A Study of Introductory Exercise Problems for Novice Students in Java Programming Learning Assistant System

Mustika Mentari, Nobuo Funabiki, Safira Adine Kinari, Komang Candra Brata, Noprianto, Yan Watequlis Syaifudin, Triana Fatmawati, Indra Dharma Wijaya

***Abstract*—Nowadays, *Java programming* is often taught in an introductory *Object-Oriented Programming (OOP)* course to novice students in universities around the world. In this course, students need to be familiar with typing source codes using computers first. Then, they will learn the grammar and syntax of *Java* from introductory levels. For effective studies, dedicated exercise problems for them should be implemented in *Java Programming Learning Assistant System (JPLAS)* that has been developed in our group to assist self-studies at home. In this paper, we present a study of three exercise problems in *JPLAS* that are dedicated to novice students, namely, *code typing problem (CTP)*, *code fixing problem (CFP)*, and *value trace problem (VTP)*. A *CTP* instance requests to type a given source code on a computer. A *CFP* instance requests to type a given erroneous code while fixing the errors. A *VTP* instance requests to answer the values of important variables in a given code. The correctness of any answer is checked through *string matching* on the *answer interface* on a web browser. For evaluations, we generated and assigned five *CTP*, 10 *CFP*, and 10 *VTP* instances to first-year undergraduate students in two related courses at State Polytechnic of Malang, Indonesia. In the results, most students correctly solved all the questions, where a statistically significant difference in the *time span* between *punctual* and *delayed students* in completing the exercises, provided positive feedback for the usability questionnaire and achieved higher final exam scores in the courses than students who did not solve the instances. Thus, the effectiveness and validity of the proposal are confirmed.**

***Index Terms*—Java programming, novice student, JPLAS, code typing problem, code fixing problem, value trace problem.**

Manuscript received February 24, 2025; revised August 25, 2025. This work was supported by Okayama University and State Polytechnic of Malang.

Mustika Mentari is a PhD candidate at the Department of Information and Communication Systems, Okayama University, Okayama, Japan (e-mail: pqt85hm5@s.okayama-u.ac.jp).

Nobuo Funabiki is a Professor at the Department of Information and Communication Systems, Okayama University, Okayama, Japan (e-mail: funabiki@okayama-u.ac.jp).

Safira Adine Kinari is a Master's student at the Department of Information and Communication Systems, Okayama University, Okayama, Japan (e-mail: safiraak@s.okayama-u.ac.jp).

Komang Candra Brata is a PhD candidate at the Department of Information and Communication Systems, Okayama University, Okayama, Japan (e-mail: p1qk35rx@s.okayama-u.ac.jp).

Noprianto is a PhD candidate at the Department of Information and Communication Systems, Okayama University, Okayama, Japan (e-mail: py3o92mw@s.okayama-u.ac.jp).

Yan Watequlis Syaifudin is a lecturer at the Department of Information Technology, State Polytechnic of Malang, Malang, Indonesia (e-mail: qulis@polinema.ac.id).

Triana Fatmawati is a lecturer at the Department of Information Technology, State Polytechnic of Malang, Malang, Indonesia (e-mail: triana@polinema.ac.id).

Indra Dharma Wijaya is a lecturer at the Department of Information Technology, State Polytechnic of Malang, Malang, Indonesia (e-mail: indra.dharma@polinema.ac.id).

I. Introduction

Nowadays, *Java programming* has become a cornerstone in computer science education. Many universities worldwide are incorporating it into their curricula as the primary *Object-Oriented Programming (OOP)* language [1][2][3]. As an introductory *OOP* language, *Java* provides a structured and practical approach to understanding fundamental concepts such as classes, objects, inheritance, polymorphism, and encapsulation. Its extensive standard libraries and user-friendly syntax of *Java* make it accessible to beginners while offering the depth required for complex applications [4].

The widespread adoption of *Java* stems from its versatility, robustness, and platform-independent nature, enabled by the *Java Virtual Machine (JVM)* [5][6]. Its real-world relevance in industries, such as web developments, mobile applications, and enterprise solutions, further solidifies its position as an essential tool for aspiring programmers. Consequently, students should be equipped with skills of *Java programming* that are both academically foundational and professionally valuable [7].

To support novice students in self-studies of *Java programming*, we have developed the *Java Programming Learning Assistant System (JPLAS)* [8], and implemented it into *Java programming* courses [9]. To facilitate step-by-step studies, *JPLAS* provides various types of exercise problems, each designed with specific learning objectives and difficulty levels:

- *Grammar Concept Understanding Problem (GUP)* requires students to ask questions regarding the grammar or libraries in the given source code based on the prompts provided [10].
- *Value Trace Problem (VTP)* requires students to provide answers regarding the values of important variables or output messages in the source code that are anticipated to appear after execution [11].
- *Mistake Correction Problem (MCP)* requires students to provide answers for pairs of incorrect code segments and their corrections, along with the correct solution, in the given source code that contains multiple errors [12].
- *Element Fill-in-blank Problem (EFP)* requires students to fill in the blanks in the given source code that contains several missing elements [13].
- *Code Completion Problem (CCP)* requires students to complete the given source code that contains several missing elements, without explicitly indicating the locations of the blanks [14].

- *Phrase Fill in blank Problem (PFP)* requires students to fill in the blanks of phrases in the given source code, where each phrase may consist of multiple elements [15].
- *Code Writing Problem (CWP)* requires students to write program code based on the specifications provided and shown in the available test code [8].

In a *Java programming* course, students must first become familiar with reading source codes and typing them on personal computers (PCs). Many of them lack prior experiences using computers, including code editors [16]. To begin learning programming, they need to adapt to the technical environment required for coding [17]. Although *Integrated Development Environments (IDEs)* offer useful features like syntax highlighting, debugging tools, and version control, they may appear intimidating or overly complex for novice users [18]. Additionally, features like auto-completion may lead to dependency, potentially hindering their understanding of basic programming concepts.

After novice students become accustomed to reading and typing source codes, they need to understand the grammar and syntax of *Java programming*. In *Java programming*, syntax refers to the set of rules that define the correct structure of a *Java* code, including the proper arrangement of symbols, keywords, and punctuations. For example, every statement in a code must end with a semicolon `;`, and a code block is enclosed within curly braces`{ }`.

Then, novices need to understand the semantic behaviour of a code. The semantic behaviour represents how the code runs and interacts by following the language's rules. A code must ensure the correct variable usage, object communications, and error handling. This behaviour is closely tied to the logic in the code, as it dictates how the program processes and produces the result. Understanding semantic behaviour will help students write a clear and reliable code. The semantic behaviour includes both structural correctness and meaningful logic [19]. Unfortunately, we have not studied proper exercise problems to be offered to novice students in *JPLAS* to address these challenges systematically.

In this paper, we present a study of three introductory exercise problems in *JPLAS* that are designed for novice students. These problems include the *code typing problem (CTP)*, the *code fixing problem (CFP)* [20], and the *value trace problem (VTP)*. A *CTP* instance requests the student to type a given source code as it is in a computer. A *CFP* instance requests student to type a given code that has several errors while fixing them. A *VTP* instance requests student to answer the values of important variables in a given code. Answer correctness is evaluated through *string matching* on an *answer interface* running on a web browser.

For evaluations of the proposal, we generated five *CTP*, 10 *CFP*, and 10 *VTP* instances using simple source codes that cover fundamental *Java programming* topics. Then, we assigned them to first-year undergraduate students who were taking two related courses at State Polytechnic of Malang, Indonesia, namely, the *basic Java programming* and the *algorithms and data structures*. Then, a part of them submitted their answers before the deadline called *punctual students*, whereas others had submissions after the deadline called *delayed students*. For comprehensive evaluations, we compared the differences between these two student groups, in addition to the analysis of the performances in each group.

The results show that most of the students quickly adapted to solving *CTP* instances and demonstrated the ability to read and type codes effectively. They also successfully corrected the errors in *CFP* instances, although they felt more difficulty than *CTP*. Subsequently, they were able to read the codes and infer the values of variables as required in *VTP* instances. Most students' opinions regarding these instances were positive. As a qualitative feedback, the *SUS (System Usability Scale)* resulted in the *acceptable category*, indicating that students found the proposal usable and well-designed. A significant difference in *time span* was observed between the *punctual* and *delayed students*, while their solution performances remained similar. Furthermore, the final exam scores comparison shows that the students who completed the given instances outperformed those who did not, with statistically significant differences. Thus, the effectiveness and validity of the proposal are confirmed.

The rest of this paper is organized as follows: Section II discusses related works in literature. Sections III-V present the *code typing*, *code fixing*, and *value trace problems*, respectively. Section VI shows their application results to novice students. Section VII contains discussions on the results. Finally, Section VIII concludes this paper with future works.

II. Related Works in Literature

In this section, we present key findings from related studies relevant to this paper. By reviewing previous research, we will highlight the significance of past studies, identify research gaps, and demonstrate how our work contributes to advancing this field.

A. Java Programming Education

Java has been a cornerstone of computer science education for decades. Its widespread adoption in both academia and industry has made it a top programming language for educational purposes [21][22]. Its object-oriented nature, simple syntax, and extensive ecosystem make it an ideal language for teaching programming concepts such as classes, inheritance, and polymorphism. Educational institutions worldwide, from high schools to universities, use *Java* as an introductory programming language to assist learners in establishing fundamental skills in programming and computational problem-solving [23][24][25].

The importance of *Java programming* education is amplified by the language's role in fostering critical thinking and problem-solving skills. Through *Java programming*, students learn not just how to code but also how to approach problems systematically. This structured problem-solving approach is fundamental in computer science and is highly valuable in professional settings, where *Java* is commonly used for developing software applications, mobile apps, and even big data solutions [26]. Moreover, *Java* is frequently employed in coding boot camps and online educational platforms, providing students with flexible environments to master the language in a relatively short time [27][28].

Despite its prominence in education, there are ongoing challenges to ensure that students fully understand the theoretical and practical aspects of *Java* programming. Researches have shown that students often struggle with abstract concepts such as memory management and concurrency, which are critical when learning *Java* [29][30]. Instructors must balance teaching fundamental concepts with providing hands-on programming experiences, effectively supporting project-based learning. Project-based learning allows students to engage in real-world scenarios, strengthening their theoretical understanding of *Java* [31][32].

To improve the effectiveness of *Java programming* education, educators must adapt to technological advancements by integrating modern tools and frameworks within the *Java* ecosystem. The introduction of *Integrated Development Environments (IDEs)* such as *IntelliJ IDEA*, *Eclipse*, and *NetBeans* equips students with powerful resources to write, debug, and optimize their codes, making learning experiences more interactive and productive [33][34]. Furthermore, the growing use of *Java* in *Android* app development opens the doors for students to explore mobile developments, further enhancing the language's relevance in today's job market [35][36].

Finally, collaboration and peer-based learning play a vital role in *Java* programming education. Studies have demonstrated benefits of collaborative learning, where students work together to solve problems and learn from each other [37][38]. This approach fosters a deeper understanding of *Java* concepts and prepares students for teamwork in professional environments, where collaboration is often essential to success. As *Java* continues to evolve, educational strategies must adapt, ensuring that future generations are well equipped to meet the challenges of the digital age [39][40][41][42].

B. Java Programming Study Tools

The study of *Java programming* has evolved significantly with introductions of various learning tools designed to facilitate understanding, enhance comprehensions, and develop practical coding skills. These tools range from interactive *Integrated Development Environments (IDEs)* to online coding platforms, automated assessment systems, and gamified learning environments. The effectiveness of these study tools lies in their ability of providing real-time feedback, actively engages learners, and simulates real-world coding scenarios [17].

One of the most widely used *Java* learning tools is the *Integrated Development Environment (IDE)*, including *IntelliJ IDEA*, *Eclipse*, and *NetBeans*. These *IDEs* not only provide structured workspaces for writing and debugging *Java* codes, but also offer intelligent code suggestions, error detections, and automated refactoring, significantly aiding students in understanding programming logics [33]. Some studies have shown that students using feature-rich *IDEs* demonstrate improved problem-solving abilities and reduced syntax errors [43].

In addition to *IDEs*, online coding platforms have become essential tools for *Java* learners. These platforms offer structured courses, interactive coding exercises, and real-world project assignments that help students build confidence in *Java programming* [43]. Researches indicate that students who actively engage in hands-on coding exercises on online platforms tend to retain *Java* concepts more effectively than those who rely solely on traditional lectures [32].

Furthermore, automated assessment and feedback tools play a crucial role in *Java* education. Tools such as *Moodle*, *CodeRunner*, and *JUnit* testing frameworks enable educators to automate code evaluations, instantly assess students' programming assignments, and provide personalized feedback [3]. Automated testing not only assists students in debugging their code but also promotes self-directed learning and iterative problem-solving [44].

Another category of *Java* learning tools gaining popularity is game-based learning platforms. Gamified environments, such as *CodeGym* and *Codingame*, employ challenges, levels, and interactive problem-solving to make *Java* learning more engaging and enjoyable [45]. Some researches have demonstrated that game-based learning significantly enhances motivation and retention rates among programming students [46].

Additionally, collaborative coding platforms such as *GitHub*, *GitLab*, and *Replit* have become indispensable tools for *Java* students. These platforms support version controls, peer code reviews, and collaborative software developments, all of which are essential for gaining real-world programming experiences [47]. Studies indicate that students who actively engage in collaborative coding develop a deeper understanding of *Java* programming concepts and teamwork skills, both of which are highly valued in the software industry [32].

Despite the advantages of these learning tools, challenges remain in ensuring that students effectively utilize them. Some students experience tool overloads, where the abundance of resources leads to confusion rather than enhanced learning [48]. Additionally, insufficient instructor guidance on the optimal integration of the tools into the learning process can hinder their effectiveness [49]. Therefore, educators must provide structured guidance on when and how to utilize *Java* study tools to optimize student learning outcomes [23].

Java programming study tools have transformed the way that students learn and practice coding. Ranging from *IDEs* and online coding platforms to automated assessment and collaborative coding tools, they enhance engagements, improve skill retention, and better prepare students for the demands of the software industries [43]. However, effective use of these tools requires structured guidance from educators and a balanced approach to prevent being overwhelmed with excessive options. As *Java programming* education continues to evolve, the integration of innovative study tools will become more essential in training future software developers [50].

III. Code Typing Problem

In this section, we introduce a new problem type in *JPLAS* called the *code typing problem (CTP)* that is designed for novice students to practice typing source code using computers.

A. Definition of CTP

CTP is designed to help novice students practice typing program codes using a computer. It will help them to get used to typing programming languages. The exercises provided in *CTP* require students to enter answers by retyping the provided source code while referring to the example code. Answer correctness is validated line by line. The core idea is to offer a quick learning tool that strengthens beginners' memory of syntax through retyping exercises. Over time, students become more comfortable with the *Java* coding environment.

B. Answer Interface of CTP

Figure 1 depicts the answer interface for a *CTP* instance. The layout is divided into two main sections. The right side displays the source code in this instance, which cannot be copied. The left side serves as the answer area, that has the same number of lines as the source code. Students are required to retype the code in this area, line by line, to ensure exact matching. If the typed code line is different from the original line, it is highlighted in *red* as an incorrect answer. If the answer is correct but the line formatting is not yet neat, it is highlighted in *yellow*. Otherwise, it remains *white*.

C. CTP Instance Generation Procedure

To generate new *CTP* instances, a teacher needs to collect source codes related to basic *Java programming* topics that novice students should study.

The answer interface for the *CTP* instance is created using the following procedure:

1) Collect a *Java basic programming* source code from a teaching material or a website.
2) Save this source code and the correct answers (code lines) in an input text file.
3) Run the *answer interface generation program* with the input text file to generate *HTML*, *CSS*, and *JavaScript* files for the answer interface, which runs on a web browser.
4) Add image files to the *HTML* file if necessary to complete the new *CTP* instance.

D. CTP Instance Example

Listing 1 shows a part of the sample source code. This code is selected as an introductory *Java programming* material that contains a class, variables, a data type, an operator, a condition, and standard outputs.

IV. Code Fixing Problem

In this section, we present the *code fixing problem (CFP)* for novice students to learn *Java programming* syntax.

A. Definition of CFP

CFP is designed to help novice students understand *Java* grammar and syntax. The answer interface for this problem has the same appearance and procedure as *CTP*. However, differences lie in the source code provided to be retyped. This code is intentionally injected with errors in several parts. Students need to understand the correct grammar appropriate in the wrong parts and fix them at retyping the code line by line.

B. Error Injection Algorithm

First, to generate a feasible *CFP* instance with a unique correct answer, the *blank element selection algorithm* in [13] is adopted to select the elements in the source code for error injections.

Second, the error injection method in [12] is applied to the selected elements for errors. This method consists of the *grammar error injection* and the *name error injection*.

1) Grammar Error Injection: The *grammar error injection* creates errors in *Java* grammar. Table I shows the list of candidates for errors. A selected candidate in a source code will be changed to another candidate in the same row. For example, if the selected candidate is "byte," the possible changes could be "int," "short," or "long.

TABLE I
GRAMMAR ERROR CANDIDATE LIST.

no	component	type
1	byte, int, short, long	data type
2	(,) and (.)	mark
3	float, double	data type
4	void, byte, int, short, long, float, double	data type
5	String, short, int, long	data type
6	boolean, Boolean	data type
7	char, Character	data type
8	byte, Byte	data type
9	short, Short	data type
10	int, Integer	data type
11	long, Long	data type
12	float, Float	data type
13	double, Double	data type
14	while, if	Java keyword
15	if, while	Java keyword
16	switch, case	Java keyword
17	break, continue	Java keyword
18	++, –	operator
19	class, interface	Java keyword
20	main, Main	Java keyword
21	static, New	Java keyword
22	(;), (:)	mark
23	+, -, *, /, %	operator
24	==, !=, ¿, ¡, ¡=	operator
25	(, , [	opening mark
26	), ,]	closing mark

2) Name Error Injection: The *name error injection* creates errors in the name of a variable and a method. A parameter can be changed from the previous one. As shown in Table II, when a name consisting of two or more names is connected by "_" or is separated from the collection of them, the first letter is changed from the lower case to the upper case or vice versa, or one character is added before the name or after the name. A similar character is selected here, as shown in Figure 2, to create confusion.

C. CFP Instance Example

Listing 2 shows an example of the generated source code with errors in a *CFP* instance. The error results are marked in *blue*. They include "interface" derived from "class", "new" derived from "static", "Index" derived from "index", and others.

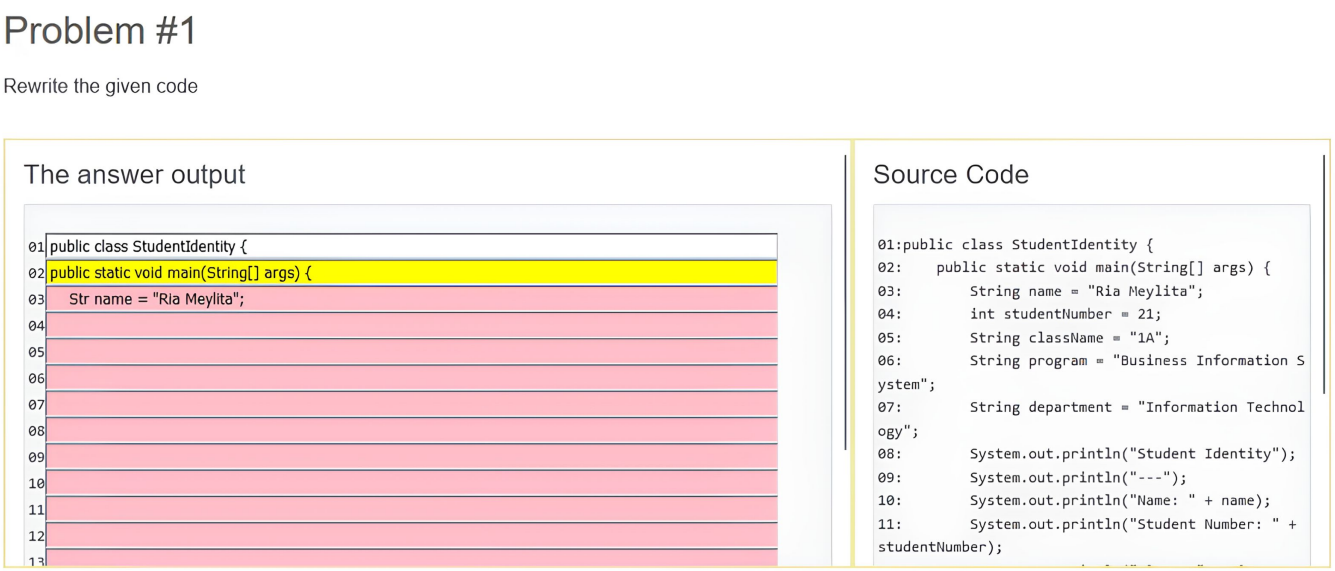

Fig. 1. Answer interface for *code typing problem*.

```
1  public class MyOperator {
2   public static void main(String[] args) {
3    int x = 15;
4    System.out.println("x-- = " + x--);
5    System.out.println("After evaluation, x = "
       + x);
6    x = 15;
7    System.out.println("--x = " + --x);
8    System.out.println("After evaluation, x = "
       + x);
9    int y = 8;
10   if (x > y) {
11    System.out.println("x is greater than y");
12   } else {
13    System.out.println("x is less than or equal
       to y");
14   }
15  }
16 }
```

Listing 1. Example code for *CTP*.

TABLE II
ERROR NAME CANDIDATE LIST.

no	before	after
1	StringName	String_Name
2	StringName	String Name
3	Name	name
4	name	Name
5	Name	Namee
6	Name	NName

D. Hint Function

The answer interface for *CFP* checks the correctness of a typed source code line by line. Thus, students can easily know which line has error elements. However, they need to find which elements have errors in the erroneous line, which can be difficult for novice students. As a result, they may continue making mistakes.

To avoid this desired situation, a *hint function* is implemented to help a student find the error elements. As shown in Figure 3, this function highlights the incorrect elements along with the input form containing the student's corresponding input. This allows a student to easily correct errors. In addition, the other correct elements are displayed, separated by spaces, to aid comprehensions.

To avoid overuse of the hint function, it becomes available only after a student attempts to answer at least three times. Once all the elements in a line are correct, this line is automatically copied to the answer form.

V. Value Trace Problem

In this section, we present the *value trace problem (VTP)* for novice students to understand the semantic behaviours of *Java* source codes that often implement fundamental algorithms or data structures.

A. Definition of VTP

The answer interface to *VTP* asks a student to guess the actual value of a variable in the source code. This code basically implements a fundamental algorithm or a data structure.

B. User Interface of VTP

Figure 4 shows the answer interface. It consists of a source code on the left and question-and-answer forms on the right. If the answer is correct, the colour of the input form keeps *white*. If the answer is wrong, it turns *red*.

C. Automated Question Generation Procedure

A *VTP* instance can be created through the following procedure:

1) Select a proper source code, manually.
2) Select the variables to be printed, insert the standard output commands for them into the source code, and save it as a text file, manually.
3) Compile and run the source code, and write the standard outputs into the text file at the bottom, as shown in Listing 3.
4) With this text file, generate the corresponding *HTML*, *CSS*, and *JavaScript* files, automatically.

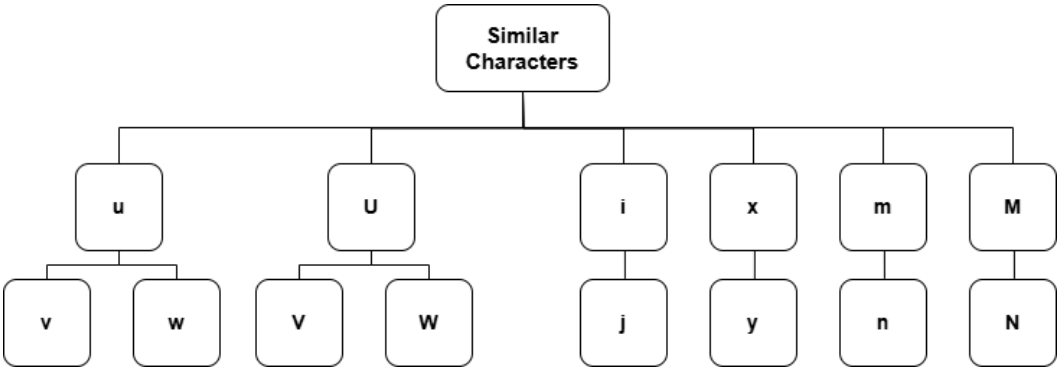

Fig. 2. Error character candidate list.

```
1  public interface Searching {
2   public new void main (String[] args){
3    double[] arr = {7.8, 8.2, 9.7, 12.6, 9.0}
4    double keyword = 12.6;
5    int index = -1;
6    for(int i = 0; i < arr.length; i++){
7     if(keyword == aarr[i]){
8      Index=i;
9      continue;
10    }
11    if(iindex == -1){
12     System.out.println("Not available");
13    }
14    else{
15     System.out.print("The data you are looking
        for is:"+arr[index]+" at index "+ index);
16    }
17   }
18  }
19 }
```

Listing 2. Example code for *CFP*.

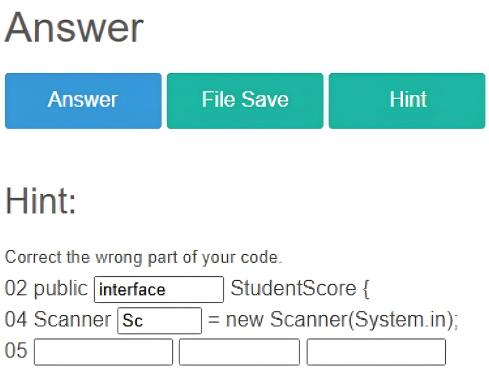

Fig. 3. Hint function interface.

VI. Application Results

In this section, to evaluate the study of three exercise problems in *JPLAS* for novice students, we show their application results to first-year undergraduate students in the information technology department at State Polytechnic of Malang, Indonesia.

A. Assignment Setup

These students were taking the two related courses at State Polytechnic of Malang, Indonesia. In the *basic Java programming* course, five *CTP* instances and 10 *CFP* instances are assigned. In the *algorithms and data structures* course, 10 *VTP* instances are assigned.

For them, some students submitted the answers before the deadline, referred to as *punctual students*. On the other hand, some students got delayed in their answer submissions, referred to as *delayed students*. In this *basic Java programming* course, there are 40 *punctual students* and 33 *delayed students*. In the *algorithms and data structures* course, there are 46 *punctual students* and 32 *delayed students*. For comprehensive evaluations, we analysed performances of the students in each group and differences between the two groups.

B. Results of Code Typing Problem

First, we examine the application results of *code typing problem (CTP)* instances.

1) CTP *Instances:* Table III presents the topics and the corresponding number of lines in each instance. Based on the students' solution results, we analysed the average number of submission attempts and the average correct answer rate.

TABLE III
GENERATED *CTP* INSTANCES.

instance ID	topic	# of lines
1	class	16
2	variable	14
3	data type	19
4	operator	20
5	scanner input	16

2) Results of Individual Students: First, we analysed the results of individual students. For *punctual students*, Figure 5 shows the student ID, the average number of submission attempts, and the correct answer rate for each student. To improve the visibility in related figures, the upper limit of the y-axis is fixed to 40.0 submissions and 100.2% accuracy. To reduce overlapping markers caused by numerous perfect accuracy scores, some 100% points of correct answer rate are not shown. A total 39 students (97.5%) achieved a perfect score of 100%, while one student (2.5%) scored 98.75% (student at ID=16). Given the high accuracy rates, more challenging *CTP* instances may be needed in future iterations. Regarding submission attempts, 37 students (92.5%) submitted their answers an average of 1 to 3 times, while three students (7.5%) required 4 to 5 attempts.

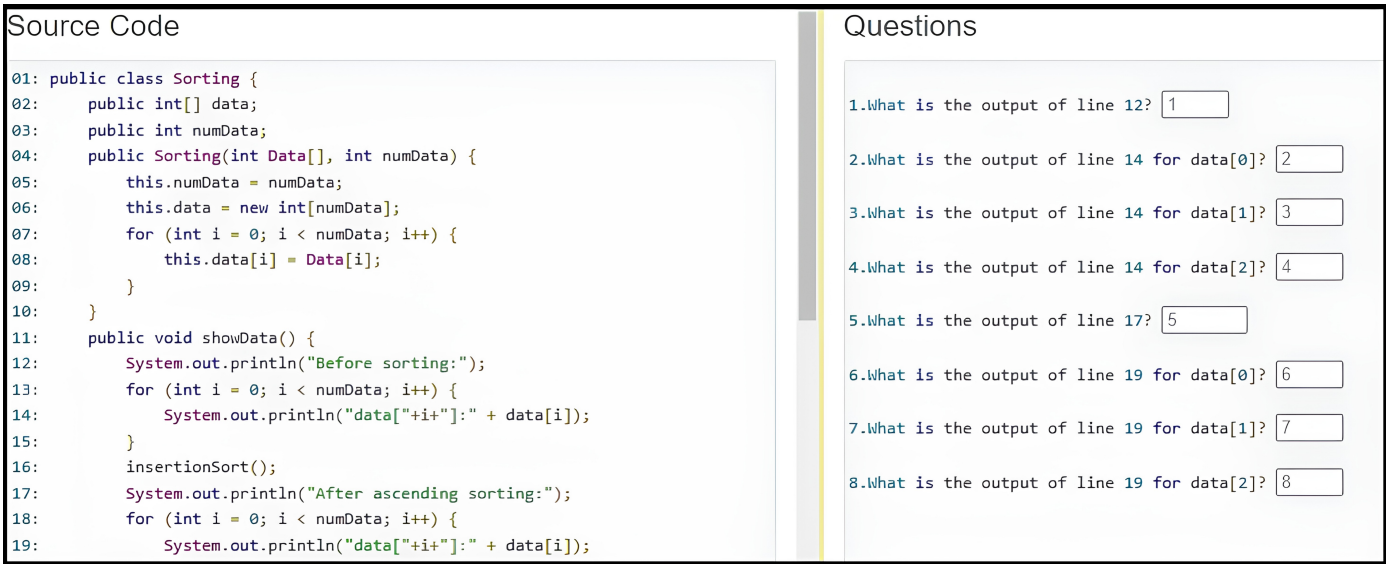

Fig. 4. Answer interface for *value trace problem*.

```
1  public class Searching {
2   public static int search(double key, double[]
        arr) {
3    for (int i = 0; i < arr.length; i++)
4     if (key <= arr[i]) return (key == arr[i]) ?
        i : -1;
5     return -1;
6    }
7    public static void main(String[] args) {
8     double[] data = {2.7, 2.9, 3.04};
9      for (double num : data) System.out.print(
       num + " ");
10      System.out.println();
11      int index;
12    for (double key : data)
13     if ((index = search(key, data))!=-1)
14      System.out.println("Data " + key + "
       found at index " + index);
15   }
16  }
17
18  Answer :
19
20  2.7 2.9 3.04
21  Data 2.7 found at index 0
22  Data 2.9 found at index 1
23  Data 3.04 found at index 2
```

Listing 3. Example code for *VTP*.

For *delayed students*, Figure 6 shows that most of them (29 out of 33, or 87.8%) scored 100%, where this rate was lower than that of *punctual students*. The student at ID=33 had the highest number of submissions on average, at 5.6 times.

The most common errors among students come from simple careless typing mistakes. Overall, the students demonstrated strong proficiency in typing *Java* codes using computers.

3) Results of Individual Instances: Next, we analysed the results of individual *CTP* instances across two student groups. Figure 7 shows the ID, average correct answer rate, and average number of submissions for each instance per student group. For *punctual students*, four instances achieved a 100% correct answer rate, while the instance at ID=5 reached 99.84% with an average of 1.9 submissions. For *delayed students*, all the instances showed the high accuracy, ranging from 98.03% to 100%, and a slightly higher submission average than the *punctual students*. The instance at ID=1 showed the highest average submissions, 3.9 for *punctual students* and 3.5 for *delayed students*. Overall, the students demonstrated consistent performances.

C. Results of Code Fixing Problem

Second, we examine the application results of *code fixing problem (CFP)* instances.

1) CFP *Instances:* Table IV shows the topic of each source code, the number of lines, and the number of injected errors in each instance. From the solution results, we calculated the average number of submissions and the average correct answer rate.

TABLE IV
GENERATED *CFP* INSTANCES.

instance ID	topic	# of lines	# of errors
1	condition part 1	23	8
2	condition part 2	19	8
3	looping part 1	16	6
4	looping part 2	21	7
5	array part 1	14	5
6	array part 2	20	10
7	function part 1	16	7
8	function part 2	20	8
9	sorting	19	5
10	searching	18	5

2) Results of Individual Students: Figure 8 shows the student ID, the average number of submissions, and the correct answer rate for each *punctual student*. 38 students (95%) scored 100%, while two students (5%) scored 99.52% (student at ID=8) and 98.89% (student at ID=39), with the average submission attempts of 2.1 and 10, respectively. Although both performed well, the student at ID=8 could have achieved a higher rate with more submissions, while the student at ID=39 made 10 attempts, indicating a slight

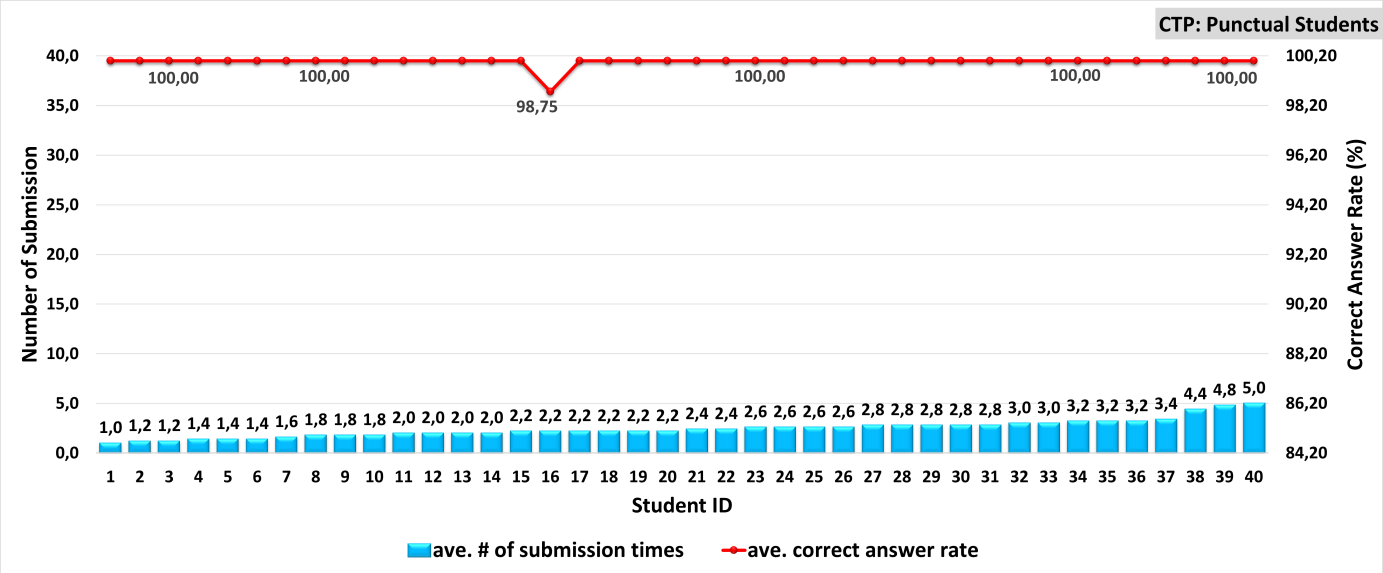

Fig. 5. Individual results for *punctual students* on *CTP*.

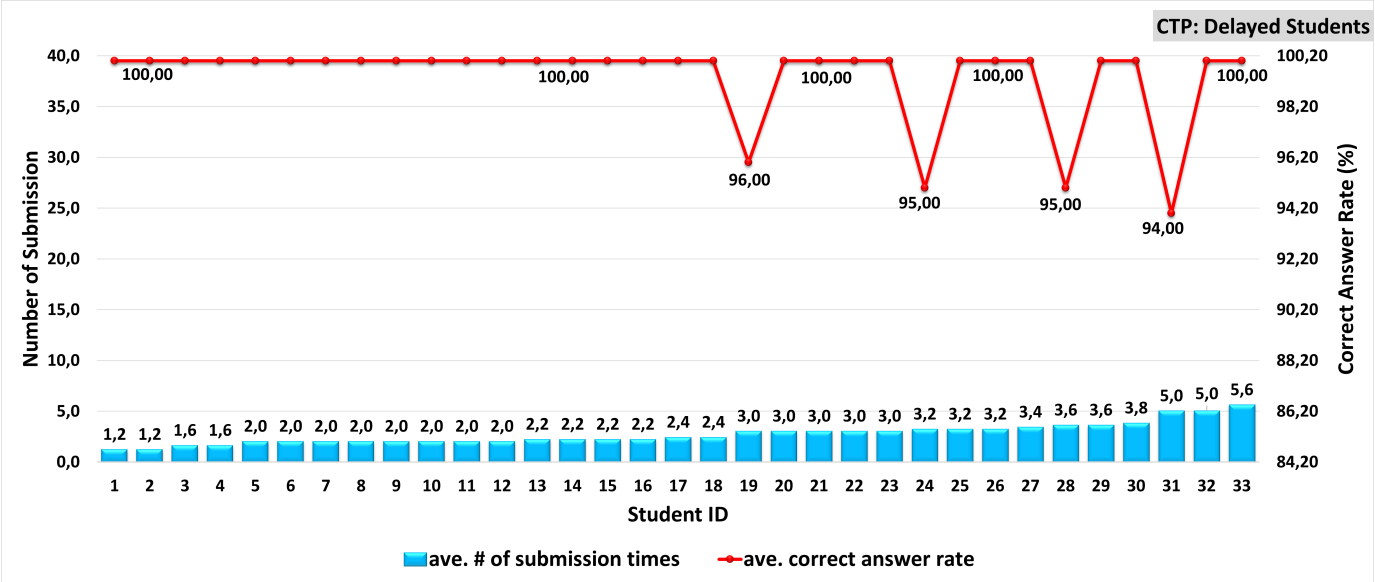

Fig. 6. Individual results for *delayed students* on *CTP*.

performance drop. Among the 40 *punctual students*, 29 (72.5%) submitted 1–5 times, 10 (25%) submitted 6–10 times, and one student submitted up to 25 times, achieving a perfect score.

For *delayed students*, Figure 9 indicates that many of them maintained the high accuracy, where 22 out of 33 (67%) achieved a perfect score, although this rate was notably lower than that of *punctual students*. The student at ID=33 required an average of 22.9 submissions, indicating challenges in solving these problems. The number of submissions by students increased, suggesting the requirements of more careful attempts or verifications.

3) Results of Individual Instances: Figure 10 shows the instance ID, the average correct answer rate, and the average number of submissions for each instance across both student groups. The highest number of submissions occurred for the instance at ID=1 in both *punctual* and *delayed* students, with 9.8 and 6.2 submissions, respectively. This is because students often require more attempts to understand in the beginning. Slight accuracy drops were observed among *delayed students* in the instances at ID=1 (98.46%), 4 (99.29%), 5 (99.57%), 6 (99.70%), 8 (98.46%), and 9 (98.26%), where 100% accuracy was achieved in almost all *punctual students*. These instances covered more advanced topics, such as *queue*, *linked list*, and *tree*. They can be challenging for novice students. Thus, additional supports or practices will be necessary for students.

4) Results of Hint Function Use: Figure 11 shows the number of students who used the *hint function* for each instance in both student groups. 20 *punctual students* and 29 *delayed students* used it. The highest usage was observed in the instance at ID=1 where students tried to know what it is. For other instances, only a limited students used it, which suggests that the hint function is helpful for students who do not understand *Java programming* well.

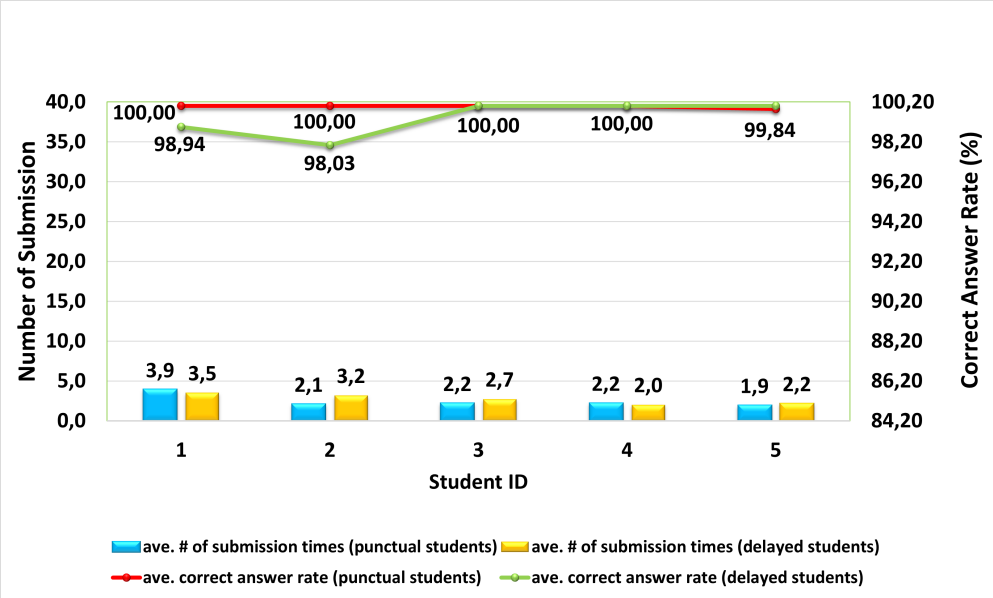

Fig. 7. Individual instance results for *CTP*.

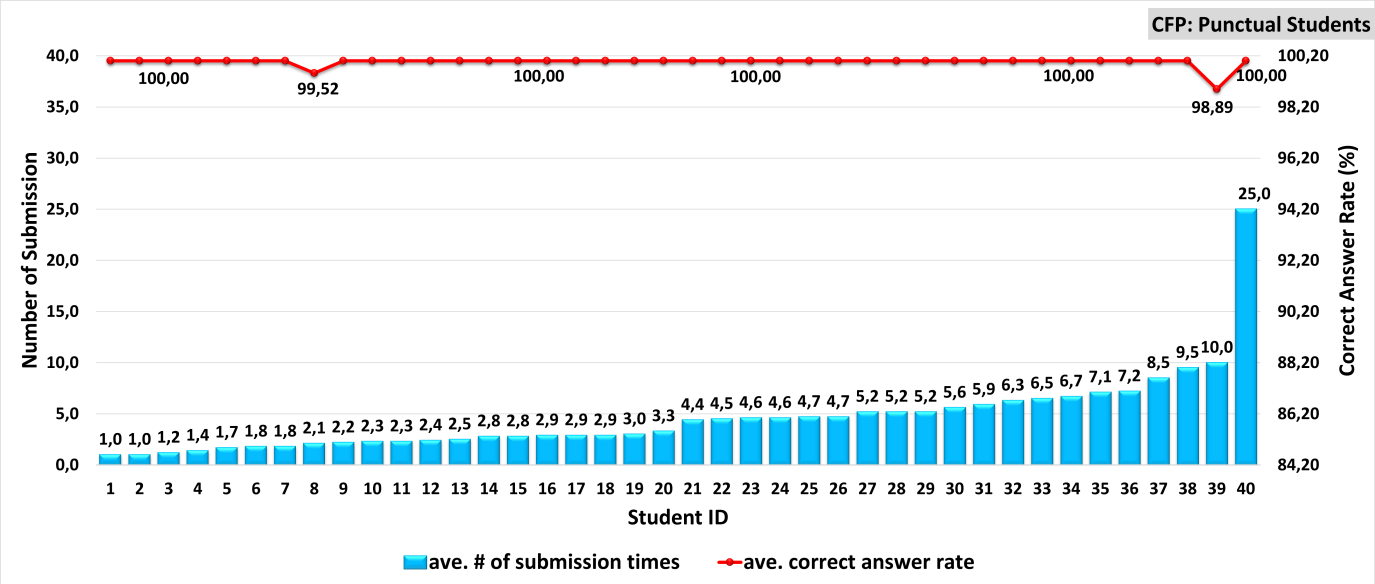

Fig. 8. Individual results for *punctual students* on *CFP*.

D. Results of Value Trace Problem

Third, we examine the application results of *value trace problem (VTP)* instances.

1) VTP *Instances:* Table V shows the topics and the number of questions for each instance. From the students' solution outcomes, we analysed both the average number of submission attempts and the average correct answer rate to assess their overall performance.

2) Results of Individual Students: Figure 12 shows the student ID, the average number of submissions, and the average correct answer rate for each of *punctual students*. Among the 46 students, 35 (76%) achieved a 100% correct rate, and 11 (24%) scored between 90.75% and 99%, with most over 98%. The lowest score of 90.75% was from the student at ID=17 with an average of 3.6 submissions. The student at ID=46 had the highest submission count (20.5) and a 97.75% correct rate, showing his/her strong effort. The students at ID=40, 42, 43, and 45 achieved 100% accuracy after more than 10 attempts, while the others had good performances with fewer than 10 submissions.

TABLE V
GENERATED *VTP* INSTANCES.

instance ID	topic	# of questions
1	searching	8
2	sorting	8
3	stack	8
4	queue	8
5	singly linked list part 1	10
6	singly linked list part 2	10
7	doubly linked list part 1	10
8	doubly linked list part 2	8
9	tree part 1	10
10	tree part 2	8

Figure 13 shows the student ID, the average number of

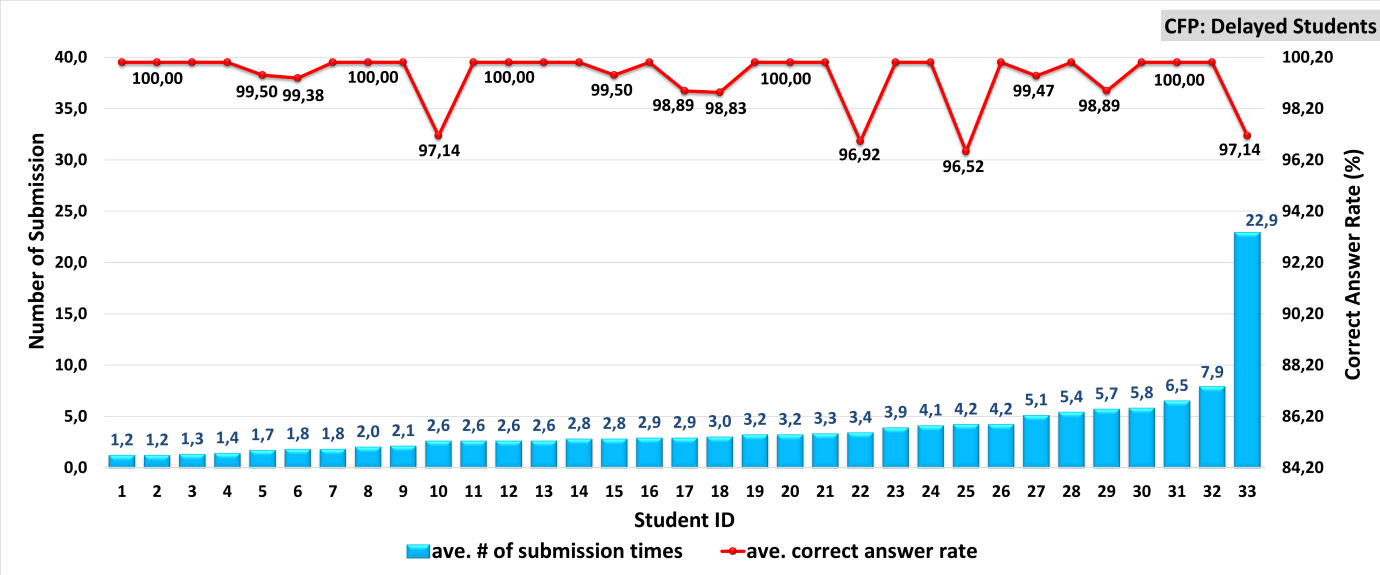

Fig. 9. Individual results for *delayed students* on *CFP*.

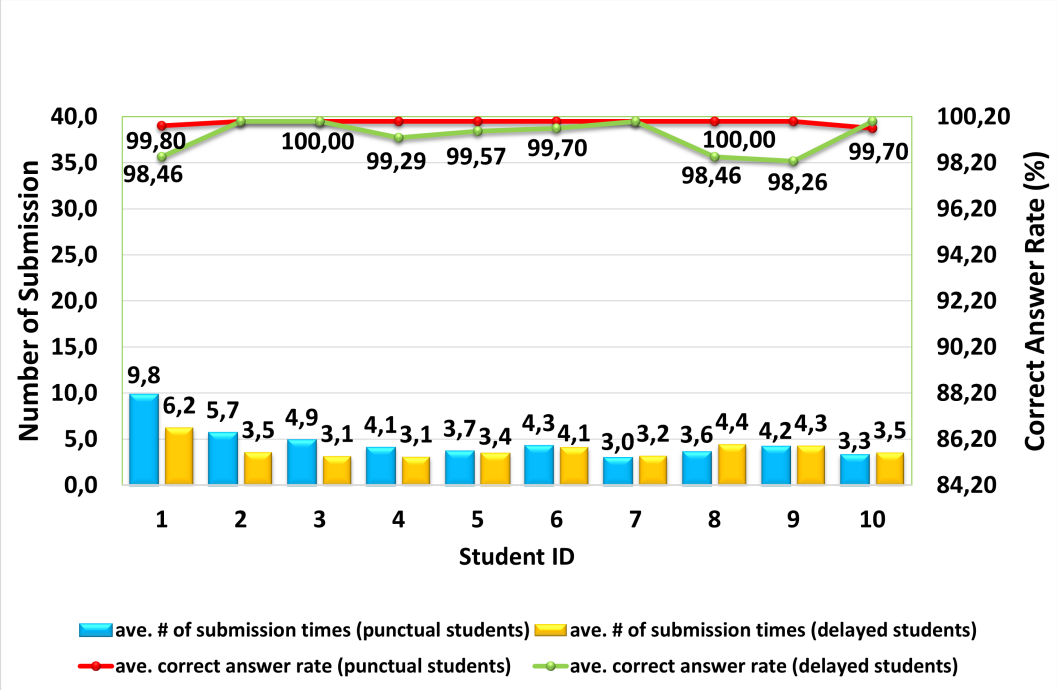

Fig. 10. Individual instance results for *CFP*.

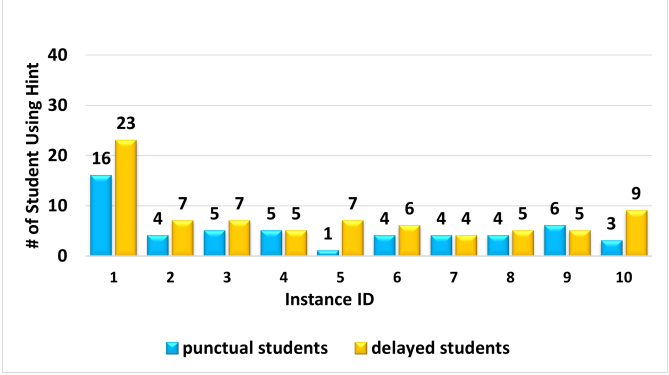

Fig. 11. Hint button click instances.

submissions, and the average correct answer rate for each of *delayed students*. Among the 32 students, 29 (91%) achieved a perfect score of 100%, while three (9%) scored slightly low, ranging from 92.5% to 98.75%. The lowest accuracy of 92.5% was observed in the student at ID=32, who also had the highest number of submissions, 12.2. Most students completed the assignments with seven or less submissions, whereas the students at ID=29, 30, 31, and 32 required notably higher attempts. Overall, *delayed students* maintained high performance, with only minor accuracy reductions.

3) Results of Individual Instances: Figure 14 presents the average correct answer rates and the number of submission times for each instance across both groups. All the instances achieved scores above 97%. In *punctual*

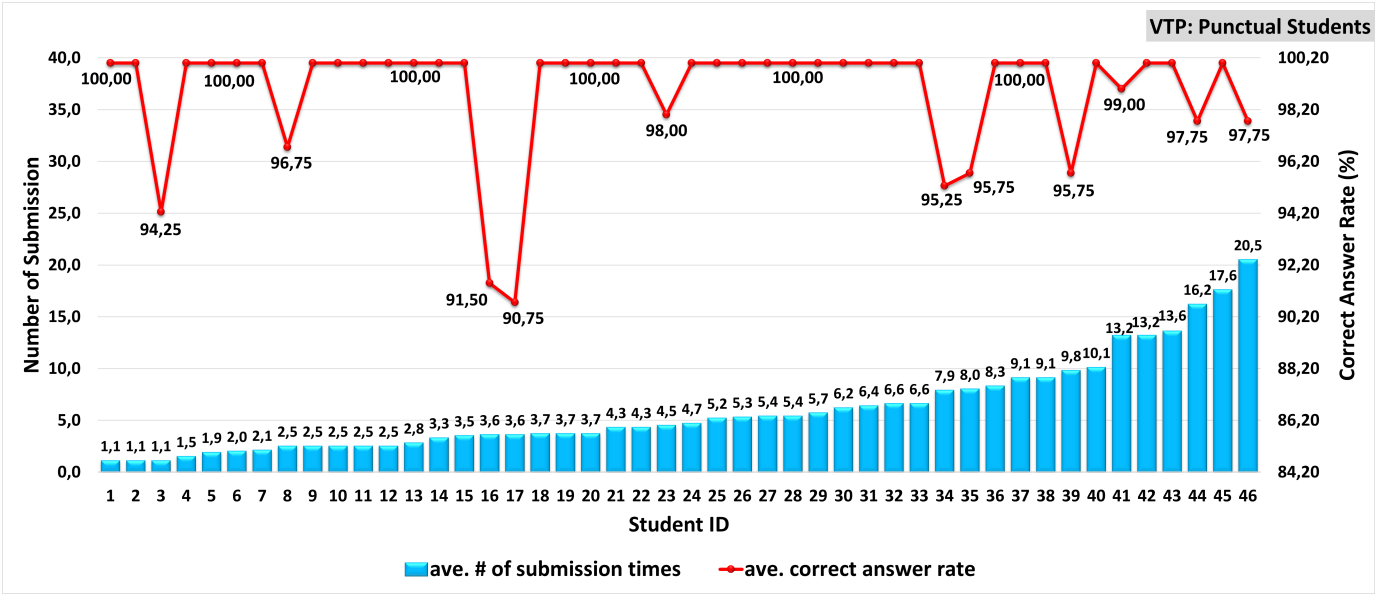

Fig. 12. Individual results for *punctual students* on *VTP*.

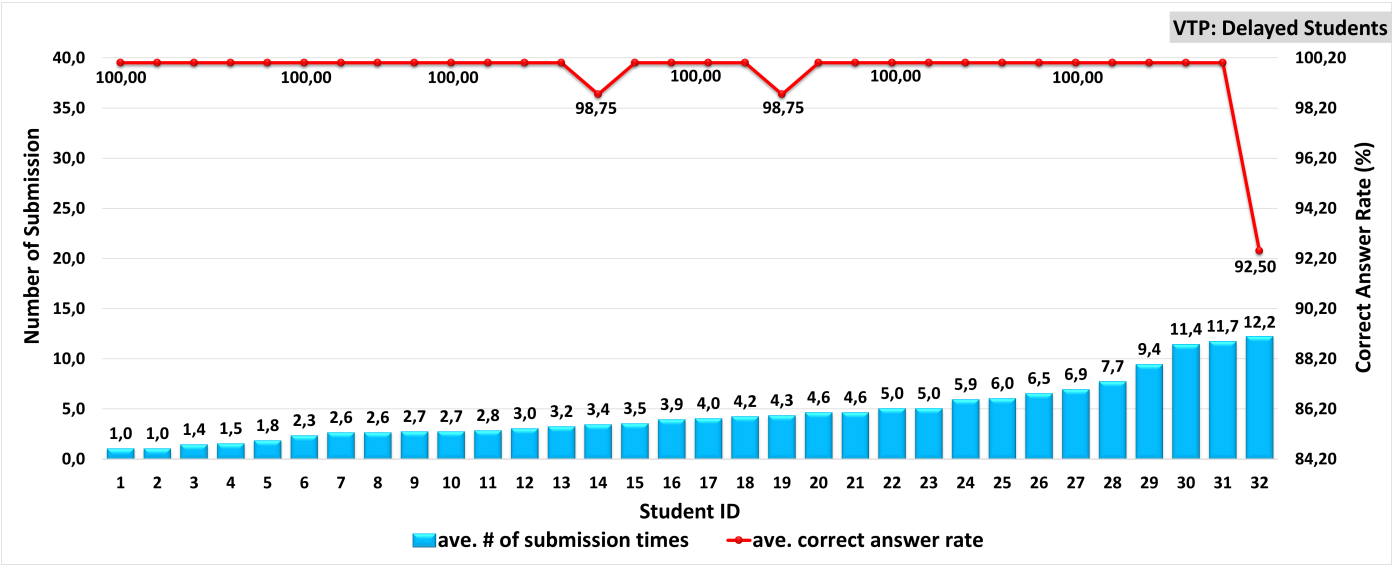

Fig. 13. Individual results for *delayed students* on *VTP*.

students, the instances at ID=2 for *sorting* and 4 for *queue* reached a perfect correct answer rate of 100% with moderate submission counts. The instances at ID=3 for *stack* and 7 for *doubly linked lists* had the highest average submissions (10.4), indicating the higher difficulty. In *delayed students*, the overall performance improved, with six instances attaining 100% accuracy and fewer submissions, suggesting enhanced understanding and efficiency.

VII. Discussions

In this section, we discuss the solution results of the *CTP*, *CFP*, and *VTP* instances to novice students, to highlight key findings, analyse implications, and note potential improvements.

A. Individual Problem Performance Comparison between Student Groups

First, we compare the solution performances of *punctual* and *delayed* students for (*CTP*, *CFP*, and *VTP*), in terms of the number of answer submissions, and the correct answer rate.

1) Comparison of Answer Submissions: Table VI presents the *t-test* results comparing the average number of answer submissions between *punctual* and *delayed* students across *CTP*, *CFP*, and *VTP*. In this table, *N* indicates the number of students, *M* does the mean, and *SD* does the standard deviation.

For the *CTP* instances, the average number of submissions by *punctual students* was 2.455, and the standard deviation was 0.897. They were slightly lower than those by *delayed students*, 2.722 and 1.053. However, the *p-value* of *0.255* indicates that this difference is not statistically significant.

For the *CFP* instances, *punctual students* submitted answers more frequently with an average of 4.643 submissions with a standard deviation of 4.036. *Delayed students* did them with an average of 3.881 submissions with a standard deviation of 3.775. However, the *p-value* of *0.41* suggests that the variation is not statistically significant.

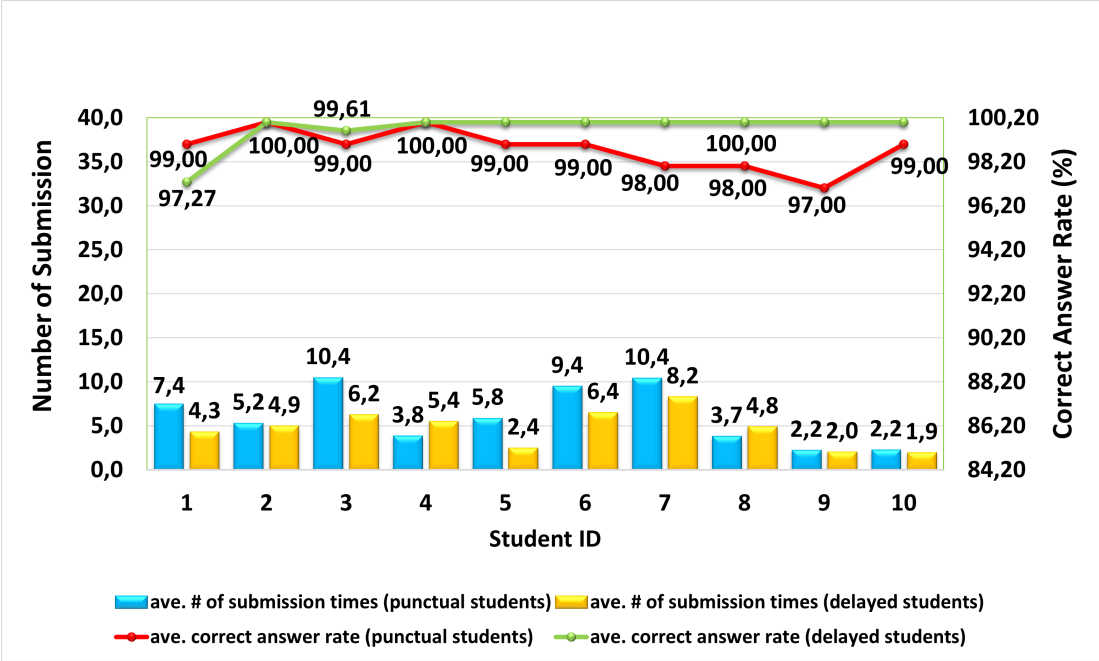

Fig. 14. Individual instance results for *VTP*.

Similarly, for the *VTP* instances, *punctual students* had a higher average number of submissions of 6.052 with a standard deviation of 4.568. *Delayed students* had an average of 4.650 submissions with a standard deviation of 3.035. The *p-value* of *0.107* remains above the 0.05 threshold, indicating no statistically significant difference between the two groups.

Overall, *punctual students* tend to submit more answers than *delayed students* across all instance types, but the differences between them are not statistically significant.

TABLE VI
COMPARISON OF AVERAGE NUMBER OF SUBMISSIONS BETWEEN *punctual* AND *delayed students*.

problem	*punctual students*			*delayed students*		
type	N	M	SD	N	M	SD
CTP	40	2.455	0.897	33	2.722	1.053
CFP	40	4.643	4.036	33	3.881	3.775
VTP	46	6.052	4.568	32	4.650	3.035

2) Comparison of Correct Rates: Table VII presents the *t-test* results for comparing the correct answer rates between *punctual* and *delayed students* across *CTP*, *CFP*, and *VTP* instances.

In the *CTP* instances, *punctual students* achieved a slightly higher correct answer rate of 0.999 and a smaller standard deviation of 0.002. *Delayed students* achieved 0.994 and 0.017 for them, respectively. However, the *p-value* of *0.059* is just above the 0.05 threshold, indicating that the difference is not statistically significant.

In the *CFP* instances, *punctual students* performed slightly better, with a mean accuracy of 0.999 and a standard deviation of 0.002. They are compared to 0.995 with a standard deviation of 0.010 for *delayed students*. The *p-value* of *0.009* indicates that this difference is statistically significant.

For the *VTP* instances, *delayed students* achieved a higher mean accuracy of 0.997 with a standard deviation of 0.013. *Punctual students* did 0.990 with the standard deviation of 0.022. However, the *p-value* of *0.081* indicates that the difference is not statistically significant.

Overall, *punctual students* generally showed slightly better performances in the *CTP* and *CFP* instances, and *delayed students* slightly outperformed in the *VTP* instances, where only the difference in the *CFP* instance was statistically significant. The results suggest that the solution accuracy between the two groups is generally comparable across the three exercise problems.

TABLE VII
COMPARISON OF AVERAGE CORRECT ANSWER RATES BETWEEN *punctual* AND *delayed students*.

problem	*punctual students*			*delayed students*		
type	N	M	SD	N	M	SD
CTP	40	0.999	0.002	33	0.994	0.017
CFP	40	0.999	0.002	33	0.995	0.010
VTP	46	0.990	0.022	32	0.997	0.013

B. Overall Problem Performance Comparison between Student Groups

Second, we compare the solution performances between the *punctual* and *delayed* students across the three problems, in terms of the number of the answer submissions, the correct answer rate, and the *time span*. The *time span* refers to the total duration between the first and last recorded submission timing for all the exercises by each student. The exercise problems in *JPLAS* given to students are open-ended, accessed anytime and anywhere outside of scheduled class hours. The *time span* is the difference between the earliest and latest submission timing. This can be affected by other course works or off-campus activities.

Table VIII presents the aggregate *t-test* results comparing the performance metrics. For the average number of

submissions, *punctual students* had an average of 4.462 and a standard deviation of 3.883. *Delayed students* had an average of 3.742 with a standard deviation of 2.940. The *p-value* was *0.115*, which is above the standard threshold of 0.05, indicating no statistically significant difference between the two groups.

For the correct answer rate, *punctual students* had an average of 0.014 with a standard deviation of 0.002, while *delayed students* had an average of 0.995 with a standard deviation of 0.013. The *p-value* was *0.639*, which is also above the standard threshold of 0.05, indicating no statistically significant difference.

For the *time span*, *punctual students* had an average of 1267.1 minutes with a standard deviation of 1694.1. *Delayed students* had an average of 3189.8 minutes with a standard deviation of 6862.6. The *p-value* was *0.0084*, indicating a statistically significant difference. Further analysis yielded a *t-value* of *–2.69* and a *Cohen's d* of *–0.40*, suggesting a small to moderate effect size. These results indicate that *punctual students* consistently completed the exercises with a shorter *time span* than *delayed students*. Figure 15 illustrates a box plot comparing the *time span* distributions between the two groups to visualize this difference.

These results suggest that both group students demonstrated similar performances. The only notable difference was the *time span* to complete the exercise problems. This finding indicates the importance of encouraging students to complete assignments on time. Since the tasks were given as optional ones and did not affect the final grades in this time, many students may have lacked motivations to complete them on time. To improve their engagements, we will consider using the exercise results in final grades evaluations.

TABLE VIII
AGGREGATE COMPARISON OF PERFORMANCE METRICS BETWEEN *punctual* AND *delayed students* ACROSS ALL PROBLEMS.

metric	*punctual students*			*delayed students*		
	N	**M**	**SD**	**N**	**M**	**SD**
ave. submissions	126	4.462	3.883	98	3.742	2.940
ave. correct rate	126	0.014	0.002	98	0.995	0.013
time span (min)	126	1267.1	1694.1	98	3189.8	6862.6

C. Distribution Analysis of Student Performance

Third, we analyse the distribution of student performances across the three exercise instances, focusing on submission frequencies and correct answer rates.

1) Distribution of Average Number of Submissions: Tables IX and X present the distribution of submission times in three exercise instances by *punctual students* and *delayed students*, respectively. Clearly, many students submitted answers within a small range (1–6 attempts) for *CTP*. From *punctual students*, 100% answers were submitted quickly for *CTP*, while only 67.39% did so for *VTP*. From *delayed* students, all of them submitted quickly for *CTP*, while only 78% did so for *VTP*. This suggests that *VTP* was the most challenging one, since it requires a deeper understanding of the semantic behaviours of a *Java* source code.

TABLE IX
COMPARISON OF SUBMISSION TIMES FOR *three problems* BY *punctual students*.

range of	# of students			rate of students (%)		
submission time	***CTP***	***CFP***	***VTP***	***CTP***	***CFP***	***VTP***
1-6	40	32	31	100	80.00	67.39
7-12	0	7	9	0	17.50	19.57
13-18	0	0	5	0	0	10.87
19-25	0	1	1	0	2.50	2.17

TABLE X
COMPARISON OF SUBMISSION TIMES FOR *three problems* BY *delayed students*.

range of	# of students			rate of students (%)		
submission time	***CTP***	***CFP***	***VTP***	***CTP***	***CFP***	***VTP***
1-6	33	30	25	100	90.00	78.00
7-12	0	2	7	0	6.00	22.00
13-18	0	0	0	0	0	0
19-25	0	1	0	0	3.00	0

2) Distribution of Average Correct Answer Rates: Tables XI and XII compare the correct answer rates across the three problems for *punctual* and *delayed students*, respectively. Most students showed a high correct rate on *CTP* and *CFP*. Actually, 97.5% of *punctual students* scored 100% on *CTP*. This rate dropped to 76.08% on *VTP*. This drop was also observed in *delayed students*, showing that *VTP* is more challenging for getting correct answers.

TABLE XI
COMPARISON OF CORRECT ANSWER RATES FOR *three problems* BY *punctual students*.

range of correct	# of students			rate of students (%)		
answer rate (%)	***CTP***	***CFP***	***VTP***	***CTP***	***CFP***	***VTP***
85-90	0	2	0	0	5.00	0
91-95	0	0	4	0	0	8.69
96-99	1	0	7	2.50	0	15.21
100	39	38	35	97.50	95.00	76.08

TABLE XII
COMPARISON OF CORRECT ANSWER RATES FOR *three problems* BY *delayed students*.

range of correct	# of students			rate of students (%)		
answer rate (%)	***CTP***	***CFP***	***VTP***	***CTP***	***CFP***	***VTP***
85-90	0	0	0	0	0	0
91-95	3	0	1	9.10	0	3.00
96-99	0	11	2	0	33.00	6.00
100	30	22	29	90.90	67.00	91.00

D. Observation of Common Mistakes

Now, we observe common mistakes among students appearing in the three problems.

1) CTP *Mistakes:* Several common mistakes occurred in both student groups when students worked on *CTP* instances, primarily related to syntax errors. Frequent errors appeared in using the lowercase letter for a class name, capitalizing the first letter of a modifier, and incorrectly spelling *Java* keywords, variables, functions, and parameters.

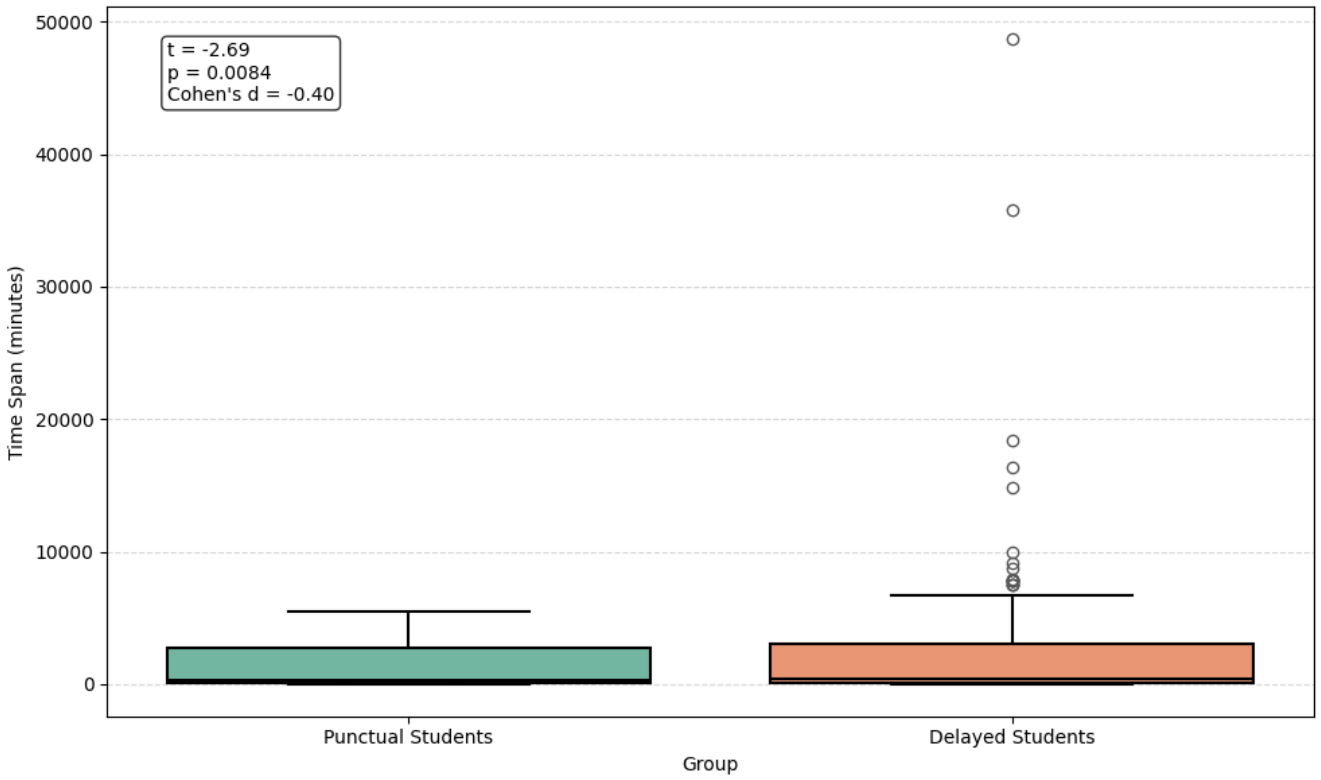

Fig. 15. Comparison of *time span* between *punctual* and *delayed students*.

2) CFP *Mistakes:* The most common errors stemmed from a lack of understanding of the *Java* grammar, such as mistakenly writing "class" as "interface," "new" as "New," and "for" as "For." Other mistakes came from simple typographical errors due to a lack of attention. Overall, the types of errors by students were similar in both student groups. *Punctual students* tended to show more efforts in solving the problems, which was reflected in their marginally higher accuracy rates. Meanwhile, *delayed students* demonstrated a variation of programming abilities, where a few students appeared less motivated.

3) VTP *Mistakes:* Students often faced issues related to logical errors, misinterpretations of questions, or difficulties in understanding English instructions. For example, some students struggled to grasp the concepts of *push* and *pop* in *stack*, leading to incorrect responses. The most frequent mistake involved flawed logic. Some students confused the functions of *add()*, *remove()*, *peek()*, and *print()*, leading to incorrect implementations.

E. Student Feedback Qualitative Response

Table XIII shows the percentages of positive, neutral, or negative opinions among the students on the three problems. Most students provided the positive feedback, where *CTP* received the highest approval of 80% from the *punctual students* and 90% from the *delayed students*.

Most students provided the positive feedback. *CFP* received the highest approval rate of 83% from *punctual students* and 75% from *delayed students*. They provided a lot of positive feedback for *VTP*, with the approval rate of 76% from *punctual students* and 77% from *delayed students*. Negative responses came from no more than 15% of them, indicating a generally good reception. These results suggest that students found the proposed three exercise problems useful.

TABLE XIII
USER FEEDBACK ON PROBLEMS.

problem	***punctual students***			***delayed students***		
type	**positive**	**neutral**	**negative**	**positive**	**neutral**	**negative**
CTP	80%	5%	15%	90%	1%	9%
CFP	83%	2%	15%	75%	10%	15%
VTP	76%	9%	15%	77%	8%	15%

F. Student Feedback Based on System Usability Scale

The *System Usability Scale (SUS)* questionnaire was administered to the students after the exercises to assess their perception of the system's usability. Table XIV lists the questions, where each is answered with a five-point *Likert* scale. Table XV explains the meaning of each scale point.

The *SUS* score was calculated as follows:

1) subtract 1 from each odd-numbered question score.
2) subtract each even-numbered item's score from 5.
3) sum all the adjusted scores.
4) multiply the total by 2.5.

The *SUS* score can be interpreted in two ways. The first one evaluates the acceptability with three categories, "not acceptable", "marginal", and "acceptable", as shown in Table XVI. The second one uses five grades (A–E), as shown in Table XVII.

Among the 73 students who completed the *CTP* and *CFP* instances, 33 responded to the *SUS* questionnaire. Thus, this usability analysis may introduce a response bias. Table XVIII

TABLE XIV
System usability scale (SUS) QUESTIONNAIRE.

no	question	score
1	I think that I would like to use this system frequently.	1 - 5
2	I found the system unnecessarily complex.	1 - 5
3	I thought the system was easy to use.	1 - 5
4	I think that I would need the support of a technical person to be able to use this system.	1 - 5
5	I found the various functions in this system were well integrated.	1 - 5
6	I thought there was too much inconsistency in this system.	1 - 5
7	I would imagine that most people would learn to use this system very quickly.	1 - 5
8	I found the system very cumbersome to use.	1 - 5
9	I felt very confident using the system.	1 - 5
10	I needed to learn a lot of things before I could get going with this system.	1 - 5

TABLE XV
SUS SCORE RESPONSE.

response	scale point
Strongly Disagree (SD)	1
Disagree (D)	2
Neutral (D)	3
Agree (A)	4
Strongly Agree (SA)	5

TABLE XVI
SUS ACCEPTABILITY RANGE.

range	note
0 - 50.9	Not acceptable
51 - 70.9	Marginal
71 - 100	Acceptable

TABLE XVII
SUS ACCEPTABILITY GRADE.

grade	range
A	Score >= 80.3
B	Score >= 74 and <80.3
C	Score >= 68 and <74
D	Score >= 51 and <68
E	Score <51

presents the *SUS* score results. The final average *SUS* score is 75, which indicates *"Acceptable"* for acceptability range and *Grade B* for acceptability grade.

Similarly, among the 78 students who completed the *VTP* instances, 32 responded to the *SUS* questionnaire. Table XIX shows the *SUS* results. The final recorded average *SUS* score was 74.7, which again indicates *"Acceptable"* for acceptability range and *Grade B* for acceptability grade.

These results suggest that the proposal meets user expectations in terms of usability. The high *SUS* scores reinforce the effectiveness of the interfaces and indicate satisfactory user experiences.

G. Comparison of Final Exam Scores

To further evaluate the effectiveness of the proposed three exercise problems by the novice students at State Polytechnic of Malang, Indonesia, we compared the final exam scores between the students who completed all the instances, called *solved students*, and the students who did not, called *non-solved students*. It is noted that they are different from the previous student groups of *punctual* and *delayed*.

In the *basic programming* course, *solved students* consist of 73 students from 5 classes, while *non-solved students* consist of 350 students from 9 classes. Similarly, in the *algorithm and data structure* course, *solved students* consist of 78 students from 5 classes, whereas *non-solved students* consist of 350 students from 9 classes.

Table XX presents the average and the standard deviation of the final exam scores for both *solved* and *non-solved* students in each course. An independent samples *t-test* was conducted to assess the statistical significance of the differences. In the *basic programming* course, the *p-value* is *1.48E-17*, which is well below the standard threshold of 0.05. This leads to the rejection of the null hypothesis and indicates a statistically significant difference, with *solved* students performing better. In the *algorithm and data structure* course, the *p-value* of *0.043* also leads to the rejection of the null hypothesis and suggests a significant difference. To evaluate the practical impact of these differences, *Cohen's d* was calculated for the *basic programming* course, yielding an effect size of *0.85*, which exceeds the 0.8 threshold and thus indicates a large effect. For the *algorithm and data structure* course, the effect size was *0.28*, this indicates a small effect size.

These findings confirm that *solved students* performed better academically compared to *non-solved students*, suggesting the proposal positively contributed to student learning outcomes.

VIII. CONCLUSIONS

This paper presented a study of three exercise problems in the *Java Programming Learning Assistant System (JPLAS)* that are dedicated to novice students. They include the *code typing problem (CTP)*, the *code fixing problem (CFP)*, and the *value trace problem (VTP)*. For their evaluations, we assigned the generated instances to first-year undergraduate students at the State Polytechnic of Malang, Indonesia. Most students correctly solved all the questions despite a significant difference in the *time span* for completing the exercises, and provided positive feedback on them by rating them in the acceptable category and a grade B in the *SUS* testing. The students who completed the instances outperformed the final exam scores of those who did not. Thus, the effectiveness and validity of the proposal are confirmed. In future works, we will generate new instances on advanced topics to enhance logical thinking and programming skills and assign them to students for evaluations.

TABLE XVIII
SUS SCORES BY STUDENTS SOLVING *CTP* AND *CFP*.

respondent	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	*SUS* score
1	4	3	4	4	5	3	5	3	3	3	62.5
2	4	2	4	1	5	2	5	1	5	1	90.0
3	3	2	4	4	5	3	3	3	4	5	55.0
4	5	1	5	2	5	1	5	1	4	2	92.5
5	4	2	4	3	5	2	5	2	5	2	80.0
6	5	1	5	1	5	1	5	1	5	1	100.0
7	3	4	3	3	3	3	4	2	3	3	52.5
8	4	2	2	2	3	3	4	1	4	2	67.5
9	5	1	5	1	5	1	5	1	5	1	100.0
10	4	2	4	2	4	2	4	2	4	2	75.0
11	5	1	5	1	5	1	5	1	5	1	100.0
12	3	3	3	3	4	3	3	3	3	3	52.5
13	4	3	4	4	4	3	4	2	4	4	60.0
14	4	4	4	4	5	3	4	3	5	4	60.0
15	4	4	3	4	4	3	4	3	3	3	52.5
16	5	3	5	3	5	3	3	3	4	3	67.5
17	5	2	5	3	4	2	4	2	5	4	75.0
18	4	3	3	5	4	4	5	4	5	4	52.5
19	4	2	4	1	4	2	5	2	4	2	80.0
20	3	1	5	1	5	1	5	1	5	1	95.0
21	5	2	5	4	5	3	5	4	5	3	72.5
22	3	3	3	3	3	3	3	3	3	3	50.0
23	4	1	5	2	5	1	4	1	5	3	87.5
24	4	2	5	5	4	1	5	1	4	1	80.0
25	4	2	3	3	4	3	4	2	4	2	67.5
26	5	4	4	2	4	3	4	4	5	3	65.0
27	5	3	5	1	4	2	5	1	5	3	85.0
28	4	2	4	1	4	2	5	1	4	1	85.0
29	4	2	2	2	3	2	3	3	2	2	57.5
30	5	3	5	3	5	1	5	5	5	2	77.5
31	5	1	5	1	5	1	5	1	5	1	100.0
32	4	1	5	1	1	1	4	1	4	1	82.5
33	5	2	5	1	5	1	5	1	4	1	95.0
average score (final result)											**75.0**

TABLE XIX
SUS SCORES BY STUDENTS SOLVING *VTP*.

respondent	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	*SUS* score
1	5	1	5	1	5	1	5	1	5	1	100.0
2	5	3	5	3	4	3	5	3	5	3	72.5
3	4	2	4	2	4	2	4	2	4	2	75.0
4	5	4	5	3	5	1	5	1	5	3	82.5
5	5	3	5	3	5	1	5	1	5	1	90.0
6	3	3	3	2	3	3	4	1	5	1	70.0
7	5	1	5	2	5	3	5	1	5	2	90.0
8	3	3	3	3	3	3	3	3	5	1	60.0
9	3	3	3	3	4	3	4	3	3	3	55.0
10	5	1	5	1	5	1	2	1	5	1	92.5
11	3	1	5	1	3	1	4	1	4	1	85.0
12	4	3	3	1	3	5	3	2	3	2	57.5
13	5	1	5	2	5	1	5	4	5	1	90.0
14	5	1	5	1	5	1	5	3	3	2	87.5
15	3	2	3	3	5	1	5	2	5	1	80.0
16	5	1	5	3	5	1	5	3	5	1	90.0
17	5	3	5	3	5	1	5	3	5	1	85.0
18	5	1	5	1	5	1	5	1	5	1	100.0
19	4	4	4	4	4	4	4	4	4	4	50.0
20	3	3	3	3	3	3	3	3	3	3	50.0
21	3	3	4	3	4	3	4	3	4	3	60.0
22	3	5	3	2	5	1	3	1	5	1	72.5
23	5	1	5	3	5	1	5	3	5	1	90.0
24	4	4	4	3	4	5	5	2	4	5	55.0
25	5	1	3	1	5	1	4	1	5	2	90.0
26	5	4	4	1	4	2	4	1	4	1	80.0
27	4	2	4	2	4	4	4	1	4	1	75.0
28	3	2	2	3	3	3	3	3	3	1	55.0
29	3	3	3	3	3	3	3	3	3	1	55.0
30	3	3	4	2	3	2	4	2	4	2	67.5
31	4	5	3	5	4	1	5	2	3	5	52.5
32	4	2	4	2	4	2	4	2	4	2	75.0
average score (final result)											**74.7**

TABLE XX
STUDENT FINAL EXAM SCORES.

student	basic programming			algo. & data structure		
groups	N	M	SD	N	M	SD
Solved	73	76.88	9.24	78	69.37	23.24
Non-Solved	350	63.66	16.53	350	63.58	19.65

REFERENCES

[1] N. Singh, S. S. Chouhan, and K. Verma, "Object oriented programming: concepts, limitations and application trends," Proceedings of 5th Int. Conf. Information Systems and Computer Networks (ISCON), Mathura, India, 2021, pp1–4. https://doi.org/10.1109/ISCON52037.2021.9702463

[2] H. Winkelmann and H. Kuchen, "Constraint-logic object-oriented programming on the Java virtual machine," Proceedings of 37th ACM/SIGAPP Symp. Applied Computing (SAC '22), Virtual Event, 2022, pp1258–1267. https://doi.org/10.1145/3477314.3507058

[3] P. Vats, Z. Aalam, S. Kaur, A. Kaur, and S. Kaur, "A multi-factorial code coverage based test case selection and prioritization for object oriented programs," ICT Systems and Sustainability, M. Tuba, S. Akashe, and A. Joshi, Eds., Singapore: Springer, 2021, pp721–731. https://doi.org/10.1007/978-981-15-8289-9_69

[4] D. I. De Silva, P. T. Jayasinghe, A. T. Illesinghe, D. E. H. Mallawaarachchi, C. S. Vithanage, and S. Vidhanaarachchi, "CodeRookie: educational Java programming environment for beginners," Lecture Notes in Networks and Systems. Springer Nature: Proceedings of Ninth International Congress Information and Communication Technology, X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds., vol. 1013, 2024, Singapore., pp243–253. https://doi.org/10.1007/978-981-97-3559-4_19

[5] D. Xu, F. Liu, B. Wang, X. Tang, D. Zeng, H. Gao, R. Chen, and Q. Wu, "GenesisRM: A state-driven approach to resource management for distributed JVM web applications," Future Generation Computer Systems, vol. 163, Art. no. 107539, Feb. 2025. https://doi.org/10.1016/j.future.2024.107539

[6] E. Marevac, E. Kadušić, N. Živić, N. Buzaija, and S. Lemeš, "Framework design for the dynamic reconfiguration of IoT-enabled embedded systems and "on-the-fly" code execution," Future Internet, vol. 17, no. 1, Art. no. 23, 2025. https://doi.org/10.3390/fi17010023

[7] S. Barakat, A. Martin-Lopez, C. Müller, S. Segura, and A. Ruiz-Cortés, "The IDL tool suite: specifying and analyzing inter-parameter dependencies in web APIs," SoftwareX, vol. 29, Art. no. 101998, Feb. 2025. https://doi.org/10.1016/j.softx.2024.101998

[8] N. Funabiki, Y. Matsushima, T. Nakanishi, K. Watanabe, and N. Amano, "A Java programming learning assistant system using test-driven development method," IAENG International Journal of Computer Science, vol. 40, no.1, pp38–46, 2013.

[9] K. H. Wai, N. Funabiki, S. T. Aung, R. Hashimoto, D. Yokoyama, and W.-C. Kao, "Analysis of solution results of code writing problems for basic object-oriented programming study in university Java programming course," Proceedings of International Conference Information and Education Technology (ICIET), pp87–92, March 2024. https://doi.org/10.1109/ICIET60671.2024.10542814

[10] S. T. Aung, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N. K. Dim, and W.-C. Kao, "A proposal of grammar-concept understanding problem in Java programming learning assistant system," Journal of Advances in Information Technology, vol. 12, no. 4, Nov. 2021. https://doi.org/10.12720/jait.12.4.342-350

[11] N. Funabiki, K. K. Zaw, M. Kuribayashi, and W.-C. Kao, "Value trace problems for graph theory algorithms in Java programming learning assistant System," International Journal of Information and Education Technology, vol. 7, no. 5, May. 2017. https://doi.org/10.18178/ijiet.2017.7.5.897

[12] Y. Jing, N. Funabiki, S. T. Aung, X. Lu, A. A. Puspitasari, H. H. S. Kyaw, and W.-C. Kao, "A proposal of mistake correction problem for debugging study in C programming learning assistant system," International Journal of Information and Education Technology, vol. 12, pp1158–1163, 2022. https://doi.org/10.18178/ijiet.2022.12.11.1733

[13] N. Funabiki, Tana, K. K. Zaw, N. Ishihara, and W.-C. Kao, "A graph-based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system," IAENG International Journal of Computer Science, vol. 44, no. 2, pp247–260, 2017.

[14] H. H. S. Kyaw, S. S. Wint, N. Funabiki, and W.-C. Kao, "A code completion problem in Java programming learning assistant system," IAENG International Journal of Computer Science, vol. 47, no. 3, pp350–359, 2020.

[15] X. Lu, N. Funabiki, K. H. Wai, S. T. Aung, M. Mentari, and W.-C. Kao, "An implementation of phrase fill-in-blank problem for test code reading study in Java programming learning assistant system," Proceedings of 13th International Conference Advances in Information Technology (IAIT '23), Bangkok, Thailand, 2023, Art. no. 38, pp1–5. https://doi.org/10.1145/3628454.3631856

[16] H. Fang, Y. Cai, E. Tempero, R. Kazman, Y.-C. Tu, J. Lefever, and E. Pisch, "A holistic approach to design understanding through concept explanation," IEEE Transactions on Software Engineering, vol. 51, no. 2, pp449–465, Feb. 2025. https://doi.org/10.1109/TSE.2024.3522973

[17] I. Riouak, N. Fors, G. Hedin, and C. Reichenbach, "IntraJ: an on-demand framework for intraprocedural Java code analysis," International Journal on Software Tools Technology Transfer, vol. 26, no. 6, pp687–705, Dec. 2024. https://doi.org/10.1007/s10009-024-00771-0

[18] M. Fawad, G. Rasool, and M. B. Riaz, "Refactoring Android source code smells from Android applications," IEEE Access, vol. 13, pp14122–14150, 2025. https://doi.org/10.1109/ACCESS.2025.3529687

[19] K. Lano, Q. Xue, and H. Haughton, "A concrete syntax transformation approach for software language processing," SN Computer Science, vol. 5, no. 645, June 2024. https://doi.org/10.1007/s42979-024-02979-y

[20] M. Mentari, N. Funabiki, Noprianto, Y. W. Syaifudin, K. H. Wai, K. C. Brata, and P. Puspitaningayu, "A study of code typing problems as start-up programming practices in Java programming learning assistant system," Proceedings of 5th International Conference on Information Technology and Education Technology (ITET), 2024, pp45–50. https://doi.org/10.1109/ITET64267.2024.00017

[21] Y. Zhang, R. Liang, Y. Li, and G. Zhao, "Improving Java learning outcome with interactive visual tools in higher education," Lecture Notes on Data Engineering and Communications Technologies: Artificial Intelligence in Education: Emerging Technologies, Models and Applications, E. C. K. Cheng, R. B. Koul, T. Wang, and X. Yu, Eds., AIET 2021, Mar. 2022, Singapore: Springer., vol. 104, pp183–195. https://doi.org/10.1007/978-981-16-7527-0_17

[22] F. Dobslaw, K. Angelin, L.-M. Öberg, and A. Ahmad, "The gap between higher education and the software industry — a case study on technology differences," Proceedings of 5th European Conf. Software Engineering Education (ECSEE), Seeon/Bavaria, Germany, 2023, pp11–21. https://doi.org/10.1145/3593663.3593690

[23] G. Alfarsi, R. Tawafak, S. Malik, and B. H. Khudayer, "Facilitation for undergraduate college students to learn Java language using E-learning model," International Journal of Interactive Mobile Technologies (iJIM), vol. 16, no. 8, pp4–17, 2022. https://doi.org/10.3991/ijim.v16i08.28689

[24] C. M. Kandemir, F. Kalelioglu, dan Y. Gülbahar, "Pedagogy of teaching introductory text-based programming in terms of computational thinking concepts and practices," vol. 29, no. 1, pp29–45, 2021. https://doi.org/10.1002/cae.22374

[25] N. C. C. Brown, P. Weill-Tessier, M. Sekula, A.-L. Costache, and M. Kölling, "Novice use of the Java programming language," ACM Transactions on Computing Education (TOCE), vol. 23, no. 1, article no. 10, pp1–24, Dec. 2022. https://doi.org/10.1145/3551393

[26] L. Belcastro, R. Cantini, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Programming big data analysis: principles and solutions," Journal of Big Data, vol. 9, Art. no. 4, Jan. 2022. https://doi.org/10.1186/s40537-021-00555-2

[27] J. Skalka, M. Drlik, L. Benko, J. Kapusta, J. C. Rodríguez del Pino, E. Smyrnova-Trybulska, A. Stolinska, P. Svec, and P. Turcinek, "Conceptual framework for programming skills development based on microlearning and automated source code evaluation in virtual learning environment," Sustainability, vol. 13, no. 6, Art. no. 3293, 2021. https://doi.org/10.3390/su13063293

[28] J. Francis, "Coding boot camps for refugees," Guide to mobile data analytics in refugee scenarios: the 'data for refugees challenge' study, A. A. Salah, A. Pentland, B. Lepri, and E. Letouzé, Eds. Cham: Springer International Publishing, 2019, pp67–85. https://doi.org/10.1007/978-3-030-12554-7_4

[29] Y. Zhang and Y. Ouyang, "Designing a course of programming language foundations that closely combines practice," Proceedings of. ACM Turing Award Celebration Conference - China (ACM TURC '21), Hefei, China, 2021, pp8–14. https://doi.org/10.1145/3472634.3472637

[30] A. Akkaya and Y. Akpinar, "Experiential serious-game design for development of knowledge of object-oriented programming and computational thinking skills," Computer Science Education, vol. 32, no. 4, pp476–501, 2022. https://doi.org/10.1080/08993408.2022.2044673

[31] H. Liu, "Comparative application of teaching methods in C language and JAVA programming courses," International Journal of New Developments in Education, vol. 6, no. 1, pp126–131, 2024. https://doi.org/10.25236/IJNDE.2024.060122

[32] W. O. Apeanti and D. D. Essel, "Learning computer programming using project-based collaborative learning: Students' experiences, challenges, and outcomes," International Journal of Innovation and Education Research, vol. 9, no. 8, pp191–207, 2021. https://doi.org/10.31686/ijier.vol9.iss8.3278

[33] D. I. De Silva, K. A. S. N. Perera, R. A. H. B. Ranasinghe, B. D. Gunawardena, R. R. A. N. N. Jayawardena, and S. Vidhanaarachch, "CodePedia: Crafting the ultimate Java learning odyssey for novice programmers," Proceedings of 9th International Congress on Information and Communication Technology, Singapore: Springer, 2024, pp55–64. https://doi.org/10.1007/978-981-97-3562-4_5

[34] C. Dong, Y. Jiang, Y. Zhang, Y. Zhang, and H. Liu, "ChatGPT-based test generation for refactoring engines enhanced by feature analysis on examples," Proceedings of 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Los Alamitos, CA, USA, May 2025, pp2714–2725. https://doi.org/10.1109/ICSE55347.2025.00210

[35] K. C. Cababasay, M. J. C. Notarte, G. P. Tan, J. A. Catain, A. P. Valdez, G. T. Salvador, "HanAPP Buhay: Android Java-based mobile commerce application for freelance home service providers and client job seekers," Proceedings of SMART GENCON, 2022, pp1–12. https://doi.org/10.1109/SMARTGENCON56628.2022.10083985

[36] A. Khaoula, L. Mohamed, E. Aya, A. O. Younes, L. M. Driss, O. Mustapha, "EduXgame: gamified learning for secondary education," Software Impacts, vol. 24, Art. no. 100761, 2025. https://doi.org/10.1016/j.simpa.2025.100761

[37] R. Kaliisa, S. López-Pernas, K. Misiejuk, C. Damşa, M. Sobocinski, S. Järvelä, and M. Saqr, "A topical review of research in computer-supported collaborative learning: questions and possibilities," Computers & Education, vol. 228, Art. no. 105246, 2025. https://doi.org/10.1016/j.compedu.2025.105246

[38] Krismadinata, Efan, C. Boudia, J. Jama, and A. Y. Saputra, "Effect of collaborative programming on students achievement learning object-oriented programming course," International Journal of Information and Education Technology, vol. 13, no. 5, pp792–800, 2023. https://doi.org/10.18178/ijiet.2023.13.5.1869

[39] S.-F. Wen, "Context-based support to enhance developers' learning of software security," Education Sciences, vol. 13, no. 7, June 2023. https://doi.org/10.3390/educsci13070631

[40] D. Miedema, T. Taipalus, V. V. Ajanovski, A. Alawini, M. Goodfellow, M. Liut, S. Peltsverger, and T. Young, "Data systems education: curriculum recommendations, course syllabi, and industry needs," Proceedings of 2024 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE 2024), Milan, Italy, 2025, pp95–123. https://doi.org/10.1145/3689187.3709609

[41] E. Dickey, A. Bejarano, and C. Garg, "AI-Lab: A framework for introducing generative artificial intelligence tools in computer programming courses," SN Computer Science, vol. 5, Art. no. 720, 2024. https://doi.org/10.1007/s42979-024-03074-y

[42] N. Bergaoui and S. A. Ghannouchi, "A BPM-based approach for ensuring an agile and adaptive learning process," Smart Learning Environments, vol. 10, Art. no. 40, 2023. https://doi.org/10.1186/s40561-023-00259-5

[43] A. Birillo, M. Tigina, Z. Kurbatova, A. Potriasaeva, I. Vlasov, V. Ovchinnikov, and I. Gerasimov, "Bridging education and development: IDEs as interactive learning platforms," Proceedings of 1st ACM/IEEE Workshop on Integrated Development Environments (IDE '24), Lisbon, Portugal, 2024, pp53–58. https://doi.org/10.1145/3643796.3648454

[44] M. A. Amasha, M. F. Areed, D. Khairy, S. M. Atawy, S. Alkhalaf, and R. A. Abougalala, "Development of a Java-based mobile application for mathematics learning," Education and Information Technologies, vol. 26, no. 1, pp945–964, 2021. https://doi.org/10.1007/s10639-020-10287-0

[45] K. D. Cuervo-Cely, J. J. Ramírez-Echeverry, and F. Restrepo-Calle, "Computer-assisted gamification in a computer programming course: an experience report," Proceedings of 13th International Conference on Education, Research and Innovation (ICERI), 2020, pp6006–6015. https://doi.org/10.21125/iceri.2020.1291

[46] T. M. H. T. Azmi, S. Baharudin, S. Ahmad, and N. M. Diah, "Developing and executing pedagogical virtual game-based learning for Java programming," Proceedings of 2024 IEEE International Conference on Computing (ICOCO), 2024, pp190–195. https://doi.org/10.1109/ICOCO62848.2024.10928230

[47] P. Orvalho, M. Janota, and V. Manquinho, "GitSEED: a Git-backed automated assessment tool for software engineering and programming education," Proceedings of 2024 ACM Virtual Global Computing Education Conference (SIGCSE Virtual 2024), Virtual Event, NC, USA, 2024, pp165–171. https://doi.org/10.1145/3649165.3690106

[48] Y. Wang, "Interactive methods to enhance attention in Java learning for underprivileged children: A case study from Singapore's code for all program," in B. K. Smith and M. Borge, Eds., Learning and Collaboration Technologies, HCII 2025, Lecture Notes in Computer Science, vol. 15808, Cham: Springer, 2025, pp345–354. https://doi.org/10.1007/978-3-031-93746-0_29

[49] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: a literature review," ACM Transactions on Computing Education, vol. 18, no. 1, pp1–24, Oct. 2017. https://doi.org/10.1145/3077618

[50] X. Chang, B. Wang, and B. Hui, "Towards an automatic approach for assessing program competencies," Proceedings of the 12th International Learning Analytics and Knowledge Conference (LAK22), pp119–129. Association for Computing Machinery. https://doi.org/10.1145/3506860.3506875

Mustika Mentari received the B.S. degree in computer science from Brawijaya University, Indonesia, in 2011, and the M.S. degree in informatics from Sepuluh Nopember Institute of Technology, Indonesia, in 2014. In 2015, she joined the State Polytechnic of Malang, Indonesia, as a lecturer. She is currently a doctoral student in the Graduate School of Environmental, Life, Natural Science and Technology at Okayama University, Japan. Her research interests include educational technology, artificial intelligence, and computer vision. She is a student member of IEICE.

Nobuo Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, and the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991 respectively. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. In 2001, he moved to the Department of Information and Communication Systems at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and web application systems. He was a vice president of IEEE Consumer Technology Society in 2023 and 2024. He is a member of IEEE, IEICE, and IPSJ.

Safira Adine Kinari received the B.S. degree in computer engineering from Sepuluh Nopember Institute of Technology, Indonesia, in 2023. She is currently pursuing a M.S. degree in the Graduate School of Environmental, Life, Natural Science and Technology at Okayama University, Japan. Her research interests include educational technology and cross-platform application development. She is a student member of IEICE.

Komang Candra Brata received the B.S. degree in informatics from Brawijaya University, Indonesia, in 2012, and the M.S. degree from the Department of Computer Science and Information Engineering, National Central University, Taiwan, in 2014. He is an associate professor at the Faculty of Computer Science at Brawijaya University, Indonesia. He is currently a doctoral student in the Graduate School of Environmental, Life, Natural Science and Technology at Okayama University, Japan. His research interests include software engineering and information technology, user experience, HCI, intuitive mobile applications, distributed systems, and augmented reality. He is a member of IAENG.

Noprianto received the B.S. degree in informatics engineering from AKAKOM Yogyakarta, Indonesia, in 2011, and the M.S. degree in electrical and information technology engineering from Gadjah Mada University, Indonesia, in 2017. In 2019, he joined the State Polytechnic of Malang, Indonesia, as a lecturer. He is currently a doctoral student in the Graduate School of Environmental, Life, Natural Science and Technology at Okayama University, Japan. His research interests include Internet of Things about air quality.

Yan Watequlis Syaifudin received the B.S. degree in Informatics from Bandung Institute of Technology, Indonesia, in 2003, the M.S. degree in Information Technology from Sepuluh Nopember Institute of Technology, Indonesia, in 2011, and the Ph.D. degree in the Graduate School of Natural Science and Technology at Okayama University, Japan, in 2021. In 2005, he joined the State Polytechnic of Malang, Indonesia, as a lecturer and is currently an associate professor and head of the Applied Informatics Laboratory in the Department of Information Technology. His research interests include learning system and technology, database technology, blockchain, and smart farming. He is the director of Academic Association of Creative Economy and is a member of IEEE, IAENG, and Consumer Technology Society.

Triana Fatmawati received the B.S. degree in informatics engineering from Bandung Institute of Technology, Indonesia, in 2003, and the M.S. degree in industrial engineering and management from Bandung Institute of Technology, Indonesia, in 2008. She began her career as a lecturer in the Information Systems program at the Polytechnic of STMI Jakarta, Indonesia, from 2005 until 2022. In 2023, she joined the State Polytechnic of Malang, Indonesia, as a lecturer. Her research interests include information systems, machine learning, and software engineering.

Indra Dharma Wijaya received the B.S. degree from National Institute of Technology, Malang, Indonesia, in 1997, the M.S. degree in management information technology from Sepuluh Nopember Institute of Technology, Indonesia, in 2006, and the Ph.D. degree from Brawijaya University, Indonesia, in 2025. In 2008, he joined the State Polytechnic of Malang, Indonesia, as a lecturer. His research interests include management information systems, information technology adoption, and information technology in business.