Development of a Tool to Analyze Source Code Submitted by Novice Programmers and Provide Learning Support Feedback With Comments

Tatsuyuki Takano, Kanto Gakuin University, Japan
Osamu Miyakawa, Tokyo Denki University, Japan
Takashi Kohama, Tokyo Denki University, Japan

The Asian Conference on Education & International Development 2023
Official Conference Proceedings

Abstract
Novice students make various mistakes in the process of learning computer programming. In courses with more than 100 students, it is difficult to provide accurate and detailed feedback regarding errors in the source code submitted for their assignments. Therefore, we created a source code analyzer and developed a tool to provide detailed feedback to each student. It performs unit tests with misspelled classes and method names. From the results, the tool generates comments, such as "Let us check the method name" or "Let us check the execution result." The tool can generate an average of more than 8,000 Japanese characters per assignment in an actual programming lecture with more than 100 students. In this study, we report on the developed tool, its adaptation to an existing learning management system, and its evaluation.

Keywords: Programming Education, Source Code Analyze, Learning Support Tools

iafor
The International Academic Forum
www.iafor.org

Introduction

Information and Communication Technology (ICT) is playing an increasingly important role in everyday society. Therefore, software development is an important element in constructing information systems, and the training of programming engineers necessary for software development is key.

According to the IPA (Information-technology Promotion Agency of Japan. 2020), the demand for IT personnel remains high, and the Ministry of Education, Culture, Sports, Science, and Technology is promoting education that incorporates programming in elementary, junior high, and high schools (Ministry of Education, Culture, Sports, Science, and Technology. 2021).

Beginner programmers make various mistakes during the learning process. Understanding programming requires practice, which includes making mistakes (Martin, R.C. 2008). It is difficult for educators to identify errors properly when there are many learners.

Therefore, we developed a tool that can generate comments from student-submitted programs, even if the program's method name is misspelled, and can even perform unit testing. This tool is based on the core functionality of our previous system for real-time evaluation of student programs during lectures (Shin Hasegawa, et al., 2011), which is designed to be output in the form of an evaluation input to Manaba, the learning management system (LMS) currently used for lectures at Kanto Gakuin University. The output of the evaluation was designed to match the format of the evaluation input in Manaba. In this study, we report the results of using the tool in a lecture attended by more than 100 students who are beginner programmers and the results of feedback through comments to the students.

Development Tool Overview

The tools were developed in Java and Groovy, a dynamically typed scripting language that runs on Java VM.

The tool uses student submissions, a configuration file of the source code evaluation method, and the source code of the correct answers. The tool has a defined folder structure, and when the tool is executed, the evaluation results are output as CSV, HTML, and XML files; the CSV and XML files are used to read the evaluation result values, and the HTML file is used by the instructor to view the evaluation results. The main evaluation parameters were compilation, indentation, class definition, and unit testing.

Manaba allows student evaluations to be entered into an EXCEL file, which contains columns for each student to enter evaluations and comments, and the tool generates a CSV file from the CSV file of the evaluation results that can be pasted into the columns for each student.

Folder and file organization

When a zip file is unzipped, there are three folders and three files:

- answer folder ・・・ File that will be the correct answer to the execution result
- check folder ・・・ File to set check items
- Test folder: Folder with the teacher's name where student submissions are stored.

- prettify.css prettify.js style.css ・・・ File for html, which is a view of scoring results

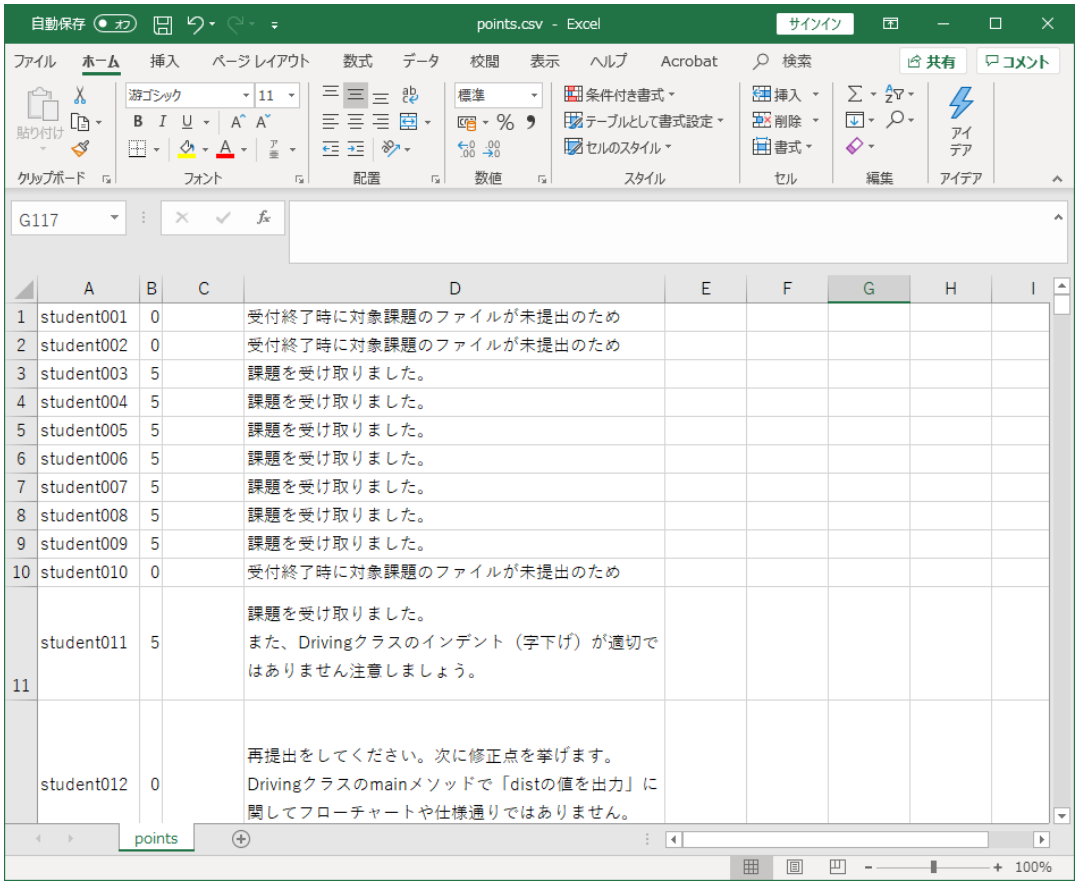
The following folders exist in the tests folder:

- mihon folder ・・・ Correct answer file (same file as "answer")
- mini folder ・・・ Files of specific students extracted when creating the checklist
- teacher folder ・・・ A folder that contains the student's submitted files

The mihon and mini folders were used only for setting up and adjusting the tools. The teacher folder contained each student's folder and the files necessary for grading. The roles of each folder are as follows:

- reportlist.xls ・・・ Registration file to LMS
- points.csv ・・・ Evaluation result file to be pasted into the registration file
- teacher.html ・・・ File for viewing the results of scoring items
- teacher_compile.txt ・・・ Dump file of compilation errors (option)
- CreateMessage.groovy ・・・ Script to output points.csv (option)

Folder and file organization

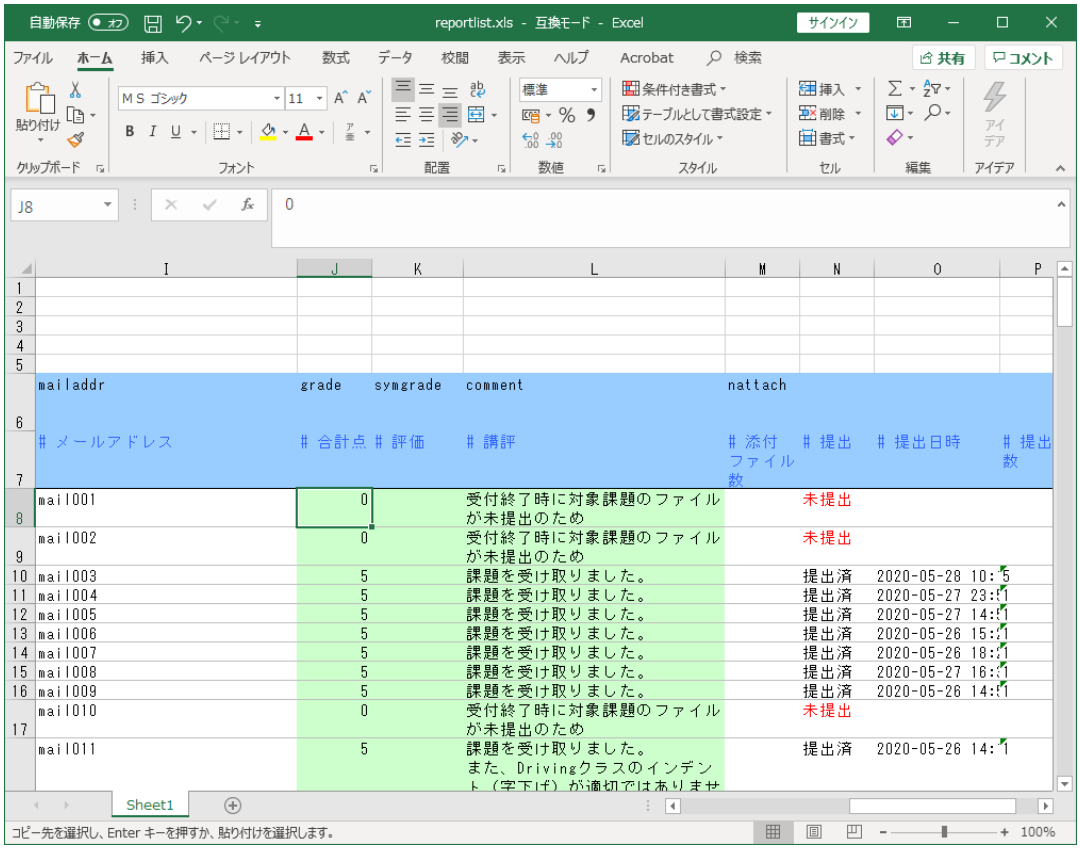


	A	B	C	D
1	student001	0		受付終了時に対象課題のファイルが未提出のため
2	student002	0		受付終了時に対象課題のファイルが未提出のため
3	student003	5		課題を受け取りました。
4	student004	5		課題を受け取りました。
5	student005	5		課題を受け取りました。
6	student006	5		課題を受け取りました。
7	student007	5		課題を受け取りました。
8	student008	5		課題を受け取りました。
9	student009	5		課題を受け取りました。
10	student010	0		受付終了時に対象課題のファイルが未提出のため
11	student011	5		課題を受け取りました。 また、Drivingクラスのインデント（字下げ）が適切ではありません注意しましょう。
	student012	0		再提出をしてください。次に修正点を挙げます。 Drivingクラスのmainメソッドで「distの値を出力」に関してフローチャートや仕様通りではありません。

Figure 1: Example of evaluation result file (points.csv).

To grade the scores, first open points.csv (Figure 1) and reportlist.xls (Figure 2) were used in Microsoft Excel. points.csv opens as follows: Column A contains the student ID number, and columns B–D are the three columns for pasting. Copy these three columns and paste them into columns J to L of reportlist.xls, and paste the total score, evaluation, and critique

according to the "Value" option. (In Manaba, if there is no total score, the evaluation is displayed in the Grades column).



	I	J	K	L	M	N	O	P
6	mailaddr	grade	symgrade	comment	nattach			
7	# メールアドレス	# 合計点	# 評価	# 講評	# 添付ファイル数	# 提出	# 提出日時	# 提出数
8	mail001	0		受付終了時に対象課題のファイルが未提出のため		未提出		
9	mail002	0		受付終了時に対象課題のファイルが未提出のため		未提出		
10	mail003	5		課題を受け取りました。		提出済	2020-05-28 10:5	
11	mail004	5		課題を受け取りました。		提出済	2020-05-27 23:1	
12	mail005	5		課題を受け取りました。		提出済	2020-05-27 14:1	
13	mail006	5		課題を受け取りました。		提出済	2020-05-26 15:1	
14	mail007	5		課題を受け取りました。		提出済	2020-05-26 18:1	
15	mail008	5		課題を受け取りました。		提出済	2020-05-27 16:1	
16	mail009	5		課題を受け取りました。		提出済	2020-05-26 14:1	
17	mail010	0		受付終了時に対象課題のファイルが未提出のため		未提出		
	mail011	5		課題を受け取りました。また、Drivingクラスのインデント（字下げ）が適切ではありませ		提出済	2020-05-26 14:1	

Figure 2: Example of copying and pasting an evaluation into a registration file (reportlist.xls).

The faculty member checks if the rows with the "Not submitted" column are evaluated as "Not submitted." This is a simple method for verifying whether the number of lines output by the tool is correct. After completing the check for "Not Submitted," the teacher should refer to the teacher.html file and adjust the evaluation and critique. This was done to allow the graders to adjust for unexpected patterns in the tool's submission and grading criteria.

Viewing evaluation results via HTML file

The results of the submissions, execution results, and graded items can be viewed by opening a teacher.html file in a browser. The results for each student are arranged according to the number of students in the single-page structure.

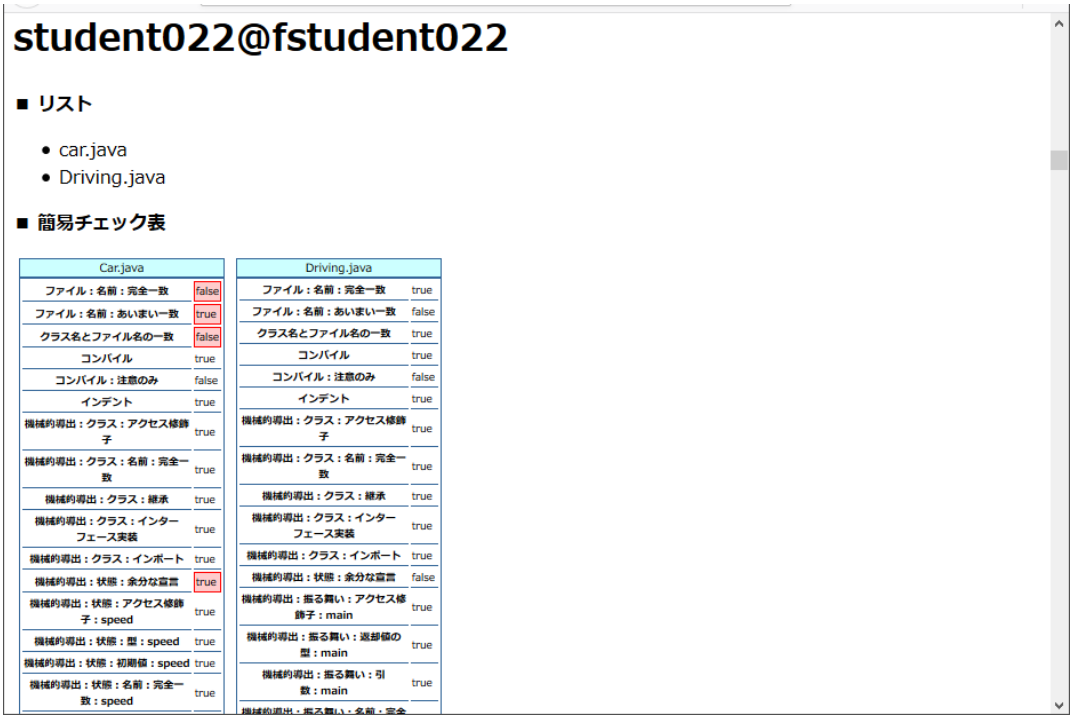


Figure 3: Example of viewing an evaluation in HTML.

Using Figure 3 as an example, "student022@fstudent022" is the name of a student folder. The files under "list" are the files used by the grading tool. Files that do not contain a ". java" or ". txt" extensions are not displayed. The "Simple Check List" shows the true/false status of the check items for each file as true and false. The files with poor results are shown in red. Critical text was generated based on these judgment results, but some items were not used. Some items were displayed in red, and there were cases in which the evaluation was acceptable, even if the background was not entirely white.

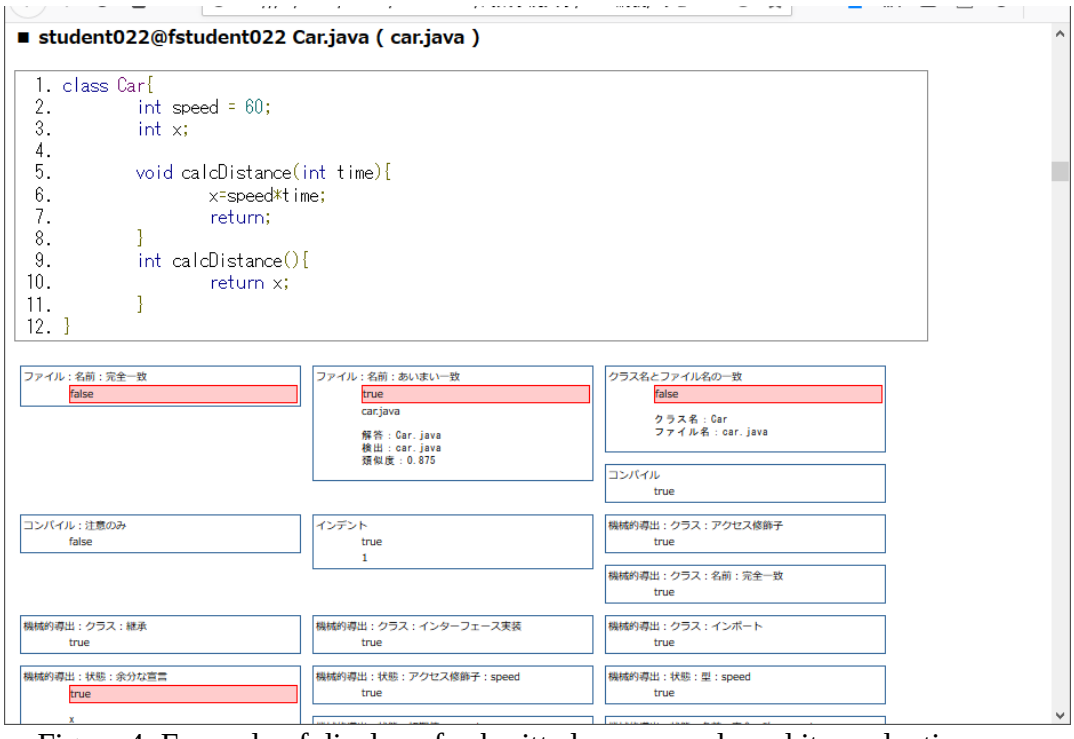


Figure 4: Example of display of submitted source code and its evaluation.

Next, in the source code section of the submission, "Folder name Issue file name (name of the submitted file)" is displayed (Figure 4), and the contents of the submission are displayed below it with line numbers. The file name inside the parentheses is the file of the submission; therefore, in this case, we know that the submission was made with the lower-case name car.java. The filename difference was true for the file "file:name:fuzzy match" indicating that the tool detected a misspelling of the filename. In this case, the generated comment is "There is a misspelling in the filename of the Car class."

Figure 5 shows an example display for each item. The items are roughly classified into "File," "Compile," "Indent," "Class Definition," "Grammar," and "Unit Test." The value for "Indent" indicates the indentation width using one-byte spaces. In the case of tabs, the value is 1.

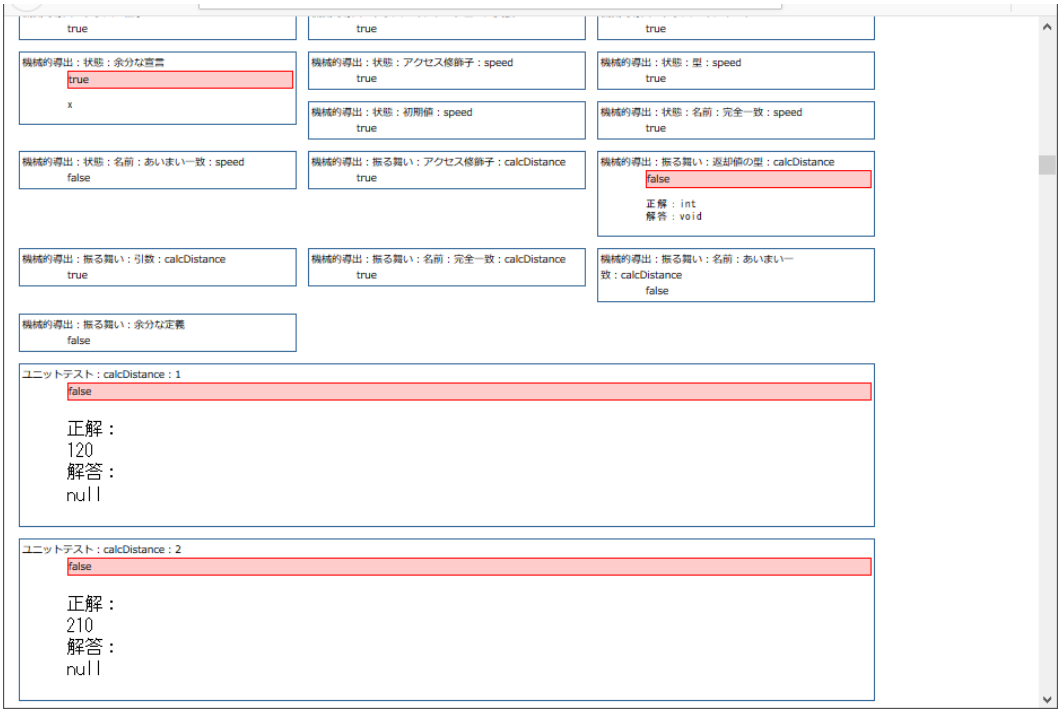


Figure 5: Example of display of each evaluation item.

Class Definition" is a check item for the specifications that form the class framework. The state refers to the field in the class, and behavior refers to the methods and concepts in the class. The results of comparing the type, variable name, etc., with the correct file are displayed.

The unit test compared the execution results with the correct answers. For methods that returned a value, the value for the test was entered as an argument, and the results were displayed to determine whether the results matched. Below "Correct:" is the result of the correct file, and below "Answer:" is the unit test result of the submitted file.

Figure 6: Example of unit test evaluation results.

Classes with a main method have a "Grammar" item, and "Unit Tests" also have a "Similarity" indicator. The "Grammar" item uses regular expressions to check whether the program is written according to the declaration method and flowchart of the variables in the main method described in the assignment.

As shown in Figure 6, the "unit test" of a method in the standard output displays the "similarity" below the correct answer and solution. The similarity is used to allow for ambiguity in judging the presence or absence of white space or double-byte and single-byte characters when questions are submitted on paper. A judgment of 1.0 was regarded as 100% agreement, and the other values are displayed in red. However, because a separate threshold is set for the results at point csv, even a false judgment may result in a pass. The corresponding full-width characters are changed to half-width characters, and the similarity is calculated by the "edit distance of strings (Levenshtein distance)" for strings that exclude all but the necessary white space.

The "unit test" in the main method isolates a specific line for each role of the output (Figure 7) and reflects the judgment in the critique.

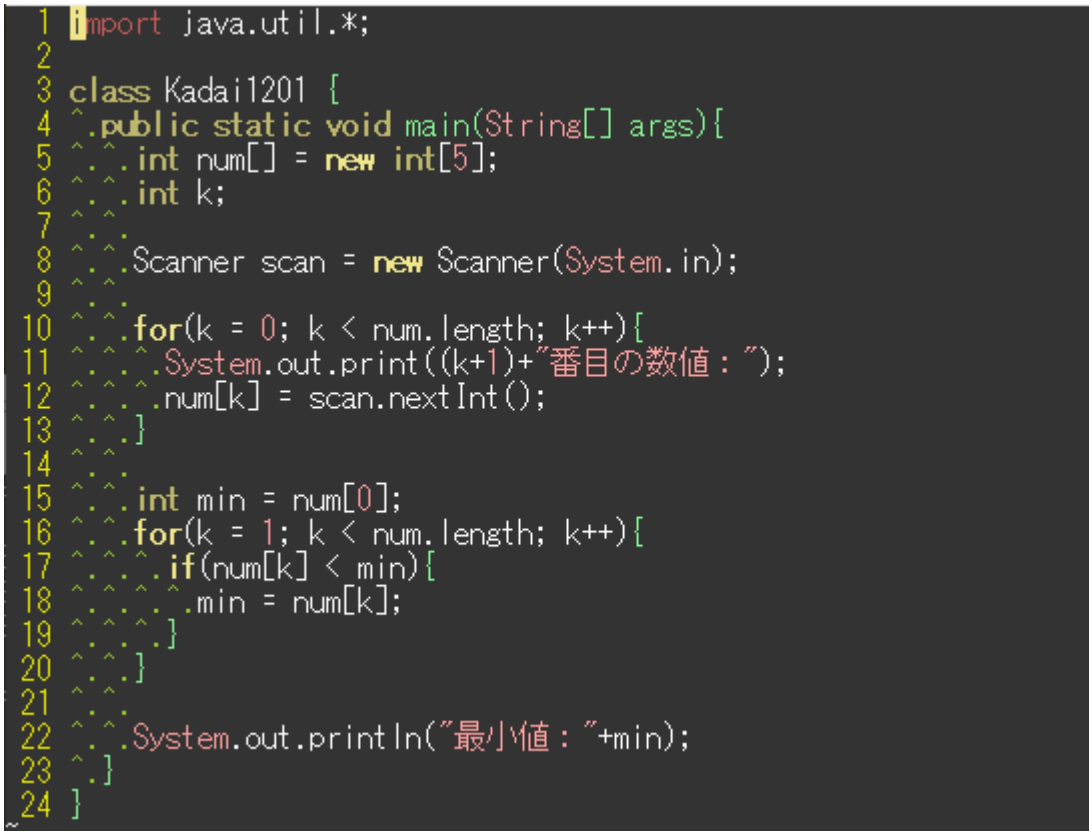
Figure 7: Example of multiple unit test results.

The unit test uses three times the width of the other items and a larger font size to make it easier to identify differences in the text.

Use in lectures by its tools

The tool was used in a lecture for beginner programmers at Kanto Gakuin University's Faculty of Science and Technology to evaluate assignments and provide feedback through generated comments. The lecture was given to 140 students, most of whom were first-year university students. The content of the lecture was based on the fundamentals of structural programming in Java and did not include object-oriented programming.

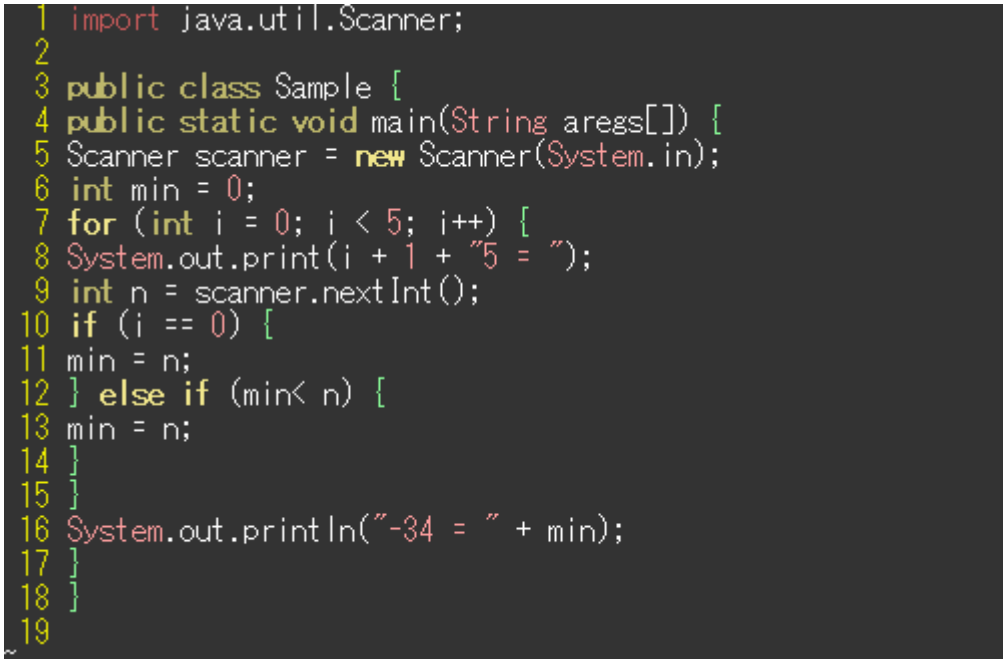
Next, we discuss the comments received while evaluating the submitted source code.



```
import java.util.*;

class Kadai1201 {
^.public static void main(String[] args){
^.^.int num[] = new int[5];
^.^.int k;
^.^.
^.^.Scanner scan = new Scanner(System.in);
^.^.
^.^.for(k = 0; k < num.length; k++){
^.^.^.System.out.print((k+1)+"番目の数値：");
^.^.^.num[k] = scan.nextInt();
^.^.}
^.^.
^.^.int min = num[0];
^.^.for(k = 1; k < num.length; k++){
^.^.^.if(num[k] < min){
^.^.^.^.min = num[k];
^.^.^.}
^.^.}
^.^.
^.^.System.out.println("最小値："+min);
^.}
}
```

Figure 8: Example of Correct Source Code.



```
import java.util.Scanner;

public class Sample {
public static void main(String aregs[]) {
Scanner scanner = new Scanner(System.in);
int min = 0;
for (int i = 0; i < 5; i++) {
System.out.print(i + 1 + "5 = ");
int n = scanner.nextInt();
if (i == 0) {
min = n;
} else if (min< n) {
min = n;
}
}
System.out.println("-34 = " + min);
}
}

```

Figure 9: Example of submitted incorrect source code.

For example, if there is a submitted source code (Figure 8) for this correct answer (Figure 9), feedback with comments such as the following will be generated by the tool.

> *We have received your assignment.*
> *The following points will be noticed here, please refer to them for future study.*
> *Check the class name of the Kadai1201 class.*

Check the main method of the Kadai1201 class for "displaying input from the keyboard.
Let us check the "Display of the minimum value" in the main method of Kadai1201 class.
Note that the indentation of Kadai1201 class is not proper.
The argument of the main method of Kadai1201 class does not seem to be String[] args.

Table 1 shows the number of characters in Japanese used for comments on each assignment and the number of submissions.

Table 1: Number of both words and submissions of comments in each assignment.

Assignment Number	Number of characters (in Japanese) for comments to the entire submitter	Number of submitters
02-1	5392	116
02-2	6966	113
03-1	6534	115
03-2	13750	112
04-1	8958	113
05-1	11663	108
05-2	13649	102
05-3	6437	97
06-1	6441	114
07-1	7697	117
10-1	8091	109
10-2	8137	105
11-1	6670	108
12-1	8144	111
12-2	7454	97
13-1	10961	105
Sum	136944	1742
Ave.	8559	108.875

The feedback from the evaluations and comments was checked by the lecturer in charge of each lecture, and the evaluations were returned to the students via the LMS. There were only a few cases in which the comments made by the tool were incorrect or corrected.

Evaluation by questionnaire to students when using the tool

A questionnaire was sent to the students at the end of the lecture period to evaluate their assessment of assignments using the tool and provide feedback through comments. Eighty-three responses were received.

The results of each question and answer were as follows.

Q1. How accurate are your remarks?
- Very accurate 36% (30)
- Generally accurate 41% (34)
- Fairly accurate 20% (17)
- Generally inaccurate 2% (2)
- Inaccurate 0% (0)

Q2. How detailed were the comments?
- More detailed 18% (15)
- A little finer 24% (20)
- Normal 55% (46)
- A little rougher 1% (1)
- More rough 1% (1)

Q3. How is the readability of the points you made?
- Very easy to read 37% (31)
- Somewhat easy to read 23% (19)
- Normal 28% (23)
- Slightly difficult to read 11% (9)
- Difficult to read 1% (1)

Q4. Is the evaluation criteria consistent and stable throughout each assignment?
- Very stable 39% (32)
- Somewhat stable 33% (27)
- Cannot say either 25% (21)
- Slightly unstable 4% (3)
- Unstable 0% (0)

Q5. Are the points you have made useful for this study?
- Very useful 37% (31)
- Somewhat useful 39% (32)
- Cannot say either way 18% (15)
- Somewhat unhelpful 6% (5)
- Not useful 0% (0)

Q6. Would you like to use a programming learning site with this type of evaluation system in the future?
- I would like to use it by all means 30% (25)
- Somewhat would like to use it 46% (38)
- Cannot say either way 22% (18)
- Somewhat unwilling to use 2% (2)
- I do not want to use it 0% (0)

The results of student evaluations using the questionnaire were positive. The questionnaire also indicated many requests for a learning site separate from the feedback method through the LMS.

Future Outlook

The results of the survey showed that there was a high demand for a learning site, and a system was being developed to make the developed tools available via a web browser. Generally, a Java program requires an execution environment to be installed on a PC or another device to run and evaluate it. However, Doppio (Vilk, John. et el. 2014), which runs on a web browser, enables program execution and evaluation using only a web browser without installing an execution environment on a PC. Currently, we are developing a prototype tool for the simple evaluation of submitted source code using only a web browser.

Conclusions

In this study, we developed a tool to evaluate the source code submitted by students for assignments, using the core functionality of a previous program evaluation system. The tool tests the submitted source code for spelling errors and automatically generates comments for students based on the evaluation results.

The tool has been used in actual assignments for beginner programmers' lectures and has provided feedback with comments of more than 8,000 characters in Japanese for an average of more than 100 submissions each time. The tool was also evaluated using a questionnaire administered to students after the lecture period, and no major problems were found in the comments generated by the tool.

In the future, we will develop an environment in which the tool can be run using only a web browser such that the results of the tool can be used more easily.

Acknowledgements

This study was supported by JSPS KAKENHI (grant number: JP21K02809).

References

Apache Software Foundation. Groovy. https://groovy-lang.org/

Asahi Net, Inc. manaba. https://manaba.jp/

Information-technology Promotion Agency, Japan. *White Paper on IT Human Resources 2020*. https://www.ipa.go.jp/archive/publish/wp-jinzai.html

Ministry of Education, Culture, Sports, Science, and Technology. *Reiwa 3 Annual White Paper of the Ministry of Education, Culture, Sports, Science and Technology*. https://www.mext.go.jp/b_menu/hakusho/html/hpab202001/1420041_00010.htm

Martin, R.C. (2008). *Clean Code: A Handbook of Software Craftsmanship*. 1st edition. Prentice Hall.

Shin Hasegawa, et al. (2011). A Real-time Instruction Support System for Introduction to Computer Programming Education. *Processing Society of Japan*.

Vilk, John. at el.(2014). Doppio: Breaking the Browser Language Barrier. *SIGPLAN Not*. 49, 6, p.508–518. ACM.

Contact email: takano@kanto-gakuin.ac.jp