# Development and Evaluation of Functions for Elementary/Secondary Programming Education: The Visual Programming Environment "AT"

Nao Kono[1], Hisayoshi Kunimune[1*], Tatsuki Yamamoto[2], Masaaki Niimura[1]
[1] Shinshu University, 4-17-1 Wakasato, Nagano, 3808553 Japan.
[2] Shobi University, 1-1-1 Toyodacho, Kawagoe, Saitama, 3501110 Japan.

* Corresponding author. Tel.:+81-26-2695502; email: kunimune@shinshu-u.ac.jp
Manuscript submitted November 21, 2016; accepted March 8, 2017.
doi: 10.17706/ijeeee.2017.7.1.13-23

**Abstract:** The Japanese government has recently addressed the issue of promoting programming education in elementary/secondary education to develop students' ability to utilize information. The curriculum guidelines for junior high school and high school, published by the Japanese government, also include programming education for understanding measurement/control systems and developing problem-solving ability. This study developed a visual programming environment called AT, which was implemented in programming classes for novices in some universities and a technical college. With the objective of introducing the AT environment into elementary/secondary programming education, this study also developed some functionalities of AT that offer users intuitive operation capabilities on tablet devices for AT's editor. We conducted a survey for evaluating the intuitive operation in an actual classroom, and the usability and comprehensibility of the editor were highly evaluated by the students.

**Key words:** Elementary/secondary education, programming education, tablet PC, visual programming environment.

## 1. Introduction

Some of the authors teach fundamental computer programming with C language to freshmen and sophomores majoring in computer science. The number of students who do not consider algorithms before they write a program in C language has been increasing in recent years. Therefore, we developed a new class for teaching "algorithmic thinking" before conducting a fundamental computer programming class for freshmen [1]. Algorithmic thinking is a problem resolution process that involves decomposing a problem into tasks that can be described in a specified granularity and then constructing these tasks. This class focuses on teaching not a programming language but the process of stepwise decomposition and construction of problems. To specify the granularity of the decomposition, we adopted paradigms of procedural and structured programming to write down tasks. We also needed the tasks to be automatically executable to make it easy to confirm the behavior of descriptions for learners and to validate the correctness of them for teachers. Generally, visual programming (VP) is used for programming without languages.

We developed a Web-based VP environment called "AT" [2] to support the teaching and learning of algorithmic thinking in class. AT provides teachers and learners with an editor for developing and executing

programs with blocks (Fig. 1) and functionalities for supporting class management. The editor works on Flash Player, and users operate it using a mouse and keyboard. The programs developed in AT work in a closed environment in the editor, and the editor shows the status of working programs (running block, the values of variables, and the program's output) to help students to grasp the behavior of the programs.

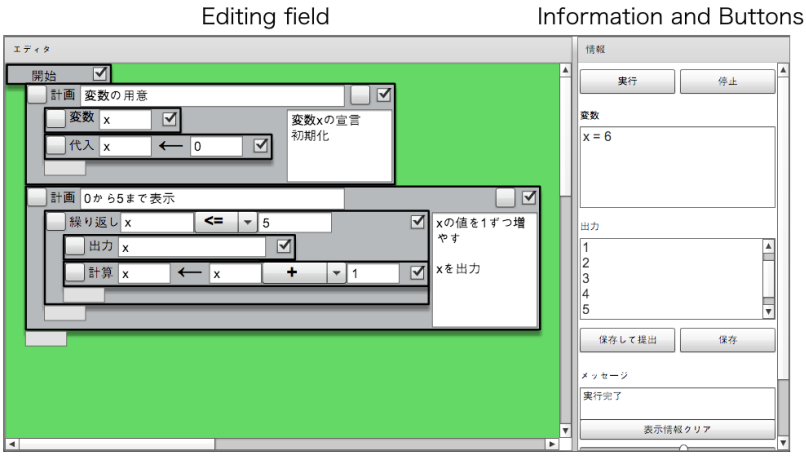



Fig. 1. AT editor.

Wing proposed "computational thinking" in 2006 [3], which became an important skill for all people as well as for people majoring in computer science. Computational thinking consists of the ability to perform decomposition, pattern recognition, abstraction, and algorithm design; algorithmic thinking is a part of computational thinking. The Japanese government has introduced programming education into secondary education in recent years. Its objective is to help students understand the basic structure of measurement and control using a computer, and to help them acquire knowledge and skills for solving problems by making simple programs. The government also intends to introduce programming education into elementary education within a few years.

The objective of this study is to improve AT for elementary and secondary programming education. The requirements for meeting this objective is that AT should work on tablet computers. The user interface of tablet computers is more intuitive than that of the keyboard-and-mouse combination, especially for young people who are familiar with touch panel operation.

This paper describes the improvements in AT that satisfy the aforementioned requirements, evaluates the improved system, and proposes a framework to achieve the requirement.

## 2. Related Works

Jose-Manuel et al. analyzed the benefits and possibilities of VP in elementary education using a VP environment called "Scratch." They also showed that using VP improved students' understanding of programming concepts, logic, and computational practice and provided fun and motivation for students [4]. VP environments like Hopscotch [5], Pyonkee [6], and ScratchJr [7] work on tablet computers. These environments make program development easier and focus on the introduction of programming language for children; however, they do not provide the users stepwise execution and class support that AT offers.

The above-mentioned studies support the notion that introducing VP environments into programming education and touch panel operations on tablet computers for VP is effective especially for children.

This work also introduces a VP environment called "AT" into elementary and secondary education and implements touch panel operations in AT.

## 3. Touch Panel Operations

AT consists of an editor and class support functions, and these are separately implemented (Fig. 2), because the former and the latter run on a Web browser and Web server, respectively. The teacher manages problems (e.g., submission deadline, scope of disclosure, restriction of usable blocks, and initially located blocks), the users, and so on with the support functions.

The AT server sends the data of the problem to a Web browser when a user begins to answer the problem. The user-side Web browser then executes the editor, and the editor reads the data of the problem and changes its available blocks, initially located blocks, etc. The editor sends the program written by the user to the server, and the server saves the program into the database by pressing the save/submit buttons on the editor.

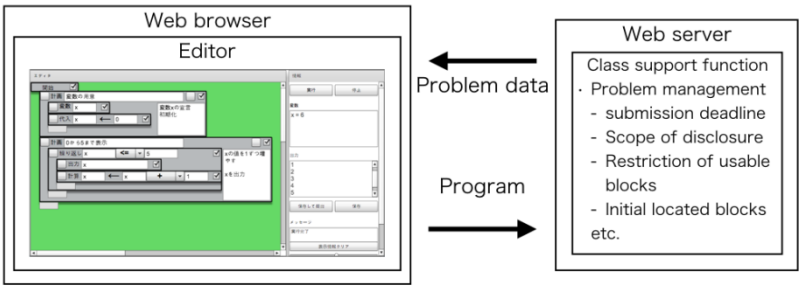



Fig. 2. Architecture of AT.

As mentioned above, the conventional editor works on Flash Player; therefore, it does not work on major operating systems for tablet computers (i.e. iOS and Android). On the other hand, AT is a web-based application, and server-side functions work well on Web servers and browsers. We decided to redesign the editor to work on Web browsers without Flash Player to achieve touch panel operations in tablet computers. To keep compatibility with the conventional editor, the new editor should send and receive data to the server in the same format as the conventional editor. This means that we do not need to change the server-side programs.

In this study, we used the VP editor Blockly (Fig. 3) [8] as the base of the new editor. It is implemented with HTML, SVG, and JavaScript; thus, it works well on almost all Web browsers on PCs and tablet computers. However, the implemented blocks and functionalities of Blockly differ from the conventional editor of AT, and we implemented and modified the functionalities of Blockly as shown below to develop a new editor (Fig. 4) that satisfies the requirement that AT should work on tablet computers.

1) Plan block
2) Variable declaration and scope
3) Stepwise execution
4) Unexecuted blocks
5) Restriction of usable blocks
6) Information display and operation buttons
7) Error hunting
8) Exchanging data on problems

### 3.1. Plan Block

A plan block provides explanations of a processing unit composed of some blocks. These blocks are nested in a block like the one shown in Fig. 5. We implemented a new plan block, because Blockly does not have such blocks. Users can hide/show the nested blocks by clicking a button on the block. Teachers can

hide the nested blocks in the setting of each problem to show programs in various granularities to students. Fig. 5 shows a program described in the smallest granularity in AT, which means that all of the blocks that compose the program are shown. The program shown in Fig. 6 has a larger granularity than the one in Fig. 5, as it hides the nested blocks in the plan blocks. Learners can make an executable program by connecting these plan blocks in the appropriate order.

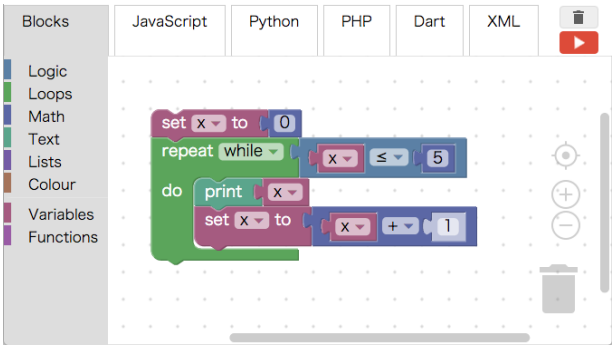



Fig. 3. Blockly.

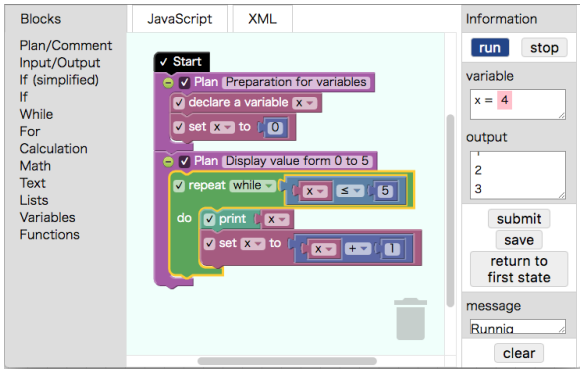



Fig. 4. New editor part.

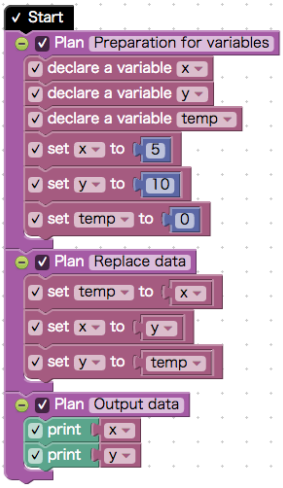



Fig. 5. Expanded plan blocks.

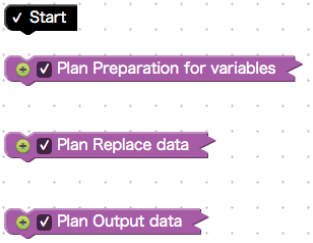
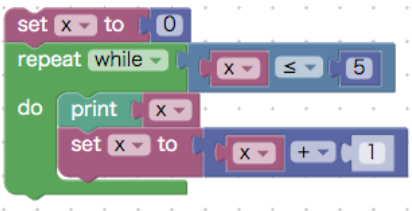



Fig. 6 Collapsed plan blocks.

## 3.2. Variable Declaration and Scope




```
var x;
while(x <= 5){
    window.alert(x);
    x = x + 1;
}
```

Fig. 7. Program generated by Blockly.

We implemented a block for declaring the variable used in a program. Blockly generates a JavaScript program from blocks at run-time. A block assigns a variable a value or refers to the value of a variable, and then, Blockly generates a code to declare the variable (i.e. var x;) at the top of the program as a global variable declaration.

Blockly uses blocks to write the definition of functions and function calls. The conventional AT editor does not support functions; however, we planned to implement functions and variable scope in AT. We also implemented variable scope by generating codes to declare variables in proper functions. Fig. 7 and Fig. 8 show the programs generated by Blockly and the new editor.

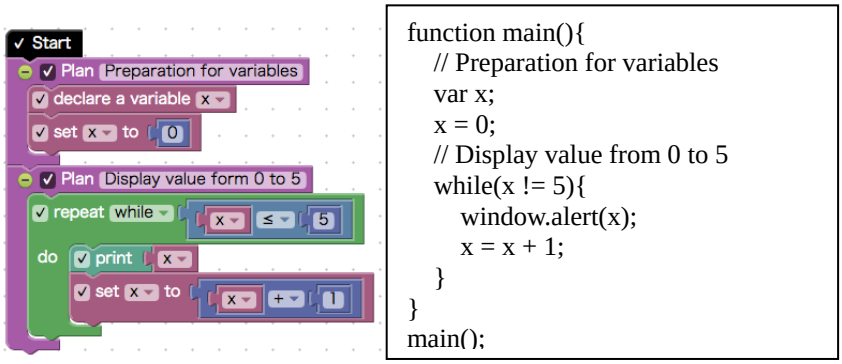



Fig. 8. Program generated by the new editor.

## 3.3. Stepwise Execution

AT offers a stepwise execution of a program to show the running block, value of variables, and history of output as the executing status and behavior of the program to its users. Users can specify breakpoints in the program, and AT can suspend program execution at the breakpoints.

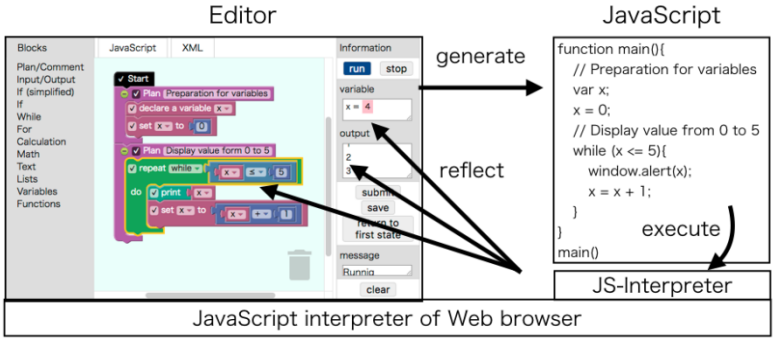



Fig. 9. Architecture of execution.

The JavaScript program that is generated by Blockly is executed on the JavaScript interpreter in Web browsers. Thus, users have to learn to use the debugger embedded in each browser to execute the program in a step-by-step manner. We modified the architecture for executing generated JavaScript programs to achieve stepwise execution. The new architecture includes JS-Interpreter [9] which is a JavaScript interpreter written in JavaScript and works on the interpreter in Web browsers to execute the generated program (Fig. 9). AT grasps and shows the execution status and behavior by referring to the internal status of the JS-Interpreter.

## 3.4. Unexecuted Blocks

AT allows its users to put in unexecuted blocks. This helps them to compare different programs and algorithms (e.g. selection, bubble, and quick sort) side by side. Blockly generates executable codes from all blocks; therefore, we implemented the start block (Fig. 10). The new editor generates an executable program based on only blocks sequentially connected from the start block.

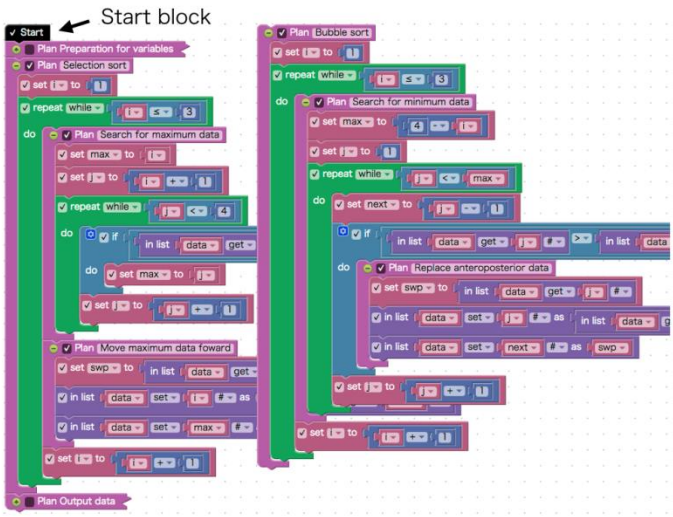



Fig. 10. Comparison of programs.

## 3.5. Restriction of Usable Blocks

AT has a functionality for restricting usable blocks according to the progress of learning to decrease learners' cognitive load. Teachers can configure each type of block as usable/unusable for each problem in the conventional editor of AT. In the new editor, the categories of blocks are as follows:

**Plan/Comment**
- Plan
- Comment

**Input/Output**
- Output value to information display
- Input value to variable

**If**
- If
- Boolean operations
- True/False
- Null
- Conditional operator

**While**
- While
- Break
- Boolean operations
- True/False
- Null
- Conditional operator

**For**
- For (from/to/step)
- Break

**Calculation**
- Number literal
- Arithmetic operation
- Remainder

**Math**
- Mathematical functions (sqrt, abs, ln, log, trigonometric functions, etc.)
- Mathematical constants (pi, e, etc.)
- Functions (is even, is odd, is prime, round, floor, ceil, etc.)
- generating random number (integer/real number)

**Text**
- Text literal

- Functions (get length, is empty, index of, etc.)
- Showing prompt for inputting text

**Lists**
- Generating empty list
- Creating list from literals/repetition
- Functions (get length, is empty, index of, etc.)
- Foreach in list

**Variables**
- Number literal
- Variable declaration
- Assignment
- Reference

**Functions**
- Definition function without return values (procedure)
- Definition function with return values
- Calling procedure/function

Thus, teachers configure each category as usable/unusable for each problem.

## 3.6. Information Display and Operation Buttons

Blockly basically provides an editor for VP and converters for generating programs written in JavaScript and other languages. The conventional editor of AT shows the status of program execution and other information and offers its users buttons for some operations. Blockly does not provide these functionalities; therefore, we implement the following functionalities in the new editor.

The information display and operation buttons are on the right side of the editor. The information display shows the values of variables and the output during the stepwise execution to help students grasp the behavior of the program. When users execute programs, the editor retrieves the value of the variables in the current scope of the JS-Interpreter for each step of program execution to show information. The information display also shows the status of program execution and whether saving/submitting of programs is successfully finished. When students save/submit their programs as the answers to problems and execute/stop the program, they use the operations buttons. Users can confirm the outputs and values of variables during and after execution.

## 3.7. Error Hunting

The conventional editor has functionalities for notifying users of descriptive errors (i.e. misspelled or undeclared names and blank for variable names) in blocks and for not executing the entire program, including such errors. However, Blockly prevents misspelling by offering a drop-down list for choosing variables and blanks and by complementing blanks as a default value (e.g. 0) in generating a JavaScript program. In the new editor, we implement a functionality for hunting the errors ---using undeclared variables and blanks in blocks--- instead of complementing blanks, because general programming languages do not allow such notations (like “var x = ;” and “x = y + ;”):

The new editor confirms variables’ names that are declared, assigned, or referred and searches for blanks in all blocks. It shows an error message in the information display if there are any errors.

## 3.8. Exchanging Data of Problems

AT allows teachers to configure the following for individual problems to support their teaching in classes:

1) Restricting usable blocks
2) Enabling/disabling showing of nested blocks in plan blocks
3) Enabling/disabling use of arrays in variables
4) Switching values of variables as an integer or real number
5) Enabling/disabling the ability to edit/save/submit/execute programs

6) Submission deadline and the scope of disclosure of the task

Configurations (a)–(e) affect the behavior of the editor. The AT server sends the data of these configurations to the conventional editor, and the editor changes its behavior according to the configurations. The editor also sends the data of written programs to the server to save/submit them. The server and editor use specified XML formats to exchange these data.

The new editor also implements functionalities for interpreting data from the server and for generating and sending data to be sent to the server in the same formats.

## 4. Evaluation of New Editor

We introduced AT with the new editor into a class at a university. The students who took the class did not major in computer science. There were 50 students in total. They attended a lecture for 90 minutes once a week, and 15 lectures were held in the class from April 18 to July 25, 2016. The goal of this class was to acquire algorithmic thinking. The students used the conventional editor of AT for several months in another class. After the classes ended, we conducted a questionnaire survey with the students. The purpose of this survey was to compare the usability of conventional and new editors.

### 4.1. Questions

The survey questions asked whether the usability and comprehensibility of the following operations and presentation of information were improved in the new editor. The students answered each question using a five-point Likert scale (5: much better than the conventional editor, 4: better, 3: neutral, 2: worse, 1: much worse), and students who did not use an operation answered "0: not used."

1) Editing program
   - Q1-1 Generating blocks
   - Q1-2 Separation of blocks for referring to variables and numbers from assignment and arithmetic blocks
   - Q1-3 Separation of binary arithmetic block from assignment block
   - Q1-4 Separation of Boolean operation block from IF block
   - Q1-5 Moving blocks
   - Q1-6 Connecting blocks
   - Q1-7 Detaching blocks
   - Q1-8 Deleting blocks
   - Q1-9 Inputting variable name and number
   - Q1-10 Choosing arithmetic (+, -, *, /, %) and Boolean (<, <=, >, >=, ==, !=) operators
   - Q1-11 Adding ELSE-IF and ELSE to IF block
   - Q1-12 Inputting comments on plan block
2) Information display
   - Q2-1 Presentation of variables value
   - Q2-2 Presentation of program's output
   - Q2-3 Presentation of other messages
3) Operation buttons
   - Q3-1 Submitting program
   - Q3-2 Saving program
   - Q3-3 Executing program
   - Q3-4 Stopping execution
   - Q3-5 Clearing displayed information and messages

### 4.2. Survey Results

We obtained 43 valid answers (86 percent of the students) in the survey. Table 1 shows the result of the survey. We eliminated the "0: not used" answers to calculate the average score for each question. Almost all the operations and presentations were highly evaluated, because the average score of each question was more than four points. However, the average score of Q1-2 (3.88) was lower than for other questions. We believe that the reason for the lower evaluation was that the participants had trouble generating and

putting in blocks for referring to variables' values and numbers in the new editor; they had been able to directly input variables' name and numbers into blocks in the conventional editor. However, the separation of these blocks helped them to write expressions with three or more arguments that were unavailable in the conventional editor. This was required in previous experiments with the conventional editor.

Table 1. Results of the Survey

| Question | (5) | (4) | (3) | (2) | (1) | (0) | Ave. |
|---|---|---|---|---|---|---|---|
| Q1-1 | 16 | 21 | 4 | 2 | 0 | 0 | 4.2 |
| Q1-2 | 11 | 21 | 7 | 3 | 1 | 0 | 3.9 |
| Q1-3 | 14 | 20 | 7 | 2 | 0 | 0 | 4.1 |
| Q1-4 | 16 | 21 | 6 | 0 | 0 | 0 | 4.2 |
| Q1-5 | 21 | 16 | 5 | 1 | 0 | 0 | 4.3 |
| Q1-6 | 25 | 14 | 5 | 0 | 0 | 0 | 4.5 |
| Q1-7 | 17 | 17 | 5 | 2 | 2 | 0 | 4.0 |
| Q1-8 | 21 | 16 | 3 | 3 | 0 | 0 | 4.3 |
| Q1-9 | 17 | 20 | 5 | 1 | 0 | 0 | 4.2 |
| Q1-10 | 21 | 17 | 5 | 0 | 0 | 0 | 4.4 |
| Question | (5) | (4) | (3) | (2) | (1) | (0) | Ave. |
| Q1-11 | 16 | 16 | 7 | 3 | 1 | 0 | 4.0 |
| Q1-12 | 12 | 20 | 7 | 1 | 0 | 3 | 4.1 |
| Q2-1 | 16 | 21 | 4 | 2 | 0 | 0 | 4.2 |
| Q2-2 | 14 | 20 | 7 | 1 | 1 | 0 | 4.0 |
| Q2-3 | 15 | 19 | 6 | 3 | 0 | 0 | 4.1 |
| Q3-1 | 24 | 14 | 4 | 1 | 0 | 0 | 4.4 |
| Q3-2 | 24 | 14 | 5 | 0 | 0 | 0 | 4.4 |
| Q3-3 | 20 | 16 | 6 | 1 | 0 | 0 | 4.3 |
| Q3-4 | 16 | 19 | 6 | 2 | 0 | 0 | 4.1 |
| Q3-5 | 16 | 16 | 8 | 1 | 0 | 2 | 4.1 |

We believe that the reason that information display was highly evaluated is that the change in the color of variables' value during program execution to emphasize the behavior of the program is acceptable to students. Moreover, we believe that the reason that operation buttons were highly evaluated is that the change in the background color of the workspace to indicate whether the program was executed or stopped decreases students' confusion. Save and submission buttons are disabled while executing a program, and they are confusing in the conventional editor, because the save or submission functions do not work although they press each button when the program is executing.

Additionally, the new editor shows a popup window to indicate whether save/submission has finished successfully, although the conventional editor shows that only in the information display.

According to these results, we can conclude that the new editor has a better interface for VP than the conventional editor.

The survey also included open-ended questions asking for participants' opinions on AT. We got requests

for an undo function to improve the usability of the editor. We are planning to implement this function.

## 5. Conclusion and Future Work

This paper described the development and evaluation of a new AT editor for introducing AT into elementary/secondary education. According to the results of the survey, the usability and comprehensibility of the new editor were highly evaluated by students; therefore, we can conclude that the new editor is an improvement over the conventional editor.

We will introduce AT into elementary/secondary education and evaluate it in actual classes. We will also continue improving it according to the evaluations by the classes.

## Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP15K01023, JP16H03074, and JP26350284.

## References

[1] Fuwa, Y., Kunimune, H., Kayama, M., Niimura, M., & Miyao, H. (2009). Implementation and evaluation of education for developing algorithmic thinking for students in computer science. *IEICE Technical Report, 109(268)*, 51-56.

[2] Kunimune, H., Yamamoto, T., Kobayashi, K., Kayama, M., & Niimura, M. (2013). Preliminary visual programming education system. *Proceedings of e-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2013* (pp. 2442-2446).

[3] Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49(3)*, 33-35.

[4] Jose, S., Marcos, R., & Esteban, V. (2016). Visual programming languages integrated across the curriculum in elementary school: A two years case study using "Scratch" in five schools. *Computers & Education, 97*, 129-141.

[5] Hopscotch Technologies, Inc. Hopscotch. Retrieved from the website: https://www.gethopscotch.com/

[6] SoftUmeYa, LLC. Pyonkee. Retrieved from the website: https://github.com/SoftUmeYa/Pyonkee

[7] DevTech Research Group at Tufts University & the Lifelong Kindergarten Group at the MIT Media Lab. ScratchJr. Retrieved from the website: https://www.scratchjr.org/

[8] Google. Blockly. Retrieved from the website: https://developers.google.com/blockly/

[9] Fraser, N. Js-interpreter. Retrieved from the website: https://developers.google.com/blockly/installation/js-interpreter

**Nao Kono** is currently a master course student in the Graduate School of Science and Technology, Shinshu University, Japan. He received his B.E. from Shinshu University in 2015. His research areas include fundamental programming education and its support environment.

**Hisayoshi Kunimune** was born in 1976. He received his B.E., M.E. and Ph.D. from Shinshu University in 1998, 2000 and 2003, respectively. He became an assistant professor at Shinshu University in 2004. His current research interest is learning/educational support environments. He is a member of the IEEE, KES International, IEICE, IPSJ, JSET, and JSiSE.

**Tatsuki Yamamoto** is currently a doctor course student in Future University Hakodate, Japan and a part-time lecturer in Shobi University, Japan. She received her M.A. from Tokyo University of Foreign Language in 2006. Her current research interest is educational evaluation. She is a member of the JSET and JSiSE.

**Masaaki Niimura** is currently an associate professor with the Faculty of Engineering, Shinshu University. He received his B.E., M.E. and Ph.D. from Shinshu University in 1988, 1990 and 2002, respectively. His research areas include educational technology and network technology. He is a member of the IEEE, IEICE, IPSJ and JSiSE.