Abstract thinking description system for programming education facilitation

Yasutsuna Matayoshi[1] and Satoshi Nakamura[1]

[1] Meiji University, Nakano, Tokyo, Japan
yasutsuna.matayoshi@gmail.com

Abstract. Programming courses in universities generally teach how to solve problems. However, there are many beginners who fail to make programming well. This is because the beginners cannot be aware of abstract thinking related to the structure of the program, and cannot share abstract thinking with their instructors. In this paper, we propose a method to describe the structure of a program with native language comments in the code tree view and to simply change the structure of the program and the source code by drag-and-drop operation in the code-tree. With this method, beginners can easily organize their thoughts, and instructors can understand the level of understanding and thinking of beginners. We constructed a prototype system using our proposed method. The experiment indicated that it was possible to deepen the understanding of the beginners and utilize it for teaching.

Keywords: Abstract Thinking, Programming Education, Programming UX.

1 Introduction

Courses on programming at universities often have more than 100 students in a class. In such a situation, it is almost impossible for instructors to teach classes according to the levels of understanding of each student. It is common that teaching assistants (TAs) are hired as support for this problem. However, since the number of TAs is limited in most cases, they often have to take care of multiple students at once. For example, in our department, six TAs are hired for supporting 120 students. In addition, every time they are asked questions, they need to read and understand source code their students wrote, which is often difficult to understand as it has variable declarations, variable initializations, description order, function scope, indentation, and functions and there are various ways to solve the problem. These are huge burdens to TAs (see Fig. 1).

On the other hand, many students do not fully understand the contents of the lectures and become frustrated at and find difficulty in programming. In other words, programming education has problems in that TAs have a hard time reading source code their students wrote and that students often find difficulty in programming.

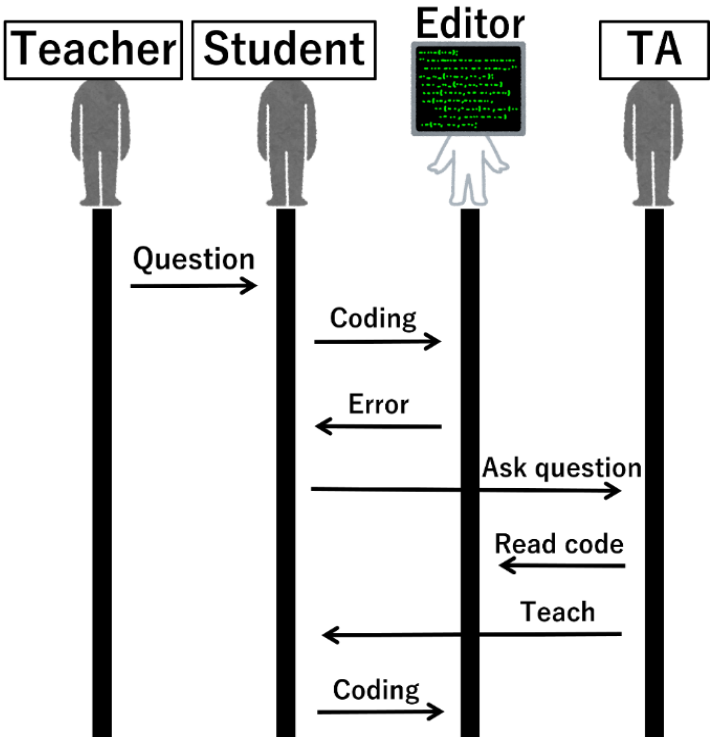

Fig. 1. Teaching flow of programming in the class at the university.

What is important in improving programming skills is to develop the ability to abstract and generalize complex processes. This process is called abstract thinking [1]. This paper focuses on abstract thinking at the time of programming as a solution to the problem in programming education. In general, TAs have excellent programming ability and are trained for abstract thinking. Therefore, if their students' abstract thinking can be directly shared with TAs, their burden to read the students' source code would be reduced. In addition, by trying to incorporate abstract thinking in source code, students can check on their understanding and improve their abstract thinking.

In this study, we propose a method to describe abstract thinking in the code tree and to enable TAs/students to edit the abstract thinking by drag-and-drop operation. We also implement a prototype system to test the usefulness of our method.

2 Related Work

There are many studies on the support for teachers and TAs in universities to give programming lessons to students. Durrheim et al. [2] proposed a system to guide to a correct answer by pointing out difference between a correct answer source code by a

teacher and source code by a student in line units. As a study to improve efficiency of TA, Kim et al. [3] proposed a method to divide programming procedure into six steps and manage the progress of the steps on their proposed system. Since this system can detect students who need assistance from TAs, it enables TAs to provide effective guidance. In addition, Nishida et al. [4] have developed a programming learning environment PEN to describe *xDNCL* that allows expression in Japanese in a short time. Although these systems have similar purposes to this study, they are not designed to facilitate answering questions from students while allowing them to think and understand the structure during class.

There are many proposals of the programming environment to support beginners. Patrice [5] has developed *AlgoTouch*, which is dedicated to algorithms for searching and sorting and designs and executes a program without writing any source code. There are some researches which can be executed as a block, and Kayama et al. [6] proposed a system which can describe an algorithm by creating a structure of a program as a structure block such as a variable declaration block, a calculation block, a conditional branch block, and a repetition block, and having nesting. *AlgoWeb* [7], which can describe the algorithm in a natural language like the proposed system, can describe the structure of the program by the program description language based on Portuguese and can execute and practice the algorithm.

These proposals help beginners to think and understand the structure of programs by visualizing the structure of programs and describing the structure in natural languages. However, they are supposed to be carried out before actual programming, so they are not designed for situations where the structure is being considered while the program is described. In this study, we consider a system to describe the structures in a way that coexists with the programming environment.

Though there are many kinds of research that support teaching and learning of programming, they have not succeeded in simultaneously supporting abstract thinking and programming of beginners. Therefore, in this study, we aim to realize a method to simultaneously support abstract thinking and programming from which both beginners and educators can benefit.

3 Proposal Method

3.1 Abstract thinking and its script

In programming education, beginners first learn the easy level of basic variables, functions and simple arithmetic operations. As they learn, they go on to the more difficult levels of contents with conditional branches, arrays, repeat and creating classes. In addition, they need to be conscious of the goals of the program. In other words, they have to think about the flow of variables, the structure of classes, the decomposition of functions and reuse for efficient source code while writing the code. These ways of thinking are called abstract thinking [1]. In programming education, Abstract thinking is often required when solving advanced problems or new unit problems, so it is important to train this thinking.

A flowchart is often used as a description of abstract thinking. By modeling and designing with flowcharts, we can determine the structure and specifications and write high-quality programs without bugs. However, it is difficult and unrealistic for beginners because they need to learn a unique notation. On the other hand, a comment which can be described directly in the source code of a program can work as a method to describe processing simply. It can also be used easily by beginners because it can be described by a natural language which they use daily. However, there are problems such that it is troublesome to comment, specifications are not decided, and it is insufficient to show the structure.

Pseudocode is known as a notation for simply expressing the structure of a program. Pseudocode is described in both a natural language and source code, and is used to express algorithms. Fig. 2 shows an example of the use of pseudocode for FizzBuzz. However, there is a problem that design errors can occur as the pseudocodes cannot confirm and execute the behavior in a real programming environment. Therefore, it is difficult for beginners to program while describing both pseudocode and source code.

In describing abstract thinking for considering program structure, it is important to comment on source code and to describe the structure of source code in correspondence with a natural language like pseudocode. In this paper, we propose a code tree method which displays the same structure as the actual source code by using natural language comments as tree nodes.

```
Increment x from 1 to 100 by 1
 |
 | If x is a multiple of 3 and 5
 | |
 | | Show FizzBuzz
 | |
 | Otherwise if x is a multiple of 3
 | |
 | | Show Fizz
 | |
 | Otherwise if x is a multiple of 5
 | |
 | | Show Buzz
 | |
 | Otherwise
 | |
 | | Show x
```

Fig. 2. Example of FizzBuzz with pseudocode

3.2 Code Tree

In order to realize abstraction of programs and abstraction-based teaching by TAs, we propose a programming mechanism in which each process of the source code is diagramed as a tree node and the program is abstracted by associating it with the node operations (see Fig. 3). In Fig. 3, the conditional branches in lines 3 to 7 of the right source code correspond to the nodes described as "If x is a multiple of 3 and 5" in the node of the left code tree. There are nesting and parent-child relation in the program, as the node of “Show Fizz” is the child node and “Otherwise if x is a multiple of 3” is the parent node. These nestings are especially important in the structure of the program, and it seems to lead to the understanding of abstract thinking to grasp these relations visually.

By making it possible to handle the code tree technique with the editor of the source code at the same time, it is expected that the understanding of the structure is deepened because of the substitution of nodes and the editing of parent-child relationships are possible with the nesting in mind. In addition, it is possible for educators to check how far beginners recognize the structure of the program by looking at the relationship between nodes of the code tree, which would make communication with students more smoothly than before.

Fig. 3. The relationship between the tree view and source code (ex. FizzBuzz).

4 Prototype System

We developed a prototype system (see Fig.4) with Processing [8], which is used in many universities.

The prototype system was implemented as a Web application. It was implemented using JavaScript and MySQL. The client-side is implemented in JavaScript, with an editor for inputting nested comments and code that can describe abstract thinking. It also edits and executes source code. The server-side is implemented in Node.js and MySQL, and stores the source code inputted by the user and performs syntax checking of the source code inputted.

A code tree showing the structure of the program is displayed on the left side of the system, an editor for writing source code in the center of the system, a processing execution screen on the right side of the system, and a console at the bottom of the system. A button at the top of the system lets you execute, stop, code completion, enable and disable code tree, and save. The code tree and editor are always synchronized, and if you rewrite one, the other is automatically updated. In the code tree, the text of the node can be rewritten, the node can be replaced, the nesting can be deepened, and new nodes can be added by a newline. When you swap nodes in the code tree, the source code is also swapped (see Fig.5 and Fig.6). The editor allows direct editing of source code. However, when the tree structure of the code tree such as "The number of open brackets does not match the number of close brackets" and "lack comments" is broken, the function of the code tree stops and it becomes impossible to change nodes. To solve this problem, it is necessary to work on the cause from the editor based on the displayed error message.

We also published our system on the Web[1].

Fig. 4. A screenshot of the prototype system.

[1] https:// boarditor.nkmr.io/

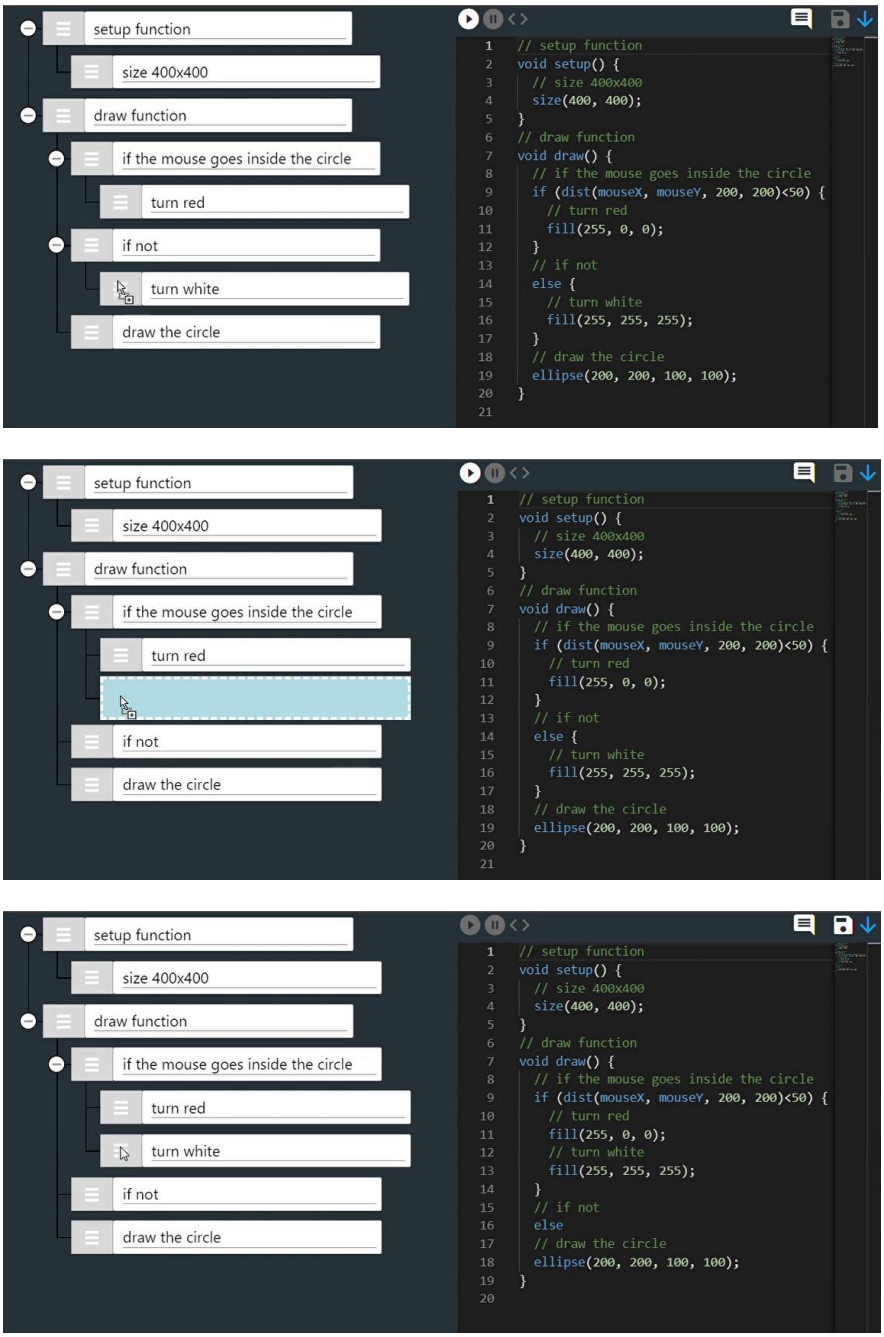

Fig. 5. An example of moving [turn white] node by drag-and-drop operation in the code tree. Then, the system automatically modified the source code.

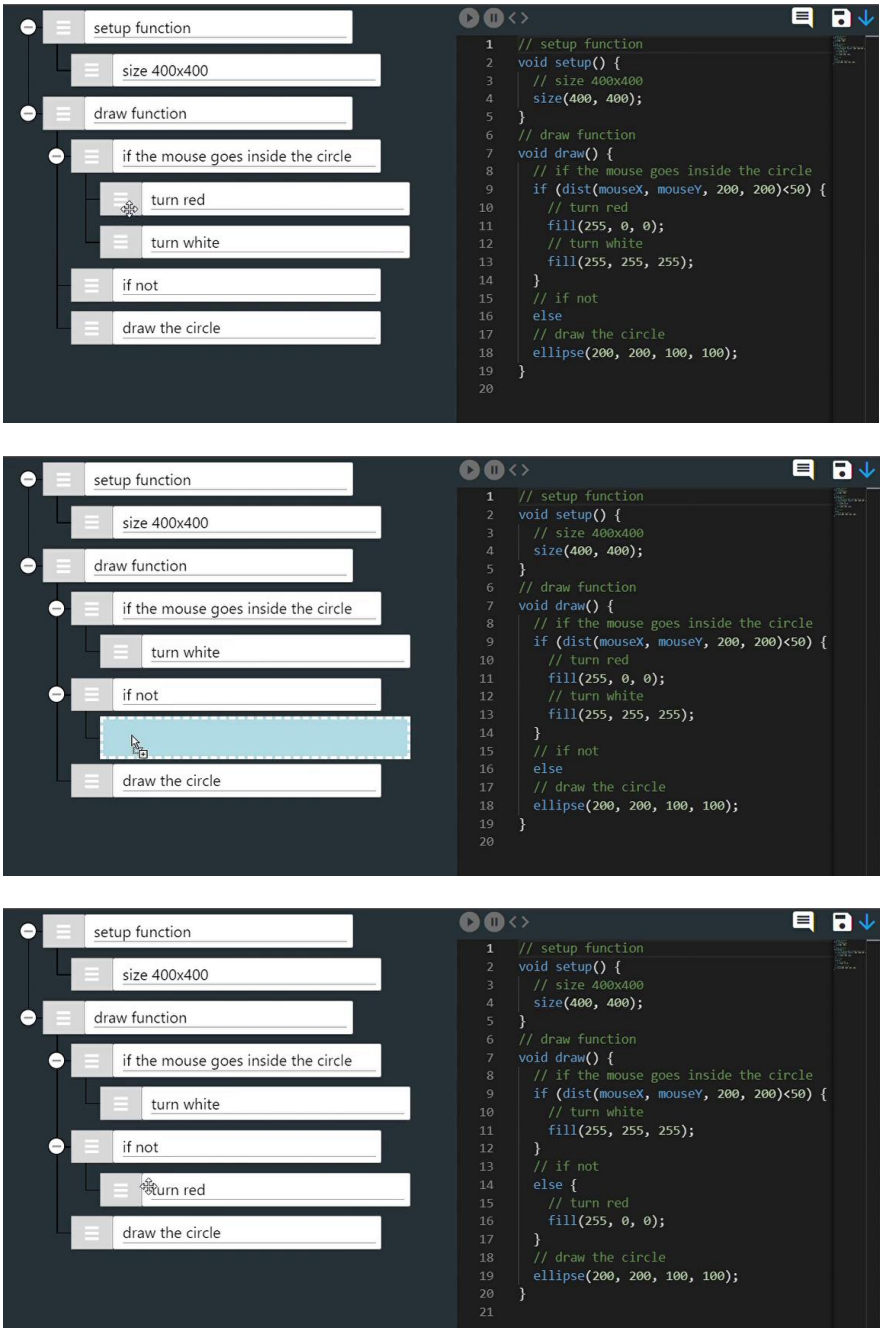

Fig. 6. An example of moving [turn red] node by drag-and-drop operation in the code tree. Then, the system automatically modified the source code.

5 Evaluation Experiment

The purpose of this experiment is to investigate whether students who are beginning to learn programming can use the system to support their learning. In addition, we examine whether the system makes it easier for TAs on the educator side to understand their students' thoughts and intentions in their questions.

5.1 Content of the evaluation experiment

We recruited nine university students whose programming level is a beginner as participants, and three graduate students as TA. We divided the participants into two different groups, one with the prototype system and the other without it. We asked them to solve two tasks with a time limit of 30 minutes per task. In addition, the tutorial was carried out using the prototype system beforehand. It was assumed that writing the source code after considering the structure of the program would promote understanding, so we instructed them to write comments in the code tree before writing the source code. At the end of the experiment, a five-step evaluation questionnaire and a free description questionnaire were conducted. During the experiment, we recorded the programming screen and checked the contents and programming.

5.2 Results of the evaluation experiment

Table 1 shows the results of a questionnaire survey (Q1 - Q7) on a prototype system for students. The distribution evaluated by 5 stages (-2: Not at all well - 2: Very well) is also listed.

Table 1. Questionnaire survey on our system for students.

	Questions	Distribution of evaluation values					Average
		-2	-1	0	1	2	
Q1	Ease of use of the system	0	4	1	2	2	0.2
Q2	Ease of resolving of the system	2	2	2	3	0	-0.3
Q3	Smoothly solved in the system	1	3	1	3	1	0.0
Q4	Deeper understanding of the system	0	1	4	3	1	0.4
Q5	Speed of resolution in the system	2	2	4	0	1	-0.4
Q6	Do you want to continue using the system?	1	2	4	1	1	-0.1
Q7	Do you think you can improve your programming skills by continuing to use the system?	0	1	1	5	2	0.9

Table 1 shows a high average value of 0.9 for Q7. However, in Q2 and Q5, the average value became negative, and the result showed that the existing method was better than the proposed method.

Table 2 shows the average value of each question item obtained by subjectively dividing the students into a group that uses the code tree frequently (four participants) and a group that uses the code tree infrequently (five participants). The result found that the average value was higher in the group which used the code tree more frequently than in the group which used it less frequently.

We received many positive comments from students, such as "It was easy to write comments and understand what to do." and "It was easy because I could operate it intuitively". However, we also received proposals for better UI of the system and comments on other improvements such as "make programs tend to belong", "When it becomes long, it is difficult to correspond source code and comments."

Table 2. An average of each survey result depending on the usage frequency of the code tree.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Group of high frequency in use	1.50	0.50	0.75	0.75	0.50	0.50	1.25
Group of low frequency in use	-0.80	-1.00	-0.60	0.20	-1.20	-0.60	0.60

Table 3 shows the results of a questionnaire survey (Q8 - Q10) of TAs who used the system to answer questions from students. The result evaluated by 5 stages (-2: Not at all well - 2: Very well) is also listed. The result showed that only Q8, which is related to the discovery of mistakes, showed negative values. Q9 (about understanding) and Q10 (about classroom use) were positive.

There were favorable comments from TAs such as "It was easy to judge whether the students were worried about syntax or had difficulty in understanding the problem." "It was easy to check the code." and "The bug was easy to find". On the other hand, there were also negative opinions and reflection points such as "I actually had to see the source code.", "TAs can't fix code tree without understanding system specifications", and "I can't judge because there were few questions."

Table 3. Questionnaire survey for TA.

Questions		Average
Q8	The system is easier for students to spot mistakes.	-0.3
Q9	The system is easier to understand the intent of the student's question.	0.7
Q10	Do you want to use the system in the actual lesson?	1.0

5.3 Consideration of the evaluation experiment

Regarding the comparison between the existing method and the proposed method, the fact that Q4 on understanding the problem and Q7 on continuous use gained positive averaged evaluation values would be because the programming structure of the proposed method can be operated intuitively. On the other hand, the distribution of the evaluation values for Q2, Q3, and Q6, which are related to the ease of solving problems, was widened, and there may be individual differences. Q5 on the speed of the solution of the problem gained a negative evaluation value presumably because it took time to write comments in the proposed method. In addition, as a result of comparing the questionnaire results of two student groups divided by the frequency of code tree use, it was found that the group of high frequency in use gave high evaluation and felt the system favorably. This suggests that the proposed method can support programming learning of the beginners.

As a reason why the questionnaire evaluation Q8 of TA was low, it seemed to be a cause that there were many students who were good at programming and that there were not many questions on programming to TA. In addition, sometimes the system could not output errors due to bug, and thus the TAs could not answer questions from the students. The reason why the evaluation results of Q9 on understanding intentions and Q10 on use of the system in classes were high would be that it was possible to grasp the structure of the program in natural language thanks to the code tree. This suggests that the code tree of the prototype system was useful for TAs.

Observation on how the students used the system for programming from the video recording of the screen, the use of the system was divided into two types of comment preceding description type and source code preceding description type. In the comment preceding description type, the code was written after the program flow was input from the code tree or comments in Japanese. On the other hand, in the source code preceding description type, all comments were written just before the end of the experiment. In the experiment, all participants were instructed to write comments first. However, there ended up being 4 participants of comment preceding description type and 5 participants of source code preceding description type. This corresponded to the group structure of the frequency of use of the code tree, and the students who used the code tree frequently were the comment preceding description type. Based on this observation, it was possible to consider that the participants of the comment precedence description type were able to think the structure of the program by writing comments first using the code tree. Some students wrote detailed comments, noticed mistakes in their source code and corrected the mistakes themselves, and often replaced the nodes. On the other hand, for the source code preceding description type, the evaluation value of Q1 was less than zero, and the question items comparing the existing method and the proposed method were also evaluated low. This may be because they wrote the source code in the editor and then commented, and the code tree stopped and could not be restored because of insufficient comments. Some students didn't use the code tree at all. Based on the above fact, it is considered that the source code precedence type had already acquired the skill to consider the structure of the program by themselves without depending on the

system, and thus felt the system troublesome. This suggests that these participants were not the beginners the proposed method is supposed to support.

5.4 Summary of the evaluation experiment

We examined whether students who started using the prototype system to learn programming could intuitively manipulate the structure of the program and deepen their understanding. Furthermore, we examined whether it is easy for educators and TAs to understand students' understanding and thoughts. The result showed regarding how the participants used the system that there were two types of users, comment preceding description type and source code preceding description type, and it was possible to support the former for their abstract thinking for programming. On the other hand, the latter turned out not to be beginners who the system can benefit, because they had already acquired a skill needed for abstract thinking without using the system. In addition, the result of the questionnaire for TAs indicated a possibility that the system helped educators smoothly understand their students' comprehension and intentions on structures. However, there were few questions from students to TAs during the experiment, so it is not clear how the system affected the educational communication between them. In the next chapter, we investigate the effect of the system on TAs by conducting additional experiments focusing on communication between TAs and students.

6 Additional Experiment

The evaluation experiments revealed that the prototype system can support students' abstract thinking. However, it is unclear how the system affects educators. In this additional experiment, we clarify the effect on TAs by observing the communication between students and TAs.

6.1 Content of the additional experiment

We recruited four university freshmen as students and two graduate students as TA. This experiment was carried out for a long period of time so that the participants become so familiar with the prototype system that they do not have to ask questions about the operation and behavior of the system. We asked the participants to solve four tasks in two days. We set a time limit of 40 minutes per task to solve the problem. Additionally, the problem was made more difficult than the previous experiments in order to increase the communication frequency between the students and the TAs. In addition, we asked the TAs to take care of their students when there was no key input within 1 minute and when they had less than 15 minutes to solve the problem. We also instructed the participants to write comments in the code tree before writing the source code. At the end of the experiment, a free description questionnaire was conducted. During the additional experiment, we recorded the programming screen and confirmed the contents and programming.

6.2 Classification of questions from students

The questions on the communication between students and TAs were first classified into five types: "questions on the solution", "questions on program errors", "questions on understanding the purpose of the problem", "questions on the specification of the problem" and "questions on the specification of the prototype system" (see Table. 4).

Question on the solution is a question asked when students did not know how to write source code for the task or when they did not get the desired results. This type of questions was most frequently asked during the experiment.

Question on program errors is one asked if students did not understand the rules of the program, such as syntax or compilation errors of the program. In the experiment, there were compilation errors due to variable declarations outside the scope and errors due to typing errors in variable names.

Table 4. The number of questions in each category.

Classification of questions	Count
Questions on the solution	11
Questions on program errors	7
Questions without understanding the purpose of the problem	0
Questions on the specification of the problem	3
Questions on the specification of the prototype system	4

Question on understanding the purpose of the problem refers to questions asked when students did not understand at all about the problem of the source code, and it was difficult for them to formulate a way of solving the problem. The system was supposed to work in such a situation. However, this type of question was not asked during the experiment because the students had already taken programming classes and had some knowledge and experience in programming.

Question on the specification of the problem refers to questions on the behavior of the program which was not clear in the problem sentence and for which the TAs only had to give explanation orally without reading the students' source code. In the experiment, a question on the behavior of the click was asked.

Question on the specification of the prototype system refers to questions to ask for solutions for a halt of the code tree on the system or bug of the system, etc. In the experiment, the code tree of the system stopped when the number of nesting did not match.

6.3 Results of the additional experiment

The answers in the questionnaire from the students included much positive feedback such as "I am glad that I could change the order of the source code at once." and "I was able to confirm each action in words, so my understanding was deepened.". However, regarding the support from TAs, there was no comment showing that the system helped it.

In the questionnaire to the TAs, there were positive opinions such as "Although there was a possibility of individual differences, it was easier to answer because the students had more questions than in the actual class." and "I think the students now have a habit of writing comments thanks to this system.". On the other hand, there were also opinions on problems of the system such as "When I was asked a question because I couldn't do certain calculations or processes, I didn't look at the code tree much because I didn't have to look at the entire source code." and "I felt resistance to the number of nodes in the code tree.".

6.4 Consideration of the additional experiment

Among the classified questions, "questions on the solution" were most frequently asked. This would be because it often happens while programming that the source code does not work as expected. The purpose of this study is to help TAs and students to communicate smoothly about how to solve such programming problems. In the experiment, there was no scene of instructing these questions using the code tree. In the interaction between the students and the TAs, sometimes the students were pointing to the editor when asking questions and the TAs' eyes were directed toward the editor. The TAs in such a situation were assumed to provide the same usual instruction as to when they were not using the system. Therefore, it was considered to be necessary for the TAs to be familiar with or instructed about the use of the code tree.

“Question on understanding the purpose of the problem” was also questions the proposed method was supposed to support, but there was no question of this type in the experiment. It is considered that the question was not asked because all students had already taken programming classes.

"Question on program errors" was not assumed to be supported by the present study because it usually comes from syntax error of programming language, and "question on the specification of the problem" and "question on the specification of the prototype system" were also not question classifications on which the purpose of the present study, the visualization of the abstract though, has effects. However, these questions were asked in the experiment. This would be because some parts of the problems were difficult for the students to understand and there were insufficient tutorials for them to get used to the system.

The result of the questionnaire revealed that the students did not have the impression that the system helped them with communication with the TAs, and the TAs felt that the contents of the questions were consistent though there might be individual differences. This may be because the students' thinking was supported as explained in the section 5, so they were able to accurately summarize their questions.

The reason why the code tree was not used when the students asked questions to the TAs would be because they had already taken a year-long programming courses and they did not need to ask questions about the whole program. Moreover, as the problem was made to be difficult and the severe time limit was set, the students had to write shorter comments with little information quantity as the remaining time got shorter. Then, it was impossible for the TAs to understand the whole program by looking at the code tree in this situation, so they probably did not use the code tree for questions.

This experiment could not clarify whether using the prototype system would help TAs teach and understand questions from students because the student participants had mastered the basics of programming and questions to understand the purpose of the problem were not asked. Then, the next chapter introduces a user study carried out to investigate what kind of effect the system has on beginners who just started learning programming.

7 User Study with Beginners

We conducted a user study with beginner students who had only three-month-long experience in programming.

7.1 Content of the user study

We recruited twelve university students as participants and two graduate students as TA. We asked the students to solve two questions about the content of the conditional branch that they had learned in their programming class. They also completed a tutorial to familiarize themselves with the system and completed a free-form questionnaire after the experiment. During the experiment, the video recording of the screen using the system and the video recording of the whole room were carried out, and the communication between the students and the TAs was checked. We also asked the TAs to use the code tree to answer students' questions as much as possible.

7.2 Result of the user study

Many students gave positive opinions such as "It is easy to do as if you are writing notes on the side.", "I write it in Japanese first, so it's good for studying.", and "It is easy to change the order of the program by changing the order of the comments.". On the other hand, there were also negative opinions such as "It was difficult to operate", "The comment function is troublesome.", and "It was difficult to determine how many nodes are in the same tree or the order of the trees".

7.3 Consideration of the user study

In the questionnaire for the student, similar opinions to the ones mentioned in the section 5 and 6 were obtained. In addition, there was an opinion that it was difficult to decide the order of the trees, and it seemed to be a cause for this that the ability to construct the structure of the program, that is, abstract thinking was not yet trained. The recording and the confirmation of the communication between the students and the TAs showed that the student who expressed the opinion above asked a question to a TA who was giving an explanation using the code tree. This fact indicates that the system enabled the student to share their abstract thinking with the TA even though they were not trained enough in it. This suggests that the system can be used to help to teach.

In this user study, the TAs were actually able to easily understand the intention of the question because the number of contents covered in the programming class the students had taken was limited and the problem was simple. Therefore, the question on understanding the purpose of the problem was not asked.

In the user study, it was clarified that the system worked effectively for beginners who had just learned programming. It was also suggested that the system can be utilized for teaching as it facilitates communication between TAs and students who are not trained in abstract thinking by visualizing and sharing abstract thinking.

8 Summary and Future Work

In this paper, we propose a method to visualize abstract thinking in a natural language with a similar structure to source code by making a tree from comments in the source code and to enable TAs and students to edit source code by drag-and-drop operation in the code tree.

In the experiment using the prototype system based on the proposed method, the proposed method worked for the beginner participants and assisted them in considering the structure of the programming.

It was also indicated that the system can be useful for educators because it helped the TAs understand their students' understanding of the programming structure and intentions of their ideas. However, we could not obtain enough evidence to clarify this because there was no question on understanding the purpose of the problem from the students in the additional experiment and the user study.

In the experiment, there was a situation in which a TA unilaterally taught the structure (Required variables, etc.). In such a situation, it is considered that the understanding of the student is deepened by not teaching the answer but making the student read and the flow of the program and develop their ideas. However, it is necessary for TAs to induce correct answers from the structure of the source code their students wrote. We will investigate whether the prototype system can make this easy in our future research. It is also necessary to investigate the effect after the system support such as whether the habit of writing appropriate comments is acquired after the users stopped using the system.

Acknowledgements

This work was supported in part by JST ACCEL Grant Number JPMJAC1602, Japan.

References

1. Kanamori, H., Tomoto, T. and Akakura, T.: Development of a Computer Programming Learning Support System Based on Reading Computer Program. Human Interface and the Management of Information, vol. 8018. pp. 63–69. Springer, Berlin, Heidelberg (2013).

2. Durrheim, S. M., Ade-Ibijola, A. and Ewert, S.: Code Pathfinder: A Stepwise Programming E-Tutor Using Plan Mirroring. Communications in Computer and Information Science, vol. 642. pp. 69–82. Springer, Cham (2016).
3. Kim, S., Kim, W. J., Park, J. and Oh, A.: Elice: An online CS Education Platform to Understand How Students Learn Programming. L@S '16: Proceedings of the Third (2016) ACM Conference on Learning @ Scale. pp. 225–228. ACM, UK (2016).
4. Nishida, T., Nakamura, R., Shuhara, Y., Harada, A., Nakanishi, M., and Matsuura, T.: A Programming Environment for Novices with Visual Block and Textual Interfaces. Journal of International Scientific Publications. Vol. 14. pp. 470–478. (2016).
5. Patrice, F.: A Teaching Assistant for Algorithm Construction. ITiCSE '15: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education. pp. 9–14. ACM, Lithuania (2015).
6. Kayama, M., Satoh, M., Kunimune, H., Niimura, M., Hashimoto, H. and Otani, M.: Algorithmic Thinking Learning Support System with eAssessment Function. 2014 IEEE 14th International Conference on Advanced Learning Technologies. pp. 315–317. IEEE, Athens, Greece (2014).
7. Dorneles, V. R., Picinin, D. J. and Adami, G. A.: ALGOWEB: A Web-Based Environment for Learning Introductory Programming. 2010 10th IEEE International Conference on Advanced Learning Technologies. pp. 83–85. IEEE, Sousse, Tunisia (2010).
8. Processing Homepage, https://processing.org/, last accessed 2020/02/22.