A COGNITIVE LOAD–DRIVEN APPROACH TO PROGRAMMING EDUCATION: FROM DIAGNOSIS TO DESIGN

[1]KRISHNA PALOD, [2]NISHITA PAREKH, [3]PRAVIN SHRINATH

[1]Ph.D Scholar, Technology Management Department, Mukesh Patel School of Technology Management & Engineering, NMIMS (Deemed-to-be-University), Mumbai, India
[2]Assistant Professor, Technology Management Department, Mukesh Patel School of Technology Management & Engineering, NMIMS (Deemed-to-be-University), Mumbai, India
[3]Associate Professor, Computer Engineering Department, Mukesh Patel School of Technology Management & Engineering, NMIMS (Deemed-to-be-University), Mumbai, India
E-mail: [1]Krishna.Palod01@nmims.in

Abstract - Today's software industry demands graduates who are competent at designing, implementing, and debugging software. Academic institutions find high failure rates persist in introductory programming courses. Addressing this gap requires more than additional practice; it demands a deeper understanding of the cognitive challenges that novices encounter. Cognitive Load Theory (CLT), which distinguishes intrinsic (content complexity), extraneous (instructional barriers), and germane (schema-building) loads, provides a framework for targeted instructional design.

In the two-phase study, programming difficulties reported by 383 undergraduates were first mapped onto CLT categories. Analyses revealed that intrinsic difficulties such as understanding concepts, designing algorithms, writing code, and interpreting problem statements were most prevalent. Students also reported extraneous barriers including limited guidance, scarce resources and time pressure along with germane challenges such as inadequate practice and debugging. Based on these findings, authors proposed a CLT-based Difficulty-Strategy Matrix, linking each difficulty type to tailored pedagogical interventions.

In Phase 2, this matrix guided the design of a workshop delivered to 85 students. Pre- and post-testing, with a paired t-test, indicated statistically significant improvement in programming competency. This CLT-driven model provides educators with a practical, replicable process: diagnose student difficulties, map them to evidence-based strategies, implement targeted instruction, and evaluate learning outcomes effectively.

Keywords - Cognitive Load Theory, Programming Education, Instructional Design, Difficulty-StrategyMatrix, Programming Competency

I. INTRODUCTION

In today's era of digital transformation, programming competency has become a critical skill across both industry and education. According to the World Economic Forum (2023), there is a growing global shortage of professionals with advanced programming expertise, underscoring the essential role of programming skills in sustaining and advancing modern digital ecosystems. Yet, introductory programming courses continue to experience high failure and dropout rates worldwide—often reaching up to 50% (Figueiredo&García-Peñalvo, 2024; Margulieux, Morrison, & Decker, 2020). Despite ongoing curricular reforms and instructional innovations, many students still struggle to understand abstract concepts, design algorithms, and debug code effectively (Ngadengon et al., 2025; Weeda et al., 2023). These persistent difficulties suggest that programming challenges extend beyond content complexity to include the cognitive demands imposed by programming tasks.

Cognitive Load Theory (CLT) provides a theoretical lens to explain why novice programmers struggle with complex learning activities. It emphasizes that learning effectiveness depends on managing the limited capacity of working memory and efficiently organizing information in long-term memory (Sweller et al., 2019). By distinguishing intrinsic load (content complexity), extraneous load (instructional barriers), and germane load (schema construction), CLT enables educators to design instruction that minimizes unnecessary cognitive effort and fosters meaningful learning.

Recent scholarship has increasingly explored the role of CLT in programming education. Berssanette and de Francisco (2022) conducted a systematic review analysing how CLT principles have been applied in programming pedagogy and found that most studies reference CLT conceptually rather than operationalizing its three load categories in instructional interventions. Similarly, Butarev (2024) demonstrated that structured application of CLT—through visualization techniques and guided instructional design—reduces cognitive load and enhances learning outcomes. Salleh, Shukur, and Judi (2018) further supported this by showing that a multi-phase scaffolding model, progressing from worked examples to independent problem solving, effectively manages cognitive load for novice programmers. However, existing research largely remains theoretical or limited to partial implementations. Few studies have systematically classified student-reported programming difficulties or translated these classifications into comprehensive instructional design models. This limitation constrains the

development of evidence-based pedagogical frameworks grounded in cognitive theory.

To address this gap, the present study applies CLT as both a diagnostic and instructional framework for programming education. In Phase 1, programming difficulties reported by undergraduates were systematically categorized into intrinsic, extraneous, and germane cognitive load types. In Phase 2, these classifications informed the creation of a CLT-based Difficulty–Strategy Matrix, implemented through a structured workshop integrating worked examples, visualization tools, guided practice, and gamified reflection.

The study contributes by providing:
1. Empirical evidence—identifying which cognitive load types most affect novice programmers
2. A validated instructional model—demonstrating how CLT-aligned interventions can improve programming competency.

By bridging cognitive science and programming pedagogy, this research presents a replicable framework for educators and curriculum designers seeking to reduce cognitive overload, enhance conceptual understanding, and improve learning outcomes in introductory programming.

II. LITERATURE REVIEW

2.1 Challenges in Introductory Programming

High failure and dropout rates in foundational programming courses are well documented. A global analysis of 161 institutions reported an average failure rate of 28% in introductory programming classes, only a modest improvement from 33% a decade earlier (Bennedsen&Caspersen, 2019). Beyond attrition, research consistently highlights novice programmers' persistent struggles with abstract concepts, algorithm design, problem comprehension, and debugging (Cheah, 2020; Anindyaputri et al., 2020). Ngadengon et al. (2025) identified difficulties in "creating algorithms, recognizing problems, devising solutions, and implementing ideas through coding," as well as challenges in syntax understanding and error diagnosis. Sim and Lau (2022) synthesized these findings, noting that cognitive overload, syntax and logic misconceptions, and motivational barriers remain primary obstacles. They emphasized the importance of integrated instructional designs—such as scaffolding, adaptive feedback, and peer learning—that address both cognitive and affective dimensions of programming education.

2.2 Cognitive Load Theory in Education

Cognitive Load Theory (CLT) provides a robust framework for understanding these challenges by explaining how instructional design can optimize limited working-memory resources. CLT distinguishes among intrinsic load (task complexity), extraneous load (instructional inefficiencies), and germane load (schema-building processes). Sweller et al., (2019) demonstrated that managing these loads through worked examples, minimizing irrelevant information, and promoting germane processing can substantially enhance learning in cognitively demanding domains.

2.3 CLT Applications in Programming Pedagogy

Interest in CLT within programming education has grown in recent years, yet many studies reference the theory conceptually rather than operationalizing its load categories in instructional interventions. Building on CLT principles, Looker (2021) integrated Social Constructivist elements—such as peer collaboration and co-constructed problem solving—into a CLT-informed pedagogical framework. This approach sequences tasks to regulate intrinsic and extraneous load while stimulating germane processing through collaboration. Winter et al. (2019) likewise demonstrated that interactive code-execution visualizations improve novice comprehension of cognitively demanding topics such as pointers and recursion, leading to faster schema acquisition and fewer debugging errors. Extending this research, Butarev (2024) confirmed that visualization-based instructional design grounded in CLT effectively reduces cognitive load and enhances conceptual understanding among novice programmers.

2.4 Gaps in CLT-Based Instructional Design

Despite these advances, few studies have systematically classified student-reported programming difficulties according to CLT's taxonomy or translated those classifications into comprehensive instructional frameworks. Existing works often focus on isolated CLT-informed strategies or general theoretical discussions, leaving a gap in evidence-based, theory-driven curriculum design that addresses the full spectrum of cognitive load challenges in programming education.

The literature underscores both the persistence of novice programming difficulties and the promise of CLT-guided instructional interventions. However, the absence of an integrated approach—one that diagnoses specific cognitive load challenges and maps them to targeted pedagogical strategies—limits the development of replicable, empirically grounded programming pedagogy. The present study addresses this gap by classifying student-reported difficulties into intrinsic, extraneous, and germane load categories and constructing a CLT-based Difficulty–Strategy Matrix to guide empirical instructional design.

III. METHODOLOGY

This study followed a two-phase design: (1) to identify and classify novice programming difficulties using Cognitive Load Theory (CLT) and (2) to design and evaluate a targeted instructional intervention informed by that classification.

3.1 Phase 1 – Difficulty Elicitation and CLT Classification

A total of 383 undergraduate students enrolled in introductory programming courses completed an online survey. They were presented with a predefined list of common programming difficulties—such as understanding concepts, developing algorithms and logic, implementing concepts, understanding problem statements, writing code, debugging and troubleshooting, lack of practice, lack of support or resources, lack of time, and lack of interest—and asked to select all that applied to their own learning experience.

Frequency counts were calculated for each difficulty. Two independent raters then classified the difficulties into CLT categories: intrinsic load (content complexity), extraneous load (avoidable instructional barriers), and germane load (productive effort for schema construction).Inter-rater reliability was assessed using Cohen's kappa coefficient ($\kappa = 0.72$), indicating substantial agreement between raters. Discrepancies were resolved through discussion until consensus was reached. The resulting classification is shown in table 1.

CLT Load	Description	Difficulties Identified
Intrinsic Load (Managing content complexity)	Cognitive effort required to understand inherently complex programming content and relationships between concepts.	Understanding programming concepts Understanding problem statements Developing algorithm & logic of the problem Implementing the concepts Writing code
Extraneous Load (Reducing avoidable barriers)	Cognitive effort caused by poorly designed instruction or lack of support/resources, which can be minimized.	Lack of support/guidance Lack of resources Lack of time
Germane Load (Promoting schema building)	Productive effort devoted to constructing and refining mental models and problem-solving strategies.	Debugging & troubleshooting Lack of practice

Table 1: Difficulties Classification into Cognitive Load Theory Category

3.2 Phase 2 – CLT-Based Difficulty–Strategy Matrix and Workshop

Findings from Phase 1 informed the development of a CLT-based Difficulty–Strategy Matrix as reported in table 2, linking each difficulty category to specific pedagogical techniques and tools.

CLT Load	Difficulty	Aligned Techniques / Tools
Intrinsic Load(Managing content complexity)	Understanding Programming concepts	Stepwise code walkthroughs Worked examples VISUALGO – Algorithm Tool (visualize data flow)
	Understanding problem statement	Case Study (realistic scenarios) ProcessOn-Mapping Tool (concept Maps)
	Developing algorithm & logic of the problem	Code Pictionary (code comprehension game) PROBSOL Tool (problem-solving tool)
	Implementing the concepts	Socratic Q&A for Implementation
	Writing code	Live Coding Challenge Scaffolded coding templates
Extraneous Load(Reducing avoidable barriers)	Lack of support/guidance	Peer Mentoring Structured Coding guides
	Lack of resources	Sololearn Platform (accessible content), Virtual labs
	Lack of time	Chunked Live Coding Challenges (time-boxed) Self-paced modules
Germane Load(Promoting schema building)	Debugging & troubleshooting	Debug the Output RoboBug – Debugging Game Reflective Code Journal

	Lack of practice	Gamified Tools – Lightbot Game, Code Saga Game Incremental Hands-on Challenges

Table 2: CLT-Based Difficulty-Strategy Matrix

Workshop participants (N=85) wereselected from a pool of eligible students enrolled in introductory programming courses at the same institution. Each session lasted 3 hours with groups of 20-25 students to ensure adequate interaction and personalized attention. This matrix guided a three-session workshop for these undergraduates.

Session 1 – Managing Intrinsic Load:
The first session addressed intrinsic load by helping students manage content complexity, targeting difficulties such as understanding concepts, interpreting problem statements, developing algorithms, implementing ideas, and writing code. Pedagogical strategies included stepwise code walkthroughs, worked examples, Socratic Q&A, case studies, concept mapping, code pictionary, and live coding challenges supported by tools such as VISUALGO, ProcessOn, and PROBSOL.

Session 2 – Reducing Extraneous Load:
It focused on reducing extraneous load by minimizing avoidable barriers like lack of guidance, limited resources, and time pressure through peer mentoring, structured coding guides, chunked live challenges, and self-paced learning modules using tools such Sololearn, virtual labs.

Session 3 – Promoting Germane Load:
It promoted germane load to encourage deep learning and schema construction by tackling debugging and practice deficits through debug the output task, reflective code journals, incremental coding challenges, and gamified activities with RoboBug, Lightbot, Code Sage tools.

3.3 Evaluation of Learning Outcomes

A Programming Competency Test consisting of 21 items covering fundamental programming concepts such as logic, problem-solving, syntax and semantics and debugging was administered before and after the workshop. The test was validated through expert review and demonstrated acceptable internal consistency (Cronbach'sα = 0.67).Paired t-tests were used to determine whether participation in the CLT-driven intervention produced a statistically significant improvement in programming competency.

IV. RESULTS

4.1 Phase 1: Programming Difficulties Identification and CLT Classification

The first phase aimed to identify and systematically classify programming difficulties reported by novice students using Cognitive Load Theory framework.A total of 383 undergraduate students participated in Phase 1 (M age = 20.1 years, SD = 1.2). The sample comprised 58% male and 42% female students. Participants reported multiple difficulties from the predefined list. Table 3 presents the frequency and percentage of each difficulty type.

Difficulty	Frequency	Percentage (%)
Debugging & trouble shooting	114	29.8
Developing algorithm & logic of the problem	158	41.3
Implementing the concepts	161	42.0
Understanding problem statement	142	37.1
Writing code	131	34.2
Understanding Programming Concepts	166	43.3
Lack of Practice	151	39.4
Lack of resources	88	23.0
Lack of support/guidance	127	33.2
Lack of time	132	34.5

Table 3: Frequency and Percentage of Reported Programming Difficulties (N=383)

Frequencies showed that intrinsic barriers such as understanding programming concepts (43%), implementing concepts (42%), and developing algorithms and logic (41%) were the most prevalent, followed by extraneous barriers (e.g., lack of support/guidance 33%, lack of resources 23%, time pressure 34%) and germane challenges (debugging & troubleshooting 30%, lack of practice 39%).

Figure 1 illustrates the distribution of programming difficulties reported by students, classified according to Cognitive Load Theory. Intrinsic load represented the largest share (55.3%), indicating that most challenges stem from the inherent complexity of programming concepts and tasks.

This shows that core content complexity is the dominant challenge for novice programmers. Extraneous load accounted for 25.3%, reflecting avoidable barriers such as limited guidance, inadequate resources, and time constraints.These problems are not about the programming content itself but about how learning is supported; better instructional design can reduce this load.

Germane load comprised 19.3%, showing that many students lacked opportunities for effective debugging and practice to build robust mental schemas. Participants reported that they lacked sufficient practice and reflective debugging opportunities, which are crucial for deeper learning.Thisdistribution

informed the subsequent CLT-based Difficulty– Strategy Matrix.

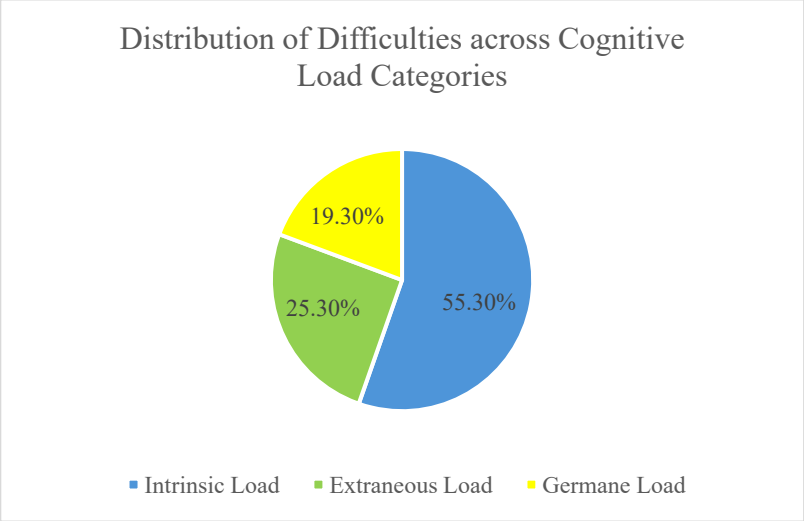

Figure 1: Distribution of Difficulties across Cognitive Load Categories

4.2 Phase 2: Workshop Intervention Outcomes

Descriptive Analysis

The descriptive statistics were calculated to inspect the distribution of scores before and after the CLT-based instructional workshop for programming ability. The results are presented in the table 4.

Descriptive Statistics					
	N	Minimum	Maximum	Mean	Std. Deviation
Programming Pre Score (out of 21)	85	2	14	7.41	2.7
Programming Post Score (out of 21)	85	5	18	11.18	3

Table 4: Descriptive Statistics

Results revealed that students' programming scores increased from a pre-test mean of 7.41 (SD = 2.7) to a post-test mean of 11.18 (SD = 3.0).The increase in mean scores indicates a clear improvement in programming competency following the CLT-based instructional intervention.The mean programming competency score improved by approximately 50.8% after the CLT-based instructional workshop.The programming competency scores were paired, continuous, and collected from the same participants before and after the workshop. Skewness (pre = 0.24, post = 0.01) and kurtosis (pre = –0.79, post = –0.65) values fell within the acceptable –2 to +2 range, indicating normal distribution and suitability for paired t-test analysis.

Measure	Mean	SD	T	Df	p (two-tailed)	Cohen's d
Pre-test	7.41	≈2.75				
Post-test	11.19	≈3.00	17.14	84	< .001	1.86

Table 5: Paired Sample t-test for Programming Competency

Table 5 shows the results of paired sample t-test. A paired t-test showed post-test scores (M = 11.19, SD = 3.00) were significantly higher than pre-test scores (M = 7.41, SD = 2.75), t(84) = 17.14, p< .001.Cohen's d was 1.86, indicating a very large effect size.These results show that the CLT-driven Difficulty–Strategy Matrix–guided workshop produced a substantial and statistically significant improvement in programming competency.

V. DISCUSSION

This study provides robust empirical evidence that Cognitive Load Theory (CLT) -guided instructional design significantly enhances programming education outcomes beyond typical educational interventions.The systematic classification of student-reported difficulties and the subsequent intervention outcomes validate theoretical predictions while offering practical solutions to persistent pedagogical challenges.

The predominance of intrinsic load difficulties (55.3%) aligns with prior research. Li (2016) observed that novice programmers are easily

overwhelmed by new concepts and tools, supporting the present study's findings that conceptual understanding (43.3%) and implementation (42.0%) are key challenges. Similarly, Issever et al. (2023)demonstrated that programming tasks generate measurable cognitive load. These results provide large-scale empirical evidence of specific intrinsic barriers in novice programming.

The large effect size (Cohen's d = 1.86) achieved by this CLT-based intervention substantially exceeds Cohen's benchmark of d = 0.8 for large effects and surpasses typical educational outcomes.While Sandoval-Medina et al. (2024)reported positive but smaller effects with CLT-based materials, these findings suggest that comprehensive multi-dimensional CLT application—addressing intrinsic, extraneous, and germane load simultaneouslyyields stronger gains than single-principle approaches.

The effectiveness of worked examples and case studies in reducing intrinsic load supports prior research on example-based learning. Menon (2023) found that worked-out code examples enhance novice problem-solving ability, while the systematic review by Sibia et al. (2024) demonstrated that visualization tools improve learning outcomes by reducing cognitive load through interactive, well-structured visual aids. The success of visualization tools such as VISUALGO, Code Pictionary, and concept mapping in this study aligns with these findings.

The categorization of programming difficulties within the CLT framework reinforces the theoretical foundations of cognitive load research. The 25.3% proportion of extraneous load challenges corresponds with Issever et al. (2023), highlighting the impact of instructional design on cognitive load. Issues such as insufficient support, time pressure, and inadequate resources indicate key areas for enhancing instructional environments.

The findings also provide empirical support for the framework advanced by Sweller et al. (2019), emphasizing that effective instructional design should align with human cognitive architecture. This intervention operationalized CLT principles by managing intrinsic load through worked examples and visualization, reducing extraneous load through structured guidance, and fostering germane load through reflective practice—leading to significant learning gains that validate the theory. Furthermore, this study extends the work of Gkintoni et al. (2025) by offering large-scale empirical evidence that CLT-guided interventions can achieve exceptionally high effectiveness (d = 1.86) in complex learning domains such as programming education.

While the results are promising, the study has certain limitations. The study's single-institution scope limits generalization, and the short duration restricts assessment of long-term learning effects. Future research should employ multi-institutional, randomized, and longitudinal approaches to further validate the effectiveness of CLT-based programming instruction.

VI. CONCLUSION

This study demonstrates that Cognitive Load Theory (CLT)–based instructional design can enhance programming education outcomes. By classifying novice difficulties into intrinsic, extraneous, and germane loads and applying a CLT-based Difficulty–Strategy Matrix, it provides a validated framework that aligns teaching strategies with learners' cognitive needs. The findings show that systematically applying CLT principles leads to substantial improvements and offers a replicable model for evidence-based pedagogical innovation.

The two-phase approach—diagnosing difficulties (Phase 1) and implementing targeted interventions (Phase 2)—proved effective, yielding a 50.8% improvement in programming competency with a large effect size (Cohen's d = 1.86). The results confirm that managing intrinsic load through worked examples and visualization, minimizing extraneous load through structured guidance, and enhancing germane load via reflective and gamified practice lead to measurable learning gains.These findings support broader application of CLT principles across STEM education domains where content complexity challenges student learning.

Overall, these findings reinforce Cognitive Load Theory (CLT) as a unifying framework for programming pedagogy that bridges theory and practice. The study presents a replicable model for educators and curriculum designers to reduce failure rates and cognitive overload through evidence-based strategies. The demonstrated success of this approach highlights how rigorous application of educational psychology principles can foster meaningful and scalable pedagogical innovation.

REFERENCES

[1] Anindyaputri, N. A., Yuana, R. A., &Hatta, P. (2020). Enhancing students' ability in learning process of programming language using adaptive learning systems: A literature review. Open Engineering, 10(1), 820-829.

[2] Bennedsen, J., &Caspersen, M. E. (2019). Failure rates in introductory programming — 12 years later. ACM Inroads, 10(2), 30–36. https://doi.org/10.1145/3324888

[3] Berssanette, J. H., & de Francisco, A. C. (2022). Cognitive load theory in the context of teaching and learning computer programming: A systematic literature review. IEEE Transactions on Education, 65(3), 440-449. https://doi.org/10.1109/TE.2021.3127215

[4] Butarev, K. (2024). Learning programming from the perspective of cognitive load theory. Информатика в Школе, 23(5), 5–11. https://doi.org/10.32517/2221-1993-2024-23-5-5-11

[5] Cheah, C. S. (2020). Factors Contributing to the Difficulties in Teaching and Learning of Computer Programming: A Literature Review. Contemporary Educational Technology, 12(2), ep272. https://doi.org/10.30935/cedtech/8247

[6] Duran, R. A., Zavgorodniaia, A., &Sorva, J. (2022). Cognitive Load Theory in Computing Education Research: A

Review. ACM Transactions on Computing Education, 22(4), 1–27. https://doi.org/10.1145/3483843

[7] Figueiredo, J., &García-Peñalvo, F. J. (2024). Design science research applied to difficulties of teaching and learning initial programming. Universal Access in the Information Society, 23(6), 1151–1161. https://doi.org/10.1007/s10209-022-00941-4

[8] Gkintoni, E., Antonopoulou, H., Sortwell, A., &Halkiopoulos, C. (2025). Challenging Cognitive Load Theory: The role of educational neuroscience and artificial intelligence in redefining learning efficacy. Brain Sciences, 15(2), 203. https://doi.org/10.3390/brainsci15020203

[9] Issever, D., Catalbas, M. C., & Duran, F. (2023). Examining Factors Influencing Cognitive Load of Computer Programmers. Brain Sciences, 13(8), 1132. https://doi.org/10.3390/brainsci13081132

[10] Li, X. (2016). Application of Cognitive Load Theory in programming teaching. Journal of Higher Education Theory and Practice, 16(6), 45-58

[11] Looker, N. (2021). A pedagogical framework for teaching computer programming: A social constructivist and Cognitive Load Theory approach. In Proceedings of the International Computing Education Research Workshop (pp. 415–416). ACM. https://doi.org/10.1145/3446871.3469778

[12] Margulieux, L. E., Morrison, B. B., & Decker, A. (2020). Reducing withdrawal and failure rates in introductory programming with subgoal-labeled worked examples. International Journal of STEM Education, 7(1), 19. https://doi.org/10.1186/s40594-020-00222-7

[13] Menon, P. (2023). Teaching tip: An example-based instructional method to develop students' problem-solving efficacy in an introductory programming course. Journal of Information Systems Education, 34(1), 1–15. https://aisel.aisnet.org/jise/vol34/iss1/1

[14] Ngadengon, Z., Subramaniam, T. S., Yasak, Z., Ideris, N. A., &Mohd, R. Z. (2025). Evaluating the difficulties in programming learning: Insights from polytechnic students. International Journal of Academic Research in Progressive Education and Development, 14(1). https://doi.org/10.6007/IJARPED/v14-i1/24518

[15] Salleh, S. M., Shukur, Z., & Judi, H. M. (2018). Scaffolding Model for Efficient Programming Learning Based on Cognitive Load Theory. International Journal of Pure and Applied Mathematics, 118(1), 77–82. https://ijpam.eu/contents/2018-118-1/7/7.pdf

[16] Sandoval-Medina, C., Arévalo-Mercado, C. A., Muñoz-Andrade, E. L., & Muñoz-Arteaga, J. (2024). Self-explanation effect of Cognitive Load Theory in teaching basic programming. Journal of Information Systems Education, 35(3), 303–312. https://doi.org/10.62273/GMIV1698

[17] Sibia, N., Liut, M., &Nobre, C. (2025). Exploring the role of visualization tools in enhancing computing education: A systematic literature review. Computer Graphics Forum, 43(3), 1–9. https://doi.org/10.2312/eved.20251027

[18] Sim, Y., & Lau, S. L. (2022). Review on challenges and solutions in novice programming education. In Proceedings of the 2022 IEEE International Conference on Computing (ICOCO) (pp. 55–61). IEEE. https://doi.org/10.1109/ICOCO56118.2022.10031657

[19] Sweller, J., Van Merriënboer, J. J. G., &Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. Educational Psychology Review, 31(2), 261–292. https://doi.org/10.1007/s10648-019-09465-5

[20] Weeda, R., Smetsers, S., &Barendsen, E. (2023). Unraveling novices' code composition difficulties. Computer Science Education, 34(3), 414–441. https://doi.org/10.1080/08993408.2023.2169067

[21] Winter, V., Friend, M., Matthews, M., Love, B., &Vasireddy, S. (2019). Using visualization to reduce the cognitive load of threshold concepts in computer programming. In Frontiers in Education Conference (pp. 1–9). IEEE. https://doi.org/10.1109/FIE43999.2019.9028612

[22] World Economic Forum. (2023). The Future of Jobs Report 2023. World Economic Forum. https://www.weforum.org/publications/the-future-of-jobs-report-2023/

★★★