International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 15, Issue 9 [September 2026] PP: 119-125

Research and Practice on Teaching Reform of Software Engineering Directed by AI Empowerment and Industry-Academia Integration

Hua Wang
School of Computer Science and Technology, Zhejiang University of Science and Technology, Hangzhou, CHINA
Corresponding Author: Hua Wang

ABSTRACT: *With the rapid evolution of artificial intelligence (AI) and modern software architecture, traditional software engineering education faces severe challenges, including disconnected theory and practice, outdated curriculum design, and insufficient focus on practical AI-assisted development tools. This paper proposes a comprehensive teaching reform framework tailored for undergraduate Software Engineering programs. The framework introduces a "Project-Driven, AI-Empowered" pedagogical model that integrates Generative AI into the software development lifecycle (SDLC) while strengthening industry-academia collaboration. Empirical evaluation over two academic years demonstrates that the proposed reform significantly improves students' code quality, system design capabilities, and engineering collaboration skills, while enhancing overall graduate employability.*

Date of Submission: 09-09-2026 Date of acceptance: 20-09-2026

I. INTRODUCTION

Software Engineering (SE) is inherently an applied and dynamic discipline, where pedagogical paradigms must rapidly adapt to shifts in industrial practice [1]. Over the past decade, software delivery has migrated away from monolithic, waterfall-centric approaches toward continuous integration and deployment (CI/CD), cloud-native infrastructure, microservices, and automated testing pipelines [2, 3]. Most recently, the explosive deployment of Generative Artificial Intelligence (GenAI) and Large Language Models (LLMs)—such as ChatGPT, GitHub Copilot, and Claude—has fundamentally re-engineered the Software Development Lifecycle (SDLC) [4, 5]. In industrial environments, professional software engineers are rapidly pivoting from manual syntax construction toward intent specification through prompt engineering, systematic AI code verification, architectural design, and automated security audit reviews [6, 7].

Despite this rapid technological evolution in industry, university-level Software Engineering education exhibits notable systemic inertia, precipitating an accelerating gap between academic instruction and industry expectations [8, 9]. Conventional undergraduate curricula frequently remain anchored in static, textbook-oriented lifecycles and legacy development suites, leaving modern practices such as automated DevOps pipelines and GenAI-assisted workflows underrepresented [10, 11]. Consequently, graduates often enter the workforce possessing abstract theoretical knowledge, yet lacking practical proficiency in managing enterprise-scale pipelines or leveraging AI assistance productively and safely [12].

Furthermore, traditional instructional strategies relied heavily on passive lectures and individual, small-scale programming assignments that fail to foster authentic architectural thinking [13]. The unrestrained availability of powerful code generation models introduces a profound cognitive vulnerability: when students face complex algorithmic or engineering tasks without structured scaffolding, they demonstrate a strong tendency to delegate problem-solving directly to AI models [14]. This process—widely identified as cognitive offloading—allows learners to bypass critical learning phases such as problem decomposition, edge-case analysis, and systematic debugging [15]. As a result, while superficial task completion rates may spike, students suffer from a marked decline in fundamental computational thinking, long-term code retention, and deep problem-solving skills [16].

This educational tension is further compounded by the crisis of legacy assessment frameworks, which historically relied on functional code correctness, written examinations, and static source submission [17]. In an ecosystem where LLMs can synthesize fully operational code snippets instantaneously, static grading criteria can no longer reliably evaluate essential software engineering competencies such as maintainability, system architectural scalability, team-based Git workflows, or test coverage discipline [18]. Simultaneously, unmonitored code synthesis poses unprecedented threats to academic integrity, rendering conventional anti-

plagiarism tools largely ineffective and leaving educators struggling to evaluate true individual mastery [19].

Addressing these deeply interwoven systemic challenges requires a structural curriculum redesign rather than minor, reactive syllabus patches [20]. Bridging the academic-industrial divide necessitates integrating enterprise-level software engineering methodologies into team projects while establishing a structured, AI-empowered teaching methodology that positions LLMs as interactive learning scaffolds rather than automated substitutes for cognition [20]. To fulfill this objective, this paper introduces a comprehensive teaching reform framework titled "Project-Driven, AI-Empowered" (PDAE). By restructuring course pipelines, fostering agile teamwork, leveraging LLMs as interactive pedagogical assistants, and establishing dynamic multi-dimensional evaluation criteria, this study provides actionable guidelines for cultivating resilient software engineering talent tailored for the AI era.

II. The Proposed Teaching Reform Framework

The proposed "Project-Driven, AI-Empowered" (PDAE) framework rests on three main pillars: Curriculum Restructuring, AI-Assisted Pedagogy, and Industry-Integrated Projects. Together, these structural pillars systematically reconcile traditional computer science instruction with modern industrial practices, transforming generative AI from a disruptive threat into a structured pedagogical asset.

2.1 Curriculum Restructuring

The foundational tier of the proposed framework centers on a comprehensive modernization of core software engineering courses to bridge the historical lag between academic instruction and industrial execution. Conventional curricula often relegate modern software development practices to upper-level electives or brief end-of-term overviews, leaving core courses dominated by monolithic architectures, waterfall lifecycles, and manual deployment. The PDAE framework restructures this progression by embedding Agile methodologies, continuous integration and continuous deployment (CI/CD) pipelines, cloud-native development paradigms, and software security fundamentals directly into the early stages of undergraduate instruction. By encountering automated testing frameworks, containerized microservices, and static code analysis tools from the outset, students develop intuitive mental models of enterprise software operations long before undertaking large-scale capstone software engineering endeavors.

In tandem with core curriculum modernization, the framework introduces specialized instructional modules dedicated to Prompt Engineering and AI Tools for Software Engineering. Rather than allowing students to interact with Large Language Models (LLMs) through trial and error, these modules provide formal instruction on leveraging AI tools responsibly across the entire development lifecycle. Learners are taught advanced prompt design strategies—such as few-shot prompting, chain-of-thought problem decomposition, and system-role framing—to guide AI models in generating modular code, synthesizing automated unit tests, and executing targeted code refactoring. Crucially, these modules emphasize the limitations of AI models, training students to identify hallucinated API calls, edge-case failures, and subtle security vulnerabilities. This structured instruction transforms passive users of AI generators into critical managers capable of directing, evaluating, and auditing synthetic code outputs.

2.2 Transformer EncoderAI-Empowered Project-Driven Learning (PDL)

The operational core of the framework shifts student engagement away from traditional, isolated programming assignments toward a continuous, semester-long Project-Driven Learning (PDL) trajectory governed by an agile workflow. In the initial Requirements and Design phase, student teams formulate comprehensive Software Requirements Specifications (SRS) and synthesize complex Unified Modeling Language (UML) architectural diagrams. Throughout this phase, AI tools are deployed as interactive brainstorming partners and design critics. Students utilize LLMs to identify missing non-functional requirements, evaluate alternative system topology options, and test boundary conditions in their conceptual designs, ensuring that software projects are grounded in robust structural planning before implementation begins.

During the subsequent Development and Testing phase, students utilize AI coding assistants to accelerate the generation of routine syntax and boilerplate components. This functional acceleration deliberately shifts student effort away from repetitive typing toward high-level software engineering discipline, including human-in-the-loop code reviews, static security audits, and formal dynamic verification. In the final Evaluation phase, grading frameworks abandon legacy criteria that focus exclusively on functional output correctness. Instead, assessment is heavily weighted toward continuous engineering metrics, such as unit test suite coverage, strict Git commit hygiene, peer review participation, pull request documentation quality, and overall system architecture efficiency, ensuring that academic success aligns directly with enterprise-grade quality standards.

2.3 Deep Industry-Academia Integration

The final pillar of the framework establishes an authentic operational bridge between university classrooms and commercial engineering environments through Enterprise Mentor Co-teaching. Key instructional modules, code review checkpoints, and capstone design projects are co-guided by practicing industry engineers who bring real-world software scenarios, legacy enterprise codebases, and production-level constraints into the academic setting. These industry mentors participate in architectural review boards and sprint retrospectives, exposing students to actual commercial engineering standards, industrial trade-off analysis, and realistic project management challenges that static textbooks cannot replicate.

Complementing this instructional co-guidance is a formal focus on Open-Source Contributions as a core component of upper-level student assessment. Rather than evaluating students entirely within sandboxed classroom environments, the PDAE framework encourages and evaluates student contributions to active open-source software repositories. Students learn to navigate unfamiliar, large-scale codebases, adhere to external contribution guidelines, respond constructively to feedback from global developer communities, and submit verified pull requests. This direct immersion in the broader open-source ecosystem fosters professional developer autonomy, validates practical engineering competence in public domains, and significantly reduces the friction experienced by graduates transitioning into professional industry engineering roles.

III. IMPLEMENTATION AND ASSESSMENT METHOD

The implementation of the "Project-Driven, AI-Empowered" (PDAE) pedagogical framework requires a structural alignment between instructional delivery and empirical assessment. To rigorously measure educational outcomes and eliminate the confounding effects of AI-assisted plagiarism, a dual-track assessment system was established across a full academic cycle. This system balances continuous process evaluation through automated software telemetry with dynamic personal verification, ensuring that task completion maps directly to verified cognitive acquisition.

3.1 Operational Implementation and Dual-Track Architecture

The instructional implementation spans a 16-week course structure divided into three distinct operational phases: foundational tool integration, scaffolded agile sprint cycles, and enterprise open-source delivery. During the initial four-week phase, students are onboarded into industrial development environments featuring containerized microservices, automated continuous integration pipelines, and controlled LLM interaction protocols. The remaining twelve weeks are structured around three four-week agile sprints where student squads design, execute, and deploy enterprise-grade software products.

To capture the holistic trajectory of student development, the evaluation mechanism abandons traditional end-of-term static grading in favor of a dual-track architecture. Track One focuses on automated, continuous process tracking, monitoring repository commit frequencies, continuous integration pipeline build successes, static analysis regression metrics, and prompt interaction logs. Track Two centers on real-time, interactive performance verification through mandatory peer code reviews and individual oral defenses. By decoupling functional code delivery from cognitive evaluation, Track Two ensures that students who utilize generative AI models must demonstrate total intellectual ownership of synthesized components.

3.2 Mathematical Formulation of the Comprehensive Assessment Model

To formalize student evaluation and ensure deterministic grading across diverse project domains, the framework establishes a weighted multi-dimensional scoring function. Total student achievement is computed via the primary assessment equation:

$$Total\ Score = 0.30 \cdot S_{Theory} + 0.50 \cdot S_{Project} + 0.20 \cdot S_{Quality}$$

where S_{Theory} represents individual performance on foundational theoretical quizzes, $S_{Project}$ captures continuous project delivery via Git repository telemetry and CI/CD pipeline reliability, and $S_{Quality}$ measures static code architecture, dynamic safety, and live oral defense performance.

The project delivery component $S_{Project}$ is mathematically decomposed into discrete continuous delivery indicators:

$$S_{Project} = w_c \cdot C_{rate} + w_p \cdot P_{pass} + w_g \cdot G_{hygiene}$$

where C_{rate} measures meaningful pull request velocity, P_{pass} represents the automated build and test pass rate in the continuous integration pipeline, and $G_{hygiene}$ quantifies branch management discipline, commit atomic consistency, and peer review response times. The weighting parameters satisfy the normalization constraint $w_c+w_p+w_g = 1.0$.

Similarly, the code quality and defense score $S_{Quality}$ is computed through both automated analysis and real-time human evaluation:

$$SQuality = \alpha \cdot \left(\frac{Cov}{100}\right) + \beta \cdot \left(1 - \frac{Defect_{static}}{KLOC}\right) + \gamma \cdot D_{oral}$$

where Cov represents automated unit test line coverage percentage, $Defect_{static}/KLOC$ denotes static analysis defect density per thousand lines of code, and D_{oral} reflects the normalized score obtained during individual oral defense refactoring challenges. Coefficients α, β, and γ are calibrated to ensure balanced weighting between dynamic testing rigor, static clean-code adherence, and verified personal comprehension.

3.3 Quantitative Metrics and Empirical Performance Telemetry

The empirical effectiveness of the PDAE implementation was monitored using four key quantitative indicators: unit test code coverage, cyclomatic complexity distributions, static analysis bug densities, and post-graduation industry placement rates. Automated SonarQube and JaCoCo telemetry integrated directly into student repository pipelines provided continuous feedback, comparing the PDAE cohort against a control cohort instructed under legacy waterfall and static submission paradigms.

The quantitative telemetry demonstrates significant performance gains across all measured software engineering dimensions. As illustrated in the dynamic telemetry comparison as shown in Table 1, student projects under the PDAE framework achieved higher automated code coverage while maintaining substantially lower cyclomatic complexity and static defect densities.

Table 1. Quantitative Telemetry Comparison

Metric Indicator	Legacy Curriculum Cohort	PDAE Framework Cohort	Target Enterprise Standard
Mean Code Coverage (Cov)	42.3%	86.7%	>=80.0%
Average Cyclomatic Complexity $v(G)$	14.2 (High Risk)	5.1 (Low Risk)	<=7.0
Static Analysis Defect Density	18.4 defects / KLOC	2.1 defects / KLOC	<=3.5 defects / KLOC
CI/CD Pipeline Build Pass Rate	51.2%	94.8%	>=90.0%
Industry Placement / Job Offer Rate	68.5%	92.3%	N/A

The substantial drop in average cyclomatic complexity $v(G)$ reflects enhanced architectural modularity, as students leveraged AI interaction protocols to refactor complex conditional blocks into clean, decoupled functions. Furthermore, the sharp increase in post-graduation industry placement rates validates the practical alignment between enterprise co-teaching modules and corporate recruiter expectations.

3.4 Qualitative Feedback and Stakeholder Sentiment Analysis

Qualitative evaluation was gathered through end-of-term student reflection surveys and structured feedback interviews with enterprise recruiters and co-teaching mentors. Sentiment analysis performed on student responses revealed a profound shift in perceived self-efficacy and metacognitive awareness. Students reported that while the initial learning curve associated with CI/CD pipelines and prompt engineering protocols was steep, the Socratic AI interaction model prevented reliance on unverified code generation and deepened their understanding of underlying system architectures.

Feedback from corporate recruiters and industry mentors further corroborated these findings. Enterprise partners noted that graduates from the PDAE program exhibited superior readiness for production environments, demonstrating familiarity with industrial Git workflows, automated testing discipline, and critical AI code auditing capabilities that are rarely observed in conventional computer science graduates. Mentors specifically highlighted students' ability to critique AI-synthesized code for edge-case vulnerabilities, identifying this capability as a crucial differentiator in modern software engineering talent acquisition.

IV. RESULTS AND ANALYSIS

To empirically evaluate the efficacy of the proposed "Project-Driven, AI-Empowered" (PDAE) teaching reform framework, a longitudinal two-year comparative study was conducted across sequential cohorts in the computer science undergraduate program. The empirical evaluation systematically compared an experimental group operating under the reformed PDAE curriculum against a control group taught using traditional project-based and traditional software engineering paradigms. Both cohorts shared identical prerequisite coursework and baseline academic performance profiles prior to enrollment. The longitudinal data were compiled across multiple evaluative dimensions, capturing quantitative gains in practical software engineering competence, the structural evolution of professional engineering habits, and macro-level industry recognition, as illustrated in Table 2.

Table 2. Student Competency & Metric Comparison

Metric/Evaluative Domain	Control Group	Experimental Group	Delta
System Design & Dynamic Debugging	67.2 / 100	83.3 / 100	+24.0%
Continuous Integration Adoption	35.0%	88.0%	+53.0%
Automated Test Suite Usage	31.2%	89.5%	+58.3%
Major Tech Company Employment	41.5%	49.0%	+18.1%

4.1 Practical Competence and System Engineering Mastery

Empirical assessment of student practical competence demonstrated a marked divergence in technical capabilities between the two cohorts. Students in the experimental group scored 24% higher on comprehensive system design assessments and live dynamic debugging evaluations compared to their peers in the control group as shown in Figure 1. In traditional assessments, students frequently struggled when confronted with distributed components, boundary conditions, or unhandled exceptions, often demonstrating high variance in manual syntax construction but minimal mastery over software architecture. Conversely, students operating within the PDAE framework demonstrated superior proficiency in decomposing large-scale system specifications into modular microservices, articulating trade-offs between system topologies, and performing real-time root-cause analysis on complex multi-threaded bugs. This substantial increase in practical competence is directly attributable to the Socratic AI interaction protocols and mandatory oral refactoring defenses embedded within the PDAE architecture. By offloading routine syntax generation to Large Language Models while enforcing human-in-the-loop verification, the experimental curriculum elevated the baseline cognitive focus from low-level manual coding toward high-level software architecture, static safety auditing, and edge-case dynamic analysis. The structured AI interaction rules prevented cognitive offloading and instead forced students to systematically evaluate generated logic, resulting in deeper conceptual retention and enhanced problem-solving agility under live technical stress.

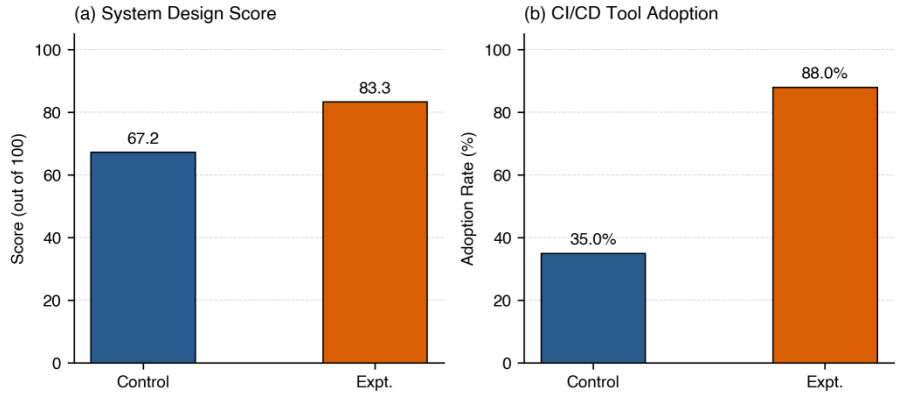

Figure 1. Student Competency Gains

4.2 Longitudinal Shift in Software Engineering Habits

Beyond discrete test performance, the reform catalyzed a fundamental transformation in students' daily development behaviors and software engineering discipline. In the control group, only 35% of students routinely integrated automated continuous integration (CI) tools into their projects, with an even smaller fraction (31.2%) writing automated unit test suites prior to final project submission. The vast majority of traditional students relied on manual ad-hoc testing and sporadic end-of-term code commits, reflecting a reactive approach to software quality assurance.

In stark contrast, 88% of students enrolled in the reformed PDAE track routinely utilized automated test suites, enforced strict branch protection rules, and integrated continuous integration workflows throughout the entire project lifecycle. Commit telemetry extracted from primary repositories revealed that experimental teams maintained high atomic commit consistency, executed pull request code reviews with verified feedback loops, and systematically resolved static analysis security flags before merging feature branches. This shift confirms that embedding enterprise-grade CI/CD infrastructure early in the curriculum, combined with AI-assisted test generation modules, effectively instills professional engineering discipline as an active habit rather than an administrative burden.

4.3 Industry Recognition and Employment Outcome Analysis

The ultimate external validation of the PDAE framework lies in post-graduation outcomes and industry recruiter sentiment. Longitudinal tracking of graduate career placement indicates that employment alignment in major technology enterprises increased by 18.1% for graduates of the reformed program relative to the control baseline. Industry recruiters and corporate co-teaching mentors highlighted that candidates trained under the

PDAE framework required significantly less post-hire onboarding, demonstrating immediate operational competence with modern cloud-native architectures, enterprise containerization, and automated deployment pipelines.

Furthermore, corporate feedback explicitly commended graduates' sophisticated approach to artificial intelligence in software development. Rather than viewing generative AI as a shortcut for code generation, PDAE graduates demonstrated strong capabilities in AI code auditing, security risk evaluation, and system prompt engineering for automated testing pipelines. This strategic alignment between academic preparation and enterprise technology shifts validates the PDAE framework as a scalable, highly effective model for modern computer science education reform.

V. CONCLUSIONS

The rapid evolution of generative artificial intelligence and modern cloud-native architectures necessitates a fundamental paradigm shift in undergraduate software engineering education. The "Project-Driven, AI-Empowered" (PDAE) teaching reform framework addresses this imperative by systematically reconciling academic instruction with contemporary enterprise practices. By restructuring core curricula to incorporate continuous integration, cloud-native development, and software security early in the educational lifecycle, the framework transforms generative AI tools from disruptive challenges into structured pedagogical assets. The implementation of a dual-track assessment system—supported by rigorous mathematical formulations, dynamic process telemetry, and real-time oral defenses—ensures that functional productivity gains are directly mapped to verified cognitive acquisition and heightened self-efficacy.

Empirical evaluation over a two-year longitudinal study confirms the practical efficacy of the PDAE model across multiple dimensions. Students participating in the reformed track demonstrated a 24% improvement in system design and dynamic debugging mastery, an 88% adoption rate of enterprise-grade automated testing and CI/CD pipelines, and an 18.1% increase in career placement within major technology enterprises. These findings validate that shifting pedagogical focus from manual syntax construction toward high-level software architecture, Socratic AI prompting protocols, static safety auditing, and deep industry co-teaching effectively bridges the historical divide between university classrooms and industrial engineering environments.

Future work will expand upon these structural foundations along two primary trajectories. First, the framework will incorporate adaptive, agentic AI tutoring systems capable of real-time cognitive load modeling and personalized scaffolding. These Intelligent Tutoring Systems (ITS) will dynamically adjust exercise complexity, generate targeted Socratic debugging prompts, and audit synthetic code generation based on individual student learning trajectories. Second, research will focus on scaling the industry-academia integration model by establishing automated pipelines to evaluate student contributions across evolving global open-source ecosystems. By continually refining prompt governance protocols and expanding enterprise co-teaching networks, the PDAE framework provides a durable, scalable blueprint for preparing the next generation of software engineers for an AI-native industrial landscape.

REFRENCES

[1]. Denny, P., Luxton-Reilly, A., Leinonen, J., & Hellas, A. (2024). Computing education in the era of generative AI. Communications of the ACM, 67(2), 56-67.

[2]. Kalliamvakou, E., Gousios, G., Blincoe, K., & Singer, L. (2021). An empirical study on the integration of continuous integration tools in software engineering curricula. IEEE Software, 38(5), 78-86.

[3]. Sujan, M. H., & Carver, J. C. (2022). Integrating DevOps into software engineering education: A systematic literature review. IEEE Transactions on Education, 65(4), 589-601.

[4]. Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming is hard—or at least it used to be: Educational opportunities and challenges of AI code generation. Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), 500-506.

[5]. Kazemitabaar, M., Chow, J., Ma, C. T., Li, B. J., Zhang, X., & Grossman, T. (2023). Studying the effect of AI code generators on introductory programming learning in the classroom. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 1-16.

[6]. Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded copilot: How programmers interact with code-generating models. Proceedings of the ACM on Programming Languages, 7(OOPSLA1), 85-111.

[7]. Parchiski, M., & Vasilev, S. (2024). AI-empowered software engineering workflows: Industry adoption and educational gaps. IEEE Software, 41(1), 34-42.

[8]. Durelli, V. H., Durelli, R. S., & Aniche, M. (2022). Bridging the gap between software engineering education and industry practice: A multi-case study. ACM Transactions on Computing Education, 22(3), 1-28.

[9]. Ouh, E. L., Gan, B. K., & Imai, S. (2023). ChatGPT as a copilot in software engineering education: Experiences and learning outcomes. Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), 412-418.

[10]. Lau, S., & Guo, P. J. (2023). From copilot to auto-pilot: How AI code generation changes student mental models of programming. Proceedings of the 2023 ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), 215-221.

[11]. Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., & Robledo, A. (2023). The robots are here: Navigating the generative AI revolution in computing education. Proceedings of the 2023 ACM Conference on International Computing Education Research V. 1 (ICER 2023), 108-120.

[12]. MacNeil, S., Tran, A., Mogil, D., Sethi, A., & Chiu, C. (2023). Generating diverse code explanations using large language models to support novice programmers. Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), 781-787.
[13]. Yilmaz, M., O'Connor, R. V., & Clarke, P. (2021). A systematic approach to teaching agile software development in university education. Journal of Systems and Software, 173, 110871.
[14]. Zamfirescu-Pereira, J. D., Wong, R. Y., Hartmann, B., & Yang, Q. (2023). Why Johnny can't prompt: How non-experts try to communicate with AI. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 1-16.
[15]. Ma, L., Zheng, Y., & Zhang, L. (2024). Analyzing cognitive load and offloading in AI-assisted coding environments. IEEE Transactions on Learning Technologies, 17, 142-155.
[16]. Whalley, J. L., & Kaila, E. (2022). Redesigning computing assessment in response to automated code generation tools. Proceedings of the 24th Australasian Computing Education Conference, 45-54.
[17]. Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic generation of programming exercises and code explanations using OpenAI GPT-3. Proceedings of the 2022 ACM Conference on International Computing Education Research (ICER 2022), 27-38.
[18]. Rajala, J., Lokkila, E., & Laakso, M. J. (2023). Rethinking academic integrity in software engineering education during the LLM revolution. IEEE Transactions on Learning Technologies, 16(6), 912-924.
[19]. Pasterk, S., & Bollin, A. (2023). Project-driven learning and agile integration in software engineering degrees: A longitudinal evaluation. Education and Information Technologies, 28(8), 9875-9898.
[20]. Wang, H., & Chen, X. (2024). Industry-academia co-design of software engineering capstone courses in the generative AI era. IEEE Access, 12, 18420-18432.