Balancing Intelligence and Integrity: Structuring the Integration of AI Tools in Software Engineering

Dr. Mansoor Abdulhak, Hima Deepika Mannam

Department of Computer Science

***Abstract*—As AI tools like ChatGPT and GitHub Copilot become increasingly common in programming contexts, concerns arise regarding their influence on code quality, problem-solving skills and academic integrity. This study investigates how experts perceive and evaluate AI-generated code through a structured assessment framework. By conducting an expert review experiment, we examine the strengths, weaknesses and risks associated with AI-assisted programming. Our findings reveal that while AI tools can accelerate code generation, critical issues in logic, design and originality persist, emphasizing the need for thoughtful integration of AI into software engineering education and evaluation.**

I. Introduction

A. Overview of AI in Education

As generative AI tools like ChatGPT and GitHub Copilot become increasingly common in classrooms, their impact on how students learn programming is hard to ignore. These tools offer quick, code-generating solutions that can be helpful during problem-solving but they also introduce risks. Students may become overly dependent on AI-generated output, skip important debugging steps or lose sight of foundational programming logic. In software engineering education where testing, reasoning and debugging are essential skills, this shift raises important questions about how we teach and assess code literacy in the age of AI. Since its public release, ChatGPT [1] has rapidly become a widely used tool in educational and professional contexts, offering natural language interactions that can generate, debug and explain code. However, early surveys have revealed a mixed academic impact. While some instructors see AI as enhancing student learning opportunities, others report significant challenges to academic integrity, as highlighted by recent findings from Wiley Newsroom [2].

Recent surveys, such as those by Enago Academy [3], note that generative AI's rapid entry into educational environments has outpaced many institution's ability to adapt their curriculam and integrity guidelines. Similarly, broader concerns about AI permanently reshaping the education field are raised by Gold Penguin [4], emphasizing that this disruption goes beyond coding skills to impact foundational educational models.

This study explores those questions through a practical experiment focused on how learners engage with flawed AI-generated code. By providing participants with intentionally incorrect Python code and asking them to analyze, test or fix it, we observed their real-time thinking patterns do they trust the AI output blindly, or do they verify it by analyzing it? Our approach centers around four instructional strategies that scaffold responsible AI use: traditional, analysis, evaluation, and reconstruction. These structures aim to encourage students to think beyond code generation and focus on understanding, testing and reflecting on their work.

Through a mixed-method survey, we gathered insights from participants on their confidence in AI, testing habits and correction strategies. Their responses highlight a growing divide between learners who treat AI as a helpful partner and those who treat it as an unquestionable authority. By examining these behaviors closely, our goal is to offer actionable pedagogical guidance: how to integrate AI into software engineering courses in ways that strengthen, rather than weaken core programming skills.

B. Purpose and Significance of the Study

This study investigates how AI tools can be effectively integrated into software engineering curriculam without compromising student learning or academic honesty. The goal is to propose and validate structures that promote responsible AI use structures that reinforce testing, reflection and debugging while ensuring students continue to develop independent programming competencies.

II. Background: AI in Software Engineering

The integration of artificial intelligence into software engineering has accelerated rapidly over the past decade. Early applications of AI in development environments focused on code suggestion, syntax correction and automated testing. However, with the rise of large language models (LLMs) such as OpenAI's GPT series, AI systems are now capable of generating entire functions, classes and even application architectures.

Tools like GitHub Copilot, Amazon CodeWhisperer, and various domain-specific copilots are increasingly embedded into Integrated Development Environments (IDEs), fundamentally altering the workflow of software engineers. Developers are no longer limited to manual coding, they interact dynamically with AI suggestions that can adapt based on partial inputs or project context.

While these innovations promise increased efficiency and broader accessibility to coding, they also introduce risks. Overreliance on AI outputs without critical review can propagate logic errors, security vulnerabilities and brittle codebases. Consequently, modern software engineering education and

professional practice must adapt not only by integrating AI literacy but also by reinforcing traditional skills like debugging, validation, and systems thinking.

Understanding how experienced professionals interact with AI-generated code offers crucial insights into designing future educational curriculam and professional training programs that embrace AI responsibly while safeguarding the core principles of software quality and integrity.

III. Previous Work

The integration of generative AI into educational and professional contexts has gained momentum across various disciplines, particularly in software engineering and design thinking. Several recent studies and articles highlight both the transformative potential and the critical limitations of AI assistance in learning environments.

Kaplan (2023) emphasizes that artificial intelligence can augment human-centered processes like design thinking by accelerating ideation, prototyping, and user testing, leading to better and faster innovation outcomes [5]. Similarly, Satsoc (2023) proposes a framework for integrating AI into complex problem-solving workflows, demonstrating how AI can reduce cognitive load while enabling more iterative experimentation [6]. Muz.li (2022) adds to this by exploring collaborative UX design environments, noting that AI can provide real-time feedback and error detection, fostering creative synergy among designers and learners [7].

In the context of software engineering education, the challenges surrounding generative AI tools are more nuanced. The study "A Hitchhiker's Guide to Jailbreaking ChatGPT" reveals that despite built-in safety protocols, AI models can be manipulated through adversarial prompts posing risks to educational integrity and misuse of generative capabilities [8]. On the other hand, the work by Abdellatif et al. introduces a transformer-based method to augment datasets for engineering chatbots, suggesting how AI can be meaningfully leveraged for educational tools when applied with discipline and oversight [9].

The FormAI dataset study provides another important perspective: researchers used GPT-3.5 to generate over 112,000 C programs and applied formal verification tools to identify vulnerabilities [10]. The finding that over half of the generated code contained security flaws underlines a major concern AI-generated code is not inherently reliable, and students must be taught to test and validate such code rather than trusting it blindly. Fan et al. (2024) further contribute to this conversation with their study on the impact of AI-assisted programming. Their results show that while AI tools increase student motivation and reduce anxiety, they can also create a false sense of competence when not paired with reflective or evaluative learning activities [11]. Finally, the broader survey "Status and Future Trends of AI in Software Engineering" outlines the evolving role of AI in development pipelines and emphasizes the urgent need for revised curricula that include AI literacy, ethical use policies, and structured learning models to preserve conceptual learning while embracing modern tools [12].

Collectively, these works reveal a common theme: AI has the potential to enhance learning, design, and engineering, but only when supported by pedagogical frameworks that promote critical evaluation, ethical awareness, and hands on problem solving.

IV. Impact of AI Assistance

A. Student Motivation and Programming Anxiety

The integration of AI tools such as ChatGPT and GitHub Copilot into software engineering coursework has had a noticeable influence on student confidence and anxiety levels. Many students expressed that having AI support reduced their stress during coding exercises. For example, ChatGPT's ability to explain errors in plain language and Copilot's auto-complete suggestions helped students get started faster and feel less intimidated by unfamiliar tasks. This initial confidence boost, while helpful, raised important concerns. Despite appearing more confident, many students showed a tendency to place unquestioning trust in AI-generated code. When asked to solve a problem, they often submitted the output provided by the AI with little or no testing, even when the code contained hidden logic errors. In interviews and post-task reflections, several students admitted that they assumed "the AI knows what it's doing," revealing a misplaced sense of certainty. This behavior reflects a potential risk to academic integrity and deep learning, as students may bypass the essential step of verifying their work something critical in both education and professional software development. These behavioral shifts reflect broader concerns documented by Carnegie Learning [13], which emphasizes that while AI can support personalized learning, it also increases the risk that students may submit AI outputs without engaging deeply with the underlying logic. Maintaining critical engagement, therefore, becomes a central challenge for preserving academic rigor in AI-enhanced environments.

The distinction between students who verified and understood AI-generated code versus those who blindly accepted it was clear. Those who paused to analyze the AI's logic or attempted to validate its output with basic test cases demonstrated greater clarity, both in their reasoning and in their ability to explain how the code worked. These students were more engaged, more accurate, and more prepared for follow-up problem-solving tasks. This contrast suggests that while AI can increase motivation and reduce anxiety, it may also foster complacency if not used in an environment that reinforces critical thinking.

B. Collaborative Learning and Individual Performance

AI tools also had an effect on how students collaborated and worked individually. In peer environments, many students naturally discussed what the AI produced questioning whether the approach made sense or comparing alternatives. These conversations often led to moments of discovery, especially when students identified flaws in the AI's suggestions or

came up with improved solutions on their own. In this way, AI acted as a catalyst for collaborative learning, sparking technical dialogue and mutual review. However, when left to complete tasks independently, a subset of students became overly dependent on AI-generated responses. These students often bypassed the reasoning process and submitted code without much editing or reflection. As a result, they struggled in follow up tasks where they could not use the AI. Their performance suggested a gap in internalized understanding, raising concerns about long-term retention of programming concepts. A real-world analogy can be seen in professional development workflows. Companies that allow junior developers to rely on AI suggestions without review often find that those developers struggle with debugging or explaining code in collaborative settings. In contrast, developers trained to treat AI as a starting point not a final solution tend to retain more knowledge and build stronger problem-solving skills over time. These patterns in the classroom emphasize that while AI can be a powerful learning accelerator, its benefits are only realized when students remain actively engaged. Without guidance, there is a real risk that AI may displace rather than develop core skills such as debugging, testing, and logical reasoning skills that are foundational to both academic success and professional software engineering. Recent institutional reports, such as those from Wiley [2], further underline that while AI can enhance learning experiences, it simultaneously risks undermining essential skill development if students are not carefully guided in its use.

V. Proposed Structures

To ensure the responsible and educationally beneficial use of AI in programming courses, we propose four instructional scaffolds that were central to this study. Each structure is designed to promote active thinking, accountability, and skill retention:

- **Traditional Structure:** Students are required to declare the use of AI tools and briefly explain what outputs were generated and why they used them. This structure encourages transparency and helps identify patterns of overreliance. By making students articulate when and why they used AI, it fosters greater self-awareness and accountability. Over time, students learn to recognize whether their use of AI is assisting or replacing critical engagement with the task.
- **Analyzing AI Structure:** Students critique the AI-generated code without directly modifying it. They identify logical inefficiencies, potential bugs or improvements. This approach cultivates code reading habits and analytical reasoning, skills that are often underdeveloped in novice programmers. By treating AI outputs as draft artifacts rather than definitive solutions, students sharpen their ability to evaluate and question code critically an essential practice in both academic and professional settings.
- **Evaluation Structure:** Students validate AI-generated code by creating thorough test cases, including edge cases and boundary conditions. This structure strengthens debugging skills and aligns closely with software testing industry practices. It trains students to anticipate failure scenarios and build resilience into their code. Furthermore, practicing structured evaluation prepares students for real-world software development workflows where validating functionality through rigorous testing is a core competency.
- **Reconstruction Structure:** Students must rewrite the logic or function themselves, referencing the AI-generated output only as a scaffold. They justify their decisions at each step. This approach reinforces conceptual understanding and discourages passive copying. The emphasis on Reconstruction Structure aligns with broader educational strategies advocated by Anthology [14], who argue that authentic assessments focusing on explanation, iteration and critical thought are more effective at maintaining integrity than detection based approaches. Requiring students to explain their reasoning, as Carnegie Learning [15] suggests, ensures that learners engage in cognitive processing rather than passive acceptance of AI outputs. Through reconstruction, students internalize problem-solving strategies, develop deeper algorithmic thinking and build confidence in crafting solutions independently.

VI. Methodology

To better understand how AI generated code is evaluated in professional contexts, we conducted a small scale experiment with eight industry experts in software engineering. Each participant had a minimum of three years of experience and was actively employed in roles such as software development, QA engineering and DevOps across industries including enterprise IT, finance and education technology.

Participants were provided with a short Python function generated by an AI tool. The function claimed to return the sum of even numbers from a list, but it contained a deliberate logical error: instead of checking for even numbers using `num % 2 == 0`, it incorrectly summed odd numbers using `num % 2 == 1`.

Experts were informed that the code was AI-generated and were asked to complete an evaluation through a structured Google Form. The form included both multiple-choice and open ended questions, covering their confidence in the AI output, whether they tested or modified the code, and their general perceptions of AI-assisted programming.

The goal of this setup was to capture how experienced developers instinctively approach AI generated content: whether they test it thoroughly, trust it immediately, or critique it logically. To elicit these insights, the survey included questions such as:

- *On a scale of 1 to 5, how confident were you that the AI-generated code was correct before testing it?*

– *Did you run the code without changes, modify it first or rewrite it entirely?*
– *How many test cases did you use to verify the correctness of the code?*
– *What specific steps did you take to check for logical correctness or hidden flaws?*
– *What are your thoughts on the use of AI tools like ChatGPT in professional or educational programming tasks?*

VII. Results and Discussion

A. Response Highlights

Confidence ratings among experts varied considerably. Half of the participants rated their initial confidence in the AI-generated code at a moderate level (3 out of 5), while two experts indicated very low confidence, citing skepticism of AI correctness without validation.

When asked about their interaction with the provided code, 50% of the experts ran the function without changes initially but quickly recognized inconsistencies upon testing. 37.5% edited the code after spotting the logical flaw, and 12.5% immediately rewrote the function without executing it first, citing distrust of AI-generated logic.

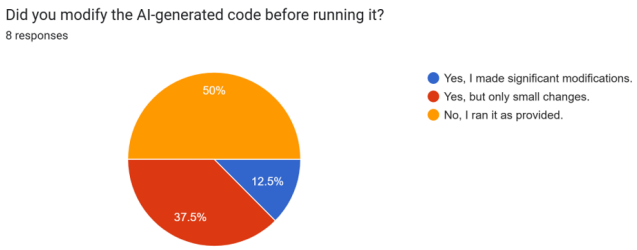

Fig. 1. Expert responses to whether they modified the AI-generated code before execution

As shown in Figure1, only 12.5% of participants made significant changes before running the code. In contrast, 50% ran the code exactly as provided, suggesting a baseline level of trust or at least curiosity toward the AI-generated solution.

Testing behavior was robust: all participants either wrote unit tests or used strategic manual test cases. Common inputs such as `[1,2,3,4]` and boundary cases like empty lists and large integers were employed.

Figure 2 illustrates that 50% of the participants did not write any test cases prior to execution. Meanwhile, 25% wrote one, and another 25% wrote two or more, revealing a spectrum of caution levels even among experienced professionals.

B. Verification Approaches

Open-ended responses revealed two main strategies among experts. Some compiled and executed the code first to observe outputs, then debugged based on anomalies. Others engaged in static analysis reading through the code logic carefully before running it.

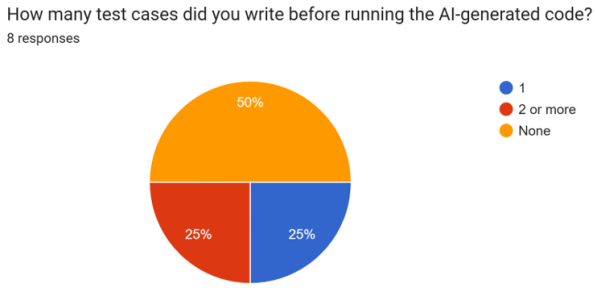

Fig. 2. Number of test cases written before running the AI-generated code

One participant noted:
"Even without execution, the modulo check felt suspicious. AI code tends to get edge conditions wrong, so I always scan for logical operators first."
Another shared:
"Initially trusted the structure, but after simple tests, it became clear something was off—too many odd sums creeping in."
These responses indicate that even skilled developers must maintain a healthy skepticism when working alongside AI outputs.

C. Implications

The expert evaluations reinforce the broader concern that AI-generated code, while often syntactically correct, may harbor subtle flaws that require human reasoning to detect. Even professionals needed to either test or logically critique the AI-generated output before trusting it.

This underscores the need to prepare future software engineering students to develop similar habits: testing, validation, and critical reflection must remain central even as AI tools grow more powerful.In professional practice, integrating AI critically treating its outputs as starting points rather than final products—emerges as a vital skill for maintaining code quality and ensuring software reliability.

D. Analysis of Expert Strategies

Notably, experts who preferred static code analysis over immediate execution cited concerns about "false confidence" the tendency of AI-generated code to produce correct outputs in limited cases while concealing subtle logical flaws. These participants emphasized the importance of verifying algorithmic correctness before trusting empirical behavior.

In contrast, experts who executed the code early tended to rely on debugging, introducing test cases to uncover issues through input output observations. This dynamic interaction mimics real-world practices in exploratory testing but may risk overlooking flaws that only appear under specific boundary conditions or rare events.

The way experts approached verification some relying on static code analysis, others on dynamic testing shows just how important it is to teach both careful code reading and thorough testing practices. These different strategies highlight that in an AI assisted programming environment, developers need a flexible mindset that combines multiple ways of checking and validating code. Encouraging this kind of multi modal thinking is key for both students learning to code and professionals maintaining software quality in real-world projects.

VIII. Limitations

While this study provides valuable insights into how industry experts evaluate AI-generated code, several limitations should be noted.

First, the participant sample was small consisting of only eight experts from a limited range of software engineering and quality assurance backgrounds. This restricts the generalizability of the findings across different industries, roles and levels of AI familiarity. Second, the code sample evaluated was intentionally simple a basic Python function to allow for controlled observation. However, real-world software projects involve far more complex layered architectures where AI-generated code flaws might manifest differently. Third, the study focused on a single interaction point. It did not capture longitudinal behaviors, such as how experts might adapt or change their trust levels with AI assistance over multiple projects or extended periods of exposure. Finally, the scope was limited to one AI-generated output from a single model. Other models, including domain-specific copilots or future iterations of generative AI, might produce different evaluation challenges.

IX. Future Work

Several opportunities exist for expanding on the insights gathered from this expert-focused study. First, future work could explore a larger, more diverse pool of participants across various industries such as finance, healthcare, cybersecurity and embedded systems to determine whether domain-specific knowledge influences how AI-generated code is evaluated.Second, future studies could track how professional developer's trust, critical thinking and verification habits change over time as they continue using AI tools—especially across longer projects that span several months. Third, future experiments could introduce more complex coding scenarios, such as integration tasks, multi-function modules or collaborative debugging exercises to better simulate real-world development pipelines influenced by AI suggestions. Finally, comparative studies examining different AI models including specialized copilots, security focused generators, and emerging prompt engineering techniques would help uncover how variations in AI output style and quality affect human critical evaluation and reliability assurance.

X. Conclusion

This study explored how generative AI tools like ChatGPT and GitHub Copilot can be effectively integrated into software engineering education without compromising academic integrity or the development of essential programming and problem-solving skills. While these tools offer valuable support in reducing coding anxiety and increasing productivity, they also pose risks when students rely on them uncritically. Through an experimental study grounded in pedagogical structures, we found that AI use becomes educationally beneficial only when paired with intentional instructional design. The proposed structures encouraged students to reflect, validate, and rebuild AI-generated code, fostering deeper understanding and accountability. Ultimately, responsible integration of AI in education does not mean removing the human from the loop which means teaching students to work alongside intelligent tools with critical awareness, technical discipline, and academic honesty.

XI. Acknowledgment

This research was conducted as part of a mentored graduate project under the guidance of Dr. Mansoor Abdulhak, in association with the CS 3440: Computer Science course at the University of Oklahoma. I would like to thank Dr. Mansoor Abdulhak for his mentorship, feedback and support throughout the project.

Special thanks are also extended to Joshua Kam, whose prior work and insights on AI-assisted code evaluation provided valuable inspiration and foundational context for this study.

References

[1] OpenAI, "Chatgpt: Optimizing language models for dialogue," 2022. [Online]. Available: https://openai.com/blog/chatgpt/

[2] W. Newsroom, "Ai has hurt academic integrity in college courses but can also enhance learning, say instructors and students," 2024.

[3] E. Academy, "The impact of chatgpt on academic integrity," 2023. [Online]. Available: https://www.enago.com/thesis-editing/blog/the-impact-of-chatgpt-on-academic-integrity

[4] G. Penguin, "How ai will permanently disrupt the education industry," 2023. [Online]. Available: https://goldpenguin.org/blog/how-ai-will-permanently-disrupt-the-education-industry/

[5] S. Kaplan, "Artificial intelligence in design thinking," 2023. [Online]. Available: https://www.sorenkaplan.com/artificial-intelligence-in-design-thinking/

[6] Satsoc, "Developing a comprehensive framework: Integrating ai into design digital thinking for complex problem-solving," 2023.

[7] Muz.li, "Collaborative ux: Integrating ai into design thinking," 2022. [Online]. Available: https://muz.li/blog/collaborative-ux-integrating-ai-into-design-thinking/

[8] Y. Liu, G. Deng, Z. Xu *et al.*, "A hitchhiker's guide to jailbreaking chatgpt via prompt engineering," *ArXiv*, 2024.

[9] A. Abdellatif, K. Badran, D. E. Costa, and E. Shihab, "A transformer-based approach for augmenting software engineering chatbots datasets," *2024 IEEE Transactions*, 2024.

[10] N. Tihanyi, T. Bisztray, R. Jain *et al.*, "The formai dataset: Generative ai in software security through the lens of formal verification," *2023 IEEE Secure Systems*, 2023.

[11] G. Fan, D. Liu, and R. Zhang, "The impact of ai-assisted programming on student motivation and performance," *Journal of Computer Science Education*, 2024.

[12] M. Vierhauser, I. Groher, and C. Sauerwein, "Status and future trends of ai in software engineering," *2023 AI and SE*, 2023.
[13] C. Learning, "Academic integrity and ai in education: Challenges and opportunities," 2023. [Online]. Available: https://www.carnegielearning.com/blog/academic-integrity-ai-in-education/
[14] Anthology, "Ai, academic integrity, and authentic assessment: An ethical path forward for education," 2023. [Online]. Available: https://www.anthology.com/resources/white-papers/ai-academic-integrity-and-authentic-assessment
[15] C. Learning, "Why students need to explain their reasoning," 2023. [Online]. Available: https://www.carnegielearning.com/blog/why-students-need-to-explain-their-reasoning/