Human Factors, Business Management and Society, Vol. 176, 2025, 228–241
https://doi.org/10.54941/ahfe1006320

Combining AI Tools, Low-Code Platforms, and Product Development in ICT Education: A Reflective Study on Educational and Practical Outcomes

Noora Nieminen[1] and Tero Reunanen[2]

[1]Turku University of Applied Sciences, Turku 20520, Finland
[2]University of Vaasa, Vaasa 65200, Finland

ABSTRACT

Programming education often faces high dropout rates and steep learning curves. This study explores integrating low-code/no-code (LCNC) platforms with AI tools in an introductory programming course structured around the CDIO (Conceive, Design, Implement, Operate) framework. Over 200 first-year engineering and business students engaged in project-based learning using AI-assisted LCNC platforms. Analysis of student feedback reveals improvements in engagement, creativity, problem-solving, and accessibility. However, challenges such as rapid pacing and instructor familiarity with platforms were noted. Findings suggest AI-driven LCNC approaches effectively support novice programmers and interdisciplinary collaboration. Future research should examine long-term impacts on programming competency and alternative assessment strategies.

Keywords: AI-assisted software development, Low-code/no-code platforms, ICT education, Collaborative learning

INTRODUCTION

Programming education is crucial for preparing future technologists, yet traditional introductory courses often suffer from high failure and dropout rates – in some cases up to 30–50% of students withdraw or fail (Margulieux et al., 2020). Common challenges include steep learning curves, student disengagement, and a persistent gap between abstract theory and practical application (Nieminen & Reunanen, 2024). Beginners must master complex problem-solving processes and new syntax simultaneously, which can be overwhelming when instructors' expertise has become tacit knowledge that is not easily transferred (Margulieux et al., 2020). These difficulties have prompted educators to seek more accessible and engaging approaches to teach computational thinking and programming fundamentals.

In recent years, low-code/no-code (LCNC) development platforms and artificial intelligence (AI) tools have emerged as promising avenues to lower the barriers to entry in programming. Low-code platforms minimize the need for textual coding by providing visual interfaces and pre-built

components, enabling even non-specialists to create functional applications with minimal coding experience (Tsakalerou et al., 2024) (Hirzel, 2023). This democratization of software development has generated enthusiasm in industry, for example, Gartner (2021) predicts that by 2025, 70% of new enterprise applications will be developed using low-code or no-code tools. In parallel, advances in AI (such as generative code assistants and chatbots) are fueling these trends, offering more intuitive development experiences (Hirzel, 2023). AI-powered programming assistants can provide personalized, instant support to learners, helping bridge the gap between concept and implementation through interactive guidance (Nieminen & Reunanen, 2024). Taken together, AI and LCNC technologies have the potential to reshape programming education by reducing the technical overhead, cognitive load, and frustration often faced by novices, while enabling creative exploration of real-world problems.

Against this backdrop, our research examines the integration of novel AI-driven LCNC platforms into a university-level introductory course as a means of renewing programming education. In particular, we present a case study of a first-year course (enrolling over 200 engineering and business students) structured around the CDIO educational framework (Conceive, Design, Implement, Operate) and incorporating project-based learning with AI-assisted low-code tools. The study analyzes both educational outcomes and practical experiences from the course's initial implementation. We focus on the course's impact on student engagement, collaboration, creativity, and problem-solving skills, and evaluate whether AI-powered LCNC platforms can serve as a more accessible entry point to programming for beginners. By reflecting on student feedback and instructor observations, we identify successes, challenges, and lessons learned, laying the groundwork for future improvements and research (including a planned follow-up course integrating AI-assisted programming for novices).

LITERATURE REVIEW

Low-Code/No-Code Platforms in Education

LCNC development platforms have evolved from earlier visual programming environments (such as block-based coding tools) into powerful professional tools, and they are increasingly being considered in educational contexts. A low-code platform provides a high level of abstraction – users manipulate graphical components and workflows instead of writing extensive code – which accelerates application development and reduces the effort required (Huang & Li, 2022). This high-level approach can complement traditional coding education by allowing students to focus on problem-solving logic and design, rather than syntax errors. In essence, low-code platforms "enable contributions from non-specialist users" and "resonate with the pedagogical goal of fostering computational thinking among students" (Tsakalerou et al., 2024). By simplifying the development process, LCNC tools may enhance learners' abstraction skills and algorithmic thinking. Studies have noted that low-code environments can foster innovation, creativity, and even entrepreneurial mindset in students by empowering them to rapidly prototype

solutions to real-world problems (Tsakalerou et al., 2024). Moreover, the collaborative features of many LCNC platforms (e.g. shared visual models, integrated version control, and modular components) can facilitate teamwork and communication in student projects (Huang & Li, 2022). These benefits align with longstanding educational aims to engage a broader diversity of learners in computing. LCNC platforms essentially extend the concept of "end-user programming," enabling "citizen developers" – individuals with little formal coding background – to create software, while also aiding professional developers in being more productive (Hirzel, 2023). The motivation for adopting low-code in education includes not only making programming more accessible, but also giving students the satisfaction and "joy of creating something useful" and thinking in a computational way (Hirzel, 2023). Recent work has begun to examine the pedagogical outcomes of low-code integration. For instance, Huang and Li (2022) propose that features of low-code platforms (such as visual modeling and instant feedback) can facilitate knowledge creation and absorption in information systems students (Huang & Li, 2022). Early evidence suggests that when used as an introductory tool, low-code development can serve as a stepping stone to traditional programming by building confidence and foundational understanding in a gentler learning curve (Tsakalerou et al., 2024). However, the academic literature on LCNC in programming education is still nascent, and further empirical studies are needed to validate its efficacy across different learning contexts.

Low-Code/No-Code Platforms in Education

The difficulties of learning to program from scratch are well documented. Beginners often struggle with abstract concepts, logical structures, and unfamiliar syntax simultaneously, leading to frustration and attrition (Nieminen & Reunanen, 2024). A lack of immediate, personalized feedback in large classes can exacerbate the situation – students may get stuck on bugs or misconceptions for long periods, contributing to feelings of inadequacy or anxiety. Standard introductory teaching methods (e.g. lectures on theory followed by coding assignments) sometimes fail to sustain engagement or to connect with real-world applications, leaving students questioning the relevance of what they are learning (Nieminen & Reunanen, 2024). Furthermore, the one-size-fits-all pace of a syllabus can alienate students at both ends of the spectrum: slower learners fall behind, while advanced or motivated students might get bored with basic examples. To address these issues, educators have explored various strategies: pair programming to improve confidence, game-based assignments to increase motivation, and visualization tools to clarify abstract concepts (e.g. memory models) (Nieminen & Reunanen, 2024). Another major challenge is the cognitive load on novices – they must decompose problems and formulate algorithmic solutions while also mastering syntax and tools. This has led to interest in pedagogies that *reduce extraneous cognitive load*, for example by using block-based or visual languages initially. Low-code/no-code platforms can be seen as an extension of this idea, removing the burden of syntax and

setup and allowing beginners to construct programs through more intuitive means. However, one concern is whether such environments sufficiently teach transferable programming skills or if students might develop a shallow understanding. This is an ongoing debate: some argue that starting with visual/low-code tools can build confidence and contextual understanding that eases the transition to text-based coding, while others caution that students still need exposure to "real code" to fully grasp computational thinking (beyond the simplified models).

AI-Driven Solutions for Novice Programmers

In parallel with LCNC developments, the integration of artificial intelligence into computing education has accelerated, offering new ways to enhance algorithmic thinking and personalized learning. AI-driven tutoring systems and coding assistants have shown promise in addressing some of the aforementioned challenges. For example, intelligent tutoring systems (ITS) can provide step-by-step guidance or hints tailored to a student's current state of understanding, effectively scaling the availability of one-on-one help (Nieminen & Reunanen, 2024). Modern AI chatbots (such as those powered by large language models) can answer student questions on demand, explain code, or suggest solutions, serving as a round-the-clock supplement to human instructors (Nieminen & Reunanen, 2024). Recent case studies indicate that AI programming assistants – like GitHub Copilot or ChatGPT – can "significantly improve the learning experience" in introductory programming by helping students overcome roadblocks and reducing anxiety. Nieminen & Reunanen (2024) observed that integrating generative AI tools in a first-year Python course eased students' understanding of basic concepts, provided personalized help on advanced tasks, and even enabled the class to cover more complex topics earlier than usual. The AI would suggest code or debugging tips, which students could then evaluate and learn from, thereby also training their critical thinking about code correctness. Such AI support can lower the entry barrier for difficult concepts and act as a form of scaffolding, allowing novices to progress further before hitting frustration. Importantly, prior studies have also noted that students need guidance on how to effectively use AI assistance – e.g. recognizing that AI suggestions might be imperfect and learning to validate and adapt them (Nieminen & Reunanen, 2024). When used appropriately, AI assistants can encourage learners to explore beyond the curriculum; for instance, students with access to AI have ventured into topics like databases or cryptography out of curiosity, going beyond class requirements. On the other hand, there are pedagogical concerns about over-reliance on AI (if students accept AI-generated solutions without understanding them) and issues of academic integrity that educators must address. Overall, the convergence of AI and LCNC tools in education represents a novel frontier – AI can enhance low-code platforms (through features like natural language code generation or automated guidance), and low-code platforms provide a sandbox where AI suggestions can be immediately applied and tested by novices. This synergy is anticipated to make programming education more dynamic, responsive,

and inclusive, as it can adapt to individual learning needs and open the field to students from non-traditional backgrounds. Our study builds on this emerging literature by examining an educational intervention that explicitly combines these two elements (AI and LCNC) in an introductory course setting.

METHODOLOGY

Course Context and Design

The study is based on a new undergraduate course titled "Low-Code Software Development Basics," designed for first-year students in engineering and business programs at a university of applied sciences. The course had an enrollment of over 200 students (split into smaller sections, including both Finnish-speaking and international groups). It was structured according to the CDIO framework, an innovative educational model that emphasizes learning through the lifecycle of product development: Conceive, Design, Implement, and Operate. The CDIO approach was chosen because of its proven benefits in engineering education – it situates learning in a practical project context and has been shown to improve students' domain knowledge as well as cognitive and interpersonal skills (e.g. problem-solving, creativity, teamwork) (Taajamaa et al., 2016). Instead of a traditional lecture/exam format, the course was delivered through hands-on projects and active learning exercises aligned with the CDIO stages. Students worked primarily in teams (3–5 members per team) to encourage peer learning and collaboration, reflecting the "cooperation and communication skills" development that CDIO-based courses often target (Taajamaa et al., 2016).

Integration of AI-Powered LCNC Platforms

A distinguishing feature of the course was the incorporation of several low-code/no-code development platforms, augmented by generative AI tools, into the learning activities. The goal was to let students *conceive* an application idea, *design* and *implement* it using accessible LCNC tools (with AI support for guidance and ideation), and then *operate* it (testing, demonstrating, and reflecting on the solution). Over the semester, students were introduced to a range of LCNC platforms covering different domains of software development: for example, a mobile app builder (FlutterFlow), a web application builder, and a game development platform (the Godot engine with visual scripting). Each platform was chosen to correspond with a module of the course, so that students gained exposure to building mobile apps, web/desktop apps, and simple games without traditional coding. These platforms typically provided drag-and-drop interfaces, visual logic editors, and pre-built components which allowed students to assemble functional prototypes quickly. However, they also presented learning challenges, as students needed to grasp the platform's concepts and limitations within a short time. To assist with this, we integrated AI-based tools in multiple ways. During the **Conceive** phase of projects, students were encouraged to use AI brainstorming assistants (such as ChatGPT) to help generate and refine

ideas for applications to solve everyday problems. Many teams engaged with the AI to iterate on their project concepts, user stories, and even to outline requirements, effectively using it as a creative partner. In the **Design** phase, some platforms offered AI features (for instance, an AI that can suggest UI layouts or color schemes), and students could leverage those when available. For general design thinking support, the course provided access to an AI assistant that could answer questions about design principles or suggest improvements to user interface mock-ups. During the **Implement** phase, AI coding assistants were available especially for the more code-oriented tasks (e.g., if using Godot, students could get help writing or debugging simple scripts via an AI pair programmer). Even in purely no-code environments, students could query the AI with *"How do I do X in this tool?"* and often receive useful tips or instructions, thereby reducing time spent searching through documentation. The teaching team explicitly trained students in prompt engineering and how to critically evaluate AI outputs – an essential skill to ensure they remained in control of the development process. Lastly, in the **Operate** phase, students deployed their prototypes (where possible, within free-tier limits) and demonstrated them in a final presentation session. AI tools were brought into the reflection process too: for example, students used an AI text analyzer to help summarize user feedback from testing their apps, and to generate suggestions for future improvements, mimicking how AI might be used in professional product development cycles.

Course Structure and Activities

The course ran for one semester (15 weeks). In the first weeks, students were given a crash course on the LCNC platforms and the concept of AI assistance in development. Small warm-up exercises were conducted (such as a tutorial on building a simple UI in the app builder, with an AI chatbot available for help). Following this, the course was divided into four mini-projects aligned with CDIO stages: each mini-project lasted 2–3 weeks and focused on one platform/technology: for instance, Project 1 – conceive and design a simple data-entry application using a no-code database app; Project 2 – implement a mobile app for a specific use case in FlutterFlow; Project 3 – implement a simple game using Godot's visual scripting; Project 4 – operate (test and refine) one of the previous projects or integrate features. Each project required a short documentation or reflection deliverable, where students described their design process, the AI assistance they utilized, and the outcome. Agile methodologies were lightly incorporated – teams followed an iterative development approach with weekly stand-up meetings and rapid prototyping, mirroring industry practices on a small scale. The instructor's role was primarily as a facilitator and coach. During lab sessions, instructors circulated among teams to answer questions and give feedback, but students were also expected to self-direct their learning, using resources such as tutorials, community forums, and of course the AI assistants. This approach was intended to foster personal and team productivity as well as self-regulation skills, in line with the course's aim to develop not just technical proficiency but also 21st-century skills like independent learning and collaboration.

Data Collection

To evaluate the outcomes of this course implementation, we collected data from multiple sources. The primary source was a post-course feedback survey administered to all students. This survey included quantitative Likert-scale questions and open-ended questions, aiming to capture both the overall satisfaction and specific experiences of students. Key survey items asked students to rate statements such as "The course was overall successful," "The workload was appropriate for the credits," "The teaching methods (low-code tools and AI support) helped my learning," and "I achieved the learning objectives of this course," on a 5-point scale from Strongly Disagree (1) to Strongly Agree (5). Other sections of the survey probed aspects like the usefulness of group work, the clarity of assessment, and the ease of accessing course materials (the latter to ensure any technical issues with tools were noted). Two open-ended prompts were included: (1) "What worked well in this course, and how would you improve it?" and (2) "Additional free-form feedback or comments." These allowed students to freely describe their likes, dislikes, challenges, and suggestions. Out of the ~200 students, 53 responded to the survey (a response rate of roughly 26%, which, while modest, provided a diverse sample including both engineering and business students from different language groups). In addition to the survey, instructors documented their observations throughout the course (e.g., common issues teams faced, notable successes, and any interventions needed). We also retained samples of student work (with consent), such as project reports and examples of AI interactions, to qualitatively assess how students were using the AI and LCNC tools in practice. The analysis presented in this paper focuses on the survey feedback (quantitative and qualitative) supplemented by these instructor insights.

STUDENT FEEDBACK ANALYSIS

Quantitative Feedback

The numeric responses from the student survey give an initial overview of how the class perceived the AI-driven LCNC approach. Overall, student satisfaction was fairly positive. When asked to evaluate the course holistically, a majority of respondents agreed that "the course was successful overall" – the average rating for this statement was 4.12 out of 5, indicating general agreement. In fact, approximately 76% of the students gave a rating of 4 ("Agree") or 5 ("Strongly Agree") on this item. Students also felt that the learning objectives were clear, with a mean rating of 4.05/5 for that statement. This suggests the course's goals and expectations (e.g. understanding basic software development processes, learning to use LCNC tools) were communicated effectively despite the unconventional format.

However, the feedback also highlighted some issues. The statement "The student's workload was appropriate for the course's credit extent" received a lower average of 3.76/5. This more lukewarm score (with a substantial 34% responding "Neutral" and about 12% disagreeing) signals that many students found the workload heavy relative to the credits earned. Similarly,

"The course's assessment criteria were clear throughout" had an average of 3.73/5, with about 15% disagreeing (choosing 1 or 2 on the scale). This indicates some confusion about grading and expectations, which was echoed in the written comments. Students were moderately positive about the pedagogy: "Teaching methods supported my learning" averaged 3.78/5. While about 68% agreed, a notable minority (17%) disagreed, reflecting a split in how students felt about the unconventional teaching approach (which relied on projects, group work, and self-guided tool learning). Interestingly, one of the highest consensus ratings was for "My own effort contributed to my learning", which averaged 4.3/5 – most students acknowledged that their personal initiative was crucial, likely because the course required significant independent (and team) work with new tools.

Beyond these general items, other survey questions probed specifics like teamwork and resources. Although detailed breakdowns are omitted here for brevity, it's worth noting that statements regarding teamwork (e.g. "Group tasks supported my learning") had mixed responses. Some students strongly agreed that working in teams was beneficial, whereas others were neutral or disagreed – an observation that aligns with the polarized sentiments seen in the open comments about group work. Likewise, when asked if they would recommend the course to others (a proxy for overall satisfaction), most respondents indicated they would, but a few emphatically said they would not (likely those who struggled the most). The quantitative data, in summary, paints a picture of a class that on average appreciated the innovative course but also felt the strain of its demands and some ambiguity in execution.

Qualitative Feedback – Thematic Analysis

The open-ended survey responses provide rich insight into the students' experiences, highlighting what they found most valuable and what challenges they encountered. We performed a thematic analysis of 53 comments (some students provided feedback in English, others in Finnish; non-English comments were translated for analysis). Several key themes emerged.

1) Hands-on Creativity and Broad Exposure: Many students praised the course for its practical, creative approach and the broad exposure to different development platforms. For instance, one student wrote, "The course is extremely useful. Learning how applications, websites, games are developed in a nutshell gave me a lot of insight. The 'learn by ourselves by doing' teaching method is great and benefited me a lot." This sentiment of learning by doing was common among positive comments – students enjoyed building actual projects (even if small) in various domains. Another student highlighted the novelty of this experience, calling the course "creative and useful". Unlike traditional programming classes that might stick to console-based exercises, this course let beginners dabble in making GUIs, mobile apps, and games right away, which some found highly motivating. The excitement of "trying different development platforms" and seeing tangible results (a running app or game) gave students a sense of accomplishment. Importantly, several respondents noted that the course boosted their confidence. Even those with no prior programming experience were able to

contribute to software projects. As one business student noted, "I now feel more confident that I can be part of technology projects, even with limited coding experience." This suggests that the AI-assisted low-code approach may serve as an empowering introduction to computing for non-traditional audiences. It demystified software development to an extent – showing that one can start creating useful applications quickly, which may encourage these students to continue learning more technical skills in the future.

2) **High Workload and Pacing Challenges**: Despite the positive remarks on what they learned, a significant theme was that the course felt overloaded and fast-paced. Students frequently mentioned the large number of assignments and the short time frame for each. One candid comment stated, "I actually spent at least half of my study time this semester just for this one course. It's NOT FAIR to the other courses!" – underscoring that the workload was perceived as disproportionate to the credit points. Indeed, the course crammed in multiple platforms and a final project; some students felt they were "never [able to] get a break from the endless tasks." The one-week per project rhythm (for some of the tool-specific assignments) was criticized as too hurried: "One week to learn a whole app is way too short to make something meaningful," wrote one student. Consequently, some felt that they only scratched the surface of each tool without fully mastering any: "We went through so many different programs, each only briefly… We didn't have time to properly delve into any of them." This aligns with cognitive load theory – introducing several complex tools in succession can overwhelm novices. The implication is that future iterations might reduce the number of tools covered and allow more time on each, a suggestion explicitly made by a few students (e.g. "I would maybe limit the number of no-code/low-code tools on the course to three instead of four"). In addition to the volume of work, deadline management was a noted issue. A couple of comments in Finnish pointed out that deadlines were shifted multiple times, which, while intended to help students, ended up demotivating some and causing confusion about scheduling. In summary, though students valued learning a lot in a short time, many felt overburdened, indicating a need to calibrate the workload and pacing.

3) **Support and Guidance (Role of AI and Instructors)**: Given the self-driven nature of the course, students' ability to effectively use resources was crucial. A dominant theme was the lack of sufficient guidance on using the new tools. While the course encouraged independence (and the use of AI assistants), some students expected more direct teaching, especially for the more complex platforms like the game engine. One student complained, "New platform every week but no real teaching. Need someone to actually show us how to use the platforms to achieve the required results. More help with technical stuff would be good." This indicates that not all students were able to utilize the AI or documentation to fill the gaps – they desired traditional instruction (demonstrations, tutorials) to get started. In a similar vein, another student noted, "There was a lot of work and it required quite a bit of self-study on YouTube to get the applications working. In my opinion, the teaching should cover how to use those programs… not [leave it] that YouTube teaches on students' own time." Interestingly, despite the course's

emphasis on AI tools, very few student comments explicitly mentioned AI. This could imply that some students did not heavily use the AI assistants or did not find them noteworthy to mention – possibly because they resorted to familiar sources like YouTube tutorials when stuck. However, one student's feedback (from the instructor's notes) mentioned that they found using ChatGPT helpful for troubleshooting but had to remember "not to blindly trust it." A noteworthy issue raised by a couple of students was that even the instructors or tutors seemed unfamiliar with the specifics of some platforms: "It was unfortunate that our tutor teacher didn't really know the things taught in the course, so we couldn't get help or hints except from friends." This highlights a challenge when introducing novel tech into a curriculum – faculty must also be well-versed or it can undermine student support. On a positive note, those who did leverage the AI tools or external resources successfully commented that they learned a lot by teaching themselves, which is a valuable skill. But clearly, finding the right balance of guidance is critical. The course might need to incorporate more structured onboarding for each tool (perhaps short instructor-led workshops or official tutorials) to ensure students aren't left floundering early on.

4) Collaboration and Teamwork: Feedback on teamwork was mixed, revealing both benefits and drawbacks. Many students acknowledged that working in teams was a realistic and useful aspect of the course. They had opportunities to share knowledge and see different approaches to problem-solving, which can deepen learning. One student reflected that discussing ideas within their team and dividing tasks according to each member's strength helped them learn more efficiently than working alone. Additionally, a few students mentioned that the interdisciplinary mix of engineering and business students was interesting – business students contributed domain ideas and design perspectives, while engineering students tended to handle more technical configuration, illustrating a microcosm of real-world software teams. However, other students were less enthusiastic about the emphasis on group work. A particularly harsh comment described an "obsession with group work" and implied that it allowed some students to coast or that it compensated for "lazy teaching." There were complaints that not all team members contributed equally (a perennial issue in group projects), and frustration when one's learning depended on others' engagement. The survey did not explicitly measure team dynamics, but these comments suggest that while collaboration was intended to be a course strength, it requires careful implementation (clear roles, peer evaluation, etc.) to avoid negative experiences. In the final project's context, collaboration took on an element of competition as well – since the course ended with a showcase (or "fair") where projects from different groups were presented, some students felt there was unequal preparation. A few international students voiced that the communication about the final presentation event was insufficient, especially for the English section. One lengthy comment detailed that "we were told how the [final demo day] would be, only the evening before. … It would have been beneficial to know how big the event was going to be and that we were actually competing with each group (English and Finnish side). Some groups had amazing games and apps (Finnish side), while we were told that

a basic working project will be enough." This student felt it was unfair that the Finnish groups seemed better informed and perhaps had more instructor involvement, leading to more polished final projects. While this is more of an administrative issue, it underscores that transparency and equal support for all sections are vital, otherwise students can feel disadvantaged.

5) **Perceived Educational Value and Suggestions**: Many students offered suggestions for improvement, which often align with the challenges they noted. A recurring suggestion was to narrow the scope: cover fewer platforms in greater depth. As one put it, "focus on a few LCNC tools more deeply and forget the others." Students felt they would learn more if they could spend say 3–4 weeks on one platform to build a substantial project, rather than 1–2 weeks on several. Another suggestion was to allocate more time to teaching the basics up front (for example, a short primer on programming or logic for those completely new, before diving into tools). This was implied in comments where students felt some tasks were "too advanced" for absolute beginners who lacked any programming background – e.g. writing concept documents or doing agile rituals without context could confuse those who have never coded or seen a software project before. Some students, while acknowledging the value of the course, questioned its placement in the curriculum. One said it "felt a bit pointless, especially as a compulsory course… this might have fit better as a recap or elective course." This perspective likely comes from students who would have preferred a traditional coding course first, and see the low-code approach as something extra. Nonetheless, the majority did see the relevance; even the critical voices often prefaced with "it was interesting, but…". Importantly, issues of assessment and feedback were highlighted. Several students complained that the grading criteria were not clear and that they received little feedback on their assignments or projects. "The assessment methods were not explained in detail and were vague. We did not receive any feedback for any of our projects… I would like to know how my projects can be better," wrote one student. This indicates that while students were producing work, they craved more formative feedback to understand the quality of their work and how to improve – a reasonable expectation that future iterations should address (perhaps through rubric-based feedback or peer reviews). Another student commented that the final project grading felt unjust, likely tied to the earlier remark about different information given to different groups. Ensuring fairness and clarity in evaluation is clearly an area needing attention.

In summary, the qualitative feedback reveals that students valued the innovative, practical nature of the course and felt it gave them unique insights and confidence, but they also experienced significant pain points: the course demanded a lot of self-learning in a short time, and gaps in support and communication sometimes left them frustrated. Many criticisms centered not on the concept of using AI/LCNC (which few explicitly opposed) but on the execution details – workload, instruction, and assessment. These insights are invaluable for refining the course design. Despite the challenges, it's notable that a number of students ended their feedback on a positive note, recognizing the experiment's intentions. For example, one student wrote (translated from Finnish), "Otherwise everything went OK, but the schedule for working on

tasks was too tight, and I don't think the final projects were evaluated fairly. The final project's evaluation criteria should have been known from the start. However, it was a well-done course. The only problem was that the use of different programs wasn't taught enough, so everyone spent a lot of their own time learning them." This encapsulates the mixed but ultimately constructive tone of many comments, acknowledging both the benefits of the course and the areas for improvement.

CONCLUSION

The AI-assisted LCNC course significantly impacted student engagement, creativity, problem-solving abilities, and learning outcomes, providing an accessible entry point into programming. Key findings include:

Enhanced Engagement and Creativity: Students displayed high motivation and creativity by developing practical, real-world applications addressing everyday problems. Projects ranged from budgeting apps to educational games, with AI acting as a catalyst for idea generation. Students valued the freedom to choose topics and appreciated the user-centered, tangible outcomes compared to traditional theoretical coursework.

Rapid Skill Development and Problem-Solving: Students quickly acquired technical skills and exhibited problem-solving resilience, handling advanced tasks like API integration, UI troubleshooting, and logic flow design. Despite challenges, students effectively learned new technologies rapidly, often with AI assistance. Notably, teams accomplished complex tasks like user authentication and database integration earlier than traditionally expected, demonstrating genuine computational thinking and logical problem-solving rather than superficial tool usage.

Role of AI Assistance: AI significantly supported student learning by providing creative prompts, code examples, debugging assistance, and quiet tutoring for less outspoken students. Students recognized AI as a beneficial but imperfect tool, cultivating critical thinking and a balanced understanding of AI's capabilities and limitations. However, benefits varied based on students' comfort with and trust in AI tools, highlighting the need for guidance on effective AI usage.

Collaboration and Interpersonal Skills: Team-based projects positively influenced students' collaboration, communication, and interpersonal skills, though outcomes varied. Stronger teams effectively employed management tools like Trello, showcasing improved teamwork over time. The final project presentations allowed students, particularly from business backgrounds, to demonstrate strengths beyond coding, underscoring the interdisciplinary benefits.

Accessibility and Inclusion: The LCNC approach lowered entry barriers, enabling non-technical students to meaningfully contribute, notably increasing course completion rates. Free and open-access tools ensured inclusivity, allowing diverse student participation. However, some students found the rapid pace challenging, suggesting that low-code might best serve as a complementary rather than initial introduction to programming.

Challenges and Limitations: Practical obstacles included limited free-tier functionality of LCNC tools and platform-specific learning curves. Some students desired deeper coding experience or clearer assessment criteria, indicating areas for pedagogical improvement. Additionally, external dependencies like platform downtimes occasionally disrupted course progress.

In summary, the AI-driven LCNC approach effectively motivated and engaged students, accelerated technical skill acquisition, and improved problem-solving and collaborative abilities, while highlighting key areas for refinement in instructional scaffolding and resource selection.

Future Research Directions: Future research should focus on tracking how LCNC students perform in subsequent programming courses, comparing LCNC-first versus code-first approaches, and further refining AI integration with specialized educational AI systems. Exploring AI-based assessment for faster feedback and the evolving role of automated low-code development in aligning education with workplace practices will be important. The AI-driven LCNC approach effectively motivated and engaged students, accelerated technical skill acquisition, and improved problem-solving and collaborative abilities, while highlighting key areas for refinement in instructional scaffolding and resource selection. It provides a promising, inclusive entry into programming education, especially for students from diverse or non-traditional computing backgrounds, suggesting a complementary role alongside traditional programming instruction lifecycle.

ACKNOWLEDGMENT

The authors wish to express their gratitude for the opportunity to conduct this research in the spirit of open science. We affirm that our work was guided solely by our commitment to contribute to the academic community, free from external demands or influences. Furthermore, we acknowledge that this research was undertaken without the benefit of external funding, underscoring our dedication to the pursuit of knowledge. We hope that our findings will serve to further enrich the discourse in our field.

REFERENCES

Crawley, E. F., Malmqvist, J., Ostlund, S., & Brodeur, D. (2007). Rethinking engineering education: The CDIO approach. Springer. https://doi.org/10.1007/978-0-387-38290-6

Friedenthal, S., Moore, A., & Steiner, R. (2008). A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann; Elsevier Science.

Gartner. (2021). Gartner forecasts 70% of new enterprise applications will use low-code by 2025. Retrieved from https://www.gartner.com/en/newsroom.

Guzdial, M. (2019). It's time for more research on learning programming with block-based languages.

Hirzel, M. (2023). Low-code programming models. Communications of the ACM, 66(2), 36–44. https://doi.org/10.1145/3573626

Huang, R., & Li, Y. (2022). The impacts of low-code development on IS learning. Northeast Association of Information Systems (NEAIS) Proceedings, 2022. Retrieved from https://aisel.aisnet.org/neais2022/.

Kim, J., Merrill, K., Xu, K., & Sellnow, D. D. (2020). My teacher is a machine: Understanding students' perceptions of AI teaching assistants in online education. International Journal of Human–Computer Interaction, 36(19), 1902–1911. https://doi.org/10.1080/10447318.2020.1817673

Margulieux, L. E., Catrambone, R., & Guzdial, M. (2020). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. International Journal of STEM Education, 7(1), 20. https://doi.org/10.1186/s40594-020-00218-z

Nieminen, N., & Reunanen, T. (2024). Artificial intelligence as a catalyst: A case study on adaptive learning in programming education. Proceedings of AHFE 2024.

Qian, Y., & Lehman, J. D. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. ACM Transactions on Computing Education (TOCE), 18(1), 1–24. https://doi.org/10.1145/3077618

Taajamaa, V., Eskandari, M., Karanian, B. A., Airola, A., Pahikkala, T., & Salakoski, T. (2016). O-CDIO: Emphasizing design thinking in CDIO engineering cycle. International Journal of Engineering Education, 32(3B), 1530–1539.

Terzopoulos, G., & Satratzemi, M. (2020). Voice assistants and smart speakers in education: A systematic review. Informatics in Education, 19(3), 473–490. https://doi.org/10.15388/infedu.2020.21

Tsakalerou, M., Evangelou, S. M., & Xenos, M. N. (2024). Unlocking the secrets of student success in low-code platforms: An in-depth comparative analysis. In Proceedings of the 2024 ASEE Annual Conference & Exposition.

Vygotsky, L. S. (1978). Mind in society: The development of higher psychological processes. Harvard University Press.

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33–35. https://doi.org/10.1145/1118178.1118215

Zhou, M. Y., & Lawless, W. (2015). An overview of artificial intelligence in education. Artificial Intelligence: Concepts, Methodologies, Tools, and Applications, 2(1), 2445–2452.