An Implementation of Element Fill-in-Blank Problems for Code Understanding Study of JavaScript-Based Web-Client Programming

Huiyu Qi, Nobuo Funabiki, Khaing Hsu Wai, Xiqin Lu, Htoo Htoo Sandi Kyaw, and Wen-Chung Kao

***Abstract*—At present, *web-client programming* using *HTML*, *CSS*, and *JavaScript* is essential in web application systems to offer dynamic behaviors in web pages. With rich libraries and short coding features, it becomes common in developing user interfaces. However, the teaching course is not common in universities due to limited time. Therefore, self-study tools are strongly desired to promote it in societies. Previously, we have studied the *programming learning assistant system (PLAS)* as a programming self-study platform. In PLAS, among several types of programming problems, the *element fill-in-blank problem (EFP)* has been implemented for *code understanding* study of C and Java programming. In an EFP instance, the blank elements in a source code should be filled in with the proper words, where the correctness is checked by *string matching*. In this paper, we implement EFP for *web-client programming* in PLAS. In a web page, *HTML* and *CSS* define the components with *tags* in the *document object model (DOM)*, and *JavaScript* offers their dynamic changes with *libraries*, which are blanked in EFP. Besides, a set of web page screenshots are given to help the solution. For evaluations, the generated 21 EFP instances were assigned to 20 master students in Okayama University. By analyzing their solution results, the effectiveness was confirmed for JavaScript programming learning.**

***Index Terms*—Web-client programming, JavaScript, HTML, CSS, element fill-in-blank, code understanding.**

I. Introduction

Nowadays, *computer systems* are offering numerous vital functions to support daily activities of human beings all over the world. They can improve the capabilities in information services, safeties, computations and communications with proper programs that have been made using some programming languages. Then, various programming languages have been invented and adopted to handle different requirements in a variety of target areas and application fields. As a result, it becomes essential to study commonly adopted programming languages to develop and manage efficient computer systems for students or novice engineers in computer science (CS) or information technology (IT) departments/sections in companies or universities.

Among a variety of utilized computer systems in our societies, *web application systems* will be most important by providing significant applications and diverse services using the Internet. Then, the *web-client programming* using *hypertext markup language* (*HTML*), *cascading style sheets* (*CSS*), and *JavaScript* is essential to offer dynamic behaviors in web pages that are made on a web browser in a web application system [1]. With rich hands-on libraries and short coding features, the *web-client programming* has become common in developing sophisticated user interfaces both at personal computers and smartphones.

However, even now, *web-client programming* courses are not opened in many universities over the world, due to curriculum limitations. Before teaching it, more traditional and basic programming languages such as *C* or *Java* should be educated first. Therefore, self-study tools are strongly desired to promote the *web-client programming* in societies.

Previously, we have studied a *programming learning assistant system* (*PLAS*) based on web technologies for self-learning of well-known programming languages, *C*, *C++*, *Java*, *JavaScript*, and *Python*. PLAS provides several types of programming problems that have diverse solving difficulties and learning goals, so that a student can gradually advance the learning stage of PLAS. They include the *grammar-concept understanding problem* (*GUP*) [2], the *value trace problem* (*VTP*) [3]-[5], the *code modification problem* (*CMP*) [6], the *element fill-in- blank problem* (*EFP*) [7], [8], the *code completion problem* (*CCP*) [9], and the *code writing problem* (*CWP*) [10]. For any problem type, the correctness of an answer from a student is checked automatically by *unit testing* in CWP or *string matching* with the stored correct answer in the others.

The outline and learning goal of each problem type are described as follows:

- *GUP* reminds the knowledge and concepts of reserved words and common libraries in the source code, for grammar study.
- *VTP* questions *the* values of important variables and output messages in the source code, for code reading study.
- *CMP demands* to modify the source code to output the requested one, for *library use* study.
- *EFP* requires that the blank elements in the source code should be *filled* with their original words by understanding syntax and semantics for *code understanding* study.
- *CCP* is *different* from *EFP* in that it does not show the missing locations.

Manuscript received February 28, 2022; revised July 6, 2022.

Huiyu Qi, Nobuo Funabiki, Khaing Hsu Wai, and Xiqin Lu are with Okayama University, Japan (e-mail: p06v8z20@s.okayama-u.ac.jp, funabiki@okayama-u.ac.jp, puag507p@s.okayama-u.ac.jp, pch55zhl@s.okayama-u.ac.jp).

Htoo Htoo Sandi Kyaw is with Tokyo University of Agriculture and Technology, Tokyo, Japan (e-mail: htoohtoosk@go.tuat.ac.jp).

Wen-Chung Kao is with National Taiwan Normal University, Taipei, Taiwan (e-mail: jungkao@ntnu.edu.tw).

doi: 10.18178/ijiet.2022.12.11.1736

- *CWP* needs *to* write the source code that can pass the given test code from scratch for *coding* study.

By solving the offered problems in this order, the students are expected to continue programming studies without dropouts and reach sufficient levels. For the *web-client programming*, we have implemented GUP and VTP so far.

In this paper, we implement EFP for *code understanding* study of *JavaScript*-based *web-client programming* in PLAS. In an EFP instance, a source code with several blanks and the screenshots of the corresponding web page are given. Since in a web page, *HTML* and *CSS* define the static components using *tags* in the *document object model (DOM)* and *JavaScript* offers their dynamic changes or actions using *libraries*, they are mainly blanked in the code. It is important to comprehend how to connect the three languages in the source code.

The screenshots of the web pages will help solving the EFP instance. In *web-client programming*, a web page is generated as the user interface to achieve human-computer interactions, running various functions together in the source code. If students read the source code while seeing the web page screenshots, they will understand it more efficiently and know how to use tags and libraries.

For the evaluation of the proposal, we generated 21 EFP instances and assigned them to 20 master students in Okayama University. Although these students have taken *C* and *Java* programming courses in undergraduates, they have not taken *web-client programming* courses. We confirm the validity of the proposal from their high solution results.

Here, we note the advantages of learning JavaScript programming as follows:

1) A source code is relatively short and easy to understand.
2) A student can see the page after the program is executed.
3) It is ready to use camera, video, and other functions with simple codes, which is conducive to learning.

The remaining sections of this paper are organized as follows: Section II discusses related works in literature. Section III presents the element *fill-in-blank* problem (EFP) for *web-client programming*. Section IV describes the evaluation of EFP. Section V summarizes this paper with future works.

II. Related Works

In this section, we discuss related works in literature on *JavaScript* programming educations.

In [11], Arawjo *et al*. proposed an educational game approach called "*Reduct*" to teach core *JavaScript* programming concepts to students, such as functions, Booleans, equality, conditionals, and mapping functions over sets. Progression design and skill acquisition theory are used in the design to support the concept while motivating players to build the correct mental model of the code. The current end goal is to teach basic and complex functional programming levels.

In [12], Appleton *et al*. proposed a prototype system to support students in learning the *web-based* language *JavaScript*. They described the implementation of a portable smart exercise and experiment with it in a web programming course. According to their survey evaluation results, the system could help students learning *JavaScript* programming.

In [13], Uehara *et al*. proposed the *JavaScript Development Environment (JDE)* to support programming education. JDE provides an environment where programming can be displayed anytime and anywhere. In addition, *JDE* can also provide rich code snippet capabilities and allow students writing code with few operations. Since it is a *browser-based* environment, it is suitable for use on the smartphones. *JDE* can edit web pages that contain *HTML*, *CSS* and *JavaScript*.

In [14], Vostinar *et al*. introduced the contribution of an interactive e-learning course as a part of the *Moodle* system for teaching web technologies containing *HTML* and *JavaScript*. This course was proper to teach by using the classic teaching methods or by using the teaching method *EduScrum* with the agile software development method *Scrum* as an alternative.

In [15], Maskelunas *et al*. presented an interactive serious programming game for teaching *JavaScript* programming in their university's introductory course. The game was developed by adopting a gamification pattern-based approach by using the *Technology Acceptance Model (TAM)* and a *Technology Augmented Training Effectiveness Model (TETEM)*. The game was based on visualizations of algorithms with different types, which are explained in the context of city life, and encouraged interactions and pursuit of deeper learning of programming concepts. They presented evaluation results for the game using pretest and post-test knowledge assessments, *TAM* and *TETEM*.

In [16], Suzuki *et al*. proposed a web-based educational support system for JavaScript programming in classroom teaching, called *ClassCode*. It provides an environment in which students can follow tutorials of interactive coding exercises that are intertwined with their learning pace, while teachers can outline how they are learning.

In [17] Zinovieva *et al*. compared different online teaching platforms under certain standards, and selected interesting tasks from the online learning platform named *hackerrank*.com. They explored experiences of using the online coding platform (OCP), and compared the characteristics of different online platforms. They recommended that these platforms should be used for the distance programming learning for future computer scientists and programmers. They also recommended the use of online programming simulators as additional tools for teaching computer science, taking into account functionality.

In [18], La źaro-Carrascosa *et al*. evaluated the subject of master's degree teacher training in Universidad Rey Juan Carlos. Students must learn how to prepare simple web pages by using *HTML*, *CSS*, and JavaScript. Flipped classroom technology was used to present the necessary contents, combined with Aronson's cooperative learning puzzle technique, which is used for the group practice. The results showed that the students were satisfied with their academic achievements.

III. Implementation of Element Fill-in-Blank Problem

In this section, we submit the implementation of the *element fill-in-blank problem (EFP)* for *JavaScript*-based *web-client programming* in PLAS.

A. EFP for Web-Client Programming

In an EFP instance for *web-client programming*, a source code with several blanks and a set of screenshots in the corresponding web page are given to a student. The first screenshot illustrates the web page that is created by the source code composed of *HTML*, *CSS*, and *JavaScript*. The second or other screenshot does the web page that will be created when some input action is taken by a user. The student needs to comprehend the source code while referring to the screenshots of the web pages, and to fill in the blanks with the appropriate words. The correctness of each answer is checked by *string matching* with the original word in the source code.

The design goals of EFP for *web-client programming* are as follows:

1) A beginner who has never studied *JavaScript*-based *web-client programming* can solve the questions in EFP without meeting problems.
2) Various effective source codes are provided to students in a complete form at learning *web-client programming*.
3) A student can learn how to understand the source code including *tags* and *libraries* to create a web page by properly filling in the blanks.
4) The correctness of each answer will be immediately verified and returned to the student.

B. Blank Element Selections for EFP Instance

A source code in *web-client programming* usually composed of three languages of *HTML*, *CSS*, and *JavaScript*. Thus, the following elements in a source code can be blanked in an EFP instance:

- HTML: tag, property, id, output message.
- CSS: selector, id, property, value.
- JavaScript: reserved word, identifier, id, value, library class/method, output message.

It is noted that a *reserved word* is given by the fixed sequence of characters that has been defined in *JavaScript* programming grammar to represent a specific function, and that an *identifier* is given by a sequence of characters that was defined by the code author to represent a variable or a function.

Then, if all the elements for one *id* or *identifier* are blanked, it becomes impossible to fill in them with the original word. For example, *tags* appear in pairs, and if all are blanked, it will be difficult for students to answer. So the *tag* information cannot be deleted all, and one must be left as a prompt information for students to answer. Additionally, any output message to be blanked must appear in a given screenshot, so the message can be completely eliminated, and students can answer according to the screenshot provided on the right side.

C. EFP Instance Generation Procedure

An EFP instance for *web-client programming* is generated by the following procedure:

1) Select a source code that contains the elements to be studied from a website or a textbook.
2) Collect the necessary screenshots of the web pages by running the source code on the web browser.
3) Select the elements in the source code to be blanked, replace them by the question numbers, and keep the blanked elements as the correct answers. Currently, this step is handled manually. Automatic processing will be carried out in the future work.
4) Combine the instance statement, the source code with blanks, and the correct answers to the blanks into one text file.
5) Use the text files to run the programs and generate *HTML*, *CSS*, and *JavaScript* files for the answer interfaces on the web browser.
6) Add the collected screenshots in the *HTML* file to complete the new EFP instance.

D. Example EFP Instance

In this section, we will use the simple source code of the web page to discuss the details of the EFP instance generation process.

1) Source Code: A proper source code with the appropriate difficulty should be selected as the original code to cover the topics of *web-client programming* using *JavaScript* to be studied. In Fig. 1, the source code implements the counter function. Every time the ”Count” button is clicked, the number in the interface will increase by 1. It is noted that this short source code does not contain *CSS*.

```
1  <html>
2  <head>
3  <meta  charset= " utf-8 ">
4  <title> easy_counter </title>
5  </head>
6  <body>
7  <p> Global variable count </p>
8  <button onclick=" myFunction()"> Count</ button >
9  <p  id= " demo">0 </p>
10 <script>
11   var counter = 0;
12   function  add() {
13     return counter += 1;
14   }
15   function  myFunction() {
16   document.getElementById("demo").innerHTML
           =  add();
17   }
18 </script>
19 </body>
20 </html>
```

Fig. 1. Source code.

2) Answer Interface: Fig. 2 demonstrates the answer interface of this EFP instance. It runs on a web browser and allows and facilitates online and offline use by the students. After running the *JavaScript* program, the answer marking will be processed on the browser. The correct answer is encrypted with *SHA256* to avoid cheating. It helps to improve the fairness of the answer and the effectiveness of the final analysis results.

The left side of the interface shows the source code that contains 11 blanks to be filled in with proper words by the students. On the other hand, the right side shows the screenshots of the original page and the updated page after clicking the button three times, respectively. They are given for references to comprehend the source code in the EFP

instance.

After the student types answers into the input forms, he/she will click the "Answer" button to record the correctness of the answers. If the blank answer is incorrect, the background color of the form will turn red. If correct, it will turn white. Students can submit the answers repeatedly until all of the answers are correct or he/she gives up searching them.

E. Blank Correctness in EFP Instance

In this subsection, we discuss the correctness of the blanks in the EFP instance in terms of the unique correct answers to them.

1) Blank for Text Message: A *text message* should be filled in blanks 1 and 4, because of *"p"*, *"/p"*, and *"/button"* tags. From the screenshot on the right side of the interface, the message must be *Global variable count* for blank 1 and be *Count* for blank 4.

2) Blank for Tag: A *tag* should be filled in blanks 2, 5, and 11. At the end of the second line is the *"/button"* tag. The *"script"* tag and *"button"* tag come in pairs. Therefore, it must be *"button"*, *"script"*, and *"/script"*.

3) Blank for Function: A *function* name should be filled in blank 3, because it exists after *onclick*=. Then it must be *myFunction()*, because the screenshot suggests that the number of button clicking times should be shown there.

4) Blank for Identifier: An identifier name *counter* should be filled in blanks 7 and 10, because the number of button clicking times is counted in *add()* function and is shown in the function *myFunction()*.

5) Blank for id: An *id* name should be filled in blank 9, because the argument of the library method *getElementById* must be an *id* name. Then, it must be *demo*, because it is the only one *id* name defined at line 9 in this source code.

6) Blank for JavaScript Grammar: A *JavaScript grammar* should be filled in blank 6, because the function *add()* is defined here. Then, it must be *function*.

7) Blank for Library Class/Method: A *JavaScript library class* should be filled in blank 8, because the library method *getElementById* appears after it with *"."*. Here, it must be *document*.

IV. Evaluation

In this section, we will assess the EFP for *web-client programming.*

TABLE I: Generated 21 EFP Instances

ID#	topic	# of lines	# of JS lines	# of blanks
1	object1	17	9	5
2	object2	21	13	13
3	changing content1	16	5	6
4	changing content2	22	7	14
5	alert() function	14	5	4
6	changing color	17	5	16
7	Date() function	16	5	10
8	prompt() function(subtraction)	15	8	6
9	prompt() function(add)	15	6	8
10	easy counter	20	9	11
11	click button	15	5	11
12	onmouse over and out	17	8	10
13	setTimeout() function	15	5	7
14	array	18	10	10
15	multiplication calculation	16	6	10
16	change background	30	20	16
17	try catch	24	14	10
18	try catch final (number)	31	19	14
19	try catch final (NaN)	30	18	18
20	custom timer	37	20	11
21	fixed timer	23	10	19

A. Evaluation of EFP Instances

For this evaluation, we generated 21 EFP instances by using the source codes covering basic topics that can help students to understand fundamentals of *web-client programming*. Table I demonstrates the instance topics, the number of all lines and the *JavaScript* (JS) lines in the source code, and the number of the blanks of each instance. We assigned them to 20 master students in Okayama University who have not received any formal *JavaScript* programming courses. Although we did not give any lecture before this assignment, we gave hints on the grammar concepts and usages that may be hard or unfamiliar to them.

B. Solution Results of Individual Instances

First, we analyzed the correct answer rate and solution results of the 21 EFP instances respectively. Figure 3 shows the average correct answer rate for each instance and the total number of the answers that submitted by 20 students. For each instance, the average correct answer rate is 96.25% and the number of average submissions is 63.19. The four instances at ID=2, 12, 14, and 15 gave the full rate of 100%. However, the four instances at ID=8, 11, 20, and 21 gave the rate lower than 92%. We will analyze the reasons of the low rates.

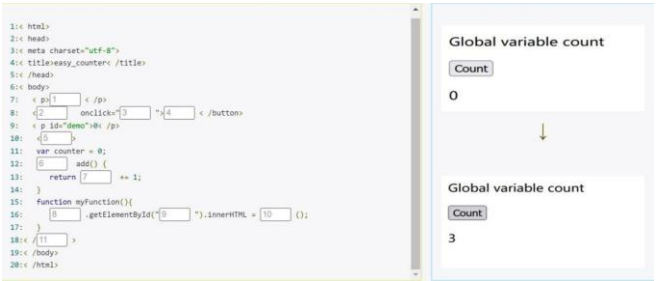

Fig. 2. Interface of ID = 10.

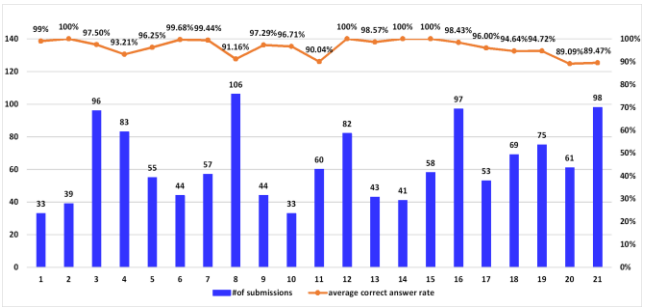

Fig. 3. Solution results for individual instances.

For the instance at ID=8, the correct answer rate is 91.16% and the number of answer submissions is 106. It is the highest among the 21 instances. The difference between two similar

library methods, prompt and alert, may not well be understood. The prompt method provides a message box that allows entering a text message by clicking the ”OK” button.

By clicking the ”Cancel” button, the user can skip it. The alert method provides a message box that only shows the message. By clicking the ”OK” button, the user can close it to continue the operations.

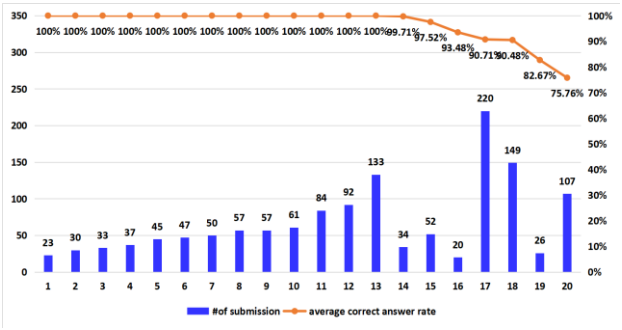

Fig. 4. Solution results for individual student.

For the instance at ID=11, they are 90.04% and 60 respectively. The five blanks in one line ask different roles in the programming on the tags and the text messages in the screenshot for alert, which can confuse the students. It is necessary to understand the grammars in *HTML* and *JavaScript* together.

For the instances at ID=20 and 21, they are 89.09%, 61, and 89.47%, 98 respectively. In both instances, the *setTimeout()* method is used to call a function when the specified time is up. It can be difficult for students to understand it.

C. Solution Results of Individual Students

Subsequently, we analyzed the solution results of 20 students. Figure 4 demonstrates the average correct answer rate and the total number of answer submission numbers for 21 instances by each student. This graph is sorted as descending order of the student performances. For each student, the average correct rate is 96.52% and the average submission time is 63.61. 13 students (65%) achieved the full rate of 100%. Only four students (20%) did the rate lower than 91%. The first rank student answered all of the 21 instances and submitted only 23 submissions, which means there are only two mistakes. These results propose that the generated EFP instances are not difficult for the students, although most of them have not learned *JavaScript* programming before.

V. Conclusion

This paper implemented the *element fill-in-blank problem (EFP)* in *programming learning assistant system (PLAS)* for novice students to learn *JavaScript*-based *web-client programming* by themselves. An EFP instance requires the students to fill in the answers at the blanks in the given source code while referring to the screenshot of the corresponding web page. The correctness of any answer in the blank will be checked through string matching with the correct answer.

For evaluations, 21 EFP instances were generated and assigned to 20 master students of Okayama University who have not received any formal *JavaScript* programming courses before. Their solution results affirmed the effectiveness of our approach.

In future works, we will implement the automatic blank element selection method, and continue to generate EFP instances for studying web-client programming in depth, and assign them to students to evaluate the effectiveness.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

H. Qi mainly conducted the research and wrote the paper. N. Funabiki and W.-C. Kao reviewed and finalized the paper. K. H. Wai and X. Lu analyzed the data. H. H. S. Kyaw collected the source codes. All the authors had approved the final version.

Acknowledgment

We would like to thank the students of Okayama University in Japan, who answered the EFP instances and gave us valuable opinions and suggestions. They are inevitable to complete this paper.

References

[1] What is JavaScript. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First steps/What is JavaScript.

[2] 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. Inform. Tech. (JAIT)*, vol. 12, no. 4, Oct. 2021.

[3] K. K. Zaw, N. Funabiki, 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, Sep. 2015.

[4] S. H. M. Shwe, N. Funabiki, Y. W. Syaifudin, E. E. Htet, H. H. S. Kyaw, P. P. Tar, N. W. Min, T. Myint, H. A. Thant, and W.-C. Kao, “Value trace problems with assisting references for Python programming self-study,” *Int. J. Web Inform. Syst*., June 2021.

[5] X. Lu, N. Funabiki, H. H. S. Kyaw, E. E. Htet, S. L. Aung, and N. K. Dim, “Value trace problems for code reading study in C programming,” *Adv. Sci. Tech. Eng. Syst. J. (ASTESJ)*, vol. 7, no. 1, pp. 14-26, Jan. 2022.

[6] K. H. Wai, N. Funabiki, K. T. Mon, S. H. M. Shwe, H. H. S. Kyaw, and K. S. Lin, ”A proposal of code modification problem for web client programming using JavaScript,” in *Proc. CANDAR*, pp. 196-202, Nov. 2021.

[7] 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, May 2017.

[8] H. H. S. Kyaw, N. Funabiki, S. L. Aung, N. K. Dim, and W.-C. Kao, “A study of element fill-in-blank problems for C programming learning assistant system,” *Int. J. Inform. Edu. Tech. (IJIET)*, vol. 11, no. 6, pp. 255-261, June 2021.

[9] 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, Aug. 2020.

[10] 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, Feb. 2013.

[11] I. Arawjo, C.-Y. Wang, A. C. Myers, E. Andersen, and F. Guimbretie`re, ”Teaching programming with gamified semantics,” in *Proc. CHI Conf. Human Fact. Comput. Syst.*, pp. 4911–4923, May 2017.

[12] J. Appleton, “Introducing intelligent exercises to support web application programming students,” in *Proc. ICICTE*, pp. 216-225, 2017.

[13] M. Uehara, "JavaScript development environment for programming education using smartphones," in *Proc. CANDARW*, pp. 272-276, 2019.
[14] P. Vostinar, "Interactive course for JavaScript in LMS Moodle," in *Proc. ICETA*, pp. 810–815, 2019.
[15] J. Swacha, "An interactive serious mobile game for supporting the learning of programming in JavaScript in the context of eco-friendly city management," *Computers*, vol. 9, no. 4, 2020.
[16] R. Suzuki, J. Kato, and K. Yatani, "ClassCode: an interactive teaching and learning environment for programming education in classrooms," arXiv:2001.08194 [cs.CY], Jan. 2020.
[17] I. S. Zinovieva, V. O. Artemchuk, A. V. Iatsyshyn, O. O. Popov, V. O. Kovach, A. V. Iatsyshyn, Y. O. Romanenko, and O. V. Radchenko, "The use of online coding platforms as additional distance tools in programming education," *J. Phys.: Conf. Ser*., vol. 1840, 2021.
[18] C. La źaro-Carrascosaa, I. Herna ń-Losadab, D. Palacios-Alonsoc, and A ´. Vela źquez-Iturbide, "Flipped classroom and Aronson ś puzzle: A combined evaluation in the master ś degree in pre universitary teaching," *Edu. Know. Soc*., vol. 22, 2021.

H. Qi received the B.A. degree in information management and information system from Dalian University of Foreign Languages, China, in 2021.

She is currently a master student in Graduate School of Natural Science and Technology, Okayama University, Japan. Her research interests include educational technology.

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

X. Lu received the B.S. degree in electronic information engineering from Hubei University of Economics, China, in 2017, and received the M.S degree in electronic information systems from Okayama University, Japan, in 2021, respectively. She is currently a Ph.D. student in Graduate School of Natural Science and Technology, Okayama University, Japan. She received the OU Fellowship in 2021. Her research interests include educational technology.

H. H. S. Kyaw received the B.E. and M.E. degrees in information science and technology from University of Technology (Yatanarpon Cyber City), Myamar, in 2015 and 2018, and Ph. D. in information communication engineering from Okayama University, Japan, in 2021, respectively. She is currently an assistant professor in Division of Advanced Information Technology and Computer Science, Tokyo University of Agriculture and Technology, Koganei, Japan. Her research interests include educational technology and web application systems. She is a member of IEICE.

W.-C. Kao received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996, respectively. He was at SoC Technology Center, ITRI, Taiwan, from 1996 to 2000, and at NuCam Corporation, Taiwan, from 2000 to 2004. Since 2004, he has been with National Taiwan Normal University (NTNU), Taipei, Taiwan, where he is currently the research chair professor at Department of Electrical Engineering and the Dean of College of Technology and Engineering. His current research interests include system-on-a-chip (SoC) as well as embedded software design, flexible electrophoretic display, machine vision system, digital camera system, and color imaging science.