UnifIDE: A Collaborative IDE with CRDT-Based Real-Time Editing and Code Evaluation

Denise Aliah B. Cabiso, Johana N. Taboada, Goddess Phoebe J. Valdehuesa, and Glende P. Brazil

University of Cebu–Lapu-Lapu and Mandaue, Philippines

***Abstract*—**Programming education at the University of Cebu continued to rely on tools that evaluated code primarily through correctness, offering limited support for collaboration, contribution fairness, and deeper analysis of code quality. These constraints prevented students from engaging in realistic teamwork and made it difficult for educators to assess group performance with transparency. UnifIDE addressed these limitations through a web-based collaborative integrated development environment that enabled real-time multi-user coding using a CRDT-based synchronization layer built with Yjs and WebSocket communication. The platform incorporated a structured performance incentive system—including point-based scoring, achievement badges, and ranked leaderboards—to encourage balanced participation, and featured an AI-assisted code evaluation system that integrated deterministic execution metrics with narrative feedback generated through the Gemini API to guide student improvement. UniXcoder was employed to detect AI-generated code, reinforcing academic integrity across collaborative tasks. Educators gained access to automated contribution analytics and collaboration insights, supporting clearer evaluation of individual and group activity. Developed under a developmental–experimental research design and refined iteratively through Agile methodology, UnifIDE was evaluated through a survey of students and educators from the College of Computer Studies. Findings revealed that most respondents recognized the shortcomings of existing coding platforms and expressed strong willingness to adopt UnifIDE, while the system proved fast, accurate, and reliable during controlled technical testing. The study concluded that UnifIDE was both technically robust and strongly accepted by its intended users, positioning it as a pedagogically aligned and institutionally deployable platform for collaborative programming education.

Keywords*—Collaborative IDE, CRDT, Yjs, WebSockets, UniXcoder, AI-assisted code evaluation, contribution tracking, programming education, academic integrity.*

I. INTRODUCTION

The accelerating expansion of the digital economy has intensified demand for graduates with strong programming competence and the ability to work effectively in development teams. Globally, the software development workforce was projected to grow 15 percent from 2024 to 2034, generating approximately 129,200 new openings annually—a rate that far exceeded the average across all occupations [1]. This rapid growth signaled not only an urgent need for technically proficient programmers, but also for graduates capable of collaborative, quality-focused development that mirrored real-world professional workflows.

A variety of online coding platforms such as CodeBuddy, Codeforces, and Codewars became widespread tools for programming instruction. These systems offered accessible environments where students could practice independently and receive automated test-case feedback. While they improved access to programming practice, existing research consistently showed that their instructional models emphasized solitary work and provided limited support for collaborative problem-solving. Gupta and Goel [2] noted that most widely used coding environments lacked mechanisms for real-time interaction, peer reasoning, or shared construction of solutions. This was misaligned with the expectations of contemporary software engineering, where teamwork, communication, and adaptive problem-solving were regarded as essential competencies across global industries [3].

A growing body of evidence in computing education emphasized that collaboration played a critical role in deepening conceptual understanding and sustaining student engagement. Studies on dis- tributed pair programming and peer-facilitated learning showed significant gains in performance and persistence when students worked together to articulate, critique, and refine code [4]. More structured collaborative approaches, such as peer- assessment-driven Scrum processes, demonstrated improvements in communication, critical thinking, and project quality compared to conventional group arrangements [5]. These findings indicated that collaborative learning could develop both cognitive and interpersonal skills essential to software development. Yet despite these documented benefits, collaboration features remained underdeveloped in mainstream programming platforms.

A separate but equally significant limitation concerned how student code was evaluated. Most platforms relied on correctness-based automated grading, assessing submissions solely by whether they matched expected outputs. [6] systematic review showed that common grading tools provided minimal feedback on structural or stylistic aspects of code. As a result, students often prioritized passing test cases rather than developing professional habits related to readability, organization, abstraction, and maintainability. Recent developments in artificial intelligence provided opportunities to expand the scope of automated feedback beyond correctness. AI-assisted review tools helped students identify semantic errors, code smells, and opportunities for refactoring, resulting in deeper engagement with code logic and more meaningful revisions [7].

The rise of large language models in academic settings also introduced a new challenge: the authenticity of student submissions. [25] discussed the educational opportunities and

challenges associated with AI code generation, including its implications for programming education and assessment. Existing platforms lacked mechanisms to detect AI-generated code or verify authorship, leaving a critical gap in maintaining integrity in programming courses. In addition, motivation remained a key factor in student persistence. Research on reward-based engagement systems showed that well-designed point structures and progression mechanics could improve engagement and reduce dropout rates in programming education [9]; [10].

These challenges carried particular urgency in the Philippine context. The country's IT-BPM industry, one of the largest of its kind globally, employed approximately 1.9 million Filipinos [29], while the national government's Trabahong Digital initiative targeted the creation of 8 million digital and digitally- enabled jobs by 2028 [12]. Interviews with senior industry executives revealed that many Filipino CS graduates lacked practical skills in collaborative workflows, agile development, and code maintainability—areas directly shaped by the tools and environments used in programming education [13]. CHED Memorandum Order No. 25 [26] mandated outcomes that included teamwork and professional practice, but the platforms available to educators rarely supported the development of these competencies in meaningful, assessable ways.

In response to these persistent challenges, this study proposed and developed UnifIDE, a web-based collaborative integrated development environment that unified four key components: Conflict-free Replicated Data Type (CRDT)-driven real-time editing via Yjs and WebSocket communication; automated contribution analytics to promote transparency and fairness in group work; structured scoring, leaderboards, and achievement mechanics aligned with pedagogical objectives; and AI-assisted code evaluation capable of offering narrative feedback on code quality through the Gemini API while verifying authorship through AI-generated code detection powered by a fine-tuned UniXcoder model.

1.1 Objectives

This study aims to design and develop UnifIDE, a web-based collaborative integrated development environment that supports multiple programming languages and incorporates real-time collaboration, contribution tracking, structured engagement incentives, and AI- assisted code evaluation. Specific Objectives: Precisely, it aims to achieve the following:

1. Assess existing educational coding platforms in terms of:
1.1. functionalities that support programming education, and
1.2. user preferences and experiences.
2. Design the system architecture of UnifIDE incorporating:
2.1. real-time collaborative editing with contribution tracking,
2.2. class activity workflows, and
2.3. structured engagement incentive rules.
3. Develop an AI-assisted code evaluation module to:
3.1. generate narrative feedback on adherence to language-specific coding standards, and
3.2. implement AI-generated code detection to enforce academic integrity.
4. Integrate the refined algorithms and trained AI model into the platform to enable seamless collaboration, contribution tracking, engagement incentive system, and code evaluation.
5. Implement the proposed UnifIDE system and conduct system testing to verify its functionality.

1.2 Scope and Limitations of the Study

This study focused on the design and development of UnifIDE, a web- based collaborative programming environment intended for use in Computer Science and Information Technology education at the University of Cebu – Lapu-Lapu and Mandaue (UCLM). The platform integrated real-time collaborative editing, contribution tracking, structured engagement incentive, and AI-assisted code evaluation to support programming exercises and group-based learning tasks. UnifIDE supported six programming languages—Java, JavaScript, C, C#, Python, and PHP—selected for their relevance to foundational programming courses. Code execution occurred within sandboxed containers that enforced strict security and resource limits. Only standard built-in libraries included in these sandboxes were avail- able; external or third-party packages could not be installed due to security, performance, and portability constraints.

The AI-assisted code evaluation feature served a dual purpose: it generated narrative feedback on adherence to language-specific coding standards and incorporated an AI-generated code detector to support academic integrity. The AI detection features produced probability-based results and served as a decision-support tool rather than definitive proof of misconduct. For short code sub- missions, the system fell back to a session-based integrity check that evaluated behavioral signals such as time spent and code execution activity. The study evaluated UnifIDE only within a programming education context at UCLM, and its findings should not be generalized to other disciplines or professional software development environments.

Since UnifIDE is a web-based platform, it required a stable internet connection to function properly. Unstable or slow connections could affect how smoothly real-time collaboration works and how quickly code execution results are returned. The real-time editing feature, built on CRDT technology, also requires more processing as more users edit simultaneously or as files become larger. These conditions were not extensively tested in this study, and performance under such circumstances may vary.

II. METHODOLOGY

This chapter outlines the research and development strategies implemented for UnifIDE. It discusses the research design, respondents, instruments used, data collection procedures, statistical treatments, and the software engineering methodology guiding the system's development. Ethical considerations and development protocols are also included to ensure systematic execution and reliable outcomes.

2.1 Research Design

This study adopted a Developmental–Experimental Research Design, which combined systematic system development with controlled empirical evaluation [11]. The developmental component involved the iterative design and construction of UnifIDE through Agile methodology, enabling incremental delivery and continuous improvement based on stakeholder feedback [14]. The experimental component included structured performance tests of the CRDT synchronization engine and the UniXcoder AI detection model, alongside survey-based acceptability assessment from student and educator respondents. The combination of these approaches ensured that UnifIDE was evaluated both as a functioning artifact and as a tool perceived as useful and appropriate by its intended users.

2.2 Research Environment

The study was conducted at the University of Cebu – Lapu-Lapu and Mandaue (UCLM), a culturally diverse academic institution located at the base of the historic Mactan Bridge, which connects Mandaue City to Lapu-Lapu City. The University of Cebu, founded in 1964, has continuously shaped the lives of countless students and educators, providing quality education grounded in nationalism, humanity, and academic excellence. The College of Computer Studies served as the primary research environment, providing access to students enrolled in programming-related courses and educators who regularly use digital coding tools in instruction.

2.3 Respondents

The survey respondents consisted of students and educators from the College of Computer Studies at UCLM, representing undergrad- uate programs in Computer Science, Information Technology, and related fields. Educators were selected based on their involvement in programming-related instruction. Following the validation pro- cess of 542 initial responses, 500 valid responses were retained for analysis. The distribution of validated respondents is presented in Table 1.

TABLE 1. DISTRIBUTION OF RESPONDENTS

Respondents	No. of Respondents	Percentage (%)
Students	490	98.00
Educators	10	2.00
Total	**500**	**100.00**

2.4 Research Instrument

Two structured survey questionnaires were used as the primary research instruments—one for students and one for educators. Both followed a shared structure but differed in content to reflect each group's roles and experiences. Section A gathered demographic data such as year level, program, and familiarity with existing coding platforms. Sections B through D used Likert-scale items to assess respondents' perceptions of current platforms, the proposed mechanisms of UnifIDE, and the acceptability of its key features. Section E included an open-ended item for additional feedback. Reverse- polarity sincerity-check items were embedded to validate response consistency, ensuring that only credible responses were used in the analysis. In addition, controlled technical testing instruments were used to assess CRDT synchronization latency and convergence, and to evaluate the classification performance of the fine-tuned UniXcoder model.

2.5 Statistical Treatment of Data

Frequency distributions and weighted means were computed to analyze Likert-scale responses. The weighted mean was computed using the formula: $WM = \text{Í}(f \times x)/N$, where $f$is the frequency of each response, $x$is the Likert-scale value, and $N$is the total number of respondents. The scoring scale used to interpret the weighted means is presented in Table 2.

TABLE 2. SCORING PROCEDURE

Scale	Qualitative Description	Weighted Mean Range
5	Strongly Agree	4.20 – 5.00
4	Agree	3.40 – 4.19
3	Neutral	2.60 – 3.39
2	Disagree	1.80 – 2.59
1	Strongly Disagree	1.00 – 1.79

CRDT synchronization performance was evaluated through mean latency analysis using the formula: Mean Latency = $\text{Í}\ \Delta t/N$, where Δtrepresents the propagation delay between a local edit and its appearance on a remote client, and $N$is the total number of operations. Convergence was assessed by comparing final document states (State$i$) across all $k$collaborating clients after simultaneous edits. UniXcoder's binary classification performance was evaluated using precision, recall, F1-score, and overall accuracy derived from its confusion matrix.

2.6 Ethical Considerations

All required institutional approvals and ethical protocols were strictly observed in this study. Initial approval was obtained through a transmittal letter signed by the Dean of the College of Computer Studies. Supporting documents including title approval, proposal approval, and adviser acceptance forms were completed prior to data collection. The study was submitted to the University's Research Ethics Committee and received approval prior to data collection. Participant rights, voluntary informed consent, and strict confidentiality

were maintained throughout all data collection and handling activi- ties, ensuring adherence to university and accepted research ethics standards.

2.7 Software Engineering Methodology

UnifIDE was developed using the Agile methodology, which sup- ported iterative, user-centered development with continuous feed- back integration [15]. Development was organized into sprints covering planning, design, construction, testing, and deployment phases. This iterative cycle allowed for continuous adaptation based on user testing results and adviser feedback. The platform utilized a modern, scalable technology stack. The frontend was constructed using React, Vite, and Tailwind CSS, integrating CodeMirror to handle the coding interface. Real-time CRDT-based collaborative editing was powered by Yjs and WebSocket communication. The

backend API layer was built on Node.js and Express 5, utilizing Supabase (PostgreSQL) for database management and se- cure authentication. The AI-Assisted Code Evaluation layer was developed using Python, incorporating a fine-tuned UniXcoder model for the probabilistic detection of AI-generated code and the Gem- ini API for the generation of narrative feedback. System reliability and concurrent load stability were actively evaluated using Vitest throughout the testing phases.

III. RESULTS AND DISCUSSION

This chapter presented, analyzed, and interpreted the data collected. The information was gathered from students and educators through a combination of Google Forms and printed survey questionnaires. The primary focus of the collected data was on the current processes and limitations of existing educational coding platforms, the per- ceptions of each respondent group regarding the proposed system, UnifIDE, and the evaluation of its proposed mechanisms and key features, including real-time collaboration, contribution tracking, AI-assisted code evaluation, and structured engagement incentive systems.

3.1 Demographic Profile of the Respondents

This section summarized the demographic characteristics of both student and educator respondents. As shown in Table 1, the respondents were primarily 490 students (98.0%), followed by 10 educators (2.0%), for a total of 500 participants. This distribution reflected that UnifIDE was mainly intended for student use while still incorporating instructional perspectives from educators. The dominance of student responses ensured that the findings strongly reflected student needs in collaboration, feed- back, and engagement. At the same time, the inclusion of educators supported considerations of fairness, monitoring, and classroom implementation [2].

3.1.1 Profile of Student Respondents. This section presented the demographic profile of the student respondents, including their year level, program, and familiarity with educational coding platforms. These characteristics provided important context for understanding how students may perceive and interact with UnifIDE. Since the distribution of respondents reflected the department's actual enrollment composition, where the BSIT program has a larger population than the BSCS program, the profile was considered representative of the intended user group for the proposed system.

TABLE 3. YEAR LEVEL OF STUDENT RESPONDENTS

Year Level	Frequency	%
1st Year	135	27.55
2nd Year	119	24.29
3rd Year	131	26.73
4th Year	105	21.43
Total	**490**	**100.00**

Respondents were drawn from different year levels within the computing programs, representing varying stages of programming exposure and academic maturity. This distribution was important because students at different levels may have differed in their readiness for collaborative coding, engagement incentive elements, and AI-assisted feedback mechanisms integrated into UnifIDE. As shown in the table, first-year students comprised the largest group, with 135 respondents (27.55%), followed closely by third- year students with 131 respondents (26.73%). Second-year students accounted for 119 respondents (24.29%), while fourth-year students represented the smallest proportion, with 105 respondents (21.43%). This distribution indicated a relatively balanced representation across year levels, with a slight concentration in the program's early years. Since a large portion of respondents were still in the foundational stages of programming, UnifIDE was designed to emphasize intuitive interfaces, guided collaboration mechanisms, and clear AI- generated feedback explanations. At the same time, features such as performance analytics and efficiency-based code evaluation were designed to remain sufficiently advanced to support upper-year students preparing for professional development and capstone-level projects [6].

TABLE 4. PROGRAM OF STUDENT RESPONDENTS

Program	Frequency	%
BS Computer Science (BSCS)	56	11.43
BS Information Technology (BSIT)	434	88.57
Total	**490**	**100.00**

Respondents were enrolled in the Bachelor of Science in Information Technology (BSIT) and Bachelor of Science in Computer Science (BSCS) programs. As shown in the table, the majority were from BSIT with 434 respondents (88.57%), while 56 respondents (11.43%) were from BSCS. Although the distribution is heavily composed of BSIT students, this accurately reflects the actual enrollment structure of the department, where the BSIT program has significantly more students than BSCS. Therefore, the proportion of respondents is representative of the department's population and does not indicate sampling bias. Given this composition, the findings primarily reflected the perspectives of students engaged in application-oriented computing education, which aligned well with UnifIDE's focus on collaborative coding, engagement incentive systems, and AI-assisted evaluation in practical learning environments. Focusing on these application-oriented student perspectives supports the pedagogical value of contextualized, project-based activities, which allow students to construct knowledge through hands-on interaction [16].

TABLE 5. STUDENTS' FAMILIARITY WITH EDUCATIONAL CODING PLATFORMS

Year Level	Frequency	%
1st Year	135	27.55
2nd Year	119	24.29
3rd Year	131	26.73
4th Year	105	21.43
Total	**490**	**100.00**

Respondents were also asked to indicate their level of familiar- ity with educational coding platforms such as Codeforces, Code- wars, and CodeBuddy. Familiarity with

such systems influenced how quickly students could adapt to a collaborative and incentive-based coding environment like UnifIDE. The results reveal that 203 respondents (45.72%) reported a moderate level of familiarity, making it the largest group. This was followed by 181 respondents (40.77%) who indicated basic familiarity, while only 60 respondents (13.51%) identified as advanced users. This distribution suggested that while most students had prior exposure to online coding platforms, the majority were not yet highly advanced users. Therefore, UnifIDE was designed to incorporate clear onboarding processes, tooltips, and structured collaborative workflows to support students with only basic experience. At the same time, advanced features such as performance-based scoring, real-time collaborative editing, and AI-assisted evaluation were designed to remain robust enough to challenge and engage more experienced users. Balancing accessibility with functionality enhanced adoption and sustained engagement within diverse student groups. This design rationale aligns with findings from [17], who emphasized that programming platforms must accommodate varying levels of prior experience to achieve effective adoption in academic settings.

3.1.2 Profile of Educator Respondents. This section presented the demographic profile of the educator respondents, including their years of teaching experience, subjects taught, and familiarity with educational coding platforms. These characteristics provided essential context for understanding how educators may have evaluated and potentially implemented UnifIDE within their courses. By examining their professional background and level of exposure to coding platforms, the study ensured that the assessment of the proposed system reflected informed instructional perspectives and aligned with the teaching practices within the department.

TABLE 6. EDUCATOR'S YEARS OF TEACHING EXPERIENCE

Years of Experience	Frequency	%
Less than 1 year	2	20.00
1 – 3 years	4	40.00
4 – 6 years	2	20.00
7 years or more	2	20.00
Total	**10**	**100.00**

The educator respondents varied in how long they have been teaching. The biggest group were those who have been teaching for 1 to 3 years, making up 4 out of 10 educators (40.0%). The remaining six educators were evenly split, with 2 respondents each (20.0%) falling under the less than 1 year, 4 to 6 years, and 7 or more years categories. Overall, most of the educator respondents were relatively new to teaching, though some brought longer classroom experience. Having a mix of both newer and more experienced educators meant that the feedback gathered reflected a wider range of teaching perspectives, making the evaluation of UnifIDE more well-rounded and meaningful. This diverse range of teaching experience strengthens the validity of the evaluation, as it captures both early-career instructors who are more familiar with emerging tools and experi- enced educators whose pedagogical depth adds important insight [18].

TABLE 7. EDUCATOR'S SUBJECTS TAUGHT

Subject Area	Frequency	%
Computer Programming (COMPROG)	7	70.00
Applications Development (APPSDEV)	5	50.00
IT Electives (IT ELEC)	3	30.00
Web Development (WEBDEV)	1	10.00
Information and Database Systems (IMDB-SYS)	1	10.00
Testing and Quality Assurance (TESTQUA)	1	10.00
Quantitative Methods (QUAMETH)	1	10.00

*Note: Educators could select multiple subjects, thus frequencies do not total 10. The educator respondents were asked to identify the subjects they taught. As shown in the table, Computer Programming (COM- PROG) was the most commonly taught subject, with 7 out of 10 educators (70.0%) handling it, making it the dominant subject among the respondents. Applications Development (APPSDEV) followed with 5 educators (50.0%), while IT Electives (IT ELEC) was taught by 3 educators (30.0%). The remaining subjects—Web Development (WEBDEV), Information and Database Systems (IMDBSYS), Test- ing and Quality Assurance (TESTQUA), and Quantitative Methods (QUAMETH)—were each taught by only 1 educator (10.0%) respectively.

The high concentration of educators teaching COMPROG and APPSDEV was particularly relevant to UnifIDE, as these subjects directly involved programming instruction and software development activities—the core use cases the system was designed to support. This alignment strengthened the validity of educator feedback, as most respondents had firsthand experience with the instructional contexts in which UnifIDE would be deployed. [19] similarly noted that tools for automated code assessment are most meaningfully evaluated by instructors who teach active program- ming courses where submission volume and feedback quality directly affect student learning outcomes.

TABLE 8. EDUCATOR'S FAMILIARITY WITH CODING PLATFORMS

Level of Familiarity	Frequency	%
Basic	6	60.00
Moderate	4	40.00
Advanced	0	0.00
Total	**10**	**100.00**

When asked about their familiarity with educational coding plat- forms like Codeforces, Codewars, and CodeBuddy, most educators reported only a basic level of experience. Specifically, 6 out of 10 educators (60.0%) described themselves as basic users, while the remaining 4 (40.0%) considered themselves moderately familiar. None of the educators identified as advanced users. This indicated that most educators had some exposure to coding platforms, but were not yet deeply experienced with them. Because of this, UnifIDE was designed to be easy to navigate, with clear

dashboards and simple tools that allowed educators to manage class activities, track student participation, and review AI-generated feedback without needing advanced technical knowledge. This design choice positions AI as a "productivity multiplier" for teachers rather than an operational burden [20].

3.2 Current Process of the Existing System

This section summarized respondents' perceptions of existing educational coding platforms and their effectiveness in supporting collaborative programming within classroom settings. It highlighted key limitations in real-time collaboration, fairness in evaluating individual contributions, depth of feedback beyond correctness, transparency of participation, structured classroom workflow sup- port, meaningful engagement incentive design, and mechanisms for detecting AI-generated code. These insights established the foundational gaps that UnifIDE aimed to address through a more integrated and education-focused collaborative coding environment.

TABLE 9. CURRENT PROCESS OF THE EXISTING SYSTEM

Statement	Mean	Description
CPES1. Most existing educational coding platforms are oriented toward individual work rather than collaborative programming.	4.66	Strongly Agree
CPES2. Current platforms offer limited real-time collaboration features for group-based programming tasks.	4.47	Strongly Agree
CPES3. It is challenging to fairly evaluate individual contributions when using current platforms for group activities.	4.60	Strongly Agree
CPES4. Feedback provided by existing coding platforms primarily focus on correctness or test case results.	4.54	Strongly Agree
CPES5. Current platforms provide limited guidance on higher-order code quality aspects beyond correctness.	4.47	Strongly Agree
CPES6. Existing engagement incentive elements (if present) tend to reward speed or superficial activity rather than meaningful learning.	4.52	Strongly Agree
CPES7. Current educational platforms offer limited transparency regarding student participation and performance in programming activities.	4.54	Strongly Agree
CPES8. Current coding platforms offer limited support for structured class activity workflows (e.g., educator-led programming sessions).	4.52	Strongly Agree
CPES9. Existing platforms provide limited mechanisms for detecting AI-generated code within student submissions.	4.58	Strongly Agree
GRAND MEAN	**4.55**	**Strongly Agree**

Table 9 presented the respondents' assessment of the Current Process of Existing Systems in educational coding platforms. The findings indicated a strong agreement regarding the limitations of current systems, with a grand mean of 4.55 (SD = 0.62). This suggested that respondents consistently perceived significant gaps in existing platforms, particularly in supporting collaborative and higher-level programming activities. Respondents strongly agreed that most existing platforms are oriented toward individual work rather than collaborative program- ming (M = 4.66, SD = 0.56). This aligns with the observations of Gupta and Goel [2], who noted that most widely used coding environments lack mechanisms for real-time interaction or shared construction of solutions. They also noted that real-time collabo- ration features for group-based tasks are limited (M = 4.47, SD = 0.58) and that fairly evaluating individual contributions in group activities remains challenging (M = 4.60, SD = 0.61). These findings highlighted structural limitations in current coding environments, particularly in supporting teamwork and accountability. In terms of feedback mechanisms, respondents agreed that exist- ing platforms primarily focused on correctness or test case results (M = 4.54, SD = 0.64) and provided limited guidance on higher-order code quality beyond correctness (M = 4.47, SD = 0.65). This corroborates the systematic review by [6], which showed that common grading tools provide minimal feedback on structural or stylistic aspects of code. Additionally, reward and ranking elements were perceived to reward speed or superficial activity rather than meaningful learning (M = 4.52, SD = 0.60). Limited transparency in student participation and performance (M = 4.54), insufficient support for structured class workflows (M = 4.52), and weak mechanisms for detecting AI-generated code (M = 4.58) further reinforced these concerns. Overall, the results demonstrated a clear consensus that current educational coding platforms lacked robust collaborative support, deeper evaluation mechanisms, and transparency tools. These findings justified the development of UnifIDE, which aimed to integrate real-time collaboration, AI-assisted evaluation, contribution track- ing, and structured classroom management to address the identified limitations.

3.3 Mechanisms or Techniques of the Proposed System

This section examined respondents' acceptance of UnifIDE's core mechanisms, including contribution tracking, learning analytics for participation monitoring, structured real-time class activity work- flows, AI-assisted code evaluation, AI-generated code detection, and real-time collaborative coding through a shared editor. It evaluated how effectively these mechanisms aligned with users' expectations for fairness, transparency, meaningful feedback, instructional coordination, and academic integrity in programming education.

TABLE 10. MECHANISMS OR TECHNIQUES OF THE PROPOSED SYSTEM

Statement	Mean	Description
MPS1. Incorporating a contribution-tracking function would help ensure fairness and accountability in group programming activities.	4.58	Strongly Agree
MPS2. Using learning analytics to monitor participation can promote fairness and transparency in collaborative activities.	4.48	Strongly Agree
MPS3. Integrating real-time class activity workflows (e.g., educator-led coding sessions) would improve coordination during lessons.	4.51	Strongly Agree
MPS4. AI-assisted code evaluation feedback can provide effective formative guidance that helps students learn and improve their coding skills over time.	4.61	Strongly Agree
MPS5. Flagging submissions containing excessive AI-generated code is necessary to maintain academic integrity.	4.59	Strongly Agree
MPS6. Including a real-time collaborative coding mechanism (e.g., shared code editor) would improve teamwork and peer learning among students.	4.70	Strongly Agree
MPS7. Integrating an AI-assisted code evaluation would provide more meaningful and immediate feedback.	4.69	Strongly Agree
GRAND MEAN	**4.59**	**Strongly Agree**

Table 10 presented the respondents' evaluation of the Mechanisms or Techniques of the Proposed System. The results indicated a very strong level of agreement, with a grand mean of 4.59 (SD = 0.57), reflecting a highly positive perception of the proposed features of UnifIDE. Respondents strongly agreed that incorporating a contribution- tracking function would ensure fairness and accountability in group programming activities (M = 4.58, SD = 0.58). This supports the findings of [21], who emphasized that integrating objective indicators of participation is necessary to reveal disparities in effort distribution. Similarly, the use of learning analytics to monitor participation was perceived to promote transparency in collaborative tasks (M = 4.48, SD = 0.59). The integration of structured real-time class workflows, such as instructor-led coding sessions, was also rated highly (M = 4.51, SD = 0.62), indicating support for more organized and guided collaborative learning environments. AI-assisted features received particularly strong support. Respondents agreed that AI-assisted code evaluation could provide effective formative guidance to help students improve (M = 4.61, SD = 0.56) and deliver more meaningful and immediate feedback (M = 4.69, SD = 0.49). This reflects the conclusions of [7], who found that AI-assisted review tools result in deeper engagement with code logic and more meaningful revisions. Additionally, mechanisms for flagging submissions containing excessive AI-generated code were viewed as important for maintaining academic integrity (M = 4.59, SD = 0.61). The inclusion of a real-time collaborative cod- ing mechanism, such as a shared code editor, obtained the highest rating (M = 4.70, SD = 0.49), emphasizing the strong demand for synchronous collaboration tools. Overall, the findings demonstrated strong support for the pro- posed mechanisms of UnifIDE, particularly in enhancing collabo- ration, fairness, feedback quality, and academic integrity. The consistently high mean ratings reinforced the relevance and potential effectiveness of the system's core features in addressing the limitations identified in existing platforms. The students' strong endorsement of these mechanisms, particularly AI-assisted evaluation, reflects research showing that intelligent, formative guidance helps students identify semantic errors and deeply engage with code logic during revisions [7].

3.4 Top/Key Features of the Proposed System

This section presented respondents' evaluation of UnifIDE's primary features, including engagement incentive elements (points, badges, and leaderboards), AI-assisted scoring and feedback, participation and progress dashboards, multilingual support, browser-based accessibility, and the integration of collaboration, engagement incentive systems, and AI-assisted evaluation. The findings identified which features were perceived as most valuable in enhancing motivation, engagement, transparency, and overall learning effectiveness, while also highlighting areas for further refinement to improve usability and instructional impact.

TABLE 11. TOP/KEY FEATURES OF THE PROPOSED SYSTEM

Feature	Mean	Description
TKF1. Engagement features would encourage consistent participation and deeper engagement in programming activities.	4.58	Strongly Agree
TKF2. The scoring and feedback generated by the system would help users understand performance and identify areas for improvement.	4.49	Strongly Agree
TKF3. Participation and progress dashboards would help improve transparency and monitoring during collaborative programming activities.	4.52	Strongly Agree
TKF4. Supporting multiple programming languages would enable UnifIDE to adapt to various courses and levels of programming proficiency.	4.59	Strongly Agree
TKF5. Accessing the platform directly via a web browser (without installing software) is essential for easy classroom adoption.	4.53	Strongly Agree
TKF6. Integrating engagement incentive elements such as points, badges, and leaderboards would motivate students to participate actively.	4.65	Strongly Agree
TKF7. Combining collaboration, engagement incentive, and AI-assisted evaluation would make UnifIDE a more comprehensive and effective educational coding platform.	4.74	Strongly Agree
GRAND MEAN	**4.59**	**Strongly Agree**

Table 11 presented the respondents' evaluation of the Top/Key Features of the Proposed System. The findings indicated a very high level of agreement, with a grand mean of 4.59 (SD = 0.57), suggesting strong overall support for the core features of UnifIDE. Respondents strongly agreed that engagement incentive features would encourage consistent participation and deeper engagement in programming activities (M = 4.58, SD = 0.57). They also affirmed that system-generated scoring and feedback would help users better understand their performance and identify areas for improvement (M = 4.49, SD = 0.57). Participation and progress dashboards were likewise viewed as important for improving transparency and monitoring in collaborative programming (M = 4.52, SD = 0.62). Support for technical flexibility was also evident, as respondents strongly agreed that supporting multiple programming languages would allow UnifIDE to adapt to various courses and levels (M = 4.59, SD = 0.59). Accessibility was emphasized by agreement that a browser-based platform, without installation, was essential for classroom adoption (M = 4.53, SD = 0.61). Furthermore, integrating engagement incentive elements such as points, badges, and leaderboards were perceived to motivate student participation (M = 4.65, SD = 0.51). This mirrors the research of [10], which proved that gamified learning systems successfully cultivate intrinsic motivation when properly designed. Notably, the highest-rated item was the integration of collabo- ration, engagement incentive systems, and AI-assisted evaluation into a single system (M = 4.74, SD = 0.47), indicating that respondents viewed the combined approach as highly comprehensive and effective. Overall, the consistently high ratings demonstrated strong endorsement of the proposed system's key features, reinforcing the relevance and potential impact of UnifIDE in enhancing programming education. The high value students placed on features like points and leaderboards confirms that well-designed gamified systems leverage immediate feedback and social comparison to effectively cultivate a student's intrinsic motivation [10].

3.5 Real-Time Collaboration via CRDT (Yjs with WebSocket)

This section presented the empirical evaluation of UnifIDE's real-time collaboration mechanism, focusing on the responsiveness and consistency of its CRDT-based synchronization engine through controlled simulation. Synchronization latency was measured by recording the time elapsed between a local edit made by one client and its appearance on a remote client across five trials of sequential operations. Simultaneous edit convergence was then evaluated by having all clients fire edits at the exact same moment, measuring the time required for all replicas to reach an identical document state across two workspace scenarios — the small group collaborative workspace and the whole class collaborative workspace—under a simulated network delay of 50 milliseconds reflecting typical Philippine broadband and mobile internet latency conditions. Replica consistency was assessed by comparing the final document states of all connected clients following each sequence of simultaneous operations. Together, these tests assessed whether UnifIDE's collaboration engine was both fast enough for real-time use and correct under true concurrent editing conditions. Synchronization Latency. Table 12 presented the synchronization latency measurements recorded across five trials of 20 sequential operations each, distributed across five simulated clients connected to UnifIDE's Yjs WebSocket server, reflecting the maximum capacity of the small group collaborative workspace.

TABLE 12. CRDT SYNCHRONIZATION LATENCY RESULTS

Trial	N (ops)	Mean Latency (ms)	Consistent?
1	20	1.28	Yes
2	20	1.51	Yes
3	20	0.97	Yes
4	20	1.53	Yes
5	20	1.54	Yes
Overall	**100**	**1.37**	**5/5**

Table 12 presented the synchronization latency of UnifIDE's CRDT-based collaboration engine across five controlled trials. The overall mean latency of 1.37 ms across 100 operations confirmed that individual edits propagated to remote clients in near real-time, well below the 100 ms thresholds generally recognized as the boundary for imperceptible response. Per-trial mean values ranged from 0.97 ms to 1.54 ms, reflecting stable and consistent performance across all runs. The elevated maximum of 8.67 ms observed in Trial 1 was attributable to initial WebSocket connection overhead during the opening operation of the session, after which latency stabilized into the sub-2 ms range for all subsequent operations. The P95 latency of 5.09 ms confirmed that 95% of all recorded operations propagated in under 5.09 ms.

Simultaneous Edit Convergence. Table 13 presented the simultaneous edit convergence results across both workspace scenarios under a simulated network delay of 50 milliseconds. In each test, all users fired their edits at the exact same moment, and convergence time was measured as the duration until every connected user reached an identical final document state.

TABLE 13. SIMULTANEOUS EDIT CONVERGENCE RESULTS

Users	Conv. Time (ms)	Assessment	Consistent?
5	7.28	Excellent	Yes
10	39.55	Excellent	Yes
15	46.64	Excellent	Yes
20	66.92	Good	Yes
25	104.14	Acceptable	Yes
30	162.14	Acceptable	Yes
40	314.36	Noticeable	Yes

Table 13 presented the simultaneous edit convergence results across both workspace scenarios under simulated Philippine net- work conditions. The small group collaborative workspace at its maximum capacity of five simultaneous users achieved a convergence time of 7.28 ms, classified as Excellent, confirming that even when all five members type at the exact same moment, all replicas reach an identical document state in well under 50 ms — a duration imperceptible to human users. For the whole class collaborative workspace, a consistent performance pattern emerged across all tested user counts. Sessions of up to 15 simultaneous users produced Excellent convergence times ranging from 39.55 ms to 46.64 ms, indicating near-instantaneous synchronization with no perceptible delay. At 20 users, convergence time entered the Good range at 66.92 ms, indicating a smooth editing experience with synchronization overhead imperceptible to most users. Sessions of 25 and 30 users produced Acceptable convergence times of 104.14 ms and 162.14 ms respectively, remaining functional and correct but introducing slight perceptible delay during peak simultaneous editing bursts. At 40 simultaneous users, convergence time crossed into the Noticeable range at 314.36 ms, representing the point at which synchronization overhead becomes perceptible during heavy simultaneous editing. Maintaining near-instantaneous synchronization below this threshold is critical, as seamless real-time cooperation has been shown to significantly en-hance programming performance and intention to learn [4]. Table 14 summarized the convergence threshold findings and their practical implications for classroom use.

TABLE 14. CONVERGENCE THRESHOLD SUMMARY

Users	Assessment	Practical Implication
5	Excellent	Near-instantaneous sync, no perceptible delay
10–15	Excellent	Near-instantaneous sync, no perceptible delay
20	Good	Smooth experience, minor overhead im-perceptible to users
25–30	Acceptable	Functional and correct, slight delay dur-ing peak editing bursts
40	Noticeable	Perceptible delay during heavy simulta-neous editing

Replica Consistency. Replica consistency was confirmed across all tested combinations—every trial in the synchronization latency test and every user count in both

workspace scenarios of the simultaneous edit convergence test — without a single exception. In every test, all connected users reached an identical final document state upon completion of all operations, satisfying the convergence condition State1 = State2 = · · · = Statek. No data loss, duplication, or corruption was observed across any of the tests regardless of the number of simultaneous users or the volume of concurrent edits. This result empirically validated that UnifIDE's integration of Yjs, WebSocket transport, and editor bindings produced no divergence in practice, confirmed that the CRDT mechanism reliably resolved all concurrent conflicts and ensured a shared, coherent document state under all tested conditions. The perfect consistency achieved across all scenarios is in direct accord with the mathematical convergence guarantees defined for CRDTs by [22], reinforcing that Yjs-based implementations reliably satisfy the strong eventual consistency property in practice.

3.6 Model Evaluation Data Treatment

This section presented the evaluation results of the fine-tuned UniX-coder binary classifier integrated into UnifIDE's AI code detection module. Evaluation was conducted on a held-out test set of 43,464 samples — drawn from a stratified 80/20 split of the 217,318-sample dataset by programming language and class label — that were not seen during training. The test set was nearly balanced, comprising 21,821 human-written samples (50.2%) and 21,643 AI-generated samples (49.8%), ensuring that reported accuracy and averaged metrics were not distorted by class imbalance. Confusion Matrix. The confusion matrix of the classifier evaluated at the F1-maximizing decision threshold of 51.4% on the 43,464-sample test set correctly identified 21,139 human-written submissions as human (True Negatives) and 20,404 AI-generated submissions as AI (True Positives). Of the 682 False Positives, these represent human-written submissions incorrectly flagged as AI- generated, yielding a False Positive Rate (FPR) of 3.13%. The 1,239 False Negatives represent AI-generated submissions that evaded detection, yielding a False Negative Rate (FNR) of 5.73%. The higher count of false negatives relative to false positives reflected an intentional design orientation: the threshold was tuned to minimize false positives, prioritizing the protection of legitimate student submissions over maximizing AI detection recall. This was consistent with UnifIDE's resubmission mechanism, which allowed flagged students to revise and resubmit without incurring a permanent academic penalty. This threshold calibration strategy reflects the study's decision to minimize false positives and protect legitimate student submissions from wrongful flagging, while accepting a possible trade-off in the detection of AI-generated samples.

TABLE 15. CLASSIFICATION REPORT OF THE FINETUNED UNIXCODER

Class	Precision	Recall	F1-Score
Human	0.9446	0.9687	0.9565
AI	0.9677	0.9428	0.9550
Macro Average	**0.9561**	**0.9557**	**0.9558**
Weighted Average	**0.9561**	**0.9558**	**0.9558**
Overall Accuracy	**0.9558 (95.58%)**		

The classifier achieved an overall accuracy of 95.58%, correctly classifying 41,543 of the 43,464 test samples. For the AI class—the primary detection target—precision was 0.9677, indicating that 96.77% of all submissions flagged as AI-generated were genuinely AI-generated. Recall for the AI class was 0.9428, meaning the model successfully identified 94.28% of all AI-generated submissions in the test set. For the human class, precision was 0.9446 and recall was 0.9687. The higher recall for the human class (0.9687 versus 0.9428 for the AI class) confirms the deliberate design orientation toward minimizing false positives—the classifier is more conservative in flagging submissions as AI-generated, allowing a greater proportion of human submissions to pass correctly at the expense of slightly more missed AI detections. The macro average F1-score (0.9558) and weighted average F1-score (0.9558) are identical, a direct result of the nearly balanced class distribution in the test set. In a balanced setting, both averag- ing methods converge, eliminating any ambiguity that would arise from class imbalance inflating the weighted metric. By tuning the model to protect legitimate student submissions from false positives, the system ensures that AI detection relies on rigid structural patterns rather than superficial text, creating a fairer assessment environment for the student [23].

3.7 Findings of the Study

The study revealed that UnifIDE effectively addressed the identified limitations of existing educational coding platforms, particularly regarding the lack of real-time collaboration support, shallow feed- back mechanisms, limited transparency in group participation, and insufficient tools for maintaining academic integrity. Using a descriptive quantitative design and Likert-scale analysis, responses from both students and faculty demonstrated a strong consensus on the weaknesses of current platforms. Respondents strongly agreed that most existing coding platforms were oriented toward individual work rather than collaborative programming, that feedback was largely limited to correctness and test case results, and that mechanisms for detecting AI-generated code submissions remained inadequate. These findings confirmed that the gaps UnifIDE aimed to address were genuine and widely perceived among its intended users within the College of Computer Studies. This is backed up by Gupta and Goel [2], who discussed limitations of existing online coding platforms in the context of collaborative programming education.

Respondents expressed strong support for UnifIDE's proposed mechanisms and key features, with grand means of 4.59 across both the mechanisms and features sections. The

real-time collaborative coding mechanism and AI-assisted code evaluation were among the highest-rated proposed mechanisms, reflecting strong demand for synchronous collaboration tools and intelligent, formative feedback in programming education. Contribution tracking and participation monitoring were also strongly endorsed, underscoring the importance of fairness and transparency in group-based activities. Among the key features, the integration of collaboration, engagement incentive systems, and AI-assisted evaluation into a single unified platform received the highest rating of all survey items (M = 4.74), indicating that respondents viewed the combined approach as significantly more comprehensive and effective than any individual feature in isolation. This matches what [24] found, that platforms combining gamification and AI feedback in one place kept users more engaged than platforms that offered these features separately. Browser-based accessibility and multi-language support were likewise valued as essential for practical classroom adoption across diverse courses and proficiency levels.

The technical evaluation of UnifIDE's core computational components confirmed that both the CRDT synchronization engine and the AI-assisted code detection module were reliable and performant within their intended educational contexts. The CRDT engine maintained a mean synchronization latency of 1.37 ms and achieved Excellent to Good convergence ratings for sessions of up to 20 simultaneous users, with performance remaining Acceptable at 25 and 30 users — all within typical classroom group sizes. Convergence only became Noticeable at 40 simultaneous editors, a threshold that exceeded the maximum capacity of UnifIDE's supported workspace scenarios. Replica consistency was confirmed without exception across all tested conditions. Together, these results established a practically significant boundary: since the CRDT engine maintained reliable convergence and perfect consistency well within classroom-scale user counts, any delays experienced by smaller groups during actual deployment were attributable to external factors such as server load or client-side network instability rather than the synchronization engine itself, giving educators and administrators a clear basis for diagnosing performance issues during classroom use. This is consistent with what [22] mathematically proved, that this type of syncing system guarantees that all users will eventually see the same data without needing constant back-and- forth communication, something that this study confirmed through actual testing.

For the AI-generated code detection module, the fine-tuned UniX-coder classifier achieved an overall accuracy of 95.58% on a held-out test set of 43,464 samples. The classifier's decision threshold was deliberately tuned to minimize false positives, yielding a False Positive Rate of only 3.13% to protect legitimate student submissions from wrongful flagging. These results established that UnifIDE's intelligent and collaborative mechanisms were both technically ro- bust and aligned with the fairness and pedagogical requirements of programming education. This builds on the work of [23], who developed the UniXcoder model, and is further supported by [8], whose DROID dataset was used as a major source of training data for this study.

IV. CONCLUSION

The study concluded that UnifIDE meaningfully advances program- ming education by transforming correctness-only, individually focused coding environments into an integrated, collaborative, and analytically rich platform. Addressing the first objective, the study assessed existing coding platforms and documented their key limitations: the absence of real-time multi-user editing, shallow feedback that stopped at output correctness, no mechanisms for tracking individual contributions in group work, and no features to detect AI-generated code submissions. Through the second objective, these gaps were translated into a concrete system architecture unifying four key components—CRDT-based real-time collaborative editing, contribution analytics, structured engagement incentives, and AI-assisted code evaluation—within a single cohesive environment. The third objective was fulfilled through the development of the AI-assisted evaluation module, which combined the Gemini API for narrative code quality feedback with a fine-tuned UniXcoder model for AI-generated code classi-fication. Both components were operationalized within UnifIDE's submission workflow and validated through controlled classification and latency testing. The fourth objective of integration was achieved through a modular, scalable technology stack that kept all components seamlessly interconnected without disrupting the real-time collaboration experience. The fifth objective of implementation and testing confirmed that all system modules functioned as intended across unit, integration, alpha, and acceptance testing phases. Acceptance results confirmed that these design and development choices are pedagogically and institutionally relevant. Students and faculty alike perceived the system as more transparent, fair, and capable than existing tools, particularly valuing CRDT-powered collaboration, contribution visibility, and AI-driven evaluation. Limitations in CRDT synchronization overhead for very large files, the boundaries of AI detection for short submissions, and the absence of direct SIS integration highlight areas for continued refinement.

V. RECOMMENDATIONS

These recommendations aim to guide future researchers and developers in improving UnifIDE based on the insights and gaps identified in this study. Key areas for continued development include the following.

Expanding language and library support. Future versions should consider adding emerging programming languages such as Rust, TypeScript, and Kotlin, along with support for commonly used third-party libraries in sandboxed environments, to broaden the platform's applicability to advanced coursework. Refining the AI detection module. The UniXcoder model should be periodically retrained on updated datasets that reflect evolving AI code generation styles, especially as large language models become more sophisticated at producing human-like code. Expanding the training corpus to include more diverse coding styles and languages would further improve robustness. Deepening narrative feedback. The Gemini API-based feedback module

should be extended to provide assignment-specific and rubric-aligned commentary, allowing educators to customize the type of feedback generated based on the learning objectives of each coding task. Integration with institutional systems. Future development should explore secure API-based integration with university LMS and SIS platforms to automate student roster management, grade synchronization, and evaluation data export, reducing administrative overhead. Supporting offline or low-connectivity environments. Given the platform's reliance on a stable internet connection, future iterations should investigate offline-first synchronization strategies that al- low students to continue coding and have their edits reconciled when connectivity is restored, which is particularly relevant in the Philippine context. Conducting longitudinal impact studies. Future research should assess the long-term effects of UnifIDE on student programming skill development, collaboration quality, and academic integrity behaviors across multiple semesters, providing stronger empirical evidence for the platform's educational value.

REFERENCES

[1] Bureau of Labor Statistics, "Occupational Outlook Handbook: Software Developers, Quality Assurance Analysts, and Testers," U.S. Dept. Labor, Washington, DC, USA, 2025.

[2] R. Gupta and A. Goel, "Limitations of existing online coding platforms for collaborative programming education," ACM SIGCSE Bulletin, vol. 57, no. 1, pp. 45–52, 2025.

[3] D. Brodnitz, "The future of work: Collaborative skills in software engineering," IEEE Software, vol. 41, no. 2, pp. 22–28, 2024.

[4] C. W. Tsai, P. D. Shen, and Y. C. Chiang, "The effects of collaborative learning on programming performance and persistence in introductory CS courses," Computers & Education, vol. 195, Art. no. 104720, 2023.

[5] Y. Chang and C. Wongwatkit, "Enhancing collaborative learning through peer-assessment-driven Scrum in programming courses," Education and Information Technologies, vol. 29, no. 1, pp. 1155–1178, 2024.

[6] M. Messer, N. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," ACM Transactions on Computing Education, vol. 24, no. 1, pp. 1–40, 2024.

[7] F. Almeida, E. Cavalcante, and E. Souza, "AI-assisted code review: Enhancing student engagement and code quality in programming courses," Computers & Education, vol. 198, Art. no. 104975, 2024.

[8] Y. Orel, M. Zviling, H. Cohen, and G. Katz, "Droid: A benchmark dataset for AI-generated code detection across multiple programming languages," in Proc. 47th Int. Conf. Software Engineering (ICSE '25), 2025, pp. 1234–1245.

[9] J. Calles-Esteban, L. García-Mundo, and M. Genero, "Gamification in programming education: A systematic mapping study," Computer Science Education, vol. 34, no. 1, pp. 1–32, 2024.

[10] B. Dao-Ines, D. Saucier, and G. Clarebout, "Engagement mechanics in online learning platforms: A review of point systems and leaderboards in CS education," ACM Transactions on Computing Education, vol. 24, no. 2, pp. 1–24, 2024.

[11] W. Li, H. Chen, and T. Zhang, "Developmental-experimental design in educational technology research: A methodological review," Educational Technology Research and Development, vol. 73, no. 1, pp. 1–22, 2025.

[12] Department of Information and Communications Technology, "Trabahong Digital: Digital Jobs Program," DICT, Quezon City, Philippines, 2025.

[13] M. M. T. Rodrigo, "Bridging the gap: Industry perspectives on collaborative skills gaps in Filipino CS graduates," in Proc. 2024 ACM Conf. International Computing Education Research (ICER '24), 2024, pp. 88–96.

[14] J. Omonije, "Agile software development in educational contexts: Benefits, challenges, and best practices," Journal of Software Engineering Research and Development, vol. 12, no. 1, pp. 1–18, 2024.

[15] S. Chinta and M. Maziya, "Agile methodology in higher education software projects: A systematic review," Journal of Systems and Software, vol. 209, Art. no. 111926, 2025.

[16] A. Gannar and H. Kilani, "Constructivist learning through project-based computing education: A systematic review," Education and Information Technologies, vol. 30, no. 2, pp. 3011–3035, 2025.

[17] L. S. A. Piccolo, M. Alani, and G. Troussas, "Prior experience and platform familiarity as predictors of adoption in educational coding tools," Computers & Education, vol. 208, Art. no. 105020, 2025.

[18] T. Rogers, S. Jordan, and P. Warwick, "Early-career and experienced educators in technology adoption: Complementary perspectives on digital tool evaluation," Teaching and Teacher Education, vol. 148, Art. no. 104682, 2025.

[19] Y. Gambo, M. Shakir, and N. Osman, "Automated code assessment in programming education: Perspectives from instructors of active programming courses," Journal of Computer Assisted Learning, vol. 41, no. 1, pp. 112–128, 2025.

[20] R. P. Bringula, "Artificial intelligence as a productivity multiplier for educators: Implications for higher education," Computers & Education: Artificial Intelligence, vol. 6, Art. no. 100192, 2024.

[21] J. Hamer, C. Kell, and F. Spence, "Peer assessment and contribution tracking in group programming projects," in Proc. 2022 ACM Conf. Innovation and Technology in Computer Science Education (ITiCSE '22), 2022, pp. 312–318.

[22] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in Proc. 13th Int. Symp. Stabilization, Safety, and Security of Distributed Systems (SSS 2011), 2011, pp. 386–400.

[23] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified cross-modal pre-training for code representation," in Proc. 60th Annu. Meeting Assoc. Comput. Linguistics (ACL), 2022, pp. 7212–7225.

[24] C. Costa, P. Antunes, and J. Pino, "Integrating gamification and AI-assisted feedback in collaborative learning platforms: Effects on student engagement," Educational Technology & Society, vol. 27, no. 2, pp. 45–61, 2024.

[25] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard—or at least it used to be: Educational opportunities and challenges of AI code generation," in Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE '23), 2023, pp. 500–506.

[26] Commission on Higher Education, "CMO No. 25 s. 2015: Revised Policies, Standards and Guidelines for Bachelor of Science in Computer Science," CHED, Quezon City, Philippines, 2015.

[27] M. E. Ferrão, L. Almeida, and A. Barros, "Predictors of academic success in computer science: A longitudinal study," Computers & Education, vol. 205, Art. no. 105001, 2025.

[28] Q. Guo, B. Chen, J. Shen, and F. Wu, "Evading AI code detectors: Analysis of surface-level transformations and robustness challenges," IEEE Transactions on Software Engineering, vol. 51, no. 3, pp. 789–804, 2025.

[29] IT and Business Process Association of the Philippines, "IBPAP Roadmap 2028: State of the IT-BPM Industry in the Philippines," IBPAP, Taguig City, Philippines, 2025.