A Study of Python Code Writing Problem and C++ Value Trace Problem for Programming Learning Assistant Systems

September, 2024

Shune Lae Aung

Graduate School of
Natural Science and Technology

(Doctor's Course)
Okayama University

Dissertation submitted to
Graduate School of Natural Science and Technology
of
Okayama University
for
partial fulfillment of the requirements
for the degree of
Doctor of Philosophy.

Written under the supervision of

Professor Nobuo Funabiki

and co-supervised by
Professor Satoshi Denno
and
Professor Yasuyuki Nogami

Okayama University, September 2024.

To Whom It May Concern

We hereby certify that this is a typical copy of the original doctor thesis of
Shune Lae Aung

Signature of

the Supervisor

Prof. Nobuo Funabiki

Seal of

Graduate School of

Natural Science and Technology

Abstract

Nowadays, *computer systems* are used in any organization, infrastructure, and service around the world. Then, *computer programming* is increasing its importance. Actually, *computer programming* is one of the most important subjects for students in universities and professional schools. It also develops the abilities of students in solving practical problems by breaking them down into a series of logical steps, known as algorithms. Many universities and professional schools are offering programming courses to cultivate programming engineers.

To assist self-studies of students, we have developed a web-based *programming learning assistant system (PLAS)*, and implemented the personal answer platform on *Node.js* that will be distributed to students via *Docker*. PLAS offers various programming exercise problems at different learning goals where any answer from a student is automatically marked in the system. The common user interface has been implemented on a web browser, where the marking function was implemented by *JavaScript* that runs on the web browser.

Python programming has gained widespread recognition for its versatility and effectiveness across various domains, including application development, task automation, and data analysis. However, the limited availability of Python programming courses in universities has created a growing demand for self-learning platforms to support novice students in their journey to mastering this language. To assist its self-study, we have studied the *code writing problem (CWP)* in PLAS that requests writing a source code that will pass the given *test code* with *unittest*.

As the first contribution in this thesis, I implemented the web-based CWP answer platform for *Python programming* on *Node.js*, by modifying the one for *Java programming*. *Docker* is adopted to help distributing it to the students. The user interface at the client displays the CWP assignments and accepts the answer code submissions. By running the *test code* on *unittest* at the server, the correctness of the answer code is automatically verified where the result is shown in the interface.

To evaluate the effectiveness of this implementation, I prepared 24 CWP instances for basic and advanced Python programming topics, and assigned them to 20 students in Japan and Indonesia. The answer results of the students indicate that most of them successfully completed the assignments, highlighting the platform's efficacy in supporting novice Python programmers. This emphasizes the significance of the platform in facilitating self-study and skill development in Python programming.

C++ programming has been used in implementing numerous practical application systems due to its high-speed execution capability and small-size codes. However, the limited availability of C++ programming courses in universities has created a growing demand for self-learning platforms to support novice students in studying this language. To assist its self-study, we have studied the *value trace problem (VTP)* in *PLAS* that asks the value of a critical variable or output message in the given source code.

As the second contribution in this thesis, I study the VTP for *C++ programming* in PLAS. *C++ programming* can be the first object-oriented programming language for undergraduate students to

start studying programming concepts and computer architecture. Since many students can struggle in studying C++ programming due to the nature in the formal language, hands-on self-study tools can be effective. In a VTP instance, actual values of important variable or standard output messages in a given source code are questioned, where the correctness of each answer is checked through string matching.

For evaluation, I generated 37 VTP instances for basic grammar concepts using source codes in textbooks or websites for C++ programming, and asked 46 students from Myanmar, Japan and Indonesia universities. The results suggest that most of the students are satisfactory, but some students need cares at early programming study stage.

In future works, I will further enhance the implemented platform for *PLAS* with the more variety of exercise problems, integrate interactive learning resources, and extend its functionality to support other programming languages.

Acknowledgements

It is my great pleasure to express my profound gratitude to those who have supported and encouraged me, making this dissertation possible. Although words may fall short in conveying my deepest appreciation, I would like to extend my heartfelt thanks to those who have been a significant blessing in my life.

Foremost, I owe my deepest gratitude to my supervisor, Professor Nobuo Funabiki, who has supported me throughout my thesis with his patience, motivation, encouragement, enthusiasm, and insightful suggestions. His countless valuable advice and unwavering support from the beginning to the end enabled me to progress in this study and navigate daily life in Japan. Furthermore, he provided precious ideas and thoughtful considerations on how to thrive academically and plan for the future. His motivation and energy were instrumental in completing my research papers and preparing for life's challenges. He offered invaluable guidance on social aspects of life in Japan, creating a warm and safe environment despite being far from my family. Needless to say, this thesis would not have been possible without his guidance and active support.

I am deeply grateful to my co-supervisors, Professor Satoshi Denno and Professor Yasuyuki Nogami, for their continuous support, guidance, thoughtful suggestions, and meticulous proofreading of this work. I also wish to express my sincere gratitude to Associate Professor Minoru Kuribayashi for his valuable insights during my research and his great ideas for composing excellent presentations. I extend my thanks to all the course teachers during my Ph.D. study for enlightening me with their wonderful knowledge and to Ms. Keiko Kawabata for her support with necessary documents and requirements during my study.

I acknowledge the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT) for financially supporting my Ph.D. study, and I am grateful to all my respected teachers at the University of Yangon (UY) for imparting valuable knowledge.

I extend my heartfelt thanks to my friends, especially my Myanmar friends, and all the members of FUNABIKI's Lab who assisted me in this study and shared unforgettable experiences with me. Your support during tough times and the thoughts and experiences shared with me are greatly appreciated.

I am eternally grateful to my beloved family and my soulmates, who are essential to my existence. Without their support and encouragement, surviving life's challenges would not have been easy. They have always supported and motivated me not only in my study but throughout my life. Your support and understanding have given me the strength, inspiration, and peace to share joyful moments and overcome any difficulties. I am truly blessed to have you all.

Shune Lae Aung
Okayama, Japan
September 2024

List of Publications

Journal Paper

1. **Shune Lae Aung**, Nem Khan Dim, Soe Mya Mya Aye, Nobuo Funabiki, and Htoo Htoo Sandi Kyaw, "Investigation of Value Trace Problem for C++ Programming Self-study of Novice Students", International Journal of Information and Education Technology (IJIET), vol. 12, no. 7, pp. 631-636, July 2022.

International Conference Papers

2. **Shune Lae Aung**, Nobuo Funabiki, San Hay Mar Shwe, Soe Thandar Aung, and Wen-Chung Kao, "An implementation of code writing problem platform for Python programming learning using Node.js," 2022 IEEE 4th Global Conference on Consumer Electronics (GCCE 2022), pp. 854-855 (Osaka, Japan, 2022).

3. **Shune Lae Aung**, Nobuo Funabiki, San Hay Mar Shwe, Evianita Dewi Fajrianti, and Sritrusta Sukaridhoto, "An application of code writing problem platform for Python programming learning," 2022 IEEE 11th Global Conference on Consumer Electronics (GCCE 2022), pp. 856-857 (Osaka, Japan, 2022).

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background

Nowadays, computer systems are essential components in any organization, infrastructure, and service around the world. Then, computer programming is a fundamental skill for developing and maintaining these systems. Among the various programming languages, Python and C++ are particularly suitable for beginners. Python is popular for its simplicity and versatility, and is widely used in many fields such as web developments, data science, and machine learning, while C++ is also well known for its reliability and portability, finding extensive use in various applications ranging from web systems to embedded systems.

With the increasing reliance on computer systems, there is a corresponding surge in demand for skilled programming engineers. To meet this demand, numerous universities and professional schools offer programming courses for equipping students with the necessary skills and knowledge. However, many universities struggle in offering comprehensive programming courses due to time and staffing constraints. There is a pressing need for high-quality self-study tools and platforms for programming education.

To address this gap, we developed a web-based *Programming Learning Assistant System (PLAS)* and implemented a personal answer platform on *Node.js* that will be distributed to students via *Docker*. PLAS offers various programming exercise problems to assist student self-studies, addressing different learning goals where student answers are automatically marked. These problems include the *Grammar-Concept Understanding Problem (GUP)* [1], the *Value Trace Problem (VTP)* [2], the *Element Fill-in-Blank Problem (EFP)* [3], *Statement Fill-in-blank Problem (SFP)* [4], the *Code Writing Problem (CWP)* [5], the *Code Amendment Problem (CAP)* [6], the *Code Completion Problem (CCP)* [7], the *Mistake Correction Problem (MCP)* [8]. These problems enable students to engage actively in programming learning, allowing them to read, write and test source code.

The server platform of *PLAS* was originally implemented on *Tomcat* where the application programs were made by Java and JSP [9]. Recently, it was newly implemented as a personal answer platform on *Node.js* under the uniform design using *JavaScript* [10]. This implementation allows JavaScript to be used for application programs on both the server and client sides, and it will be distributed via Docker [11]. Node.js [12] serves as the web application server, allowing JavaScript-based development for both server and client applications. The user interface of the code writing problem platform is dynamically controlled using *EJS*, which simplifies complex syntax structures [13].

Python is widely accepted as a versatile and user-friendly object-oriented programming language, used extensively across industries and academia. Its simplified syntax and emphasis on

readability make it an excellent choice for beginners, facilitating a smoother learning curve [14]. Building upon this success of the PLAS platform, I extended the platform to incorporate CWP for Python programming. The extended platform facilitates students in learning Python source code writing from scratch. Students engage by reading the provided test code, writing their source code, and subsequently testing, modifying, and resubmitting the code if errors occur.

On the other hand, C++ is also significantly important in programming education. Its widespread use in practical applications contrasts with the limited availability of comprehensive courses in universities. To bridge this gap, we focus on the VTP within PLAS, for C++ programming learning. Through solving VTP, students engage with critical variables and output messages, enhancing their understanding through practical exercises and string matching evaluations.

1.2 Contributions

In this thesis, as the first contribution of the thesis, I implement the web based *CWP* platform for *Python* programming learning on *Node.js*. The user interface of this platform is dynamically controlled with *EJS* that can avoid the complex syntax structure. *Docker* is adopted to make students easily install the platform software in their own PCs. The user interface at the client shows the CWP assignments and accepts the answer code submissions from the students. Then, by running the *test code* on *unitest* at the server, the correctness of each answer code is automatically verified and is returned to the interface to be shown.

For evaluations, we prepared the usage manual and requested 20 students from Japan and Indonesian universities to install and use the platform. Then, we collected their feedbacks by 10 questionnaires. Most students were satisfied with this platform, which confirms the validity. However, two students found some difficulty in the installation.

In the second contribution, I explore the extension of the web based CWP platform to *Python* programming learning. First, I collected source codes from websites and textbooks, covering basic grammar concepts in *Python* programming. Then, I manually generated the corresponding test codes to test the source codes so that the correctness of the answer source codes from a student can be validated automatically at the answer platform.

For evaluations, we distributed the generated CWP instances to novice students in Japan and Indonesia, who were asked to use the answer platform to solve them. The results showed that the majority of the students successfully completed the code writing tasks, which confirmed the usefulness and effectiveness of the proposal in supporting self-study for *Python* programming. However, some students are still not familiar with the collection data types of *list*, *dictionary*, and *set*, and feel difficult to handle the *JSON* data type. So, some students will need cares at programming study.

In the third contribution, we study the VTP for C++ programming learning. C++ is important for practical applications due to its speed and efficiency. However, limited university courses have increased the need for self-learning tools. To help beginners, I study VTP for C++ programming, which asks students to find the value of key variables or outputs in source code, emphasizing the selection of source codes that cover basic grammar concepts to support novice learners.

For evaluations, we collected 37 source codes from websites and textbooks for basic grammar concepts in C++ programming, and generated VTP instances manually, after analyzing important variables and outputs messages in the codes. Then, to verify the effectiveness of the generated 37 VTP instances, we assigned 46 students from Myanmar, Japan and Indonesia universities. The results show that among 46 students, 35 students solved all the questions correctly, and only two

students may have the difficulty in solving them. These students need cares from the teacher at this early programming study stage.

In future works, I plan to further improve the PLAS platform by adding a wider range of exercise problems, incorporating interactive learning resources, and extending its support to include additional programming languages. These enhancements will provide more comprehensive learning opportunities and better support for students' self-study needs.

1.3 Contents of This Dissertation

The remaining part of this thesis is organized as follows: In Chapter 2, I review the overview of *Programming Learning Assistant System (PLAS)*. In Chapter 3, I present the implementation of the CWP platform for the *code writing problem (CWP)*. In Chapter 4, I present its application results. In Chapter 5, I investigate the VTP for C++ programming. In Chapter 6, I review relevant works in literature.

Chapter 2

Review of Programming Learning Assistant System

2.1 Overview of PLAS

This chapter reviews the web-based *Programming Learning Assistant System (PLAS)*, focusing on server platform, software architecture, and the various problem types implemented in *PLAS*.

2.1.1 Server Platform

Originally, *JPLAS* was implemented using *JSP* with *Java* 1.6.2 as the web application on a server. It uses *Ubuntu-Linux 10.04* as the operating system running on *VMware* for portability. *Tomcat 6.0.26* serves as the web application server to run *JSP* source codes, a scripting language that embeds *Java* codes within *HTML* codes. *Tomcat* returns the dynamically generated web pages to the client web browser. *MySQL 5.0.27* is used for managing the data in *JPLAS*. Figure 2.1 illustrates the server platform of *JPLAS*.

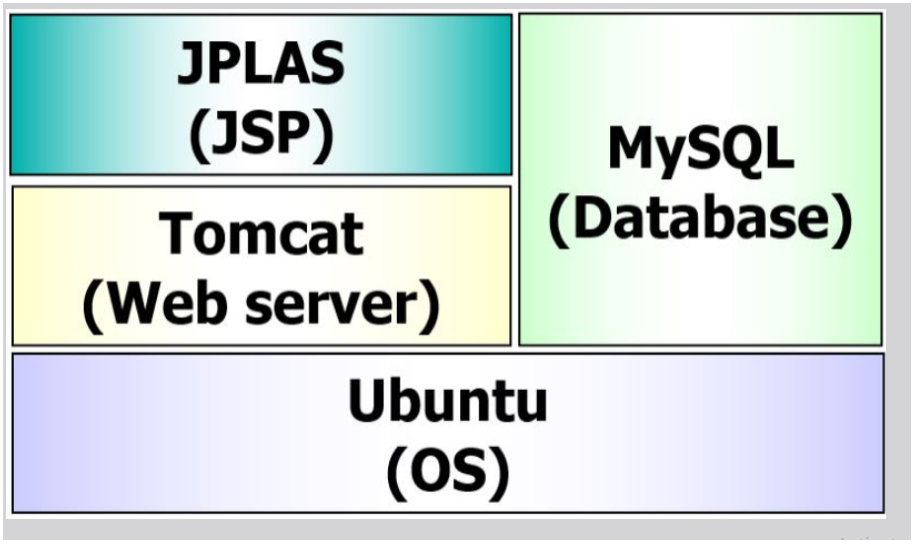

Figure 2.1: Original server platform for JPLAS.

2.1.2 Software Architecture

The software architecture of *JPLAS* follows the *MVC model* as the common architecture of the web application system. It basically uses *Java* for the *Model (M)*, *HTML/CSS/JavaScript* for the *View (V)*, and *JSP* for the *Control (C)*.

The *Model (M)* implements the logic functions of *JPLAS* using *Java*. For the independence from the view and controller, any input/output to/from the model uses a string or its array that does not contain *HTML* tags. *Servlet* is not used to avoid the possible redundancy that could happen between *Java* codes and *Servlet* codes where the same function may be implemented. A design pattern called *Responsibility Chain* is adopted to handle marking functions of the student answers, and the specific functions for the database access are implemented such that the controller does not handle them.

The *View (V)* implements the user interfaces of *JPLAS* by using a *CSS* framework to provide integrated interfaces using *Cascading Style Sheet (CSS)* in the web standard. The user interface is dynamically controlled with *Ajax* to reduce the number of *JSP* files.

The *Control (C)* in *JPLAS* is implemented by *JSP*. When it receives a request from the view, it sends it to the program in the model and requests the corresponding process. When the program in the model returns the processing results by strings, the control program changes the format for *HTML*. The procedure is elaborated as follows:

1) to show the assignment list in the view, *JSP* in the control receives the list with strings in the two dimensional array, changes them into the table format in *HTML*, and sends them to *JavaScript* in the view,

2) to demonstrate the selected assignment in the view, *JSP* receives the details with strings, changes them into the table in *HTML*, and sends it to *JavaScript*, and

3) to mark the answers from the student, *JSP* receives them from *JavaScript* in the view and sends them to *Java* in the model. After completing the marking in the model, *JSP* receives the marking results from *Java*, changes them into the table format in *HTML*, and sends it to *JavaScript* in the view [15].

The overall software architecture in *JPLAS* can be seen in Figure 2.2.

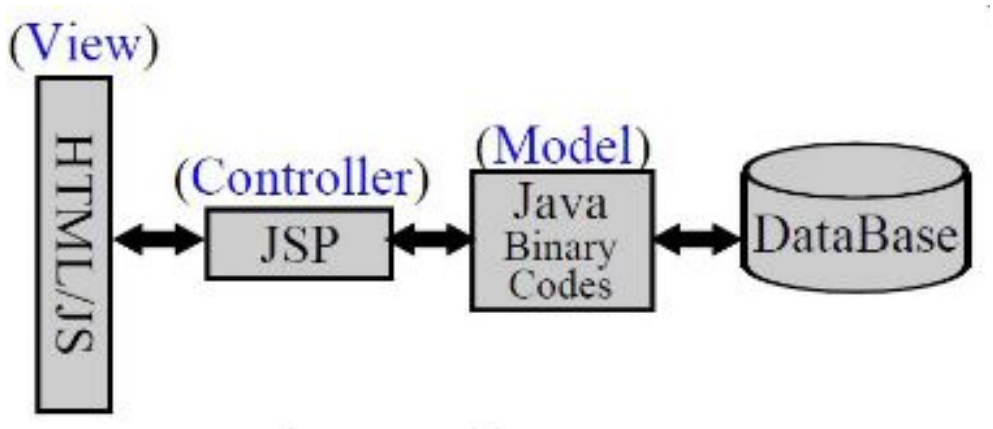

Figure 2.2: Software architecture for JPLAS.

2.1.3 Implemented Problem Types

Currently, *JPLAS* has several types of exercise problems to support students at difference learning levels. The problem types in *JPLAS* are as follows:

1) *Grammar Concept Understanding Problem (GUP)*: This problem requires students to understand the important elements in a source code. An GUP instance consists of a source code and a set of questions on grammar concepts or behaviors of the code. Each answer can be a number, a word, or a short sentence, whose correctness is marked through string matching with the correct one. The algorithm is implemented to automatically generate a *GUP* instance from a given source code by 1) extracting the registered keywords in the source code, 2) selecting the registered question corresponding to each extracted keyword, and 3) detecting the data required in the correct answer from the code [1].

2) *Value Trace Problem (VTP)*: This problem requires students to trace the actual values of important variables in a code when it is executed. The correctness of the answers is also marked by comparing them with their correct ones stored in the server [2].

3) *Element Fill-in-blank Problem (EFP)*: This problem requires students to fill in the blank elements in a given code. The correctness of the answers is marked by comparing them with their original elements in the code that are stored in the server. The original elements are expected to be the unique correct answers for the blanks [3]. To help a teacher to generate a feasible element fill-in-blank problem, the blank element selection algorithm has been proposed [16].

4) *Statement Fill-in-blank Problem (SFP)*: This problem asks students to fill in the blank statements in a code. The correctness of the code is marked by using the test code on *JUnit* that is an open source software for the *test-driven development (TDD)* method. To help a teacher select blank statements from a code, the *program dependency graph (PDG)* has been used [4].

5) *Code Writing Problem (CWP)*: This problem asks students to write a whole code from scratch that satisfies the specifications described in the test code. The correctness of the code of students is also marked by the test code [5].

6) *Code Amendment Problem (CAP)*: In this problem type, a source code that has several missing or error elements, called a problem code, is shown to student. A student needs to identify the locations of missing or error elements in the code, and to fill in them or correct them with the correct elements. The correctness of any answer will be marked through string matching of the whole statement with the corresponding original one in the code [6].

7) *Code Completion Problem (CCP)*: In this problem, a source code with several missing elements is shown to the students without specifying their existences. Then, a student needs to locate the missing elements in the code and fill in the correct ones there. The correctness of the answer from a student is verified by applying string matching to each statement in the answer to the corresponding original statement in the code. Only if the whole statement is matched, the answer for the statement will become correct. Moreover, merely one incorrect element will result in the incorrect answer [7].

2.2 Code Writing Problem in JPLAS

The code writing problem (CWP) in JPLAS is designed for students how to write source code from scratch. This problem asks a student to write a whole source code that satisfies the specifications given by the test code. The JPLAS function for this problem is implemented based on the test-driven development (TDD) method using an open source framework JUnit [17]. When a student submits their code, it is automatically tested on the server to ensure its correctness. To create a new assignment in JPLAS, a teacher must prepare both the specifications and the test code.

2.2.1 TDD Method

The test-driven development (TDD) method is reviewed [18].

2.2.1.1 JUnit

JUnit assists the automatic unit testing of a Java source code or a class. Since *JUnit* has been designed with the Java-user friendly style, including the use of the test code programming, is rather simple for Java programmers. In *JUnit*, tests are performed by running methods whose names start with whose name starts with "assert". For instance, "assertEquals" method is used to compare the execution result of the source code with its expected value.

2.2.2 Test Code

A test code should be written by using libraries in JUnit. The following **Listing** 2.1 shows *MyMath* class source code that is used to introduce how to write a test code. *MyMath* class returns the summation of two integer arguments.

Listing 2.1: Source code for MyMath class

```
1 public class Math {
2     public int plus(int a, int b) {
3         return (a + b);
4     }
5 }
```

Then, the following **Listing** 2.2 shows test code tests plus method in *MyMath* class.

Listing 2.2: Test code for MyMath class

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3  public class MyMathTest {
4      @Test
5      public void testPlus() {
6          MyMath ma = new MyMath();
7          int result = ma.plus(1, 4);
8          assertEquals(5, result);
9      }
10 }
```

The test code imports *JUnit* packages containing required test methods at lines 1 and 2, and declares *MyMathTest* at line 3. @Test at line 4 indicates that the succeeding method represents the test method. Then, it describes the procedure for testing the output of plus method. This test is performed as follows:

1) An instance ma for *MyMath* class is generated.

2) plus method for this instance *ma.plus* is called with given arguments.

3) The result result is compared with its expected value using *assertEquals* method.

2.2.2.1 Features in TDD Method

In the TDD method, the following features can be observed.

1) The test code represents the specifications of the source code, because it must describe every function which will be tested in the source code.

2) The testing process of a source code is efficient because each function can be tested individually.

3) The refactoring process of a source code is simplified as the modified code can be tested immediately.

2.3 Offline Answering Functions in JPLAS

In addition to the online platform, the offline answering function has been implemented to allow students to answer the problems in *JPLAS* even if the students cannot access to the *JPLAS* server when the Internet is unavailable. This feature is really handy when there's no internet access. For offline use, students can get the problems and submit their answers using a USB drive. *Offline JPLAS* includes operation flow, file generation, and cheating prevention

2.3.1 Operation Flow

The operation flow of the offline answering function is as follows:

1) Problem instance download: a teacher accesses to the *JPLAS* server, selects the problem instances for the assignment, and downloads the required files into the own PC on online.

2) Assignment distribution: a teacher distributes the assignment files to the students by using a file server or USB memories.

3) Assignment answering: the students receive and install the files on their PCs, and answer the problem instances in the assignment using Web browsers on offline, where the correctness of each answer is verified instantly at the browsers using the *JavaScript* program.

4) Answering result submission: the students submit their final answering results to the teacher by using a file server or USB memories.

5) Answering result upload: the teacher uploads the answering results from the students to the *JPLAS* server to manage them.

2.3.2 File Generation

Table 2.1 shows the necessary files with their specifications for the offline answering function in *JPLAS*. These files are designed for the problem view, the answer marking, and the answer storage.

2.3.3 Cheating Prevention

In *Offline JPLAS*, the correct answers need to be distributed to the students so that their answers can be verified instantly on the browser. To prevent disclosing the correct answers, they will be distributed after taking hash values using *SHA256* [19]. In addition, to avoid generating the same hash values for the same correct answers, the assignment ID and the problem ID are concatenated with each correct answer before hashing. Then, the same correct answers for the different blanks are converted to the different hash values, which ensures the independence between the blanks.

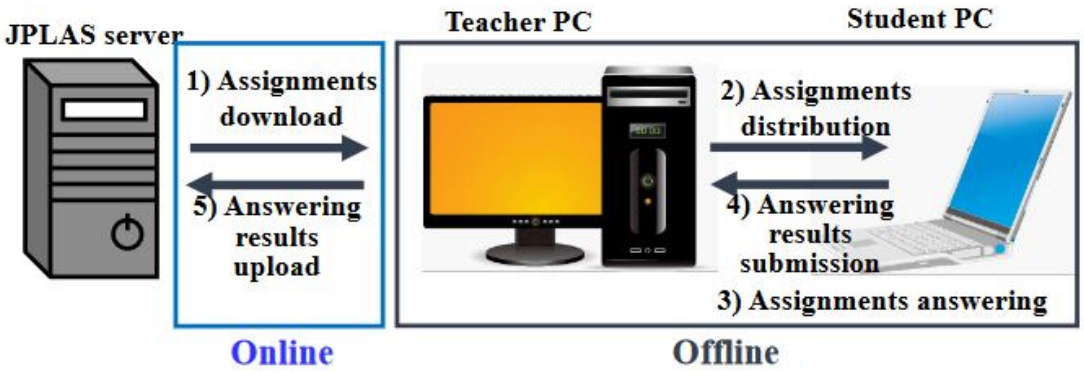

Figure 2.3: Operation flow for offline answering function.

Table 2.1: Files for distribution.

File name	Outline
css	CSS file for Web browser
index.html	HTML file for Web browser
page.html	HTML file for correct answers
jplas2015.js	js file for reading the problem list
distinction.js	js file for checking the correctness of answer
jquery.js	js file for use of jQuery
sha256	js file for use of SHA256
storage.js	js file for Web storage

2.4 Review of PLAS using Node.js

In this section, we provide an overview of the architecture and design principles of the *Node.js-based Programming Learning Assistant System (NPLAS)*. We discuss the features and functional-

ities offered by the platform, the utilization of *Node.js* for developing the web application system, and the integration of *Docker* for easy distribution and deployment [10].

2.4.1 Open Source Software

We introduce the open-source software utilized for implementing the NPLAS system with completeness and readability.

2.4.1.1 Node.js

Node.js is an *open-source* server environment compatible with various PC platforms like *Windows*, *Linux*, and *macOS*. It acts as an interpreter and runtime environment for executing *JavaScript* source codes on the server. This versatility allows developers to use *JavaScript* for building both desktop and server applications, streamlining the development process. *Node.js* adopts an event-driven, non-blocking I/O model, making it lightweight, and efficient compared to threaded servers like Apache.

In the context of web application servers, *Node.js* provides a built-in *HTTP* module for handling data transfer over the *HTTP* protocol. It can create *HTTP* servers that listen to server ports and respond to client requests. The *HTTP* module consists of two main components: the *request* module for retrieving or sending data resources to/from the server and the *response* module for serving data resources based on client requests. This capability enables *Node.js* to serve as a bridge between clients and servers, facilitating efficient communication and resource delivery.

2.4.1.2 Express.js

Express.js is the minimal and flexible framework specifically for *Node.js* that can provide a robust set of features for a web application development. A framework generally provides ready-made components or solutions and may include supporting programs, compilers, code libraries, and APIs that can be useful to develop applications and be customized to speed up it. *Express.js* provides the following features:

- For *routing*, it enables developers to define request handlers using different *HTTP* verbs and *URL* paths, facilitating efficient routing within the application.
- For *template engine*, it integrates with ”view” rendering engines, allowing developers to generate dynamic responses by inserting data into templates.
- For *HTTP server* and *template location*, it simplifies the configuration of common web application settings such as port connections and template locations, streamlining the development process.
- For *middleware*, it provides extensive middleware support, enabling developers to add additional request processing functions anywhere within the request handling pipeline. This allows for the inspection and filtering of incoming *HTTP* requests, improving application security and performance.

2.4.1.3 Embedded JavaScript (EJS)

The user interface is implemented using *Embedded JavaScript (EJS)*, *CSS*, and *Bootstrap* files, and is controlled by *Node.js* and *Express* programs. *EJS* is a template engine to help rendering *JavaScript* codes on the client-side. Basically, *EJS* is used for embedding *JavaScript* codes inside *HTML* codes. *EJS* is used on *Node.js* when it is working in *Express* framework.

2.4.1.4 Docker

Docker provides the flexibility and portability for running various software on different platforms. *Docker* is a framework of simplify managing and deploying containers or applications. A *Docker container* can be built from plain text files called a *Dockerfile* that describes human and machine readable recipes for creating computing environments and interacting with data. The *Docker container* can be run on anywhere as long as *Docker* is installed, and is built from the *Docker image* to offer the software environment for running the target application, which may include the source codes, the libraries, the middleware, and the parameters. Using *Docker*, the application can run on any platform without considering the installed software there. Thus, the software version or source code incompatibility problem can be easily solved on a PC.

2.4.2 Implementation of JPLAS platform using Node.js

We review the software architecture, server-side implementation, and the connection between the server and the browser.

2.4.2.1 Software Architecture

NPLAS is a web application system designed to enable educators to furnish programming exercises to numerous students while overseeing their learning activities on the server. In the server architecture depicted in Figure 2.4, *Node.js* is employed as the web application server, complemented by the *Express.js* framework. In realizing the MVC model-based application, Java is utilized for the *model (M)*, executing *JUnit* for testing answer source codes in code-writing problems. The *view (V)* is constructed using *EJS*, *CSS*, *JavaScript*, while *JavaScript* is employed in the *controller (C)*. Notably, no database system is incorporated for data management.

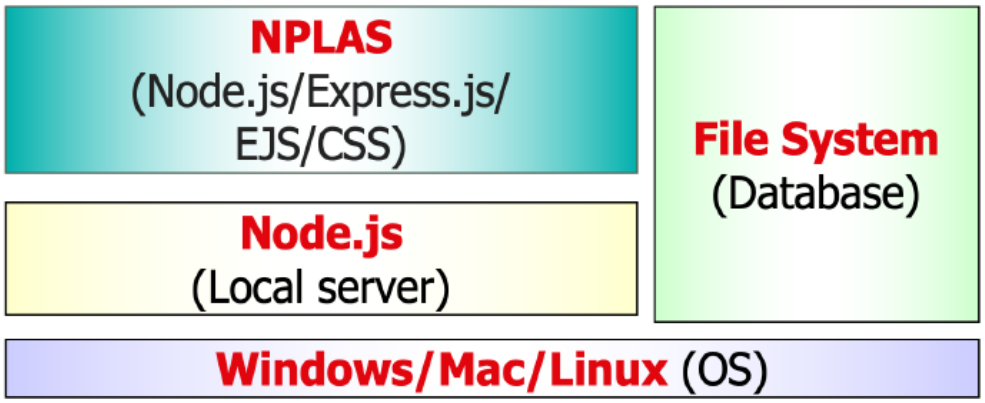

Figure 2.4: *NPLAS* server platform.

2.4.2.2 Server Side Implementation

Basically, *Node.js* has a complex structure and is not easy to maintain. Therefore, *Express* is used together in our implementation. *Express.js* follows the *MVC* structure as a programming design pattern. To use *Express*, we have to install *Node.js* and the *node package manager (npm)* together from the prepared binary packages for each operating system. Actually, when *Node.js* is installed, *npm* is automatically installed. *npm* is the important tool for working with *Node.js* applications. It provides the access to hundreds of thousands of reusable packages of *JavaScript* libraries that the applications need for developments, testing, or productions, and may also be used to run tests and tools used during the development process.

Other dependencies that are necessary to run the application, such as the frameworks and template engines, are imported into the application environment using the *npm* package manager. After installing *Express.js*, the directories for *bin*, *node-modules*, *public*, *routes*, and *views* are generated. The details will be described in the next section, including how each directory is working and why it is important to run the application well.

In *Node/Express.js*, each web application creates and runs its own web server. *Express.js* provides the methods to specify which function is called for a particular *HTTP* verb (GET, POST, SET, etc.) and *URL* pattern *"route"*, and to specify which template *"view"* engine is used, where template files are located, and which template is used to render a response.

2.4.2.3 File for Connection between Server and Browser

When a client navigates to the server name (*URL*) using the port such as *localhost:8000* from the browser, the implemented web application will perform the following procedures as shown in Figure 2.5.

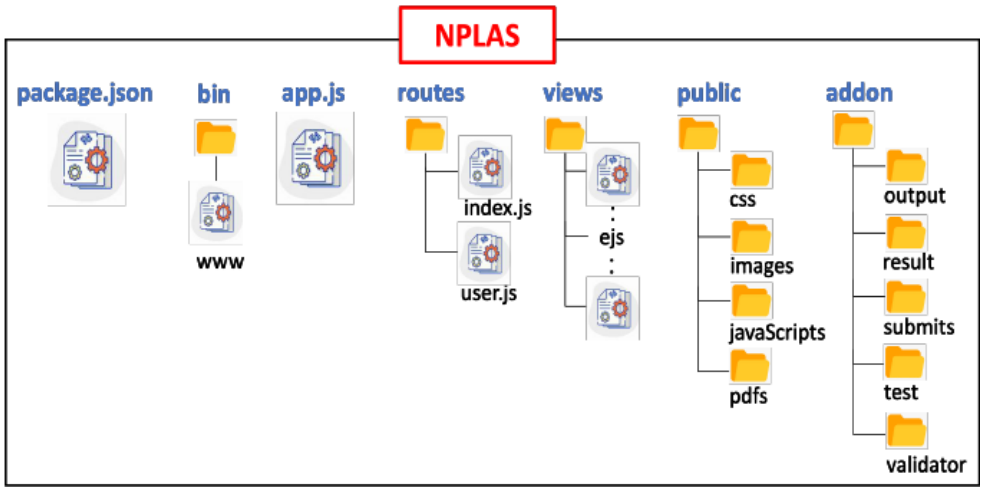

Figure 2.5: NPLAS application directory structure using Express.js.

- *package.json:* The application invokes the *package.json* file, which meticulously enumerates all dependencies for the specific *JavaScript* "package". This comprehensive listing encompasses 1) package's name, 2) version, 3) description, 4) initial executable file, 5) production dependencies, 6) development dependencies, and 7) compatible version of Node.js. The *package.json* file contains all the necessary information for *npm* to fetch and execute the application. By identifying the "start" key and the value "node ./bin/www" in the script tag,

it signifies that the *Node.js* project references the file named "www" within the bin folder. This file gathers data for *Express.js* to utilize within the application.

- *www file:* The *www* file defines the entry point of the application and accommodates various setup configurations. Within this application, three distinct scripts—namely, app, debug, and http—are configured in this file.
- *app.js:* The *app.js* will declare the package that is required by the application globally and will be the main root file of the whole application directory.
- *index.js:* The *index.js* file under the routes folder will route the application's requests to its appropriate controller and then, render the associated view.
- *views:* The views folder includes all the *NPLAS* user interface files that will be displayed to the browser by using *EJS*.
- *public:* All the static files, such as the *CSS*, images, and *JavaScript* files, are set up under the public folder.
- *addon:* The customized addon folder serves as the database, encompassing the test code files, answer source code files, and validator files.

Here, the *user.js* route was a result of it being part of the default structure provided by *Express.js* during the project initialization. It is noted that the current backend does not require the user route, and the authentication and authorization features are not implemented. As this application focuses on aspects of programming learning through the personal use, we have not implemented the user management functionality at this stage. Details were not described on the user.js route.

2.4.2.4 MVC Model Software Architecture in NPLAS

The proposed software architecture for the *NPLAS* platform adheres to the *MVC model*, recognized as the standard architecture for a web application system, as illustrated in Figure 2.6. *Java* is used in the *model*, *EJS/CSS/JavaScript* are in the *view*, and *JavaScript* is in the *controller*.

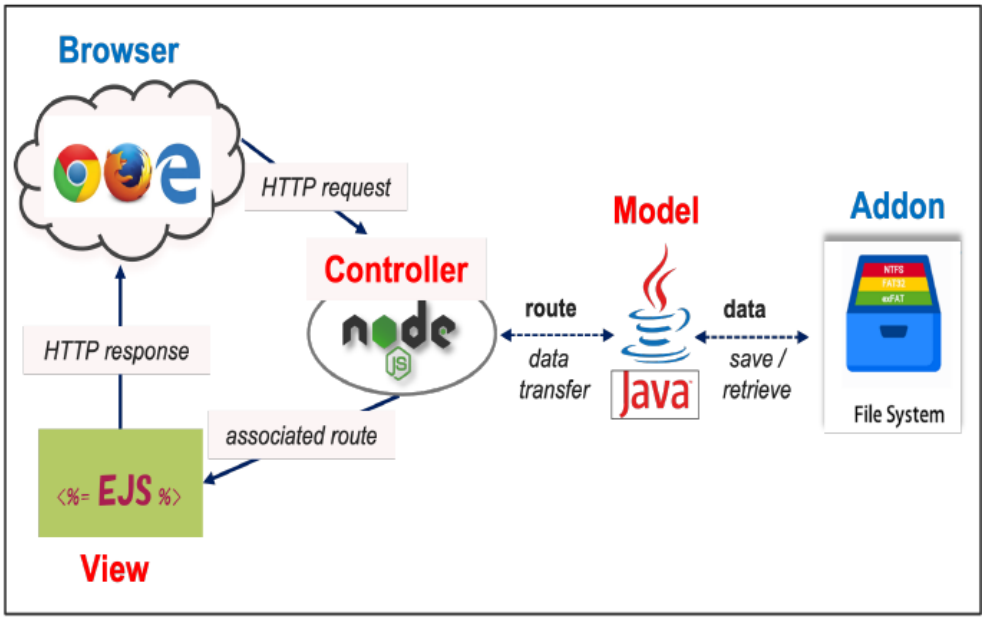

Figure 2.6: *MVC model* in *NPLAS* for *Java*.

1) *Model*: The *model* reads the required data files in the predefined file system called *"addon"*. They include the test code files, the answer source files, and the validator files. The marking function in *JPLAS* is different according to the problem type. For the *code writing problem*, the marking function of testing the answer code runs in the background of the server. This function is implemented by *Java* using *Eclipse*. It is exported as a *jar* file and is imported in *Node.js*. This function automatically tests any answer source code submitted from the browser by running the given *test code* on *JUnit*. The results including the *JUnit* log are recorded in the file system and can be viewed by the student in the browser.

   A student submits the assignment answer code and receives the result of the testing function to/from the *Node.js* server in the web browser. The folders to contain the necessary files to run the testing function are prepared in the file system under the *"addon"* folder. Then, the *JavaScript* program in the web browser can detect these folders and read/write the files in them. The testing function uses the files under these folders with the input parameters to test the answer source code on *JUnit*.

   For the other problem types in *JPLAS*, the marking function is implemented in the *JavaScript* program that runs in the browser when the student solves the problems and submits their answers on the browser. The marking function compares the answer with the predefined correct answer. The answers and the marking results are saved on the browser's local storage temporarily with the *unique* key. Then, when the student downloads them to a text file to be sent to the teacher, the *JavaScript* program saves the data in the local storage in a text file in the output folder under the *"addon"* directory of the application.

2) *View*: The *view* implements the user interface in *JPLAS* by using the *EJS* template engine to render dynamic contents in the browser. The *SkyBlue CSS* framework is used for the style. *EJS* is enabled to construct an *HTML* code along with *JavaScript* codes that are passed in via the backend of the application. All the *EJS* files for the application are included in the *"views"* directory. The application will call the file starting from the *index* file, which is the home page of the application. It consists of the title, the menu, and the main body for each

problem type. As a feature of this architecture, the fixed parts of the interface are made by *EJS* and *CSS*, while the changeable parts are made by *JavaScript* functions that are stored in the public directory. The main body needs to be exchanged with the related *EJS* file according to the route that the client requested. It can reduce the code size and simplify the code architecture.

3) *Controller*: The *controller* in *JPLAS* is implemented by *JavaScript*. When a request is received, the application works out what action is needed based on the *URL* pattern and the associated information contained in *POST* data or *GET* data. It may read or write the information from the file system, or perform the tasks required to satisfy the request.

   Here, *Express.js* will transfer the data via the related route. For the data transfer, the appropriate name is assigned in the route. Then, the application will return the response to the web browser, by often dynamically creating an *EJS* page for the browser to render the view related to the assigned name and to display it by inserting the retrieved data into the placeholders in the *EJS* template. *Node.js* and *Express.js* are perfectly capable of running a website with dynamic data. Besides, the structure has hierarchy structures or grouping concepts that can be maintained by anyone who understands *Node.js* and *Express*.

2.4.3 Adoption of Docker for JPLAS

In this section, we present the adoption of *Docker* to help students to install the *JPLAS* platform into their PCs.

2.4.3.1 JPLAS Docker Workflow

Figure 2.7 illustrates the procedure of adopting *Docker* for the installation of the *JPLAS* platform in a PC.

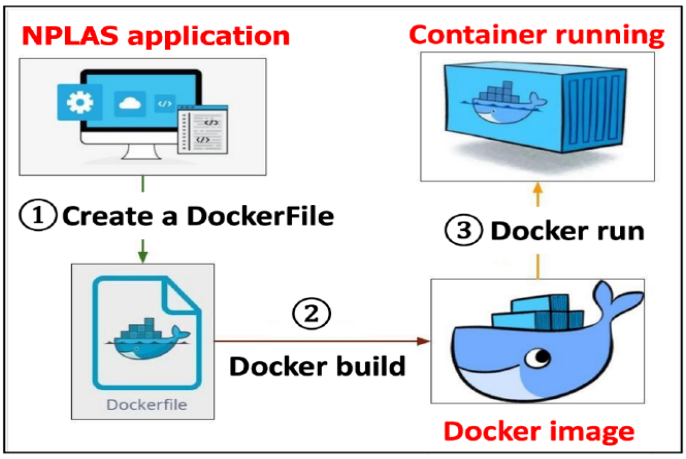

Figure 2.7: Installation procedure of *JPLAS* platform using *Docker*.

A *Docker container* can be generated using a plain text file named *Dockerfile*, providing both human-readable and machine-readable instructions for crafting computing environments and interacting with data. The *Docker container* operates independently anywhere, provided *Docker* is installed, and is constructed from the *Docker image* to provide the necessary software environment for executing the target application. To facilitate easy distributions of the *NPLAS Docker image* to

students, it is stored in the *Docker Hub* [20] account using the *Docker push* command, as illustrated in Figure 2.8.

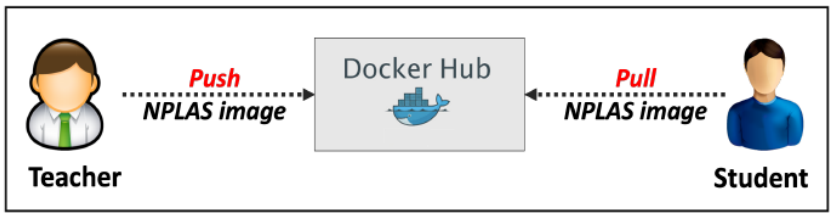

Figure 2.8: Upload and download JPLAS image from *Docker Hub*.

2.4.3.2 Usage Procedure

To utilize the *NPLAS* platform, students are required to execute the following steps:

1. Download and install *Docker* corresponding to the student's PC operating system. For *Windows OS*, the installation of *Windows Subsystem for Linux (WSL)* is also necessary.

2. After initiating *Docker* on the PC, download the *NPLAS Docker image from Docker Hub* using the *Docker pull command*.

3. Execute the *Docker run* command to operate the image on the PC. The specifics of this command may vary based on the OS of the PC. Following the execution of this command, the student should examine the *"output"* folder to store the answer files.

4. Access the *NPLAS platform* by opening the browser and entering *localhost:8000* in the address bar.

2.5 Summary

In this chapter, we reviewed *PLAS* projects, exploring its software architecture and core functionalities. Additionally, we also reviewed the implementation of the PLAS platform using *Node.js* and *Docker*.

Chapter 3

Implementation of CWP Platform for Python Programming

This chapter presents the implementation of *Code Writing Problem (CWP)* platform for *Python Programming Learning Assistant System (PyPLAS)*.

3.1 Introduction

Previously, the platform for *Java programming learning assistant system (JPLAS)* as the self-study tool for *Java* programming was implemented on *Node.js* under the uniform design using *JavaScript*[10]. The programs on both the server and client sides can be made using *JavaScript*.

Python is commonly used in industries and academics due to rich libraries and short coding features. *Python* has been widely involved in developing applications, task automation, data analysis, data visualizations, and machine learning. Thus, the cultivation of *Python* programming engineers has been highly demanded from industries. However, many universities are not offering *Python* programming courses, due to time and teaching-staff limitations. Thus, high-quality self-study tools are very important. To bridge this gap, I implement the *CWP* platform for *Python* programming learning by extending the *JPLAS platform* [21].

In *CWP* of *PyPLAS*, each assignment or instance requests a student to write a source code that will pass the given *test code*. The test code describes the specifications to be satisfied in the assignment. The primary purpose of *CWP* is to assess and enhance the problem-solving and code writing skills of a student based on the *test-driven development (TDD)* method.

3.2 Implementation of CWP Platform

In this section, I present the software architecture, server-side implementation, and the connection between the server and the browser for the implementation of the *CWP* platform. For each CWP instance, a pair of the source code and the test code needs to be prepared by the teacher. The teacher should check the correctness of the source code by running the test code on *unittest*. The user interface of this *PyPLAS* is dynamically controlled with *EJS* that can avoid the complex syntax structure. *Docker* is adopted to make students easily install the platform in their own PCs. The user interface at the client shows the CWP assignments and accepts the answer code submissions from the students. Then, by running the *test code* on *unitest* at the server, the correctness of each answer code is automatically verified and is returned to the interface to be shown.

3.2.1 Software Architecture

PyPLAS is a web application system designed to allow educators to provide programming exercises to many students while managing their learning activities on the server. As illustrated in Figure 3.1, the server architecture employs *Node.js* as the web application server, using the Express.js framework. In implementing the *MVC (Model-View-Controller) model*, *Python* is used for the *model (M)* component, with *unittest* used for testing the answer source codes in coding exercises. The *view (V)* is built with *EJS*, *CSS*, and *JavaScript*, while *JavaScript* is used for the *controller (C)*. Notably, the system operates without incorporating a database for data management.

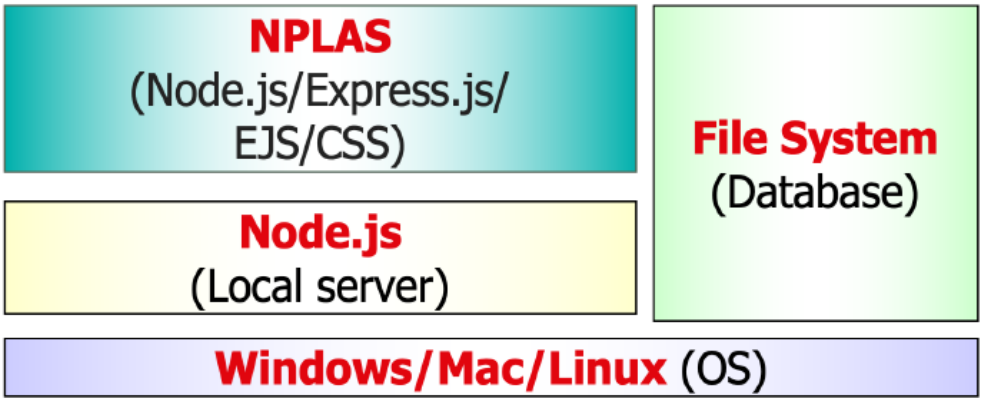

Figure 3.1: Server platform.

3.2.2 Server Side Implementation

Node.js is basically complex and challenging. To simplify the process, I use *Express.js*, which adheres to the *MVC* design pattern. To get started with *Express.js*, I first install *Node.js* along with the *node package manager (npm)* from the available binary packages for each operating system. It's important to note that installing *Node.js* also automatically installs *npm*. *npm* is a crucial tool for working with *Node.js* applications as it provides access to a vast repository of reusable *JavaScript* libraries necessary for development, testing, and production. Additionally, *npm* can be used to run tests and other tools during the development process.

Other essential dependencies, such as frameworks and template engines, are imported into the application environment using the *npm* package manager. After installing *Express.js*, the directory structure is set up, creating directories for *bin*, *node-modules*, *public*, *routes*, and *views*. Each directory has a specific role, which will be detailed in the next section to explain their importance in ensuring the application runs smoothly.

In *Node.js* and *Express.js* setup, each web application creates and operates its own web server. *Express.js* provides the methods to define which functions are called for specific *HTTP* verbs (GET, POST, etc.) and *URL* patterns ("routes"). It also allows specifying which template ("view") engine to use, the location of template files, and the templates used to render responses.

3.2.3 Connection between Server and Browser

When a client navigates to the server name with *URL* using the port number, such as *localhost:8000*, from the browser, the implemented web application at the server will perform the following procedures as depicted in Figure 3.2.

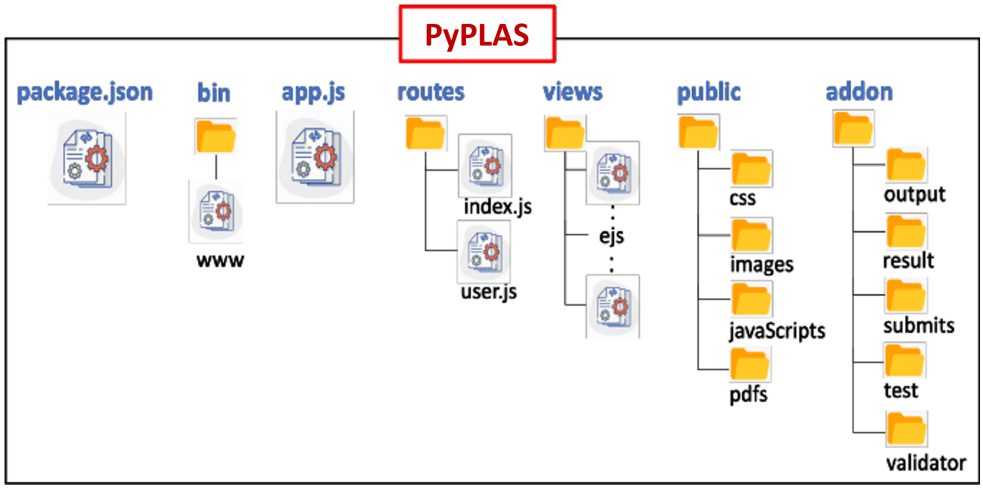

Figure 3.2: PyPLAS application directory structure using *Express.js*.

- **package.json:** First, the application calls the *package.json* file. This file contains all the dependencies and scripts needed by *npm* to fetch and run the application. The *"start"* script, typically set to *"node ./bin/www"*, instructs the *Node.js* project to execute the "www" file located in the *bin* folder, which sets up *Express.js* for the application.
- **www:** This file acts as the entry point of the application and allows for different setup configurations. It includes three main scripts, *app*, *debug*, and *http*, where each serves the specific initialization purpose.
- **app.js:** This is the main root file of the application directory, declaring all the globally required packages and serving as the central hub for the application's configuration.
- **index.js:** Located in the routes folder, this file directs incoming application requests to the appropriate controllers and renders the corresponding views.
- **views:** This folder contains all the user interface files, which are rendered in the browser using EJS.
- **public:** This folder contains all static assets, such as CSS, images, and JavaScript files, used by the application.
- **addon:** This customized "addon" folder includes test code files, answer source code files, and validator files necessary for the application's functionality.

3.2.4 Validation Process of Answer Source Code

In *CWP*, the answer source code submitted from a student is validated by the unit testing using the *unittest* library. Figure 3.3 illustrates this validation process in the answer platform. *unittest* offers a range of features for effective *unit testing* of a *Python* source code, including test fixtures, test cases, test suites, and test runners. *Test fixtures* provide fixed environments for running tests, ensuring repeatability. *Test cases* define conditions to determine whether the source code under the test works correctly. *Test suites* collect test cases for various specifications or behaviors of source codes. *Test runners* set up test executions and provide outcomes to the user.

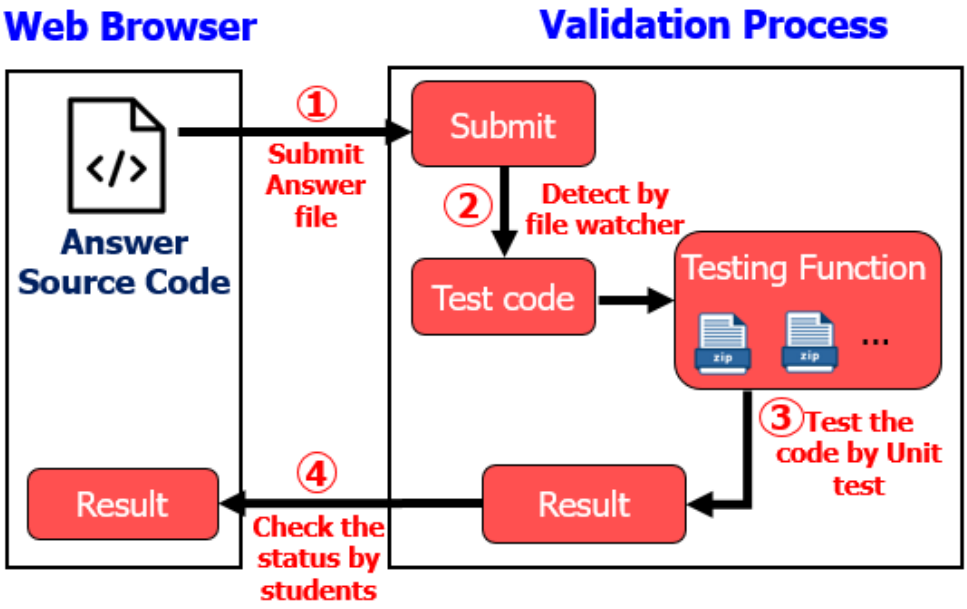

Figure 3.3: Answer code validation process.

3.2.5 MVC Model Software Architecture

The answer platform is implemented as a web application. *Node.js* is employed as the web application server and is combined with the *Express.js* framework to facilitate web application developments. The *EJS* (Embedded JavaScript) template engine is employed for rendering dynamic contents in the view. The file system of the operating system is utilized for efficient data management within the platform. The *unittest* library is employed for testing the answer source codes. The platform eliminates the need for clients to manually navigate the server name (URL) along with the port, such as *localhost:8000*, when accessing the platform through the browser.

The software architecture of the answer platform follows the *Model-View-Control (MVC) model* that has been widely recognized as the standard architecture for web applications. Figure 3.4 illustrates the software architecture of the answer platform. The model is implemented by *Python*. It reads the necessary data files from the predefined file system, including test code files, answer source files, and validator files. Then, it automatically tests the submitted answer source code by executing the provided test code on the *unittest* framework. The testing results along with the *unittest* logs are stored in the file system and can be accessed by students through the browser.

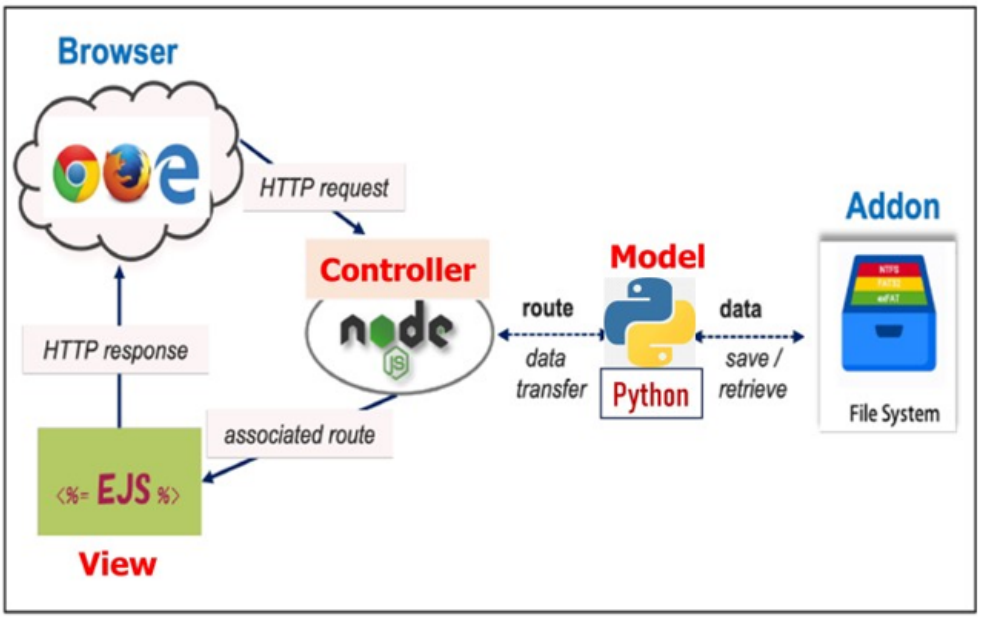

Figure 3.4: *MVC model* in *PyPLAS* on *Node.js*.

1) *Model*: The *model* reads the required data files in the predefined file system called *"addon"*. They include test code files and the answer source code files. Each source code file is verified by running the corresponding *test code* on *unittest*. The results including the *unittest* log are recorded in the result file, and can be viewed by the student at the browser. From the web browser, a student can submit the assignment answer code and receive the testing results to/from the server.

2) *View*: The *view* implements the user interface in *PyPLAS* using the *EJS* template engine to render dynamic contents at the browser. All the *EJS* files for this application are included in the *"views"* directory. The application will call the file starting from the *index* file, which is the home page of the application. It consists of the title, the menu, and the main body for each problem type. The main body needs to be exchanged with the related *EJS* file according to the route that the client requested.

3) *Controller*: The *controller* in *PyPLAS* is implemented by *JavaScript*. When a request is received from the browser, the application works out what action is needed from the *URL* and the associated information contained in *POST* data or *GET* data. It may read or write the information in the file system, or perform the tasks required to satisfy the request.

3.2.6 Problem Answer Interface

Figure 4.1 shows the problem answer interface of *CWP platform*. Students can write the source code in the designated place, click "Submit" button to validate the code, see the validation result in the black box, and use buttons to select the next and previous problems.

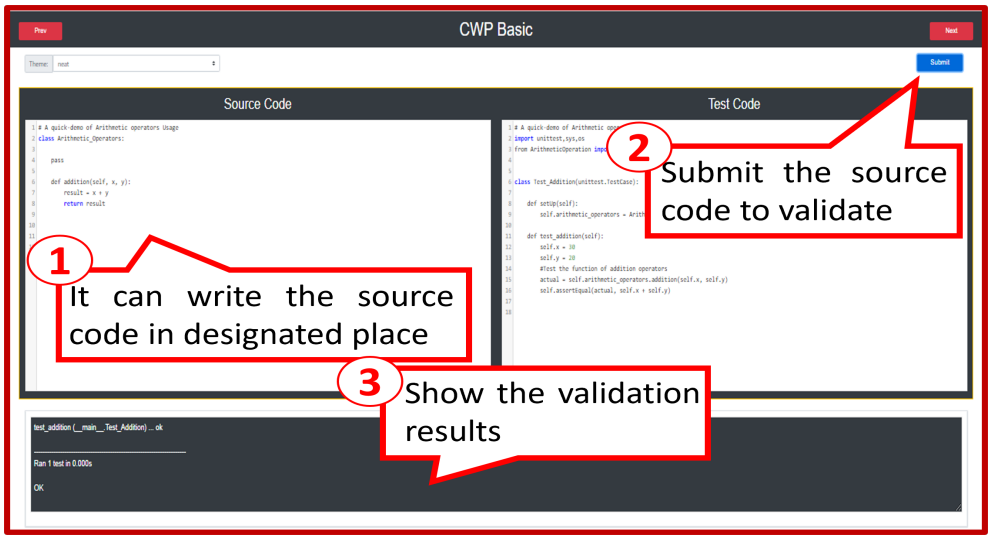

Figure 3.5: Problem answer interface.

3.2.7 Problem List Interface

Figure 3.6 shows the list of the problem instances to be solved in this category. The student can easily see which instance is successfully finished or still trying or not solving.

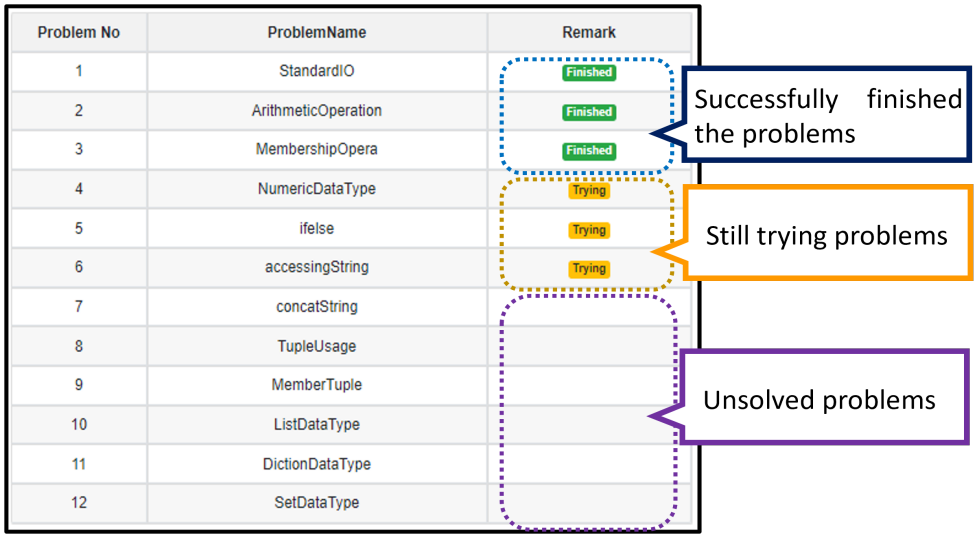

Figure 3.6: Problem list interface.

3.3 Docker for Answer Platform Distribution

Docker provides the flexibility and portability for running various software on different platforms. *Docker* is a framework of simplify managing and deploying containers or applications. A *Docker container* can be built from plain text files called a *Dockerfile* that describes human and machine readable instructions for creating computing environments and interacting with data. The *Docker container* can be run on anywhere as long as *Docker* is installed, and is built from the *Docker image* to offer the software environment for running the target application, which may include the source codes, the libraries, the middleware, and the parameters. Using *Docker*, the application can

run on any platform without considering the installed software there. Thus, the software version or source code incompatibility problem can be easily solved on a PC.

3.3.1 Dockerfile Creation

Figure 3.7 illustrates the installation procedure of the answer platform on a PC using *Docker*. The process involves creating a *Dockerfile* that outlines the necessary instructions for copying files and installing the required software components. Once the *Dockerfile* is created, a *Docker image* is built by executing the instructions in the file. This image includes all the dependencies including the libraries needed for the platform, and is stored in either *Docker Hub* or a local registry. After generating the *Docker image*, it can be run as a container using the *Docker* "run" command. This container allows access to student answer results, which are stored in specific folders within the container directory. This installation procedure using *Docker* streamlines the setup process and provides a reliable and efficient environment for running the answer platform on a PC.

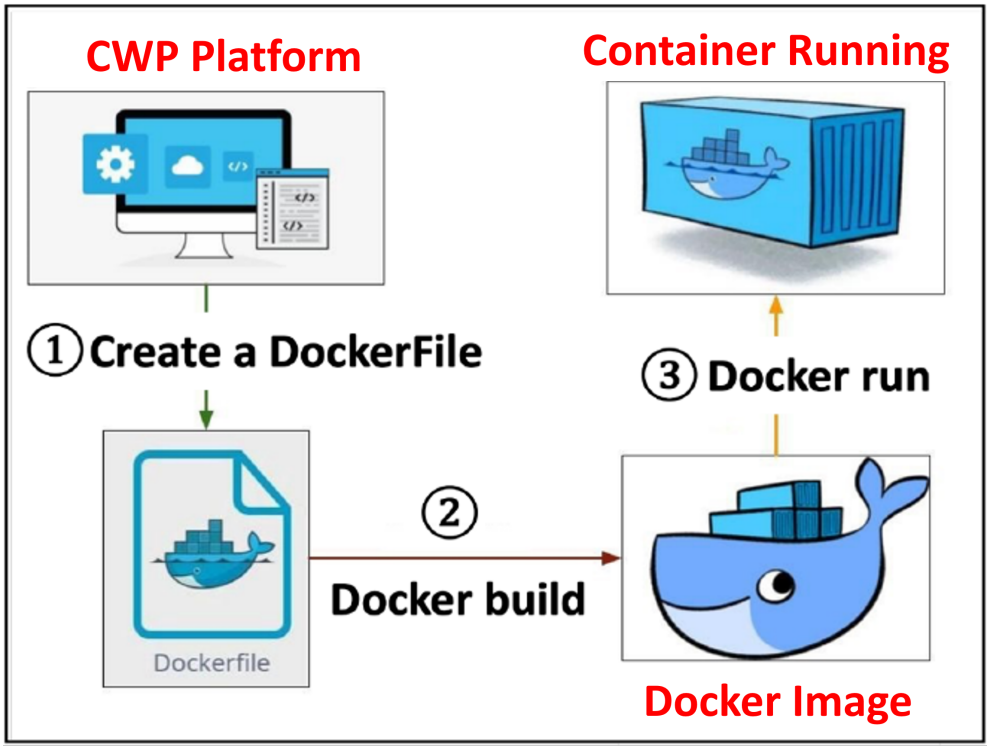

Figure 3.7: Installation procedure of *PyPLAS* platform using *Docker*.

3.3.2 Installation of Platform

To distribute the *Docker* image to students easily and correctly, it is stored in a *Docker Hub* account using the "push" command. *Docker Hub* is the platform provided and managed by *Docker* for sharing container images. Creating a *Docker Hub* account is a straightforward process, as it can be easily done through the *Docker Hub* website. Once the *Docker image* is uploaded to the *Docker Hub* account, students who have been granted the access to the account can download the image using the "pull" command as shown in Figure 3.8 . This allows them to seamlessly obtain the *Docker image* from the *Docker Hub* and set up the answer platform for their use.

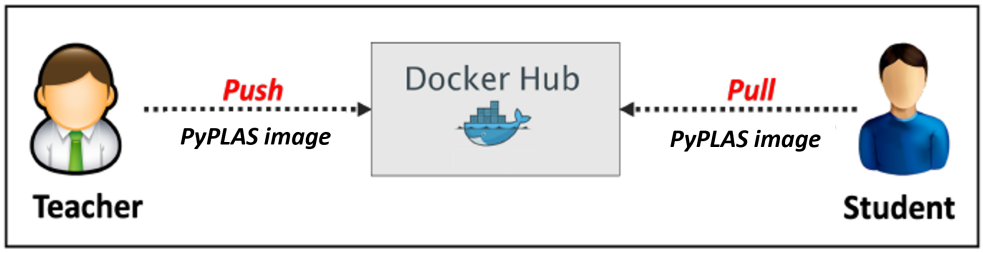

Figure 3.8: Upload and download of *PyPLAS* image from *Docker Hub*.

The *Docker* image for our environment is influenced by the "python:3.8-slim-buster" *Docker image*. This image includes the *Python* runtime and a slim version of the *Debian Linux* operating system. The choice of "python:3.8-slim-buster" helps balance the need for essential components and libraries for Python execution with a smaller overall image size, as it excludes nonessential components.

The image offers compatibility with our software packages and libraries, providing a larger package repository and robust security updates. By using this base image, I ensure that the necessary *Python* environment is set up efficiently, supporting the execution of our applications within the container. The image may take a few minutes to download initially. However, once downloaded, it allows for seamless execution of *Python* applications without the need for an Internet connection, enabling users to work on their projects anytime and anywhere.

3.4 Evaluation

In this section, I evaluated the implemented CWP platform through the use by novice students. In this evaluation, I prepared the usage manual and requested to students from Japan and Indonesian universities to install and use the platform. Then, I collected their feedback by questionnaires. Table 3.1 shows 10 questions and their replies, where 5 is best and 1 is the least. Most students were satisfied with this platform, which confirms the validity.

However, some students found some difficulty in the installation. The manual should be improved. Besides, some students mentioned that the use of *integrated development environment (IDE)* such as *VS code* is more convenient to solve *CWP* than this platform, because the error messages and the solved test cases are easily seen there. It will be helpful to recommend students to use IDE for solving *CWP*.

3.5 Summary

This section presented the implementation of the *code writing problem (CWP)* platform for *Python* programming learning. Several software including *Node.js*, *Express.js*, *EJS*, *Docker*, and *unittest* are used together. In the evaluation, 20 students installed and used the platform, where the questionnaire results confirmed the validity and the effectiveness. In future works, I will implement helpful functions for programming learning in the platform, include other exercise problems for *Python* programming, and extend the platform to other programming languages such as *C*, *C++*, and *JavaScript*.

Table 3.1: Questions and results on CWP platform.

no.	**question**	**# of students**				
		1	2	3	4	5
1	Is it easy to solve the assignments using the platform?	0	0	2	12	6
2	Do you think the platform is useful for studying *Python* programming?	0	0	1	10	9
3	Are the instructions in the manual clear?	0	0	5	8	7
4	Is the installation process of the platform easy?	0	0	2	9	9
5	Do you think the working status for each problem is accurate?	0	0	1	10	9
6	Do you think the answers' result are clearly to know after solving the problem?	0	0	2	11	7
7	Do you think programming study is improved after solving the problem?	0	0	2	12	6
8	Are you satisfied with the platform?	0	0	3	9	8
9	Do you think this platform is better than IDE?	0	0	5	8	7
10	How many rate do you want to give the platform?	0	0	2	11	7

Chapter 4

Software Testing in CWP Platform for Python Programming

In this chapter, I present the *software testing* mechanism for the *Code Writing Problem (CWP)* in the *Python Programming Learning Assistant System (PyPLAS)* using its web-based answer platform [22].

4.1 Overview of CWP

A CWP instance is intended for the student to write the *source code* that will pass all the tests described in the given *test code* at running *unitest* [23]. Each test represents the specification in the assignment that must be satisfied in the answer code. The test result is immediately returned to the student. When all the tests are passed, the code is regarded as the correct one. Otherwise, the student needs to modify and correct the code. Therefore, once the test codes are made by the teacher, the student can continue writing the codes for the assignments by themselves until reaching the correct ones. The assignments cover fundamental concepts and syntax of Python programming, including control structures, data types, loops, and conditional statements [24].

4.2 CWP answer interface

Figure 4.1 shows the screenshot of the answer interface for a CWP assignment. When a student accesses to this page, the test code on the right side is displayed. Then, the student needs to write the source code on the left side while reading the test code and analyzing the required specifications for the source code from the test code. After completing the source code, the student can submit it by clicking the button, and the source code is tested by *unittest*. The test result will appear automatically on the lower side of the page [21].

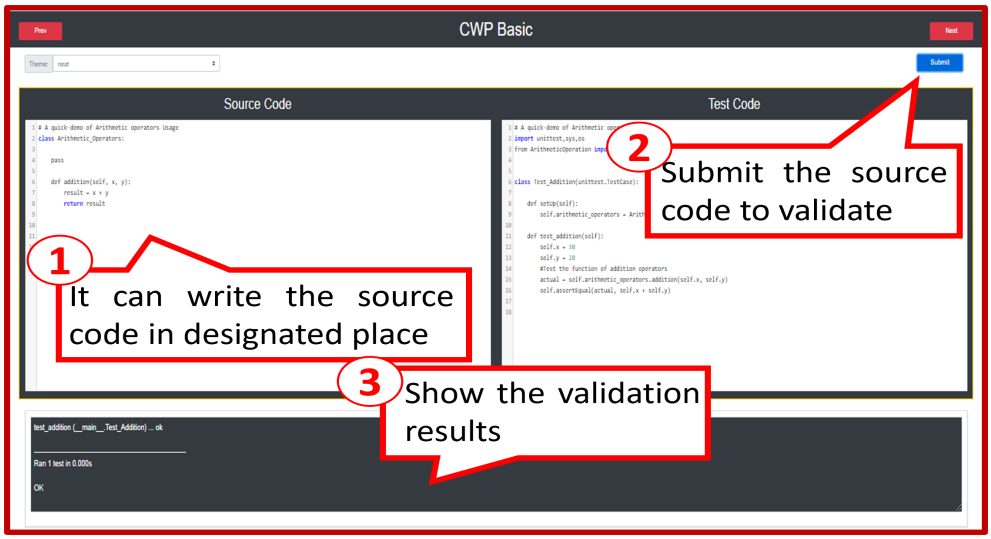

Figure 4.1: CWP answer interface.

4.3 Test Driven Development

CWP is implemented based on the *test driven development (TDD)* method. In TDD, the *test code* should be written before or while the *source code* is implemented, so that it can verify whether the current source code satisfies the required specifications during its development process. Modifications of the source code can be repeated until it passes every test in the test code.

4.3.1 Unit Testing Framework

unittest is a built-in unit testing framework in the *Python* programming platform to test given source codes. It assists the automatic unit testing of a source code or a class by running a *test code*. The *test code* needs to import the *unittest* module and create the class for testing by extending the library class *TestCase*. Each test method in the *test code* compares the execution result of the source code with its expected result. When they are equal, the test is passed. Otherwise, it is failed.

4.4 Test Code

In a CWP assignment, the *test code* plays the important role of checking whether the answer source code from a student satisfies the required specifications given by the test code or not. The test code needs to import the *unittest* library. The *unittest* library offers several *assertion* functions for testing. By properly using them, a test code can verify various functions in a source code. Some of the *assertion* functions are as follows:

- *assertEqual()* compares the produced output of the source code with the expected value.
- *assertIn()* checks whether a given value is presented within the output being tested.
- *assertNotIn()* checks whether a given value is not presented within the output being tested.
- *assertIsInstance()* checks whether a given object is the instance of the expected class.

4.4.1 Test Code Example

First, I present the test code for a source code in *basic Python programming concepts*. **Listing** 4.1 shows a *test code* to test the `addition` method in the *Arithmetic_Operators* class that will return the *addition* result of the two integer arguments. For reference, **Listing** 4.2 shows the *source code* that will pass the test code.

Listing 4.1: Test code in example CWP

```
1  import unittest,sys,os
2  from ArithmeticOperation import Arithmetic_Operators
3  class Test_ArithmeticOperation(unittest.TestCase):
4      def setUp(self):
5          self.arithmetic_operators = Arithmetic_Operators()
6      def test_addition(self):
7          self.x = 2
8          self.y = 2
9          actual = self.arithmetic_operators.addition(self.x, self.y)
10         self.assertEqual(actual, self.x + self.y)
11 if __name__ == '__main__':
12 unittest.main()
```

Listing 4.2: Source code in example CWP

```
1  class Arithmetic_Operators:
2      def addition(self, x, y):
3          return x + y
```

The class/method names in the test code should be related to the corresponding names in the source code so that their correspondences become clear:

- The class name in the test code is given by `Test_tested class name` in the source code.
- The method name in the test code is given by the `test_tested method name` in the source code.

The test code imports the *unittest* library at lines 1 and 3. At line 4, an object of the `ArithmeticOperators` class in the source code, named `self.arithmetic_operators`, is generated. At line 9, the `addition` method in the `Arithmetic_Operators` class is executed with 2 and 2 for the two arguments. Then, at line 10, the result is compared with the correct one 4 using the *assertEqual* method. In general, a test code performs the following procedure [25]:

1. It generates an object of the target class to be tested in the source code.
2. It calls the method of the object to be tested with the test input data.
3. It compares the output result of the method with its expected value as the test output data through the *assertEqual* method.

Table 4.1: CWP instances and results.

ID	topic	# of test cases	avg. correct answer rate
1	Arithmetic Operation	6	100%
2	Standard I/O	3	100%
3	Membership Operators	3	100%
4	Numeric Data Type	4	100%
5	If Else Condition	3	100%
6	Accessing String	4	100%
7	Concat String	2	100%
8	Tuple Usage	2	100%
9	Member Tuple	2	100%
10	List Data Type	3	92%
11	Dictionary Data Type	3	97%
12	Set Data Type	3	93%
13	Obj Creation	3	100%
14	Obj and Class	6	100%
15	Array Creation	2	100%
16	Array Accessing	4	100%
17	Array Removing	2	100%
18	Array Slicing	3	97%
19	Array Searching	2	100%
20	Array Updating	3	100%
21	JSON to String	1	95%
22	Updating JSON	1	95%
23	Various to JSON	9	96%
24	Dictionary to JSON	1	95%

4.5 Application Results

In this section, I generate 24 CWP instances for basic and advance grammar topics from textbooks [24] on the CWP platform, and assign them to 20 novice students from Japan and Indonesia universities.

4.5.1 CWP instances and solution results

Table 5.2 shows the instance ID, the topic, the number of test cases (test methods), and the average number of test cases that were passed by the student answer codes for each CWP instance.

4.5.2 Discussions

The solution results in Table 5.2 show that all the students solved the CWP instances correctly, except for ID=10, 11, 12, 18, 21, 22, 23, 24. The results indicate that some students are still not

familiar with the collection data types of *list*, *dictionary*, and *set*, and feel difficult to handle the *JSON* data type. In future works, we will improve the descriptions with the hints of the related instances in the CWP platform.

4.6 Summary

In this chapter, I presented the *software testing* mechanism for the *Code Writing Problem (CWP)* in the *Python Programming Learning Assistant System (PyPLAS)* using its web-based answer platform on*Node.js*. 24 instances on basic and advance grammar topics were generated and distributed to 20 students in Japan and Indonesia. The results confirmed that most of the students well solved them whereas some students will need cares at programming study. In future works, I will improve test codes with functions for assisting students to understand test codes and debug source codes, and continue generating CWP instances for other concepts in *Python* programming.

Chapter 5

Investigation of VTP for C++ Programming

In this chapter, I present the *Value Trace Problem (VTP)* for *C++ Programming in Programming Learning Assistant System (PLAS)*. While CWP focuses on enhancing students' coding and problem-solving skills, VTP is designed to improve their understanding of program execution and variable tracking. Both CWP and VTP are problem types of PLAS, each addressing different but complementary aspects of programming education.

5.1 Overview of VTP

The *VTP* for *C++ programming* is designed to provide students with opportunities to read and analyze *C++* code, emphasizing the importance of *code reading* for writing high-quality code. Each *VTP* instance includes a *C++* source code, a set of questions with answer forms, and the correct answers. Students must determine the actual values of important variables or output messages when the source code is executed, which are displayed on the standard output in the code. To create a new *VTP* instance, corresponding standard output statements must be added to the source code, with important variables and output messages manually selected. By carefully reading and tracing the source code, learners can enhance their understanding of the corresponding study. The correctness of the answers is marked by comparing them with the correct ones stored on the server.

The design goals of *VTP* are:

1. To present a variety of useful source codes for *C++ programming* study in complete form for novice learners.
2. To provide references that describe essential *C++ programming* topics related to the *VTP* instances, aiding novice learners.
3. To ensure that learners can correctly answer the questions by carefully reading and understanding the source codes.
4. To enable automatic marking of answers through string matching.

5.2 Generation of Value Trace Problem

In this section,the generation of new *the value trace problem (VTP)* for *C++ programming* is presented.

5.2.1 Generation Procedure of VTP

To compose a VTP instance, a teacher needs to prepare a source code, a set of questions, the answer forms (blanks), and the correct answers. To generate a new VTP instance, the procedures are as follow:

1) Select a source code to be studied by the students from a website or a textbook.

2) Find the important variables and standard output messages in the source code, and make the questions of asking the actual values or messages of them.

3) Insert the standard output statements into the source code to output them by compiling and running it.

4) Collect the correct answers to the questions by running the code and observing the corresponding standard outputs from the code.

5) Make the source code, the questions, and the correct answers into one text file.

6) Run the program in [26] with the text file in e), and generate the HTML/CSS/JavaScript files for the offline answering function for this VTP instance.

7) Register the generated VTP instance in the assignment to students.

For the automatic execution of this procedure, we implemented the necessary programs in Java and the script by Bash.

5.2.2 Selection of C++ Source Code

To clarify the *VTP* instance generation procedure, we explain the details by using the following **Listing** 5.1 source code. This code is designed to aid learners in comprehending the concept of one-dimensional arrays in *C++ programming*.

Listing 5.1: Source code example for VTP instance

```
1  # include < iostream>
2  # include < string>
3  using namespace std;
4
5  int main() {
6      string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
7
8      for (int i = 0; i < 4; i++) {
9          cout << i << ": " << cars[i] << "\n";
10     }
11     return 0;
12 }
```

Listing 5.2: Input file example for VTP instance

```
# include < iostream >
# include < string >
using namespace std;

int main() {
    string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};

    for (int i = 0; i < 4; i++) {
        cout << i << ": " << cars[i] << "\n";
    }
    return 0;
}

The outputs are:
_1_ : _2_
_3_ : _4_
_5_ : _6_
_7_ : _8_
0, Volvo, 1, BMW, 2, Ford, 3, and Mazda.
```

5.2.3 Generating Assignments

Once the source code is selected, the teacher needs to manually add the relevant standard output statements in the source code, and must prepare the corresponding questions of asking the values/messages at the standard output along with their correct answers. Next, the teacher needs to put together the source code, the questions, and the correct answers into a single text file as shown in **Listing** 5.2. This file is then utilized as input to execute the program responsible for generating the *HTML/CSS/JavaScript* files for the offline answering function.

5.2.4 Answer Interface for VTP

The answer interface for a *VTP* instance is accessible through a web browser, facilitating student engagement with the exercises both online and offline. The answer marking is processed by running the JavaScript program on the browser. To avoid cheating, correct responses are encrypted using *SHA256*.

Figure 5.1 illustrates the answer interface for an example *VTP* instance. This question asks the values of important variables, i and cars[i]. Their correct answers are 0, Volvo, 1, BMW, 2, Ford, 3, and Mazda. A student needs to read the source code carefully to understand it, fill in the forms, and click the "Answer" button. Then, the form becomes white if the answer is correct, and red otherwise as shown in Figure 5.2. The student can repeat the answering process until all the answers become correct.

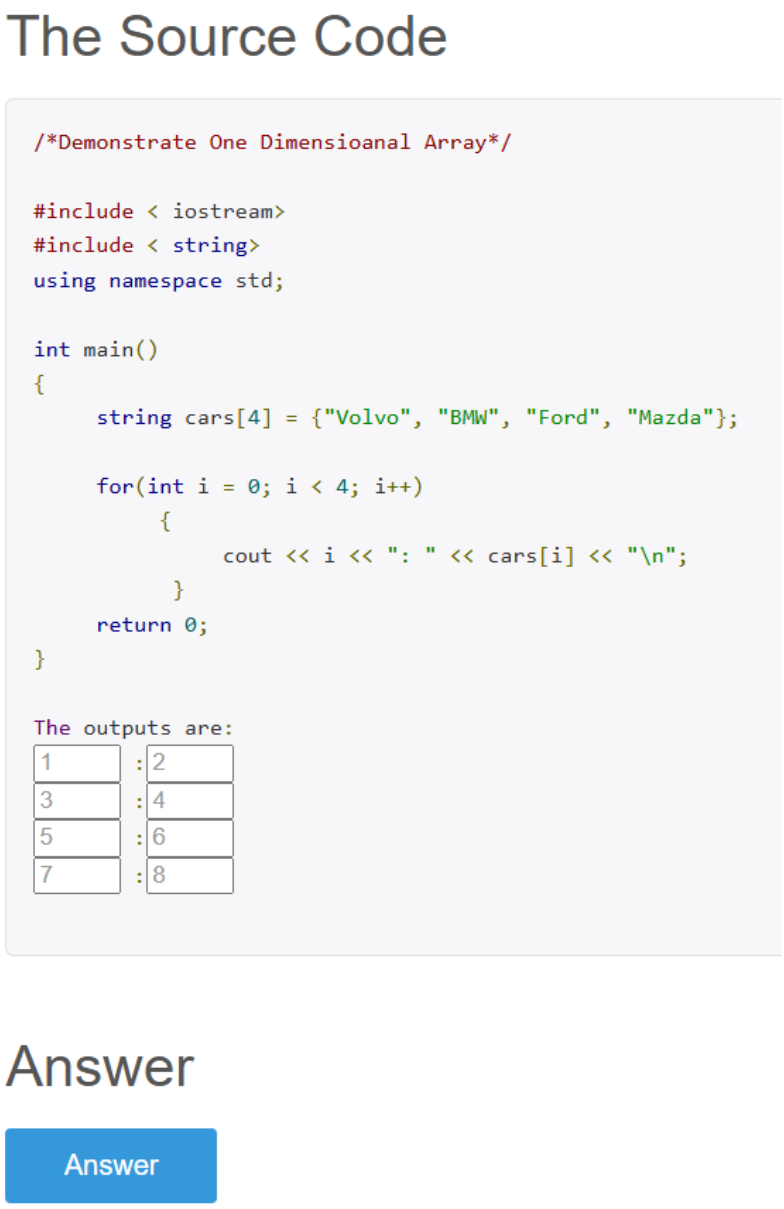

Figure 5.1: VTP answer interface.

5.3 Evaluation

In this section, we evaluate the generated 37 VTP instances for C++ programming through applications to 17 first-year or second-year undergraduate students in Yangon University, Myanmar, 13 graduate students in Okayama University, Japan, and 16 graduate students in Electronic Engineering Polytechnic Institute of Surabaya University, Indonesia based on student's solution results and VTP instance results.

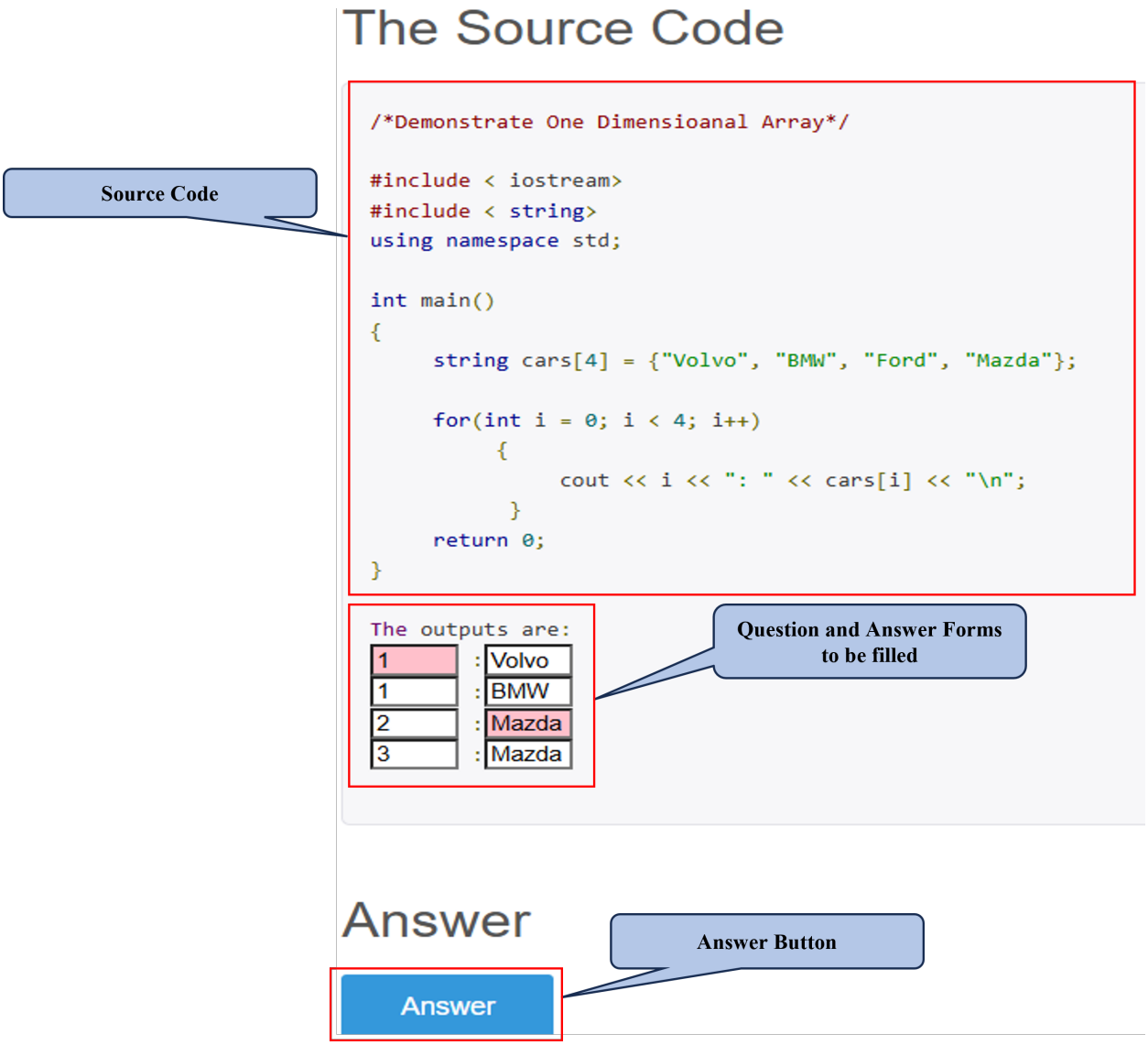

Figure 5.2: VTP answer interface with explanation in details.

5.3.1 VTP Instances for Basic Grammar Concepts

As shown in Table 5.1, the 37 instances contain the total of 126 answer forms (blanks). I generated these instances using source codes in websites [27]-[28] and textbook [29] for basic grammar concepts in C++ programming. Table I shows the instance ID, the basic grammar concept, the number of lines in the source code, the number of questions, and the number of answer forms on each VTP instance. I asked the students to solve them using the offline answering function for JPLAS at home. These students have studied basic grammar concepts for C++ programming. The first-year students have studied the C++ programming for three months, the second students have studied it for at least one year. The graduate students have studied it for several months at undergraduate programs.

Table 5.1: VTP for basic grammar concepts

ID	Topic	# of lines	# of questions	# of forms
1	Variable usage	10	3	3
2	Data type, size usage	12	6	6
3	Arithmetic operators	14	4	4
4	Relational operators	14	3	5
5	If statement	19	2	2
6	If else statement	19	2	2
7	Nested if statement	22	2	2
8	Switch statement	31	2	2
9	Default value in switch statement	18	2	2
10	Nested switch statement	20	4	4
11	For loop	11	1	1
12	While loop	12	5	5
13	Do while loop	14	5	5
14	Nested loop	13	1	18
15	One dimensional array	12	1	18
16	Finding average of a set of values	24	2	2
17	Two dimensional array	16	6	6
18	String copy function	13	2	2
19	String length function	9	1	1
20	Combining string and number	13	3	3
21	Pointer	11	2	2
22	Reverse case using array indexing	18	2	2
23	Function	14	2	2
24	Function with parameters	14	3	3
25	Function adding two numbers	13	1	1
26	Local variable	14	2	2
27	Global variable	16	2	2
28	Return value in function	11	1	1
29	Function with call by reference	19	2	4
30	Function overloading	25	3	4
31	Class and object	22	4	4
32	Method in class	20	1	2
33	Constructor	26	6	6
34	Encapsulation	29	1	1
35	Polymorphism	37	1	3
36	Inheritance	24	1	2
37	Exception handling	22	2	2
	Average	17.6	2.5	3.4
	Total	651	93	126

5.3.2 Student Solution Results

First, I analyze the performance of each student in solving the VTP instances. Table 5.2 shows the student ID, the number of correctly solved answer forms, the total number of answer submission times to mark the answers, the missing instance ID in solving, and the correct answer rate among the 126 forms.

Table 5.2: Result for each student

Student ID	# of correct answers	# of submissions	# of missing instances	Correct answer rate
1	126	81	0	100%
2	126	50	0	100%
3	126	37	0	100%
4	126	63	0	100%
5	120	61	0	91%
6	122	38	0	98%
7	126	37	0	100%
8	126	52	0	100%
9	126	54	0	100%
10	126	40	0	100%
11	126	50	0	100%
12	126	92	0	100%
13	126	59	0	100%
14	107	39	0	81%
15	109	42	0	85%
16	121	56	20	97%
17	121	43	18	96%
18	126	43	0	100%
19	126	37	0	100%
20	126	50	0	100%
21	126	55	0	100%
22	126	40	0	100%
23	125	53	0	99%
24	125	87	0	99%
25	126	53	0	100%
26	126	84	0	100%
27	116	65	0	92%
28	126	87	0	100%
29	122	55	0	97%
30	122	55	0	97%
31	126	96	0	100%
32	126	43	0	100%
33	126	74	0	100%
34	126	54	0	100%
35	126	58	0	100%
36	126	43	0	100%
37	126	62	0	100%
38	126	46	0	100%
39	126	72	0	100%
40	126	61	0	100%
41	126	48	0	100%
42	126	41	0	100%
43	126	88	0	100%
44	126	80	0	100%
45	126	78	0	100%
46	126	69	0	100%
average	124.3	58.1	0.8	98.5%
SD	4.1	16.8	3.9	0.00

5.3.2.1 Individual Student Analysis

Based on the result, most of the students solved all the instances correctly. Among them, three students with ID=3, 7and 19 are excellent, because they solved every VTP instance with only one

Table 5.3: Correct answer rate distribution of students

Range of Correct Answer Rate	# of Students
80% - 89%	2
90% - 99%	9
100%	35

Table 5.4: Submissions times distribution of students

Submission Time Range	# of Students
0 – 37	3
38 – 74	34
75 - 111	9

submission. On the other hand, students with ID=5, 6, 14, 15, 16, 17, 23, 24, 27, 29 and 30 could not solve some instances. Two students with ID=16 and ID=17 missed to solve one instance, although they show the high answer rate 97% and 96%. They should be more careful in solving all the instances. It will be necessary to improve the interface of the answering function to avoid it. Two students with ID=14 and 15 show the lower correct rate than 90%. These students will need a lot of efforts to improve the understanding of C++ programming.

The average number of submission times 58.1 indicates that a student submitted answers for one instance two times on average. The standard deviation is 16.8, which suggests the large diversity among the students. The average correct answer rate 98.5% indicates that a student correctly solved 98.5% of the questions on average.

5.3.2.2 Correct Answer Rate Distribution

Table 5.3 shows the distribution of the correct answer rates of the students. This table indicates that 35 students among 46 achieved the 100% rate, and 9 did over 90% correct rate. On the other hand, two students did under 90% rate, who will need to take time for learning the fundamental C++ programming.

5.3.2.3 Submission Time Distribution

Table 5.4 shows the distribution of the numbers of answer submission times by the students. Three students with ID=3, ID =7 and ID=19 correctly solved each instance by submitting the answer only one time for any VTP instance. They are excellent students who understand C++ programming very well. Students with ID=1, ID=12, ID=24, ID=26, ID=28, ID=31, ID=43, ID=44 and ID=45 submitted the answers more than 75 times, which is the largest. However, ID=1, ID= 12, ID=26 and ID=28, ID=31, ID=43, ID=44 and ID=45 achieved the 100% correct rate respectively. It means that these students were seriously solving the VTP instances by submitting answers many times, and actually achieved the solutions. They may need more practices in C++ programming.

5.3.3 Individual VTP Instance Result

Next, I analyze the solving results of the individual VTP instances by the students. Table 5.5 shows the instance ID, the number of students who did not attempt to solve, the total number of answer submissions to mark the answers, and the average correct answer rate among the 46 students.

Table 5.5: Result of each VTP instance

Instance ID	# of Unattempted Students	# of Submissions	Average Correct Rate
1	0	72	100%
2	0	122	99%
3	0	58	100%
4	0	96	100%
5	0	70	100%
6	0	51	100%
7	0	50	100%
8	0	55	100%
9	0	55	100%
10	0	66	100%
11	0	87	100%
12	0	55	100%
13	0	67	100%
14	0	65	100%
15	0	56	100%
16	0	113	98%
17	0	70	94%
18	1	103	99%
19	0	54	100%
20	1	72	99%
21	0	70	96%
22	0	86	100%
23	0	53	100%
24	0	71	100%
25	0	50	100%
26	0	51	100%
27	0	70	100%
28	0	54	100%
29	0	51	100%
30	0	50	100%
31	0	69	96%
32	0	82	94%
33	0	72	100%
34	0	53	96%
35	0	69	96%
36	0	215	88%
37	0	68	100%
Average	0.054	72	99%
SD	0.229	29.9	0.03

Table 5.6: Correct answer rate distribution of instances

Range of Correct Answer Rate	# of Instances
80% – 90%	1
90% – 100%	36

Table 5.7: Grammar concepts of hard instances

Instance ID	Topic
2	data type, size usage
16	finding average of set of values
17	two dimensional array
18	string copy function
20	combining string and number
21	pointer
31	class and object
32	method in class
34	encapsulation
35	polymorphism
36	inheritance

5.3.3.1 Individual Instance Analysis

Table 5.5 indicates that in general, the 37 VTP instances are relatively easy for them. The average correct answer rate is 100% for any instance, except for the instances with ID=2, 16, 17, 18, 20, 21, 31, 32, 34, 35, and 36. These instances may be relatively difficult for them, where up to eleven students made mistakes in them.

5.3.3.2 Correct Answer Rate Distribution

Table 5.6 shows the distribution of the correct answer rates of the VTP instances. 36 instances among 37 achieved over the 90% correct rate. Thus, they are suitable for self-studies of novice students. The one instance ID=36 achieved less than 90% correct rate. This instance may be difficult for novice students. In future works, I will improve these instances so that more students can try to solve and answer them correctly.

5.3.3.3 Analysis of Hard Instances

Table 5.7 shows the grammar concepts of the VTP instances where the average correct rate did not reach 100%. They include data type, array, string, pointer, and object-oriented programming specific concepts that are generally hard for novice students at studying C++ programming. The teacher need to explain them in more comprehensible ways using illustrations or subsidiary tools. Besides, we should implement hint functions to help students understand them, which will be in future works.

5.4 Summary

In this chapter, I investigated the effectiveness of the *value trace problem (VTP)* for self-study of C++ programming through code reading. I generated 37 VTP instances using simple source codes for basic grammar concepts of C++ programming, and asked 46 students to solve them using the offline answering function at home.

By analyzing their answer results, I could detect the understanding levels of the students in C++ programming that prove the generated VTP instances are suitable for the novice students and the hard concepts for them. The hard concepts include data type, array, string, pointer, and object-oriented programming specific ones that are generally difficult topics for novice students at studying C++ programming. The teacher should take time in explaining them in comprehensible ways using illustrations or subsidiary tools. Besides, a hint function to encourage students' understanding should be implemented at the answering function to VTP instances, which will be in our future works.

In future studies, I will generate hard instances in addition to hint function to improve students' understanding in their studies. Moreover, we will generate new VTPs using source codes for other grammar concepts or programming topics such as libraries, data structure, and algorithms and object-oriented instances, and will use them in C++ programming courses.

Chapter 6

Related Works

In this section, I introduce related works to this thesis.

In [30], Hwang et al. proposed the *web-based programming assisted system for cooperation (WPASC)* designing learning activity for facilitating cooperative programming learning, and investigated cooperative programming learning behaviors of students and the relationships with learning performances.

In [31], Quinson et al. presented the *programmer's learning machine (PLM)* as an interactive exerciser aimed at learning programming and algorithms. It targets students in (semi) autonomous settings, using an integrated and graphical environment that provides a short feedback loop. This generic platform also enables teachers to create specific programming micro-worlds that match their teaching goals. PLM provides two main panels to provide information for students to solve exercises.

In [32], Samy et al. developed the CPP tutor, an intelligent tutoring system designed to create an interactive learning environment for students. This system aims to facilitate the learning of C++ programming by providing personalized feedback and adaptive learning pathways. As a result, students can grasp programming concepts more quickly and effectively than with traditional teaching methods, enhancing their overall learning experience.

In [33], Ihantola et al. conducted a systematic literature review on automated assessments of programming assignments. Their work encompasses a detailed analysis of the major features and approaches from both pedagogical and technical perspectives. The review highlights various methodologies for automated grading and discusses their effectiveness in improving student learning outcomes, along with the technical challenges involved in implementing these systems.

In [34],, Jain et al. developed an educational tool for understanding algorithm, and building and learning programming language. This tool provides an innovative and unified graphical user interface for developments of multimedia objects, educational games, and applications. It also provides an innovative method for code generations to enable students to learn the basics of programming languages using drag-n-drop methods of image objects.

In [35], Zingaro et al. proposed a web-based tool that enables an instructor to use code writing assignments in a classroom. It supports use cases and scenarios for classroom implementations. Submitted codes are evaluated by the suite of tests designed to highlight common misconceptions, so that the instructor receives real-time feedback as students submit code. The system also allows the instructor to pull specific submissions into the editor and visualizer for use as in-class examples.

In [36], Pritchard et al. presented *Computer Science Circles* as a free programming website for beginners of *Python* programming. It offers the auto-grader function based on the *stdin/stdout* approach with randomizing test cases, code scramble exercises of correcting wrong ordered codes,

and hints.

In [37], Tosun et al. investigated impacts of the test-driven development on the effectiveness of unit test cases compared to the incremental test-last development in industrial contexts. This study reveals that TDD may have positive impacts on software development productivity. Moreover, TDD is characterized by the higher ratio of active development time in the total development time than the test-last development approach.

In [38], Aniche et al. conducted both open source and industry projects related assert instructions in unit tests with quality measures of codes being tested. This study observed that when a production method has a unit test using the "assert" instructions for more than one objects, it often exhibits the higher cyclomatic complexity, the number of lines of a code, or the higher number of method invocations. It means that developers should monitor the number of assert instructions in the unit test as it may indicate drawbacks in the produced code.

In [39], Thirumalesh et al. used the TDD methodology in two different environments (Windows and MSN divisions) at Microsoft. This study measured various contexts, products, and outcome measures to compare and evaluate the efficacy of TDD. It also observed the significant increase in quality of the code (greater than two times) for projects developed using TDD compared to similar projects developed in the same organization in non-TDD fashions. Additionally, the unit test is served as the auto documentation for the code when libraries and APIs are used as well as for the code maintenance.

In [41], Garner presented learning resources and tools to help novices learn programming, addressing the challenges of introductory software development. The author discusses tools such as micro-worlds, video clips, flowchart interpreters, and program animators, framed within the four phases of the software lifecycle: problem analysis, solution design, algorithm implementation, and testing/revision.

In [42], Abu-Naser et al. presented an intelligent tutoring system for learning *Java* objects. By bringing together recent developments of tutoring systems, cognitive science, and artificial intelligence, they constructed an intelligent tutor system to help students learn *Java* programing.

In [43], Cai et al. studied performances of scientific applications with *Python* programming. They investigated several techniques for improving computational efficiencies of serial *Python* codes and discussed basic programming techniques for parallelizing serial scientific applications.

In [44], Bogdanchikov et al. suggested *Python* use for teaching programming to novice students, because the programming language has neatly organized syntax and powerful tools to solve any task. They gave some examples of program codes written in *Java*, *C++*, and *Python*, and made comparisons between them. They also pointed advantages of *Python*.

In [45], Adawadkar described the main features of *Python* programming and listed out the differences between *Python* and other programming language with helps of some codes. The author discussed applications of *Python* programming and showed good examples.

In [46], the author presented online *Python* tutor so that the teachers and students can write *Python* programs directly in the web browser (without installing any plugin), step forwards and backwards through executions to view the run-time state of data structure, and share their program visualizations on the web.

In [47], Helminen et al. introduced a program visualization and programming exercise tool for *Python* by aiming to target students apparent fragile knowledge of elementary programming, which manifests as difficulties in tracing and writing even simple programming. It provides an environment for visualizing the line-by-line execution of *Python* programs and for solving programming exercises with support for immediate automatic feedback and an integrated visual debugger.

In [48], Robinson et al. introduced an automated approach for generating maintainable regres-

sion unit tests for programs. Unlike previous studies, this method focuses on libraries rather than applications and aims to identify bugs rather than create maintainable regression test suites. The study proposes techniques to enhance existing unit test generation systems, resulting in tests with good coverage and readability for developers.

In [49], Elenbogen et al. presented a collection of interactive web exercises and development environments aimed at aiding language acquisition in introductory *C++* courses. These resources are specifically crafted to support beginners in grasping fundamental programming concepts and honing their coding skills through hands-on practice.

Chapter 7

Conclusion

In this thesis, as the first contribution, I implemented a web-based *code writing problem (CWP)* platform for *Python programming* learning using *Node.js* by extending the platform for *Java programming*. The user interface of this platform is dynamically controlled with *EJS*, simplifying the syntax structure. To facilitate easy installation of the platform software on students' personal computers, *Docker* is employed. The client-side user interface displays the CWP assignments and accepts code submissions from students. The submitted code is then tested on the server using *unittest*, which automatically verifies the correctness of each answer. The results are promptly returned and displayed on the user interface.

For evaluations, I prepared a usage manual and requested 20 students in universities in Japan and Indonesia to install and use the platform. Feedback was collected from these students. The majority expressed satisfactions with the platform, affirming its validity. However, two students reported encountering difficulties during the installation process.

As the second contribution, I explored the software testing of the web-based *CWP platform* for *Python* programming learning. Initially, I collected source codes from various websites and textbooks, focusing on fundamental grammar concepts in *Python* programming. Subsequently, I manually generated the corresponding test codes to verify the source codes, ensuring that the correctness of the students' answer source codes can be automatically validated on the answer platform.

For evaluations, I distributed the generated CWP instances to novice students in Japan and Indonesia, who were asked to use the answer platform to solve them. The results showed that the majority of the students successfully completed the code writing tasks, which confirmed the usefulness and effectiveness of the proposal in supporting self-study for *Python* programming. However, some students are still not familiar with the collection data types of *list*, *dictionary*, and *set*, and feel difficult to handle the *JSON* data type. So, some students will need cares at programming study.

As the third contribution, I studied the *VTP* for *C++* programming learning. *C++* is important for practical applications due to its speed and efficiency. However, limited university courses has underscored the necessity for self-learning tools. To aid beginners, I investigate VTP for *C++* programming, which prompts students to determine the value of key variables or outputs in source code. This approach emphasizes the selection of source codes covering basic grammar concepts to provide support for novice learners.

For evaluations, I collected 37 source codes from websites and textbooks for basic grammar concepts in *C++* programming, and generated *VTP* instances manually, after analyzing important variables and outputs messages in the codes. Then, to verify the effectiveness of the generated

37 *VTP* instances, we assigned 46 students from Myanmar, Japan and Indonesia universities. The results revealed that out of the 46 students, 35 successfully solved all the questions, while only two encountered difficulties. These findings underscore the importance of providing additional support and guidance to students facing challenges in their early stages of programming study.

In future works, I plan to further improve the *PLAS* platform by adding a wider range of exercise problems, incorporating interactive learning resources, and extending its support to include additional programming languages. These enhancements will provide more comprehensive learning opportunities and better support for students' self-study needs.

Bibliography

[1] 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," J. Adv. Inf. Technol., vol. 12, no. 4, pp. 342-350, 2021.

[2] K. K. Zaw, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N. K. Dim, and W.-C. Kao, "A proposal of value trace problem for algorithm code reading in Java programming learning assistant system," Inf. Eng. Express, vol. 1, no. 3, pp. 9-18, 2015.

[3] 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 Int. J. Comput. Sci. vol. 44, no. 2, pp. 247-260, 2017.

[4] N. Ishihara, N. Funabiki, and W.-C. Kao, "A proposal of statement fill-in-blank problem using program dependence graph in Java programming learning assistant system," Info. Engr. Exp., vol. 1, no. 3, pp. 19-28, Sept. 2015.

[5] N. Funabiki, Y. Matsushima, T. Nakanishi, N. Amano, "A Java programming learning assistant system using test-driven development method," IAENG Int. J. Comput. Sci., vol. 40, no. 1, pp. 38-46, 2013.

[6] H.H.S. Kyaw, N. Funabiki, and W.-C. Kao, "A proposal of code amendment problem in Java programming learning assistant system," Int. J. Inf. Educ. Technol., vol. 10, No. 10, pp. 751-756, Oct. 2020.

[7] H. H. S. Kyaw, S. S. Wint, N. Funabiki, and W.-C. Kao, "A code completion problem in Java programming learning assistant system," IAENG Int. J. Comput. Sci., vol. 47, no. 3, pp. 350-359, 2020.

[8] 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," Int. J. Inf. Educ. Technol., vol. 12, pp. 1158-1163. 2022.

[9] N. Ishihara, N. Funabiki, M. Kuribayashi, and W.-C. Kao, "A software architecture for Java programming learning assistant system," Int. J. Comp. Soft. Eng., vol. 2, no.1, 2017.

[10] S. T. Aung, N. Funabiki, L. H. Aung, H. Htet, H. H. S. Kyaw, and S. Sugawara "An implementation of Java programming learning assistant system platform using Node.js," in Proc. ICIET, pp 47-52, April 2022.

[11] R. McKendrick, Monitoring docker, first edition, Packt, 2015.

[12] D. Herron, Node.js web development - fifth edition, Packt, 2020.

[13] Express, https://expressjs.com/.

[14] Top Programming Languages 2023, https://spectrum.ieee.org/top-programming-languages-2023/.

[15] N. Ishihara, N. Funabiki, M. Kuribayashi, and W.-C. Kao, "A software architecture for Java programming learning assistant system," J. Comp. Soft. Eng., vol. 2, no. 1, Sept. 2017.

[16] N. Funabiki, Tana, K. K. Zaw, N. Ishihara, andW.-C. Kao, "A graph-based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system," IAENG Int. J. Comput. Sci. vol. 44, no. 2, pp. 247-260, 2017.

[17] JUnit, http://www.junit.org/.

[18] K. Beck, Test-driven development: by example, Addison-Wesley, 2002.

[19] SHA-256 Cryptographic Hash Algorithm, https://www.movable-type.co.uk/scripts/sha256.html/.

[20] Docker Hub, https://hub.docker.com/signup/.

[21] S. L. Aung, N. Funabiki, S. H. M. Shwe, S. T. Aung, and W-C. Kao, "An implementation of code writing problem platform for Python programming learning using Node.js," in Proc. GCCE, pp. 854-855, Oct. 2022.

[22] S. L. Aung, N. Funabiki, S. H. Shwe, E. D. Fajrianti, and S. Sukaridhoto, "An application of code writing problem platform for Python programming learning," in Proc. GCCE, pp. 856-857, Jan. 2022.

[23] Unit testing framework, https://docs.python.org/3/library/unittest.html/.

[24] John M. Zelle. *Python Programming - An Introduction to Computer Science*, 3rd ed.; Franklin, Beedle & Associates, 2017.

[25] N. Funabiki, Y. Matsushima, T. Nakanishi, and N. Amano, "A Java programming learning assistant system using test-driven development method," IAENG Int. J. Comput. Sci., vol. 40, no. 1, pp. 38-46, 2013.

[26] N. Funabiki, H. Masaoka, N. Ishihara, I-W. Lai, and W.-C. Kao, "Offline answering function for fill-in-blank problems in Java programming learning assistant system," in Proc. ICCE-TW, pp. 324-325, May 2016.

[27] C++ Programming, https://www.programiz.com/cpp-programming/.

[28] C++ Programming, https://beginnersbook.com/2017/08/c-plus-plus-tutorial-for-beginners/.

[29] H. Schildt, "C++ : A begineer's guide, 2nd edition", McGraw-Hill Education, 2003.

[30] W.-Y. Hwang, R. Shadiev, C.-Y. Wang b, and Z.-H. Huang, "A pilot study of cooperative programming learning behavior and its relationship with students learning performance," Comput. Edu. vol. 58, pp. 1267–1281, 2012.

[31] P. Brusilovsky and S. Sosnovsky, "Individualized exercises for self-assessment of programming knowledge: an evaluation of QuizPACK," J. Edu. Res. Comput., vol. 5, no. 6, 2005.

[32] S. S. A. Naser, "Developing an intelligent tutoring system for students learning to program in C++," Inform. Tech. J., vol. 7, no. 7, pp. 1055-1060, 2008.

[33] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppala, "Review of recent systems for automatic assessment of programming assignments," in Proc. Koli Call. Int. Conf. Comput. Edu. Research, pp. 86-93, Oct. 2010.

[34] A. K. Jain, M. Singhal, and M. S. Gupta, "Educational tool for understanding algorithm building and learning programming languages," in Proc. World Cong. Eng. Comput. Sci., pp. 292-295, Oct. 2010.

[35] S. Zingaro, Y. Cherenkova, O. Karpova, and A. Petersen, "Facilitating code writing in PI classes," in Proc. ACM Tech. Symp. Comput. Sci. Educ., pp. 585-590, Mar. 2013.

[36] D. Pritchard and T. Vasiga, "CS Circles: An in-browser Python course for beginners," in Proc. ACM Tech. Symp. Comput. Sci. Educ., pp. 591-596, Mar. 2013.

[37] A. Tosun, M. Ahmed, B. Turhan, and N. Juristo, "On the effectiveness of unit tests in test-driven development," in Proc. Int. Conf. Softw. Syst. Process, pp. 113-122, May 2018.

[38] M.F. Aniche, G.A. Oliva, and M.A. Gerosa, "What do the asserts in a unit test tell us about code quality? A study on open source and industrial projects," in Proc. Eur. Conf. Softw. Maint. Reeng., pp. 111-120, Mar. 2013.

[39] T. Bhat and N. Nagappan, "Evaluating the efficacy of test-driven development: Industrial case studies," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. (ISESE), pp. 356-363, Sep. 2006.

[40] S. S. Wint, N. Funabiki, and M. Kuribayashi, "Design and implementation of desktop-version Java programming learning assistant system," Proc. HISS, pp. 254-257, Nov. 2018

[41] S. S. Garner, "Learning resources and tools to aid novices learn programming," in Proc. Int. Conf. Informing Science, vol. 2, no. 2, June 2003.

[42] S. Abu-Naser, A. Ahmed, N. Al-Masri, A. Deeb, E. Moshtaha, and M. Abu-Lamdy, "An intelligent tutoring system for learning Java objects," Int. J. Art. Intell. Appli., vol. 2, no. 2, pp. 68-77, April 2011.

[43] X. Cai and H. P. Langtangen, "On the performance of the Python programming language for serial and parallel scientific computations," Sci. Programm., vol. 13, no. 1, pp. 31-56, Jan. 2005.

[44] A. Bogdanchikov, M. Zhaparov, and R. Suliyev, "Python to learn programming," J. Physics: Conf. Series, vol. 432, 2013.

[45] K. Adawadkar, "Python programming - applications and future," Sci. J. Impact Factor, pp 849–857, April 2017.

[46] P. J. Guo, “Online Python tutor: embeddable web-based program visualization for CS education,” in Proc. ACM Tech. Symp. Comput. Sci. Edu., pp. 579-584, Mar. 2013.

[47] J. Helminen and L. Malmi “JYPE- a program visualization and programming exercise tool for Python,” in Proc. Int. Symp. Soft. Visual., pp 153-162, Oct. 2010.

[48] B. Robinson, M. D. Ernst, J. H. Perkins, V. Augustine and N. Li, ”Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs,” 26th IEEE/ACM Int. Conf. on Auto. SW Eng. (ASE 2011), pp. 23-32, 2011.

[49] B. S. Elenbogen, B. R. Maxim, and C. McDonald, “Yet, more web exercises for learning C++,” in Proc. ACM SIGCSE Bulletin, vol. 32, no.1, pp. 290-294, May. 2000.