The Augmentative Role of AI in Software Engineering: A Global Expert Survey

Avighna Mane[1]

Received September 24, 2025
Accepted December 6, 2025
Electronic access December 15, 2025

This study investigates how artificial intelligence (AI) is reshaping the practice of software engineering through a qualitative set of expert interviews with professionals across multiple global organizations. It examines how AI tools affect daily workflows, required skills, automation choices, and developer training. Findings show that AI can improve productivity, documentation, and prototyping speed, shifting engineers from repetitive coding toward design, verification, and collaboration. New skills, such as prompt engineering and AI oversight, are becoming essential as organizations utilize risk-based frameworks to determine which tasks to automate and which to keep under human review. Overall, AI currently acts as an augmentative force rather than a replacement for developers or software engineers. The human-in-the-loop approach is essential for creativity, ethics, and context-sensitive decision-making. For students and educators, the message is clear: mastering coding fundamentals is still critical, but learning to guide and validate AI systems is now part of what it means to be an engineer.

Keywords: *AI, software engineering, productivity, prompt engineering, survey, generative AI, human-in-the-loop, automation, developer skills, governance, education, innovation.*

Introduction

AI tools are increasingly being integrated into nearly every stage of modern software engineering, from planning and design to coding, testing, documentation, and operations. This change has sparked a loud debate: will AI take developer jobs, or will it help developers do more? Recent ecosystem studies show that adoption is already widespread, with developers reporting higher productivity and frequent usage.

The goal of this study is to capture what experienced engineers and leaders across industries are observing in practice so that high school and college students can prepare realistically. To achieve this, the researcher conducted interviews with experts across very different settings and asked each of them the same five questions. Their answers tell a clear story: AI is shifting the work of engineers upward, away from repetitive tasks and toward design, verification, safety, and creative problem-solving.

The evidence points toward augmentation of human work rather than outright replacement.

Historically, tools that raised the level of abstraction in programming led to more software, not fewer engineers. Compilers abstracted assembly; IDEs automated build/test cycles; low-code tools let non-experts prototype ideas. Each time, the field expanded, and the nature of work shifted.

The experts I interviewed describe AI as another similar leap: it removes friction on common tasks and makes it faster to get to a working draft, but humans still decide what should be built, check that it is safe, and connect it to real-world needs. One expert captured the mood simply: "Every single company is experimenting with AI, not to cut engineers, but to help them do higher-level work."

This project is a qualitative expert interview study designed to capture professional perspectives across multiple industries. Therefore, the results capture the professional views and practices rather than experimental measurements, with the intent to surface early patterns that future, data-driven studies can test.

Background and related work

Recent academic and industry studies have shown that AI coding assistants are producing measurable gains in developer productivity and code quality. Large empirical evaluations such as[1,2] have demonstrated that tools like GitHub Copilot can reduce time to completion and contribute to developer focus on complex logic rather than syntax errors.

Controlled experiments have further revealed that generative-AI pair programming can accelerate documentation and testing without lowering overall code reliability[3]. Conversely, several studies caution that AI-generated code may introduce subtle vulnerabilities or flaws in logic, underscoring the need for human oversight[4,5].

[1] ***Class of 2026 - Saint John's High School, Shrewsbury, MA***

Methodical reviews have confirmed (to a great extent) such mixed outcomes emphasizing that productivity benefits depend heavily on developer experience and validation practices[6].

Together, these works position AI as a supportive but fallible collaborator, rather as an enabler that enhances speed and consistency but requires human judgment to ensure intended quality outcome.

While most organizations emphasize the importance of skills like "prompt engineering" and "human-in-the-loop" validation, research shows there are substantial variations in how these concepts are practiced across software engineering contexts. Studies at major conferences highlight that prompt engineering, how developers craft and refine prompts for AI models has become a key determinant of code quality and documentation effectiveness[7,8].

Other empirical work reveals that effective prompting and collaborative model orchestration improve both comprehension and maintainability of generated code[9].

Multi-agent frameworks and AI-assisted reviewer systems demonstrate how human oversight can be systematically integrated into the development lifecycle[10].

Educational researchers also report that exposure to these tools reshapes how students learn problem-solving and debugging in computing courses.

Collectively, these findings confirm that prompt engineering and responsible oversight are not transient trends but emerging competencies that every engineer and student must master to effectively collaborate with AI systems. As a general definition, "prompt engineering" is the process of writing and refining prompts so AI tools (models) return useful and accurate results, and human-in-the-loop validation is the process where human reviewers check and approve AI outputs before use, although these practices vary across organizations.

Some experts describe prompt engineering as a core, lasting skill comparable to learning programming with natural language, while others view it as transitional as AI models become more context aware. Empirical studies on generative AI in collaborative and multilingual development environments support this view, showing that prompting is gradually evolving into "vibe-coding," where developers express intent rather than explicit syntax[11–13].

Broader reviews of AI-augmented workflows emphasize that this transformation is altering not only how code is written but how engineers conceptualize automation and reasoning in their work[14].

These findings suggest that while future tools may require less deliberate prompt crafting, the underlying communication and abstraction skills will remain integral to software engineering practice. Similarly, human-in-the-loop validation remains a cornerstone of responsible AI adoption, though the degree of rigor varies across organizations and regulatory contexts. Empirical frameworks show that human validation is indispensable in managing automation risk, particularly in safety-critical and privacy-sensitive domains.

Several controlled studies confirm that organizations employing risk-tiered or confidence-based review systems achieve higher reliability and maintain code quality while scaling AI-assisted workflows[15]. Comparative analyses of enterprise and academic settings further suggest that maintaining human checkpoints mitigates hallucinations, bias propagation, and security vulnerabilities introduced by automated generation. Collectively, these works reinforce that effective collaboration between humans and AI is not merely procedural but a governance mechanism ensuring both ethical and technical robustness.

These findings echo the patterns my interviewees described; usefulness in routine work, measurable gains in development speed, and the continued importance of human oversight. Controlled field studies on generative AI's role in software teams show that productivity improvements arise primarily through faster iteration, better documentation, and increased developer satisfaction rather than full automation.

At the same time, large-scale empirical evaluations emphasize that AI-assisted coding must be accompanied by robust review and security processes to mitigate potential vulnerabilities. Comparative analyses of industry and educational environments highlight that collaboration between human reviewers and AI systems strengthens both code reliability and learning outcomes[16].

Collectively, these results, together with prior meta-reviews of AI integration in software engineering[17] confirms that AI currently acts as a collaborative augmentation tool enhancing productivity and innovation while reinforcing the irreplaceable value of human validation.

These results align closely with prior studies that were much broader. For example, McKinsey & Company (2024)[18] also reported that AI improves developer productivity mainly through faster iteration and better documentation rather than full automation, as described by most interviewees[19] found that human-AI "copiloting" structures are most effective when software engineers retain control of design and validations. My study reinforces these findings through qualitative evidence from industry experts across multiple sectors, implying that AI currently acts as a collaborative tool rather than a replacement for engineers.

The following sections describe the study's methodology and present the results, organized around five Research Questions (RQ1–RQ5) that align with the interview prompts. This paper focuses on a few representative samples of existing studies rather than an exhaustive literature review. Because the research was conducted at the high-school level, the emphasis was placed on collecting original expert insights rather than compiling many peer-reviewed sources. I plan to expand the

research by integrating a broader body of work.

Methods

This study used a semi-structured interview method with five predetermined questions to ensure comparable answers. This study uses a qualitative research design focused on expert interviews rather than a quantitative survey, underscoring insights based on perceptions from experienced professionals.

Participants and Data Collection

The researcher interviewed experts from organizations listed below over a period of two months in the summer of 2025 and captured their responses in digital format.

All interviews were conducted virtually through video or phone calls from late June, July and early August 2025. Each conversation lasted about 30 minutes. With every participant's permission, audio recordings were captured and transcribed to allow for review and ensure accuracy. The recordings were transcribed and then summarized into detailed notes.

Table 1

Organization	Description
Google	AI, cloud, and search solutions provider
IBM	Enterprise technology and services company
SAP	Enterprise software company
Boston Scientific	MedTech device company
KPMG	One of the Big Four consulting firms
McKinsey & Company	Strategy consulting firm
UNICEF	Global NGO
JAFRA Cosmetics	Consumer goods and beauty brand
Tech Mates Group (TMG)	Technology services and systems integrator
Independent AI Consultant	AI and data independent consultant

Interviewees were identified through professional networks and referrals to capture a diverse cross-section of perspectives. Selection criteria included:

- direct involvement in the adoption or oversight of AI in software engineering.
- seniority or subject-matter expertise that enabled them to speak to both technical and organizational practices; and
- representation across sectors, including technology providers, consulting firms, MedTech, NGOs, and consumer goods.

While most interviewees held senior roles, the diversity of industries provided a broad view of how AI is shaping software engineering globally.

Table 2 below captures key demographic information about the participants. Most of them were senior professionals such as software engineering managers, architects, or AI specialists with 8 to 25 years of experience. The sample included experts from North America (6), Europe (3), and Asia-Pacific (2), ensuring a mix of regional perspectives. The organizations represented both private-sector firms (technology, consulting, MedTech, and consumer goods) and public sector or nonprofit institutions (UNICEF). All participants had direct involvement in AI adoption or oversight within software development workflows in their respective organizations.

Table 2

Role Type	# of Participants	Years of Experience (Range)	Region	Industry Category
Software Engineers / Architects	5	8–18 years	North America, Europe	Technology & Cloud
AI / ML Specialists	3	10–20 years	Global	AI / Data Science
Engineering or Product Managers	5	12–25 years	North America, Asia-Pacific	Enterprise & Consulting
NGO / Public Sector Technologists	2	8–15 years	Global	Nonprofit / Public Sector
Total	**15**			

Although the sample covers a wide range of industries, it is indeed small (15 people from 11 organizations) and thus should be viewed as exploratory. Given that most interviewees were senior leaders, the findings may reflect a high-level organizational strategy more than the day-to-day experiences of junior or mid-level engineers. Because all the data and input come from the interviews, the findings should be interpreted as qualitative insights based on perception, and not statistically validated effects.

Here are the questions covered:

1. Day-to-day changes with AI;
2. How roles/skills are evolving;
3. What to automate with AI and what to keep manual;
4. How engineers are trained to work with AI; and
5. Which metrics teams use to measure impact?

Coding and Analysis

Each conversation was summarized and coded for repeating themes as follows:

- workflow automation (code generation, testing, documentation, refactoring, ticketing),
- role/skill changes (prompting, tool orchestration, security/privacy, architecture/design, data literacy),
- automation decision frameworks (risk tiering, human-in-the-loop),
- training models (formal programs, peer learning, self-study), and
- metrics (time, quality/security indicators, and developer satisfaction).

Each interview summary was entered into an Excel sheet, where every distinct comment was matched with one or more of the five themes.

For example, if a participant mentioned "AI helps us generate test code faster," that statement was placed under workflow automation. If another organization mentioned the same idea, I added one more count to that theme. The percentage values shown in the figures equal the number of organizations that mentioned a theme divided by the total (11). This method demonstrated how frequently a topic appeared across all the interviews rather than measuring it statistically.

The coding process followed a single-coder approach. To maintain consistency, each transcript file was reviewed twice and coded according to clear inclusion rules. The five main themes were defined as follows:

- **Workflow automation**—mentions of AI tools used for coding, testing, documentation, or debugging.
- **Role and skill changes**—statements about evolving developer responsibilities or emerging technical skills.
- **Automation decisions**—comments describing when to automate versus when to keep human oversight.
- **Training approaches**—examples of programs, bootcamps, or informal learning used to teach AI skills.
- **Evaluation metrics**—any measures or indicators (time saved, bugs, satisfaction) used to assess AI's effect.

Comments that did not clearly relate to any of the above categories were excluded. Because the coding was completed by one researcher, the chair of computer science at my school reviewed them to confirm category placement and overall consistency.

Note that the same five questions listed above were asked in each interview to keep responses consistent. I also used short follow-up questions when the experts mentioned something new or answers that I didn't understand. Some examples include "Can you give a quick example of that?" or "How does that compare with before AI tools were used?" Such clarifying questions helped to address the overall goals and structure of the interview.

It is important to note that most of the quantitative figures provided by participants (e.g., % of time saved or acceptable AI output) were estimates based on their professional experience rather than results of any controlled experiments. These numbers reflect perceptions of impact within their organizations rather than statistical proof, and as such, they should be interpreted accordingly.

To analyze the qualitative data systematically and show the findings visually, I created an Excel workbook where each organization's responses were coded into themes (e.g., workflow automation, role changes, training models). The workbook allowed calculation of how many organizations mentioned each theme, which in turn informed the percentages reported in the Results section.

Charts and figures presented in the paper are drawn directly from this coded mapping workbook, ensuring consistency and transparency.

The coded Excel workbook included one worksheet per research question (RQ1 to RQ5). Each row represented an organization, and each column corresponded to a theme, such as workflow automation, skill changes, or training methods. When someone mentioned a specific theme, that cell was coded "1." Furthermore, percentages shown in the Results were calculated by dividing the number of organizations that mentioned a theme by the total number of organizations (11).

An anonymized version of this template, with example coding rows, can be provided upon request.

Ethics and informed consent

All participants gave informed consent before the interviews. To protect confidentiality, interviewees are anonymized in this report; no confidential information, such as source code, protected health information, or private code identifying organi-

zational details, was requested or recorded. Transcripts and coded data are stored securely and accessed only by the author. All quotes used are short, anonymized, and presented with permission.

All quotations and numerical estimates attributed to interviewees were checked against original transcripts during manuscript preparation.

Results

All numeric percentages and estimates reported in this section are participants' informed estimates or coded counts drawn from the interviews and the author's coding of those interviews (i.e., based on perception or estimates), not measurements from controlled experiments.

Whenever percentages or numerical ranges appear in this section, they merely represent interviewee estimates rather than measured values.

The percentages in the figures mentioned above come from counting how many of the 11 organizations mentioned each theme in the coded Excel sheet, not from quantitative evaluation.

These findings only summarize what the expert interviewee subjects report and what was counted in the coding workbook; they are certainly not outcomes from controlled experiments.

RQ1. How has AI changed day-to-day engineering work?

Across organizations, generative AI helps with similar kinds of tasks, like generating boilerplate code, suggesting unit tests, writing or improving documentation, and helping developers navigate unfamiliar codebases faster. The consistent claim was that these tools do not remove engineers from the loop; instead, they lower the time to a reasonable draft and make it easier to explore options.

These frequencies are derived from the coded Excel workbook described in the Methods section, where each theme was counted based on how many organizations mentioned it.

The table below summarizes how many of the 11 organizations mentioned each opportunity theme during the interviews.

Productivity gains and code quality improvements were most frequently cited, while innovation and collaboration were also common benefits. Several interviewees shared specific metrics for acceptable AI output in their teams' routine work.

In these interviews, "acceptable output" referred to AI-generated code, documentation, or suggestions that software developers could use with only minor edits or quick testing - essentially AI outputs that saved time without compromising accuracy or compliance.

This suggests that while nearly all organizations are seeing faster development and higher code quality (even at these early days of generative AI), the benefits to innovation and cross-functional collaboration are beginning to emerge, and the impact will depend on how widely AI tools are adopted within teams.

Opportunity	%
Productivity gains (time savings, faster scaffolding/docs)	100
Improved code quality (bug reduction, better tests)	82
Enhanced developer satisfaction & empowerment	73
Innovation enablement (rapid prototyping, demos)	64
Cross-functional collaboration (bridging business & tech)	55

Fig. 1 Reported opportunities of AI in software engineering show how many organizations benefit. Productivity and code quality gains were most common; collaboration and innovation appeared less frequently.

Some comments from experts included estimates such as:

- '. . . when roughly 70–80% of AI recommendations are acceptable, confidence in the tool increases significantly,' according to one interviewee.
- one participant noted that "..some 80% of the recommendations made by the generative AI tool are acceptable to you, then it increases the confidence."
- another expert said ". . . starting with 60–80% acceptable output as a benchmark for evaluating AI confidence".

A MedTech organization described noticeable speedups in drafting code and documentation while keeping strict review standards. The expert I spoke with said, *"If developers can't explain the AI output, we discard it and do it manually."*

An enterprise software team reported faster creation of demo environments and internal examples that previously took multiple days now took hours, shifting effort toward validation.

Consulting firms emphasized a faster ramp-up on unknown repositories because engineers can ask the tool to summarize structure and patterns before making changes.

Including both quotes and simple frequency counts helps present a balanced view showing not only what the expert participants said but also how often each point appeared in the interviews.

Skill	# of Orgs
Prompt engineering	9
Security/privacy oversight	7
System architecture & design	6
Cross-functional fluency	5
Cost/performance tuning	4
Data literacy	3

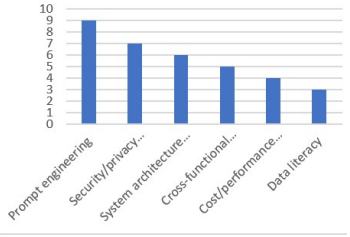

Fig. 2 Emerging skills identified by organizations. Prompt engineering was cited most often, followed by security oversight and system design, highlighting new priorities in software developer training.

While prompt engineering and tool orchestration were identified as essential by most organizations, security/privacy oversight and system architecture are also key areas where engineers' roles are expanding.

Across the 11 organizations involved in the survey, these views were very consistent. Nine (9) mentioned faster documentation or setup time, eight (8) noted improved code quality, and seven (7) pointed to better collaboration between the technical and non-technical teams. These frequencies support the qualitative quotes presented above.

Collectively, these numbers indicate that most teams are prioritizing prompt engineering and security, while some have established data literacy and AI fluency practices, showing that upskilling is key for success.

RQ2. In what ways have roles and skills evolved?

The common response to this question was that the role of a developer is expanding from a person who writes code to a person who decides what the AI should do and verifies the result. Engineers called this shift empowering because they can spend more time on design and correctness, even when the AI writes the first draft.

Core skills mentioned across interviews were *prompt engineering, tool orchestration* (choosing the right model or tool for a task), *security/privacy oversight, cost awareness* and *system architecture.*

As one engineer put it: "The role is less about writing code and more about deciding what the AI should do, then checking it." The independent software consultant compared the shift to craftsmanship: *"Like a master woodworker with better tools, AI lets us do higher-level work faster."*

Another expert compared the shift to moving from hand-sawing every board to running a workshop: the craft does not disappear, but it changes. Developers must still know fundamentals to evaluate AI output, but now also need data literacy (including multimodal inputs) and an ability to translate product needs into prompts, tests, and guardrails.

RQ3. What should be automated and what should stay manual?

Organizations consistently segment tasks by risk: low-risk work is automated, moderate-risk work uses AI with human review, and high-risk work remains manual. This pattern was reported across industries.

Category	# of Orgs
Low-risk (fully automated)	5
Moderate-risk (AI + human review)	8
High-risk (manual only)	7

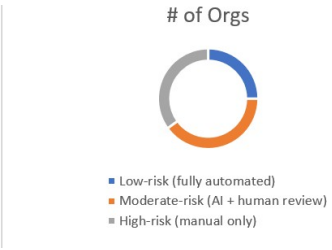

Fig. 3 Risk based automation frameworks. Organizations automate low-risk tasks, use human review for medium-risk work, and keep high-risk processes manual for compliance and governance purposes.

Similar three-tier automation patterns appear in empirical studies of AI-assisted software development, where low-risk repetitive tasks are delegated to automation, moderate-risk activities require AI-human collaboration, and high-risk processes remain manually verified.

Industry fieldwork confirms that maintaining a human-in-the-loop at key decision points improves trust and reduces security regressions[20].

These frameworks mirror the governance models proposed in large-scale reviews of generative-AI workflows, which recommend combining automation with ethical and compliance validation to prevent hallucinations or biased recommendations.

Collectively, these results align with the interviews in this study, illustrating that risk-tiered oversight is becoming a standard practice across sectors adopting AI in software engineering. Two brief quotes capture this mindset:

1. One expert stressed, *"We aim for complete accuracy, especially in high-risk environments."*

2. Another expert highlighted, *"In manufacturing, even small errors can lead to major financial or safety consequences, so 100% accuracy is essential."*

Reported challenges include training needs, security/privacy concerns, and reliability issues. Skill gaps were the most widely cited barrier, reflecting the need for continual upskilling.

Challenge	# of Orgs
Skill gaps/training needs	8
Security/privacy concerns	7
Reliability & hallucinations	6
Regulatory / compliance limits	5
Integration with workflows	5
Cost & value trade-offs	4

Metric Type	# of Orgs
Time savings	11
Code quality (bugs/tests)	9
Developer satisfaction	8
Performance reviews	6
Integration success	4

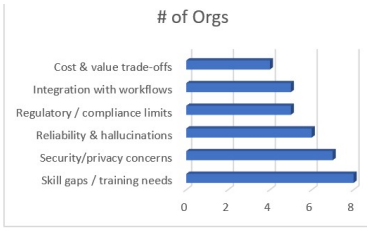

Fig. 4 Challenges of AI adoption and metrics used to measure impact. Key challenges include training gaps and privacy issues, while metrics focus on time savings, quality, and developer satisfaction.

The key challenge outlined in the chart 4 was characterized by an independent consultant who works with several different industries as, *"If you're not upskilling every six months, you're smoked."*

Not every participant viewed AI positively. For example, one of the experts, who is an engineer, cautioned,*"The models can sound confident but still be wrong; you can't just copy and paste what they generate."*Another noted, *"If we rely on AI too much, new developers might skip learning the basics."*Such comments reflect a shared concern that, while AI can speed up work, overreliance or lack of transparency, explainability could reduce long-term skill growth and accountability.

As per the compliance needs for a regulated environment, the study found a 3-tier model in practice:

- no AI for high-risk tasks where mistakes are unacceptable,
- AI with mandatory human review for most development activities, and
- full automation for routine, well-bounded tasks.

Other organizations used confidence thresholds for automation (for example, accepting AI output when confidence is above a set level at their org) and value-based formulas to prioritize where AI brings the biggest net benefit.

Time savings and code quality were the most common metrics. Developer satisfaction and performance reviews complement these quantitative measures, giving teams a holistic view of AI's impact.

Overall, this three-tier pattern, full automation for routine tasks, human review for medium-risk work, and manual control for high-risk cases, appeared consistently across all sectors interviewed, implying a cautious but confident adoption trend.

RQ4. How are engineers trained to work with AI?

Most organizations mix formal and informal learning. Formal programs include publishing an enterprise wide responsible-use guidelines, ethics, and prompt engineering, as well as model-specific certifications or bootcamps, etc. Some informal approaches include 'AI Champion' meetups, prompt cookbooks, hands-on demos, and hackathons.

Self-learning via online courses fills in gaps which was captured in the comment: "We teach engineers to practice every day with these tools. Otherwise, they don't stick."

Training is not one-and-done. Experts advise regular refreshers because models, tools, and best practices change quickly. In highly regulated settings, onboarding also covers privacy, security, and governance checklists so engineers understand where AI must never be used without review.

RQ5. What metrics are used to evaluate impact?

Teams measure a mix of productivity, quality, security, and human factors. In the context of these interviews, "productivity gains" did not mean writing more lines of code, but rather completing tasks faster, closing more tickets within a time window, or cutting the time needed to test and debug.

These qualitative patterns are consistent with recent quantitative evaluations showing that AI integration improves throughput and defect detection but only when metrics incorporate human review and governance factors. Controlled experiments and field data reveal measurable gains in time-to-delivery and code quality when productivity is tracked through sprint velocity, test coverage, and pull-request throughput.

Empirical research further suggests that adding developer-satisfaction surveys and peer-review quality scores provides a more holistic measure of AI impact.

These findings reinforce that productivity metrics must balance efficiency with quality and human oversight to capture AI's real contribution to engineering performance.

Productivity measures include cycle/lead time, story-point throughput, commit activity, and tickets closed. Quality and security indicators include defect counts in releases, static-analysis warnings, vulnerability and secret-scan results, and test coverage.

Several teams also watch developer satisfaction to understand whether the tools are making work better. Here are some responses from interviews to this question:

- One participant noted that test coverage improved from roughly 60% to nearly 90% after introducing AI tools
- Another participant said that he was able to test only 60%
- Another participant said that with the AI tools, they are covering 90%

These metrics fit easily into existing dashboards, which made adoption smoother. One organization noted that adoption itself can be measured with prompt frequency and acceptance rates of AI-generated suggestions but emphasized that hard outcomes (like time-to-change or bugs escaped) matter most.

To complement the surveys' findings presented above, I have included a table in the appendix (Table A1) that provides a comparative view of how different organizations reported AI's impact on daily work, emerging skills, automation boundaries, and metrics.

Discussion

The main result of this study is that AI is augmenting, not replacing, the work of software engineers across many industries. Recent empirical evidence reinforces that generative-AI tools accelerate development while still demanding rigorous governance to prevent quality or security regressions [21,22].

Systematic reviews on AI governance emphasize that human oversight, through validation steps, audit trails, and ethical review, remains essential to mitigating bias and ensuring compliance in large-scale software pipelines [23,24].

Together, these findings validate that augmentation occurs within boundaries shaped by ethical responsibility and transparent human control. These observations are consistent with McKinsey's (2024) industry analyses, which similarly concluded that productivity gains from AI arise primarily through better collaboration and automation of repetitive work rather than workforce reduction.

Although experts interviewed in this study were generally optimistic, several also cautioned that AI systems are not flawless. Similar concerns appear in recent experiments revealing how code assistants can generate insecure or biased outputs that propagate through production systems [25].

Other field studies describe the phenomenon of "over-automation," where developers become overly dependent on AI suggestions, leading to a decline in fundamental coding skills and problem-solving capacity [26].

These risks underline why formal training, secure development practices, and responsible-AI governance must accompany technical adoption. Together, these findings suggest a responsible adoption pattern that is already becoming a normal practice in the companies that participated in the survey.

This industry-level pattern aligns with broader adoption trends reported by the Stanford HAI Index Report 2025 [27] which documents rising enterprise investment in AI alongside growing attention to governance, education, and workforce adaptation.

The quantitative summaries in the Results section (For example, percentages and frequency counts) are drawn from this coding approach, allowing qualitative insights to be supported by basic numerical trends. Another important theme is the shift in skills. Prompt engineering is a real, teachable skill, but it sits on top of fundamentals. In this paper, "prompt engineering" refers to crafting clear yet detailed inputs that guide AI tools toward accurate, consistent, and reliable outputs from the models.

To judge an AI-generated diff (short for "difference" in coding), a developer still needs to know algorithms, data structures, and system design. Interviewees also emphasized data literacy (including multimodal inputs) and tool orchestration-choosing the right model or service for a task and composing them safely.

Education should adapt; accordingly, for students, the message is not 'stop learning to code,' but 'learn to code and learn to supervise AI'. That includes writing good prompts, building tests up front, inspecting outputs carefully, and understanding legal/ethical constraints. The engineers I spoke with also stressed the value of writing clear documentation and using AI to draft it, then improving it by hand.

Finally, there is convergence across sectors. Even organizations with heavy regulation, like a MedTech company, are using AI extensively in low-risk areas and cautiously in medium-risk areas with human review. Consulting and cloud organizations reported similar patterns. This suggests these practices are likely to spread further as tools improve, not fade away.

Because the sample size was small and spanned a limited number of organizations, these findings should be viewed as generating a hypothesis rather than being conclusive. The insights highlight emerging patterns in professional practice; as such, much broader quantitative research will be needed to confirm the generalization of findings.

Implications for Education and Early Careers

For students preparing to enter computer science and related fields, the message from experts was clear: coding fundamentals remain essential, but new skills like prompt engineering, AI oversight, and ethical reasoning are now part of the job.

One interviewee said, "If you're not upskilling every six months, you're smoked." That sense of urgency reflects how fast AI tools evolve. High school and college students can benefit by learning both the technical foundations and practicing with AI tools daily, focusing on how to guide them and check their outputs responsibly.

This suggests that students who combine strong fundamen-

tals with AI oversight skills will be especially valuable in the job market over the next decade.

Based on these findings mentioned here, educators in schools and early-career programs can begin by introducing a balanced framework that combines computer-science fundamentals with responsible AI practice. A simple model could include:

1. **Core coding skills:** learning programming languages and algorithms through traditional problem-solving methods.
2. **AI literacy:** required practicing with tools like GitHub Copilot or ChatGPT to understand how prompts affect outcomes.
3. **Ethics and safety:** courses and training on data privacy, bias, and human oversight.
4. **Hands-on projects:** small group assignments where students build or debug AI-assisted software.

Incorporating these steps can create a solid foundation where students learn to code as well as monitor AI thereby aligning education with professional work in the future.

Limitations and Future Work

This study has limitations. The sample size is small (11 organizations, 15 people), and the conversations were about 30 minutes each. This limited number means that the results cannot be generalized to software engineers across all organization types. Future research should include a larger and more balanced sample with representation from junior and mid-level participants to capture a wider range of experiences.

Most of the people I spoke with were senior people at their organizations, which can tilt answers toward strategy rather than day-to-day tactics used by junior developers. Consequently, the results are exploratory and hypothesis-generating. Future work should add quantitative validation. For example, standardized productivity metrics (cycle time, pull-request (PR) throughput, defect rates, and regression issues to name a few), controlled A/B comparisons between AI-assisted workflows and established baselines, or longitudinal studies to verify these perceptions with data.

A clear limitation with this study is that most participants were senior leaders or experts, which may bias the findings toward organizational strategy and high-level perspectives rather than those of junior or mid-level developers who experience it daily. Additionally, all interviews were conducted in English, which may have excluded viewpoints from non-English-speaking professionals.

Also, many numbers discussed by interviewees were estimates rather than the result of randomized trials. Finally, the analysis focused on cumulative trends and did not attempt to compare organizations one-by-one or reveal their internal metrics.

Because the data reflect expert estimates and qualitative observations, experimental validation, such as randomized trials comparing AI-assisted and non-assisted workflows, would strengthen the evidence base and is recommended for future research.

Future work could include a larger survey with stratified sampling by role (junior, mid, senior), longitudinal tracking of a few teams over many months, and experiments that measure code quality and reliability with and without AI. Future research could triangulate small-scale interview data like mine with global adoption statistics reported by the Stanford AI Index findings referenced earlier in this report.

It would also be valuable to study how AI tools change onboarding for new developers and how education can prepare students to supervise AI responsibly.

Conclusion

Based on interviews across industries, AI is helping engineers do more of the work that matters and less of the work that does not. It speeds up routine steps like scaffolding and documentation, but it also demands focus in a human-in-the-loop approach for validation and security.

This follows a long historical pattern: earlier tools such as compilers, IDEs, and low-code platforms did not replace developers but expanded what they could achieve, and AI appears to be the next step in that trajectory.

Organizations are adopting clear, risk-based frameworks: automate low-risk tasks, use AI with review for medium-risk tasks, and keep humans in charge where mistakes are unacceptable. Training programs - formal, peer-led, and self-guided, etc., are already building these skills.

The big takeaway for students is that software engineering is not disappearing. It is evolving. The best path forward is to learn fundamentals, practice with AI tools every day, measure results with real metrics, and keep humans in the loop.

As one interviewee said, "Instead of worrying about losing my job, I spend more time learning how to guide the AI." That mindset, i.e., curious, careful, and optimistic, is what will shape the next generation of software engineering.

These results are mainly qualitative and drawn from expert interviews; they are hypothesis-generating rather than definitive.

Consequently, these conclusions apply primarily to the specific contexts represented in this study and should not be generalized to all software engineering organizations without further study data.

This study contributes early qualitative evidence that AI currently functions as an augmentative partner in software en-

gineering, expanding our human capability rather than replacing it.

Acknowledgments

I am grateful to all the industry experts who generously shared their time and perspectives. I also want to thank my teachers at St. John's, especially Mr. Gregory Blondin, for the review and feedback on the paper. I'm grateful to my parents for always being there, facilitating the interviews, and encouraging me throughout this project.

Any interpretations or errors that remain are entirely my own.

References

1 A. Ziegler, E. Kalliamvakou, X. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian. Measuring github copilot's impact on productivity.

2 S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. The impact of ai on developer productivity. Evidence from GitHub Copilot. arXiv.

3 K. El Haji, C. Brandt, and A. Zaidman. Using github copilot for test generation in python: An empirical study. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024). Association for Computing Machinery*, page 45–55, New York, NY, USA.

4 F. Tambon, A. Moradi-Dakhel, A. Nikanjam, F. Khomh, and M.C. Desmarais. Bugs in large language models generated code: An empirical study.

5 G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt. Lost at c: A user study on the security implications of large language model code assistants. In *Proceedings of the 32nd USENIX Conference on Security Symposium (SEC '23). USENIX Association*, volume Article 124, page 2205–2222, USA.

6 X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang. Large language models for software engineering: A systematic literature review.

7 D. O'Brien, S. Biswas, S.M. Imtiaz, R. Abdalkareem, E. Shihab, and H. Rajan. Are prompt engineering and todo comments friends or foes? an evaluation on github copilot. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024). Association for Computing Machinery*, volume Article 219, page 1–13, New York, NY, USA.

8 H. Kruse, T. Puhlfürß, and W.Maalej Can Developers Prompt? A controlled experiment for code documentation generation. arXiv.

9 L. Wang, Y. Zhou, H. Zhuang, Q. Li, D. Cui, Y. Zhao, and L. Wang. Unity is strength: Collaborative llm-based agents for code reviewer recommendation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE 2024). Association for Computing Machinery*, page 2235–2239, New York, NY, USA.

10 J. He, C. Treude, and D. Lo. Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead. arXiv.

11 L. Aguiar, M. Paixao, R. Carmo, E. Soares, A. Leal, M. Freitas, and E. Gama. Multi-language software development in the llm era: Insights from practitioners' conversations with chatgpt. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24*, page 489–495, New York, NY, USA. Association for Computing Machinery.

12 K. Koyanagi, D. Wang, K. Noguchi, M. Kondo, A. Serebrenik, Y. Kamei, and N. Ubayashi. Exploring the effect of multiple natural languages on code suggestion using github copilot. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024). Association for Computing Machinery*, page 481–486, New York, NY, USA.

13 M. Leotta, H.Z. Yousaf, F. Ricca, and B. García. Ai-generated test scripts for web e2e testing with chatgpt and copilot: A preliminary study. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024). Association for Computing Machinery*, page 339–344, New York, NY, USA.

14 A. Carleton, D. Falessi, H. Zhang, and X.Generative A.I. Xia. Redefining the future of software engineering.

15 C.A. Gonçalves and C.T. Gonçalves. Assessment on the effectiveness of github copilot as a code assistance tool: An empirical study. In *Progress in Artificial Intelligence (EPIA 2024). Lecture Notes in Computer Science*, volume 14969. Springer, Cham.

16 F. Cirett-Galán, R. Torres-Peralta, R. Navarro-Hernández, J.L. Ochoa-Hernández, S. Contreras-Rivera, L.A. Estrada-Ríos, and G. Machado-Encinas. Assessing the use of github copilot on students of engineering of information systems.

17 T. Bazzan, B. Olojo, P. Majda, T. Kelly, M. Yilmaz, G. Marks, and P.M. Clarke. Analysing the role of generative ai in software engineering – results from an mlr. systems, software and services process improvement (eurospi 2024.

18 McKinsey & Company (2024). Can Software Developer Productivity Really Be Measured?

19 Leonardo Banh, Florian Holldack, and Gero Strobel. Copiloting the future: How generative ai transforms software engineering.

20 R. Ulfsnes, N.B. Moe, V. Stray, and M. Skarpen. Transforming software development with generative ai: Empirical insights on collaboration and workflow.

21 N. Perry, M. Srivastava, D. Kumar, and D. Boneh. Do users write more insecure code with ai assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23*, page 2785–2799, New York, NY, USA. Association for Computing Machinery.

22 N. Tihanyi, T. Bisztray, M.A. Ferrag, R. Jain, and L.C. Cordeiro. How secure is ai-generated code: A large-scale comparison of large language models. arXiv.

23 K. Stutz, K. Sandkuhl, and M. Möhring. Empirical insights into the usage of generative ai in software engineering. In *Perspectives in Business Informatics Research (BIR 2025). Lecture Notes in Business Information Processing*, volume 562. Springer, Cham.

24 F. Saoiabi, N. Kharmoum, C. Elasri, S.N. Lagmiri, and S. Ziti. Generative ai in software engineering: Enhancing development and innovation. In *Smart Business and Technologies (ICASBT 2024). Lecture Notes in Networks and Systems*, volume 1330. Springer, Cham.

25 A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M.C. Desmarais, and Z.M. Jiang. Github copilot ai pair programmer: Asset or liability?

26 P. Denny, J. Prather, B.A. Becker, J. Finnie-Ansley, A. Hellas, J. Leinonen, A. Luxton-Reilly, B.N. Reeves, E.A. Santos, and S. Sarsa. Computing education in the era of generative ai.

27 H.A.I. Stanford. Ai index report 2025.

Appendix

Questions asked in the interview study:

1. How has AI changed day-to-day engineering work?
2. In what ways have roles and required skills evolved with AI?
3. How do you decide what to automate with AI and what to keep manual?
4. How are software engineers trained to work alongside AI?
5. What metrics or indicators do you use to evaluate AI's impact?

The following table provides a comparative summary of how different organizations reported AI's impact on software engineering. It highlights patterns across daily work, emerging skills, automation boundaries, and evaluation metrics to complement the findings discussed in the Results section.

Table A1

Org.	Work Impact	Skills	Automation	Metrics
1	Faster dev cycles, consistency, better docs	Prompting, AI orchestration, AI ops roles	Automates low-risk tasks; humans for security/compliance	Commits, bug rates, cycle time, test coverage
2	Automates boilerplate, reduces repetitive work	UX, system design, domain expertise	AI for setup/drafts; humans for design & UX	No formal metrics
3	25–30% faster delivery, more review-focused	Prompt Eng., MLOps, strong code review	Risk-based 3-tier: low auto, medium AI review, high manual	Bugs, features delivered, ROI, security
4	Automates debugging, docs, and design ideas	AI tools, model training, data modeling	Engineer-driven; AI excluded from life-critical uses	5-Dimensions (quality, privacy, standards, accessibility, security)
5	Faster coding, legacy code translation, and support	CS fundamentals + AI tool fluency	Repetitive tasks automated; humans for production/architecture	Time saved, bug resolution speed, and deployments
6	Faster prototyping, work shifted to higher-value tasks	Fundamentals + AI fluency	Value-driven: start with code gen & testing	Qualitative: peer feedback, 360 reviews
7	15–25% faster setup, 45% productivity gain	Prompt Eng. core, focus on business impact	Human-in-the-loop; value equation (capability ÷ cost)	Prompt usage, time-to-market, performance reviews
8	Faster demos (days → hours), config automation	Multi-LLM fluency, QA, compliance	Automated: routine; Manual: sensitive/privacy-critical	Demos/week, deadlines, bug reduction
9	Knowledge access (legacy code), IDE assistance	Prompting, performance/cost/security awareness	Automates high-ROI tasks; humans for complex integrations	Qualitative: faster iteration, planning efficiency

Table A1 continued

Org.	Work Impact	Skills	Automation	Metrics
10	Faster coding, debugging, and higher expectations	Prompt Eng., AI/ML fundamentals	Automates customer queries; business logic manual	Speed, commits, faster bug resolution
11	3–4x productivity for skilled devs, faster troubleshooting	Cross-functional fluency, multimodal AI, continuous upskilling	AI for scale tasks (docs, synthesis); humans for interpretation	Adoption levels (GitHub), docs quality, throughput

This table consolidates individual responses from each organization into one comparative summary, reinforcing the main patterns identified in the Results section above and showing how similar themes appeared across different interviews.