2026 IEEE 50th Annual Computers, Software, and Applications Conference (COMPSAC) | 979-8-3315-4497-3/26/$31.00 ©2026 IEEE | DOI: 10.1109/COMPSAC69091.2026.00213

Where Do Expectations Diverge? An Empirical Analysis of Software Engineering Graduate Preparedness in Industry

Yicheng Sun†, Jacky Keung†, Hi Kuen Yu†, Yihan Liao†, Yishu Li‡, Xiaoxue Ma‡,*
†Department of Computer Science, City University of Hong Kong, Hong Kong, China
‡Department of Electronic Engineering and Computer Science, Hong Kong Metropolitan University, Hong Kong, China
{yicsun2-c, hikuenyu2-c, yihanliao4-c}@my.cityu.edu.hk, jacky.keung@cityu.edu.hk, sliy@hkmu.edu.hk
*Corresponding author: kxma@hkmu.edu.hk

***Abstract*—This study investigates the preparedness of software engineering graduates for professional roles in the evolving software industry, focusing on the alignment between academic curricula and industry expectations. Using a mixed-method exploratory design, data were collected from 21 graduates and 5 senior engineers through questionnaires, in-depth interviews, and course assessment reports. Advanced analytical techniques, including thematic analysis, sentiment classification, and topic modeling, were employed to evaluate the gaps in theoretical knowledge, practical skills, and industry-aligned competencies. Findings reveal a persistent mismatch between graduates' self-perceived readiness and employer expectations, particularly in hands-on experience, familiarity with industry-standard tools, and collaborative workflows. Additionally, online learning and reliance on Large Language Models (LLMs) emerged as contributing factors, enhancing coding efficiency but raising concerns about over-reliance and diminished problem-solving depth. This study offers actionable recommendations to bridge educational gaps and improve the alignment between software engineering curricula and industry demands, providing insights valuable to educators, researchers, and practitioners.**

***Index Terms*—Software engineering, Graduate Industry readiness, Curriculum-industry alignment, Online learning**

I. Introduction

In recent years, software engineering, as an essential sub-discipline of computer science, has witnessed a remarkable surge in popularity among university students, driven primarily by rapid technological advancements and an expanding digital economy [1], [2]. The attractiveness of software engineering-related education continues to rise globally [3]. Taking China as an illustrative example, according to statistics released by China's Ministry of Education, the number of software engineering graduates in 2024 increased by approximately 35% compared to 2020. This significant growth highlights both the growing enthusiasm among students for software engineering programs and the increasing demand for highly skilled professionals in industry and academia alike [4].

Despite the promising development and enthusiasm around software engineering education, several challenges have emerged concurrently. The first issue arises from the unprecedented shift toward online education triggered by the global COVID-19 pandemic [5]–[7]. Remote teaching and home-based online learning quickly became prevalent methods to sustain educational continuity [8]–[10]. However, software engineering, unlike many other disciplines, requires extensive practical coding, collaborative programming exercises, and immediate instructor feedback, making it particularly sensitive to the delivery medium and teaching methods employed. The core reason for such skepticism lies in the discipline's inherent characteristics: software engineering demands substantial hands-on practice, immediate debugging interactions, continuous peer-to-peer communication, and effective mentoring [11]. Thus, questions regarding whether remote learning can fully replicate the experiential learning necessary for acquiring critical software engineering competencies remain largely unanswered.

Furthermore, we surveyed software engineering graduates from seven universities in China in 2024 and found that 12.5% of them pursued careers outside the software engineering field. Of the remaining graduates, over 80% began their careers in front-end development, back-end development, or software testing. This distribution highlights the strong alignment between academic preparation and industry roles within the software development sector. However, it remains unclear whether current software engineering curricula and instructional methodologies adequately address and align with industry demands [12], [13]. Specifically, there is an ongoing debate regarding how effectively universities equip software engineering graduates with the practical skills and relevant knowledge required to excel immediately upon entering the workforce. In addition to concerns surrounding curriculum-industry alignment, another critical factor has recently emerged, driven by the proliferation and widespread adoption of Large Language Models (LLMs) [14]–[17]. Increasingly, software engineering students have begun extensively utilizing these AI tools to assist in academic tasks, including programming assignments, debugging processes, and software design activities [18], [19]. This phenomenon raises essential questions regarding the educational implications of students heavily relying on artificial intelligence assistance throughout their academic journey.

Given these identified gaps and concerns, this exploratory study aims to critically assess the preparedness of software

2836-3795/26/$31.00 ©2026 IEEE
DOI 10.1109/COMPSAC69091.2026.00213

engineering graduates for industry demands, explicitly focusing on examining potential discrepancies between academic preparation and industry expectations. Graduates were specifically chosen for this investigation because they have recently completed structured professional training, possess clear career objectives, and currently occupy a unique transitional stage between academic and professional life. This transitional phase provides valuable insights, as graduates can effectively reflect on the relevance and adequacy of their educational experiences while simultaneously confronting real-world professional demands. Through empirical investigation, this research intends to provide insights into whether existing educational practices adequately prepare software engineering graduates to effectively handle professional roles in the industry. Furthermore, this research will analyze the impacts of online education and the increasing use of AI-based tools on graduates' competencies and readiness, ultimately proposing actionable recommendations aimed at enhancing the overall quality and effectiveness of software engineering education. Based on these focal concerns, this study seeks to answer the following key research questions:

- ***RQ1:*** *What specific gaps exist between the skills and competencies gained through current software engineering curricula and the actual demands encountered by graduates upon entering the software industry?*
- ***RQ2:*** *How has the widespread adoption of online learning, prompted by the COVID-19 pandemic, influenced software engineering graduates' mastery of essential practical coding skills and their readiness to meet industry expectations?*
- ***RQ3:*** *In what ways does reliance on Large Language Models (LLMs) during academic study affect software engineering graduates' coding skills, problem-solving capabilities, and overall preparedness for professional work?*
- ***RQ4:*** *What actionable recommendations can be proposed to bridge identified educational gaps and improve the preparedness of software engineering graduates for professional industry demands?*

In summary, our primary contributions can be summarized in the three points:

- We present a detailed exploration into the preparedness of software engineering graduates for industry demands by thoroughly investigating how online education and AI-assisted learning methods impact professional readiness.
- We uniquely integrate student perspectives with employer expectations, offering comprehensive insights into potential discrepancies between current software engineering curricula and actual industry requirements.
- We conduct an extensive empirical investigation to identify specific gaps and provide actionable recommendations aimed at enhancing the alignment between academic training and industry expectations.

II. Methodology

A. Participants

The study involved a total of 26 participants, comprising 21 recent software engineering graduates and 5 senior engineers with substantial industry experience. Graduates were predominantly bachelor's degree holders, while a smaller proportion had master's or doctoral degrees. Participants represented primarily China, supplemented by others from countries such as the USA, the UK, Canada, and Australia to provide diverse perspectives. The graduates, aged between 22 and 29, were selected based on their recent entry into professional roles within software engineering, thus ensuring the relevance of their educational experiences and initial industry encounters.

Additionally, five senior engineers aged between 32 and 50 participated, each with significant experience in software development and management. These senior engineers had at least two months of direct interaction with newly graduated engineers, thus providing in-depth evaluations of the graduates' industry readiness. Participants' current company sizes varied across small (S), medium (M), and large (L) firms. Specifically, small-sized (S) firms employed fewer than 50 employees, medium-sized (M) firms had between 50 and 250 employees, and large-sized (L) firms included enterprises with more than 250 employees, as well as industry-leading technology companies, such as Tencent and Alibaba. The detailed participant demographic information is summarized in Table I.

TABLE I: Overview of Interview Participants

ID	Gender	Age	Educational Attainment	Country	Participant Type	Identified Role*	Current Company Size**
G1	M	25	Master	China	Graduate	3	M
G2	M	22	Bachelor	China	Graduate	1	L
G3	M	24	Master	China	Graduate	1	L
G4	F	22	Bachelor	China	Graduate	1	S
G5	F	22	Bachelor	China	Graduate	2	M
G6	M	22	Bachelor	China	Graduate	3	M
G7	F	21	Bachelor	China	Graduate	2	S
G8	F	22	Bachelor	China	Graduate	1	L
G9	M	24	Master	USA	Graduate	3	M
G10	F	23	Bachelor	China	Graduate	2	S
G11	M	28	Ph.D.	UK	Graduate	1	M
G12	F	22	Bachelor	China	Graduate	2	L
G13	M	23	Bachelor	China	Graduate	3	L
G14	M	25	Master	Canada	Graduate	4	M
G15	F	23	Bachelor	China	Graduate	1	S
G16	M	24	Master	Australia	Graduate	2	M
G17	F	21	Bachelor	China	Graduate	3	L
G18	F	22	Bachelor	Australia	Graduate	4	S
G19	M	22	Bachelor	China	Graduate	1	S
G20	M	27	Ph.D.	Australia	Graduate	2	L
G21	F	29	Master	UK	Graduate	3	M
E1	M	44	Master	China	Senior Engineer	1	L
E2	F	41	Ph.D.	USA	Senior Engineer	2	M
E3	M	35	Master	China	Senior Engineer	3	L
E4	F	47	Master	China	Senior Engineer	1	M
E5	M	38	Master	Australia	Senior Engineer	4	S

*Identified Roles for participants: (1) Front-end Developer; (2) Back-end Developer; (3) Software Tester; (4) Others (e.g., System Analyst, DevOps Engineer, Data Analyst). **Current Company Size: S = Small-sized (<50 employees); M = Medium-sized (50-250 employees); L = Large-sized (>250 employees, including both major enterprises and industry-leading technology companies such as Tencent and Alibaba.)

B. Course Assessment Reports

To supplement the qualitative insights gathered through interviews, we collected course assessment reports from four universities offering software engineering programs. Specifically, these reports included data from two universities located in China, one in Australia, and one in the United States,

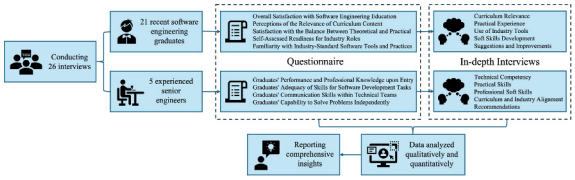

Fig. 1: The Workflow of Steps for RQ1.

allowing a diverse perspective on software engineering education practices. The international selection ensured a broad understanding of software engineering curricula and provided comparative insights across different educational contexts.

The reports covered a comprehensive range of core and elective courses central to software engineering education, such as *Software Requirements Engineering*, *Object-Oriented Programming*, *Software Testing and Quality Assurance*, and *Web Application Development*. Feedback collected from both students and faculty members focused on various elements including course design, instructional quality, assessment methods, practical assignments, laboratory exercises, and overall curriculum effectiveness. Feedback considered irrelevant or inconsistent—such as general comments unrelated to course content (e.g., campus facilities or extracurricular activities), or ambiguous and incomplete feedback—was systematically removed to ensure accuracy and relevancy of the analysis. After the filtering process, the finalized dataset comprised 42,800 individual student feedback entries and 4,978 distinct faculty comments.

C. RQ1: Differences Between Software Engineering Curricula and Industry Demands

To investigate the skills gap identified in RQ1, we conducted semi-structured interviews with 21 recent software engineering graduates and 5 senior engineers possessing substantial experience in software development and management. A semi-structured approach combining both questionnaires and in-depth interviews was adopted, as shown in Figure 1. This methodology allowed flexibility to gather nuanced insights while systematically addressing all core research questions.

The interview process consisted of two main phases: an initial questionnaire distribution followed by in-depth interviews. Prior to interviews, participants completed brief questionnaires containing primarily multiple-choice questions using a 5-point Likert scale (from 1 = strongly disagree to 5 = strongly agree). For the graduates, the questionnaire focused on aspects such as overall satisfaction with their software engineering education, perceptions regarding the relevance of curriculum content, satisfaction with the balance between theoretical and practical knowledge, self-assessed readiness for industry roles, and familiarity with industry-standard software tools and practices. For senior engineers, the questionnaire assessed their satisfaction with graduates' performance, graduates' professional knowledge level upon entry, adequacy of skills related to software development tasks, communication skills within technical teams, and their capability for independent problem-solving.

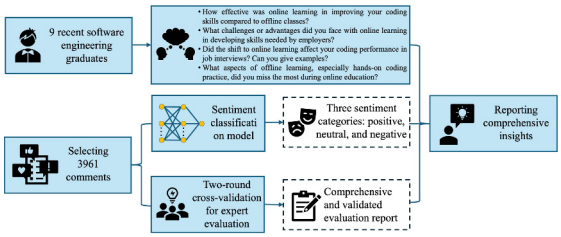

Fig. 2: The Workflow of Steps for RQ2.

Following the questionnaire, we conducted in-depth interviews lasting approximately 20 to 40 minutes. The semi-structured format allowed participants to elaborate on key points identified in their questionnaire responses, facilitating a deeper exploration of underlying issues, gaps, and expectations from educational programs. The interview questions for both graduates and senior engineers were carefully categorized into five thematic dimensions, each containing 4–6 detailed questions to facilitate a structured and comprehensive analysis.

The data collected from the questionnaires and interview transcripts were analyzed using both qualitative and quantitative methods. Quantitative analysis involved descriptive statistics derived from the Likert-scale questionnaire responses, summarizing central tendencies and variations in participants' assessments. For qualitative data gathered from interview transcripts, we conducted thematic analysis following Braun and Clarke's approach. This process included initial coding to identify recurring concepts, followed by the organization of these codes into overarching themes reflecting core issues and patterns. NVivo qualitative analysis software was utilized to ensure systematic and consistent coding, enabling a comprehensive interpretation of the insights regarding skill gaps and industry demands.

D. RQ2: Effect on Coding Skills and Industry Readiness

We involved targeted in-depth interviews with selected participants who experienced transitions from offline to online learning, as well as extensive sentiment analysis of course assessment feedback referencing learning modalities. A two-round expert validation process further enriched our findings. The detailed research procedure employed in addressing RQ2 is illustrated in Figure 2.

To explore the impact of online learning on coding skills and industry readiness, at the outset, we conducted focused in-depth interviews with nine selected participants (*G1, G3, G9, G11, G13, G14, G16, G20, G21*). These participants were specifically chosen due to their unique educational experiences, as they transitioned from face-to-face (offline) to online education during their bachelor's, master's, or doctoral studies following the onset of the COVID-19 pandemic. Among these participants, two had experiences extending into doctoral programs, ensuring diverse insights into multiple educational levels. The in-depth interviews, each lasting approximately 20–30 minutes, were semi-structured yet strictly targeted around perceptions of online versus offline education, particularly concerning the development of coding skills and preparation for employment-related technical assessments.

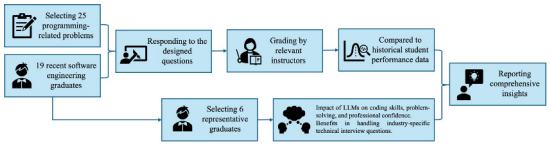

Fig. 3: The Workflow of Steps for RQ3.

Additionally, to supplement the qualitative insights from interviews, we systematically analyzed the Course Assessment Reports to extract feedback specifically referencing "online learning" or "offline learning." We employed a keyword-based text mining method using NVivo software, systematically identifying and coding all relevant comments, resulting in a total of 3,961 individual comments explicitly addressing online or offline educational experiences. To quantitatively evaluate the attitudes reflected in the extracted comments and thus better understand participants' perceptions regarding online and offline learning, we first conducted sentiment classification. Specifically, we utilized a pre-trained RoBERTa model combined with an attention-based Bi-LSTM network. This combined approach has previously demonstrated high accuracy and strong performance in sentiment classification tasks. Comments were categorized into three sentiment categories—positive, neutral, and negative—to provide clear quantitative insights into perceptions of online versus offline learning modalities.

Furthermore, to enhance reliability and provide deeper qualitative insights, we engaged five senior evaluation experts, all of whom are university professors with extensive backgrounds in software engineering education. Each expert possesses over ten years of teaching experience and has direct experience delivering software engineering courses through both online and offline modalities. The evaluation process utilized a two-round cross-validation method to ensure accuracy and thoroughness. In the first round, three professors independently reviewed the extracted comments, systematically categorizing feedback according to specific courses and identifying core themes, strengths, and limitations associated with each educational format. In the subsequent round, the remaining two professors conducted a detailed review of the initial evaluation outcomes, carefully supplementing overlooked points, clarifying ambiguities, and correcting inconsistencies to finalize a comprehensive and validated evaluation report.

E. RQ3: Impact to Coding Skills, Problem-Solving, and Preparedness

To systematically evaluate the impact of relying on Large Language Models (LLMs) on software engineering graduates' coding skills, problem-solving abilities, and professional preparedness, we conducted a structured assessment combining quantitative testing and qualitative interviews. This comprehensive process involved selecting participants based on their LLM usage habits, administering targeted programming assessments, comparing performance results with historical benchmarks, and conducting follow-up in-depth interviews. The detailed experimental design and analytical procedures employed for RQ3 are illustrated in Figure 3.

To examine the impact of reliance on LLMs on software engineering graduates' coding skills, problem-solving capabilities, and overall preparedness, we selected 19 of our original 21 graduate participants, excluding participants *G7* and *G15*. These two individuals explicitly indicated that they had never utilized any LLM tools during their software engineering coursework. The remaining 19 graduates reported frequent reliance on LLMs, such as ChatGPT, Copilot, and similar tools, to assist in various academic activities including conceptual explanations, automated code generation, debugging assistance, code optimization suggestions, software testing, and understanding complex software architecture principles.

To quantitatively evaluate the effect of LLM reliance, we designed an assessment consisting of 25 programming-related problems extracted from four representative software engineering courses: *Object-Oriented Programming*, *Software Design Patterns*, *Algorithm Analysis and Design*, and *Deep Learning and AI Application Development*. These problems covered multiple formats, including conceptual understanding questions, code completion exercises, method and algorithm design tasks, debugging challenges, and scenario-based deep learning model implementation. Questions were specifically chosen to represent typical industry-required competencies, thereby enabling precise measurement of participants' problem-solving skills and practical readiness.

The selected 19 participants completed the designed assessment under supervised conditions, and their responses were independently graded by instructors who had originally taught these courses. The obtained scores were subsequently compared with historical student performance data collected from identical or comparable assessment questions used during the academic years 2018–2020, thus offering a reliable baseline for comparison. Furthermore, to gain deeper insights into LLM usage and its effects, we conducted follow-up in-depth interviews with a subset of six graduates. These participants were specifically chosen based on their reported levels of LLM usage: two frequent users (daily or near-daily usage), two moderate users (weekly usage), and two occasional users (task-specific usage). The interviews further explored how the introduction of LLMs influenced their coding competencies, problem-solving approaches, professional confidence, and perceived benefits in handling industry-specific technical interview questions.

F. RQ4: Suggestions to Bridge Educational Gaps and Improve Graduate Preparedness

To formulate actionable recommendations aimed at bridging identified educational gaps and improving the overall preparedness of software engineering graduates, we employed a structured and multi-step analytical procedure. This involved systematically synthesizing and analyzing insights collected from interviews, questionnaire responses, course assessment reports, and expert consultations. The comprehensive research and recommendation development process is illustrated in Figure 4.

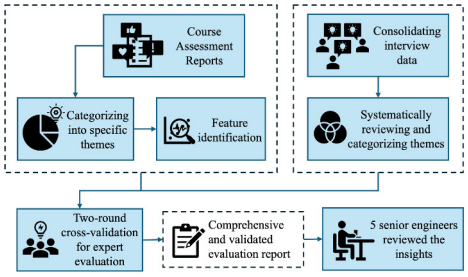

Fig. 4: The Workflow of Steps for RQ4.

Initially, we consolidated all interview data gathered in RQ1 through RQ3, systematically reviewing interview transcripts and questionnaire responses obtained from graduates and senior engineers. Using thematic analysis techniques facilitated by NVivo qualitative analysis software, we identified recurring themes and subthemes related to educational gaps, industry preparedness, online learning challenges, and the impact of LLM utilization. These thematic categorizations allowed us to clearly identify specific areas requiring improvements within current software engineering curricula and teaching methods.

Additionally, we conducted an in-depth analysis of the Course Assessment Reports, examining a total of 42,800 individual student feedback entries and 4,978 distinct faculty comments. Initially, keyword-based text mining using NVivo software was applied for preliminary categorization, systematically classifying feedback into clearly defined thematic categories. These included course levels (bachelor's, master's, doctoral), course difficulty, practical relevance, theoretical-practical integration, instructional methods, and curriculum alignment with industry needs. Such thematic categorizations enabled clear identification of specific areas requiring improvements within current software engineering curricula and instructional methods.

Following categorization, we conducted feature extraction to identify the most frequently discussed and representative characteristics within each category. Term Frequency-Inverse Document Frequency (TF-IDF) was employed to quantitatively evaluate and rank key terms based on their relevance and frequency within each thematic category. To further strengthen and verify the identified features, we employed Latent Dirichlet Allocation (LDA) topic modeling, an unsupervised probabilistic method capable of discovering underlying semantic structures and thematic clusters within textual data. Combining TF-IDF and LDA methodologies enabled us to gain a comprehensive, objective, and nuanced understanding of critical curriculum-related features consistently mentioned by students and faculty across the analyzed feedback. Finally, five senior evaluation experts independently performed qualitative validation through a two-round cross-validation process.

III. Results

A. Skills Gap Analysis (RQ1)

1) Likert-scale Results: The Likert-scale results presented in Table II reveal notable discrepancies between graduates' self-assessment and the evaluations provided by senior engineers. These differences highlight key concerns regarding the preparedness of software engineering graduates for industry demands, particularly in areas such as industry readiness and familiarity with industry-standard tools.

TABLE II: Likert Scale Scores for Graduates and Engineers.

Category	Mean Score
Graduates' Assessment	
Satisfaction with SE Education	4.33
Relevance of Curriculum Content	4.45
Balance of Theory and Practice	4.47
Industry Readiness (Self-assessed)	4.45
Familiarity with Industry Tools	3.98
Engineers' Assessment	
Satisfaction with Graduates' Performance	3.14
Graduates' Professional Knowledge Level	3.16
Adequacy of Technical Skills for Development Tasks	3.69
Communication Skills within Teams	4.02
Capability for Independent Problem-Solving	3.94

One of the most significant observations is the difference in the perception of industry readiness. Graduates rated their own preparedness for industry roles at 4.45, indicating a high level of confidence in their ability to transition into professional environments. However, engineers' evaluation of graduates' professional knowledge level was considerably lower at 3.16, and their satisfaction with graduates' overall performance was rated at 3.14. This suggests a potential overestimation by graduates regarding their industry readiness.

Another key finding is the contrast in the assessment of familiarity with industry-standard tools. Graduates rated themselves at 3.98, suggesting moderate confidence in their ability to use relevant software tools. However, engineers rated graduates' actual technical skills for development tasks at 3.69, indicating a perception that graduates may lack hands-on experience with widely used technologies.

2) Graduates' Perspective from In-Depth Interviews: The in-depth interviews with graduates revealed several recurring themes regarding their software engineering education experience. While they generally felt well-prepared in theoretical aspects, challenges arose when transitioning to industry environments, exposing gaps in their readiness.

Strong Theoretical Foundation but Limited Practical Exposure. A total of 16 graduates reported that while they had a firm grasp of theoretical concepts, they lacked sufficient hands-on experience in large-scale software projects. Many described university coursework as being too focused on academic exercises rather than industry-relevant tasks. **G12** shared, *"I did well in all my exams and coursework, but when I started my first job, I realized I had never really worked on a large-scale project. Most of our university assignments involved small, isolated coding exercises."* This concern was echoed by several other participants, who noted that university projects rarely reflected real-world software engineering challenges. While assignments typically focused on implementing small algorithms or components, they seldom required students to engage in large-scale system design, debugging, or software maintenance. As a result, many graduates struggled when first exposed to industry environments, where they needed to

understand legacy codebases and work with complex, multi-module applications.

Interestingly, 3 graduates did not perceive this as a major issue, arguing that university education should prioritize foundational knowledge rather than industry-specific applications. **G9** stated, *"I think universities should focus on the core principles of software engineering rather than specific industry tools. Once we understand the concepts, we can pick up any tool quickly in the workplace."* This highlights a key debate in software engineering education: whether universities should aim to produce immediately job-ready graduates or prioritize deeper theoretical knowledge, expecting companies to provide practical training.

Mismatch Between Course Content and Industry Technologies. Out of the 21 graduates interviewed, 12 expressed concerns that the technologies they learned in school were either outdated or insufficient for industry applications. **G3** explained, *"During our database systems course, we primarily used MySQL, but when I started my job, I had to work with NoSQL databases like MongoDB. I had no prior exposure, which made the learning curve quite steep."* This issue was particularly evident in web and mobile application development. Many graduates reported that their coursework covered fundamental concepts but did not expose them to popular frameworks and tools widely used in industry. For example, while many universities still teach Java-based servlets for web development, most modern companies rely on JavaScript frameworks like React and Vue.js. Similarly, mobile development coursework often emphasizes Android development with Java, whereas industry has largely shifted to Kotlin and cross-platform frameworks like Flutter.

However, 4 graduates defended the university approach, arguing that the rapid evolution of technology makes it unrealistic to expect academia to keep pace with every industry trend. **G14** noted, *"If universities start teaching React today, in five years it might be outdated. Instead of focusing on tools, they should teach us the underlying principles of frontend development."* This suggests that rather than attempting to cover every emerging technology, universities might be better off teaching adaptable skill sets that allow graduates to quickly learn new tools as needed.

Confidence in Problem-Solving but Challenges in Adapting to Industry Workflows. While 14 graduates felt confident in their algorithmic and problem-solving skills, many struggled when transitioning into structured industry workflows. **G21** described, *"I had practiced competitive programming extensively, so I was good at writing efficient algorithms. But I had no experience working in an Agile team, using Jira for task tracking, or participating in proper code reviews. It was overwhelming at first."*

Graduates also reported difficulties in managing software repositories collaboratively. While Git was introduced in some courses, its real-world usage—such as handling branching strategies, resolving merge conflicts, and conducting peer code reviews—was far more complex. This aligns with industry reports indicating that new graduates often require an additional six months of training to become fully productive in a professional team setting. Some graduates questioned whether universities should bear responsibility for teaching such industry-specific workflows. **G17** argued, *"Agile, Scrum, and Jira are company-specific practices. Isn't it better for universities to focus on teaching core problem-solving skills and let companies handle the rest?"* However, this view contrasts with the feedback from senior engineers, who expressed frustration that new hires often lacked even basic familiarity with these concepts.

3) Engineers' Perspective from In-Depth Interviews: The senior engineers largely confirmed graduates' self-assessments but also raised specific concerns regarding industry readiness. They highlighted several practical deficiencies that often resulted in difficulties during onboarding and initial project assignments.

Graduates Demonstrate Strong Coding Skills but Lack Practical Debugging Experience. Engineers largely agreed (4 out of 5) that new graduates possess strong algorithmic skills and a solid understanding of fundamental programming concepts. They praised graduates' ability to solve complex problems efficiently, particularly in isolated coding tasks or algorithm-based assessments. **E2** remarked, *"Most of the new hires have excellent problem-solving skills. They can handle complex algorithmic challenges efficiently. However, they often lack experience in real-world debugging and optimization."* While graduates expressed confidence in their technical abilities, engineers noted that their coding expertise was often limited to idealized academic settings, where problems were well-defined and self-contained. In contrast, industry-level software engineering involves dealing with legacy code, debugging unexpected runtime issues, and optimizing large-scale applications, skills that many graduates had little exposure to. **E3** explained, *"They're very good at writing new code from scratch, but when asked to troubleshoot an unfamiliar codebase or fix a complex bug, they struggle. They haven't been trained to deal with messy, real-world code, which is a critical skill in the industry."*

Limited Familiarity with Industry-Standard Development Tools and Workflows. A notable concern was that all engineers found graduates unfamiliar with commonly used industry tools. Although universities often introduce software development tools, engineers reported that graduates' knowledge was often theoretical rather than practical. One prominent example was the preference for Eclipse over IntelliJ IDEA for Java development. **E2** explained, *"Eclipse was popular a decade ago, but now, IntelliJ IDEA offers far better integration with modern frameworks, improved refactoring capabilities, and more powerful debugging tools. When new graduates insist on using Eclipse, it slows down their workflow and requires additional training."* Beyond IDEs, engineers noted that graduates often lacked hands-on experience with other critical industry-standard tools, such as Docker for containerization, Kubernetes for deployment, and CI/CD pipelines for automated testing and integration. **E1** observed, *"They may have learned about these tools in theory, but when it comes*

to actually setting up a Docker container or configuring a Jenkins pipeline, they have no practical experience. This leads to significant delays when onboarding new developers."

Challenges in Adapting to Agile Workflows and Team-Based Development. Engineers also highlighted that graduates struggled with structured industry workflows, particularly Agile methodologies, team-based collaboration, and project management tools like Jira. **E1** noted, *"New hires can write excellent code, but they often struggle with explaining their design decisions or participating in meaningful technical discussions. Many of them are not used to code reviews and tend to take feedback personally rather than constructively."* Engineers reported that while graduates were comfortable working independently on coding tasks, many lacked experience in working collaboratively within a professional development team. One significant challenge was adjusting to peer code reviews. Unlike academic projects, where students work in isolation, industry coding practices require frequent collaboration, version control management, and constructive feedback exchanges. **E3** commented, *"They're not used to receiving critical feedback on their code. When we conduct a code review and ask them to refactor something, some take it as a personal criticism rather than a learning opportunity."* Additionally, engineers also noted that understanding Agile workflows—such as sprint planning, backlog grooming, and daily stand-up meetings—was often a steep learning curve for new graduates.

Difficulties in Understanding Large-Scale Software Architectures. Another frequently mentioned issue was that graduates struggled with working on large, complex codebases. While they were adept at developing small-scale applications, many found it difficult to navigate enterprise-level systems with thousands of interdependent modules. **E1** noted, *"New hires are great at implementing small features but struggle when tasked with modifying an existing system. They don't yet know how to approach refactoring or extending a large, multi-layered application."* **E2** elaborated on this point, *"In school, they work on clean, newly written code. In the real world, they inherit old codebases that are messy, poorly documented, and full of technical debt. We have to spend months training them to read, understand, and modify legacy systems effectively."* This suggests that universities should introduce projects that require students to engage with pre-existing codebases, perform refactoring tasks, and work within real-world software architectures.

In summary, the engineers' perspectives strongly aligned with the concerns expressed by graduates, reinforcing the notion that while theoretical knowledge is well-covered in university programs, practical and industry-oriented experiences are lacking. Engineers acknowledged that new graduates were quick learners and highly motivated, but they often required substantial training before becoming fully productive in a professional environment. The main skill gaps identified by engineers—lack of debugging experience, unfamiliarity with industry tools, challenges with Agile workflows, and difficulty working with large-scale codebases—highlight key areas where university curricula could be improved to better prepare graduates for industry demands.

Answer to RQ1: The findings indicate a gap between graduates' self-perceived readiness and industry expectations, particularly in terms of practical software development experience, familiarity with industry-standard tools, and the ability to adapt to collaborative workflows and large-scale codebases.

B. Online Learning Impact (RQ2)

1) Findings from In-Depth Interviews: The in-depth interviews with nine participants revealed diverse perspectives on how online learning affected their coding skills and industry preparedness. While some participants acknowledged the benefits of flexibility and self-paced learning, others expressed concerns about the limitations of remote education.

Effectiveness of Online Learning for Coding Skills Among the nine participants, four (*G1, G9, G14, G20*) reported that online learning had little impact on their coding proficiency, as they were already self-motivated learners who supplemented their education with personal projects and external coding resources. **G9** shared, *"I didn't feel a significant difference because I had already been practicing coding independently. Online courses gave me the theoretical background, but real coding skills came from my personal projects."* In contrast, five participants (*G3, G11, G13, G16, G21*) believed that the shift to online learning negatively impacted their ability to develop hands-on coding expertise. **G16** explained, *"In offline classes, I could immediately ask the instructor when I faced a coding problem. With online learning, debugging became frustrating because I had to rely solely on documentation and forums."* The lack of immediate feedback was a common issue, especially for complex programming tasks requiring real-time guidance. This suggests that while highly self-driven students could adapt to online learning effectively, others who relied more on interactive problem-solving struggled in this environment.

Challenges and Advantages in Industry Preparation Several participants (*G1, G14*) felt that online learning helped them develop self-discipline and independent problem-solving skills, which are crucial for industry readiness. However, seven participants (*G3, G9, G11, G13, G16, G20, G21*) emphasized that online education did not adequately simulate collaborative industry environments. **G13** stated, *"In offline settings, we had team-based projects with real-time collaboration. In online classes, teamwork felt artificial, and it was harder to develop communication skills."* These findings align with industry concerns that graduates who completed their education primarily online may lack experience in collaborative software development, code reviews, and Agile workflows.

Impact on Coding Assessments in Job Interviews Five participants (*G3, G9, G11, G13, G21*) reported difficulties in technical job interviews due to the lack of hands-on practice in online learning environments. **G9** explained, *"I felt confident answering theoretical questions in job interviews, but when given a live coding test, I struggled because I hadn't practiced*

coding in a high-pressure setting." However, four participants (*G1, G14, G16, G20*) disagreed, stating that online learning did not affect their job interview performance. **G14** argued, *"If you practiced coding enough outside of class, it didn't matter whether the lectures were online or offline."* These contrasting experiences highlight a key finding: students who relied heavily on structured coursework for coding practice faced more challenges in job interviews, whereas those who engaged in self-driven learning were less affected by the shift to online education.

2) Sentiment Classification of Course Assessment Reports: To further understand the impact of online learning on coding skills and industry readiness, we analyzed 3,961 course assessment comments that referenced online and offline learning experiences. Table III presents the sentiment classification results by course level, categorized into bachelor's, master's, and doctoral levels, highlighting the differences in students' perceptions across different stages of education.

TABLE III: Sentiment Classification of Online Learning by Course Level

Level	Positive (%)	Neutral (%)	Negative (%)
Bachelor's	60.33	26.45	13.22
Master's	68.62	23.97	7.41
Doctoral	71.65	18.43	9.92

The results reveal a clear trend: as the level of education increases, students report a higher positive perception of online learning. Doctoral students showed the highest positive sentiment (71.65%), followed by master's students (68.62%), and bachelor's students (60.33%). One possible reason for this trend is that doctoral and master's students rely more on independent research, reading, and self-driven learning, which are well-suited for online education formats. In contrast, undergraduate students, who often require more structured guidance, interactive learning, and hands-on experience, may have struggled more with the transition to online education. The negative sentiment was highest at the bachelor's level (13.22%), significantly higher than the master's (7.41%) and doctoral (9.92%) levels. This aligns with our in-depth interview findings, where undergraduate participants reported greater difficulties in learning programming through online platforms due to limited real-time interaction, reduced hands-on debugging assistance, and weaker peer collaboration.

Answer to RQ2: The findings indicate that while online learning was generally well-received in higher education levels, particularly for theoretical and research-based courses, undergraduate students, especially in programming-intensive courses, faced significant challenges due to the lack of real-time instructor feedback, hands-on coding practice, and collaborative learning opportunities, highlighting a disparity in the effectiveness of online education across different academic levels.

C. LLM Reliance (RQ3)

1) Comparison of Graduates Performance: To systematically assess the impact of LLM reliance on software engineering graduates, we designed a programming assessment comprising 25 questions, extracted from four core software engineering courses: *Object-Oriented Programming*, *Software Design Patterns*, *Algorithm Analysis and Design*, and *Deep Learning and AI Application Development*. These courses reflect essential competencies required for software engineering professionals, covering topics fundamental to both academic learning and industry practice. The assessment was structured to evaluate graduates' proficiency in multiple dimensions, including theoretical knowledge, coding implementation, debugging, optimization, and real-world software design challenges.

The conceptual understanding questions assessed graduates' grasp of fundamental software engineering principles, ensuring they could articulate key concepts such as object-oriented paradigms, algorithm complexity, and architectural patterns. The code completion/implementation tasks required graduates to complete partially implemented code or design functional software components, testing their ability to translate theoretical knowledge into practical applications commonly encountered in professional software development. The debugging/optimization questions challenged graduates to identify and resolve logical errors, improve computational efficiency, and refactor poorly structured code, which are critical for maintaining and enhancing large-scale software systems. Additionally, one of the scenario-based problem-solving questions included an open-ended algorithm design challenge, where graduates were given only the problem requirements and had to independently devise a computational model to address the task. This problem specifically focused on machine learning or deep learning applications, requiring graduates to determine the most suitable algorithm, design the model architecture, and implement it from scratch.

To evaluate the impact of LLM reliance on these competencies, we compared the scores from 19 software engineering graduates who regularly used LLMs with historical student performance data from 2018–2020. Both groups completed identical or comparable assessment questions, with historical students having no access to LLM tools during their coursework. The assessment consisted of 25 questions, each composed of 3-4 sub-questions, with the total score for each sub-question being 5, 10, 15, and 15 (if applicable) points. Table IV presents the detailed scores for each sub-question.

TABLE IV: Comparison of Graduates Performance with Historical Data (2018-2020).

ID	Participant Score				Historical Student Performance				Improvement
	Sub-Qs 1	Sub-Qs 2	Sub-Qs 3	Sub-Qs 4	Sub-Qs 1	Sub-Qs 2	Sub-Qs 3	Sub-Qs 4	
Q1	4.13	7.42	10.25	-	4.06	7.12	8.96	-	↑ 7.61%
Q2	3.96	7.18	9.49	10.72	3.69	6.92	7.34	8.27	↑ 16.36%
Q3	4.27	7.26	11.08	8.26	4.24	7.03	8.96	7.39	↑ 10.53%
Q4	3.98	6.82	9.92	8.53	4.03	6.80	8.29	7.20	↑ 10.02%
Q5	4.19	6.79	10.98	7.68	4.05	6.22	8.99	7.21	↑ 10.70%
Q6	4.02	7.04	10.46	-	3.82	6.92	9.02	-	↑ 8.18%
Q7	3.75	5.91	9.25	-	3.76	5.28	7.35	-	↑ 13.33%
Q8	3.81	6.27	8.74	8.59	3.66	6.02	7.93	6.48	↑ 12.11%
Q9	3.62	7.09	9.58	7.25	3.15	6.16	9.54	5.92	↑ 10.06%
Q10	4.14	6.88	10.05	9.06	3.96	6.09	9.28	7.65	↑ 10.45%
Q11	3.92	6.24	9.26	10.04	4.02	6.13	9.20	8.17	↑ 6.59%
Q12	4.03	6.21	9.07	-	3.94	6.05	7.02	-	↑ 11.91%
Q13	3.83	5.82	10.35	-	3.80	5.66	7.09	-	↑ 17.25%
Q14	4.47	7.02	11.28	12.62	4.21	6.84	9.03	9.71	↑ 15.82%
Q15	4.06	7.08	10.16	-	3.89	6.68	8.94	-	↑ 8.40%
Q16	3.92	7.21	8.96	8.06	3.68	6.33	7.66	6.42	↑ 14.42%
Q17	3.55	6.36	7.28	7.64	3.46	6.27	6.98	5.98	↑ 8.62%
Q18	3.64	6.41	9.88	7.93	3.72	6.19	7.85	6.31	↑ 13.60%
Q19	4.06	6.82	12.86	9.19	4.18	6.24	10.02	7.92	↑ 13.88%
Q20	4.09	6.94	10.59	10.07	4.02	6.32	10.30	8.05	↑ 9.47%
Q21	4.17	7.06	12.79	-	3.95	6.58	10.08	-	↑ 14.20%
Q22	3.89	6.45	9.42	7.63	3.62	6.02	9.39	5.92	↑ 8.91%
Q23	3.68	5.86	7.37	8.02	3.59	5.47	7.02	5.37	↑ 13.96%
Q24	3.26	5.21	8.08	6.64	3.31	5.18	7.53	5.08	↑ 9.01%
Q25	3.59	6.13	8.95	7.14	3.52	5.36	8.98	6.25	↑ 6.59%

The assessment results indicate that students who frequently relied on LLMs generally performed better than their historical counterparts. However, the improvement was not significantly

pronounced across all question types. Notably, in conceptual understanding and scenario-based problem-solving questions, the difference between LLM users and historical students remained moderate. The most substantial gains were observed in code completion and debugging tasks, where LLM-generated responses provided direct assistance in correcting syntax errors and optimizing algorithms. Additionally, students relying on LLMs demonstrated a higher accuracy in addressing the third or fourth sub-question of multi-part problems, as these often involved refactoring or improving pre-existing code—tasks where LLM-generated suggestions tend to be more effective. The structured nature of these problems made them well-suited for AI assistance, allowing students to enhance their solutions with minimal manual intervention.

While these quantitative results highlight the efficiency gains facilitated by LLMs, instructors expressed concerns regarding students' depth of reasoning and independent problem-solving skills. Specifically, when required to design software architectures, optimize complex algorithms, or propose scalable solutions for industry-based scenarios, students who relied heavily on LLMs often produced highly similar or formulaic responses, suggesting an over-reliance on AI-generated templates. This tendency was particularly evident in software or algorithm design and system architecture questions, where students failed to demonstrate a thorough understanding of trade-offs, alternative approaches, or real-world constraints. This pattern suggests that while LLMs improve structured programming efficiency, they may inadvertently limit students' ability to critically evaluate software development decisions, leading to reduced creativity and a weaker grasp of fundamental software engineering principles.

2) Findings from In-Depth Interviews: The interviews with six graduates revealed significant insights into the impact of LLM reliance on coding efficiency and job market preparedness. In summary, while all six graduates recognized the efficiency benefits of LLMs, particularly in code generation and debugging, their perspectives on its impact on long-term skills development varied. The interviews revealed that graduates generally found LLMs beneficial in improving coding efficiency and technical interview preparation. Several participants noted that AI-assisted tools helped streamline their workflow, particularly in debugging and optimizing code. **G3** remarked, *"LLMs helped me refine my code during LeetCode practice, making my solutions cleaner and more efficient."* This sentiment was echoed by others who found that AI-generated suggestions allowed them to focus more on problem-solving rather than syntax or implementation details. Many felt that this improved their speed in solving coding challenges, which was especially useful for **technical assessments** during job applications. However, concerns were raised about the potential drawbacks of heavy reliance on LLMs, particularly in the context of job preparedness. **G12** questioned, *"If recruiters know I rely heavily on LLMs, will they doubt my real ability to solve problems independently?"* Similarly, **G19** expressed reservations about whether AI assistance might mask deficiencies in fundamental problem-solving skills. These concerns suggest that while LLMs enhance efficiency, they might also create perceptions of dependency, raising doubts among employers about graduates' ability to tackle complex software engineering tasks without AI support. Despite these concerns, most graduates agreed that the balanced usage of LLMs could be an asset in both learning and job preparation.

Answer to RQ3: The findings indicate that while reliance on LLMs enhances problem-solving abilities, coding efficiency, and job interview preparation for software engineering graduates, it also raises concerns about diminishing independent problem-solving skills and creativity. This suggests the need to balance AI tool usage with the development of critical thinking.

D. Recommendations (RQ4)

Table V presents the most frequently mentioned terms and their corresponding importance scores for key categories, including practical relevance, course difficulty, and instructional quality. The TF-IDF analysis shows that key terms like "industry tools", "real-world projects", and "coding practice" are most important in bachelor's and master's courses, reflecting the emphasis on aligning coursework with practical applications. For example, in Bachelor's level Software Engineering courses, students highly valued learning industry-standard tools and gaining real-world coding experience through hands-on projects. Similarly, master's students highlighted the importance of research integration and using professional tools to prepare for industry applications. The LDA topics further emphasized the need for theory-practice integration across all levels. At the bachelor's level, the focus was on project-based learning, while at the master's level, students emphasized applied knowledge and collaborative development with industry-driven projects. At the doctoral level, a major theme was research application, where students sought opportunities to bridge advanced theoretical knowledge with practical software engineering problems. This indicates a growing desire across all levels to integrate academic learning with real-world industry practices.

To validate and refine our findings, five senior evaluation experts participated in a two-round cross-validation process. **Ex3** pointed out that doctoral students should focus on research-practice integration, stating, *"Doctoral students should be encouraged to work on industry-linked research projects to enhance the relevance of their work."* Additionally, **Ex4** noted the importance of integrating industry-driven projects in Master's programs. *"Master's students need to engage in projects that reflect real-world problems. Without such projects, students may struggle to bridge the gap between academic learning and industry needs."* This was reflected in the LDA findings, where applied knowledge and collaborative development were frequently mentioned by students and faculty. The experts also emphasized the need for interdisciplinary collaboration across all levels, particularly between software engineering and fields such as business and data science. This feedback underscored the importance of cross-disciplinary learning to better prepare graduates for complex, multi-faceted problems encountered in the industry.

TABLE V: TF-IDF and LDA Results for Key Curriculum Features.

Category	Top Terms (TF-IDF)	LDA Topics
Practical Relevance (Bachelor's)	Industry tools, Real-world projects, Coding practice, Hands-on experience	Integration of theory and practice, Project-based learning, Real industry cases
Practical Relevance (Master's)	Industry applications, Research integration, Professional tools, Practical coding	Industry-driven projects, Applied knowledge, Collaborative development
Course Difficulty (Bachelor's)	Complex assignments, Steep learning curve, Advanced algorithms, Challenging content	Curriculum rigor, Assignment complexity, Challenge perception
Course Difficulty (Master's)	Advanced topics, High-level algorithms, Research-based tasks, Theoretical depth	Research application, Theory vs practice, Complex problem solving
Instructional Quality (Bachelor's)	Clear explanations, Interactive teaching, Timely feedback, Supportive instructors	Effective teaching, Student engagement, Collaborative learning
Instructional Quality (Master's)	Mentorship, Research guidance, In-depth tutorials, Peer collaboration	Research mentoring, Tutorial effectiveness, Academic mentorship
Course Practicality (Ph.D.)	Industry-specific tools, Research application, Software prototyping, Lab work	Real-world applications, Industry partnerships, Research-practice integration
Course Theory vs Practice (Ph.D.)	Advanced research, Theory application, Software simulation, Problem-based learning	Real-world research, Conceptual application, Deep research integration

Answer to RQ4: The findings suggest that bridging the educational gaps in software engineering requires a balanced integration of theoretical and practical experience, and industry collaboration across all academic levels, supported by curriculum enhancements that reflect real-world tools, interdisciplinary learning, and the development of both technical and soft skills.

IV. Conclusion

This study provides a comprehensive exploration of the preparedness of software engineering graduates for industry roles, addressing critical gaps in theoretical knowledge, practical skills, and industry alignment. By leveraging insights from graduates, educators, and industry professionals, this study proposes targeted recommendations to enhance curriculum design, bridge educational gaps, and foster stronger collaboration between academia and industry.

Acknowledgment

The work described in this paper is partially supported by the Hong Kong Metropolitan University Research Grant (Project Reference No. RD/2025/1.24 and RD/2025/1.21), and partially supported by the grant from the Research Grant Council (RGC) of the Hong Kong Special Administrative Region, China (Project Reference No. UGC/FDS16/E27/25).

References

[1] S. Pirelli, "Scalable teaching of software engineering theory and practice: An experience report," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, 2024, pp. 286–296.

[2] S. Ouhbi and N. Pombo, "Software engineering education: Challenges and perspectives," in *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2020, pp. 202–209.

[3] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, "Aligning software engineering education with industrial needs: A meta-analysis," *Journal of Systems and Software*, vol. 156, pp. 65–83, 2019.

[4] O. Cico, L. Jaccheri, A. Nguyen-Duc, and H. Zhang, "Exploring the intersection between software industry and software engineering education-a systematic mapping of software engineering trends," *Journal of Systems and Software*, vol. 172, p. 110736, 2021.

[5] D. Russo, "Pandemic pedagogy: Evaluating remote education strategies during covid-19," *Journal of Systems and Software*, vol. 226, p. 112392, 2025.

[6] A. Gordillo, D. López-Fernández, and E. Tovar, "Comparing the effectiveness of video-based learning and game-based learning using teacher-authored video games for online software engineering education," *IEEE Transactions on Education*, vol. 65, no. 4, pp. 524–532, 2022.

[7] A. Alam and A. Mohanty, "Design, development, and implementation of software engineering virtual laboratory: A boon to computer science and engineering (cse) education during covid-19 pandemic," in *Proceedings of third international conference on sustainable expert systems: ICSES 2022*. Springer, 2023, pp. 1–20.

[8] N. T. Kerman, S. K. Banihashem, M. Karami, E. Er, S. Van Ginkel, and O. Noroozi, "Online peer feedback in higher education: A synthesis of the literature," *Education and Information Technologies*, vol. 29, no. 1, pp. 763–813, 2024.

[9] N. N. Vo, N. H. Vu, T. A. Vu, Q. T. Vu, and B. D. Mach, "Crs: a hybrid course recommendation system for software engineering education," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, 2022, pp. 62–68.

[10] E. Ceh-Varela, C. Canto-Bonilla, and D. Duni, "Application of project-based learning to a software engineering course in a hybrid class environment," *Information and Software Technology*, vol. 158, p. 107189, 2023.

[11] S. Ahmed, S. A. H. Bukhari, A. Ahmad, O. Rehman, F. Ahmad, K. Ahsan, and T. W. Liew, "Inquiry based learning in software engineering education: exploring students' multiple inquiry levels in a programming course," in *Frontiers in Education*, vol. 10. Frontiers Media SA, 2025, p. 1503996.

[12] N. Meißner, N. Koch, S. Speth, U. Breitenbücher, and S. Becker, "Unveiling hurdles in software engineering education: the role of learning management systems," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, 2024, pp. 242–252.

[13] I. Fronza, L. Corral, and C. Pahl, "End-user software development: Effectiveness of a software engineering-centric instructional strategy," *The Journal of Information Technology Education: Research*, vol. 19, pp. 367–393, 2020.

[14] R. Choudhuri, D. Liu, I. Steinmacher, M. Gerosa, and A. Sarma, "How far are we? the triumphs and trials of generative ai in learning software engineering," in *Proceedings of the IEEE/ACM 46th international conference on software engineering*, 2024, pp. 1–13.

[15] O. Manakina and C.-H. Lung, "Designing reusable llm-enhanced assignments: A quality-oriented framework for software engineering education," in *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2025, pp. 2212–2217.

[16] J. Pereira, J.-M. López, X. Garmendia, and M. Azanza, "Leveraging open source llms for software engineering education and training," in *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2024, pp. 1–10.

[17] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From llms to llm-based agents for software engineering: A survey of current, challenges and future," *arXiv preprint arXiv:2408.02479*, 2024.

[18] V. Kozov, G. Ivanova, and D. Atanasova, "Practical application of ai and large language models in software engineering education." *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 1, 2024.

[19] V. D. Kirova, C. S. Ku, J. R. Laracy, and T. J. Marlowe, "Software engineering education must adapt and evolve for an llm environment," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 666–672.