Adaptive Student Assessment Method for Teaching Programming Course

Sadia Amin, Maida Shahid*, Talha Waheed, Muhammad Awais Hasan
Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan.
***Correspondence**: maida.shahid@uet.edu.pk

Citation | Amin. S, Shahid. M, Waheed. T, Hasan. M. A, "Adaptive Student Assessment Method for Teaching Programming Course", IJIST, Vol. 07, Issue. 03 pp 1659-1676, July 2025
Received | June 14, 2025 **Revised |** July 26, 2025 **Accepted |** July 27, 2025 **Published |** July 28, 2025.

Computer programming is a core component of computer science education and is widely recognized as a vital skill for aspiring professionals. Repetitive coding assessments help students improve their programming abilities, but the manual creation and evaluation of these assessments can be time-consuming and challenging for instructors. To address this, we developed an Adaptive Student Assessment System (ASAS) that automatically generates subjective programming questions aligned with Course Learning Objectives (CLOs) and assists in evaluating student responses. The system was evaluated using a controlled study involving two groups: a test group and a control group. Results demonstrated that the test group consistently outperformed the control group across cognitive assessments, with overall performance improvements of 13.5%. Affective feedback collected through a post-term survey showed a 48.20% higher agreement rate in the test group regarding motivation, clarity, and satisfaction with the assessment process. Teacher evaluations further confirmed the system's effectiveness, with improvements of 23.33% in assessment creation, 26.67% in assessment conduction, and 43.33% in result compilation compared to traditional methods. Teachers reported reduced workload, increased efficiency, and a positive attitude toward long-term adoption of the system. These findings highlight that ASAS not only enhances student engagement and academic performance but also improves instructional efficiency, making it a scalable and effective solution for programming education.

Keywords: Assessment Systems; Programming Fundamentals; Automatic Question Generation; Interactive Learning; Skill-based Learning

Introduction:

Humans are continuously evolving. From the pebble and stone tools of the Stone Age to the smartphones and computer systems of the current Iron Age, humans have improved their way of living. This is all because of scientific and technological development, and the field of Computer Science has vital importance in this evolution.

Computer Science is now considered a very important field in Education [1]. Recent studies have shown that the requirement for computer programmers has increased, due to which computer science degree growth has surged to 76% as compared to the previous growth of 16% in the last 12 years [2]. This degree offers many courses, out of which Programming Fundamentals (PF) has great significance in Computer Science degree programs [3]. Programming fundamentals help beginners understand how to code and how to get the desired output from given input data [4]. In a way, this course broadens students' minds by encouraging them to think out of the box [5][6]. Keeping a check on the minor details of the problems helps students in developing better solutions [4]. Therefore, it is very important for the students to grasp and master the PF course contents so that they can excel in the field of computer science [7].

Authorsuggests in [8] that if a student is not skilled in practical programming, they cannot master the significant concepts of the computer science field. To develop these skills in Computer Science students, a huge responsibility comes to the teachers to create the course contents, assessments, and exam questions that help students practice the theoretical knowledge learned [9]. Traditional assessment methods are most commonly used to create assignments manually. Then, the teachers evaluate all of the assignments one by one.

One of the drawbacks of this manual assessment system is that it takes quite a lot of time and effort from the teachers and requires considerable coordination [10]. As the manual methods can only generate limited programming tasks, the students tend to pass the assessments mostly through memorization and recitation techniques instead of learning the concepts taught in the class [11]. As a result, students' cognitive skills are confined to a box, and their focus shifts towards cramming, and their ability to explore and learn new things is restrained [12]. Hence, the grades do not reflect students' skills and abilities. Rather, they depend on how well a student can cram [13].

Because of excessive cramming habits, students are unable to map theoretical concepts to real-world problems and find their solutions. In order to resolve this challenge, the need for repetitive exercises arises [14]. These continuous activities performed in the classroom can be a bit hard to follow for an increasing number of computer science students [2]. Moreover, the repetition of the same course and the same assessments in different student sections can be a very tiresome and monotonous task for the teachers. Thus, the teachers find it difficult to conduct seamless exams for a growing number of computer science students manually.

The second challenge with the manual assessment method is the timely checking of the exams and provision of feedback to the students [15]. Continuous feedback is crucially important for students for their improvement and skill polishing. In addition, assessing all the students on the same criteria is not a good approach when everyone has their own learning pace [16].

As a solution to these challenges, automated programming assessment systems (APAS) [17] have recently replaced the traditional manual methods and have evolved to the most suitable forms to cater to teachers' needs and responsibilities. The idea of the automatic examination system roots back to the 1960s when the first system was developed by Hollingsworth [18] to identify the issues for assembly language programs. Since then, this system has been a topic of interest to educators.

APAS offers Automatic Question Generation, the most frequently used assessment method to generate students' assessments [19]. Automatic question generation is the most common form of grading that is preferred by teachers [20][21] for the accuracy of the results and the timely delivery of grades.

It has also been observed in recent studies that automatic questions enable the students to improve their skill set, thus they can perform better in exams [22]. Hence, the students can answer questions of any complexity level within the learned skills [23]. This type of question taxonomy is linked with the theory stated by Bloom Benjamin in 1956 [24]. This theory supports the argument that cognitive skills (in our case, Programming Skills) are one of the most important learning domains for human beings, which can be developed and improved via repetitive assessments. To include these programming skills in students, continuous assessments have to be conducted so that the students can master the skills. In order to ease the assessment process for teachers, an automatic examination system is required that could continuously generate unique questions in each iteration.

There are many different techniques used for automatic question generation. The rule-based question generation [25] focuses on multiple-choice analogy-based questions. The algorithm-based question generation [26] is used to generate algebra questions for school-level students, or any questions that fall under the if-else bracket. The template-based question generation [2] and schema-based question generation [27], both being the same variants of a problem, are also mainly focused on multiple-choice questions. However, there is work done on the generation of problems with controlled complexity. Moreover, these techniques are used to cover the syntactic analysis of already generated objective-wise questions.

Now, the question arises of how we can overcome the problem to automatically generate questions that are not only brief in their nature but also are comprehensive, specifically for computer science students. Moreover, the students are usually not willing to learn from the theoretical concepts taught in class. Instead, they give more attention to practical assessments [14]. The system, already available for question generation, does not do well when it comes to creating comprehensive/logical programming questions [28] that are also subjective. It is hard to map real-world problems into a set of questions based on the student's learning skills using these techniques. Hence, teachers often face issues with these techniques in grading computer science assessments.

Despite the advancements in Automated Programming Assessment Systems (APAS), existing solutions mostly rely on rule-based or template-based question generation, which limits their ability to produce complex, subjective programming questions [29][30][31][32][33]. These systems often fail to align with course learning objectives (CLOs) or adapt to varying student skill levels, making it difficult to effectively assess students' problem-solving and coding abilities. Additionally, current systems focus primarily on objective assessments and lack mechanisms to evaluate open-ended programming tasks or provide meaningful, skill-based feedback. This creates a significant research gap in developing a system that not only automates the creation of subjective programming questions but also evaluates student responses in a way that supports both personalized student learning and teacher efficiency. This study addresses this gap by proposing an adaptive assessment system that generates subjective questions mapped to CLOs and automatically evaluates students' answers to assist teachers in assessment and result compilation.

Literature Survey:

Student Learning Skills:

The author suggested [34] that student skills can be significantly improved with the help of written presentations. In this research, students were provided with 20 research publications. A rubric table was developed listing the criteria for student evaluation. Each rubric corresponded to a specific student learning level. Positive results were seen in a few rubric factors. Moreover, students were ranked between beginning, developing, competent, and accomplished levels. The issue with this approach was that the data provided to students was limited. Students had to extensively study the publications before submitting their answers. The study strongly supports that students' critical thinking can be improved if they are provided with real-life examples.

Author[35] came up with a meta-analysis process to keep up with the latest research in the general science field. The research was conducted on papers, dissertations, and theses from the past decade, and the results were then used in the classroom by teachers and educators. A statistically significant effect was observed in student learning. Nevertheless, this approach uses a comprehensive way of teaching and thus does not cover student assessment via automated systems, such as automatic questions and automated assessment methods.

Automatic Assessment Systems:

Authorhas pointed out [29] that learning Programming Fundamentals can be hard for fresh students. He proposed an AutoLEP, an automatic learning and assessment system, that helps teachers to check the programming assignment given to the students, and students are given feedback according to their test results, hence performing better next time. The AutoLEP system has 3 main components: student clients, a dynamic testing server, and the main server. The only lag in it is the unavailability of an automatic question generation system that would pick questions for the students according to their current learning level.

Author[30] suggested an eGrader. eGrader works both on dynamic and static analysis. The dynamic analysis process creates dynamic tests for different types of problems that have been proven effective, complete, and precise. The static analysis process consists of two parts: the structural similarity, which is based on the graph representation of the program, and the quality, which is measured by software metrics. The framework of eGrader consists of three components: Grading Session Generator, Source Code Grader, and Reports Generator. The limitation of this eGrader is that the Engineering Graphs can be tricky for non-technical teachers to comprehend and would require a long time to extract the results.

[31] caters to the problem of student assignment assessment based on the fact that the number of students keeps increasing with respect to the number of teachers. JavAssess is composed of four different modules, three of which can work in an isolated way. The three independent modules are classes that are located in the codeAssess; javAssessment package, and are called introspector, intercessor, and tester. The problem, again, with this solution is that it does not incorporate the automatic generation of student problem questions.

Author[32] has proposed a solution to identify the common problems made by novice programmers. There are 10 problems to be solved with increasing complexity. The code evaluations were divided into ALL_PASS, NOT_ALL_PASS, COMPILE_ERROR, RUNTIME_ERROR, and REPEAT, but only those were considered to have the label NOT_ALL_PASS. A manual examination of the code was also performed, and the code errors were divided into types: algorithmic, misinterpretation, and misconception. This solution involves the assessment of only 10 pre-defined problems and does not entertain the automatic question generation for teacher ease. The main focus is the identification of programming errors in the given set of programming problems.

Author[33] has focused mostly on the semantic analysis of the code APIs that are changed during code maintenance. FindBugs was developed using a bug-driven methodology. It includes the detection of Infinite recursive loops, Statements, or branches. The solution works great for the corporate sector, where APIs are mostly used for inter-domain communication. It is not very suitable for educational purposes and does not cater to the problem of automatic question generation as well.

Author[36] also suggested the semantic analysis of the code that is syntactically correct. He studied nearly 10 million static analysis errors found in over 500 thousand program submissions made by students over a five-semester period. The scope of this research does not include the automatically generated questions.

Author[37] worked on the identification of the logical errors made by the students. Logic Error Detection Algorithm: works on the comparison between the wrong code submitted by a student and the correct code, already available in the system. Comparison of structure patterns (a structured pattern can be considered to express the general form of source code, including a list of expressions and block statements). The error detection algorithm does not support automatic problem generation.

Our system will provide relief to the teachers by generating problem questions for programming assignments based on the students' learning skill sets. Our focus is to create an easy-to-use system that would not only generate questions automatically but also provide feedback that is easy to understand for non-technical educators as well.

Research Objectives:

To create a system that can automatically make programming questions for students with different skill levels.

To build a system that can check and grade students' programming answers using automatically generated questions.

To find out how automatic question generation can help teachers in making and managing exams more easily.

To develop a smart student profile system that helps students learn programming better by understanding their learning needs.

Material and Methods:

This research adopts a Template-Based Automatic Question Generation (AQG) approach, enhanced by AI-driven mechanisms in question personalization, code assessment, and adaptive student profiling. The methodology is organized into multiple stages, each leveraging AI concepts such as automation, adaptive feedback, and data-driven decision-making. Figure 1 explains the brief details of the proposed methodology.

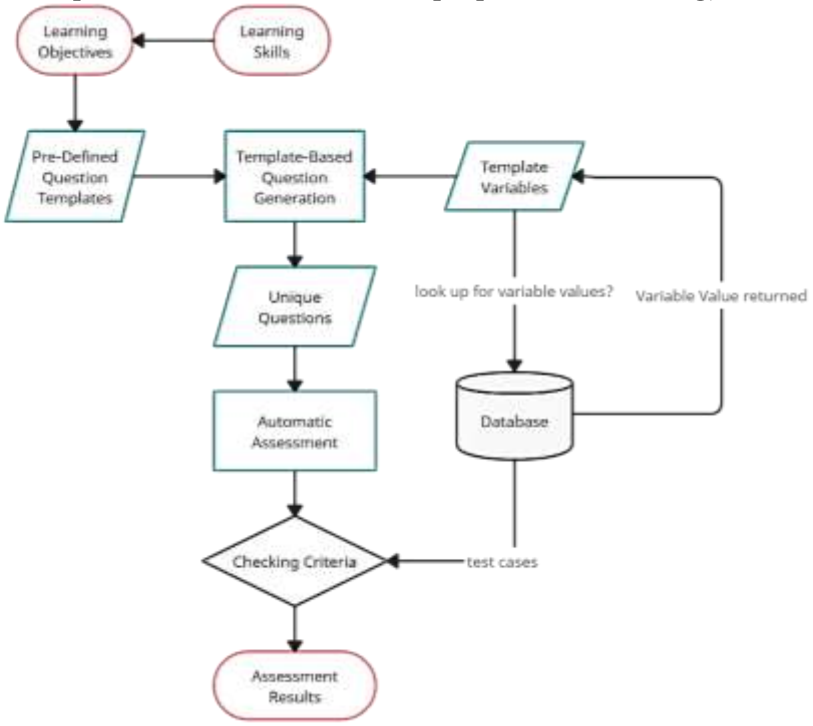

Figure 1. Proposed Methodology

Learning Objectives Extraction from Learning Skills:

The first step is choosing topics from a Programming Fundamentals course. A third party helps by providing a list of learning skills. Each skill is broken down into smaller parts called learning objectives. These objectives become the topics for our question generator.

For example, the Learning Skill of Conditional Statements states:

“Write a C++ program that can perform different tasks based on single or multiple conditions.”

To create templates, we carefully analyze each learning objective under this skill. One of the learning objectives extracted from the learning skill of conditional statements is stated below:

“Write a C++ program for a Conditional Statement with a Single Boolean Expression consisting of Any Comparison Operator.”

The goal of this learning objective is to help students understand how to use comparison operators within conditional statements. This classification enables the system to generate questions aligned with cognitive learning stages of the students, which is a principle drawn from AI in adaptive learning systems [38].

Template-Based Question Generation:

For each learning objective, generic question templates are created. These templates follow the same pattern and include variables (like $V1, $V2) that can be filled in with different values to make dynamic questions. For this purpose, many templates are devised, one of which is given below:

T01: “Write a program to check if **$V1** is **$V2,** then print True, where **$V3** is taken as input from the command line. Hint: **$V4**”

Each template is linked to a dedicated database table that stores lookup values for all variables. Table 1 shows the lookup table used for Template 01.

Table 1. Lookup values for template 01 variables

Sr#	V1	V2	V3	V4	Test Case
1	Polygon	Triangle	Sum of Interior Angles	Sum of angles = 180	[{input:180; inputType: int; output: true; outputType: bool; TestResult=true}, {input: range(0:179);inputType: int; output: false; outputType: bool; TestResult =false}, {input:range(181:999); inputType: int; output: false; outputType: bool; TestResult =false}]
2	Person	Senior Citizen	Age	Senior Citizen Age > 60	[{input:range(60:999); inputType: int; output: true; outputType: bool; TestResult=true}, {input: range(0:59); inputType: int; output: false; outputType: bool; TestResult =false}]
3	Letter	Vowel	Alphabet	[a, e, i, o, u]	[{input:['a','e','I','o','u']; inputType: list; output: true; outputType: bool; TestResult=true}, {input:['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']; inputType: list; output: false; outputType: bool; TestResult =false}]
4	Angle	Acute Angle	Angle	Acute angle < 90°	[{input:range(0:89); inputType: int; output: true; outputType: bool; TestResult=true}, {input: range(90:999); inputType: int; output: false; outputType: bool; TestResult =false}]

The number of rows inserted in the above table would be equivalent to the number of unique questions generated from a template. The variable values are carefully designed to create many different versions of the same question, match real-world examples, and help build test cases to check student answers automatically. This flow is explained in Figure 2.

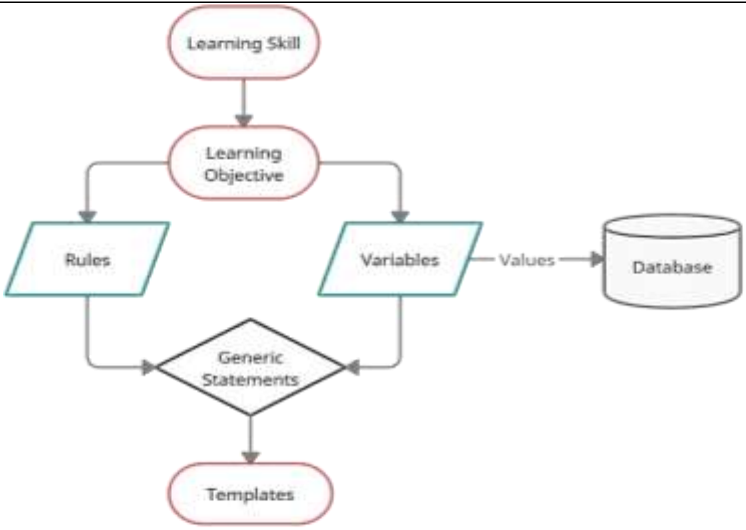

Figure 2. Question Generation via Templates

This method is based on a rule-based way of generating text, which is part of AI in the field of Natural Language Processing [39].

Question Generation Engine:

An algorithmic engine is implemented to generate unique questions from each template. It operates as follows:

Iterates through variable sets from the database.
Replaces template variables with corresponding values.
Logs each generated question with metadata (e.g., skill level, template ID).

This engine mimics the data-driven logic found in AI-powered intelligent tutoring systems (ITS) [40]. Generated questions from template T01:

Q1: Write a program to check if a Polygon is a Triangle, then print True, where the Sum of **the** Interior Angles is taken as input. Hint: Sum of angles = 180

Q2: Write a program to check if a Person is a Senior Citizen, then print True, where the Age is taken as input. Hint: Senior Citizen Age > 60

Q3: Write a program to check if a **Letter** is a **Vowel,** then print True, where the **Alphabet** is taken as input from the command line. Hint: **vowels: [a, e, i, o, u]**

Q4: Write a program to check if the **Angle** is an **Acute Angle,** then print True, where the **Angle** is taken as input from the command line. Hint: **Acute angle < 90°**

Each question is unique due to different combinations of variables, thus making the system adaptive for individual students' assessments.

Automated Code Assessment:

Each question has a corresponding test case defined in JSON format. The test cases simulate how the code should behave under various inputs. These cases are stored in the same database table linked to the template and are passed to the automatic assessment module. When a student submits code, the automated code assessment engine operates as follows:

The predefined test cases are automatically injected into a secure code execution environment.
The student's code is executed with these inputs, and the resulting output is captured.
This output is then compared against the expected output defined for each test case.
If the actual output matches the expected output exactly, the test case is marked as Pass.
If the outputs do not match, or if the code results in an error (e.g., runtime error, timeout, or incorrect logic), the test case is marked as Fail.

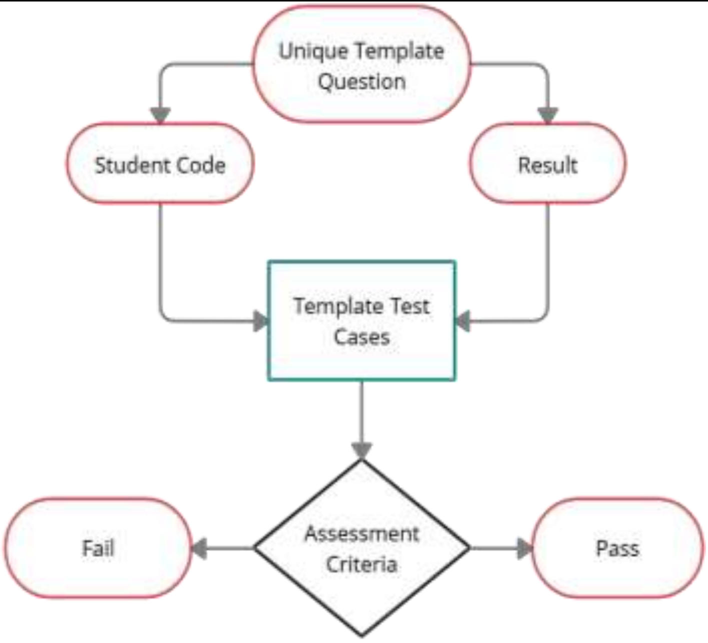

Figure 3. Automatic Assessment Method

This module, shown in Figure 3, mimics automatic grading systems (e.g., those used in coding platforms like HackerRank [41] by executing multiple conditions, evaluating edge cases, and providing real-time feedback.

Uniqueness Algorithm for Class Assessments:

To ensure fairness and avoid duplication, a randomized variable selection algorithm is applied. More than 50 variable sets per template are stored in the database, and the algorithm selects a unique set per student. This ensures that no two students receive the same question in assessments, personalization, and content diversity.

Experimentation:

Experiments were performed with respect to teachers' perspectives and students' perspectives.

Dataset Description:

Two groups of students were created to experiment with the effectiveness of our student assessment methods. All students were enrolled in the first year of their Computer Science degree program and were taking the Programming Fundamentals course, which spans 16 weeks. The students' ages ranged between 18 and 20 years.

Group A: 150 students from session 2020 1st semester session were selected for the control group. Out of 150 students, 83 were female and 67 were male. Traditional teaching methods were used for quizzes, midterms, and final exams for student assessment.

Group B: 150 students from session 2021 1st semester were selected for the test group. Out of 150 students, 78 were female and 72 were male. Our devised system was used for the assessment of this group. Quizzes, midterms, and final exams were generated using our Automated System.

Table 2 shows the division between the test and control group students.

Table 2. Student division for test and control groups

Group	Female Students	Male Students	Total Students	Session	Semester	Duration of Study (Weeks)
Test	78	72	150	2021	Fall	16
Control	83	67	150	2020	Fall	16

A group of teachers, containing 3 teachers, was selected who applied the traditional assessment methods to the Control Group. The same teacher group used our proposed automated student assessment method on the Test Group.

Teachers Evaluation:

The experimentation was classified into three categories. The same group of teachers first filled out this survey for the control group, where traditional methods were used for assessment creation, assessment conduction, and result compilation. Later, the same teachers filled out the same survey for the test group. The results were compared for both groups to reach a conclusion.

Assessment Creation: A survey was created with a total of 10 questions. The survey was divided into 3 sections with 4, 3, and 3-point sections for qualitative analysis, relevancy, and prerequisites, respectively, given in Appendix A.

Assessment Conduction: A survey was created with a total of 10 questions. The survey was divided into 3 sections with 4, 3, and 3-point sections for qualitative analysis, motivation, and prerequisites, respectively, given in Appendix B.

Result Compilation: A survey was created with a total of 10 questions. The survey was divided into 3 sections with 4, 4, and 2-point sections for qualitative analysis, approach, and prerequisites, respectively, given in Appendix C.

Student Evaluation:

A survey was conducted among the Control and Test group students to evaluate the effectiveness of our system based on students' improvement in their Cognitive and Affective Skills.

Cognitive Level: To evaluate the cognitive skills of the students, we used the quizzes, midterms, and final exams of both the Control and Test groups. A comparison was made to observe any improvement between the two groups.

In a 4-month semester duration, one quiz was taken after 4 weeks, both for the Control and Test groups. Midterm exams were carried out after 2 months. The second quiz was taken at the end of 3 months. And lastly, final term exams were conducted at the end of the fourth month. Therefore, we had a total of two quizzes, one midterm, and one final exam, both for the Control and Test groups. Also, 2 term projects were part of the experimentation; one after the midterm, and one after the final exams. Mark's division is given in Table 3.

Table 3. Marks distribution for student cognitive level experimentation

Assessment Type	Marks
Quiz-I	10
Mid-term	20
Project-I	15
Quiz-II	10
Finals	30
Project-II	15
Total Marks	100

Affective Level: To evaluate the affective skills of the students, a survey was conducted at the end of the term for both the Control and Test group students. Students were asked how the applied assessment method helped them polish their skills and how they were able to perform in their quizzes, midterm, final exams, and term projects. Survey division is 4, 3, and 3 points for student performance, assessment approach, and results, respectively, given in Appendix D.

Result and Discussion:

Teachers Evaluation:

The graphs below give brief details about the survey results for teachers' evaluation. The horizontal axis represents the number of teachers who took part in our research. The vertical axis shows the survey question number. The green colours show the number of teachers who agreed with the question asked in the survey. The yellow colour represents the neutral answers, and the red colour shows the disagreement with the questions asked.

Assessment Creation:

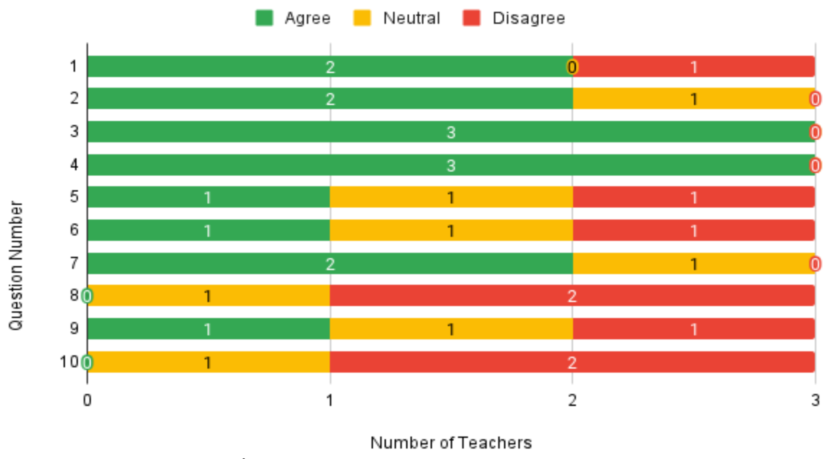

Figure 4. Teachers' evaluation for assessment creation - test group

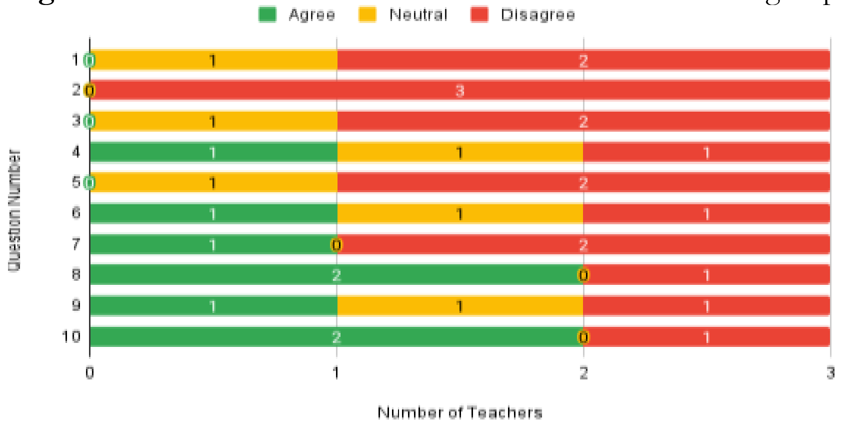

Figure 5. Teachers' evaluation for assessment creation - control group

Table 4 below summarizes the survey responses from teachers for the Assessment Creation section, divided into three categories: Qualitative Analysis, Relevancy, and Prerequisites. Three teachers answered each question in the survey, and responses were categorized as Agree, Neutral, or Disagree. The Total Responses for each section are calculated as:

$$Total\ \text{Responses} = \text{Number of Questions} * \text{Number of Teachers}$$

The Total Agree value for each section represents the sum of all Agree responses across the questions within that section. To calculate the Agreement Percentage, this formula is used:

$$\text{Agreement \%} = \left(\frac{Total\ Agree\ Responses}{Total\ Responses}\right) * 100$$

The overall improvement between the test and control groups is then calculated as follows:

$$\text{Improvement \%} = \text{Test Group Agreement \%} - \text{Control Group Agreement \%}$$

This highlights the positive or negative change brought by the proposed system.

Table 4. Summary of the survey responses from teachers on Assessment Creation

Section	Questions	Total Responses	Test Group		Control Group		% Improvement
			Total Agree	Agree%	Total Agree	Agree %	
Qualitative	Q1-Q4	12	10	83.33%	1	8.33%	+75.0%

Relevancy	Q5-Q7	9	4	44.44%	2	22.22%	+22.2%
Prerequisites	Q8-Q10	9	1	11.11%	5	55.56%	-44.4%
Overall	Q1-Q10	30	15	50%	8	26.67%	+23.33%

Assessment Conduction:

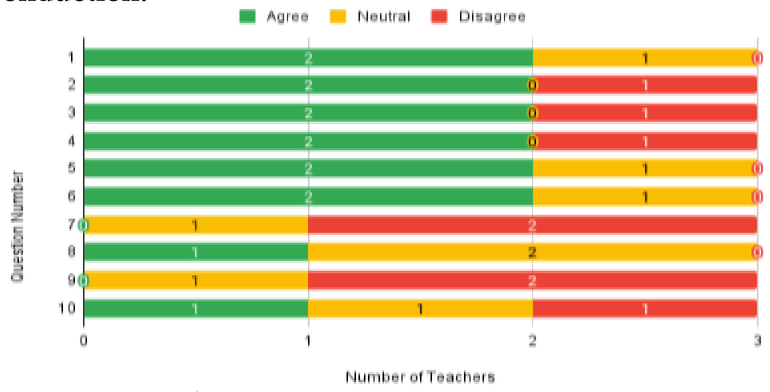

Figure 6. Teachers' evaluation for assessment conduction - test group

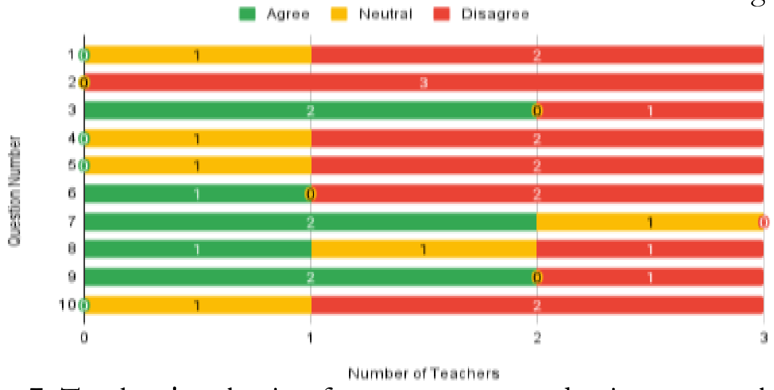

Figure 7. Teachers' evaluation for assessment conduction - control group

Table 5. Summary of the survey responses from teachers on Assessment Conducting

Section	Questions	Total Responses	Test Group		Control Group		% Improvement
			Total Agree	Agree%	Total Agree	Agree %	
Qualitative	Q1-Q4	12	8	66.67%	2	16.67%	+50.0%
Motivation	Q5-Q7	9	6	66.67%	1	11.11%	+55.56%
Prerequisites	Q8-Q10	9	1	11.11%	4	44.44%	-33.33%
Overall	Q1-Q10	30	15	50%	7	23.33%	+26.67%

Result Compilation:

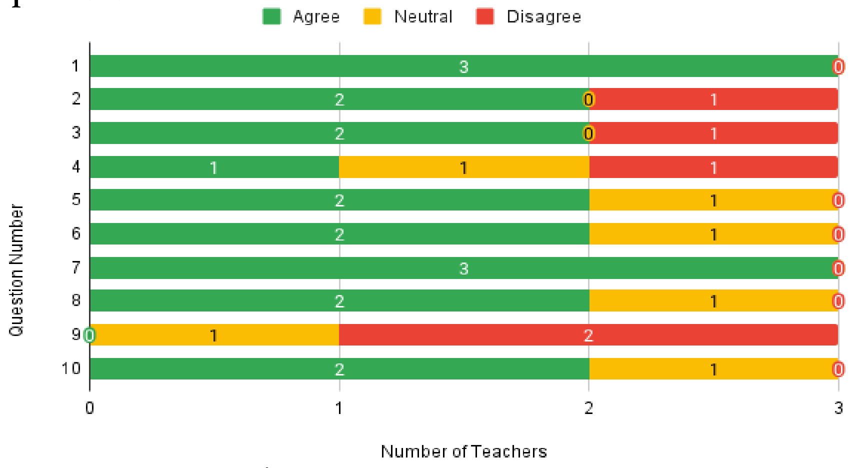

Figure 8. Teachers' evaluation for result compilation - test group

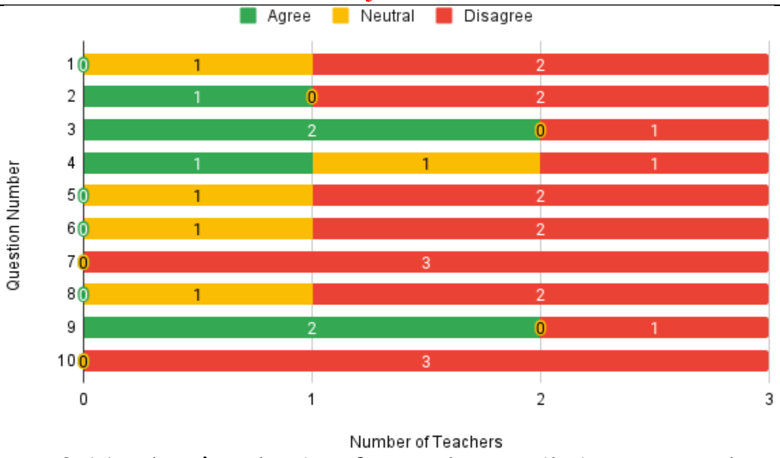

Figure 9. Teachers' evaluation for result compilation - control group

Table 6. Summary of the survey responses from teachers on Result Compilation

Section	Questions	Total Responses	Test Group		Control Group		% Improvement
			Total Agree	Agree%	Total Agree	Agree %	
Qualitative	Q1-Q4	12	8	66.67%	4	33.33%	+33.34%
Approach	Q5-Q8	12	9	75.0%	2	16.67%	+58.33%
Prerequisites	Q9-Q10	6	2	33.33%	0	0.0%	+33.33%
Overall	Q1-Q10	30	19	63.33%	6	20.00%	+43.33%

Student Evaluation:

Cognitive Level:

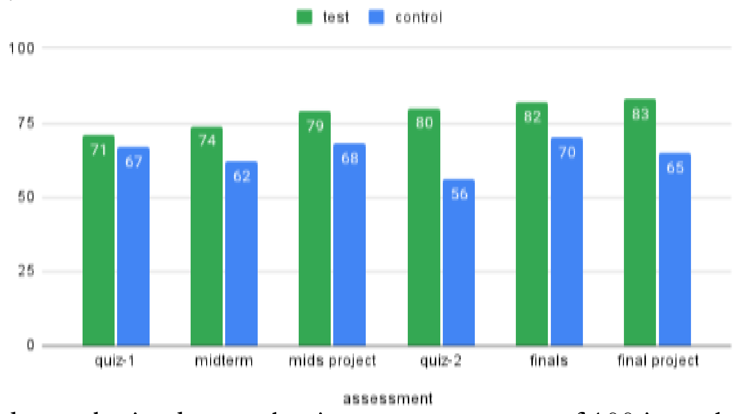

Figure 10. Students obtained an evaluation percentage out of 100 in each assessment for their cognitive level

Affective Level:

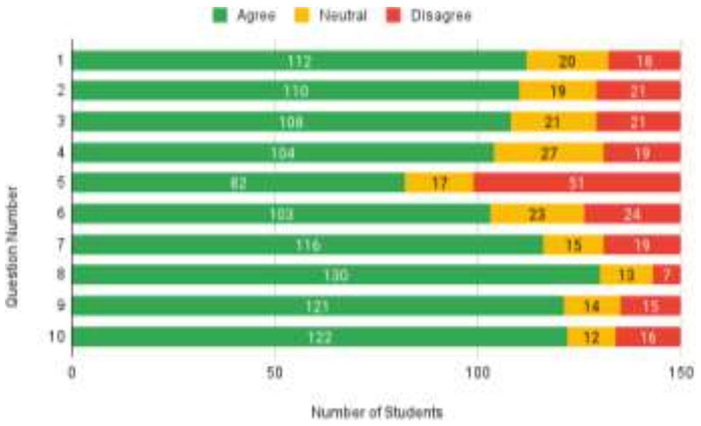

Figure 11. Students' evaluation for their affective level - test group

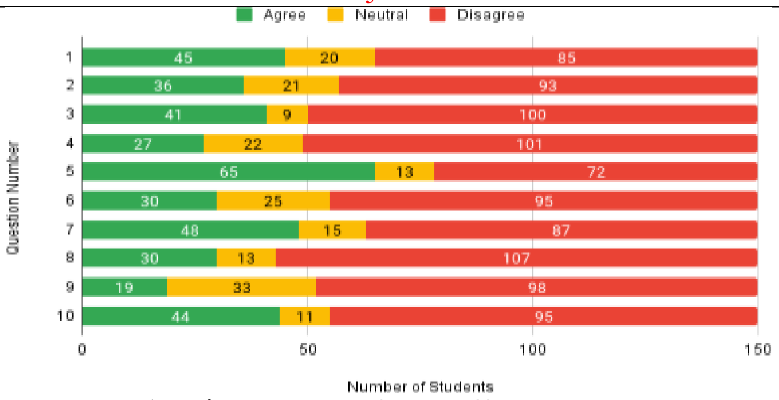

Figure 12. Students' evaluation of their affective level - control group

Table 7. Summary of the survey responses from students on the Affective Level

Section	Questions	Total Responses	Test Group		Control Group		% Improvement
			Total Agree	Agree%	Total Agree	Agree %	
Student Performance	Q1-Q4	600	434	72.33%	149	24.83%	+47.50%
Assessment Approach	Q5-Q7	450	301	66.89%	143	31.78%	+35.11%
Results	Q8-Q10	450	373	82.89%	93	20.67%	+62.22%
Overall	Q1-Q10	1500	1108	73.87%	385	25.67%	+48.20%

Discussion:

Teachers' Evaluation: The teachers' evaluation results strongly support the effectiveness of the proposed automated assessment system across all stages: assessment creation, conduction, and result compilation. Overall, the test group consistently reported higher agreement rates compared to the control group, confirming a positive shift in teacher satisfaction, engagement, and process efficiency.

In the Assessment Creation survey, teachers in the test group showed a substantial increase in satisfaction, as shown in Table 5. The qualitative analysis section saw the highest gain, with 83.33% agreement in the test group versus 8.33% in the control group, which is a 75% improvement. This suggests that the teachers appreciated the quality, uniqueness, and structure of the automatically generated questions. In the relevancy section, agreement improved by 22.2%, indicating better alignment of generated questions with course learning objectives. However, in the prerequisite section, a drop of 44.4% in agreement was observed, highlighting concerns around the need for training and resource readiness before adopting the system. Despite this, the overall agreement rate improved by 23.33%, affirming the positive reception of the system in facilitating and enhancing the assessment creation process.

In the Assessment Conduction stage, the results were similarly encouraging. As shown in Table 6, agreement on qualitative aspects rose by 50%, reflecting a smoother and more streamlined execution of assessments. Teachers in the test group reported that less time, space, and fewer invigilators were required, and that students asked fewer clarifying questions during the assessment. The motivation section showed a notable 55.56% improvement, with teachers expressing increased willingness to conduct assessments more frequently using the proposed system. However, the prerequisite section showed a 33.33% drop, again reflecting the system's dependency on adequate training and infrastructure. Even with this limitation, the overall agreement improved by 26.67%, reinforcing the system's benefits in operational efficiency and teacher confidence during assessment conduction.

In the final stage, Result Compilation, the test group again outperformed the control group across all sections, as shown in Table 7. The approach section saw the most significant rise, with agreement increasing from 16.67% to 75%, which is a 58.33% improvement. This indicates that teachers found the system effective in automating and simplifying result processing. The qualitative section saw a 33.34% increase, suggesting improved clarity and reliability of the results. Even the prerequisites section showed a positive gain of 33.33%, although it still highlights the need for further training and system familiarization. In total, the overall agreement rate increased by 43.33%, confirming enhanced satisfaction in result compilation, reduced manual workload, and quicker access to performance insights.

Overall, teachers in the test group reported that the system was easy to follow, reduced paperwork, and showed potential for long-term use. They appreciated the user-friendly interface and the time saved in compiling results and conducting assessments. However, some concerns were raised about the need for prior training and equipment dependency, especially for question generation and during initial setup phases. In contrast, the control group highlighted significant issues in terms of time consumption, lack of motivation, and difficulty in managing traditional assessment processes, with most teachers expressing disinterest in continuing with the manual approach.

Student Evaluation:

The evaluation of students comprised both cognitive and affective domains to measure the overall impact of the proposed assessment system. Cognitive outcomes were assessed through academic performance in various evaluation components, while affective outcomes were captured through a structured end-of-term survey measuring students' attitudes and perceptions.

As shown in Figure 7, cognitive performance consistently favored the test group across all evaluation stages. Students in the test group outperformed the control group in Quiz 1 (71% vs. 67%), Midterms (74% vs. 62%), Midterm Project (79% vs. 68%), Quiz 2 (80% vs. 56%), Final Exams (82% vs. 70%), and Final Project (83% vs. 65%). The overall improvement of 13.5% in students' cognitive performance reflects increased content understanding, problem-solving ability, and skill application, likely influenced by the structured and transparent approach of the proposed assessment methodology.

On the affective side, as detailed in Table 6, the test group also reported significantly higher agreement rates across all surveyed dimensions. The Student Performance section saw a 47.5% improvement, suggesting enhanced confidence and engagement with assessments. Agreement on the Assessment Approach improved by 35.11%, indicating a better perception of fairness, clarity, and ease of assessment. In the Results section, agreement increased by 62.22%, reflecting students' satisfaction with the timeliness, accuracy, and transparency of result reporting. Overall, the test group achieved a 73.87% agreement rate compared to 25.67% in the control group, which is an overall improvement of 48.20%.

Despite these encouraging outcomes, a few students in the test group expressed initial difficulty in understanding the new assessment conduction process, which suggests a need for better onboarding and communication in the early stages.

In conclusion, while the proposed system requires some initial investment in training and technical setup, the substantial gains in efficiency, engagement, and scalability make it a strong candidate for broader adoption. The trade-off between initial learning and long-term benefit appears favorable. Notably, concerns about student unfamiliarity and infrastructure can be mitigated over time with regular usage and institutional support, ensuring smooth integration into existing educational workflows.

Comparison with Existing Studies:

When compared to existing systems discussed in the literature, such as AutoLEP[29] and eGrader [30], our system offers significant enhancements. While AutoLEP provided useful feedback on assignments, it could not generate adaptive programming questions based on student skill levels. Our system fills this gap by integrating both automatic question generation and adaptive difficulty scaling. Similarly, eGrader focused on code evaluation using dynamic and static analysis, but its complexity made it less approachable for non-technical educators. Our system, in contrast, was praised by teachers for its ease of use and user-friendly feedback mechanisms.

Moreover, systems like JavAssess [31] and FindBugs [33] do not support automated question generation or adaptive assessment. In comparison, our approach combines question generation and assessment, offering a more comprehensive educational solution tailored for classroom environments. Furthermore, Noblitt's study relied heavily on manual rubric evaluations; our system provides automated and adaptive assessment, making it more scalable for large student cohorts. In contrast to Ettles [32], which identified typical student programming errors using predefined problems, our system generates new and unique questions for each student based on their learning trajectory. This not only supports skill development but also discourages rote memorization.

Conclusion:

The growing complexity and scale of computer science education have made traditional assessment methods increasingly inefficient and difficult to manage. Manual creation, conduction, and result compilation of assessments pose significant challenges for educators, particularly in handling subjective programming questions at scale. To address these limitations, this study introduced an automated assessment system designed to generate programming questions mapped to student learning outcomes, conduct assessments efficiently, and compile results with minimal manual intervention.

Comprehensive experiments conducted with test and control groups demonstrated the effectiveness of the proposed system across all stages of assessment. Teacher evaluations showed significant gains in satisfaction, efficiency, and engagement. The test group consistently outperformed the control group, with overall improvements of 23.33% in assessment creation, 26.67% in assessment conduction, and 43.33% in result compilation. While there were some concerns about the need for prior training and infrastructure readiness, the overall feedback from teachers supported the adoption of the proposed system for long-term use.

On the student side, both cognitive and affective improvements were evident. Students in the test group showed a consistently better performance across quizzes, midterms, finals, and projects. The overall improvement of 13.5% reflects better understanding, confidence, and application of programming skills. Affective survey responses further confirmed increased motivation, satisfaction with the assessment process, and appreciation for the clarity and fairness of the system. An overall agreement rate improvement of 48.20% in the affective survey highlights the students' positive reception.

Although initial hurdles such as system unfamiliarity and dependency on equipment were reported, these are transitional challenges that can be addressed through structured training and gradual integration. The trade-off between initial setup effort and long-term benefits is favorable.

In conclusion, the proposed automated assessment system significantly enhances both teaching and learning experiences in computer science education. It not only alleviates the manual burden on educators but also improves student outcomes by providing a more streamlined, transparent, and scalable approach to assessments. With continued refinement and institutional support, this system has the potential to transform traditional educational practices and promote a more efficient, learner-centered academic environment.

Acknowledgement: The Authors would like to acknowledge the staff and students of the University of Engineering and Technology, Lahore, for their cooperation during the conduction of experimentation.

Author's Contribution: Author 1 was responsible for the initial write-up of the paper, the design of examination materials, as well as the review and final formatting of the manuscript. Author 2 contributed by defining and implementing the templates, designing the research methodology and experimentation, and participating in the review and formatting process. Author 3 supported the review of the paper, refinement of the argumentation and logical structure, re-articulation of the text, and formatting.

Conflict of interest: There exists no conflict of interest for publishing this manuscript in IJIST.

Project details: This research was not conducted as a result of any project.

References:

[1] D. P. C. & M. M. S. Mary Webb, Niki Davis, Tim Bell, Yaacov J. Katz, Nicholas Reynolds, "Computer science in K-12 school curricula of the 2lst century: Why, what and when?," *Educ. Inf. Technol.*, vol. 22, pp. 445–468, 2016, doi: https://doi.org/10.1007/s10639-016-9493-x.

[2] Z. O. G. of S. E. in T. P. Radošević, D., Orehovački, T., Stapić, "Automatic On-Line Generation of Student's Exercises in Teaching Programming," *Cent. Eur. Conf. Inf. Intell. Syst.*, pp. 22–24, 2010, [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2505722

[3] M. Karakus, S. Uludag, E. Guler, S. W. Turner, and A. Ugur, "Teaching computing and programming fundamentals via App Inventor for Android," *2012 Int. Conf. Inf. Technol. Based High. Educ. Training, ITHET 2012*, 2012, doi: 10.1109/ITHET.2012.6246020.

[4] P. Tuomi, J. Multisilta, P. Saarikoski, and J. Suominen, "Coding skills as a success factor for a society," *Educ. Inf. Technol.*, vol. 23, no. 1, pp. 419–434, Jan. 2018, doi: 10.1007/S10639-017-9611-4/METRICS.

[5] M. G. S. Prof. Dr/ Eid Abdel Wahid Ali, Hanan Mohammad Maher, "Thinking out of the box: Educational Applications," *Clin. J. Am. Soc. Nephrol.*, vol. 37, no. 1, 2022, [Online]. Available: https://mathj.journals.ekb.eg/article_216941_10a3b2592ffe6be3ea05528438ac2b5f.pdf

[6] Marlborough, "Why is Problem Solving Important in Child Development?," 2020, [Online]. Available: https://www.marlborough.org/news/~board/health-and-wellness/post/why-is-problem-solving-important-in-child-development

[7] Raad A. ALTURKI, "Measuring and Improving Student Performance in an Introductory Programming Course," *Informatics Educ.*, vol. 15, no. 2, pp. 183–204, 2016, doi: https://doi.org/10.15388/infedu.2016.10.

[8] S. Fincher, "What are we doing when we teach programming?," *Proc. - Front. Educ. Conf.*, vol. 1, 1999, doi: 10.1109/FIE.1999.839268.

[9] H. U. Sezer Kanbul, "Importance of Coding Education and Robotic Applications For Achieving 21st-Century Skills in North Cyprus," *Int. J. Emerg. Technol. Learn.*, vol. 12, no. 1, 2017, [Online]. Available: https://online-journals.org/index.php/i-jet/article/view/6097

[10] Morningside College, "Advantages and Disadvantages of Various Assessment Methods", [Online]. Available: https://www.clark.edu/tlc/outcome_assessment/documents/AssessMethods.pdf

[11] "What are traditional methods of teaching? | ResearchGate." Accessed: Aug. 06, 2025. [Online]. Available: https://www.researchgate.net/post/What-are-traditional-methods-of-teaching

[12] C. Ahn *et al.*, "Stanford Memory Trends," 2018. Accessed: Aug. 06, 2025. [Online]. Available: https://purl.stanford.edu/nj167fx1481

[13] A. Sutherland, "Grades don't correlate with a student's intelligence," *State Press*, 2017, [Online]. Available: https://www.statepress.com/article/2017/04/spopinion-grades-do-not-correlate-with-a-students-intelligence

[14] H. Hagger, K. Burn, T. Mutton, and S. Brindley, "Practice makes perfect? Learning to learn as a teacher," *Oxford Rev. Educ.*, vol. 34, no. 2, pp. 159–178, Apr. 2008, doi: 10.1080/03054980701614978.
[15] K. Koh and A. Luke, "Authentic and conventional assessment in Singapore schools: an empirical study of teacher assignments and student work," *Assess. Educ. Princ. Policy Pract.*, vol. 16, no. 3, pp. 291–318, 2009, doi: 10.1080/09695940903319703;WGROUP:STRING:PUBLICATION.
[16] National University of Learning Disabilities, "Transforming Assessments and Learning from the Ground up." Accessed: Aug. 06, 2025. [Online]. Available: https://www.education-first.com/wp-content/uploads/2019/12/Education-First-State-of-Flux-Part-Two-Assessment-and-Accountability-Landscape-Scan-July-2019-2.pdf
[17] I. Mekterović, L. Brkić, B. Milašinović and M. Baranović, "Building a Comprehensive Automated Programming Assessment System," *IEEE Access*, vol. 8, pp. 81154–81172, 2020, doi: 10.1109/ACCESS.2020.2990980.
[18] J. Hollingsworth, "Automatic graders for programming classes," *Commun. ACM*, vol. 3, no. 10, pp. 528–529, 1960, doi: https://doi.org/10.1145/367415.367422.
[19] Atiq Ur Rehman, "MOST FREQUENT TEACHING STYLES AND STUDENTS' LEARNING STRATEGIES IN PUBLIC HIGH SCHOOLS OF LAHORE, PAKISTAN," *Academia*, 2016, [Online]. Available: https://www.academia.edu/26149922/MOST_FREQUENT_TEACHING_STYLES_AND_STUDENTS_LEARNING_STRATEGIES_IN_PUBLIC_HIGH_SCHOOLS_OF_LAHORE_PAKISTAN
[20] N. W. George E. Forsythe, "Automatic grading programs," *Commun. ACM*, vol. 8, no. 5, pp. 275–278, 1965, doi: https://doi.org/10.1145/364914.36493.
[21] P. Naur, "Automatic grading of students' ALGOL programming," *BIT*, vol. 4, no. 3, pp. 177–188, Sep. 1964, doi: 10.1007/BF01956028/METRICS.
[22] N. T. Le, T. Kojiri, and N. Pinkwart, "Automatic Question Generation for Educational Applications – The State of Art," *Adv. Intell. Syst. Comput.*, vol. 282, pp. 325–338, 2014, doi: 10.1007/978-3-319-06569-4_24.
[23] R. A. C. Ming Liu, "G-Asks: An Intelligent Automatic Question Generation System for Academic Writing Support," *Dialogue & Discourse*, vol. 3, no. 2, p. 3, 2012, [Online]. Available: https://journals.uic.edu/ojs/index.php/dad/article/view/10724
[24] P. Armstrong, "Bloom's Taxonomy," *Vanderbilt Univ. Cent. Teach.*, 2010, [Online]. Available: https://cft.vanderbilt.edu/wp-content/uploads/sites/59/Blooms-Taxonomy.pdf
[25] B. P. Tahani Alsubait, "Automatic generation of analogy questions for student assessment: an Ontology-based approach," *Res. Learn. Technol.*, vol. 20, no. 8, 2012, [Online]. Available: https://journal.alt.ac.uk/index.php/rlt/article/view/1366
[26] E. O'Rourke, E. Butler, A. Díaz Tolentino, and Z. Popović, "Automatic Generation of Problems and Explanations for an Intelligent Algebra Tutor," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11625 LNAI, pp. 383–395, 2019, doi: 10.1007/978-3-030-23204-7_32.
[27] U. S. & S. A.-E. Ghader Kurdi, Jared Leo, Bijan Parsia, "A Systematic Review of Automatic Question Generation for Educational Purposes," *Int. J. Artif. Intell. Educ.*, vol. 30, pp. 121–204, 2020, doi: https://doi.org/10.1007/s40593-019-00186-y.
[28] K. Terada and Y. Watanobe, "Automatic generation of fill-in-The-blank programming problems," *Proc. - 2019 IEEE 13th Int. Symp. Embed. Multicore/Many-Core Syst. MCSoC 2019*, pp. 187–193, Oct. 2019, doi: 10.1109/MCSOC.2019.00034.
[29] T. Wang, X. Su, P. Ma, Y. Wang, and Kuanquan Wang, "Ability-training-oriented automated assessment in introductory programming course," *Comput. Educ.*, vol. 56, no. 1, pp. 220–226, 2011, doi: https://doi.org/10.1016/j.compedu.2010.08.003.
[30] A. E. Fatima Al Shamsi, "An Intelligent Assessment Tool for Students' Java Submissions

in Introductory Programming Courses," *J. Intell. Learn. Syst. Appl.*, vol. 4, no. 1, p. 2, 2012, [Online]. Available: https://www.scirp.org/journal/paperinformation?paperid=17557

[31] D. Insa and J. Silva, "Automatic assessment of Java code," *Comput. Lang. Syst. Struct.*, vol. 53, pp. 59–72, 2018, doi: https://doi.org/10.1016/j.cl.2018.01.004.

[32] A. Ettles, A. Luxton-Reilly, and P. Denny, "Common Logic Errors Made By Novice Programmers," *ACM Int. Conf. Proceeding Ser.*, pp. 83–89, Jan. 2018, doi: 10.1145/3160489.3160493;JOURNAL:JOURNAL:ACMCONFERENCES;PAGEGROUP:STRING:PUBLICATION.

[33] B. Cole, D. Hakim, D. Hovemeyer, R. Lazarus, W. Pugh, and K. Stephens, "Improving your software using static analysis to find bugs," *Proc. Conf. Object-Oriented Program. Syst. Lang. Appl. OOPSLA*, vol. 2006, pp. 673–674, 2006, doi: 10.1145/1176617.1176667.

[34] M. L. D. S. Lynnette Noblitt, Diane E. Vanc, "A Comparison of Case Study and Traditional Teaching Methods for Improvement of Oral Communication and Critical-Thinking Skills," *J. Coll. Sci. Teach.*, vol. 39, no. 5, 2010, [Online]. Available: https://www.researchgate.net/publication/234678040_A_Comparison_of_Case_Study_and_Traditional_Teaching_Methods_for_Improvement_of_Oral_Communication_and_Critical-Thinking_Skills

[35] Y. A. et al Ters-Yüz Edilmiş Öğrenme, "The Effect of Flipped Learning Approach on Academic Achievement: A Meta-Analysis Study," *Hacettepe Üniversitesi Eğitim Fakültesi Derg. (H. U. J. Educ.*, vol. 34, no. 3, pp. 708–727, 2019, doi: 10.16986/HUJE.2018046755.

[36] N. K. Stephen H. Edwards, "Investigating Static Analysis Errors in Student Java Programs," *ICER '17 Proc. 2017 ACM Conf. Int. Comput. Educ. Res.*, pp. 65–73, 2017, doi: https://doi.org/10.1145/3105726.3106182.

[37] Y. W. Yuto Yoshizawa, "Logic Error Detection System based on Structure Pattern and Error Degree," *Adv. Sci. Technol. Eng. Syst.*, vol. 4, no. 5, 2019, [Online]. Available: https://www.astesj.com/v04/i05/p01/

[38] P. L. S. Barbosa, R. A. F. do Carmo, J. P. P. Gomes, and W. Viana, "Adaptive learning in computer science education: A scoping review," *Educ. Inf. Technol.*, vol. 29, no. 8, pp. 9139–9188, Jun. 2024, doi: 10.1007/S10639-023-12066-Z/METRICS.

[39] Ö. Aydın and E. Karaarslan, "Is ChatGPT Leading Generative AI? What is Beyond Expectations?," *SSRN Electron. J.*, Jan. 2023, doi: 10.2139/SSRN.4341500.

[40] F. de O. S. Rodrigo Elias Francisco, "Intelligent Tutoring System for Computer Science Education and the Use of Artificial Intelligence: A Literature Review," *CSEDU*, 2022, [Online]. Available: https://www.scitepress.org/Papers/2022/110844/110844.pdf

[41] S. Vamsi, V. Balamurali, K. S. Teja, and P. Mallela, "Classifying Difficulty Levels of Programming Questions on HackerRank," *Learn. Anal. Intell. Syst.*, vol. 3, pp. 301–308, 2020, doi: 10.1007/978-3-030-24322-7_39.