European International Journal of Pedagogics (ISSN: 2751-OOOX)

Received: 28 July 2026 | Accepted: 26 August 2026 | Published: 24 September 2026

DOI: 10.55640/eijp-06-09-08

RESEARCH ARTICLE

Development of An Integrated Methodological Model for Teaching Programming Basics in A Digital Learning Environment

Ahtamkulov Muhriddin Ahtamkul ugli

Lecturer at the Department of Medical and Biological Physics, Zarmed University, Uzbekistan

VOLUME: Vol.06 Issue09 2026

PAGE: 50-55

Abstract

In this article, an integrated methodological model aimed at increasing the effectiveness of teaching programming basics to higher education students in a digital learning environment was developed and tested in practice. The research utilized an interventional experimental design involving Experimental (n=60) and Control (n=60) groups. The integrated methodological model includes interactive learning platforms, automated code assessment systems, and blended mentoring approaches. Mastery levels and statistical differences were analyzed using Student's t-test and Effect Size (Cohen's d) indicators. The results showed a significant increase in programming competence in the experimental group ($p < 0.001$, $d = 2.80$).

KEYWORDS

Digital learning environment, programming basics, integrated methodological model, automated assessment, hybrid learning, Student's t-test, pedagogical experiment.

INTRODUCTION

In modern information society, forming programming competencies has become a priority task in training specialists in engineering and technical fields [1]. However, classic methodology based on traditional lectures and practical classes fails to fully meet the needs of digital generation students [2]. When learning programming, students often exhibit low performance due to syntax errors, difficulties in logical thinking, and a lack of independent practical skills [3].

The development of digital learning environments (LMS, interactive platforms, and automated analysis systems) allows personalizing the educational process and enhancing teaching effectiveness through integrated methodological models [4, 5]. The aim of this study is to develop an integrated methodological model for teaching programming basics in a digital learning environment, implement it in practice, and prove its effectiveness using statistical methods.

METHODS

1. Research Design and Participants

The study was conducted during the 2025–2026 academic year among 1st-year students majoring in Mathematics and Information Technology at higher education institutions. A total of N = 120 students were divided into two equal groups: Experimental group ($n_1 = 60$) and Control group ($n_2 = 60$).

Table 1. Demographic and group-wise distribution of study participants

Group	Total Students (n)	Male (%)	Female (%)	Average Age	Initial Knowlge Level test score)
Experimental Group (EG)	60	63.3%	36.7%	18.6 ± 0.8	2.5± 8.2
Control Group (CG)	60	61.7%	38.3%	18.5 ± 0.7	43.1 ±7.9

(Note: ± (Plus-minus): Deviation around the mean and distribution range).

2. Pedagogical Mentoring Intervention

In the experimental group, programming basics (Python/C++) were taught based on the newly developed Integrated Methodological Model (IMM). In the control group, the traditional teaching methodology was maintained.

IMM consists of three main blocks:

1.Conceptual-visual block: Visualizing theoretical materials through interactive platforms and LMS [6].

2.Practical-automated block: An automated system (Auto-grader) that checks student code in real-time, along with Git version control [7].

3.Reflexive-mentoring (Blended Mentoring) block: Hybrid support from teachers and peer-to-peer (student-to-student), along with weekly code-phase analyses [8].

Table 2. Comparison between Integrated Methodological Model and Traditional Model

Component	Traditional Model (CG)	Integrated Methodological Model (EG)
Theory elivery	Passive lectures, textbooks	LMS video lectures and interactive simulations
Practical assignments	Submission on paper/file	Automated real-time testing system
Feedback	After 1–2 weeks	Within a few seconds (Auto-grader) + Mentor commentary
Pedagogical Approach	Teacher-centered	Student- and project-centered (PBL)

3. Evaluation Criteria

Students' programming competence was evaluated on a 100-

point scale across 4 main criteria [9, 10]:

Table 3. Evaluation criteria and weight indicators for programming competence

Criterion Code	Criterion Name	Responsible Direction	Maximum Score (Weight)
M1	Syntactic and semantic literacy	Code structure and adherence to rules	20 points
M2	Algorithmic thinking	Problem-solving logic and optimality	30 points
M3	Practical programming and debugging	Finding and fixing errors	30 points
M4	Code architecture and style	Comments, variable naming, and modularity	20 points

4. Statistical Analysis

SPSS (v.26) and Python SciPy library were used to analyze the data. Differences between groups were evaluated using Independent Samples t-test for independent samples:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

Cohen's d indicator was used to determine the effect size:

$$d = \frac{\bar{X}_1 - \bar{X}_2}{S_{pooled}}$$

Results were considered statistically significant at $p < 0.05$ [11].

RESULTS

1. Pre-intervention Results (Pre-test)

Before the study began, the initial knowledge and skill levels of all participants were determined through an entry test.

Table 4. Pre-intervention (Pre-test) results of Experimental and Control groups

Criteria	Experimental Group ($\bar{X}_1$ ±S1)	Control Group ($\bar{X}_2$ ±S2)	t-value	p-level
M1 (Syntax)	11.2 ± 2.4	11.5 ± 2.3	-0.69	0.489(n.s.)
M2 (Algorithms)	12.8 ± 3.1	13.0 ± 3.0	-0.35	0.723(n.s.

Criteria	Experimental Group ($X^{-}1$ ±S1)	Control Group ($X^{-}2$ ±S2)	t-value	p-level
				)
M3 (Debug/Practical)	10.5 ± 2.8	10.8 ± 2.9	-0.57	0.567 n.s.)
M4 (Style/Architecture)	8.0 ± 1.9	7.8 ± 2.0	0.56	0.576 n.s.)
Total Score (out of 100)	**42.5 ± 8.2**	**43.1 ± 7.9**	**-0.41**	**0.682 (n.s.)**

(Note: n.s. — statistically non-significant ($p > 0.05$). Initial groups had equal capabilities).

2. Post-intervention Results (Post-test)

After a 16-week pedagogical intervention, a final evaluation was conducted in both groups.

Table 5. Post-intervention (Post-test) results of Experimental and Control groups

Criteria	Experimental Group ($X^{-}1$ ±S1)	Control Group ($X^{-}2$ ±S2)	Growth (EG)	Growth (CG)
M1 (Syntax)	17.8 ± 1.8	14.2 ± 2.1	+58.9%	+23.4%
M2 (Algorithms)	25.4 ± 2.9	18.5 ± 3.4	+98.4%	+42.3%
M3 (Debug/Practical)	24.8 ± 3.2	16.9 ± 3.5	+136.1%	+56.4%
M4 (Style/Architecture)	16.5 ± 2.1	11.4 ± 2.4	+106.2%	+46.1%
Total Score (out of 100)	84.5 ± 7.8	61.0 ± 8.9	+98.8%	+41.5%

3. Statistical Comparison

Student's t-test was conducted to verify the statistical reliability of the difference between groups according to post-intervention indicators.

Table 6. Post-test statistical comparison indicators of Experimental and Control groups

Indicator	Experimental Group (n=60)	Control Group (n=60)	Difference (Δ)	t-statistic	Degrees of Freedom (df)	p-value	Cohen's d (Effect size)
Post-test Score	84.50	61.00	+23.50	15.35	118	< 0.001	2.80 (Very high)
Knowledge Growth (**Δ**)	+42.00	+17.90	+24.10	16.82	118	< 0.001	3.07 (Very high)

Statistical comparison yielded t = 15.35 and p < 0.001, confirming that the integrated methodological model is highly effective.

DISCUSSION

The study results indicate that the integrated methodological model for teaching in a digital learning environment significantly improves students' mastery of programming fundamentals. In particular, the highest growth rates were recorded in criteria M3 (Debug/Practical) and M2 (Algorithmic thinking).

Comparative Analysis of Obtained Results with International Scientific Literature

The high growth rate (+136.1%) observed in criterion M3 (Debug and practical assignments) fully confirms the research conducted by Hattie & Timperley as well as Becker et al. on immediate feedback theory [12, 14]. In the traditional teaching model, students waited several days for teacher feedback on syntactic or logical errors in their code. During this period, student motivation to understand and correct errors decreased [2]. The automated assessment (Auto-grader) module within IMM provided students with error types and correction instructions immediately upon submitting the assignment. As a result, mastery in practical criteria doubled.

Optimizing Cognitive Load (Cognitive Load Theory): Interactive platforms and dynamic visualization tools facilitated the understanding of theoretical concepts (specifically, pointers, memory resources, and complex data structures). This served to reduce excessive cognitive load in students' working memory [4, 15].

Hybrid Mentoring and Social Constructivism: In line with Vygotsky's [5] concept of the "Zone of Proximal Development", students succeeded in solving complex logical problems with the help of hybrid mentoring (peer-to-peer and teacher mentor) when unable to solve them independently using the automated system [8].

Development of Metacognitive Skills: Passing automated tests before submitting code fostered discipline in students regarding critical evaluation of their code, systematic error searching (debugging), and improving code style [13].

These results align with recent studies in scientific literature [12, 13]. The use of automated code grading (Auto-grading) systems allowed students to identify errors in real-time, fully satisfying the immediate feedback rule in pedagogical psychology [14].

However, some limitations were observed during practical implementation:

. Dependence on technical infrastructure: Stable internet and continuous operation of automated grading

servers are required [15].

· Initial adaptation period: A 1–2 week adjustment period was necessary for teachers and students to get used to working with the platform.

CONCLUSION

Within the framework of the study, an integrated methodological model for teaching programming basics in a digital learning environment was developed, and its effectiveness was proven in experimental testing.

Main Conclusions:

1. The integrated methodological model increased student achievement scores by 23.5 points (or 38.5%) compared to traditional teaching methods ($p < 0.001$).

2. The automated real-time assessment and hybrid mentoring system increased students' practical debugging skills by 136%.

REFERENCES

1. Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer Science Education, 13(2), 137-172.
2. Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 15 years later. ACM Inroads, 10(2), 30-36.
3. Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. ACM Transactions on Computing Education (TOCE), 18(1), 1-24.
4. Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. Morgan & Claypool Publishers.
5. Vygotsky, L. S. (1978). Mind in society: The development of higher psychological processes. Harvard University Press.
6. Crow, T., Luxton-Reilly, A., & Wu, W. (2022). Intelligent tutoring systems for programming: A systematic review. IEEE Transactions on Learning Technologies, 15(3), 345-358.
7. Ihantola, P., Ahoniemi, H., Karavirta, V., & Seppälä, O. (2010). Review of automated assessment tools for programming assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research, 86-93.
8. Siemens, G. (2005). Connectivism: A learning theory for the digital age. International Journal of Instructional Technology and Distance Learning, 2(1), 3-10.
9. Ala-Mutka, K. M. (2005). A survey of automated assessment of programming assignments. Journal of Information Technology Education: Research, 4(1), 219-236.
10. McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A multi-institutional study of introductory programming course performance. SIGCSE Bulletin, 33(4), 125-142.
11. Cohen, J. (1988). Statistical power analysis for the behavioral sciences (2nd ed.). Lawrence Erlbaum Associates.
12. Becker, B. A., Denny, P., Pettit, R., Prather, J., Rivers, K., & Smith, J. M. (2019). Compiler error messages in failure and success. ACM Transactions on Computing Education, 19(4), 1-28.
13. Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, metacognition, and self-regulation. Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, 5293-5308.
14. Hattie, J., & Timperley, H. (2007). The power of feedback. Review of Educational Research, 77(1), 81-112.
15. Margulieux, L. E., Catrambone, R., & Guzdial, M. (2016). Employing sub-goal labels to improve computing education. Computers & Education, 92, 38-51.