Rethinking Programming Skills in the Age of Generative AI

Francesco Bolici
University of Cassino and Southern Lazio
f.bolici@unicas.it

Kevin Crowston
Syracuse University
crowston@syr.edu

Alberto Varone
University of Cassino and Southern Lazio
alberto.varone@unicas.it

Michael Fudge
Syracuse University
mafudge@syr.edu

Abstract

What does it mean to be skilled in a world where machines can now write computer code? We explore how generative AI is not only accelerating productivity, but reshaping the very meaning of programming expertise. Adopting a relational perspective, we focus on three interdependent skills that define effective human–AI collaboration: task framing, prompt design, and output interpretation.

Drawing on research in programming skills development and human–AI interaction, we trace the emergence of hybrid forms of competence that blend technical reasoning with contextual judgment, skills like strategic prompting, critical debugging, and situated problem framing. These signal a broader shift in programming: from producing code to coordinating AI-assisted problem solving, requiring new forms of cognitive effort and evaluative thinking.

As AI becomes an active collaborator, the focus is moving away from writing code line-by-line toward orchestrating adaptive systems. This transformation has deep implications for how technical skills are learned, applied, and socially valued in AI-mediated environments.

Keywords: Programming skills, skill development and retention, generative AI, human-AI collaboration

1. Introduction

The increased capability of modern artificial intelligence (AI) systems, generative AI in particular, has led to concerns about their impact (Crowston & Bolici, 2019; Dell'Acqua et al., 2023). We define AI as "an emergent family of technologies that build on machine learning, computation and statistical techniques, as well as rely on large data sets to generate responses, classifications, or dynamic predictions that resemble those of a knowledge worker" (Faraj et al., 2018, p. 62). In this paper, we address a long-standing concern about information technology use, namely its effects on user skills, programming skills in particular.

Rather than focusing solely on individual performance or tool capabilities, we propose a relational model of programming skill that emphasizes how developers interact with AI systems, tasks and outputs to shape their competence. The question of how AI tools affect programming skill gains importance in light of the widespread use of such tools in professional contexts. For instance, in October 2024 the CEO of Google and Alphabet, Sundar Pichai, stated that 25% of the code at Google is now written by AI (Pichai, 2024).

Public and academic debates often focus on AI's performance gains or potential for job displacement. We instead ask: how does AI reshape programming as a skillful activity, not only through what it automates, but through the ways it transforms learning, judgment and the division of labour? Based on a conceptual synthesis of research on programming skills development and human–AI interaction, we present a model that conceptualizes programming as a system of human-AI interactions rather than a solitary technical act. We explore its implications for how skills are acquired, applied and socially recognized. We examine these dynamics across experience levels, considering both beginners and more advanced practitioners.

2. Skills and AI tools

2.1. Skills

The concept of skill, central to our study, encompasses three key characteristics as proposed by Green (2011): productivity (value creation), expandability (enhancement through training) and social recognition. *Productivity* defines skill as the ability to autonomously perform tasks to create valued outcomes, with higher skill levels enabling more complex task performance. Following Wood (1986), we consider task complexity along three dimensions: component (number of distinct elements), coordinative (coordination requirements) and dynamic complexity (changing conditions over time). Increases in these dimensions raise the knowledge and skill needed for successful performance. *Expandability* is the potential to develop skills from basics to fluency. *Social recognition* acknowledges that societal forces influence

URI: https://hdl.handle.net/10125/112269
978-0-9981331-9-5

skill valuation and shape opportunities for skill development. We explore how AI's productivity gains may limit skill growth while redefining "skilled" work to include prompt design and oversight.

Our focus is on the skills involved in programming. Computer programming is a cognitively demanding activity that necessitates the development of multi-layered skills (Kiesler, 2024). Shneiderman (1976) established four fundamental programming skills: comprehension (understanding written code), composition (analyzing problems and writing solutions), debugging (identifying and correcting errors) and modification (adapting existing programs). Collins & White (1984) advanced Shneiderman's framework with empirical evidence supporting a hierarchical progression in programming skill development, noting that students could comprehend specific code snippets while struggling to generate comparably complex code themselves.

Research further suggests that failing to develop lower-order skills can undermine more advanced ones. Lopez et al. (2008) demonstrated that code reading skills are prerequisites for code writing, with basic programming construct knowledge forming the foundation of the hierarchy. Zavala & Mendoza (2017) identified significant correlations between code comprehension, completion and writing abilities, establishing five supporting skills: programming construct identification, verbal explanation, technical documentation comprehension and code refactoring proficiency. Debugging further emerged as a critical skill, with empirical work demonstrating that explicit debugging practice improves overall programming competence (Martinović & Rozić, 2025).

Though much research focuses on beginning programmers, Ryan et al. (2022) highlighted critical differences between educational and professional programming contexts. They identified five dimensions of complex programming environments typically underemphasized in education: working with pre-existing codebases, large-scale system architecture, extended development timeframes, multiple contributor involvement and established collaborative practices. Studies of real-world coding tasks (Xia et al., 2019) emphasize additional professional skills such as code integration, version control and modular design. Collaborative skills and domain-specific applications were also identified as crucial for programming effectiveness in shared programming environments (Sánchez et al., 2025). Bailey and Stefaniak's (2001) industry analysis described a comprehensive framework of 85 distinct skills across technical, soft and business categories, underscoring the complexity of professional programming practice.

2.2. AI for programming

Recent advancements in AI have significantly impacted programming. An example is Copilot, which can perform code completion given a comment or the start of a function. Chatbots such as ChatGPT can also write code in response to a request. Tools are adept at using different packages, reducing the need to learn application programming interfaces, an example of the potential tension between the productive and expandable dimension of programming skills.

2.2.1. Impact of AI use on programming performance (i.e., productive dimension of skills). Much of the public and organizational enthusiasm around generative AI in programming is driven by its perceived impact on productivity: faster code generation, reduced repetition and time savings across common tasks. However, productivity gains may reflect surface efficiency, not deep or transferable skill. In this section, we examine existing evidence on the performance effects of AI tools, while questioning whether enhanced output necessarily translates into robust or sustainable expertise.

AI programming tools have shown to support substantial productivity gains. For instance, GitHub Copilot users completed JavaScript tasks in less than half the time of those without AI, with similar success rates (Peng et al., 2023). Research has started to uncover factors affecting performance. Use of GitHub Copilot in enterprise contexts has shown time savings depending on the task, with up to 50% savings for documentation and autocompletion and 30-40% for repetitive coding, testing and debugging tasks (Pandey et al., 2024). Task complexity influences efficiency gains, with repetitive tasks seeing approximately 60% time reduction compared to 26% for complex multi-file tasks (Pandey et al., 2024). Becker et al. (2025) found that AI tools actually slowed down highly-experienced developers working on complex codebases, though possibly reducing effort.

Studies also reveal tradeoffs. For instance, in open-source environments, use of GitHub Copilot was found to increase merged pull requests by 6.5% and individual productivity by 5.5%, but integration time increased by 41.6%, suggesting higher coordination costs (Song et al., 2023). Vaillant et al. (2024) surveyed 207 software developers, finding that 73% reported increased productivity with ChatGPT and 70% reported reduced coding time, but 21% noted increased time spent on code review. A comparative analysis of tool interfaces found that AI tools enabled programmers to implement approximately 65% more requirements than traditional coding methods, but different interfaces produce distinct usage patterns:

auto-complete (e.g., Copilot) was used to generate more frequent but smaller code snippets and conversational interfaces (e.g., ChatGPT), fewer but larger code blocks (Weber et al., 2024).

The effectiveness of AI assistance extends to both advanced and beginner users. Programmers using ChatGPT-3 for HTML development completed tasks approximately 30% faster than without AI assistance. Interestingly, non-programmers utilizing ChatGPT achieved comparable completion times to professional programmers (Campero et al., 2022). Shan et al. (2025) found that professional programmers using a generative AI tool increased productivity by 18% on average, as measured by user stories completed, with a stronger effect for more experienced programmers, suggesting the importance of complementary skills to be able to make use of the outputs.

One problem is that, as helpful as AI tools seem to be, AI-generated code is not always correct. AI bug fixing competes with deep learning and outperforms traditional methods (Sobania et al., 2023), but its code often lacks quality compared to human solutions. Significant deficiencies have been found in documentation and exception management in AI-generated Java and Python code (Almanasra & Suwais, 2025). AI systems demonstrate concerning rates of hallucinations, presenting plausible but incorrect explanations even when provided with explicit error logs (Mora, 2025). These limitations reflect broader struggles with nuanced engineering concepts despite strong performance on structured problems (Hope et al., 2025).

As well, despite technology advancement, fundamental reasoning limitations persist across tools (Jain et al., 2025; Mora, 2025). As a result, AI programming tools struggle with complex coding tasks involving interdependencies among code components and different files. Approaches have been suggested to address these limitations, e.g., prompting strategies that break complex tasks into manageable components. These findings highlight that programming with AI is not a simple replacement of human effort, but rather requires careful output assessment and request formulation.

2.2.2 Impact of AI use on skill development (i.e., the expandable dimension of skills). A central concern remains: does increased productivity come at the cost of "learning less," as users engage less with the problem space and so bypass the effortful processes that drive long-term skill development? When code is produced faster but understood less, programmers may achieve more in the short term while building less in terms of long-term capability. Thus, there are important questions about how AI tools shape not just performance, but the process of skill formation.

Skill Acquisition Theory states that skills typically develop by proceduralizing and automatizing knowledge through practice (DeKeyser, 2015; Dreyfus, 2004). Research suggests that AI programming tools may undermine this process by removing both the need to exercise skills and the motivation to develop them. For instance, in one study, undergraduate students identified several advantages of using ChatGPT for programming learning, including quick responses, time savings, debugging assistance and support for thinking skills (Yilmaz & Karaoglan Yilmaz, 2023a). However, limitations include encouraging laziness, causing occupational anxiety and providing incorrect information. Recent research has extended this model to incorporate the effects of AI assistance, with several studies examining how these tools influence transitions between expertise levels (Nguyen, 2025; Pulk & Koris, 2025), including the development of hybrid intelligence frameworks that distribute cognitive tasks across human and AI components.

A significant concern is overreliance on AI outputs. An experimental study with professional writers demonstrated that while ChatGPT users saved time and improved output quality, they often used AI-generated content with minimal editing (Noy & Zhang, 2023), requesting complete solutions rather than using AI to enhance understanding (Rahe & Maalej, 2025; Vanacore et al., 2025). Studies have contrasted two usage strategies: seeking knowledge about general concepts to support a task vs. directly generating solutions, with regular AI users more likely to request complete solutions (Rahe & Maalej, 2025). Used properly, tools could potentially act as tutors, enhancing learning, but the predominant mode of interaction focuses on providing results rather than offering structured support for the learning process. Over-reliance on AI programming tools may also generate a sense of complacency and illusion of competence (Perry et al., 2023), with beginning programmers particularly prone to these effects. Over-reliance may also pose risks if developers do not feel responsible for the code they produce, raising questions of accountability for the outcomes.

Thus traditional models of skill acquisition, grounded in theories of proceduralization and gradual abstraction, are disrupted by AI systems that can instantly generate, refactor or even evaluate code. As mindlessness prevents users from properly attending to the work and learning from it, one author went as far as to suggest that systems should be deliberately imperfect to require people to consider outputs carefully (Dell'Acqua et al., 2023).

In summary, the acceleration of productive output enabled by AI may come at the cost of long-term skill development, particularly when users bypass deliberate practice and fall into patterns of over-reliance on AI-generated solutions. Such efficiency gains may foster shallow engagement and a false sense of competence, particularly among novices (Macnamara et al., 2024; Prather et al., 2023; Rahe & Maalej, 2025). This tension, between doing and learning, suggests that AI-assisted productivity and skill development may not progress in parallel and may even diverge without intentional design.

2.2.3 New skills for effective AI tool use. As suggested above, effective use of AI tools requires the development of new skills, complementary to but distinct from traditional programming skills. First, *prompting*, the ability to translate structured intent into inputs that guide AI systems, has emerged as a second foundational skill in Human–AI programming. It addresses what (Hutchins et al., 1985) referred to as the "gulf of execution", making the system do what the user wants (Crowston & Bolici, 2025). Indeed, studies have shown that students' benefits from AI tool use largely depend on "prompt literacy", i.e., their ability to formulate effective inputs for AI (Yilmaz & Karaoglan Yilmaz, 2023b). Prompting strategies—conversational or structured—can optimize outputs, enhancing clarity and efficiency (Beurer-Kellner et al., 2023; Wang et al., 2024). More recently, prompting has expanded to include providing information resources to guide a model (Mei et al., 2025). However, crafting effective prompts is not intuitive, even though it may seem to be. As stronger metacognitive skills are generally better mastered by more expert users, beginners have been found to struggle in conveying their intent to AI tools, for instance, needing to overcome inappropriate mental models derived from human interaction (Zamfirescu-Pereira et al., 2023).

A critical yet under discussed skill in Human–AI programming is *task framing*: structuring problems in contextually meaningful and solvable ways. Framing involves identifying the essential features of a problem, decomposing its components and articulating goals in forms that can be delegated to AI tools. Framing precedes prompting: without a clear conceptual model of the task, users may formulate vague or misleading prompts, resulting in low-quality outputs (Jonassen, 1997, 2000). Unlike prompting, which focuses on how to communicate with the model, framing concerns what should be communicated and why. It requires metacognitive skills such as abstraction and decomposition, the ability of programmers to break down complex problems and translate them into instructions the AI can understand (Prather, Denny, et al., 2024). It requires reasoning not only about code but also about objectives, constraints and downstream consequences, what Bolici et al. (2025) describe as orchestration across interdependent tasks. Framing the problem correctly enables clearer prompting, better AI alignment and more effective Human–AI collaboration (Annapureddy et al., 2025; Prather, Reeves, et al., 2024; Nguyen, 2025).

A final key competence is *interpretation*: evaluating and iterating on what the system has done. This skill addresses Hutchins et al.'s (1985) "gulf of evaluation": requiring users to make sense of system outputs and assessing whether the AI output aligns with the user's expectations and the problem's goals. It involves identifying flaws, gaps, or misalignments in the generated code and deciding whether to revise the prompt, fix the output manually, or reframe the task entirely (Prather, Denny, et al., 2024; Wang et al., 2024). As AI-generated code often has bugs or fails to handle corner cases correctly (Jain et al., 2025) particularly in complex tasks (Bowen et al., 2024; Chen et al., 2025; Mora, 2025), output evaluation is a core competency for effective AI tool use (Annapureddy et al., 2025). These skills can interact, as evaluation of an output may suggest ways to improve the prompt rather than simply fixing the code. Effective prompt engineering can significantly enhance AI performance, particularly when initial outputs are incorrect or incomplete (Denny et al., 2023; Wang et al., 2024). Fixing AI output often requires iterative prompt refinement—similar to debugging (Prather, Denny, et al., 2024).

Together, these three interdependent skills—framing, prompting and interpreting—form the foundation of a new model of Human–AI programming that we elaborate in Section 3.2. Table 1 summarizes how each of the three skill areas translates into concrete user actions, common pitfalls and key skills in AI-assisted programming.

A critical complication is that code interpretation skills are typically thought to depend on code writing skills (Collins & White, 1984; Lopez et al., 2008), precisely what AI tools replace. Experienced programmers are better at output evaluation, likely due to their more developed domain knowledge and higher levels of skill (DeKeyser, 2015; Dreyfus, 2004). This advantage partly explains the observed stronger impact of AI tools on experienced programmers' productivity, as they know what to ask for and can better assess and fix generated code (Shan et al., 2025). In contrast, beginner programmers have often been found to lack the necessary skills to leverage AI tools effectively (Mailach et al., 2024). If beginners fail to develop fundamental skills due to AI over-reliance, they are likely to struggle to develop advanced skills needed for

effective AI use. Even if skills are developed, insufficient practice may prevent proceduralization, making them more susceptible to decay (Macnamara et al., 2024; Rinta-Kahila et al., 2023).

Table 1. Three core skill areas (framing, prompting, interpreting) emerging in Human–AI programming

Relational Anchor	Key Action	Common Pitfall	Key Skills
Framing	Define the problem and its constraints	Vague or overly broad task definitions	Task decomposition, goal definition, contextualization
Prompting	Translate intent into AI-understandable input	Underspecified or overly generic prompts	Instruction formulation, abstraction, iterative query design, context engineering
Interpreting	Assess AI output and decide next step	Blind trust or minimal scrutiny of flawed outputs	Output validation, critical judgment, uncertainty handling

As an example of these difficulties, a recent study analyzing students' interactions with GitHub Copilot identified two modes of solution generation: "shepherding", attempting to coerce desired code through techniques like typing suggestions, deleting code after accepting it or modifying accepted code versus "drifting", moving aimlessly between suggestions without clear direction. Many students expressed confusion about AI suggestions while relying on the tool for guidance when stuck (Prather et al., 2023). Previously identified metacognitive difficulties persisted even with AI assistance, but three new difficulties emerged specifically related to generative AI: "progression", being conceptually behind while having false confidence, "interruption", concentration difficulties due to frequent code suggestions and "mislead", being led down incorrect paths (Prather, Reeves, et al., 2024). These difficulties highlight a growing divide between students who struggle and those who succeed with AI tools. High-performing students can recognize unhelpful suggestions and use AI to accelerate work, while struggling students develop an "illusion of competence" while being hindered by the technology, accepting AI suggestions at higher rates. Indeed, students with higher self-efficacy tended to use AI less frequently and later in the problem-solving process, while those with lower self-efficacy or higher fear of failure used AI more frequently and earlier (Margulieux et al., 2024).

3. Programming skills of the future

The widespread integration of AI into the software development life cycle necessitates a rethinking of what constitutes programming skill. Past research has identified a clear set of skills needed for conventional programming. The use of AI tools suggests an alternative possibility, that prompting and output evaluation build on different skills than traditional coding. In particular, while code-reading is generally considered fundamental to other programming skills (Luxton-Reilly et al., 2018), studies show that it is a lower-order skill than writing, suggesting that one may be able to successfully read code without being able to produce it (Robins et al., 2003; Winslow, 1996).

It is thus conceivable that students could learn to evaluate and make effective use of an AI tool's outputs without being able to create them themselves. However, other computational skills, e.g., problem decomposition, are still critical to framing. This perspective suggests that AI tool use requires computational thinking but not necessarily coding, shifting how programming may be taught and practiced (Beurer-Kellner et al., 2023; Prather, Denny, et al., 2024) and what being skilled in programming even means. Programming no longer revolves solely around syntactic mastery or isolated problem-solving, but increasingly involves the ability to collaborate with intelligent agents, formulate effective prompts and critically evaluate machine generated outputs.

These emerging skills challenge existing pedagogical approaches in computer science education. Consequently, the very definition of programming skill is expanding to include competencies that are metacognitive (e.g., decomposition and abstraction), collaborative (e.g., iterative prompting and correction) and evaluative (e.g., validation of AI-generated solutions). While some of these skills have always been considered relevant to programming, they are sometimes viewed as advanced rather than foundational. These changes align with Green's (2011) view of skill as a socially-contextualized expandable capacity for value creation. As programming shifts from code authoring to outcome orchestration (Bolici et al., 2025), programmers must develop hybrid expertise that blends traditional technical fluency with new forms of communicative and supervisory literacy. Mitigating skill decay and the illusion of competence thus becomes an integral part of future-oriented skill design (Rinta-Kahila et al., 2023).

3.1. Future programming skills taxonomy: A relational perspective

To understand programming skills in the age of generative AI, we must move beyond the individual and adopt a relational lens. As Crowston & Bolici (2019) argue, AI reshapes work not by replacing humans, but by transforming how they relate to tools and tasks. In this light, three core relationships define the new programming skillset.

First, the interaction between the human and the LLM centers on prompting: turning intent into structured input. This ability requires abstraction, decomposition and an understanding of how AI interprets instructions and provided information resources, skills that bridge what Hutchins et al. (1985) call the "gulf of execution." Prompting becomes a form of cognitive translation, essential to making AI tools useful. Studies have shown that prompt literacy significantly affects users' ability to benefit from AI tools (Prather, Denny, et al., 2024; Yilmaz & Karaoglan Yilmaz, 2023b).

Second, the human-to-task relationship remains critical. Even when an AI tool writes code, humans must understand the problem, define constraints and ensure alignment with broader goals. This kind of expertise anchors the use of AI in meaningful contexts. Foundational coding capabilities still matter—but increasingly resemble literacy skills, necessary but insufficient for higher-level effectiveness. As noted by Shan et al. (2025), more experienced programmers benefit more from AI assistance because of their deeper understanding of both the problem space and code semantics.

Third, the human-to-output relationship involves interpreting, validating and iterating on AI-generated solutions. Navigating the "gulf of evaluation" demands judgment and domain knowledge, particularly where outputs may be misleading, flawed, or incomplete. AI tools are known to hallucinate outputs (Mora, 2025) or produce plausible but incorrect code (Jain et al., 2025), reinforcing the importance of interpretive and debugging skills (Annapureddy et al., 2025). If these interpretive skills are not actively developed, there is a risk that users may adopt AI outputs too readily, reinforcing shallow understanding and reinforcing mindless usage patterns.

These three relationships (prompting, expertise and interpretation) form a new skill triangle, as shown in Figure 1. Rather than following a linear progression, they interact in modular and adaptive ways. In this new landscape, programming skill is not displaced but redistributed from execution to orchestration, from syntax to strategy (Bolici et al., 2025).

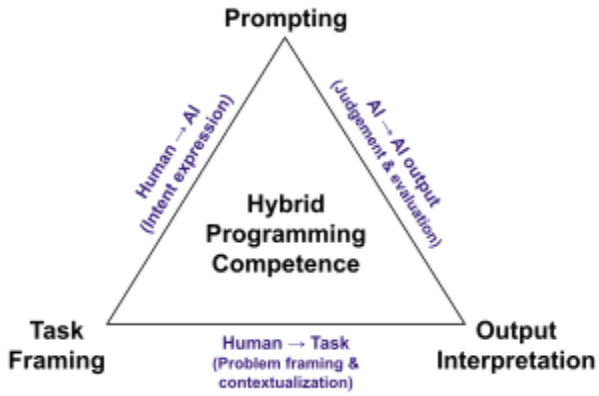

Figure 1. A relational perspective of programming competence in the age of Generative AI

3.2. Redefining what it means to be a "programmer"

As generative AI systems increasingly handle code generation, the role of the human programmer is being redefined. Instead of focusing primarily on writing code, programmers are now expected to engage in multi-dimensional coordination tasks: shaping prompts, framing problems and interpreting complex outputs in context. Programming becomes a practice of directing intelligent systems, grounded in both technical fluency and reflective judgment. Yet this shift also introduces new cognitive risks, such as false confidence and degraded procedural knowledge, particularly when learners rely heavily on AI without sufficient opportunities for explanation, iteration and correction (Macnamara et al., 2024; Prather, Reeves, et al., 2024). As shown in earlier sections (2.2.2–2.2.3), over-reliance on AI tools can hinder the development of procedural fluency, foster an illusion of competence and impair long-term retention. Programmers may appear effective while lacking a deep understanding of the code they orchestrate.

This transformation brings forward three emergent hybrid skill areas, where traditional competencies are recombined with new demands introduced by AI.

3.2.1. Meta-modeling and AI strategy: Blending systems thinking with the capacity to mentally model and direct AI behavior. This skill involves understanding how AI interprets input, structuring interactions over time and anticipating downstream effects. It is a strategic skill set that bridges prompt design, task decomposition and architectural foresight. These capabilities tend to be more accessible to experienced programmers, who benefit from more advanced mental models and greater comfort in

problem decomposition (Shan et al., 2025; Zamfirescu-Pereira et al., 2023).

3.2.2. Interpretive and critical debugging: Extending traditional debugging to include critical evaluation of AI-generated outputs—not only code, but also explanations, test suggestions, or incomplete rationales. This skill involves interpreting diverse formats, identifying subtle errors or misalignments and deciding whether to refine the prompt, rewrite the code, or reframe the task. While traditional debugging typically focuses on fixing syntax errors or logical flaws in one's own code, debugging in AI-assisted contexts must go further—combining interpretive analysis and critical reflection. This expanded form of debugging requires not only understanding the structure and logic of machine-generated code, but also questioning its validity, appropriateness, and possible bias. Rather than treating code as self-contained and correct-by-default, this approach encourages programmers to evaluate both what the code does and whether it should be trusted or used. Recent studies have shown that learners who lack this evaluative layer tend to follow flawed AI suggestions with minimal revision, reinforcing surface-level understanding (Prather, Reeves, et al., 2024; Rahe & Maalej, 2025).

3.2.3. Situated problem framing and responsibility: Combining domain knowledge, contextual awareness and ethical reasoning to define problems that matter and ensure that AI-generated solutions align with real-world needs. This skill emphasizes accountability, especially in complex or ambiguous environments, where impact matters more than correctness alone. As highlighted by Dell'Acqua et al. (2023), delegating reasoning to AI without critical reflection can lead to disengagement and the erosion of responsibility in socio-technical contexts.

These three new hybrid skills reflect a move from linear to adaptive, cross-functional expertise. The programmer is no longer just a builder of solutions, but a curator of intent, interpreter of meaning and steward of quality in an increasingly automated and mediated environment. Rather than replacing foundational programming skills, generative AI calls for their redistribution and augmentation toward a practice where human insight, contextual intelligence and reflective action are more central than ever.

4. Discussion

The model developed above has practical implications, specifically its implications for skill acquisition and maintenance efforts, in both formal and informal learning settings.

4.1. Implications for education and workforce development

First, the relational nature of programming in the age of AI demands a shift in both education and training. Curricula should evolve to include prompting, interpretation and contextual expertise as core learning outcomes, not peripheral skills. Teaching must go beyond syntax to include AI modeling, interaction refinement and output judgment. This shift calls for pedagogical models that embrace iteration, uncertainty and productive failure: conditions in which students learn how to guide AI systems rather than uncritically accept and adopt their outputs. Without guided reflective learning approaches, learners may become overly reliant on AI-generated solutions, experiencing short-term gains at the cost of deeper skill acquisition and long-term retention (Macnamara et al., 2024; Rahe & Maalej, 2025). Likewise, assessment strategies must move beyond output correctness to capture a learner's ability to steer and evaluate generative systems effectively and ethically.

In professional settings, reskilling efforts must support not only tool adoption but also the development of adaptive expertise—the capacity to apply judgment across diverse domains while maintaining responsibility over outcomes generated in collaboration with AI. Addressing the risks of skill decay and illusion of competence will require not only technical training, but also metacognitive awareness and deliberate reflection on when and how to rely on AI (Prather et al., 2023; Rinta-Kahila et al., 2023). This training includes helping learners recognize when to trust, revise or reject AI-generated outputs, all skills increasingly essential to work effectively.

4.2. Risk of skill polarization

Second, one of the most urgent challenges is the emergence of a skills divide between those who can effectively collaborate with AI and those who cannot. Research shows that more experienced programmers are better equipped to benefit from generative tools, thanks to stronger domain knowledge and interpretive skills. In contrast, novices may become over-reliant, accepting outputs without understanding, which risks stagnating their development. This over-reliance can foster a false sense of mastery, where users feel competent but lack the underlying procedural knowledge needed to improve or adapt independently (Macnamara et al., 2024; Prather, Reeves, et al., 2024).

Left unaddressed, this gap could foster stratified labor markets, where a narrow group of "AI-fluent" professionals oversees orchestration and validation (Bolici et al., 2025), while others are relegated to

lower-value tasks. To prevent this polarization, inclusive training and access to foundational skills must be prioritized alongside the teaching of new AI-centric capabilities. As Mailach et al. (2024) and Margulieux et al. (2024) show, students with lower self-efficacy or fear of failure are especially prone to early and excessive AI use—suggesting that psychological as well as technical factors shape this divide. Crucially, such interventions must not only broaden access, but actively counteract the passive use of AI by promoting reflective, critical engagement.

5. Conclusion

As generative AI transforms the nature of programming (and work more generally), traditional notions of skill are no longer sufficient. This paper reframes programming not as code production but as hybrid coordination, where expressing intent, interpreting outputs and contextualizing tasks become core competencies. Our relational perspective reveals that future expertise lies in orchestrating interactions with intelligent systems, not just mastering syntax. The central challenge is not automation, but meaning-making: ensuring that the human remains accountable, reflective and skilled in a world of powerful but fallible AI. In this landscape, programming is not disappearing, rather it is evolving.

This conceptual paper provides several opportunities for future research. A major need is to test the model proposed in this paper. Research can also extend from our focus on programming to software engineering more broadly and to other skills that can now benefit from technological support.

6. References

Almanasra, S., & Suwais, K. (2025). Analysis of ChatGPT-Generated Codes Across Multiple Programming Languages. *IEEE Access*, *13*, 23580–23596. https://doi.org/10.1109/ACCESS.2025.3538050

Annapureddy, R., Fornaroli, A., & Gatica-Perez, D. (2025). Generative AI Literacy: Twelve Defining Competencies. *Digit. Gov.: Res. Pract.*, *6*(1), 13:1-13:21. https://doi.org/10.1145/3685680

Bailey, J. L., & Stefaniak, G. (2001). Industry perceptions of the knowledge, skills, and abilities needed by computer programmers. *Proceedings of the 2001 ACM SIGCPR Conference on Computer Personnel Research*, 93–99. https://doi.org/10.1145/371209.371221

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). *Measuring the impact of early-2025 AI on experienced open-source developer productivity* (No. arXiv:2507.09089). arXiv. https://doi.org/10.48550/arXiv.2507.09089

Beurer-Kellner, L., Fischer, M., & Vechev, M. (2023). Prompting Is Programming: A Query Language for Large Language Models. *LMQL as Described in Prompting Is Programming: A Query Language for Large Language Models*, *7*(PLDI), 186:1946-186:1969. https://doi.org/10.1145/3591300

Bolici, F., Varone, A., & Diana, G. (2025). Beyond AI Automation: How Replace, Reinforce, Reveal, and Orchestrate Modes Align Task Interdependence and Organizational Design for Effective AI Implementation. *Proceedings of Theorizing Data & AI 2025*. Theorizing Data & AI 2025, Amsterdam, Netherlands.

Bowen, C., Sætre, R., & Miyao, Y. (2024). A Comprehensive Evaluation of Inductive Reasoning Capabilities and Problem Solving in Large Language Models. In Y. Graham & M. Purver (Eds.), *Findings of the Association for Computational Linguistics: EACL 2024* (pp. 323–339). Association for Computational Linguistics. https://aclanthology.org/2024.findings-eacl.22

Campero, A., Vaccaro, M., Song, J., Wen, H., Almaatouq, A., & Malone, T. W. (2022). *A test for evaluating performance in human-computer systems* (No. arXiv:2206.12390). arXiv. https://doi.org/10.48550/arXiv.2206.12390

Chen, J., Wei, Z., Ren, Z., Li, Z., & Zhang, J. (2025). *LR^{2}Bench: Evaluating Long-chain Reflective Reasoning Capabilities of Large Language Models via Constraint Satisfaction Problems* (No. arXiv:2502.17848). arXiv. https://doi.org/10.48550/arXiv.2502.17848

Cheng, K., Yang, J., Jiang, H., Wang, Z., Huang, B., Li, R., Li, S., Li, Z., Gao, Y., Li, X., Yin, B., & Sun, Y. (2024). *Inductive or Deductive? Rethinking the Fundamental Reasoning Abilities of LLMs* (No. arXiv:2408.00114). arXiv. https://doi.org/10.48550/arXiv.2408.00114

Collins, R. W., & White, K. B. (1984). An Investigation of Componential Skill Relationships in Programming. *AEDS Journal*, *18*(2), 123–129. https://doi.org/10.1080/00011037.1984.11008393

Crowston, K., & Bolici, F. (2019). Impacts of Machine Learning on Work. *Proceedings of the 52nd Hawaii International Conference on System Sciences*.

Crowston, K., & Bolici, F. (2025). Deskilling and upskilling with generative AI systems. *Proceedings of the iConference*. iConference, Indiana Bloomington, USA.

DeKeyser, R. (2015). Skill Acquisition Theory. In *Theories in Second Language Acquisition* (Second Edition). Routledge.

Dell'Acqua, F., McFowland, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., & Lakhani, K. R. (2023). Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4573321

Denny, P., Kumar, V., & Giacaman, N. (2023). Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 1136–1142.

https://doi.org/10.1145/3545945.3569823
Dreyfus, S. E. (2004). The Five-Stage Model of Adult Skill Acquisition. *Bulletin of Science, Technology & Society*, *24*(3), 177–181. https://doi.org/10.1177/0270467604264992
Faraj, S., Pachidi, S., & Sayegh, K. (2018). Working and organizing in the age of the learning algorithm. *Information and Organization*, *28*(1), 62–70. https://doi.org/10.1016/j.infoandorg.2018.02.005
Green, F. (2011). *What is Skill? An Inter-Disciplinary Synthesis* (Research Paper No. 20; LLAKES). Centre for Learning and Life Chances in Knowledge Economies and Societies, Institute of Education, University of London. https://www.llakes.org/wp-content/uploads/2011/02/Green-What-is-Skill-reduced.pdf
Hope, B., Bracey, J., Choukir, S., & Warner, D. (2025). *Assessment of ChatGPT for Engineering Statics Analysis* (No. arXiv:2502.00562). arXiv. https://doi.org/10.48550/arXiv.2502.00562
Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1985). Direct manipulation interfaces. *Human–Computer Interaction*, *1*(4), 311–338. https://doi.org/10.1207/s15327051hci0104_2
Jain, R., Thanvi, J., & Subasinghe, A. (2025). The evolution of ChatGPT for programming: A comparative study. *Engineering Research Express*, *7*(1), 015242. https://doi.org/10.1088/2631-8695/ada51d
Jonassen, D. H. (1997). Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational Technology Research and Development*, *45*(1), 65–94. https://doi.org/10.1007/BF02299613
Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development*, *48*(4), 63–85. https://doi.org/10.1007/BF02300500
Kiesler, N. (2024). *Modeling Programming Competency: A Qualitative Analysis*. Springer International Publishing. https://doi.org/10.1007/978-3-031-47148-3
Liu, E., Neubig, G., & Andreas, J. (2024). *An Incomplete Loop: Instruction Inference, Instruction Following, and In-context Learning in Language Models* (No. arXiv:2404.03028). arXiv. https://doi.org/10.48550/arXiv.2404.03028
Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the Fourth International Workshop on Computing Education Research*, 101–112. https://doi.org/10.1145/1404520.1404531
Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: A systematic literature review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 55–106. https://doi.org/10.1145/3293881.3295779
Macnamara, B. N., Berber, I., Çavuşoğlu, M. C., Krupinski, E. A., Nallapareddy, N., Nelson, N. E., Smith, P. J., Wilson-Delfosse, A. L., & Ray, S. (2024). Does using artificial intelligence assistance accelerate skill decay and hinder skill development without performers' awareness? *Cognitive Research: Principles and Implications*, *9*(1), 46. https://doi.org/10.1186/s41235-024-00572-8
Mailach, A., Gorgosch, D., Siegmund, N., & Siegmund, J. (2024). "Ok Pal, we have to code that now": Interaction patterns of programming beginners with a conversational chatbot. *Empirical Software Engineering*, *30*(1), 34. https://doi.org/10.1007/s10664-024-10561-6
Margulieux, L. E., Prather, J., Reeves, B. N., Becker, B. A., Cetin Uzun, G., Loksa, D., Leinonen, J., & Denny, P. (2024). Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 276–282. https://doi.org/10.1145/3649217.3653621
Martinović, B., & Rozić, R. (2025). Perceived Impact of AI-Based Tooling on Software Development Code Quality. *SN Computer Science*, *6*(1), 63. https://doi.org/10.1007/s42979-024-03608-4
Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z.-Z., Zhang, D., Zhou, C., Mao, J., Xia, T., Guo, J., & Liu, S. (2025). *A Survey of Context Engineering for Large Language Models* (No. arXiv:2507.13334). arXiv. https://doi.org/10.48550/arXiv.2507.13334
Mora, R. (2025). *Assessing ChatGPT's ability to detect and correct programming errors in stata do-files*. https://hdl.handle.net/10016/45949
Nguyen, A. (2025). Human-AI shared regulation for hybrid intelligence in learning and teaching: Conceptual domain, ontological foundations, propositions, and implications for research. *Proceedings of the 58th Hawaii International Conference on System Sciences*.
Noy, S., & Zhang, W. (2023). Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, *381*(6654), 187–192. https://doi.org/10.1126/science.adh2586
Pandey, R., Singh, P., Wei, R., & Shankar, S. (2024). *Transforming Software Development: Evaluating the Efficiency and Challenges of GitHub Copilot in Real-World Projects*. https://doi.org/10.48550/arXiv.2406.17910
Pichai, S. (2024, October 29). Q3 earnings call: CEO's remarks. *Google: The Keyword*. https://blog.google/inside-google/message-ceo/alphabet-earnings-q3-2024/
Prather, J., Denny, P., Leinonen, J., IV, D. H. S., Reeves, B. N., MacNeil, S., Becker, B. A., Luxton-Reilly, A., Amarouche, T., & Kimmel, B. (2024). *Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models* (No. arXiv:2401.10759). arXiv. https://doi.org/10.48550/arXiv.2401.10759
Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., & Santos, E. A. (2023). "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans.*

Comput.-Hum. Interact., *31*(1), 4:1-4:31. https://doi.org/10.1145/3617367

Prather, J., Reeves, B. N., Leinonen, J., MacNeil, S., Randrianasolo, A. S., Becker, B. A., Kimmel, B., Wright, J., & Briggs, B. (2024). The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, *1*, 469–486. https://doi.org/10.1145/3632620.3671116

Pulk, K., & Koris, R. (2025). *Generative AI in Higher Education: The Good, the Bad, and the Ugly*. Edward Elgar Publishing.

Rahe, C., & Maalej, W. (2025). *How Do Programming Students Use Generative AI?* (No. arXiv:2501.10091). arXiv. https://doi.org/10.48550/arXiv.2501.10091

Rinta-Kahila, T., Penttinen, E., Salovaara, A., Soliman, W., & Ruissalo, J. (2023). The vicious circles of skill erosion: A case study of cognitive automation. *Journal of the Association for Information Systems*, *24*(5), 1378–1412. https://doi.org/10.17705/1jais.00829

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Ryan, B., Soria, A. M., Dreef, K., & van der Hoek, A. (2022). Reading to write code: An experience report of a reverse engineering and modeling course. *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, 223–234. https://doi.org/10.1145/3510456.3514164

Sánchez, O. R., Salazar, A. B., & Bolaños, M. E. (2025). Collaborative programming as a didactic learning strategy in CS1 courses. *IEEE Revista Iberoamericana de Tecnologias Del Aprendizaje*, 1–1. IEEE Revista Iberoamericana de Tecnologias del Aprendizaje. https://doi.org/10.1109/RITA.2025.3541541

Shan, G., Rivera, M., Anand, P., & Kumar, S. (2025). How Does Generative AI Usage Affect the Coding Performance of Developers? *Proceedings of the 58th Hawaii International Conference on System Sciences*. https://hdl.handle.net/10125/109713

Shneiderman, B. (1976). Exploratory experiments in programmer behavior. *International Journal of Computer & Information Sciences*, *5*(2), 123–143. https://doi.org/10.1007/BF00975629

Song, F., Agarwal, A., & Wen, W. (2023). *The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot* (SSRN Scholarly Paper No. 4856935). Social Science Research Network. https://doi.org/10.2139/ssrn.4856935

Vaillant, T. S., Almeida, F. D. de, Neto, P. A. M. S., Gao, C., Bosch, J., & Almeida, E. S. de. (2024). *Developers' Perceptions on the Impact of ChatGPT in Software Development: A Survey* (No. arXiv:2405.12195). arXiv. https://doi.org/10.48550/arXiv.2405.12195

Vanacore, K., Pankiewicz, M., & Baker, R. (2025). *Unpacking the Impact of Generative AI Feedback: Divergent Effects on Student Performance and Self-Regulated Learning*. EdArXiv. https://doi.org/10.35542/osf.io/tbpn3_v1

Wang, T., Zhou, N., & Chen, Z. (2024). *Enhancing Computer Programming Education with LLMs: A Study on Effective Prompt Engineering for Python Code Generation* (No. arXiv:2407.05437). arXiv. https://doi.org/10.48550/arXiv.2407.05437

Weber, T., Brandmaier, M., Schmidt, A., & Mayer, S. (2024). Significant Productivity Gains through Programming with Large Language Models. *Proc. ACM Hum.-Comput. Interact.*, *8*(EICS), 256:1-256:29. https://doi.org/10.1145/3661145

Winslow, L. E. (1996). Programming pedagogy—A psychological overview. *ACM SIGCSE Bulletin*, *28*(3), 17–22. https://doi.org/10.1145/234867.234872

Wood, R. (1986). Task complexity: Definition of the construct. *Organizational Behavior and Human Decision Processes*, *37*, 60–82. https://doi.org/10.1016/0749-5978(86)90044-0

Wu, Z., Qiu, L., Ross, A., Akyürek, E., Chen, B., Wang, B., Kim, N., Andreas, J., & Kim, Y. (2024). Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks. In K. Duh, H. Gomez, & S. Bethard (Eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (pp. 1819–1862). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.naacl-long.102

Xia, X., Wan, Z., Kochhar, P. S., & Lo, D. (2019). How Practitioners Perceive Coding Proficiency. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 924–935. https://doi.org/10.1109/ICSE.2019.00098

Yilmaz, R., & Karaoglan Yilmaz, F. G. (2023a). Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. *Computers in Human Behavior: Artificial Humans*, *1*(2), 100005. https://doi.org/10.1016/j.chbah.2023.100005

Yilmaz, R., & Karaoglan Yilmaz, F. G. (2023b). The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, *4*. Scopus. https://doi.org/10.1016/j.caeai.2023.100147

Zamfirescu-Pereira, J. D., Wong, R. Y., Hartmann, B., & Yang, Q. (2023). Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 1–21. https://doi.org/10.1145/3544548.3581388

Zavala, L., & Mendoza, B. (2017). Precursor skills to writing code. *J. Comput. Sci. Coll.*, *32*(3), 149–156.