Pedagogical Innovations in Computer Science Education
https://doi.org/10.4018/979-8-3373-6546-6.ch009

Book Chapter

Assessment of Coding Skills and Programming Knowledge in the Age of Generative AI: Best Practices and Effective Strategies for Computer Science Education

Manuel B. Garcia [a] *, Ahmed Mohamed Fahmy Yousef [b], Ramazan Yılmaz [c], Ramesh C. Sharma [d], Thomas K. F. Chiu [e], Robertas Damaševičius [f]

[a] Educational Innovation and Technology Hub, FEU Institute of Technology, Manila, Philippines
[b] Educational Technology Department, Fayoum University, Egypt
[c] Faculty of Science, Bartın University, Bartın, Turkey
[d] School of Global Affairs, Dr. B.R. Ambedkar University Delhi, India
[e] Faculty of Education, Chinese University of Hong Kong, China
[f] Department of Applied Informatics, Vytautas Magnus University, Lithuania

*** Correspondence:**
Manuel B. Garcia, University of the Philippines Diliman and FEU Institute of Technology.
mbgarcia@feutech.edu.ph

How to cite this article:
Garcia, M. B., Yousef, A. M. F., Yilmaz, R., Chiu, T. K. F., Sharma, R. C., & Damaševičius, R. (2026). Assessment of Coding Skills and Programming Knowledge in the Age of Generative AI: Best Practices and Effective Strategies for Computer Science Education. In Pedagogical Innovations in Computer Science Education (pp. 275-304). IGI Global. https://doi.org/10.4018/979-8-3373-6546-6.ch009.

Article History:
Received: 8 May 2025
Revised: 22 Jan 2026
Accepted: 5 Feb 2026
Published: 26 Feb 2026

Abstract:

The rapid emergence of artificial intelligence (AI) and generative AI tools poses a significant threat to the validity of traditional programming assessments. As the boundary between authored and AI-generated code becomes increasingly indiscernible, long-standing assessment models centered on output correctness and code submission are at risk of obsolescence. Despite its urgency, prior work has largely concentrated on detection, with limited emphasis on reimagining assessment design. This chapter addresses that gap by proposing strategies for assessing programming proficiencies in an AI-mediated context. Its objective is to help educators move beyond surveillance-based models and adopt approaches that emphasize uniquely human cognitive capacities. These pedagogical strategies advance the field by shifting the discourse from reactive prevention to proactive, pedagogically aligned assessment design. In doing so, the chapter affirms that the future of programming assessment is not about resisting AI but about designing systems that assess human thinking over code output.

Keywords:

Artificial Intelligence, AI-Assisted Coding, Assessment Integrity, Ethical Programming, Generative AI, Human-AI Collaboration, Programming Education

INTRODUCTION

Perhaps nowhere outside of computer programming can a simple typographical error result in hours of failure and introspection. Such an experience reveals the blend of linguistic, logical, and design precision the discipline requires. Programming demands not only technical acuity but also the capacity for abstraction, problem decomposition, and iterative reasoning (Garcia, 2024). As software systems permeate nearly every domain of modern life, the ability to write and comprehend code has become a foundational skill across both academic and professional contexts. Consequently, the evaluation of programming proficiency holds significant importance to cultivate a culture of computational rigor and innovation. Accurate assessments help ensure that learners internalize not just how to code, but why specific design choices, algorithms, and structures matter in diverse contexts. In the era of artificial intelligence (AI), however, the landscape of programming assessment has undergone a profound shift (Lepp & Kaimre, 2025; Wilson & Nishomoto, 2023). Tools such as ChatGPT and other large language models (LLMs) are now capable of generating syntactically valid and semantically plausible code with minimal human prompting. As emphasized by Bringula (2024) and Garcia (2025), the increasing adoption of these AI technologies in educational settings is reshaping how programming is taught and learned. Novice programmers may rely on LLMs not as scaffolds but as surrogates for the reasoning process, which obscures the boundary between assistance and authorship. The practice of assessing programming knowledge is, therefore, undergoing urgent and necessary scrutiny due to the disruptive capabilities of generative AI.

While generative AI offers promising enhancements to programming instruction, its integration into learning environments introduces substantial challenges to assessment validity. According to rapid review of programming education literature (Garcia, 2025), issues such as plagiarism, over-reliance on AI-generated outputs, and the attenuation of individual cognitive engagement pose serious risks to skill development. These concerns manifest in observable shifts in student behavior, where engagement with foundational problem-solving may be supplanted by superficial interaction with AI tools. Moreover, the traditional indicators of programming competence (e.g., syntactic correctness, output accuracy, and completion time) are increasingly inadequate in distinguishing between work produced through authentic understanding and work generated or heavily augmented by AI. The opacity of generative models, combined with their fluency in mimicking human-like problem solving, makes it exceedingly difficult to detect academic dishonesty using conventional evaluative methods. Left unaddressed, this threatens to undermine both the credibility of assessment outcomes and the reliability of programming credentials. Preserving the integrity of programming education consequently requires a recalibration of assessment strategies that can distinguish between genuine comprehension and passive consumption. Institutions must now develop and adopt methodologies that promote authentic learning, reward process over product, and uphold the core pedagogical goals of computing curricula. Therefore, there is a pressing need to update programming assessment strategies to remain aligned with both technological realities and educational imperatives.

MAIN FOCUS OF THE CHAPTER

In computational disciplines where correctness, efficiency, and algorithmic thinking are paramount, rigorously designed assessments are indispensable. Effective evaluation in programming education is not solely about verifying whether a student can produce syntactically correct or functionally adequate code. Rather, it must determine whether the learner possesses the conceptual depth required to design, analyze, and refine software systems under real-world constraints. Although scholarly interest in the pedagogical implications of generative AI has grown in recent years, a pronounced research gap persists in the domain of educational assessment (Garcia et al., 2025). Most prior works have concentrated on the instructional affordances of AI tools, while comparatively few have addressed the design of assessment practices capable of withstanding the epistemological, ethical, and procedural challenges introduced by such technologies. There is also a notable absence of structured guidance on how to assess programming knowledge and skills in ways that maintain authenticity, safeguard academic integrity, and account for the evolving dynamics of human-AI code collaboration.

Therefore, the primary objective of this chapter is to address this critical gap by articulating a set of best practices and effective strategies for assessing programming competencies in the age of generative AI. It seeks to examine how traditional assessment paradigms are being disrupted by AI-assisted code generation and to offer educators and institutions a set of evidence-informed recommendations that reflect both pedagogical rigor and technological realism. This study is timely and relevant as the absence of a principled redefinition of assessment frameworks creates a tangible risk of accrediting competencies that have not been authentically acquired. As generative AI becomes increasingly ubiquitous and sophisticated, conventional assessments lose their discriminatory capacity to evaluate deep understanding. Unchecked reliance on these AI tools threatens not only the credibility of academic outcomes but also the readiness of graduates to meet the complex demands of professional software development. By addressing this emerging challenge, this chapter contributes to sustaining excellence, trust, and accountability in computer programming education.

METHODS

This chapter adopted an integrative approach to identify and articulate best practices and effective strategies for assessing programming competencies in the age of generative AI (Bozkurt et al., 2024). The methodological orientation is grounded in expert-informed synthesis, drawing upon a broad and interdisciplinary foundation that includes practitioner knowledge, empirical literature, and instructional artifacts. The primary source of insights stems from the authors' cumulative experiences as educators, researchers, and practitioners in the field of computer programming, educational technology, and AI. This experiential knowledge served as a foundational lens for identifying emerging challenges, pedagogical tensions, and practical interventions relevant to assessment in AI-mediated learning environments. In parallel, a wide-ranging review of published case studies, instructional reports, and teaching reflections was

conducted. These papers included materials disseminated through academic journals, conference proceedings, institutional white papers, and open-access repositories where faculty and instructional designers have shared implementation strategies and observed outcomes.

To ensure methodological robustness and relevance, the study also incorporated an analysis of contemporary trends in computing education literature. This involved examining recent scholarship on the integration of LLMs in the classroom, the evolving discourse on academic integrity in the age of AI, and pedagogical innovations aimed at promoting authentic learning (e.g., Garcia, 2025; Mahon et al., 2024). Sources were selected based on their scholarly rigor, contextual relevance, and contribution to ongoing conversations in computer science education, learning sciences, and AI ethics. Additionally, perspectives from peer educators and instructional technologists were solicited through informal interviews, community forums, and reflective blogs. This supplementary evidence offers situated knowledge of classroom-level implications that may not yet be fully captured in peer-reviewed publications. These varied inputs were not treated in isolation but were critically evaluated, triangulated, and synthesized into a coherent framework of recommended practices. The methodological intent of this chapter is not to produce generalizable empirical claims, but rather to construct a well-informed, practice-oriented guide that bridges the gap between theoretical discourse and pedagogical implementation. By integrating diverse sources of evidence and expertise, this work seeks to advance the field's understanding of how programming assessment can be both resilient and responsive in the face of rapidly evolving technological capabilities.

BEST PRACTICES AND STRATEGIES

Emphasize Problem-Solving and Conceptual Understanding

As generative AI systems increasingly possess the capacity to synthesize syntactically valid and functionally accurate code with minimal human input, the pedagogical imperative shifts from assessing rote code production to evaluating higher-order cognitive competencies. In this new paradigm, assessments must interrogate learners' abstract conceptualization, algorithmic reasoning, and procedural metacognition to ensure authentic learning. Emphasizing process over product allows educators to discern genuine computational thinking from superficial correctness.

Emphasize Algorithmic Reasoning and Logical Structure

To circumvent the superficial mimicry afforded by LLMs, assessments should foreground algorithmic construction and the logical architecture of solutions ((Wilson & Nishimoto, 2024). Requiring students to explicate their design rationale (e.g., through structured pseudocode, annotated flow representations, or formal logic statements) elicits evidence of deliberate problem decomposition and heuristic selection. This approach engages learners in cognitive processes aligned with revised Bloom's taxonomy (Krathwohl, 2002), particularly within the domains of "*analyzing*," "*evaluating*," and "*creating*." By privileging algorithmic reasoning over syntactic

fidelity, instructors can appraise the depth of students' procedural schemata and their capacity to adapt algorithms to novel problem constraints. For instance, rather than asking learners to implement a sorting algorithm, a more revealing prompt might be: "*Compare the asymptotic behavior of your chosen approach against a divide-and-conquer alternative and justify your selection under memory-limited conditions*." Such practices promote schema-based learning, foster transfer, and decenter the code artifact in favor of the cognitive operations that precede it.

Assess Conceptual Understanding Through Explanatory Reasoning

A robust indicator of programming proficiency lies in the ability to articulate foundational principles abstracted from specific implementations. Conceptual assessments that require explanatory reasoning (e.g., distinguishing recursion from iteration, delineating stack versus heap memory behavior, or critiquing object-oriented versus functional paradigms) surface the learner's ontological grasp of programming constructs. These prompts transcend surface-level recognition and necessitate semantic elaboration, aligning with constructivist views of learning in which knowledge is actively constructed and contextually grounded. Moreover, they provide a mechanism for assessing the learner's epistemic positioning, whether programming is perceived as a rule-based technical activity or as a conceptual discipline governed by theoretical principles and design trade-offs. For example, a question such as "*Why might tail-recursive functions be preferable in functional languages but largely irrelevant in imperative languages?*" demands a deliberate understanding that is difficult to synthesize without deep disciplinary engagement.

Require Manual Code Tracing and Execution Simulation

Among the most cognitively diagnostic assessment techniques is manual code tracing, wherein learners simulate the runtime behavior of a given code segment to determine outputs, identify latent bugs, or describe the evolution of memory state (Andrei & Nabi, 2023). This practice elicits procedural knowledge and exposes the learner's mental model of execution flow, particularly in the presence of control structures, recursion, and dynamic data handling. Code tracing is underpinned by dual-process theories of reasoning. While novices may rely on heuristic shortcuts, expert performance requires deliberate, system-2 engagement to logically simulate state transitions. Tasks might involve predicting variable values at specific execution points, annotating call stacks, or identifying off-by-one errors in iteration constructs. All these tasks demand meticulous attention to operational semantics. By disallowing reliance on code-generation tools, code tracing compels learners to internalize the causal mechanics of computation. Importantly, it reinforces the bidirectional relationship between code construction and comprehension, which is a pedagogical objective often overlooked in output-focused assessments.

Implement Robust AI-Resistant Assessment Strategies

With generative AI systems increasingly obfuscating the boundary between authored and synthesized code, educators must adopt assessment modalities that render unauthorized

algorithmic assistance less feasible or pedagogically irrelevant (Damaševičius, 2024; Xu & Sheng, 2024). AI-resistant assessments are epistemically aligned with evaluating authentic cognitive labor, design intentionality, and expressive fluency in programming. These modalities foreground human explanation, adaptive reasoning, and real-time synthesis, which are capacities that remain outside the operational scope of current generative models. Traditional assessment practices that overemphasize syntactic correctness and direct implementation are no longer adequate to capture the depth of learners' computational understanding (Kendon et al., 2023). The goal is not to eliminate the presence of AI but to elevate the evidentiary threshold of understanding such that its covert use becomes pedagogically inert (Pan et al., 2024).

Facilitate Live Code Reviews to Evaluate Design Intent

Live code reviews serve as powerful instruments for eliciting authentic evidence of comprehension, decision-making, and iterative reasoning (Selvaraj et al., 2021). These activities are structured dialogues wherein learners present and defend their code to an evaluator. Unlike static code submission, this format requires the learner to articulate the motivations behind architectural choices, algorithmic trade-offs, and stylistic conventions in real time (Indriasari et al., 2020). For example, a student tasked with implementing a pathfinding algorithm might be prompted to justify the choice of A* over Dijkstra's algorithm in a game development context, including an explanation of heuristic admissibility and computational overhead. These interactions surface the learner's depth of conceptual integration and reveal the extent to which the submitted code reflects deliberate construction rather than passive replication. Moreover, code reviews mirror professional software engineering practices that foster industry-aligned competencies (e.g., code readability, maintainability, and peer critique). This format also provides dynamic feedback loops, which allows instructors to probe for conceptual weaknesses, detect overfitting to templates, and challenge overreliance on superficial correctness.

Conduct Oral Defenses to Probe Algorithmic Reasoning

Oral examinations, or viva voce assessments, provide a discursive medium through which learners are invited to externalize their algorithmic thought processes, design logic, and computational intuitions. These examinations are particularly potent in assessing the learner's ability to reason adaptively, respond to critical inquiry, and articulate non-obvious design decisions (Ardito, 2024). For instance, after solving a problem involving backtracking, a student might be asked: "*Under what conditions would a greedy algorithm suffice in this domain, and why might backtracking offer a more generalizable solution?*" Such questions require comparative reasoning, contextual awareness, and the ability to navigate abstract algorithmic paradigms. The oral format disrupts the possibility of answer pre-generation via AI tools, as learners must synthesize responses dynamically and demonstrate genuine fluency in the conceptual grammar of programming. This assessment type also encourages the cultivation of verbal articulation in computational contexts, which is a vital yet often underdeveloped skill in the discipline.

Administer Controlled-Environment Programming Assessments

While not inherently innovative, in-person programming assessments administered in AI-restricted environments are regaining relevance as a baseline safeguard against unauthorized augmentation. Conducting such assessments in supervised laboratories, with network access disabled and development environments monitored, ensures that learners operate solely on internalized knowledge and procedural memory. However, to transcend mere constraint, such assessments should incorporate open-ended prompts that resist brute-force memorization. For example, rather than requesting the implementation of a standard binary search, instructors might ask: "*Design a modified search algorithm for a rotated sorted array and analyze its time complexity*." This type of prompt engages analytical thinking, adaptability, and creative recombination of prior knowledge, which are dimensions of competence that are also difficult to fabricate extemporaneously (Taeb et al., 2024). Controlled-environment tasks also provide valuable data on the learner's coding fluency, debugging approach, and efficiency under cognitive load, all of which are key indicators of real-world programming readiness.

Integrate Constructive AI Use into Assessment Design

The pedagogical response must evolve from prohibition to principled integration as generative AI becomes increasingly embedded in the programming workflow. Rather than viewing AI tools solely as threats to academic integrity, educators can leverage them as metacognitive amplifiers, provided their use is governed by clear parameters and assessed with epistemic nuance (Acut et al., 2025; Gantalao et al., 2025). Integrating AI-use policies and tasks into assessment design promotes transparency, encourages critical engagement with machine-generated outputs, and cultivates students' ability to distinguish between augmentation and automation. This approach reframes AI not as an end but as a catalyst for reflective practice, comparative evaluation, and human-AI collaboration (Boguslawski et al., 2025).

Evaluate Proficiency in AI-Augmented Problem Solving

Assessments can be structured to explicitly permit the use of AI tools during designated stages of the problem-solving process. Rather than penalizing reliance, instructors can evaluate the learner's discernment in how, when, and why such tools were employed—and whether their outputs were critically interrogated or blindly accepted. For example, a programming task might instruct students to use an AI assistant to brainstorm algorithmic approaches to a constraint satisfaction problem (e.g., Sudoku solving). Students would then implement one strategy and annotate their code to indicate which elements were AI-derived and which were their own refinements. Evaluation would center on how effectively the student adapted, optimized, or rejected AI suggestions based on context-specific reasoning and correctness constraints. This mode of assessment positions generative AI as a design interlocutor where students must critically evaluate rather than emulate. It also mirrors contemporary industry practices, where AI-enhanced development requires discernment, ethical awareness, and domain-specific validation.

Elicit Reflective Metacognition Through AI-Use Documentation

Requiring students to submit structured reflections detailing their interaction with AI systems cultivates both metacognitive awareness and academic accountability (Adler & Shani, 2026). These reflective artifacts serve as qualitative evidence of students' decision-making processes, error-correction strategies, and evaluative judgments regarding AI reliability. Reflection prompts might include: "*Describe a point during the task where you considered AI output but chose an alternative approach. What guided your decision?*" Such reflective scaffolds encourage learners to operationalize critical thinking and to engage with AI as a tool to be scrutinized. They also allow educators to assess cognitive independence and epistemic vigilance, which are central to cultivating trustworthy and resilient programmers in AI-augmented contexts.

Compare Human and AI Solutions to Cultivate Evaluative Judgment

An advanced form of critical engagement involves assigning parallel problem-solving tasks (e.g., one completed manually and the other with AI assistance) followed by a comparative analysis of the two solutions. This task structure enables learners to develop evaluative judgment regarding code quality, elegance, robustness, and adherence to problem constraints. For instance, students might be asked to implement a dynamic programming solution to a combinatorial optimization problem (e.g., the knapsack problem) unaided, and then to generate an AI-based solution for the same task. Their comparative reflection could address computational efficiency, code readability, use of abstractions, and error propagation. This strategy serves dual purposes: it demystifies AI capabilities and limitations while simultaneously deepening learners' own analytical competencies. By situating human cognition and machine output in dialogic tension, educators can foster the development of hybrid intelligence wherein students learn not only to program but to reason about programming with and against machines.

Design Open-Ended and Creative Programming Tasks

In contrast to narrowly scoped programming problems that can be trivially solved or replicated by generative AI, open-ended and creative tasks foreground the learner's capacity for innovation, judgment, and integrative thinking. These tasks emphasize originality over replication, process over product, and justification over mere functionality (Silva et al., 2024). By introducing ambiguity, multiple solution paths, and authentic constraints, such assessments compel learners to engage in design-oriented reasoning. In this context, success is defined not by a single correct answer, but by the coherence, viability, and expressiveness of their solution. These tasks are critical in preserving human agency in a landscape increasingly mediated by automated outputs.

Implement Project-Based Assessments to Foster Creative Synthesis

Project-based assessments serve as expansive formats in which students are required to design, implement, and iteratively refine a software artifact that reflects both technical

competency and creative agency. These projects typically extend beyond textbook exercises, involving user-centric design, integration of external libraries or APIs, and adherence to real-world constraints. For instance, students may be asked to develop an educational application for language learners that adapts content based on performance metrics, or to create a simulation tool for visualizing algorithm performance under varying conditions. Such projects require learners to define their own specifications, navigate ambiguous requirements, and make trade-offs across performance, usability, and maintainability. Critically, project-based tasks enable the assessment of not only coding ability but also architectural thinking, version control fluency, collaborative skills (if team-based), and reflective documentation.

Assess Real-World Problem Solving Through Multivariate Scenarios

Designing assessments around authentic, real-world problems invites learners to contend with complexity, multiplicity, and imperfection (Obaidellah et al., 2020). These problems are intentionally underdetermined, offering several plausible solutions, each with its own set of trade-offs. The assessment, therefore, hinges not only on the solution's efficacy but also on the rationale underpinning design decisions. A representative prompt might be: "*Develop a data visualization dashboard for municipal waste management using publicly available datasets. Your solution should balance performance, interpretability, and accessibility for non-technical stakeholders*." Here, students must justify choices about data preprocessing, visualization libraries, UI design, and even ethical considerations in data presentation. This form of assessment prioritizes evaluative reasoning, stakeholder awareness, and system-level thinking, which are core attributes of creative problem-solving. Furthermore, it renders AI-generated solutions insufficient unless they are strategically adapted and substantiated by the learner's design rationale.

Engage Learners Through Code Customization and Extension

Rather than presenting students with blank slates or fully specified problems, instructors can provide scaffolded or partially completed codebases and require learners to customize or extend them in meaningful, context-aware ways. This format disrupts pattern-matching strategies employed by generative AI tools and shifts the cognitive load onto the learner's ability to interpret existing logic, identify constraints, and design contextually appropriate modifications. For example, a student might receive a minimal version of a two-dimensional (2D) physics simulation engine and be asked to implement new object behaviors (e.g., friction or elasticity), refactor the code for modularity, or enhance its performance through algorithmic optimizations. Alternatively, students might be tasked with internationalizing a web application to support right-to-left languages or integrating new accessibility features into a user interface. These tasks assess a blend of comprehension, synthesis, and reengineering, which are also competencies central to real-world development and difficult to automate. They also encourage students to engage with legacy code, interface specifications, and pre-existing design decisions, preparing them for the complexities of collaborative and inherited software environments.

Leverage Automated Tools and Code Quality Metrics

In an educational landscape increasingly shaped by automation, the strategic deployment of automated analysis tools provides instructors with scalable, objective mechanisms for evaluating code authenticity, development behavior, and software quality (Chen et al., 2022; Gambo et al., 2025; Novak et al., 2019). These tools do not merely function as surveillance instruments, but as pedagogical supports that can uncover patterns of engagement, reveal deviations from normative coding practices, and offer diagnostic insights into the learner's developmental trajectory. When judiciously applied, automated tools complement qualitative assessment by enabling the continuous monitoring of both code provenance and structural integrity, which are domains that are susceptible to obfuscation in the age of generative AI.

Detect Anomalous Code Similarity Through Algorithmic Plagiarism Analysis

Automated similarity detection tools (e.g., Measure of Software Similarity, JPlag, or Codequiry) offer robust methods for identifying textual or structural overlap across student submissions and known code corpora. These tools apply token-based, tree-based, or fingerprinting algorithms to detect semantic equivalence or suspicious convergence in code artifacts, even when superficial modifications (e.g., variable renaming, control flow inversion) are introduced to evade detection. In the context of generative AI, such tools become particularly relevant for flagging artifacts that exhibit high degrees of similarity with publicly available training data or canonical code templates. For instance, an introductory assignment on linked list implementation that yields structurally identical solutions across cohorts may warrant further inspection, especially when accompanied by limited verbal justification from the learner. While such tools should never serve as the sole arbiter of academic misconduct, they provide empirical grounds for initiating reflective dialogue and further investigation. Their integration into the assessment pipeline promotes a culture of originality and attribution that are essential in both academia and industry.

Analyze Version Control Histories to Evaluate Development Authenticity

Mandating the use of distributed version control systems such as *Git* not only inculcates professional software engineering practices but also provides a temporal record of development activity that can be pedagogically invaluable. Commit histories, branch structures, and diff logs offer insight into the evolution of student work, revealing whether code was developed iteratively or submitted as a monolithic artifact. This aspect is an indicator often associated with last-minute synthesis or potential AI intervention. For example, an analysis of commit logs may show a student progressively implementing modules across several days, interspersed with debugging and refactoring activities. In contrast, a single commit containing the entirety of a completed project, followed by negligible activity, may suggest copy-paste behavior or generative augmentation. Instructors may further operationalize version control analysis through rubrics that reward process transparency, granular development, and meaningful commit messages. These metrics reflect not only the student's technical proficiency but also their reflective engagement.

Employ Static Code Analysis to Measure Software Quality

Automated tools that assess code quality (e.g., *SonarQube*, *ESLint*, or *Pylint*) offer quantitative indicators of software robustness, readability, and adherence to language-specific idioms. These tools can detect issues ranging from cyclomatic complexity and dead code to naming convention violations and poor modularization, all of which serve as proxies for maintainability and technical debt. Importantly, while generative AI can often produce syntactically valid code, it does not consistently enforce idiomatic best practices, context-specific conventions, or maintainable architectural patterns. Thus, integrating code quality metrics into assessment schemes allows educators to reward not just functional correctness, but also craftsmanship and discipline. For example, a student might submit a fully functional web scraper that passes all tests but contains hard-coded dependencies, nested loops with excessive complexity, and poor documentation. Static analysis would flag these issues, and the student's grade could reflect the difference between merely working code and professionally viable code. These metrics can be transparently shared with learners to support formative feedback, enabling them to iteratively improve code hygiene and internalize best practices over time.

Assess Debugging Proficiency and Maintenance Skills

While traditional programming assessments often emphasize code generation, real-world software development disproportionately involves debugging, refactoring, and maintaining existing systems. These maintenance-oriented tasks demand not only fluency in program comprehension but also critical thinking, error localization, and architectural foresight. In the age of generative AI, where functional code can be synthesized rapidly (Garcia, 2025; Garcia et al., 2023), the ability to diagnose, improve, and sustain code quality over time emerges as a distinctive marker of human expertise. Assessing these capacities foregrounds the learner's ability to engage with the full lifecycle of code to ensure robustness, clarity, and longevity.

Assess Debugging Proficiency Through Fault Localization and Resolution Tasks

Deliberately introducing semantic, logical, or structural errors into code and tasking students with identifying and resolving them serves as a rigorous diagnostic for their debugging acumen. Unlike generative code synthesis, debugging requires granular comprehension of control flow, data dependencies, and potential side effects. Programmers are expected to possess such competencies that are cognitively intensive and pedagogically revealing. For example, students might receive a recursive implementation of a depth-first search algorithm that fails on edge cases due to incorrect base conditions or mutable default arguments. The assessment would require them not only to correct the behavior but to articulate the source of the failure, propose a fix, and validate it through targeted testing. Such tasks align with real-world software engineering practices, where the ability to efficiently isolate faults and deploy minimally invasive corrections distinguishes competent programmers from novice coders. They also shift the focus from generation to explanation, which AI tools currently struggle to perform authentically.

Evaluate Code Maturity Through Structured Refactoring Exercises

Refactoring tasks assess a learner's capacity to enhance the internal structure of code without altering its external functionality. This activity involves applying design principles (e.g., modularity and abstraction), as well as identifying and eliminating code smells (i.e., suboptimal patterns that may impede scalability, readability, or maintainability). A representative task might involve presenting students with a monolithic data processing script and asking them to refactor it into a series of modular, reusable functions, each with clearly defined input-output contracts. Alternatively, students could be required to introduce design patterns (e.g., strategy, observer) into an object-oriented system to improve extensibility. These assessments surface the student's architectural thinking and ability to transform working but inelegant code into a maintainable artifact, which is an essential skill in collaborative, long-term software projects. Moreover, unlike AI-generated code, which often prioritizes minimal working examples, human-led refactoring reflects deeper comprehension of maintainability as a core software quality attribute.

Assess Documentation and Testing as Markers of Software Reliability

The ability to produce clear, coherent documentation and comprehensive test suites constitutes a critical yet often neglected component of programming proficiency. While AI can generate plausible docstrings or test stubs, it lacks the context sensitivity and domain judgment required for high-quality specification and coverage assurance. In assessment contexts, students may be tasked with writing module-level documentation that explains code behavior, design rationale, and expected usage patterns for a non-expert developer. Concurrently, they could be evaluated on their ability to develop unit, integration, or boundary tests using frameworks such as *pytest*, *JUnit*, or *Mocha*, with attention to test coverage, assertions, and edge case handling. For example, a task might involve augmenting an existing codebase with meaningful inline comments, API documentation, and a suite of parameterized tests that achieve at least 80% coverage. Evaluation would consider not only completeness but also clarity, consistency, and pedagogical value, which are attributes essential to maintainable and collaborative code. This mode of assessment reinforces the notion that high-quality software is not merely code that runs, but code that communicates intent, resists regression, and invites reuse.

Facilitate Collaborative and Peer-Based Assessments

Collaboration is not ancillary but foundational in software development. Modern programming environments are inherently social, with tools and practices designed to facilitate shared code ownership, distributed problem-solving, and continuous integration (Garcia, 2023). As such, assessments must capture individual technical proficiency and evaluate interpersonal and collaborative competencies that are central to real-world programming practice. Collaborative and peer-based assessment strategies cultivate dialogic reasoning, expose learners to diverse design perspectives, and reinforce metacognitive reflection (Prather et al., 2024). These dimensions of learning are both resistant to automation and essential to professional formation.

Evaluate Collaborative Fluency Through Structured Pair Programming

Pair programming is an agile practice wherein two programmers work in tandem, alternating between "driver" and "navigator" roles. According to prior studies, this strategy offers a pedagogically rich context for assessing both technical execution and real-time collaborative discourse. In academic settings, this modality can be structured to ensure role rotation, equitable participation, and reflective accountability. Assessment can include observational rubrics evaluating communication quality, task division, and responsiveness to feedback, supplemented by post-session reflections that document problem-solving strategies, conflict resolution, and individual learning gains. For instance, students might be assigned a problem involving concurrent data access in a multithreaded environment, where one partner implements synchronization primitives while the other audits for race conditions and data integrity. This format promotes code comprehension as a social process, reinforcing the development of soft skills (e.g., active listening, articulation of technical rationale, and shared debugging) that are essential in collaborative engineering contexts but often neglected in traditional assessments.

Assess Critical Engagement Through Peer Code Reviews

Peer code review exercises expose students to diverse programming styles, implementation strategies, and design rationales, fostering evaluative judgment and collective learning. Instructors can scaffold this process with review templates that prompt analysis along key dimensions: correctness, readability, modularity, adherence to conventions, and potential for optimization. Tools such as *PuzzleMe* and *AI-Lab* have been shown to support live peer discussion and feedback, enabling students to collaboratively debug, optimize, and understand alternate implementation strategies in real time (Dickey et al., 2024). For example, students may be tasked with reviewing a peer's implementation of a machine learning model pipeline, offering comments on data preprocessing practices, feature selection, and hyperparameter tuning logic. Each reviewer would be assessed on the technical validity of their critique, constructiveness, specificity, and ability to justify suggestions with reference to established practices. This approach encourages learners to articulate tacit knowledge, surface latent misconceptions, and internalize quality benchmarks through exposure to both exemplary and flawed code. Moreover, the reciprocal nature of peer reviews fosters a culture of accountability and reflective refinement that AI-driven assessments cannot replicate (Wang et al., 2021; Yang et al., 2024).

Design Team-Based Projects to Simulate Professional Development Ecosystems

Team projects, especially those structured to emulate real-world development cycles (Garcia, 2021), represent a comprehensive assessment format. They capture a wide spectrum of learning outcomes, from systems thinking and division of labor to conflict negotiation and integration testing. Empirical evidence suggests that such collaborative and open-ended prototyping tasks produce more engaged learners and superior learning outcomes compared to instructor-led feedback models (Soomro et al., 2024). These projects should be sufficiently

complex to necessitate genuine collaboration, interdependence, and project management. For example, students might be assigned the task of developing a cross-platform mobile application with back-end integration, requiring the delineation of roles such as front-end development, API design, database modeling, and deployment automation. Assessment rubrics can incorporate peer evaluation, role-based deliverables, shared repositories, and sprint reviews to capture both individual accountability and collective output (Alipour et al., 2024). By engaging in sustained collaboration under authentic constraints, students develop not only technical and architectural skills but also professional habits of mind. These skills include version control etiquette, asynchronous coordination, and codebase stewardship, which can prepare them for industrial software ecosystems. These competencies remain beyond the scope of generative AI and affirm the human-centered nature of complex problem-solving.

Ensure Submission Authenticity through Structural Design

In the current educational climate, where generative AI can produce seemingly original code artifacts instantaneously, safeguarding the authenticity of student submissions is an urgent pedagogical imperative. The challenge is not simply one of detection, but of designing assessment protocols that promote transparency, traceability, and evidentiary accountability. Authenticity must be verified through punitive measures and structural features that render inauthentic shortcuts impractical or pedagogically ineffective. By embedding temporal markers, requiring multimodal demonstrations, and diversifying input conditions, educators can foster a setting where authentic engagement becomes the most viable path to success (Velicea et al., 2025).

Document Development Through Progressive Snapshots

Requiring students to submit incremental versions of their work (e.g., design sketches, pseudocode drafts, partial implementations, and early test results) provides a temporal footprint of their problem-solving trajectory. These progress snapshots serve both formative and forensic purposes (Garcia & Revano Jr, 2021; Obaidellah et al., 2020; Semenov et al., 2025). They encourage iterative development while simultaneously furnishing instructors with evidence of authentic authorship. For example, in a two-week assignment on developing a file encryption utility, students might be required to submit: (1) a design outline detailing algorithm choice and file handling logic, (2) a mid-project codebase with placeholder functions, and (3) a final implementation with testing scripts. This approach not only deters last-minute AI-generated submissions but also reinforces habits of incremental problem-solving and reflection. Furthermore, instructors can analyze changes in naming conventions, code structuring, and logic refinement to identify whether the work exhibits evidence of authentic intellectual labor. In effect, progress documentation transforms authenticity verification into a pedagogical opportunity for intervention and growth. This scaffolding approach also facilitates early intervention, which allows instructors to provide targeted feedback before the final submission. This strategy consequently transforms authenticity enforcement into a pedagogical support mechanism.

Elicit Verbal Explanation Through Code Walkthrough Videos

Video demonstrations wherein students record themselves explaining and executing their programs provide a compelling mechanism for verifying conceptual ownership. Unlike static code, which may be copied or generated, the ability to fluently articulate design decisions, control flow, and edge case behavior in real time serves as a proxy for internalized understanding (Hutson et al., 2024). A typical requirement might be a 5-minute screen recording in which the student walks through the code structure, explains key functions and their interactions, and narrates a test run using example input data (Xue et al., 2024). Evaluators can use rubrics to assess not only technical accuracy but also fluency, precision of terminology, and responsiveness to hypothetical variations. Such demonstrations are difficult to outsource or fabricate using generative tools and offer a human-centered form of assessment that privileges explanation over artifact.

Mitigate Plagiarism Through Input Randomization and Individualization

One of the most effective design-based deterrents to AI-assisted plagiarism is the incorporation of randomized or individualized parameters into assessment prompts. By modifying datasets, function specifications, or constraint sets across students, instructors reduce the feasibility of sharing solutions or querying generic AI tools for problem-specific code (Karnalim, 2023). For example, in a data processing task, students might receive structurally distinct CSV files, compelling them to customize parsing routines. Tools such as *CodeRunner* (Hickman et al., 2023), when integrated into LMS environments, can automate the generation of customized task variants, making the deployment of randomized assessments scalable and consistent. Furthermore, evidence suggests that generative models like ChatGPT perform poorly on algorithmically complex or highly contextualized problems, especially when compared to generic tasks (Dunder et al., 2024). In one study involving 127 Kattis-style problems, ChatGPT succeeded on only 19, underscoring the effectiveness of complexity and variation in preserving assessment integrity. Randomization thus accomplishes multiple objectives: it deters plagiarism, promotes problem decomposition and abstraction, and upholds fairness by distributing unique but conceptually aligned problems across a cohort (Zhang et al., 2024). When combined with verbal explanations and process documentation, it forms a triadic model of AI-resilient assessment that is grounded in reasoning, originality, and individualized challenge.

SYNTHESIS AND FUTURE DIRECTIONS

The rapid diffusion of generative AI has irrevocably altered the epistemological terrain of programming education. These systems have blurred traditional boundaries between production and reproduction, authorship and assistance, rendering obsolete any assessment approach that relies solely on syntactic generation or surface-level correctness. As argued throughout this chapter, the imperative now is to reorient programming assessments toward the elicitation of epistemic authenticity, or evidence that learners can reason algorithmically, reflect metacognitively, and engage critically with computational tools. This shift must not be viewed as a

defensive posture but rather as a reimagining of educational design for an era of human–AI collaboration (Bozkurt et al., 2024; Xiao et al., 2025). In this emerging paradigm, assessments must interrogate students' capacity to curate, evaluate, and contextualize AI-generated outputs rather than merely replicate or reject them (Yilmaz & Karaoglan Yilmaz, 2023). The most meaningful metrics of programming proficiency will increasingly reside not in code artifacts themselves, but in the cognitive and ethical sophistication with which those artifacts are critiqued and refined. To support this reorientation, Figure 1 presents the AI-Resilient Programming Assessment Framework that offers a coherent structure for designing assessments that are not only resistant to generative AI misuse but also pedagogically enriching.

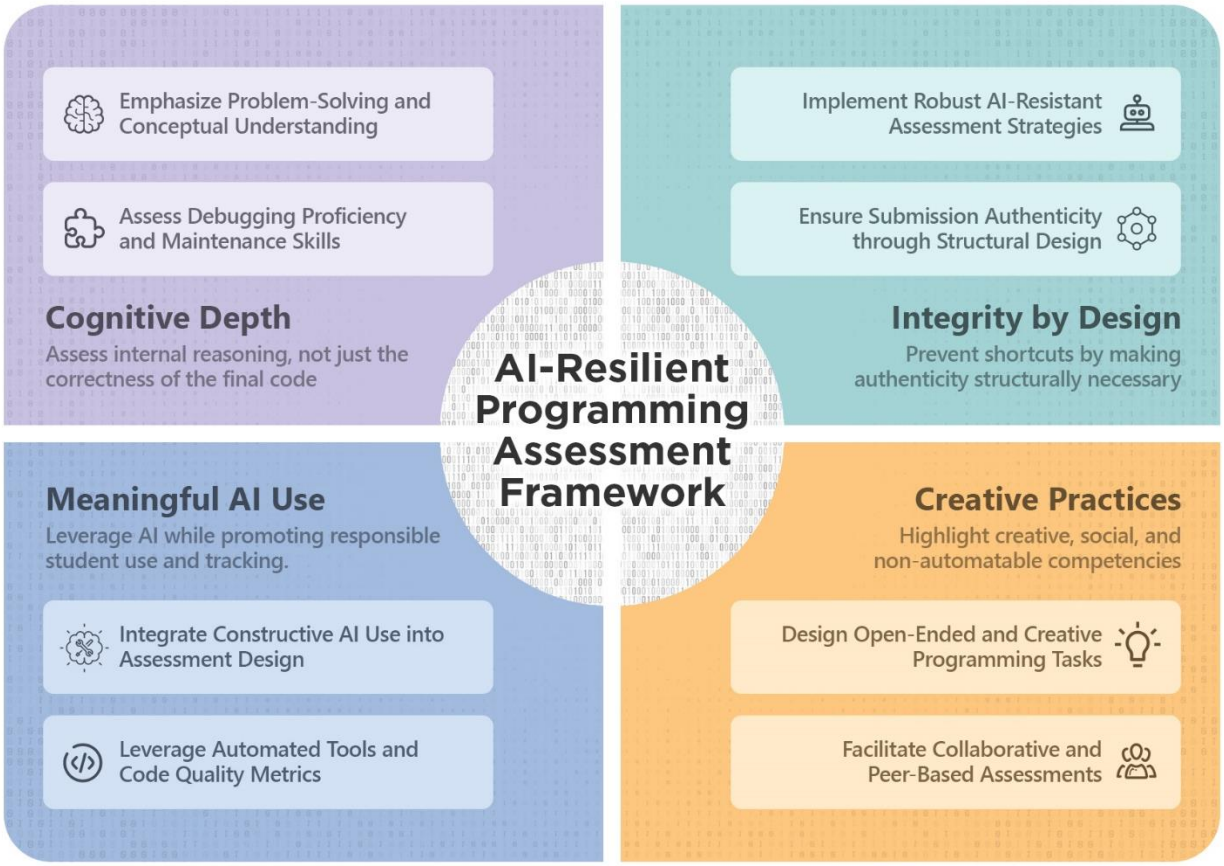

Figure 1. AI-Resilient Programming Assessment Framework

Adaptive Platforms and Personalized Learning Trajectories

Short-term trends already point to the mainstream adoption of AI-driven instructional systems capable of dynamically tailoring task difficulty, feedback, and pacing based on granular learner data (Hasanah et al., 2025). These platforms leverage reinforcement learning and predictive modeling to scaffold student progress in real-time, resulting in a shift from static curricula to adaptive cognitive ecologies. While such systems hold promise for reducing attrition and enhancing personalization, they simultaneously raise critical concerns regarding algorithmic

opacity, data privacy, and pedagogical accountability. As Garcia et al. (2023) emphasized, it is essential that adaptive systems be governed by design principles that foreground transparency, auditability, and inclusiveness. Without such safeguards, personalization may inadvertently entrench digital privilege or reinforce deficit-based learning models.

Elevating Higher-Order Cognitive and Ethical Competencies

As generative systems automate increasingly complex coding tasks, human value will shift toward competencies that resist automation: abstraction, self-regulation, interdisciplinary integration, and ethical reasoning. Assessment frameworks must evolve accordingly to center these higher-order constructs. For instance, tasks that ask students to deconstruct flawed AI-generated code, anticipate emergent behavior, or mitigate algorithmic bias provide opportunities to assess computational judgment as distinct from code proficiency. Moreover, the ethical dimensions of programming must be re-centered. This aspect includes evaluating how students reason about fairness, sustainability, intellectual responsibility, and unintended consequences. Interdisciplinary assessments (Garcia et al., 2025), such as designing socially impactful systems in domains like health informatics or environmental modeling, can serve as platforms for holistic evaluation, connecting code to context and logic to lived reality.

Blockchain for Verifiable Learning and Academic Integrity

Blockchain infrastructures offer a novel mechanism for decentralizing credentialing and preserving the integrity of student-generated work (Babu et al., 2026; El Koshiry et al., 2023). By embedding time-stamped, tamper-evident records of student contributions (e.g., code commits, design iterations, and peer feedback), blockchain systems enable the construction of verifiable learning transcripts that transcend static summative evaluations. Such architectures are particularly potent in countering the obfuscation introduced by generative AI, as they provide granular evidence of developmental progression rather than isolated end products. From an institutional perspective, blockchain may further function as a deterrent to academic misconduct and an enabler of scalable, trust-based assessment in distributed learning ecosystems.

Immersive Technologies and Experiential Assessment

The affordances of immersive technologies, particularly virtual and augmented reality, enable assessment modalities that go beyond what traditional screen-based environments can capture (Alexander et al., 2022). These platforms allow for embodied coding experiences where learners interact with algorithms spatially, engage in real-time collaboration, or co-program alongside AI agents in 3D simulations. In such contexts, the scope of assessment can be extended to include spatial reasoning, situated problem-solving, and real-time decision-making. For example, debugging a robot's pathfinding routine in a virtual warehouse or explaining machine logic to a simulated stakeholder offers a multidimensional view of learner competence that integrates communication, systems thinking, and performance under uncertainty.

Continuous, Modular, and Stackable Credentialing

The linearity of traditional academic credentialing systems is increasingly misaligned with the rapid skill turnover in computing fields. Micro-credentialing and modular assessment frameworks address this disconnect by offering granular, just-in-time verification of competencies across the learning continuum (Alenezi et al., 2024; Gamage & Dehideniya, 2025; Ralston, 2021). These systems enable students to assemble credential portfolios over time, validated through rigorously assessed modules (e.g., secure coding, testing automation, or algorithmic ethics), often in partnership with external platforms such as *GitHub* or *Coursera*. Beyond providing flexibility, these models promote learning as a lifelong, recursive process that emphasizes depth, adaptability, and evidence-based progress. When integrated with institutional standards and assessment rigor, stackable credentials can complement formal qualifications while making learning trajectories more transparent and transferrable (Louder, 2025).

Governance, Regulation, and Faculty Preparedness

The incorporation of AI into programming assessment requires not only pedagogical innovation but also structural reform (Garcia, 2025). Regulatory bodies must address emerging concerns around liability for AI-generated outputs, attribution of authorship, and intellectual property rights (Mangubat et al., 2025). Without such frameworks, the legitimacy of assessment systems will remain contested. Equally critical is the professionalization of the academic workforce in response to these shifts. Faculty must be equipped with the conceptual tools and technical fluency necessary to design, manage, and interpret AI-integrated assessments. Institutions should prioritize sustained in-service training that spans ethical AI use, data literacy, and the design of human-centered evaluation systems (Azevedo et al., 2025; Barus et al., 2025; Luo, 2024). Finally, ensuring equitable access to emerging tools and infrastructure must remain a central priority. In a global education system still marked by digital stratification, efforts to innovate must be accompanied by commitments to inclusion—lest the benefits of AI-enhanced assessment accrue only to those with privileged access to connectivity and computational resources. Finally, digital equity remains a non-negotiable imperative. Ensuring that all students, regardless of geography or socioeconomic status, have reliable access to high-quality tools and connectivity is essential to prevent the amplification of existing educational inequities.

CONCLUSION

The stark reality confronting programming education today is that generative AI can now produce correct, efficient, and stylistically coherent code on demand and at scale. This new normal threatens to collapse long-standing assessment frameworks that equate output with understanding, and performance with authorship. In this rapidly shifting terrain, educators are no longer asking whether students can write code, but whether they understand what it means, where it comes from, and why it matters. This chapter contributes a timely and necessary intervention into the emerging literature on programming assessment by proposing a

comprehensive framework for evaluating coding knowledge and skills in an era saturated by AI augmentation. While prior studies have largely focused on AI's pedagogical affordances in isolation, this work foregrounds the intersections of cognitive authenticity, technological disruption, and ethical design. In doing so, it responds to an urgent need for research-informed guidance on how assessment practices must evolve alongside technological capabilities.

The proposed strategies speak to multiple constituencies. Educators gain concrete, implementable models for designing assessments that foreground learner intent, cognitive process, and reflective capacity. Institutional leaders are equipped with a roadmap for developing resilient, future-oriented assessment infrastructures. Students, in turn, benefit from a learning environment that recognizes and rewards metacognitive engagement and ethical judgment, not just technical output. Meanwhile, the framework calls on policymakers to enact regulatory measures and infrastructural investments that uphold academic integrity and equitable access in digitally mediated contexts. The implications extend far beyond classroom practice. From curricular innovation and credentialing models to global conversations on AI governance in education, the future of programming assessment will demand continuous dialogue between pedagogy, policy, and technological development. Without deliberate recalibration, there is a risk that AI will erode the very conditions under which meaningful learning occurs. But with thoughtful design, these same tools can elevate assessment from a measure of output to a demonstration of human thought, purpose, and values.

REFERENCES

Acut, D. P., Gamusa, E. V., Pernaa, J., Yuenyong, C., Pantaleon, A. T., Espina, R. C.,...Garcia, M. B. (2025). AI Shaming Among Teacher Education Students: A Reflection on Acceptance and Identity in the Age of Generative Tools. In *Pitfalls of AI Integration in Education: Skill Obsolescence, Misuse, and Bias*. IGI Global. https://doi.org/10.4018/979-8-3373-0122-8.ch005

Adler, I., & Shani, Y. (2026). Hybrid Human-GenAI Cognitive Apprenticeship: Encouraging Pre-Service Teachers to Implement Instructional Practices to Support Students' Self-Regulated Learning. *Teaching and Teacher Education*, *169*, 1-19. https://doi.org/10.1016/j.tate.2025.105268

Alenezi, M., Akour, M., & Alfawzan, L. (2024). Evolving Microcredential Strategies for Enhancing Employability: Employer and Student Perspectives. *Education Sciences*, *14*(12), 1-24. https://doi.org/10.3390/educsci14121307

Alexander, B., Hou, Y., Khan, B., & Jin, J. (2022, 28-31 March 2022). Learn Programming In Virtual Reality? A Case Study of Computer Science Students. 2022 IEEE Global Engineering Education Conference (EDUCON),

Alipour, S., Elahimanesh, S., Jahanzad, S., Mohammadi, I., Morassafar, P., Neshaei, S. P., & Tefagh, M. (2024). Improving Grading Fairness and Transparency with Decentralized Collaborative Peer Assessment. *Proceedings of the ACM on Human-Computer Interaction*. https://doi.org/10.1145/3637350

Andrei, O., & Nabi, S. W. (2023). On Students' Experiences with Algorithm Tracing using Pair Programming. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 2*. https://doi.org/10.1145/3568812.3603477

Ardito, C. G. (2024). Generative AI Detection in Higher Education Assessments. *New Directions for Teaching and Learning*. https://doi.org/10.1002/tl.20624

Azevedo, L., Robles, P., Best, E., & Mallinson, D. J. (2025). Institutional Policies on Artificial Intelligence in Higher Education: Frameworks and Best Practices for Faculty. *New Directions for Adult and Continuing Education*, *2025*(188), 70-78. https://doi.org/10.1002/ace.70013

Babu, S., Anbumozhi, A., S, P., N, V., & Kaliamoorthy, V. (2026). Blockchain for Credentialing and Academic Record-Keeping. In J. Moore, S. Gupta, M. Sharma, A. Garg, & H. Josephine V. L (Eds.), *Transforming Education With Data Science in the AI Era* (pp. 407-448). IGI Global Scientific Publishing. https://doi.org/10.4018/979-8-3373-2185-1.ch014

Barus, O. P., Hidayanto, A. N., Handri, E. Y., Sensuse, D. I., & Yaiprasert, C. (2025). Shaping Generative AI Governance in Higher Education: Insights from Student Perception. *International Journal of Educational Research Open*, *8*, 1-12. https://doi.org/10.1016/j.ijedro.2025.100452

Boguslawski, S., Deer, R., & Dawson, M. G. (2025). Programming Education and Learner Motivation in the Age of Generative AI: Student and Educator Perspectives. *Information and Learning Sciences*, *126*(1/2), 91-109. https://doi.org/10.1108/ILS-10-2023-0163

Bozkurt, A., Xiao, J., Farrow, R., Bai, J. Y. H., Nerantzi, C., Moore, S.,...Asino, T. I. (2024). The Manifesto for Teaching and Learning in a Time of Generative AI: A Critical Collective Stance to Better Navigate the Future. *Open Praxis*, *16*(4), 487-513. https://doi.org/10.55982/openpraxis.16.4.777

Bringula, R. (2024). ChatGPT in a Programming Course: Benefits and Limitations. *Frontiers in Education*, *9*, 1-6. https://doi.org/10.3389/feduc.2024.1248705

Chen, H.-M., Nguyen, B.-A., & Dow, C.-R. (2022). Code-Quality Evaluation Scheme for Assessment of Student Contributions to Programming Projects. *Journal of Systems and Software*, *188*, 1-18. https://doi.org/10.1016/j.jss.2022.111273

Damaševičius, R. (2024). ChatGPT-Supported Student Assessment – Can we rely on it? *Journal of Research in Innovative Teaching & Learning*, *17*(2), 414-416. https://doi.org/10.1108/JRIT-09-2024-195

Dickey, E., Bejarano, A., & Garg, C. (2024). AI-Lab: A Framework for Introducing Generative Artificial Intelligence Tools in Computer Programming Courses. *SN Computer Science*, *5*(6), 720. https://doi.org/10.1007/s42979-024-03074-y

Dunder, N., Lundborg, S., Wong, J., & Viberg, O. (2024). Kattis vs ChatGPT: Assessment and Evaluation of Programming Tasks in the Age of Artificial Intelligence. *Proceedings of the 14th Learning Analytics and Knowledge Conference*. https://doi.org/10.1145/3636555.3636882

El Koshiry, A., Eliwa, E., Abd El-Hafeez, T., & Shams, M. Y. (2023). Unlocking the Power of Blockchain in Education: An Overview of Innovations and Outcomes. *Blockchain: Research and Applications*, *4*(4), 1-19. https://doi.org/10.1016/j.bcra.2023.100165

Gamage, K. A. A., & Dehideniya, S. C. P. (2025). Unlocking Career Potential: How Micro-Credentials Are Revolutionising Higher Education and Lifelong Learning. *Education Sciences*, *15*(5), 1-18. https://doi.org/10.3390/educsci15050525

Gambo, I., Abegunde, F.-J., Gambo, O., Ogundokun, R. O., Babatunde, A. N., & Lee, C.-C. (2025). GRAD-AI: An Automated Grading Tool for Code Assessment and Feedback in Programming Course. *Education and Information Technologies*, *30*(7), 9859-9899. https://doi.org/10.1007/s10639-024-13218-5

Gantalao, L. C., Calzada, J. G. D., Capuyan, D. L., Lumantas, B. C., Acut, D. P., & Garcia, M. B. (2025). Equipping the Next Generation of Technicians: Navigating School Infrastructure and Technical Knowledge in the Age of AI Integration. In *Pitfalls of AI Integration in Education: Skill Obsolescence, Misuse, and Bias*. IGI Global. https://doi.org/10.4018/979-8-3373-0122-8.ch009

Garcia, M. B. (2021). Cooperative Learning in Computer Programming: A Quasi-Experimental Evaluation of Jigsaw Teaching Strategy with Novice Programmers. *Education and Information Technologies*, *26*(4), 4839-4856. https://doi.org/10.1007/s10639-021-10502-6

Garcia, M. B. (2023). Facilitating Group Learning Using an Apprenticeship Model: Which Master is More Effective in Programming Instruction? *Journal of Educational Computing Research*, *61*(6), 1207-1231. https://doi.org/10.1177/07356331231170382

Garcia, M. B. (2024). Profiling the Skill Mastery of Introductory Programming Students: A Cognitive Diagnostic Modeling Approach. *Education and Information Technologies*, *30*(5), 6455-6481. https://doi.org/10.1007/s10639-024-13039-6

Garcia, M. B. (2025). Teaching and Learning Computer Programming Using ChatGPT: A Rapid Review of Literature Amid the Rise of Generative AI Technologies. *Education and Information Technologies*, 1-25. https://doi.org/10.1007/s10639-025-13452-5

Garcia, M. B., & Revano Jr, T. F. (2021). Assessing the Role of Python Programming Gamified Course on Students' Knowledge, Skills Performance, Attitude, and Self-Efficacy. *2021 IEEE 13th International Conference on*

Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM). https://doi.org/10.1109/HNICEM54116.2021.9731935

Garcia, M. B., Revano Jr., T. F., Maaliw III, R. R., Lagrazon, P. G. G., Valderama, A. M. C., Happonen, A.,...Yilmaz, R. (2023). Exploring Student Preference Between AI-Powered ChatGPT and Human-Curated Stack Overflow in Resolving Programming Problems and Queries. *2023 IEEE 15th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*. https://doi.org/10.1109/HNICEM60674.2023.10589162

Garcia, M. B., Rosak-Szyrocka, J., Yılmaz, R., Metwally, A. H. S., Acut, D. P., Ofosu-Ampong, K.,...Bozkurt, A. (2025). Rethinking Educational Assessment in the Age of Generative AI: Actionable Strategies to Mitigate Academic Dishonesty. In *Pitfalls of AI Integration in Education: Skill Obsolescence, Misuse, and Bias*. IGI Global. https://doi.org/10.4018/979-8-3373-0122-8.ch001

Hasanah, N. A., Aziza, M. R., Junikhah, A., Arif, Y. M., & Garcia, M. B. (2025). Navigating the Use of AI in Engineering Education Through a Systematic Review of Technology, Regulations, and Challenges. In *Pitfalls of AI Integration in Education: Skill Obsolescence, Misuse, and Bias*. IGI Global. https://doi.org/10.4018/979-8-3373-0122-8.ch016

Hickman, H., McKeown, P., & Bell, T. (2023). Beyond Question Shuffling: Randomization Techniques in Programming Assessment. *2023 IEEE Frontiers in Education Conference (FIE)*. https://doi.org/10.1109/FIE58773.2023.10342976

Hutson, J., Fulcher, B., & Ratican, J. (2024). Enhancing Assessment and Feedback in Game Design Programs: Leveraging Generative AI for Efficient and Meaningful Evaluation. *IJERI: International Journal of Educational Research and Innovation*(22), 1-20. https://doi.org/10.46661/ijeri.11038

Indriasari, T. D., Luxton-Reilly, A., & Denny, P. (2020). A Review of Peer Code Review in Higher Education. *ACM Transactions on Computing Education*, *20*(3), 1-25. https://doi.org/10.1145/3403935

Karnalim, O. (2023). Maintaining Academic Integrity in Programming: Locality-Sensitive Hashing and Recommendations. *Education Sciences*, *13*(1), 1-23. https://doi.org/10.3390/educsci13010054

Kendon, T., Wu, L., & Aycock, J. (2023). AI-Generated Code Not Considered Harmful. *Proceedings of the 25th Western Canadian Conference on Computing Education*. https://doi.org/10.1145/3593342.3593349

Krathwohl, D. R. (2002). A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice*, *41*(4), 212-218.

Lepp, M., & Kaimre, J. (2025). Does Generative AI Help in Learning Programming: Students' Perceptions, Reported Use and Relation to Performance. *Computers in Human Behavior Reports*, *18*, 1-12. https://doi.org/10.1016/j.chbr.2025.100642

Louder, J. R. (2025). Alternative Educational Pathways: Stackable and Micro-Credentials to Bridge Education and Employment. In *The Emerald Handbook on International Higher Education: Navigating Workforce and Leadership Changes in a Digital Age*. Emerald Publishing Limited. https://doi.org/10.1108/978-1-83549-788-320251007

Luo, J. (2024). A Critical Review of GenAI Policies in Higher Education Assessment: A Call to Reconsider the "Originality" of Students' Work. *Assessment & Evaluation in Higher Education*, *49*(5), 651-664. https://doi.org/10.1080/02602938.2024.2309963

Mahon, J., Namee, B. M., & Becker, B. A. (2024). Guidelines for the Evolving Role of Generative AI in Introductory Programming Based on Emerging Practice. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. https://doi.org/10.1145/3649217.3653602

Mangubat, J. C., Mangubat, M. R., Uy, T. B. L., Acut, D. P., & Garcia, M. B. (2025). Safeguarding Educational Innovations Amid AI Disruptions: A Reassessment of Patenting for Sustained Intellectual Property Protection. In *Pitfalls of AI Integration in Education: Skill Obsolescence, Misuse, and Bias*. IGI Global. https://doi.org/10.4018/979-8-3373-0122-8.ch013

Novak, M., Joy, M., & Kermek, D. (2019). Source-Code Similarity Detection and Detection Tools Used in Academia: A Systematic Review. *ACM Transactions on Computing Education*, *19*(3), 1-37. https://doi.org/10.1145/3313290

Obaidellah, U., Blascheck, T., Guarnera, D. T., & Maletic, J. (2020). A Fine-Grained Assessment on Novice Programmers' Gaze Patterns on Pseudocode Problems. *ACM Symposium on Eye Tracking Research and Applications*. https://doi.org/10.1145/3379156.3391982

Pan, W. H., Chok, M. J., Wong, J. L. S., Shin, Y. X., Poon, Y. S., Yang, Z.,...Lim, M. K. (2024). Assessing AI Detectors in Identifying AI-Generated Code: Implications for Education. *Proceedings of the 46th International*

Conference on Software Engineering: Software Engineering Education and Training. https://doi.org/10.1145/3639474.3640068

Prather, J., Reeves, B. N., Leinonen, J., MacNeil, S., Randrianasolo, A. S., Becker, B. A.,...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*. https://doi.org/10.1145/3632620.3671116

Ralston, S. J. (2021). Higher Education's Microcredentialing Craze: a Postdigital-Deweyan Critique. *Postdigital Science and Education*, *3*(1), 83-101. https://doi.org/10.1007/s42438-020-00121-8

Selvaraj, A., Zhang, E., Porter, L., & Raj, A. G. S. (2021). Live Coding: A Review of the Literature. *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. https://doi.org/10.1145/3430665.3456382

Semenov, S., Kolomiitsev, O., Hulevych, M., Mazurek, P., & Chernyk, O. (2025). An Intelligent Method for C++ Test Case Synthesis Based on a Q-Learning Agent. *Applied Sciences*, *15*(15), 1-31. https://doi.org/10.3390/app15158596

Silva, C. A., Ramos, F. N., de Moraes, R. V., & Santos, E. L. (2024). ChatGPT: Challenges and Benefits in Software Programming for Higher Education. *Sustainability*, *16*(3), 1-23. https://doi.org/10.3390/su16031245

Soomro, S. A., Nanjappan, V., Casakin, H., & Georgiev, G. V. (2024). Fostering Technical Skills and Creativity in the Digital Fabrication Spaces: An Open-Ended Prototyping Approach. *International Journal of Technology and Design Education*. https://doi.org/10.1007/s10798-024-09940-3

Taeb, M., Chi, H., & Bernadin, S. (2024). Assessing the Effectiveness and Security Implications of AI Code Generators. *Journal of The Colloquium for Information Systems Security Education*, *11*(1), 1-6. https://doi.org/10.53735/cisse.v11i1.180

Velicea, F. P., Ghimbășan, A. E., Filip, R. C., Danciu, G. M., & Kertész, C. Z. (2025). Programming Examination Platform: Generative AI-Driven Evaluation Tool for Computer Programming Classes. *Futureproofing Engineering Education for Global Responsibility*. https://doi.org/10.1007/978-3-031-83520-9_33

Wang, A. Y., Chen, Y., Chung, J. J. Y., Brooks, C., & Oney, S. (2021). PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises. *Proceedings of the ACM on Human-Computer Interaction*. https://doi.org/10.1145/3479559

Wilson, S., & Nishomoto, M. (2023). Assessing Learning of Computer Programing Skills in the Age of Generative Artificial Intelligence. *Journal of Biomechanical Engineering*, *146*(5), 1-6. https://doi.org/10.1115/1.4064364

Wilson, S. E., & Nishimoto, M. (2024). Assessing Learning of Computer Programing Skills in the Age of Generative Artificial Intelligence. *Journal of Biomechanical Engineering*, *146*(5), 1-6. https://doi.org/10.1115/1.4064364

Xiao, J., Bozkurt, A., Nichols, M., Pazurek, A., Stracke, C. M., Bai, J. Y. H.,...Themeli, C. (2025). Venturing into the Unknown: Critical Insights into Grey Areas and Pioneering Future Directions in Educational Generative AI Research. *TechTrends*, 1-16. https://doi.org/10.1007/s11528-025-01060-6

Xu, Z., & Sheng, V. S. (2024). Detecting AI-Generated Code Assignments Using Perplexity of Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(21), 23155-23162. https://doi.org/10.1609/aaai.v38i21.30361

Xue, Y., Chen, H., Bai, G. R., Tairas, R., & Huang, Y. (2024). Does ChatGPT Help With Introductory Programming?An Experiment of Students Using ChatGPT in CS1. *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. https://doi.org/10.1145/3639474.3640076

Yang, S., Zhao, H., Xu, Y., Brennan, K., & Schneider, B. (2024). Debugging with an AI Tutor: Investigating Novice Help-seeking Behaviors and Perceived Learning. *Proceedings of the 2024 ACM Conference on International Computing Education Research*. https://doi.org/10.1145/3632620.3671092

Yilmaz, R., & Karaoglan Yilmaz, F. G. (2023). Augmented Intelligence in Programming Learning: Examining Student Views on the Use of ChatGPT for Programming Learning. *Computers in Human Behavior: Artificial Humans*, *1*(2), 1-7. https://doi.org/10.1016/j.chbah.2023.100005

Zhang, W., Guan, Y., & Hu, Z. (2024). The Efficacy of Project-Based Learning in Enhancing Computational Thinking Among Students: A Meta-Analysis of 31 Experiments and Quasi-Experiments. *Education and Information Technologies*, *29*(11), 14513-14545. https://doi.org/10.1007/s10639-023-12392-2

KEY TERMS AND DEFINITIONS

Artificial Intelligence: A broad field of computer science focused on creating systems capable of performing tasks that normally require human intelligence, such as reasoning, learning, decision-making, perception, and language understanding.

AI-Assisted Coding: The use of generative AI tools (e.g., ChatGPT, GitHub Copilot) to support programming tasks such as code generation, debugging, refactoring, and documentation, which can improve productivity but also influence learning and assessment validity.

Assessment Integrity: The assurance that the outcomes of an evaluation genuinely reflect a learner's own knowledge, effort, and skill, free from unauthorized or unacknowledged assistance.

Ethical Programming: An evaluative lens that considers the societal, legal, and moral implications of software design that emphasize issues such as algorithmic fairness, data privacy, bias mitigation, and responsible AI use.

Generative AI: A class of artificial intelligence models (e.g., LLMs like ChatGPT or Codex) capable of producing code, text, or other artifacts based on learned patterns.

Human-AI Collaboration: A pedagogical and professional paradigm where learners or developers work alongside AI systems to enhance productivity, problem-solving, and creativity, while maintaining human oversight and critical judgment.

Programming Education: A structured teaching-and-learning process that develops learners' ability to understand computational concepts and apply programming skills such as algorithm design, debugging, code comprehension, and software development practices.

RELATED RESEARCH

Journal Article

Teaching and Learning Computer Programming Using ChatGPT: A Rapid Review of Literature Amid the Rise of Generative AI Technologies

Garcia, M. B. (2025). *Education and Information Technologies*, 30, 16721–16745.
https://manuelgarcia.info/publication/chatgpt-programming-education

Journal Article

Profiling the Skill Mastery of Introductory Programming Students: A Cognitive Diagnostic Modeling Approach

Garcia, M. B. (2024). *Education and Information Technologies*, 30, 16721–16745.
https://manuelgarcia.info/publication/programming-skill-mastery-profile

Conference Paper

Exploring Student Preference Between AI-Powered ChatGPT and Human-Curated Stack Overflow in Resolving Programming Problems and Queries

Garcia, M. B., Revano Jr., T. F., Maaliw III, R. R., Lagrazon, P. G. G., Valderama, A. M. C., Happonen, A., Qureshi, B., & Yilmaz, R. (2023). In *2023 IEEE 15th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)* (pp. 1-6). IEEE.
https://manuelgarcia.info/publication/chatgpt-stackoverflow-programming

LET'S COLLABORATE!

If you are looking for research collaborators, please do not hesitate to contact me at mbgarcia@feutech.edu.ph.

ABOUT THE CORRESPONDING AUTHOR:

Manuel B. Garcia is a professor of information technology and the founding director of the Educational Innovation and Technology Hub (EdITH) at FEU Institute of Technology, Manila, Philippines. His interdisciplinary research interest includes topics that, individually or collectively, cover the disciplines of education and information technology. He is a licensed professional teacher and a proud member of the National Research Council of the Philippines – an attached agency to the country's Department of Science and Technology (DOST-NRCP).