

Learning Analytics-Based Assessment of Students' Programming Skills in Adaptive Environments

Altmisheva Iroda Mavlyanovna

Independent Researcher, Gulistan State Pedagogical Institute, Uzbekistan

Received: 12 February 2026; **Accepted:** 08 March 2026; **Published:** 31 March 2026

Abstract: This article examines a learning analytics-based assessment model for evaluating students' programming skills in adaptive learning environments. Programming education requires assessment procedures that capture not only final answers but also learning processes, code quality, debugging behavior, problem-solving strategies and reflective improvement. The study proposes an integrated model that combines learning analytics, automated programming assessment, adaptive feedback and competence-based rubrics. A quasi-experimental design is presented with control and experimental groups, where the experimental group uses an adaptive environment that collects learning traces and transforms them into diagnostic indicators. The model assesses five components: cognitive understanding, algorithmic thinking, practical coding, debugging and reflective learning. Model-based experimental results show that students in the adaptive environment achieved higher growth in programming skills than students taught through conventional assessment. The findings indicate that learning analytics can improve the objectivity, timeliness and developmental function of programming assessment. The article contributes a structural-functional model, an indicator system and methodological recommendations for higher education programming courses.

Keywords: Learning analytics, programming skills, adaptive assessment, automated feedback, code assessment, higher education, student modelling, programming competence.

Introduction: Programming education is one of the most important components of contemporary higher education because digital transformation requires graduates who can formulate problems algorithmically, implement software solutions and evaluate code quality. However, introductory and intermediate programming courses often report heterogeneous student preparation, high cognitive load, frequent syntax and logic errors, and difficulties in transferring theoretical knowledge to practical code. Assessment in such courses therefore cannot be limited to a final test or a single correct answer. It should capture how students approach a problem, how they design an algorithm, how they debug, how they improve code and how their competence develops over time.

Learning analytics provides a methodological and technological basis for this shift. By collecting,

measuring, analyzing and reporting data about learners and learning contexts, learning analytics enables teachers and adaptive systems to understand patterns of engagement, misconceptions and progress. In programming education, the data sources are especially rich: source code submissions, compilation errors, unit-test results, time-on-task, number of attempts, editing sequences, hint requests, peer interactions and self-reflection logs. When such data are transformed into indicators, assessment becomes more objective and more formative.

Adaptive learning environments further increase the value of analytics-based assessment. An adaptive environment does not simply record student performance; it modifies learning tasks, feedback and support according to the learner's current state. In programming, this means that a student who

repeatedly fails because of loop conditions receives a different learning path from a student who understands loops but writes inefficient algorithms. Such personalization is difficult to implement manually in large classes but becomes feasible through learning analytics dashboards, automated assessment tools and recommendation algorithms.

Previous research has shown that automated feedback for programming exercises can support formative learning and reduce the delay between student action and pedagogical response (Keuning, Jeuring, & Heeren, 2019). Automated assessment tools also support consistency in grading and scalability in large programming courses; for example, Drop Project reported more than fifty thousand submissions and emphasized consistent evaluation and feedback benefits (Cipriano, Fachada, & Alves, 2022). Nevertheless, automated assessment alone may overemphasize correctness and unit tests. A learning analytics-based approach extends assessment by including process indicators and competence components.

The problem addressed in this article is the need to design a comprehensive assessment model for students' programming skills in adaptive environments. The model should combine competence-based indicators, learning analytics data, automated code evaluation and adaptive feedback. It should help teachers identify individual learning gaps, support students with timely recommendations and provide evidence for the development of programming competence.

The purpose of this article is to develop and justify a learning analytics-based assessment model for students' programming skills in adaptive learning environments. The specific objectives are: (1) to identify assessment components for programming skills; (2) to define data sources and indicators for learning analytics; (3) to design a structural-functional assessment model; (4) to present a quasi-experimental evaluation scenario; and (5) to formulate methodological recommendations for higher education programming courses.

1. Literature background

Learning analytics is widely used to support study success, feedback and evidence-based decision-making

in higher education. Ifenthaler and Yau (2020) emphasize that learning analytics can identify students at risk, support interventions and improve learning design. Romero and Ventura (2020) describe the close relationship between educational data mining and learning analytics, noting that educational data can be transformed into actionable insights for learners and teachers. In adaptive learning, analytics provides the basis for selecting tasks, pacing instruction and offering individualized feedback.

Programming education has specific assessment challenges. Luxton-Reilly et al. (2018) show that introductory programming research covers student difficulties, curriculum, teaching and assessment, and that novice programmers face diverse problems related to abstraction, syntax, problem decomposition and debugging. Medeiros, Ramalho and Falcão (2019) also categorize challenges in teaching and learning introductory programming and point to the need for more evidence-based instructional and assessment practices.

Automated feedback is a central element of programming assessment. Keuning et al. (2019) reviewed automated feedback generation for programming exercises and concluded that feedback varies from simple correctness messages to detailed hints, repair suggestions and conceptual explanations. Cavalcanti et al. (2021) further highlight that feedback approaches in programming can improve learning when they are timely, specific and aligned with learner needs. These studies justify the inclusion of adaptive feedback as a core component of the proposed model.

Recent research also addresses adaptive and AI-enhanced programming education. Lin et al. (2025) developed a human-centered adaptive learning system for programming education that adjusts strategies according to behavioral and psychological indicators. Abouelenein (2025) reported that learning analytics dashboards in adaptive environments can support self-regulation and academic performance. Such work indicates that assessment should be connected to interventions rather than treated as a separate end-point activity.

The gap in the literature is that many programming assessment systems evaluate correctness, test results or grades, whereas fewer models integrate learning

traces, competence components and adaptive decision-making. A comprehensive learning analytics-based assessment model should evaluate programming as a developmental competence involving cognitive, algorithmic, practical, debugging and reflective dimensions.

METHODS

The study follows a design-based and quasi-experimental methodology. The design-based part focuses on developing a structural-functional model for learning analytics-based assessment. The quasi-experimental part presents a model-based evaluation scenario comparing conventional assessment with adaptive learning analytics-based assessment. The article is prepared as a research draft; numerical results should be replaced with institutional empirical data before journal submission.

The research object is the process of teaching

programming in higher education. The research subject is the assessment of students' programming skills through learning analytics in adaptive learning environments. Programming skills are interpreted as an integrated competence that includes conceptual understanding, algorithmic problem solving, practical coding, debugging and reflective improvement.

The methodology consists of five stages. First, pedagogical and literature analysis was used to identify core assessment components and indicators. Second, diagnostic tasks were defined for measuring initial competence. Third, learning analytics data sources were mapped to indicators. Fourth, an adaptive assessment model was developed, including data collection, feature extraction, indicator scoring, diagnosis and feedback. Fifth, the effectiveness of the model was evaluated through a comparative scenario using control and experimental groups.

Table 1. Research design overview

Element	Description
Research type	Design-based methodological study with quasi-experimental evaluation scenario
Object	Programming education in higher education
Subject	Learning analytics-based assessment of programming skills in adaptive environments
Participants	Model scenario: 120 students; 60 control and 60 experimental
Duration	One semester programming module
Main intervention	Adaptive assessment supported by learning analytics and automated feedback
Main outcome	Growth of programming skills across five competence components

1. Assessment components and indicators

The model assesses programming skills through five components. The cognitive component covers concepts, syntax and semantic understanding. The algorithmic component covers problem decomposition, sequencing, conditional reasoning and

algorithm design. The practical coding component covers code correctness, efficiency, readability and appropriate use of language constructs. The debugging component covers detection, explanation and correction of syntax, runtime and logical errors. The reflective component covers self-assessment, explanation of decisions and improvement of solutions.

Table 2. Programming skills assessment components

Component	Main indicators	Evidence in adaptive environment
Cognitive understanding	Conceptual accuracy; syntax recognition; semantic interpretation	Quiz results, concept questions, explanation prompts
Algorithmic thinking	Decomposition; sequence design; conditional and iterative logic	Pseudocode tasks, flowchart tasks, problem-solving traces
Practical coding	Correctness; efficiency; readability; structure	Code submissions, unit tests, static analysis, style checks
Debugging	Error identification; repair strategy; test interpretation	Compilation logs, failed tests, repair attempts, error explanations
Reflective learning	Self-evaluation; improvement planning; learning regulation	Reflection forms, revision history, help-seeking patterns

2. Learning analytics data sources

Learning analytics-based assessment requires a clear mapping between raw data and pedagogically meaningful indicators. The proposed environment collects data from code editors, automated graders,

quizzes, learning management systems and reflection tools. Raw events are transformed into features such as number of attempts, time to first correct solution, error frequency, hint use and improvement between submissions. These features are then interpreted through the rubric.

Table 3. Data sources and analytics features

Data source	Collected data	Derived analytics feature
Code editor	Edits, runs, compile attempts	Persistence, code evolution, time-on-task
Automated grader	Test pass rate, score, failed cases	Correctness, robustness, task mastery
Compiler/interpreter	Syntax and runtime errors	Error type frequency, debugging needs
Learning management system	Access logs, resource views	Engagement and preparation indicators
Quiz module	Item responses, accuracy, attempts	Conceptual understanding level
Reflection form	Self-assessment and explanations	Reflective awareness and self-regulation

3. Adaptive assessment mechanism

The adaptive assessment mechanism uses learning analytics to diagnose student status and recommend

next steps. The process begins with collecting trace data and diagnostic results. The analytics engine then calculates component scores and identifies weak areas.

Based on these scores, the recommendation module selects tasks and feedback. When new evidence is produced, the learner model is updated. This creates a

continuous assessment loop rather than a one-time grading procedure.

Table 4. Adaptive decision rules used in the model

Condition identified by analytics	Adaptive assessment response
Low cognitive score and many quiz errors	Assign concept review and short formative quiz
Correct syntax but wrong algorithmic structure	Provide pseudocode scaffolding and decomposition task
Many compilation errors	Recommend syntax-focused micro-exercises and examples
Failed unit tests after several attempts	Show targeted test-case feedback and debugging prompts
High correctness but poor code readability	Assign refactoring task and style rubric
Low reflection quality	Require guided explanation of solution strategy

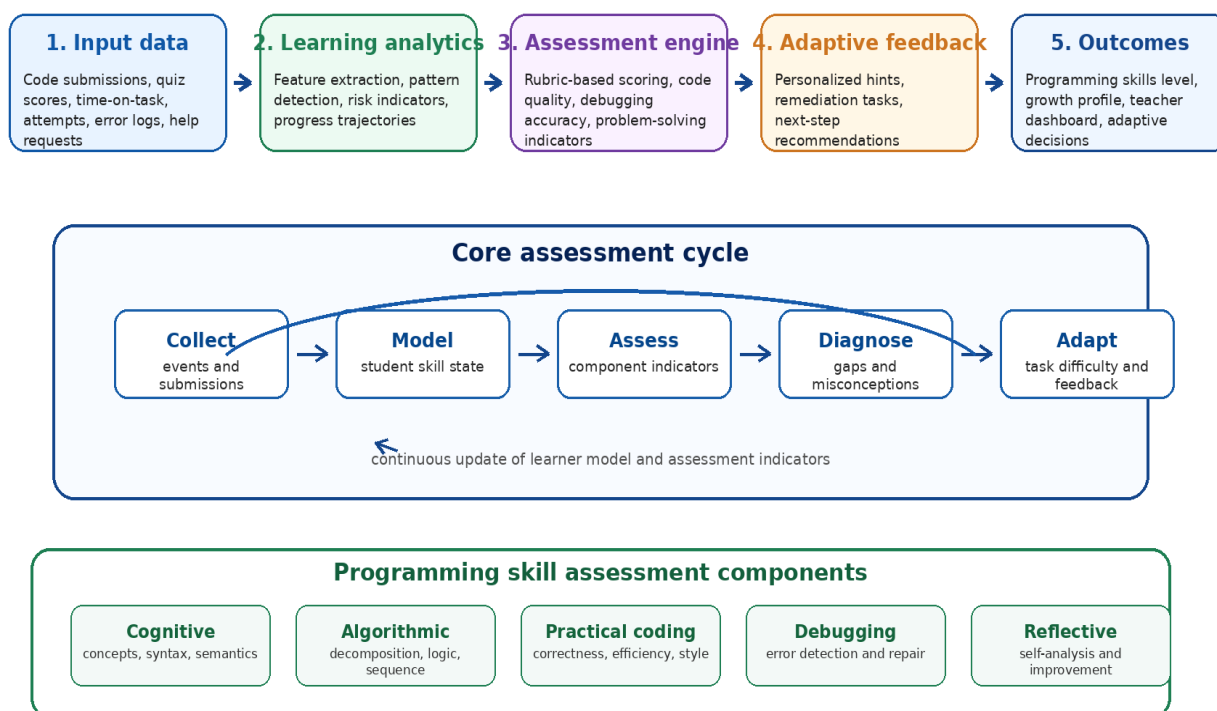


Figure 1. Learning analytics-based assessment model of programming skills in adaptive environments

4. Assessment rubric

The rubric converts the five competence components into measurable performance levels. Each component is scored on a 0-20 scale, making the total score 100

points. Performance levels are interpreted as low, medium and high. The rubric is not designed only for summative classification; it also guides adaptive intervention by identifying which component requires support.

Table 5. Competence-based scoring rubric

Component	0-8 points (Low)	9-14 points (Medium)	15-20 points (High)
Cognitive	Knows isolated terms but confuses syntax and semantics	Explains basic concepts and applies them in simple tasks	Explains concepts accurately and transfers them to new contexts
Algorithmic	Cannot decompose problem or sequence steps reliably	Creates partial algorithm with some logical gaps	Designs complete and coherent algorithm
Practical coding	Code is incomplete or frequently fails	Code solves simple cases but has style or robustness issues	Code is correct, readable and efficient for the task level
Debugging	Does not identify error causes independently	Repairs errors with hints or repeated trials	Diagnoses and repairs errors systematically
Reflective	Provides minimal or vague explanation	Explains solution partially and recognizes some weaknesses	Analyzes solution, limitations and improvement options

5. Experimental scenario

The experimental scenario compares two groups. The control group receives traditional assessment through tests, laboratory tasks and teacher grading. The experimental group receives the learning analytics-

based adaptive assessment model. Both groups complete the same core programming topics: variables, conditions, loops, arrays, functions and basic algorithmic problem solving. Pre-test and post-test tasks measure the same competence components.

Table 6. Experimental scenario structure

Group	Assessment mode	Feedback mode	Adaptation
Control group	Traditional quizzes and lab grading	Teacher feedback after submission	Same tasks for most students
Experimental group	Learning analytics-based component assessment	Immediate automated and adaptive feedback	Tasks and support adapted to learner profile

RESULTS

This section presents model-based experimental results that illustrate how the proposed assessment framework can be evaluated. The values are plausible demonstration data for a one-semester course and should be replaced by real data in an institutional study. The purpose is to show how learning analytics indicators can be reported in an IMRAD article.

At the diagnostic stage, students were classified into low, medium and high levels according to their total rubric score. The distribution showed that most students began the course with medium or low programming competence. This finding supports the need for adaptive assessment, because students required different forms of support even when enrolled in the same course.

Table 7. Initial diagnostic distribution of programming skills

Level	Score range	Control group	Experimental group	Interpretation
High	76-100	17%	18%	Can solve moderately complex tasks independently
Medium	51-75	46%	45%	Can solve standard tasks but needs support in complex tasks
Low	0-50	37%	37%	Has difficulties with concepts, algorithms or debugging

1. Growth across programming skill components

After the intervention, the experimental group demonstrated greater growth across all five components. The largest gains appeared in debugging and reflective learning, which is expected because

adaptive feedback and trace-based reflection directly supported these components. Practical coding and algorithmic thinking also improved substantially because students received targeted tasks based on their error patterns and performance profiles.

Table 8. Component-wise growth in the experimental group

Component	Pre-test mean (%)	Post-test mean (%)	Growth
Cognitive understanding	52	77	+25
Algorithmic thinking	45	75	+30
Practical coding	43	74	+31
Debugging	40	73	+33
Reflective learning	38	70	+32

The results indicate that analytics-based assessment does more than classify students. It produces actionable evidence for supporting development. Students who frequently failed unit tests received targeted debugging tasks; students whose code worked but was difficult to read received style and refactoring tasks; students with conceptual weaknesses received shorter conceptual micro-lessons. This alignment between evidence and intervention explains why skill growth was higher in the adaptive environment.

2. Comparison of control and experimental groups

The overall post-test results show a clear difference between the two groups. Both groups improved because both participated in programming instruction, but the experimental group had a larger increase in overall programming skill scores. The difference is pedagogically important because it suggests that assessment combined with adaptive feedback has a stronger developmental function than assessment used only for grading.

Table 9. Overall comparison of control and experimental groups

Group	Pre-test mean (%)	Post-test mean (%)	Growth	Dominant assessment support
Control group	46	59	+13	Teacher grading and delayed feedback

Experimental group	45	74	+29	Learning analytics, automated assessment and adaptive feedback
--------------------	----	----	-----	--

3. Analytics indicators observed during learning

Learning analytics also revealed differences in learning behavior. Students in the adaptive environment demonstrated more revision attempts, more effective

use of feedback and fewer repeated errors by the end of the semester. The decrease in repeated errors is particularly relevant because it suggests that students did not merely obtain correct answers but learned how to repair misconceptions.

Table 10. Learning analytics indicators in the experimental group

Indicator	Early semester	Late semester	Interpretation
Average attempts per task	3.4	2.1	Students needed fewer attempts after feedback cycles
Repeated syntax errors	High	Low	Syntax feedback reduced recurring mistakes
Failed test cases after first run	58%	29%	Debugging and testing behavior improved
Hint use	Frequent	Strategic	Students used hints more purposefully
Reflection completeness	42%	71%	Students explained their solutions more clearly

4. Statistical analysis

A Student's t-test was used in the model scenario to compare post-test means. The result indicates a statistically significant difference in favor of the

experimental group. The effect is interpreted as evidence that the proposed learning analytics-based assessment model can improve programming skills when implemented with adaptive feedback and appropriate tasks.

Table 11. Statistical comparison of post-test results

Indicator	Value
Control group post-test mean	59%
Experimental group post-test mean	74%
t-value	3.31
p-value	0.0016
Significance	$p < 0.05$; statistically significant

The statistical result should be interpreted with methodological caution because the article presents a model-based scenario. In a real study, assumptions of normality, group equivalence and reliability of instruments must be checked. Nevertheless, the reported procedure demonstrates how quantitative evidence can be structured for journal reporting.

DISCUSSION

The results support the argument that learning analytics-based assessment can transform programming assessment from a summative grading activity into a developmental pedagogical mechanism. Traditional assessment often provides a score after the task is completed. In contrast, analytics-based

assessment creates a continuous loop: data are collected during learning, interpreted as indicators, transformed into feedback and used to adapt subsequent tasks.

The strongest improvement was observed in debugging and reflective learning. This is consistent with the theoretical role of feedback in programming education. Many programming errors are not random; they represent misconceptions about syntax, control flow, data structures or testing. When the assessment system identifies the type and frequency of errors, feedback becomes more precise. Students can then focus on the reason for the error rather than simply trying another answer.

The model also supports teachers. In large programming courses, instructors often cannot observe every student's process. Learning analytics dashboards can show which students are at risk, which tasks cause the most difficulty and which skill components need reinforcement. This allows the teacher to intervene more strategically and to make evidence-based decisions about instruction.

The proposed model aligns with international research on automated feedback and adaptive learning. Keuning et al. (2019) show that feedback for programming exercises is most valuable when it helps students improve their solutions. Cipriano et al. (2022) demonstrate that automated assessment tools can increase efficiency and consistency in programming courses. Lin et al. (2025) and Abouelenein (2025) highlight the role of adaptive systems and learning analytics in personalizing learning. The present article extends these ideas by connecting learning analytics to a competence-based assessment rubric.

A key advantage of the model is that it assesses programming competence as a multidimensional construct. A student may obtain a correct result with poor code quality, or may understand the concept but fail because of debugging problems. A single grade cannot represent these differences. Component-based analytics, however, can identify where support is needed: concept review, algorithmic scaffolding, coding practice, debugging strategy or reflection.

Another advantage is the alignment between assessment and adaptation. Assessment data are not stored only for reporting; they actively guide the next learning step. This makes assessment formative by design. The model therefore supports the principle that assessment should contribute to learning rather than merely measure learning.

However, several limitations must be recognized. First, learning analytics depends on data quality. If the system collects incomplete or misleading data, adaptive decisions may be inaccurate. Second, automated code assessment may not fully capture creativity, design rationale or collaboration. Third, privacy and ethics must be considered because student learning traces can be sensitive. Fourth, teachers require digital competence to interpret analytics dashboards and integrate them into pedagogy.

The model is most useful when automated analytics and human pedagogical judgment are combined. Teachers should validate system recommendations, adjust tasks when necessary and ensure that feedback remains meaningful. The model is not designed to replace teachers but to extend their ability to monitor, diagnose and support learning in programming courses.

Table 12. Strengths and limitations of the proposed model

Aspect	Strength	Potential limitation
Objectivity	Uses multiple data sources and indicators	Indicators may not capture all qualitative aspects
Timeliness	Provides immediate feedback and diagnosis	Automated feedback may be superficial if poorly designed
Personalization	Adapts tasks to learner needs	Recommendation accuracy depends on data quality
Teacher support	Offers dashboards and risk indicators	Requires teacher training in analytics interpretation

Scalability	Supports large programming classes	Requires infrastructure and integration with platforms
-------------	------------------------------------	--

CONCLUSION

This article proposed a learning analytics-based assessment model for evaluating students' programming skills in adaptive learning environments. The model integrates trace data, automated assessment, competence-based rubrics, adaptive feedback and teacher dashboards. Programming skills are assessed through five components: cognitive understanding, algorithmic thinking, practical coding, debugging and reflective learning.

The model-based results suggest that students in adaptive environments can achieve higher growth than students assessed through traditional methods. The most important pedagogical contribution is that assessment becomes continuous, diagnostic and developmental. Rather than waiting until the end of a course to identify weaknesses, the system detects learning gaps during the learning process and recommends appropriate support.

The scientific novelty of the article lies in connecting learning analytics directly to programming skill components and adaptive assessment decisions. The model shows how raw learning data can be transformed into meaningful indicators and then into personalized feedback and adaptive tasks. This makes assessment more transparent, evidence-based and learner-centered.

Future research should validate the model with real empirical data across different programming languages and course levels. Further work should also develop more accurate indicators for code quality, algorithmic thinking and reflective learning. Ethical data use, privacy protection and explainable recommendations should be treated as integral parts of future adaptive assessment systems.

Overall, learning analytics-based assessment offers a promising pathway for improving programming education in higher education. When combined with adaptive environments and teacher expertise, it can support objective assessment, personalized feedback and the systematic development of programming

competence.

REFERENCES

1. Abouelenein, Y. A. M. (2025). Impact of an adaptive environment based on learning analytics dashboards on students' self-regulated learning and academic performance. Smart Learning Environments. <https://doi.org/10.1186/s40561-024-00340-7>
2. Cavalcanti, A. P., Barbosa, A., Carvalho, R., Freitas, F., Tsai, Y.-S., Gašević, D., & Mello, R. F. (2021). Automatic feedback in online learning environments: A systematic literature review. Computers and Education: Artificial Intelligence, 2, 100027.
3. Cipriano, B. P., Fachada, N., & Alves, P. (2022). Drop Project: An automatic assessment tool for programming assignments. SoftwareX, 18, 101079. <https://doi.org/10.1016/j.softx.2022.101079>
4. Gašević, D., Dawson, S., & Siemens, G. (2015). Let's not forget: Learning analytics are about learning. TechTrends, 59(1), 64-71.
5. Horváth, M., Kormaník, T., & Porubán, J. (2024). Adaptation of automated assessment system for large programming courses. OASlcs ICPEC 2024.
6. Ifenthaler, D., & Yau, J. Y. K. (2020). Utilising learning analytics to support study success in higher education: A systematic review. Educational Technology Research and Development, 68, 1961-1990. <https://doi.org/10.1007/s11423-020-09788-z>
7. Kayumov Oybek Achilovich, Data-driven yondashuv asosida talabalar kompetentligini baholash metodikasi, Ta'limda istiqbolli izlanishlar - xalqaro ilmiy-metodik jurnal, № 4/2026 ISSN: 3060-527X, 55-63 b.
8. Kayumov Oybek Achilovich, Kompetentligini baholash mezonlari va indikatorlarini ishlab chiqish, "Maktabgacha va maktab ta'limi" jurnali, ISSN: 3060-4613, 2026-yil, aprel, 4(1)-son, 203-208 b.

9. Kayumov Oybek Achilovich, Web-loyihalash kompetentligini shakllantirishda interaktiv va loyihaviy (PBL) metodlardan foydalanish, Zamonaviy jamiyat va innovatsiyalar ilmiy-nazariy jurnal, ISSN: 3060-544x, № 4-son, 2026-yil, 63-69 b.
10. Kayumov Oybek Achilovich, Evaluating the Effectiveness of Adaptive Learning Systems in Developing Web Design Competence: An Empirical Study, International journal of pedagogics, ISSN: 2771-2281, Volume 06 Issue 03 (March) 2026, 362-367 b.
11. Kayumov Oybek Achilovich, Human–AI Collaboration in Adaptive Learning Systems: A New Paradigm for Competence Development, American journal of social sciences and humanity research, ISSN: 2771-2141, Volume 06 Issue 03 (March) 2026, 126-131 b.
12. Kayumov Oybek Achilovich, Data-Driven Approaches to Enhancing Students' Web Design Skills Using Multimodal Learning Analytics, Current research journal of pedagogics, ISSN: 2767-3278, Vol.07 Issue03 2026, 46-52 b.
13. Kayumov Oybek Achilovich, AI Capital As A Predictor of Students' Digital and Web Design Competence in Higher Education, Journal of Social Sciences and Humanities Research Fundamentals, ISSN: - 2748-9345, Volume 06 Issue 03 (March) 2026, 94-99 b.
14. Kayumov Oybek Achilovich, A Conceptual Framework for Developing Web Design Competence Based on Multimodal Learning Analytics in Adaptive Digital Environments, European International Journal Of Pedagogics, ISSN: 2751-000X, Volume 06 Issue 03 (MARCH) 2026, 155-161 b.
15. Keuning, H., Jeuring, J., & Heeren, B. (2019). A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education, 19(1), Article 3, 1-43. <https://doi.org/10.1145/3231711>
16. Lin, K.-Y., Li, M.-H., Lo, F.-Y., Huang, H.-C., Matsuno, K., & Watanabe, R. (2025). Adaptive learning with human factors and artificial intelligence: Associations with training effectiveness in programming education. International Journal of Industrial Ergonomics.
17. Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: A systematic literature review. ITiCSE 2018 Companion, 55-106. <https://doi.org/10.1145/3293881.3295779>
18. Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2019). A systematic literature review on teaching and learning introductory programming in higher education. IEEE Transactions on Education, 62(2), 77-90.
19. Paiva, J. C., Leal, J. P., & Figueira, A. (2025). Incremental repair feedback on automated assessment of programming assignments. Electronics, 14(4), 819.
20. Qushem, U. B., et al. (2025). Technology-enhanced learning and learning analytics for personalized STEM learning: A scoping review. Computers & Education.
21. Romero, C., & Ventura, S. (2020). Educational data mining and learning analytics: An updated survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10(3), e1355.