The Indonesian Journal of Computer Science
www.ijcs.net
Volume 14, Issue 1, February 2025
https://doi.org/10.33022/ijcs.v14i1.4592

A Systematic Review of Challenges in Teaching and Learning Computer Programming Modules

Elegbeleye Femi Abiodun[1], Bassey Isong[2]
26600536@nwu.ac.za[1], bassey.isong@nwu.ac.za[2]
[1,2] Computer Science Department, North-West University, Mafikeng, 2790, South Africa

Article Information

Received : 28 Dec 2024
Revised : 24 Jan 2025
Accepted : 24 Feb 2025

Keywords

Computer programming, Computer Science Education, Programming challenges.

Abstract

In today's rapidly changing digital world, computer programming (CP) has become an indispensable skill across various fields which fuels innovation and growth. Despite its critical importance, many students pursuing Computer Science and related fields struggle to comprehend fundamental CP concepts such as logic, syntax, data structures, and data types. These challenges often lead to high failure rates and decreased motivation, resulting in poor academic performance. This study examines the specific programming problems that students encounter, explores the contributing factors, and identifies the most challenging issues on their learning journey. In addition, it investigates whether these obstacles affect computing and non-computing students differently and identifies various strategies to improve learning outcomes. We employed a systematic literature review approach and extracted relevant information from existing literature selected from reputable electronic databases such as IEEE Xplore, ACM, ScienceDirect, and Google Scholar. The findings identified different learning styles among students, the complexity of CP concepts, insufficient resources, and educators' abilities as critical challenges. It also offers insights into best practices for improving motivation and engagement in programming education, such as adaptive learning tools, game-based applications, and artificial intelligence-driven support systems personalized to meet individual student needs.

A. Introduction

In today's digital age, computer programming (CP) has become an indispensable skill that transcends several disciplines and organizations. [1][2] CP is foundational to global innovations, enabling the development of artificial intelligence (AI) and machine learning (ML) technologies. However, these emerging technologies were only referenced in this context as an example of fields influenced by the advancements in programming. [3] [4]This study will only focus on how teaching and learning programming languages is pivotal for effectively enhancing students' capacity to engage with such technologies. Also, despite the importance of CP, CP remains a challenging subject for students, regardless of whether they are from computer science or non-computer science fields. Understanding these challenges is critical for researchers aiming to propose practical solutions that improve teaching and learning programming outcomes. [5] [6]. To this end, undertaking a comprehensive study on teaching approaches and methodologies is vital for addressing challenges hindering students' ability to learn and apply programming knowledge. [6]. Additionally, the ability to write and comprehend code is central to reshaping industries and essential for driving innovation in advanced software applications and emerging technologies like AI and ML. These technologies, while transformative, serve as illustrations of the broader impact of programming on global technological progress. [7]. Therefore, students will be better equipped by focusing on strategies that directly improve programming education.

Even though CP is essential, learning programming has presented some challenges among students, and these challenges can be attributed to various factors; due to the abstract nature of coding itself, some students find it very difficult to acquire this skill; also, students not understanding programming syntax, and logic are some of the challenges they faced by the students. [11]. These difficulties have caused the students to become frustrated, and some have lost confidence in learning programming; if proper measures are implemented, these will eventually affect students' academic performance and general learning outcomes. Furthermore, the creation of fresh teaching resources and tools might benefit from a thorough analysis of the challenges students face when learning programming languages. These tools could be gamified learning environments, interactive coding platforms, or AI-powered tutoring programs that adjust to each student's unique needs. [6]. Prepare them for the needs of the digital workforce by putting forth or suggesting practical solutions to these problems.

This study aims to comprehensively analyze students' challenges when offering computer programming modules, evaluate and identify interventions and trends, and provide insightful concepts for improving how programming languages can be taught to learners.

This article will be based on the following research questions (RQs):

RQ1: What programming challenges do students face in CP modules, and which challenges impact them the most?
RQ2: What factors contribute to students' difficulties in CP?
RQ3: What class of students are most affected by these challenges? CS students, NCS students, or Both?

RQ4: To what extent have challenges been identified, and what interventions have been offered to address them?
RQ5: What are the best practices that can be used to motivate and engage students in learning programming?

The key contribution of this study is, therefore, briefly summarized as follows:

1) We systematically identified key challenges students and educators face in CP modules, such as cognitive overload, lack of poor background knowledge, and ineffective teaching approaches.
2) The study provides excellent insight into which teaching methodologies (e.g., gamification, flipped classrooms, or project-based) are the most effective methods for solving these challenges.
3) We systematically reviewed existing studies, identified research gaps, and provided a guide for future researchers to develop an innovative solution.
4) The study also provides actionable interventions and recommendations for overcoming student challenges.

A. Planning

This section describes the systematic review planning process, starting with a detailed search plan to identify relevant studies from key databases and repositories. Data collection focused on peer-reviewed sources and reputable publications. Inclusion and exclusion criteria ensured studies were relevant, recent, and methodologically sound. Titles, abstracts, and full texts were screened, and eligible studies were critically analyzed. This structured approach ensured a comprehensive and unbiased review.

B. Search strategy

All search terms used in the process were documented to ensure the search processes are replicable and transparent. This was achieved by keeping a systematic review search log of all search terms used. We also kept track of previous studies that were accepted and rejected from all the databases. Together with my supervisor's guides, the search terms were created as shown in Table 2; how this was done is outlined in Table 1 as proposed by [12]. The databases listed in Table 3 were all used in this study.

Table 1. Search terms construction process

No	Search terms construction process
1	The paper should only be from publication date 2019-2024
2	By checking the keywords in some papers, we already have
3	The paper must only be written in English.
4	The paper should only be Conference, Journal articles, and book chapters.
5	The primary term is formed from the RQs identifying the solutions, outcomes, interventions, and challenges.

Table 2. Search terms

No	Search terms
1	CS programming challenges
2	CS programming difficulties
3	NCS programming challenges
4	NCS programming difficulties
5	CS programming interventions NCS programming interventions
6	CS programming solutions
7	NCS programming solutions

Table 3. Databases used

No	Database name
1	ACM Digital Library
2	IEEE Xplore
3	Science Direct
4	Google Scholar

C. Study selection criteria and procedures

The study began by performing an extensive search across the ACM Digital Library, IEEE Xplore, SpringerLink, and ScienceDirect to identify relevant studies done previously by scholars/researchers. Our search focused on studies addressing the specific challenges faced by both CS and NCS students and the interventions designed to overcome these challenges. We applied strict inclusion criteria, limiting our selection to papers published between 2019 and 2024, written in English, and categorized as either journal articles or conference papers. After gathering the initial pool of documents, we carefully screened titles and abstracts to ensure they aligned with our focus on student challenges and interventions. Studies that did not address these core themes or fell outside the specified time frame or language requirement were excluded. Following this, we conducted a full-text review of the remaining papers to verify that they met all inclusion criteria and provided meaningful insights into student challenges and corresponding interventions. This systematic process allowed us to curate a focused body of literature that directly supports our research's overall goals and objectives. Detailed selection is presented in Table 2.

Table 4. Detailed Study Inclusion and Exclusion Criteria

	Inclusion Criteria
1	The journal article must be available in full text on the web.
2	The journal article must be a literature review, systematic literature, survey, case study, or comparative study.
3	The journal must be a peer-reviewed
4	NCS students' programming interventions
5	CS students' programming interventions
6	Non-CS students' programming challenges
7	CS students' programming challenges
	Exclusion Criteria
1	Journal articles that do not match the inclusive criteria will be excluded.
2	Journals that are not related to NCS students' programming will be excluded.
3	Journals not related to CS student's programming will be excluded.

E. Quality assessment

Key variables were examined to evaluate the quality of the studies included in the systematic literature review. First, we assessed the research design to ensure the methods employed were appropriate for the study objectives. We also evaluated whether the sample sizes were adequate and clearly reported to support reliable and valid conclusions. We also carefully examined the explanations provided for the interventions and whether there was sufficient data to back them, as illustrated in each study. The finding's applicability to our focus on non-computing and computing students was another crucial factor that was considered. We ensured that the research was published in credible journals and underwent peer review. Table 3 illustrates how the quality of the selected paper was validated.

Table 5. Quality criteria

No	Quality criteria
1	Does the study clearly describe the intervention or solution used to engage and motivate the students and the strategies and tools deployed for the solution?
2	Is the research methodology used well-defined and clearly outlines the steps to address the identified problem?
3	Does the research methodology clearly state the appropriate steps to solve the problem?
4	Is the design and conceptual framework selected in the study explicitly supported by literature?
5	Does the research methodology align with the study design and answer the RQs?
6	Are the validity threats related to the study results reported?
7	Are negative findings and gaps related to the study reported?
8	Are there any restrictions or limitations on the results of the study reported?

F. Data Extraction

We used a systematic process to extract data from all the journal databases to guarantee consistency in the data collection stages and ensure accuracy across all the chosen studies. Likewise, essential details from every paper selected were key in ensuring the extracted information was correct. Details such as the names of author(s), year of publication, the title of the studies, and article source were noted. Later, we extracted data based on the various difficulties encountered by students studying CS and those who don't (NCS students), along with solutions provided to solve the suggested interventions and problems. We also took note of specifics regarding the sample size, research design, and necessary conclusions to ensure that each study's contributions were fully understood.

G. Execution and Results

We used inclusive and exclusive selection criteria to systematically review and sort out all the studies selected during the execution phase. We started by searching the various designated databases and then used publication date, language, and article type to refine the results; the review process is presented in Figure 1.

Table 6. Results found per database

Database name	Total found	Total selected
ACM Digital Library	3168	30
IEEE Xplore	3008	55
Science Direct	4233	40
Google Scholar	35,300	50
Total	45,709	175

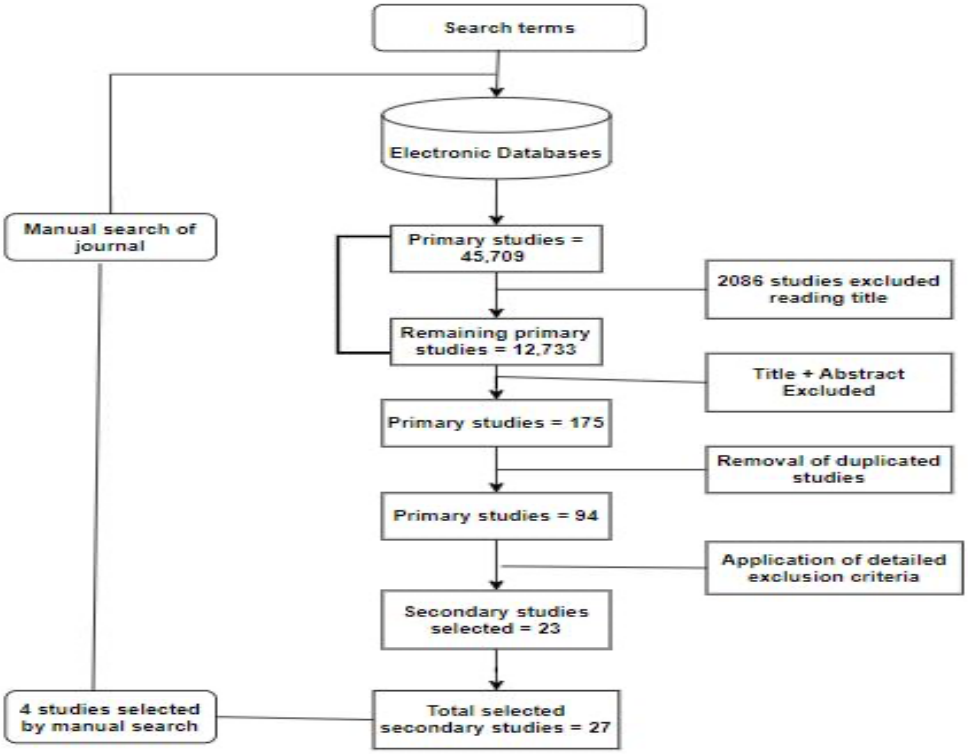

Figure 1. SLR Workflow

Figure 1 illustrates the systematic literature process, identifying all the journal paper warehouses, exclusive and inclusive criteria used in selecting the paper needed in the study, and ensuring the inclusion of relevant and high-quality papers chosen.

Table 7. Article selected from the systematic review

Id	Ref	Year	Study name
1	[6]	2024	Computing education in the era of generative AI
2	[13]	2021	A study of difficulties in teaching and learning programming
3	[11]	2022	Exploration of C++ Teaching Reform Method Oriented by Ability Output
4	[14]	2024	Impact of Computer Programming on Mathematics Education in K-12
5	[15]	2021	A gamified web-based system for computer programming learning
6	[16]	2024	Integrating online meta-cognitive learning strategy and team regulation to develop students
7	[17]	2023	Applying an evidence-based learning analytics intervention to support computer programming
8	[18]	2024	Students' perspectives on using digital tools in programming courses

9	[19]	2024	A Rule-Based Chatbot Offering Personalized Guidance in Computer Programming Education
10	[20]	2024	A study of factors influencing programming anxiety among non-computer students
11	[21]	2024	Experience Teaching Non-Computer Science Majors Computer Programming
12	[22]	2023	An enhanced career prospect prediction system for non-computer stream students in software companies
13	[23]	2021	A Phenomenographic Analysis of College Students' conceptions of and approaches to
14	[24]	2024	Interdisciplinary integration of computational thinking in K-12 education
15	[25]	2024	An Interactive Tool for Improving Python Syntax Mastery in Non-Computing Students
16	[26]	2024	Using Accessibility Awareness Interventions to Improve Computing Education
17	[27]	2024	Professional Development in Computational Thinking: A Systematic Literature Review
18	[28]	2024	Digital Game Making and Game Templates Promote Learner Engagement in Non-computing Classroom Teaching.
19	[29]	2021	Designing IDE interventions to promote social interaction and improved programming outcomes in early computing courses
20	[30]	2024	Integrating online partial pair programming & socially shared metacognitive regulation for the improvement of students' learning
21	[31]	2022	Analysis of factors contributing to the difficulties in learning computer programming among non-computer science students
22	[32]	2023	Evaluation Of The Impact Of Training In Programming With Scratchjr On Future Pedagogy Professionals
23	[33]	2023	Develop Digital Competencies Through Information Tech & Computer Modelling Lessons In Elementary School Education
24	[34]	2022	Increasing programming self-efficacy (PSE) through a problem-based gamification digital learning ecosystem (DLE) model
25	[35]	2024	Coding attitudes of fourth-grade Latinx students during distance learning
26	[36]	2021	Collaborating learning in an educational robotics environment
27	[37]	2024	Programming education and learner motivation in an age of generative AI: student and educator perspective
28	[38]	2024	An AI-driven virtual tutor for computer science education
29	[39]	2023	Effectiveness of flipped classroom pedagogy in programming education: A meta-analysis

Twenty-nine studies related to CP were found through an SLR, and most address some of the students' unique challenges in CP courses. As reported in the study [6], student overdependence on AI-generated code has affected the rate at which the learners will understand the fundamental programming syntaxes. Also, the lack of logical and essential problem-solving skills was another challenge mentioned in another paper. [13]. Another study noted that many students also struggle with understanding Object-Oriented Programming (OOP), such as combining primary programming key concepts like syntax and logic when writing codes and debugging. [11]. Challenges in teachers' having the proper training and the students viewing programming with incorrect perceptions further hinder the students' motivation. At the same time, an overreliance on automated evaluation systems diminishes students' engagement in the learning process. Teaching programming online, mainly to large classes, poses unique difficulties with student engagement, real-time analytics, and understanding key concepts like variables. Additionally, the lack of personalized guidance systems, such as chatbots, exacerbates these issues as students struggle to apply programming to real-life problems. Anxiety in learning programming languages is also prevalent, particularly among non-CS majors and students with limited foundational knowledge, who often find it difficult to use standard language libraries and functions. [14].

The challenge of teaching introductory programming to NCS students, like in CSC 101, is compounded by a lack of employability-focused platforms tailored to non-computer science students. This issue also extends to K-12 education, where computational thinking (CT) is not well-integrated into teacher training or assessment programs, and valid assessment tools are lacking. [15]. Additionally, there is no clear correlation between problem-solving time and solution quality, complicating the effectiveness of current tools. Addressing the accessibility and fairness of learning software remains a significant gap, and integrating digital games into the curriculum is underutilized. [16]. Poor feedback mechanisms, limited collaboration, and lack of teamwork hinder students' progress, as tasks and assignments are rarely completed in a collaborative setting. Together, these factors create complex barriers in programming education, requiring more integrated, personalized, and collaborative approaches to improve learning outcomes. Figure. 2 summarises the number of studies found according to a yearly publication. [17].
With high impact scores close to 9 on a 10-point scale, as shown in Figure 3, the graph displays the results and findings from previous studies that have the most impact on the student. As reported, the main factors influencing programming education are Automated Assessment, Real-time Feedback Systems, and Educator Challenges. Real-time feedback is essential for programming because it enables students to comprehend and fix mistakes rapidly. Improving these systems may increase learning outcomes and student engagement. Secondly, the introduction of automated systems will help the students to evaluate their work on their own and get immediate results. Although this practice may have reported some benefits, there's a chance that students will now prioritize test-taking over comprehending some key fundamentals of programming ideas, which, therefore, is the fair need for having assessment practices.

H. Analysis

Trends in publications and authors' contributions

Trends in publications

This paper gives a very insightful analysis of challenges, innovations, interventions, and trends in computer programming education, as identified through a systematic review of relevant studies from four journal databases. Some key findings provided by researchers helped in the analysis process to give an in-depth understanding of some challenges encountered by students and educators in teaching programming languages. Particular attention has been drawn to the recurring themes such as cognitive barriers [16], lack of engagement and motivation strategies, and the excessive use of emerging technologies by the student as also drastically affected the way and manner the student understands programming and the influence of the use of generative AI on the current educational practices [6]. Finally, this analysis highlights challenges the non-computer science students faced, such as accessibility issues, poor background, and lack of problem-solving skills [31]. Accessibility issues and the role of pedagogical interventions in addressing these difficulties will further help to curb some of the challenges in the current state of programming education and identify critical gaps and opportunities for further

research and innovation, according to Table 7. We shall be dividing the trends, challenges, and interventions of analysis provided into three key areas: (i) teaching methodologies and pedagogy, (ii) learner skills, motivation, and psychological barriers, and (iii) expanding the scope of programming education.

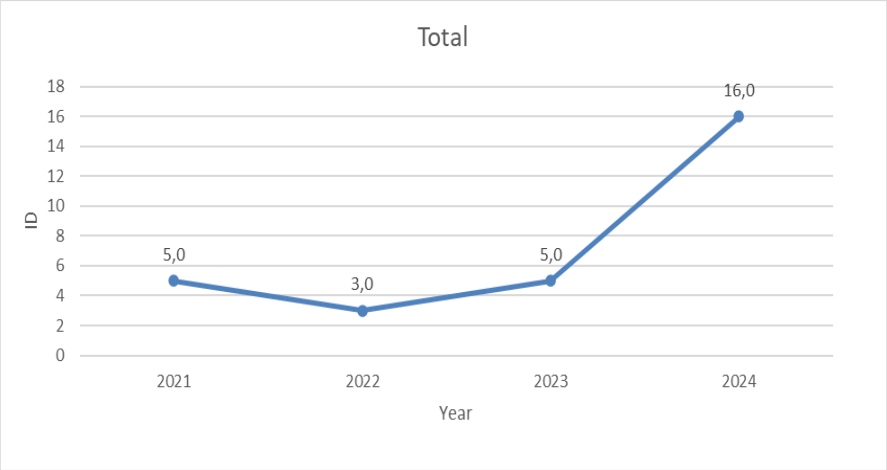

Figure 2. Sketch showing yearly publication trend (2021-2024)

Figure 2: The graph above shows the number of publications over the years, reflecting fluctuations influenced by the study's scope as we try to conduct a systematic review of some challenges that students face when offering programming modules. While many papers were published, only a limited number aligned with the primary objectives of this study. In 2021, five publications were selected as they were papers closely aligned with our goals. In 2022, it was observed that the number of documents has decreased. Therefore, we selected three papers because these studies are closely related to our research title. The number increased again to five publications in 2023, reflecting a renewed alignment with our research focus. Lastly, in 2024, more papers were selected because we noticed a surge in published publications, indicating an increased volume of relevant research or a notable expansion in the field directly addressing our study goal and objective. Figure 2 highlights the precise papers selected each year, emphasizing quality and relevance over quantity in selecting publications.

Authors' contributions

In this section, we shall summarize the authors' contributions in all 29 selected papers and what each author did with their objective and limitations as per individual contributions in each paper addressed how to improve programming education, focusing on their goals, methods, findings, limitations, and impact.

This research systematically examined various interventions and strategies presented by each paper. For example, integrating generative AI into computing education examines its transformative potential. [6]. At the same time, studies on chatbots providing holistic, personalized guidance [19] and AI-driven virtual tutors [38] highlighted using intelligent tools to support programming learners. Gamified systems [15] and interactive tools for Python syntax mastery [25] also boost

students' engagement and ease learning difficulties, especially for NCS students. Moreover, the role of accessibility awareness interventions [26] also points out how to make programming education more inclusive.

Collaboration and data-driven approaches are recurring themes in most of the papers. A study such as integrating pair programming (PP) with metacognitive regulation [30] and IDE interventions promoting social interaction [29] Demonstrates the value of teamwork and peer learning. Evidence-based learning analytics [17] and systematic reviews of professional development programs for computational thinking [27] Underline the importance of data-driven decision-making and educator support in improving outcomes. Innovative pedagogical strategies play a critical role. The flipped classroom pedagogy [39] and gamification-driven digital ecosystems [34] They are analyzed for their effectiveness in enhancing programming self-efficacy. C++ teaching reforms [11] Propose ability-oriented methodologies while integrating computational thinking into K-12 curricula. [24] and exploring interdisciplinary impacts on mathematics [14] To showcase the broader educational applications of programming. Some papers also address the challenges students face, particularly NCS major students. These include studies that deal with some students having programming anxiety. [20], difficulties in learning [13] and approaches to teaching NCS students [21]. One of the papers also reported a career prospect prediction system. [22] and an enhanced focus on digital competencies for future educators [33], reflecting efforts to align programming education with practical outcomes. The study analyzes students' conceptions and approaches. [23] and insights into Latinx students' coding attitudes during distance learning [35] Offer valuable perspectives on diverse learner experiences.

Game-based learning and engagement strategies are also prominent. Digital game-making and templates [28] and the impact of programming on future pedagogy professionals [32] Highlight ways to foster creativity and engagement. Similarly, the potential of robotics for collaborative learning [36] and the integration of training programs like ScratchJr [32] Underscores the role of hands-on, interactive learning. Finally, the role of computational tools and approaches in addressing learning barriers is explored. These include studies on online meta-cognitive strategies. [16], difficulties in learning programming [31], and evidence-based teaching reforms [37]. Several works also analyze the impact of programming education in improving computational thinking. [27] and fostering learner motivation [37], especially in emerging technologies.

In summary, these studies collectively came out with findings that reshape the ever-evolving programming education ecosystem through innovative tools, collaborative methods, pedagogical strategies, and targeted interventions that can help a wide range of learners across several fields of study. While they present actionable insights, many highlight the need for further research, broader validation, and long-term evaluation to maximize their impact on programming education to improve student performance and increase student motivation toward programming.

RQ1: **What programming challenges do students face in CP modules, and which challenges impact the most?**

In answering RQ1, recent studies, as described in Figure 3 below, illustrated the various challenges associated with programming modules, and this was presented in the 3D graph below. The challenges registered from the papers are categorized into eight types, namely: (Syntax error-related challenges) Difficulty in mastering syntax errors, debugging syntax errors, and applying the right syntax in solving meaningful real-life problems, therefore leading to the students having very low confidence in learning programming languages such as Python, C++, Java, etc. [11] [25] [31]. (Conceptual challenges) that is, students, lacking the basic knowledge of understanding abstract programming logic, such as recursion, condition statements, and inheritance [11], [13], [19], [24], [31], which therefore involves reducing student interest in the learning of programming language; the non-cs students are more impacted by the conceptual challenges [20]. (Cognitive overload challenges) This talked more about the difficulty in processing theoretical programming concepts by the students. [34]. Another issue discussed here is that students do not have a collaborative learning programming environment. [30], so they cannot handle multiple programming problems simultaneously [18].

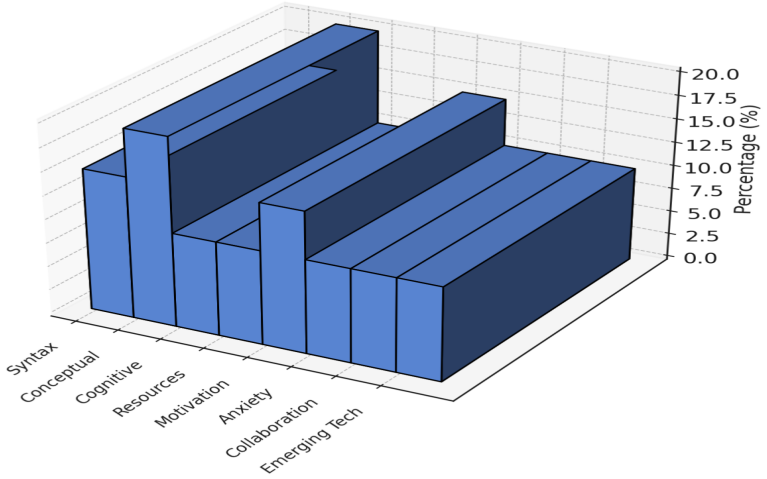

Figure 3. Programming challenges types

(lack of resources and tools) another challenge is students' limited access to well-needed resources such as educational tools and e-learning/distance learning materials [35]. Students are not used to the integrated development environment (IDE) [29], and accessibility barriers exist in programming education [26]. (Motivation and engagement issues) As reported in the papers, the lack of motivation in the traditional teaching approach also demotivates the students in learning programming. [15] [18] [35], limited access to gamification or interactive learning environments to engage the students [28]. The impact of this challenge on the students is causing a decline in performance and negative attitudes towards programming, especially the non-cs students. [28]. (Anxiety and psychological barriers) Programming anxiety among non-cs students arises from unfamiliarity with the subject, fear of failure when tackling challenging assignments, and stress caused by inadequate feedback or guidance. This anxiety leads to decreased confidence, increased avoidance of programming tasks, lower grades, reduced course participation, and limited career aspirations in computing fields. [16], [18],

[20], [31], [38]. (Collaborative learning and social interaction issues) Collaboration challenges in programming language include difficulties in communication, coordination, and limited social interaction in teaching programming education, leading to poor teamwork, reliance on individual work, and missed peer learning opportunities.[26],[29], [30]. (Integration of emergence technologies) Overreliance on generative AI tools and difficulty integrating them effectively lead to dependency, weaker foundational skills, ethical concerns, and challenges adapting to non-AI environments [6] [37], [38].

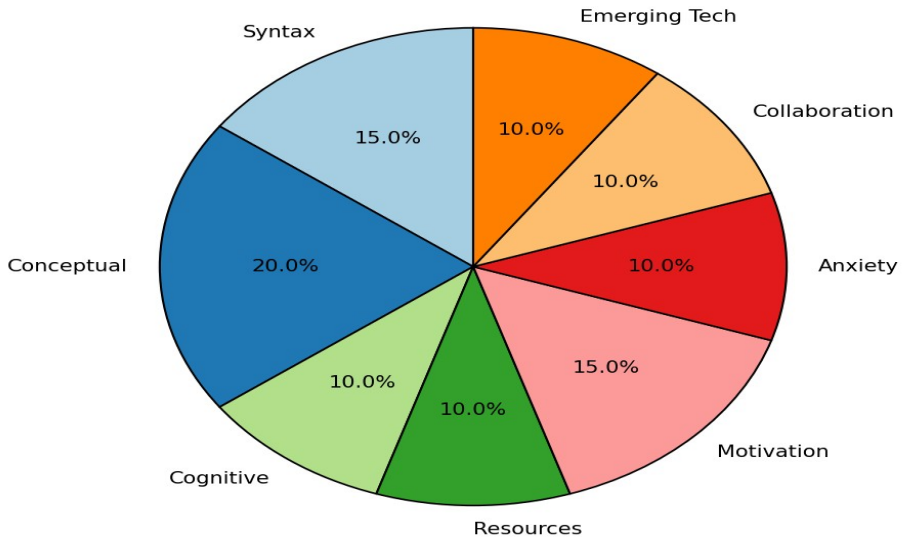

Figure 4. Programming challenges percentage distribution

Figure 4 analysis reveals various challenges students face in programming education, with conceptual understanding emerging as the most significant challenge, accounting for 20%. This highlights students' difficulty grasping fundamental programming concepts, the foundation for coding. Issues related to syntax (15%) also pose a considerable challenge, as errors in code structure, missing characters, and improper formatting often lead to frustration and hinder progress. Similarly, motivation (15%) remains a critical barrier as students struggle to maintain interest and engagement, particularly when faced with the complexity of programming tasks.

Other challenges include cognitive load (10%), where the mental demands of processing, retaining, and applying programming knowledge can overwhelm learners. The lack of adequate resources (10%), such as learning materials, tools, and software, further compounds the problem, limiting opportunities for effective practice. Anxiety (10%) also impacts learning, as fear of failure, debugging errors, and making mistakes erode students' confidence. In addition, difficulties in collaboration (10%) hinder students' ability to work effectively in teams and learn from peers, while keeping pace with emerging technologies (10%) creates a disconnect between educational content and industry expectations. These

challenges highlight the complex nature of learning barriers in programming education, with conceptual understanding, syntax, and motivation being the most prominent challenges. Addressing these issues through targeted support, improved resources, and innovative teaching strategies is crucial for enhancing students' learning experiences and outcomes.

Learners Challenges related to programming learning ability.

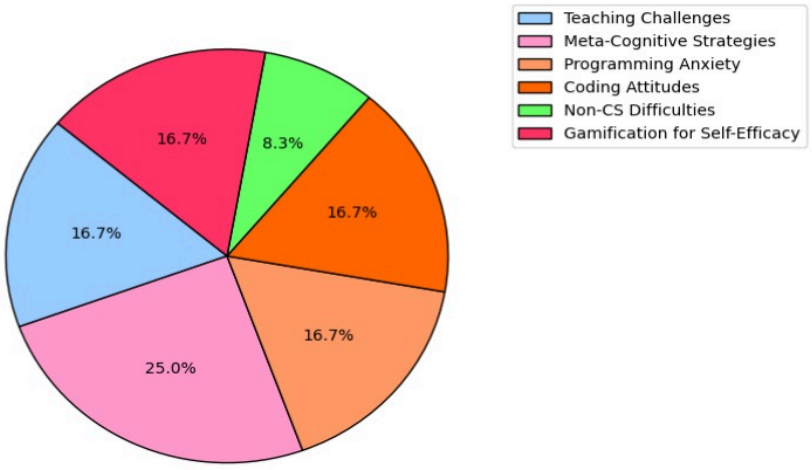

Figure 5. Students' programming challenges

Figure 5 shows the rate at which students struggle with learning programming modules. The 25% of teaching challenges seen in Figure 5 show that traditional teaching methods are ineffective, as many students struggle to understand. There are numerous challenges in delivering practical programming modules, as seen in several articles in Table 1. For example, learning programming is complex because it's a complicated subject, and different students have varying levels of preparedness. It is also discussed in the article [13] that a very high percentage of students indicate that this is a global challenge that needs to be addressed to develop better and more effective teaching strategies for teaching students the programming modules. Programming Anxiety (16.7%) As someone interested in programming coding, another critical psychology hinders individuals; people are well aware of it as it is reasonably well documented. In the study of factors influencing programmers' Coding skills: "A Study of Factors Influencing Programming Anxiety Among Non-Computer Students" (2024), it is noted that anxiety around programming affects students' learning ability in a significant way. The students suffer from various emotional and psychological blocks in learning to program, such as the fear of failure, low self-efficacy, and bad previous experiences with programming. Meta-Cognitive Strategies (8.3%) Cross-border strategies, including self-monitoring and reasoning for students to enable them to learn how to monitor their learning, are yet another significant aspect of dealing with barriers to meta-learning in programming education. As stated in the title of the article "Integrating Online Meta-Cognitive Learning Strategy and Team Regulation to Develop Students" (2024), meta-cognitive strategies serve the purpose of enhancing the anxiety skills of the students. This section, in comparison, is smaller.

Nonetheless, it indicates the recognition that students are taught how to think and reflect on their feelings, which enhances the acquisition of problem-solving skills and decreases angry outbursts while learning to code. Coding Attitudes (16.7%) The construction of coding attitudes stems from the psychology and motivation with which students come to understand. [35] It gives an example of how a student's views about coding affect the outcome of the coding task. It is well known that layperson learners often negatively perceive programming or coding. This negative perception usually leads to failure, avoidance, minimal effort in avoidance, or minimal engagement in the task. The chart's representation of this category shows the importance of having or developing a good attitude towards programming, which would help lower or overcome other psychological factors, including anxiety. NCS Difficulties (16.7%) The NCS students' difficulties are related to the issues of NSC students. However, this time, the view tends toward personal resource constraints, student support, and access to proper materials. Articles [21] Highlight the difficulties NCS students encounter when learning programming due to poor background knowledge. These factors may have contributed to a student's inability to achieve the skills needed in CP modules. Gamification for Self-Efficacy has these (16.7%). This talked more about self-efficacy using game-like elements in teaching programming, which has increased the students' understanding of learning programming modules and has boosted their confidence and motivation [34]. It discusses how integrating gamification into the learning process can help students build self-efficacy by providing immediate feedback, rewards, and a sense of achievement. This method allows students to engage with programming less intimidatingly, thus reducing barriers like anxiety and negative attitudes.

According to Figure 5, Programming Anxiety and Coding Attitudes are critical psychological barriers that must be addressed to help students succeed. Smaller segments like Meta-Cognitive Strategies and Gamification for Self-Efficacy show promising approaches to overcoming these barriers, while NCS Difficulties highlight the importance of addressing broader educational challenges. The interplay of these factors emphasizes the need for a holistic approach to improving programming education, considering both technical and psychological aspects of student learning.

RQ2: What factors contribute to students' difficulties in CP?

The teaching method used in all papers reviewed was discussed in this section, as shown in Figure 6. In one paper review, generative AI tools offer personalized feedback and self-paced learning [6]. Traditional teaching methods are critiqued for lack of engagement and clarity, focusing on improving practical application in programming [13]. As mentioned in this study, ability-based teaching approaches emphasize project-based learning to connect theory to real-world problems [11]. Another teaching method integrated gamification techniques like leaderboards to enhance motivation and engagement in programming courses [15]. Meta-cognitive learning strategies and team regulation to improve self-regulation and collaborative skills in students were some of the teaching methods used also in the papers [16]. Learning analytics provides real-time, data-driven feedback to support student progress [17].

Most of the papers emphasize using digital tools, such as IDEs and compilers, and rule-based chatbots for personalized guidance in debugging, which are explored to enhance learning efficiency [18][19], as described in Figure 6. Techniques to reduce programming anxiety, such as scaffolding and simplified exercises, benefit NCS students [20]. Teaching for NCS students focuses on real-world programming applications. At the same time, career-oriented learning helps bridge the gap between theory and professional use [21][22], which are also some teaching strategies used. Phenomenographic analysis of student learning approaches suggests adaptive teaching methods [23]. Interdisciplinary integration of computational thinking across subjects broadens the scope of programming education [24].

As reported, interactive tools for syntax mastery, especially in Python, help beginners improve their skills with instant feedback [25], another teaching method used in some papers. Accessibility awareness interventions ensure inclusive learning environments for students with disabilities [26]. Professional development in computational thinking enhances educators' teaching effectiveness [27]. Digital game-making and game templates engage students in programming through interactive activities [28]. Collaborative learning via educational robotics fosters teamwork and problem-solving skills [36]. Finally, AI-driven virtual tutors and the flipped classroom model allow personalized learning and active problem-solving during in-class sessions very quickly for the learners [38][39].

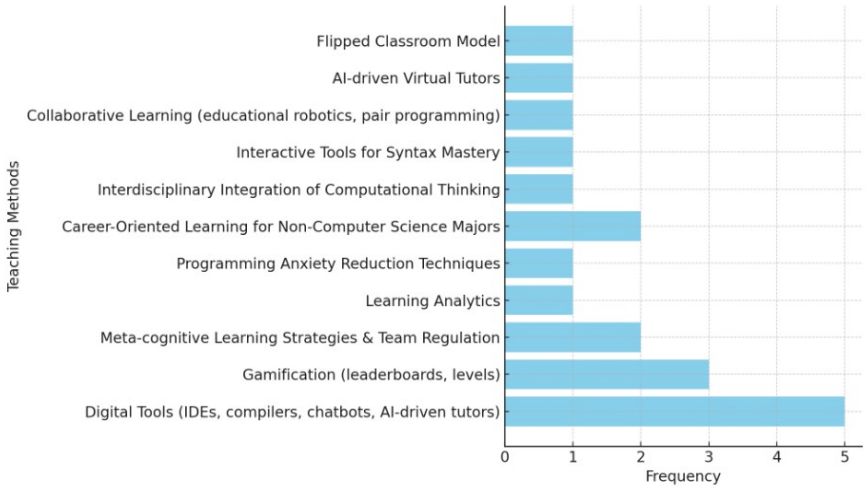

Figure 6: Frequency of the teaching methods used

Various challenges, including insufficient resources, complex syntax, and ethical considerations around using generative AI models, hinder the effective teaching of programming skills. [18]. Educators face difficulty helping learners grasp fundamental programming concepts for solving real-life problems, particularly when integrating programming into subjects like mathematics. This difficulty is compounded by a lack of interactive, real-time assessment tools and platforms that provide immediate student feedback, manage student engagement, and offer support, which are critical for developing problem-solving skills and reasoning. Programming's impact on learners' reasoning and problem-solving skills has shown

potential, especially when combined with tools like team regulation (TR) learning platforms and metacognitive learning strategies (MCLS) or supported by intervention-tracking systems like the I-Ntervene platform for automated grading. Enhanced learning environments, such as those that integrate visualization, online discussion, and immediate feedback platforms, aim to engage students effectively.

Using rule-based chatbots tailored for languages like Java can aid in improving response time, query accuracy, and overall user satisfaction, addressing learners' immediate needs in programming. In addition, the effectiveness of learning tools can be bolstered by educators' support platforms and teaching beliefs, impacting students' self-efficacy and satisfaction. Deep learning applications are also being explored to further support students with different learning needs. Studies highlight the differences in learning approaches between computer science and NCS students, emphasizing the importance of interactive learning tools, such as PyLe, for enhanced usability and educational value. However, there is still a lack of empathy education and accessibility resources, which calls for experiential learning approaches to make learning inclusive. The overreliance on individual work, with limited teamwork, often leads to negative experiences and potential dropouts. Research supports the benefits of socially shared metacognitive control (SSMR) and online partial pair programming (PPP) to mitigate these issues. Focusing on individual work and early instructional methods, combined with a lack of real-time evaluation tools and game-based templates, contributes to declining performance in programming education. Enhancing programming education will require balancing active learning, block-based languages, and integrating technology with formative assessments in an inclusive, collaborative environment that prepares students for practical problem-solving.

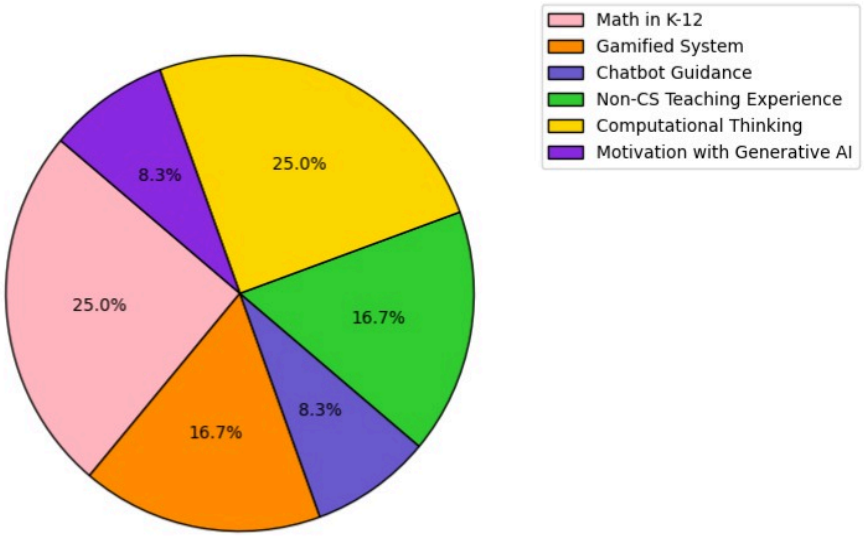

Figure 7. Teaching methodologies and approaches.

Figure 7. *Teaching methodologies and approaches* outline various approaches and methods that influence programming education, focusing on frameworks, models, and interventions in programming education; the following analysis emerges: Math in K-12 (25%). In this graph, we learn how K-12 education addresses real-world problems through programming, abbreviating problems scientifically,

and explaining the benefits of teaching math early. For instance, the paper titled "Mathematics as a Precursor for Computational Thinking: A Case Study in Programming Education, 2023" stresses the strong correlation between early math skills and programming success. The 25% portion illustrates how math-centric education equips students with problem-solving skills and reasoning logic vital to programming. System Gamified (25%) According to this study [28], there is A recurrent theme of gamification in programming education. Students who might ordinarily find programming boring or scary are primarily engaged by this approach, which uses points, levels, and prizes. Chatbot Guidance (16.7%) The growing usage of AI to give students immediate feedback and help is reflected in chatbot-based learning interventions (e.g., "AI-Powered Chatbots for Personalized Programming Education") [19]. The chart's 16.7% allotment emphasizes how interactive AI tools can assist conventional teaching strategies and meet the unique learning demands of each student, especially those who need extra help outside of the classroom. NCS Teaching Experience (16.7%) This section highlights the difficulties experienced by teachers without a computer science background when teaching programming. Articles such as [21], which examine how professional development programs and frameworks might assist in preparing such educators, are cited in Table 7. Computational Thinking (8.3%) The graph highlights the importance of computational thinking in programming education through studies such as [24]. The distribution of the chart highlights that although computational thinking is essential, its successful use necessitates supplementary tactics (such as gamification or chatbot instruction). Motivation with Generative AI (8.3%) ChatGPT, DALL·E, and other generative AI tools have become cutting-edge resources for increasing student involvement and programming innovation. Studies like [37] and Table 7 explore how these tools can motivate students by personalizing and enhancing the interactive nature of learning.

Furthermore, the graph visually represents the crucial elements and interventions influencing programming education. While Chatbot Guidance and NCS Teaching Experience showcase creative and valuable ways to enhance teaching and learning, K–12 math and gamified systems focus equally on these fundamental functions. Computational Thinking and Motivation with Generative AI's minor but significant contributions highlight the necessity of a well-rounded, multifaceted strategy to address various programming education issues.

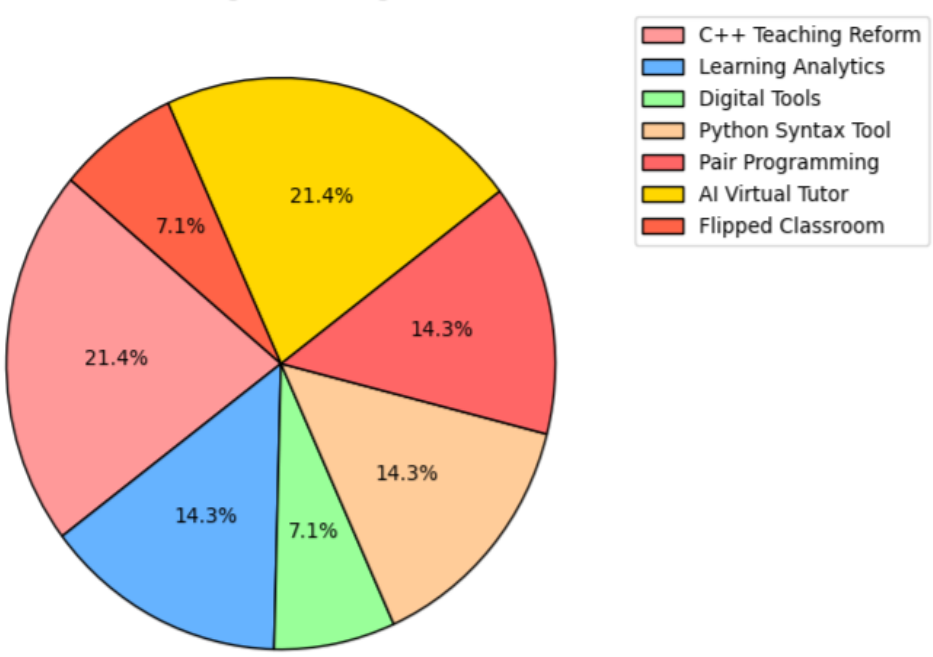

Figure 8. Teaching methodologies and pedagogical approaches

Figure 8. outlines several teaching methodologies used in programming education. The methods are based on Table 7, which brought a broad spectrum of innovative pedagogical approaches designed to improve programming instruction. C++ Teaching Reform (21.4%) This segment illustrates the great attention paid to the teaching reform of C++, as in the article. [11] Teaching reform represents one of the significant aspects of contemporary programming education, which requires changes in conventional methodologies based on demands from students' learning experiences. Such a high percentage implies a considerable interest in rethinking how C++ is taught, perhaps through competency-based education or modifications to curricular materials that make them more accessible and impactful for learners. Learning Analytics (14.3%) The paper demonstrates that learning analytics is another crucial approach in programming education. [17]. The authors use tools from learning analytics that equip educators with data to make informed decisions about teaching. This approach identifies students who might be struggling and provides targeted interventions to improve outcomes. The 14.3% allocation signifies its increased relevance within educational strategies. Digital Tools (21.4%) The paper [18] Highlights that digital tools used in programming education are also significant. Digital tools include integrated development environments (IDEs), coding platforms, and collaboration tools. For students, these tools offer a much more experiential and interactive approach to dealing with programming, which in turn encourages better habituation to coding practices. Python Syntax Tool (7.1%) An interactive tool for enhancing Python syntax mastery among NCSs has been discussed in [25]." Although representing a smaller demographic, such tools show an explicit effort toward aiding learners in acquiring particular programming languages through tailored immediate feedback on their coding efforts. PP (14.3%), as detailed in [30], is a tandem instructional strategy wherein two learners tackle an identical problem. One student types the code while the other critiques it. This method has been established to enhance learning outcomes due to its facilitation of

peer learning and heightened engagement levels. The chart represents well and thus reflects its relevance in the context of contemporary programming education. AI Virtual Tutor (14.3%) The emergence of AI-powered virtual tutors, as discussed [38] [40] [41], Underscores a trend toward more tailored learning experiences that artificial intelligence enables. Such systems can offer personalized feedback, tackle student queries, and assist learners with intricate programming tasks. The AI virtual tutor's approach is catching on, evident from its inclusion in the graph. Flipped Classroom (7.1%) The flipped classroom model, where students absorb new material at home by watching videos or reading texts and then use the acquired knowledge in school, is not very common but still has a notable place within programming education, as noted in [39]. This model encourages students to engage in practice and practice more during the class to master particular programming concepts and skills, which is why there is very little content delivered during the lessons.

In summary, graphical representation further substantiates Table 1 in that it assists in showing the different models that are emerging in programming education. The highlighted places of learning programming languages, such as C++, Java, Python, and learning analysis, are the strategies to enhance the interactivity, data focus, and adaptability of programming education. The provision of AI tutors, PP, and the inclusion of flipped classrooms illustrate that novel, active, and technology-oriented pedagogies are continuously incorporated into the programming syllabus. These strategies are essential in responding to the problems existing in programming education, improving student participation, their interest, and, consequently, the learning outcomes.

RQ3: What class of students are most affected by these challenges? CS students, NCS students, or Both?

As noticed in Figure 9, the challenges in computing education vary, impacting both CS students and NCS students fields. For instance, the emergence of generative AI poses challenges in adapting traditional teaching methods to new technological advancements, affecting both fields [6]. NCS students, specifically, encounter issues with learning programming, including programming anxiety and insufficient prior exposure to computational thinking, as highlighted in various research studies [13], [20], [31]. Furthermore, integrating programming education into NCS fields reveals challenges related to pedagogical approaches and student involvement [21], [[25], [31], [32].

CS education also contends with notable challenges, especially the need to revise programming languages like C++, Python, and Java to improve student outcomes and prepare them for practical applications in the job market. [11]. Additional hurdles arise from the growing integration of digital tools and AI into programming educational curricula, as students and educators face difficulties in effectively using these resources to enhance learning and motivation. [18], [37], [38]. While strategies like gamification and online learning show promise, they also create challenges in sustaining student interest and addressing problems in skill development. [15], [28], [34].

Moreover, teaching programming in early education and K-12 environments presents specific challenges regarding curriculum integration, particularly with educational tools like Scratchjr and the advancement of digital skills. [24], [34]. The research underscores the critical role of social interaction in programming classes, noting that IDE interventions have potential but also encounter obstacles during implementation because some students face challenges adapting and understanding the working environment. [29], [30]. Finally, there is an increasing emphasis on the professional development of educators in computational thinking, which offers both opportunities and challenges in improving teaching methods within programming education. [16] [27]. Lastly, Figure 9 shows the number of paper distributions of the challenges and the percentage of studies that affected both CS and NCS, showing that researchers should consider students who can accommodate both disciplines.

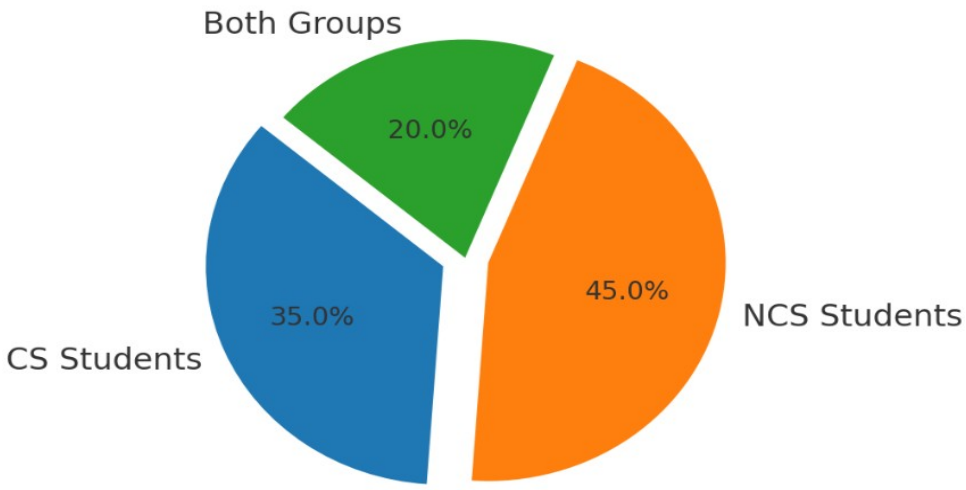

Figure 9. Graph showing the distribution of challenges (CS), (NCS), or Both students

RQ4: To what extent have challenges been identified, and what interventions have been offered to address them?

Each of these four papers emphasizes some key areas of the application of CP, including the involvement of participants, the techniques used, and the study's suggested interventions or practices.

Study one from Table 5 uses various techniques, including exercises, surveys, case studies, experiments, and university-level assignments, to examine CS and NCS students. The study, which includes 350 graduate and undergraduate students, investigates how AI tools might be incorporated into the curriculum to give students individualized feedback. The suggested approach addresses ethical issues in AI-based education and emphasizes using generative AI to improve instructional content.

Also, study 2 from Table 7 uses an SLR approach at the undergraduate university level, focusing on NCS students. It looks at issues instructors and students have when teaching programming to NCSs without mentioning the total number of participants. The study recommends improving teaching and learning strategies to encourage critical and logical thinking in students. It also suggests using learner assistance tools like visualizations to help overcome challenges.

Likewise, [11] Table 7 is an experiment conducted in a university setting, specifically for undergraduates, focusing on computer science students. The study investigates how traditional teaching methods and competition can be integrated, though the number of participants is not disclosed. The suggested approach includes a five-point teaching model integrating live code management, student-centered systems, problem-solving, and an integrated assessment system to enhance students' programming skills. Finally, the integration of CP into high school mathematics curricula is the main topic of study shown in Table 7. This literature review examined 12 important and relevant journal articles and explored fundamental mathematical ideas using different programming languages.

Table 8: Quality Assessment of Papers (Yes/No)

Ref	Is an Introduction provided?	Is the research methodology defined	Is the design of the study stated?	Are validity threats reported?	Are negative findings reported?
1	yes	yes	yes	yes	yes
2	yes	yes	yes	no	yes
3	yes	yes	yes	no	yes
4	yes	yes	no	yes	yes
5	yes	yes	yes	no	yes
6	yes	yes	yes	yes	yes
7	yes	yes	yes	no	yes
8	yes	yes	no	no	yes
9	yes	yes	yes	yes	yes
10	yes	yes	yes	yes	yes
11	yes	yes	yes	yes	yes
12	yes	yes	no	yes	yes
13	yes	yes	yes	yes	Yes
14	yes	yes	yes	yes	yes
15	yes	yes	yes	no	yes
16	yes	yes	yes	no	yes
17	yes	yes	no	yes	yes
18	yes	yes	yes	no	yes
19	yes	yes	yes	yes	yes
20	yes	yes	yes	no	yes
21	yes	yes	no	no	yes
22	yes	yes	yes	yes	yes
23	yes	yes	yes	yes	Yes
24	yes	yes	yes	yes	yes
25	yes	yes	yes	yes	no
26	yes	yes	yes	no	no
27	yes	yes	yes	yes	yes
28	yes	yes	yes	yes	yes
29	yes	yes	yes	yes	yes

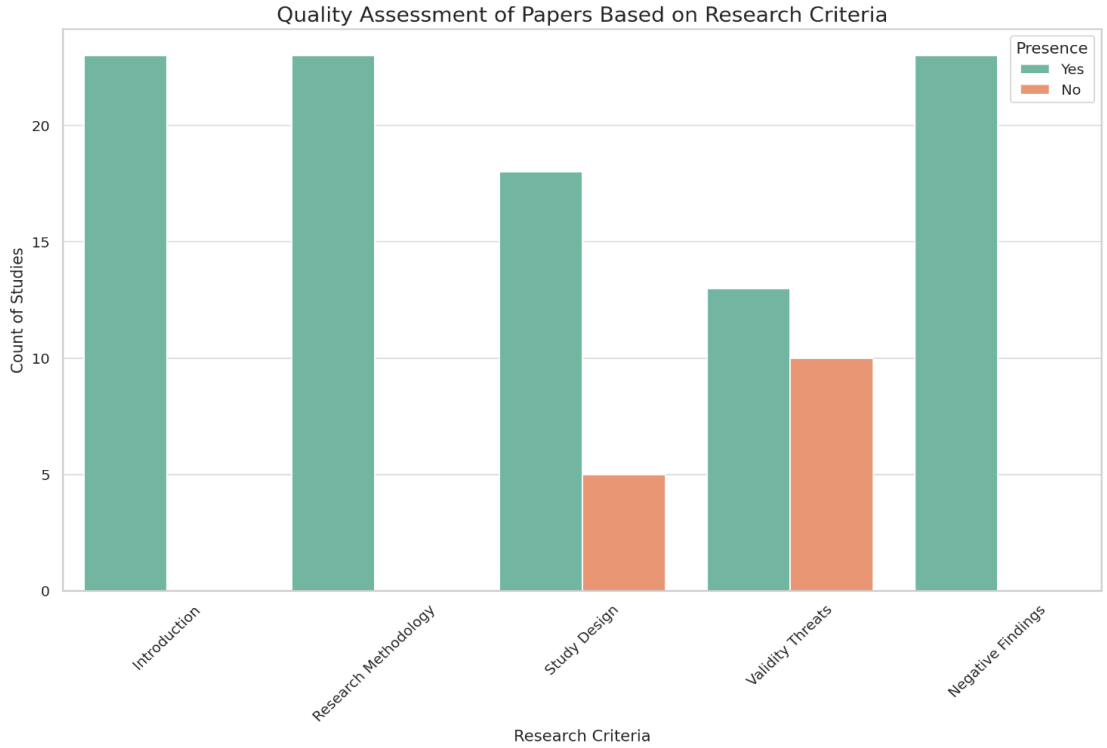

Figure 10. Quality Assessment

Figure 10 shows the quality assessment of 29 selected papers given in the table based on multiple criteria, including the inclusion of an introduction, a clearly defined research methodology, a stated study design, the reporting of threats to validity, and the acknowledgment of negative findings**.**
Every paper included an introduction and outlined the research methodology, demonstrating a consistent approach to explaining the methods and research context. Of 29 studies, 23 explicitly reported the study design, while seven did not address it. Twelve publications discussed validity threats, reflecting a mixed rigor in addressing potential study limitations. Notably, nearly all the studies (27 out of 29) reported negative results, suggesting that most researchers in this group were transparent about the challenges or limitations of their findings.

I. Limitations of the study

The methods used in this study have diverged from the recommendations given in Kitchenham's 2004 guidelines. [42]. Rather than using an automated search engine system, the paper search was conducted manually by selecting journals and conference proceedings. This method is consistent with the work of previous researchers who, instead of researching a particular software evaluation technique, use more educational trends in CP and NCS students. We verify that the journals and conference proceedings excluded during the manual search process are not used when researching, as advised by Brereton et al., ensuring that the extracted and verified information extracted from all the journals and the conferences are accurate. [43].

Additionally, conferences devoted to educational methodologies might have included papers dealing with issues more specifically with programming education

than general educational trends. Our findings should be interpreted cautiously, mainly concerning systematic literature reviews in prestigious general and educational technology conferences and significant international education and computer science journals.

J. Conclusions

This paper conducted an SLR and presented some specific programming difficulties students encounter and their contributing factors. The findings show that teaching and learning computer programming is no easy task and comes with many challenges. These include different learning styles among students, the complexity of programming concepts, insufficient resources, and educators' abilities. Moreover, the fast-paced evolution of technology offers new opportunities and presents difficulties in keeping the curriculum up-to-date. Based on the findings, several strategies were identified to address these challenges, such as using a mix of teaching strategies. In particular, personalized learning, hands-on experiences, and integrating modern technologies such as adaptive learning systems and game-based learning are key. Also, developing a teaching and learning framework that provides timely support for both CS and NCS students is essential. This framework should encourage a growth mindset in programming and include varied learning approaches like problem-based learning, project-based learning, and gamification. These methods can reduce the stress of learning new skills, promote critical thinking, and shift the focus from memorizing syntax to solving real-world problems. Furthermore, teamwork-focused interventions can be particularly beneficial for NCS students. This is crucial because they create a collaborative environment that lessens individual pressure, fosters peer learning, and improves problem-solving skills. Ongoing professional development for educators and supportive learning environments are vital for enhancing student engagement and performance in programming courses.

In our future work, we intend to integrate adaptive and game-based learning approaches to address programming difficulties and create a more effective and inclusive learning experience for all students.

K. References

[1] A. Saraç and N. Özdener, "Adapting to the Industry 4.0 Era: Transdisciplinary IoT Education," in *Transdisciplinary Approaches to Learning Outcomes in Higher Education*: IGI Global, 2024, pp. 95-153.

[2] M. A. Hashmi, J. P. Mo, and R. C. Beckett, "Transdisciplinary systems approach to realization of digital transformation," *Advanced Engineering Informatics,* vol. 49, p. 101316, 2021.

[3] I. H. Sarker, "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions," *SN computer science,* vol. 2, no. 6, p. 420, 2021.

[4] T.-C. Hsiao, Y.-H. Chuang, T.-L. Chen, C.-Y. Chang, and C.-C. Chen, "Students' Performances in Computer Programming of Higher Education for Sustainable Development: The Effects of a Peer-Evaluation System," *Frontiers in Psychology,* vol. 13, p. 911417, 2022.

[5] P. R. Daugherty and H. J. Wilson, *Human+ machine: Reimagining work in the age of AI*. Harvard Business Press, 2018.
[6] P. Denny *et al.*, "Computing education in the era of generative AI," *Communications of the ACM,* vol. 67, no. 2, pp. 56-67, 2024.
[7] F. Buitrago Flórez, R. Casillas, M. Hernández, A. Reyes, S. Restrepo, and G. Danies, "Changing a generation's way of thinking: Teaching computational thinking through programming," *Review of Educational Research,* vol. 87, no. 4, pp. 834-860, 2017.
[8] M. C. Marino, *Critical code studies*. MIT Press, 2020.
[9] M. Noone and A. Mooney, "Visual and textual programming languages: a systematic review of the literature," *Journal of Computers in Education,* vol. 5, pp. 149-174, 2018.
[10] N. Gedik, "Investigation of student experiences in flipped programming using epistemic network analysis," Middle East Technical University, 2024.
[11] Y. Zheng, "Exploration of C++ Teaching Reform Method Oriented by Ability Output," in *International Conference on Computer Science and Education*, 2022: Springer, pp. 16-27.
[12] J. E. Hannay, D. I. Sjoberg, and T. Dyba, "A systematic review of theory use in software engineering experiments," *IEEE Transactions on Software Engineering,* vol. 33, no. 2, pp. 87-107, 2007.
[13] R. Kadar, N. A. Wahab, J. Othman, M. Shamsuddin, and S. B. Mahlan, "A study of difficulties in teaching and learning programming: a systematic literature review," *International Journal of Academic Research in Progressive Education and Development,* vol. 10, no. 3, pp. 591-605, 2021.
[14] A. AMIMOUR, "Impact of Computer Programming on Mathematics Education in K-12: Literature Review and Perspectives," 2024.
[15] G. Polito and M. Temperini, "A gamified web-based system for computer programming learning," *Computers and Education: Artificial Intelligence,* vol. 2, p. 100029, 2021.
[16] C.-W. Tsai, L.-Y. Lee, Y.-P. Cheng, C.-H. Lin, M.-L. Hung, and J.-W. Lin, "Integrating online meta-cognitive learning strategy and team regulation to develop students' programming skills, academic motivation, and refusal self-efficacy of Internet use in a cloud classroom," *Universal Access in the Information Society,* vol. 23, no. 1, pp. 395-410, 2024.
[17] P. Utamachant, C. Anutariya, and S. Pongnumkul, "i-Ntervene: applying an evidence-based learning analytics intervention to support computer programming instruction," *Smart Learning Environments,* vol. 10, no. 1, p. 37, 2023.
[18] M. Asgari, F.-C. Tsai, L. Mannila, F. Strömbäck, and K. M. Sadique, "Students' perspectives on using digital tools in programming courses: A cross country case study between Sweden and Taiwan," *Discover Education,* vol. 3, no. 1, p. 57, 2024.
[19] C. Papakostas, C. Troussas, A. Krouska, and C. Sgouropoulou, "A Rule-Based Chatbot Offering Personalized Guidance in Computer Programming Education," in *International Conference on Intelligent Tutoring Systems*, 2024: Springer, pp. 253-264.

[20] Y. Jiang, H. Wu, X. Yu, and T. Ji, "A study of factors influencing programming anxiety among non-computer students," in *Proceedings of the 2024 9th International Conference on Information and Education Innovations*, 2024, pp. 63-69.

[21] Y. Tseng, "Experience Of Teaching Non-Computer Science Majors Computer Programming," In *Inted2024 Proceedings*, 2024: Iated, Pp. 6575-6579.

[22] B. Abraham and P. Ambili, "An enhanced career prospect prediction system for non-computer stream students in software companies," in *Computational Intelligence for Engineering and Management Applications: Select Proceedings of CIEMA 2022*: Springer, 2023, pp. 811-819.

[23] T.-L. Chou, K.-Y. Tang, and C.-C. Tsai, "A Phenomenographic Analysis of College Students' Conceptions of and Approaches to Programming Learning: Insights from a Comparison of Computer Science and Non-computer Science Contexts," *Journal of Educational Computing Research,* vol. 59, no. 7, pp. 1370-1400, 2021.

[24] S. Yeni, N. Grgurina, M. Saeli, F. Hermans, J. Tolboom, and E. Barendsen, "Interdisciplinary integration of computational thinking in K-12 education: A systematic review," *Informatics in Education,* vol. 23, no. 1, pp. 223-278, 2024.

[25] A. K. Mbiada, B. Isong, and F. Lugayizi, "PyLe: An Interactive Tool for Improving Python Syntax Mastery in Non-Computing Students," *Journal of Information Systems and Informatics,* vol. 6, no. 2, pp. 1008-1034, 2024.

[26] Y. Liu, D. Mangano, K. P. Neupane, S. Malachowsky, and D. Krutz, "Using Accessibility Awareness Interventions to Improve Computing Education," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, 2024, pp. 66-71.

[27] A. Espinal, C. Vieira, and A. J. Magana, "Professional Development in Computational Thinking: A Systematic Literature Review," *ACM Transactions on Computing Education,* vol. 24, no. 2, pp. 1-24, 2024.

[28] T. Hughes-Roberts, D. Brown, A. Burton, N. Shopland, J. Tinney, and H. Boulton, "Digital Game Making and Game Templates Promote Learner Engagement in Non-computing Based Classroom Teaching," *Technology, Knowledge and Learning,* pp. 1-25, 2023.

[29] D. Olivares, C. Hundhausen, and N. Ray, "Designing IDE interventions to promote social interaction and improved programming outcomes in early computing courses," *ACM Transactions on Computing Education (TOCE),* vol. 22, no. 1, pp. 1-29, 2021.

[30] C.-W. Tsai *et al.*, "Integrating online partial pair programming and socially shared metacognitive regulation for the improvement of students' learning," *Universal Access in the Information Society,* pp. 1-17, 2024.

[31] R. Kadar, S. B. Mahlan, M. Shamsuddin, J. Othman, and N. A. Wahab, "Analysis of factors contributing to the difficulties in learning computer programming among non-computer science students," in *2022 IEEE 12th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2022: IEEE, pp. 89-94.

[32] P. Ibáñez-Cubillas, M. S. M. Pinto, And S. L. Rodríguez, "Evaluation Of The Impact Of Training In Programming With Scratchjr On Future Pedagogy Professionals," In *Edulearn23 Proceedings*, 2023: Iated, Pp. 6900-6904.

[33] D. Atanasova And I. Minkova, "Developing Digital Competencies Through Information Technology And Computer Modelling Lessons In Elementary School Education," In *Iceri2023 Proceedings*, 2023: Iated, Pp. 7411-7417.
[34] S. Chinchua, T. Kantathanawat, and S. Tuntiwongwanich, "Increasing programming self-efficacy (PSE) through a problem-based gamification digital learning ecosystem (DLE) model," *Journal of Higher Education Theory and Practice,* vol. 22, no. 9, 2022.
[35] L. Garcia, M. Parker, and M. Warschauer, "Coding attitudes of fourth-grade Latinx students during distance learning," *Computer Science Education,* pp. 1-39, 2024.
[36] G. T. Papadopoulos, M. Antona, and C. Stephanidis, "Towards open and expandable cognitive AI architectures for large-scale multi-agent human-robot collaborative learning," *IEEE Access,* vol. 9, pp. 73890-73909, 2021.
[37] S. Boguslawski, R. Deer, and M. G. Dawson, "Programming education and learner motivation in the age of generative AI: student and educator perspectives," *Information and Learning Sciences,* 2024.
[38] P. Bassner, E. Frankford, and S. Krusche, "Iris: An AI-driven virtual tutor for computer science education," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 2024, pp. 394-400.
[39] M. Almassri and R. Zaharudin, "Effectiveness of Flipped classroom pedagogy in programming education: A meta-analysis," *International Journal of Instruction,* vol. 16, no. 2, pp. 267-290, 2023.
[40] F. Sarshartehrani, E. Mohammadrezaei, M. Behravan, and D. Gracanin, "Enhancing E-Learning Experience Through Embodied AI Tutors in Immersive Virtual Environments: A Multifaceted Approach for Personalized Educational Adaptation," in *International Conference on Human-Computer Interaction*, 2024: Springer, pp. 272-287.
[41] M. Rızvı, "Investigating AI-Powered Tutoring Systems that Adapt to Individual Student Needs, Providing Personalized Guidance and Assessments," *The Eurasia Proceedings of Educational and Social Sciences,* vol. 31, pp. 67-73, 2023.
[42] S. Pizard, F. Acerenza, X. Otegui, S. Moreno, D. Vallespir, and B. Kitchenham, "Training students in evidence-based software engineering and systematic reviews: a systematic review and empirical study," *Empirical Software Engineering,* vol. 26, pp. 1-53, 2021.
[43] V. Garousi and M. Felderer, "Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 170-179.