations that follow.

Table II: Summary of Principal Challenges and Opportunities

Challenges	Opportunities
Sharp drop in entry-level hiring	Faster iteration; less time on boilerplate coding
Skill atrophy from unscrutinised AI output	Wider access to programming for non-specialists
Elevated security debt in AI-generated code	New specialist roles in AI oversight and governance
Rising code churn and technical debt	More time freed for architecture and product judgment
Weakening apprenticeship pipeline	Wage premiums for strong AI-collaboration skills

4. Policy Recommendations

1) *Recommendations for Employers*

- Keep entry-level hiring deliberate; treat it as investment in the future senior pipeline, not a cost to trim.
- Make human review mandatory for AI-generated code before production, especially in security- or business-critical systems [34].
- Redesign junior roles around AI-assisted evaluation rather than assuming less code needs writing; preserve tasks that build architectural and debugging judgment.
- Track productivity and quality together (time-to-merge alongside defect rates and churn) to avoid individual gains masking system-level costs [29].
- Provide structured training on verifying and critically evaluating AI output.

2) Recommendations for Educational Institutions

- Keep foundational instruction in algorithms, debugging, and system design rigorous and AI-independent.
- Build AI-collaboration skills explicitly into the curriculum.
- Expand applied, project-based learning and industry partnerships to offset shrinking apprenticeship opportunities.

3) Recommendations for Policymakers and Standards Bodies

- Encourage transparency requirements for AI-generated code in safety-critical software.
- Support reskilling and apprenticeship funding for displaced early-career workers [18][19].
- Continue funding independent research into labour-market and security effects [1][17].
- Encourage consistent industry metrics for evaluating AI coding tools.

5. Discussion and Future Outlook

The evidence supports one conclusion: AI is not eliminating software engineering but redistributing where value, opportunity, and risk sit within it. Value is moving from routine implementation toward architecture, verification, and judgment; opportunity is narrowing at entry level even as it widens for experienced engineers and new AI-adjacent roles; and risk is moving from the point code is written to the point it must be maintained.

This creates a tension organisations have not resolved: the judgment that makes senior engineers valuable was

historically built through years of junior-level work — precisely the work AI now handles most readily. If junior hiring keeps shrinking without a deliberate alternative for building judgment, the long-run supply of senior expertise that safe AI-augmented development depends on could itself decline.

The pace of change is notable- METR's estimate flipped from a 19% slowdown to an 18% speed-up within about a year [1][17]- so findings should be treated as provisional. Future research should prioritise longitudinal tracking of skill development under AI-assisted work, causal labour-market research separating AI's effect from macroeconomic factors, and comparative studies of which organisational practices best capture AI's benefits while containing its risks.

6. Conclusion

This report has examined how AI is affecting human expertise, employment, and the direction of software engineering. AI is not replacing the profession, but changing what expertise means within it — moving value from routine code production toward architecture, verification, and accountable judgment. Alongside this sits a measurable contraction in early-career employment, rising security and technical debt tied to unreviewed AI code, and a real risk of skill atrophy without disciplined verification.

At the same time, the shift is opening specialist roles, widening participation in building software, and freeing experienced engineers for judgment-intensive work. The central challenge is not whether to adopt AI- adoption is already near-universal- but how to do so while preserving the judgment, skill-formation pipeline, and accountability that safe AI-augmented engineering depends on.

References

[1] J. Becker, N. Rush, E. Barnes, and D. Rein, "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity," METR / arXiv:2507.09089, 2025.
[2] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot," arXiv:2302.06590, 2023.
[3] McKinsey & Company, developer productivity survey (≈4,500 developers), cited in "AI Coding Productivity Study Data: What METR, McKinsey, and GitHub Found in 2026," Value Add VC, 2026.
[4] Google DORA Team, "State of AI-Assisted Software Development," referenced in "AI-Generated Code Poses Security, Bloat Challenges," Dark Reading, 2025.
[5] Stanford Digital Economy Lab, cited in "The Productivity-Reliability Paradox: Specification-Driven Governance for AI-Augmented Software Development," arXiv:2605.01160, 2026.
[6] Stanford HAI, "AI Index Report 2026," referenced in "The Productivity-Reliability Paradox," arXiv:2605.01160, 2026.
[7] World Economic Forum, "The Future of Jobs Report 2025," Geneva, Switzerland, Jan. 2025.
[8] Veracode, "AI-Generated Code Security Risks: What Developers Must Know," Veracode Blog, 2025.
[9] B. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," IEEE Symposium on Security and Privacy, 2022; and N. Perry et al., "Do Users Write More Insecure Code with AI Assistants?" ACM CCS, 2023.
[10] G. Voskoboynikov et al., "AI's Impact on Software Engineers in 2026: Key Trends," The Pragmatic Engineer newsletter, 2026.