Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X
Volume 3, Issue 6, 2025

Quantitative Analysis of Barriers to Undergraduate Computer Science Learning

[1]Ashraf Zia, [2]Umar Hayat Khan, [*3]Hashim Ali, [4]Kiran Falak Sher, [5]Umer Tanveer
[1-5]Department of Computer Science, Abdul Wali Khan University Mardan, Mardan, Pakistan

[*3]hashimali@awkum.edu.pk

DOI: https://doi.org/10.5281/zenodo.19050686

Keywords
Computational Readiness, Acknowledged Barriers, Educational Approaches, Programming Apprehension, Computer Science Education

Article History
Received on 18 May, 2025
Accepted on 10 June, 2025
Published on 12 June, 2025

Corresponding Author's: Hashim Ali

Abstract
There is a long-standing discrepancy between planned student learning and student achievement in undergraduate computer science. This research examined the role played by cognitive and affective variables in determining the learning and liking of computer science amongst students. In particular, it evaluated a Built-In Computer Science Learning Difficulty Model (ICSLDM) through the analysis of the correlation between basic computational readiness, programming anxiety, learning strategies, perceived difficulty of the subject and academic achievement. The research used a cross-sectional survey design which was descriptive in nature. The sample population included science students of all years in the district of Mardan and 112 students were sampled with the help of a 5-point Likert-scale tool. A diagnostic assessment was also used to enhance the findings by triangulation. The data were analysed with the help of one-sample t-test, paired t-tests, and Pearson correlation coefficients. Findings showed that the computer science seemed easier to students with better background knowledge (e.g. logical thinking and simple mathematics as applied to programming). It was discovered that there was a disparity in self-reported readiness of students to complete computing tasks and their diagnostic evaluation scores. On the whole, the students perceived computer science as a difficult subject. Although the students stated that the overall orientation towards self-regulated learning was positive, they expressed little confidence in learning programs and had low self-efficacy. Reduced self-efficacy was correlated with reduced cognitive and affective performance. The results uphold the main aspects of the ICSLDM, where the cognitive resources and affective factors significantly forecast the perceived difficulty and student strategic orientations to learning computer science. The paper suggests specific measures to alleviate programming anxiety and enhance the basic computational ability.

Introduction

Undergraduate learning is a major problem of the domain of education in computer science, as the rates of attrition are high, and the success of undergraduate students in the foundational classes in the field of programming varies (Pereira et al., 2020). The problem has been especially severe in first-year programming courses, sometimes also known as CS1, with global failure rates consistently reported to be high (between 28 per cent and 33 per cent) (Sobral & Oliveira, 2021; Utamachant et al., 2023). Such high failure rates have been a consistent issue and not only affect the students who face difficulties but also the pipeline of skilled professionals that are required in many technology-based industries (Bosse & ' Gerosa, 2017). In fact, it is a challenging task to learn the art of programming, as it requires not only an understanding of programming tools and languages but also advanced problem-solving abilities and effective program design and implementation approaches (Fagbola et al., 2019). This challenge makes the generalization of course failure rates in programming even more widespread, and such rates have surprisingly risen instead of declined over the years (Watson and ℕ atson, 2014). This is an alarming trend considering the fact that there is a shortage of competent professionals in highly advanced technologies worldwide as noted by the World Economic Forum (Zhidkikh et al., 2024). Certain courses of CS2 have actually experienced failure rates of more than 50 % within the last five years in certain areas, like in Thailand, which highlights the gravity and prevalence of this educational issue (Utamachant et al., 2023). These levels of attrition and failure are very worrisome because they call into question the efficacy of the existing pedagogical methods and the factors that lead to the inability of students to master basic computer science principles (Bennedsen & Caspersen, 2007). The objective of the paper is to quantitatively examine cognitive, affective, and pedagogical barriers to learning computer science at undergraduate level with an aim of providing empirical evidence on the complex nature of the challenges faced by students. Although these difficulties have been identified, detailed quantitative studies that directly focus on the interaction between cognitive and affective and pedagogical obstacles are still scarce (Watson and & Li, 2014).

Literature Review

This literature review collates the available literature to clarify the core factors that lead to student challenges in introductory computer science classes so that the complexity of cognitive, affective, and pedagogical factors is interconnected. Particularly, early computer science courses are more likely to lose students in the first year due to the inability to learn core concepts, and introductory programming courses have a failure rate of about 30 per cent (Berssanette and Francisco, 2021). Another difficulty related to these courses is that abstract thinking and logical reasoning are not always intuitive to students and are robust predictors of the success of programming, especially in object paradigm (McCulloh et al., 2025). Moreover, students often develop cognitive overload when trying to compile new syntactical rules, debugging techniques, and principles of fundamental programming, which aggravates their learning problems (Casillano, 2025). It is complicated and, as a consequence, can cause anxiety and lack of motivation, which affect academic performance and engagement (Ishaq et al., 2024). The rigorousness of the programing courses, which may seem challenging and overwhelming, may adversely affect the motivation and learning performance of students (Ishaq and Alvi, 2023). The latter is also complicated by the overall absence of pre-exposure to computer science concepts in many undergraduates, and the introductory notions are especially hard to learn in this situation (Nolan et al., 2019). The general inaccessibility of programming education, particularly in such topics as the abstraction and the program construction has been validated by the numerous responses of students

worldwide (Menolli and 7 Strik, 2025). Common myths and inability to grasp the basics of programming being an additional burden that students find difficult to get out of on their own (McCulloh et al., 2025). The inability to master syntax and semantics is one notable feature of this cognitive load, as beginning programmers easily commit mistakes and fail to understand the rules, which prevents them from writing meaningful code (McCulloh et al., 2025). In addition to syntax, students are usually challenged by so-called threshold concepts like recursion or object-oriented concepts, which are the radical changes of perception at the core of which, once mastered, will change the way they view the topic (Amiri and Islam, 2025).

Cognitive Barriers in Computer Science Education

These obstacles often include problem solving, algorithmic problem-solving and application of learned material to new situations (Yusoff et al., 2020). The process of program learning is by nature very complicated and demands that students learn new materials and strategies in programming, as well as train and train their skills in the field (Srivatanakul, 2022). The inherent cognitive load of the process of understanding programming constructions and syntax, and the usage of these provisions to very specific problems, also play a major role in this complexity (Kumar and Mohamad, 2023). The abstract concepts of programming and an extremely technical syntax, which adds to the cognitive load in particular novice programmers, complicate the situation (Mutero, 2018; Yan et al., 2025).

Affective Barriers in Computer Science Education

Emotions and motivation are important issues of determining the learning process and achievement among computer science students (Fan et al., 2025; Pabon and Machuca-Villegas, 2019). Such affective barriers may be in the form of loss of motivation, rise in anxiety, and a sense of alienation, which significantly affect the persistence and academic achievement of students (Belsam, 2017; Yao and Lin, 2025). Such negative feelings as frequent failures or the sense of incompetence may create a vicious cycle as students no longer want to participate in the learning process which in turn increases their challenges (Bain & Barnes, 2014).

Pedagogical Barriers in Computer Science Education

Instructional constraints, including the mismatch between teaching methods and student learning styles, overemphasis on the theoretical or theoretical-based learning and practical problem-solving, etc. are major obstacles to pedagogy (Rovshenov & Sarsar, 2023). As an example, the conventional classroom instructional approaches often cannot meet the needs of students with different backgrounds and learning speed, thus, hindering the acquisition of necessary problem-solving skills (Tilahun, 2022). Cognitive and affective obstacles can also be worsened by ineffective pedagogical practices, which results in a bimodal distribution of grades in introductory courses in which initial difficulties affect further academic outcomes considerably (Green, 2024). This may lead to premature disconnection and demotivation, especially when students have no concept of programming as a complicated and abstract skill (Coklar and Akcay, 2018).

Quantitative Approaches to Analyzing Learning Barriers

To gain a holistic picture on how to overcome these challenges, scholars have resorted to using quantitative research designs to determine the prevalence and effects of such barriers on learning outcomes of students. These methods are usually performed through statistical analysis of performance results, questionnaires of student attitudes and perceptions, and experimental designs that are used to investigate the effectiveness of various instructional interventions. Although qualitative research yields in-depth data on the experience of each specific person, comprehensive quantitative research is critical

to pinpointing widespread patterns and confirming that the challenges noticed in large groups of students are universal (Fan et al., 2025). As an illustration, quantitative research has continually found a positive relationship between self-efficacy and performance in programming; low-achieving students are found to have lower confidence in their abilities (Amoozadeh et al., 2024; Lakanen and Isomottonen, 2023). Besides, strict statistical examination of grades distributions may question the premises about student abilities, exposing bias of instructors instead of the lack of abilities (Malmi and Johri, 2023). Also based on such analyses is the empirical evidence of the role of previous mathematical ability in successful programming, and the necessity of underlying cognitive ability (Zingaro, 2014). This quantitative view points to the fact that strong mathematical background, specifically in the area of algebra, is necessary to teach students to succeed in introductory programming classes (McCulloh et al., 2025).

Conceptual Framework:

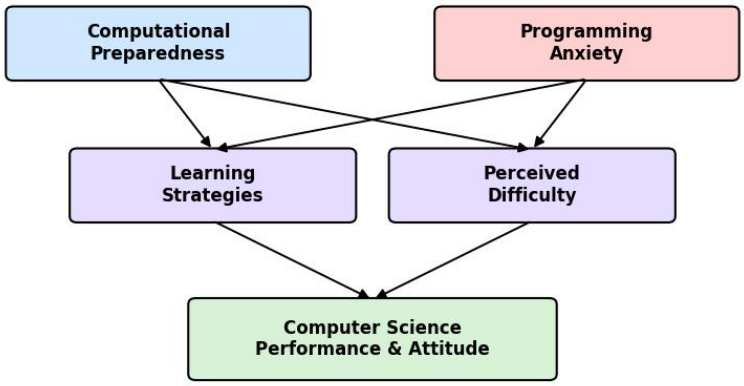


Fig. 01. Integrated Computer Science Learning Difficulty Model (ICSLDM)

Methodology:

The research was a descriptive study and its design was a cross-sectional survey to provide the quantitative data upon a sample. The target population included students in both the public sector and the private sector colleges in the 2025- 26 academic year. The sample size of 112 students was obtained online through a convenience sampling method. The sample size was restricted to the attock district, and all the respondents belonged to the private and government colleges. Sample demographics were 52 percent men, 48 percent women, 34 percent urban and 66 percent rural.

Data Collection Instruments

a) Integrated Computer Science Learning Difficulty Scale (ICSLDS)

A structured 5-point Likert-type scale was developed to measure the core constructs of the Integrated Computer Science Learning Difficulty Model. The instrument included both positively and negatively worded statements and was designed to yield quantifiable responses. The scale consisted of the following sections: demographic information, Perceived Computer Science Difficulty Scale (PCSDS), Computer Science Learning Strategy Scale (CSLSS) **(08 statements)**, Computer Science Self-Efficacy Scale (CSSES) **(06 statements)**, Programming Anxiety Scale (PrAS) **(06 statements)**, and Computational Readiness and Resources **(04 statements)**. The instrument was administered through a Google Form.

b) Development of Computational Readiness Diagnostic Test (CRDT)

To assess students' cognitive readiness for learning college-level computer science, a teacher-developed multiple-choice diagnostic test was constructed. The test covered prerequisite skills commonly expected

from earlier grades (e.g., basic mathematical reasoning, logical thinking, and problem-solving fundamentals relevant to introductory programming). Only concepts directly aligned with learning introductory computer science were included. The instrument was reviewed and validated by expert college-level computer science teachers to ensure content relevance and accuracy. Initially, **30 items** were drafted, and after review and refinement, **24 items** were retained in the final version of the test. A scoring and evaluation criterion was also developed and validated for consistent interpretation of results.

Table 1: ***Criteria to judge students' performance in the Computational Readiness Diagnostic Test (CRDT)***

Score Range	Performance Level	Description of Student Preparedness
85–100%	Very Well Prepared	Demonstrates strong mastery and high readiness for introductory college-level computer science (e.g., programming/problem-solving). Typically requires no intervention.
70–84%	Well Prepared	Understands most prerequisite concepts and can progress with regular instruction. May require minor support (e.g., targeted practice).
50–69%	Partially Prepared	Shows noticeable gaps in prerequisite reasoning/skills (e.g., logic, basic math for programming, algorithmic thinking). Requires remediation or structured bridging support.
Below 50%	Not Prepared	Lacks essential prerequisite knowledge needed for introductory computer science. Immediate remediation and foundational support are recommended.

Results

Table:

Correlations of the factors affecting students' performance in Computer Science (n = 112)
Note: * $p < .01$

Variable	n	M	S.D	1	2	3	4	5
1. Perceived Difficulty	112	19.10	4.70	—				
2. Learning Strategies	112	20.35	3.40	-0.29*	—			
3. Self-Efficacy	112	18.25	3.10	-0.33*	0.34*	—		
4. Programming Anxiety	112	12.05	3.60	0.46*	-0.06	-0.39*	—	
5. Computational Preparedness	112	21.10	3.75	-0.31*	0.28*	0.24*	-0.22*	—

Table 3: ***Hypotheses Testing Summary***

Hypothesis	Correlation (r)	p-value	Decision
H_{01}: No relationship between Computational Preparedness & Learning Strategies	0.28	.002	H_{01} rejected
H_{02}: No relationship between Programming Anxiety & Learning Strategies	-0.06	.520	H_{02} accepted
H_{03}: No relationship between Computational Preparedness & Perceived Difficulty of CS Topics	-0.31	.001	H_{03} rejected
H_{04}: No relationship between Learning Strategies & CS Performance/Attitude (Self-Efficacy)	0.34	<.001	H_{04} rejected
H_{05}: No relationship between Perceived Difficulty & CS Performance/Attitude (Self-Efficacy)	-0.33	<.001	H_{05} rejected

Table 3 indicates that **$H_{0\ 1}$ is rejected**, showing a statistically significant positive relationship: students with stronger **computational Readiness** tend to report using **more effective computer science learning strategies**. This suggests that foundational readiness (e.g., logic, basic mathematical reasoning for programming, and problem decomposition) is closely linked with strategic learning behaviors in CS. **$H_{0\ 2}$ is accepted**, indicating no statistically significant relationship between **programming anxiety** and

reported **learning strategies**. In other words, students' anxiety around coding does not reliably predict whether they claim to use effective strategies, even though anxiety may influence other outcomes (such as confidence or perceived difficulty).

$H_{0\ 3}$ is rejected, confirming a statistically significant negative relationship: students with higher **computational Readiness** perceive computer science topics as **less difficult**. Strong prerequisite skills appear to reduce the perceived complexity of programming and problem-solving tasks. **$H_{0\ 4}$ is rejected**, showing a statistically significant positive relationship: the use of **effective learning strategies** is associated with higher **computer science performance and attitude**, reflected here through stronger self-efficacy.

Finally, **$H_{0\ 5}$ is rejected**, demonstrating a statistically significant negative relationship: students who perceive CS topics as **more difficult** tend to show **lower self-efficacy** and a weaker performance/attitude profile. Perceived difficulty functions as a major barrier to success in introductory computer science.

Table 4: ***Comparison of mean scores of variables with benchmark (One-Sample t-test)***

Construct (n=112)	Test Value	t	df	Sig. (2-tailed)	Mean Difference	95% CI Lower	95% CI Upper
Perceived Difficulty	17	4.73	111	< .001	2.10	1.22	2.98
Learning Strategies	18	7.31	111	< .001	2.35	1.71	2.99
Self-Efficacy	21	-9.39	111	< .001	-2.75	-3.33	-2.17
Programming Anxiety	12	0.15	111	.883	0.05	-0.62	0.72
Computational Preparedness	18	8.75	111	< .001	3.10	2.40	3.80

Table 4 reports a set of one-sample t-tests comparing the sample mean of each construct with its benchmark (test value). The results show statistically meaningful differences for most variables.

(i) **Perceived Difficulty:** $t(111) = 4.73$, $p < .001$. Students' perceived difficulty of computer science topics was significantly higher than the benchmark. The positive mean difference (2.10) indicates that, on average, students experience CS content (especially introductory programming/problem-solving) as more difficult than the reference level.

(ii) **Learning Strategies:** Students scored significantly above the benchmark in the use of effective learning strategies, $t(111) = 7.31$, $p < .001$. The mean difference (2.35) suggests students report using more strategic, self-regulated approaches to learning than expected.

(iii) **Self-Efficacy:** Self-efficacy scores were significantly lower than the reference value, $t(111) = -9.39$, $p < .001$. This indicates students are not sufficiently confident about learning computer science, particularly programming tasks. Lower self-efficacy is consistent with weaker outcomes in both cognitive performance and affective engagement.

(iv) **Programming Anxiety:** There was no statistically significant difference between programming anxiety scores and the benchmark, $t(111) = 0.15$, $p = .883$. This suggests anxiety is present at approximately an average level relative to the reference point.

(v) **Computational Readiness:** Students' self-reported computational Readiness was significantly higher than the benchmark, $t(111) = 8.75$, $p < .001$. This implies that students generally believe they have the prerequisite skills/resources needed for learning college-level computer science.

To assess students' **computational Readiness** for learning computer science at the college level, a teacher-made **Computational Readiness Diagnostic Test (CRDT)** was developed and administered.

Diagnostic Test Performance

Students' performance on the diagnostic test was recorded and then evaluated using the established Readiness criteria. For comparison purposes, the **benchmark score was set at 70%**, which represents the **lower threshold of the "Well Prepared"** category.

Table 5: ***Descriptive Statistics of students for Computational Readiness Diagnostic Test (CRDT) performance***

Measure	N	Mean	Std. Deviation	Std. Error Mean
CPDT (%)	112	54.20	14.80	1.40

Table 6: ***One-Sample t-test Comparing CRDT Performance to Benchmark (Test Value = 70%)***

Measure	Test Value	t	df	Sig. (2-tailed)	Mean Difference	95% CI Lower	95% CI Upper
CPDT (%)	70	-11.30	111	< .001	-15.80	-18.57	-13.03

Table 7: ***Perceived Readiness vs CRDT (Paired Sample Correlations)***

Pair	Mean	N	Std. Deviation	Std. Error Mean	Correlation (r)	Sig.
Perceived Computational Preparedness (%)	72.50	112	13.20	1.25		
CPDT (%)	54.20	112	14.80	1.40	0.15	.110

Table 8: Comparison of Perceived Readiness and Actual Performance (Paired Differences)

Pair (Perceived − CPDT)	Mean Difference	Std. Deviation	Std. Error Mean	95% CI Lower	95% CI Upper	t	df	Sig. (2-tailed)
Preparedness Gap (%)	18.30	24.00	2.27	13.80	22.80	8.06	111	< .001

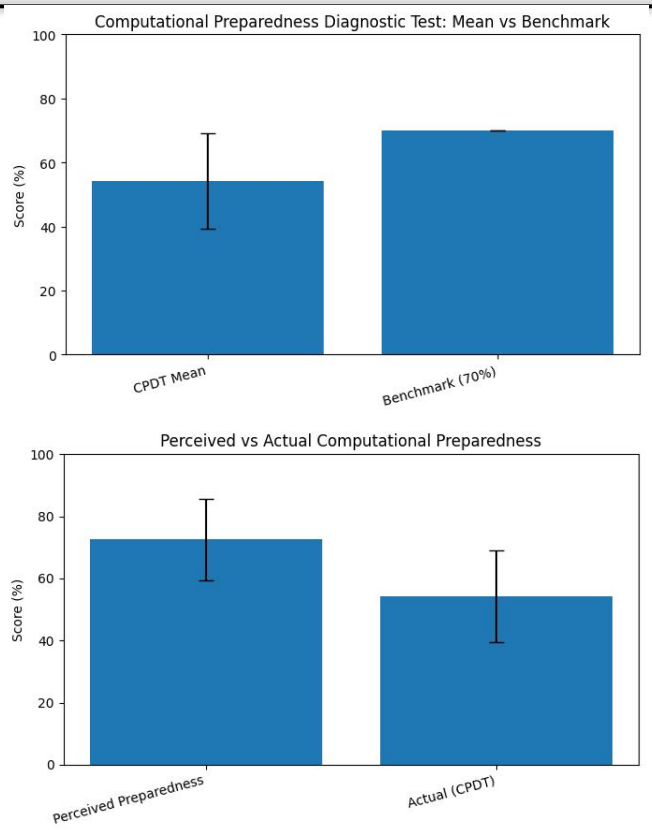
- Graph 1: **CRDT Mean vs 70% benchmark**
- Graph 2: **Perceived Readiness vs Actual (CRDT)**

(Shown above.)

One sample t -test was used to find out whether or not there was a significant change in the score of the students on the Computational Readiness Diagnostic Test (CRDT) as compared to the proficiency standard of 70%. The findings indicated that mean CRDT score **(M = 54.20, SD = 14.80)** was statistically less than 70, **t(111) = -11.30, p =.001**. The negative value of the mean difference **(-15.80)** shows that the students had actual mastery of the prerequisites that were lower than the set standard of proficiency, implying the lack of preparedness in the level of the basic reasoning and fundamental skills in introductory computer science (particularly programming/problem-solving).

Comparison of Readiness perceived vs actual performance

The perceived Readiness in students was rated quite high **(M = 72.50%),** whereas the performance in CRDT was **significantly lower (M = 54.20%).** The paired samples t-test has affirmed that there was a **statistically significant difference, t(111) = 8.06, p <.001,** with a **mean difference** of **18.30** percentage points**.** The relationship between perceived Readiness and CRDT scores did not show **significant correlation (r = 0.15, p =.110),** which meant that the self-perceptions of students were not quite the predictor of their actual computational readiness.



Discussion

The study investigated how thinking-related (cognitive) and emotion-related (affective) factors work together to shape achievement in an introductory, college-level computer science course. One consistent result stood out: many students felt confident that they were ready for CS, but their diagnostic results told a different story. In particular, students rated their computational Readiness as relatively strong (72.50%), yet their mean score on the Computational Readiness Diagnostic Test (CRDT) was much lower (54.20%) and fell well short of the 70% proficiency target. This mismatch between confidence and performance implies that a sizeable group of learners is misjudging their Readiness for foundational CS requirements, such as logical reasoning, the basic mathematical thinking embedded in programming, and algorithmic problem-solving. The finding supports recent work in computing education emphasizing that early measures of readiness and performance are essential for spotting students who require timely support, before repeated difficulties solidify into persistent failure trajectories.

Interpretation of Findings

The current results outline a complicated interaction of cognitive, affective, and pedagogical aspects, which the barriers to learning computer science are not independent but mutually dependent phenomena. This interdependence indicates the importance of the need to employ holistic intervention approaches which will simultaneously target various types of barriers instead of addressing them separately. As an example, the process of remediation of a cognitive barrier, e.g., the inability to debug (Lee et al., 2014), can also address the issue of affective problems, e.g., frustration and low self-efficacy. In addition, the discovery of counterproductive problem-solving patterns among the learners, which have frequently been dubbed as antipatterns, can be used to make pedagogic modifications that will help them develop more efficient debugging and problem-solving capabilities (Lee et al., 2014). According to the work of Prather et al. the generative AI tools may increase the gap between the well-

prepared and under-prepared students but can be used to assist them to be more metacognitive and problem-solving (Amoozadeh et al., 2024).

Comparison with Existing Literature

The discussion of diverse barriers to multifaceted learning of computer-science presented in the current study is consistent with existing studies that emphasize the importance of cognitive load and abstract thinking in the programming education process (Green, 2024). In particular, the abstractness of constructs in programming and a significant cognitive load in tackling a problem have been established as the most obvious pitfalls (Recsky et al., 2024). Moreover, our results are also aligned with the research highlighting mathematics competency as a primary indicator of the achievement in introductory programming classes, thus supporting the necessity of learners being equipped with strong foundational quantitative skills (Sadik et al., 2020). These findings confirm the need to have some form of mathematical background to engage with concepts of computers productively, beyond a strictly vocational definition of computer-science education.

Implications for Computer Science Education

Such findings justify the introduction of specific metacognitive assistance and emotional regulation interventions into the curriculum that would enhance resilience and learning outcomes (Li et al., 2024). This may include the use of interactive learning modes and use of games to improve motivation and fun thus reducing affective barriers (Pachumwon et al., 2025). In addition, generative AI systems have shown potential to support the scaffolding of student learning by providing syntactically accurate and complete code completion, syntax errors with explanation feedback that can reduce the cognitive load of programming (Margulieux et al., 2024).

Limitations of the Study

Although the current research provides useful quantitative results, its overall applicability may be limited by the sample demographic and the institution-related setting. The study needs to be extended in future to cover a wider range of institutions and student groups to confirm the generalizability of these results. Also, whereas the current study found out the correlations between different barriers, causal relationship would require longitudinal designs and controlled experimental interventions. In addition, even though AI assistance can be helpful, it has been shown that too much overuse of these tools can impede the process of independent problem-solving and critical thinking, especially in less experienced learners (Bienefeld et al, 2025; Rojas-Galeano et al, 2025).

Recommendations

1. Enhance computational requirements with remedial and bridging assistance.

Computational Readiness must be seen as a pre-requisite to CS success that cannot be compromised. Colleges ought to introduce short bridging courses (logic, simple math-for-programming, algorithmic thinking, tracing) and rely on early diagnostic feedback to focus on remediation prior to entry into core programming courses.

2. Discuss affective factors directly (anxiety and confidence, in particular)

Through positive classroom culture (safe error culture), low-stakes practice in classrooms, and organised help-seeking routines, active instruction should aim to mitigate programming anxiety and low self-efficacy. There is systematic evidence indicating that programming anxiety is a quantifiable obstacle that can be curtailed through specific intervention.

3. Change to student-centered, high-practice teaching that makes it sound less difficult.

The perceived difficulty can be reduced as students repeatedly get manageable success by means of guided labs, worked examples, and progressively increasing difficulty in the task. Since debugging is a significant obstacle in beginning programmers, explicit teaching on debugging (strategies, checklists, tracing practices) must be integrated into the teaching instead of presumed as an implicit skill.

4. Expand beyond cognitive scores to measure SRL, confidence and perceived difficulty.

The conventional evaluation is usually only capture of output (grades), as opposed to the learning process. Regular monitoring of self-efficacy and perceived difficulty of students, as well as the use of the strategy paired with diagnostic checkpoints, can assist the instructors to detect the patterns that are at risk, and to attempt to prevent disengagement or failure prior to its onset.

References

Amiri, S., & Islam, M. M. (2025). Enhancing Python Programming Education with an AI-Powered Code Helper: Design, Implementation, and Impact. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2509.20518

Amoozadeh, M., Nam, D., Prol, D., Alfageeh, A., Prather, J., Hilton, M., Ragavan, S. S., & Alipour, M. A. (2024a). Student-AI Interaction: A Case Study of CS1 students. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2407.00305

Amoozadeh, M., Nam, D., Prol, D., Alfageeh, A., Prather, J., Hilton, M., Ragavan, S. S., & Alipour, M. A. (2024b). *Student-AI Interaction: A Case Study of CS1 students*. 1. https://doi.org/10.1145/3699538.3699567

Bain, G. M., & Barnes, I. (2014). *Why is programming so hard to learn?* 356. https://doi.org/10.1145/2591708.2602675

Belsam. (2017). Visualizing Computer Programming in a Computer-based Simulated Environment. *International Journal of Advanced Computer Science and Applications*, *8*(8). https://doi.org/10.14569/ijacsa.2017.080848

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*(2), 32. https://doi.org/10.1145/1272848.1272879

Berssanette, J. H., & Francisco, A. C. de. (2021). Active Learning in the Context of the Teaching/Learning of Computer Programming: A Systematic Review [Review of *Active Learning in the Context of the Teaching/Learning of Computer Programming: A Systematic Review*]. *Journal of Information Technology Education Research*, *20*, 201. Informing Science Institute. https://doi.org/10.28945/4767

Bienefeld, N., Keller, E., & Grote, G. (2025). *AI Interventions to Alleviate Healthcare Shortages and Enhance Work Conditions in Critical Care: Qualitative Analysis*.

Bosse, Y., & Gerosa, M. A. (2017). Why is programming so difficult to learn? *ACM SIGSOFT Software Engineering Notes*, *41*(6), 1. https://doi.org/10.1145/3011286.3011301

Casillano, N. F. B. (2025). Extracting Error Resolution Patterns for Novice Programming Students using Apriori Algorithm. *Journal of Information Systems Engineering & Management*, *10*, 567. https://doi.org/10.52783/jisem.v10i24s.3942

Çoklar, A. N., & Akçay, A. (2018). Evaluating programming self-efficacy in the context of inquiry skills and problem-solving skills: A perspective from teacher education. *World Journal on Educational Technology Current Issues*, *10*(3), 153. https://doi.org/10.18844/wjet.v10i3.3556

Fagbola, T. M., Adeyanju, I., Olaniyan, O., Esan, A., Omodunbi, B., Oloyede, A., & Egbetola, F. (2019). Development of Mobile-Interfaced Machine Learning-Based Predictive Models for Improving

Students Performance in Programming Courses. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1901.06252

Fan, G., Liu, D., Zhang, R., & Pan, L. (2025). The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, *12*(1). https://doi.org/10.1186/s40594-025-00537-3

Green, K. J. (2024). AI-Driven Programming Education for Black, Latinx, and Afro-Caribbean Communities. *Deep Blue (University of Michigan)*. https://doi.org/10.7302/23676

Ishaq, K., & Alvi, A. (2023). Personalization, Cognition, and Gamification-based Programming Language Learning: A State-of-the-Art Systematic Literature Review. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2309.12362

Ishaq, K., Alvi, A., Haq, M. I. ul, Rosdi, F., Choudhry, A. N., Anjum, A., & Khan, F. (2024). Level up your coding: a systematic review of personalized, cognitive, and gamified learning in programming education [Review of *Level up your coding: a systematic review of personalized, cognitive, and gamified learning in programming education*]. *PeerJ Computer Science*, *10*. PeerJ, Inc. https://doi.org/10.7717/peerj-cs.2310

Kumar, J. R., & Mohamad, F. S. (2023). Interventions for Enhancing Indigenous Undergraduates' Programming Learning: A Systematic Review [Review of *Interventions for Enhancing Indigenous Undergraduates' Programming Learning: A Systematic Review*]. *Journal of Cognitive Sciences and Human Development*, *9*(1), 83. UNIMAS Publisher. https://doi.org/10.33736/jcshd.4902.2023

Lakanen, A.-J., & Isomöttönen, V. (2023). CS1: Intrinsic Motivation, Self-Efficacy, and Effort. *Informatics in Education*. https://doi.org/10.15388/infedu.2023.26

Lee, M. J., Bahmani, F., Kwan, I., LaFerte, J., Charters, P., Horvath, A., Luor, F., Cao, J., Law, C., Beswetherick, M., Long, S., Burnett, M., & Ko, A. J. (2014). *Principles of a debugging-first puzzle game for computing education*. 57. https://doi.org/10.1109/vlhcc.2014.6883023

Li, Y., Chen, M., Hunt, A., Zhang, H., & O'Rourke, E. (2024). *Exploring the Interplay of Metacognition, Affect, and Behaviors in an Introductory Computer Science Course for Non-Majors*. 27. https://doi.org/10.1145/3632620.3671119

Malmi, L., & Johri, A. (2023). A Selective Review of Computing Education Research [Review of *A Selective Review of Computing Education Research*]. *Routledge eBooks*, 573. Informa. https://doi.org/10.4324/9781003287483-31

Margulieux, L. E., Prather, J., Reeves, B. N., Becker, B. A., Uzun, G. C., Loksa, D., Leinonen, J., & Denny, P. (2024). *Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems*. 276. https://doi.org/10.1145/3649217.3653621

McCulloh, I., Rodríguez, P., Kumar, S., Gupta, M., Sharma, V., Johnson, B., & Johnson, A. N. (2025). Generative AI in Computer Science Education: Accelerating Python Learning with ChatGPT. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2505.20329

Menolli, A., & Strik, B. (2025). Educational Insights from Code: Mapping Learning Challenges in Object-Oriented Programming through Code-Based Evidence. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2507.17743

Mutero, T. (2018). An Investigation on the Challenges Faced in Teaching and Learning Computers in Higher Education. *International Journal of Engineering Research And*, *7*. https://doi.org/10.17577/ijertv7is070143

Nolan, K., Bergin, S., & Mooney, A. (2019). *An Insight Into the Relationship Between Confidence, Self-efficacy, Anxiety and Physiological Responses in a CS1 Exam-like Scenario*. 1. https://doi.org/10.1145/3351287.3351296

Pabón, O. S., & Machuca-Villegas, L. (2019). Fostering Motivation and Improving Student Performance in an introductory programming course: An Integrated Teaching Approach. *Revista EIA*, *16*(31), 65. https://doi.org/10.24050/reia.v16i31.1230

Pachumwon, T., Jantakoon, T., & Laoha, R. (2025). A Design Thinking-Driven Conceptual Framework for a Creative AI Learning Environment to Enhance Programming Skills (CAILE). *Higher Education Studies*, *15*(4), 312. https://doi.org/10.5539/hes.v15n4p312

Pereira, F. D., Oliveira, E., Oliveira, D. B. F. de, Cristea, A. I., Carvalho, L. S. G. de, Fonseca, S. C., Toda, A. M., & Isotani, S. (2020). Using learning analytics in the Amazonas: understanding students' behaviour in introductory programming. *British Journal of Educational Technology*, *51*(4), 955. https://doi.org/10.1111/bjet.12953

Recsky, C., Rush, K. L., MacPhee, M., Stowe, M., Blackburn, L., Muniak, A., & Currie, L. M. (2024). Clinical Informatics Team Members' Perspectives on Health Information Technology Safety After Experiential Learning and Safety Process Development: Qualitative Descriptive Study. *JMIR Formative Research*, *8*. https://doi.org/10.2196/53302

Rojas-Galeano, S., Tejada, J., & Marmolejo-Ramos, F. (2025). Between Tool and Trouble: Student Attitudes Toward AI in Programming Education. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2508.05999

Rovshenov, A., & Sarsar, F. (2023). Research trends in programming education: A systematic review of the articles published between 2012-2020 [Review of *Research trends in programming education: A systematic review of the articles published between 2012-2020*]. *Journal of Educational Technology and Online Learning*, *6*(1), 48. Süleyman Demirel University. https://doi.org/10.31681/jetol.1201010

Sadık, O., Ottenbreit-Leftwich, A., & Brush, T. (2020). Secondary Computer Science Teachers' Pedagogical Needs. *International Journal of Computer Science Education in Schools*, *4*(1), 33. https://doi.org/10.21585/ijcses.v4i1.79

Sobral, S. R., & Oliveira, C. (2021). PREDICTING STUDENTS' PERFORMANCE IN INTRODUCTORY PROGRAMMING COURSES: A LITERATURE REVIEW [Review of *PREDICTING STUDENTS' PERFORMANCE IN INTRODUCTORY PROGRAMMING COURSES: A LITERATURE REVIEW*]. *INTED Proceedings*, *1*, 7402. International Academy of Technology, Education and Development. https://doi.org/10.21125/inted.2021.1485

Srivatanakul, T. (2022). Emerging from the pandemic: instructor reflections and students' perceptions on an introductory programming course in blended learning. Education and Information Technologies, 28(5), 5673. https://doi.org/10.1007/s11239-022-11328-6

Tilahun, T. (2022). ENHANCING THIRD-YEAR COMPUTER SCIENCE STUDENTS' COMPUTER PROGRAMMING SKILLS AT HAWASSA UNIVERSITY, ETHIOPIA. *International Journal of Advanced Research in Computer Science*, *13*(6), 26. https://doi.org/10.26483/ijarcs.v13i6.6934

Utamachant, P., Anutariya, C., & Pongnumkul, S. (2023). i-Ntervene: applying an evidence-based learning analytics intervention to support computer programming instruction. *Smart Learning Environments*, *10*(1). https://doi.org/10.1186/s40561-023-00257-7

Watson, C., & Li, F. W. B. (2014). *Failure rates in introductory programming revisited*. 39. https://doi.org/10.1145/2591708.2591749

Yan, Y.-M., Chen, C., Hu, Y., & Ye, X. (2025). LLM-based collaborative programming: impact on students' computational thinking and self-efficacy. *Humanities and Social Sciences Communications*, *12*(1). https://doi.org/10.1057/s41599-025-04471-1

Yao, D., & Lin, J. (2025). Cognitive enhancement through competency-based programming education: a 12-year longitudinal study. *Education and Information Technologies*. https://doi.org/10.1007/s10639-025-13582-w

Yusoff, K. M., Ashaari, N. S., Siti, T., & Mohd, N. (2020). Analysis on the Requirements of Computational Thinking Skills to Overcome the Difficulties in Learning Programming. *International Journal of Advanced Computer Science and Applications*, *11*(3). https://doi.org/10.14569/ijacsa.2020.0110329

Zhidkikh, D., Heilala, V., Petegem, C. V., Dawyndt, P., Järvinen, M., Viitanen, S., Wever, B. D., Mesuere, B., Lappalainen, V., Kettunen, L., & Hämäläinen, R. (2024). Reproducing Predictive Learning Analytics in CS1. *Journal of Learning Analytics*, *11*(1), 132. https://doi.org/10.18608/jla.2024.7979

Zingaro, D. (2014). *Peer instruction contributes to self-efficacy in CS1*. https://doi.org/10.1145/2538862.2538878