Development of an Automatic Multiple Choice Question Generation System to Promote Understanding of Programming Concepts

Yoshiki Sugihara, Graduate School of Tokyo Denki University, Japan
Tatsuyuki Takano, Kanto Gakuin University, Japan
Takashi Kohama, Tokyo Denki University, Japan
Osamu Miyakawa, Tokyo Denki University, Japan

The Asian Conference on Education 2023
Official Conference Proceedings

Abstract
Gagné, a learning psychologist, defined four hierarchical elements of learning skills: problem-solving, rules, concepts, and discriminations. These intellectual skills form a hierarchical structure with problem-solving at the top, followed by rules, concepts, and discriminations. Mastering lower-order skills is essential for higher-order skill acquisition. We apply this theory to programming education. In programming education, students are sometimes given questions that require them to use a programming language to implement an algorithm or create a program that satisfies a certain specification. Such questions belong to the problem-solving type in the Gagné classification. Problem-solving is at the top of the hierarchy and may be difficult for beginning students to solve from the beginning. Ideally, students should start with simpler problems and progress to more complex ones as their understanding deepens. Programming is based on fundamental "concepts" like variables and methods, and "rules" such as syntax for transforming class diagrams into source code. These two skills are the lower level and are important for mastering programming skills. Hence, we have developed an automated system that generates problems promoting a step-by-step understanding of these concepts. The system marks programming concepts such as variables and methods, and outputs multiple-choice questions in PDF format that require the user to select the option that matches the specified concept. The system generated questions for approximately fifty study items that the authors defined. The system's output was improved by reviewing the problem and incorporating the feedback.

Keywords: Programming Education, Intellectual Skills, Multiple-Choice Question

Introduction

The IT industry has developed remarkably in recent years and the demand for programmers has increased accordingly. In Japan, programming education has become compulsory in elementary and junior high schools since the 2020 academic year, as it allows students to learn logical thinking skills and programming languages (Ministry of Education, Culture, Sports, Science and Technology n.d.). In the high school curriculum, the information subject has been reorganised since the 2022 academic year and students learn the basics of programming through writing programs (Ministry of Education, Culture, Sports, Science, and Technology, n.d.). The environment for teaching IT-related technologies to young people has improved, thus promoting the development of human resources to realise an advanced IT society.

The concept of instructional design is important for creating learning materials. Gagné (1985), a learning psychologist, identified four skills, namely, problem solving, rules, concepts, and discrimination, as elements of a hierarchical structure of learned skills, which he collectively referred to as intellectual skills. In addition, behaviours that are demonstrated after these skills are acquired are shown. Problem-solving is at the top of this hierarchy, followed by rules, concepts, and discrimination. Higher-level skills include several lower-level skills, and the acquisition of higher-level skills requires the mastery of a lower-level skill.

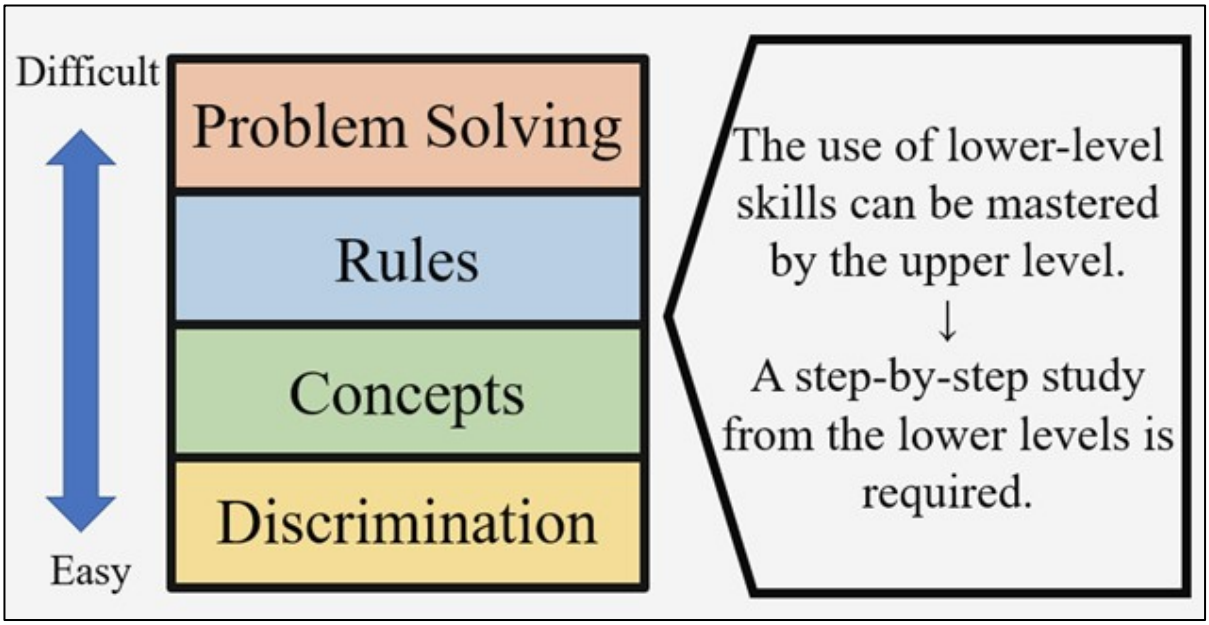

Figure 1: Gagné's intellectual skills and hierarchical structure

This theory has been used to teach programming. Programming tasks, which is typically assigned in programming education, assess the capability of solving problems based on specific requirements and use the concepts and rules acquired in programming. Gagné's cognitive skills enhance the acquisition of problem-solving aptitude. Solving programming problems requires advanced intellectual skills, which renders it difficult for students to solve them while learning and retaining programming concepts in lectures. A better approach would be for students to begin at a lower level and progress to a higher level after they understand the subject better.

Programming comprises primarily ‘concepts’ such as variables and methods, and ‘rules’ such as syntax for creating source code from class diagrams and coding styles. The behaviour acquired by learning ‘concepts’ is ‘identification and classification’. Identification refers to the action of detecting two or more objects that have the same characteristics, whereas classification refers to the action of classifying the objects based on the characteristics of the concept. Understanding unknown ‘concepts’ and ‘rules’ is necessary for acquiring the ability to create programs.

Exercises should be designed that enable students to learn the concepts and rules. Multiple-choice questions are useful in helping students understand the items. However, since the concepts and rules are at a lower level compared with the problem-solving phase, the questions are simpler and thus require more questions during the exercises. The preparation of such questions by the teacher can be a bottleneck in teaching. Therefore, we designed and developed a system that automatically generates questions to promote a stepwise understanding of these concepts and to smoothly shift to the ‘problem-solving’ stage.

Related Studies

Efforts to improve programming education are thriving (Makihara et al., 2016; Taguchi et al., 2007; Ichimura et al., 2013). Many methods have been investigated to automate the creation of selection issues.

Tsumori et al. (2009) analysed methods for generating choice questions adapted to learners’ comprehension statuses. To help learners understand vocabulary with lexical structures, they attempted to generate questions whose difficulty varied depending on to the conceptual similarity of the vocabulary. The comprehension status was determined by the correctness and difficulty of the question. Furthermore, they assumed that the difficulty level was determined by the combination of the source code used for the question and the question choices.

Funaoi et al. (2010) generated multiple-choice questions for physics mechanics using a specific process, which resulted in the solution to the question. The process was classified into three categories, each of which contained a set of error candidates, and an incorrect answer option was generated by replacing one process with a wrong process. This study focused on generating math and physics questions that require computation through a process involving reading the question text, understanding the situation, selecting the knowledge necessary to solve the question, and performing calculations using the acquired knowledge. Programming was performed without calculations, thus allowing automatic generation to be realised by replacing the computational process. In addition, the questions generated included errors caused by factors other than the concepts.

Nagataki et al. (2008) proposed an automatic generation method for introducing error-detecting-type questions into algorithm learning [9]. They attempted to generate two-choice questions using a C program that described the algorithm as a subject. This study focused on learning algorithms.

Several similar programming concepts exist. Therefore, if we use Tsumori et al.’s method to generate questions, we are likely to generate questions from vocabularies with similar concepts, thus resulting in insufficient variation in the difficulty level. In addition, the difficulty level of a question is assumed to vary not only due to the combination of

alternatives but also due to other factors. Methods such as those of Funau et al. and Nagataki et al.'s, which generate questions by modifying section of an algorithm or calculation process, aim to measure one's comprehension of the processing contents and sequence but are unsuitable for confirming one's comprehension of the concepts. We believe that one of the reasons contributing to the difficulty of programming is the presence of many similar basic programming concepts before algorithms, and that programs cannot be created as desired because of confusion. Hence, these concepts must be clearly distinguished and questions that can systematically deepen one's understanding of these concepts must be provided.

In the current study, we focus on the concepts of programming languages and generate questions that require users to select one among multiple options for each concept. The number of options is varied to increase the degree of freedom of the questions that can be generated. The programming topics covered in this study can be learned by novice programmers to create basic programs.

Study Items in Programming Education

In this study, approximately 50 learning items, as shown in Table 1, were extracted from Java, which is a programming language with procedural and object-oriented paradigms. These items are critical for learning programming and essential for creating programs. They were extracted based on the basic elements of Java: variables, methods, data structures, syntax, and object orientation.

Item Name		
Argument	Comment	OperatorLogic
ArrayElement	CompileAndRun	Parameter
ArrayIndex	Constructor	ParameterArgument
ArrayIndexNumber	ForStatement	ParameterType
ArrayListElementType	Getter	PrimitiveType
ArrayType	Indent	ReferenceType
Brackets	InstanceMethod	ReturnType
CallConstructor	InstanceVariable	ReturnValue
CallInstanceVariableMethod	InstanceVariableDeclaration	ReturnValueDefault
ClassCall	Interface	Scope
ClassDiagram	InterfaceMethod	Setter
ClassDiagramToSourceCode	LiteralString	SourceCodeToClassDiagram
CodingStyleClassName	LocalVariable	StaticMethodCall
CodingStyleInstanceVariable	Method	Variable
CodingStyleMethod	ObjectDiagram	VariableSentence
CodingStyleParameter	OperatorAssignment	
CodingStyleVariable	OperatorComparison	

Table 1: List of extracted study items

Variable

A variable is a number or object managed in a program and whose role is rendered explicit via naming. Because Java is classified as a strongly statically typed language, the type information is important. Therefore, to manage the variables, one must understand their types. In addition, the roles of local variables, instance variables, and other variables change depending on the position of the variable declaration; therefore, the differences among them must be elucidated.

Method

A method defines an operation as a single coherent process and can be referred to as a function. Methods such as variables can be used to define methods. Once a method is defined, it can be reused elsewhere in the program; this eliminates the necessity to write the same process repeatedly, thus rendering the program more readable. To use this method, it must be called. At that time, the necessary information may be passed to the method or the information processed in the method may be returned to the calling program. To create and use one's own methods, minimal understanding of the method definitions, invocation methods, arguments, and return values is required.

Data Structure

Several different types of data exist in the program. In addition to numbers and strings, users can use their own data. Occasionally, a single dataset is used. In such cases, data structures are used. Arrays are one of the most widely used data structures and are used to implement other data structures, such as lists and queues. When arrays are available, more advanced programming tasks can be performed.

Syntax

Syntax is a programming language that is recognised by the programming language via a particular method that describes it. For a series of processes to be recognised as a single behaviour, the processes are enclosed in braces to clarify their scope. In Java, some items require brackets to define blocks when describing classes and methods. To express a string literally, it is enclosed with double quotation marks. The statement that expresses the repetition of processing is classified into three section by semicolons: an initialisation section to set the counter, a conditional section to set the repetition duration, and an iteration section to change the value of the counter variable during the repetition. The section enclosed by a slash and an asterisk, or the section after two slashes to a new line, is regarded as a comment and disregarded during compilation. An operator is an element of syntax used to express a mathematical or logical expression or to assign a value to a variable. All of these are defined as elements of the Java syntax, and their incorrect use will result in compilation errors; therefore, correct notations must be utilised.

Object Oriented

Java supports object-oriented paradigms. An object-oriented paradigm is a paradigm in which real-world entities are regarded as objects with attributes and operations in the memory space, and real-world processing is mimicked by messaging between objects. Classes define the data and methods to be managed by a program. Objects with actual data are created by

instantiating them based on the class information. Classes and objects can be represented as diagrams using (unified modelling language). Therefore, smooth system design and development can be realised by converting objects generated in the classes and methods described in the source code into diagrams and by creating source code from diagrams.

Question Format

Various formats exist in which exercises are assigned (Hidano, 1972). These include the essay format, in which students are instructed to express their opinions regarding a topic based on their knowledge, and a format in which students are instructed to fill in the blanks in a text with appropriate keywords. In this section, we examine the appropriate question formats in programming education for developing the ability to classify concepts and apply rules.

Essay/Description Format

In the essay/description format, students craft an answer to a specific topic based on their knowledge and provide a single deliverable. Many introductory textbooks and university lectures require students to write a program that implements an algorithm that satisfies a specification and a control flow that returns an expected result. These questions are categorised into essay/descriptive formats. When answering questions in this format, the learners are expected to apply the knowledge obtained to create a program. As mentioned in Section 1, this format belongs to the hierarchy of problem solving. When performing exercises, learners should attempt questions that consolidate and deepen their understanding at a lower level and then attempt questions such as example questions.

To deepen the learning of concepts, one may benefit from identifying and classifying them. For rules or principles, one may benefit from applying rules using concepts.

Multiple-Choice Format

A multiple-choice format is typically used to answer conceptual questions. This format is highly objective and easy to implement. Furthermore, it allows learners to include many questions in a single exercise because it does not require a significantly amount of time to perform the exercises. We believe that this question format is appropriate because it allows us to assess the learners' ability to apply the rules by providing options in which identification, classification, and rule applications are performed correctly and those in which they are not.

In a multiple-choice format, either single- or multiple-choice answers may be used. In the multiple-choice format, multiple correct answers can be set and the learner must consider the possibility that another option is the correct answer, which can be laborious. However, the single-choice method simplifies the process of solution identification as one correct answer exists.

Formats Considered in Current Study

The multiple-choice format allows beginners to perform the exercises more easily. In addition, it allows the correct answer to a question in the system to be determined more easily as compared with the written form. Therefore, we designed and developed a system that automatically generates single- and multiple-choice questions.

System Details

The system proposed herein uses source code and class diagrams as choices and materials for the questions. Therefore, a Java source code that can be compiled is provided as input for question generation, and choices are generated based on the compiled source code. Information is provided to the system in string form to generate a question for a specified study item. The output is a JSON file that describes the question information to be constructed when generating the questions and a PDF file if the questions are to be saved. The system user can freely specify the input and output directories.

Figure 2 presents an overview of the proposed system. The processing of the system is divided into three main sections. The first is the input process, which processes the information provided as input to enable question generation; the second is the generation process, which generates questions using data such as the number of choices, number of correct choices, and question text; and the third is the output process, which generates a PDF file from the generated questions and a JSON file containing the information attached to the questions.

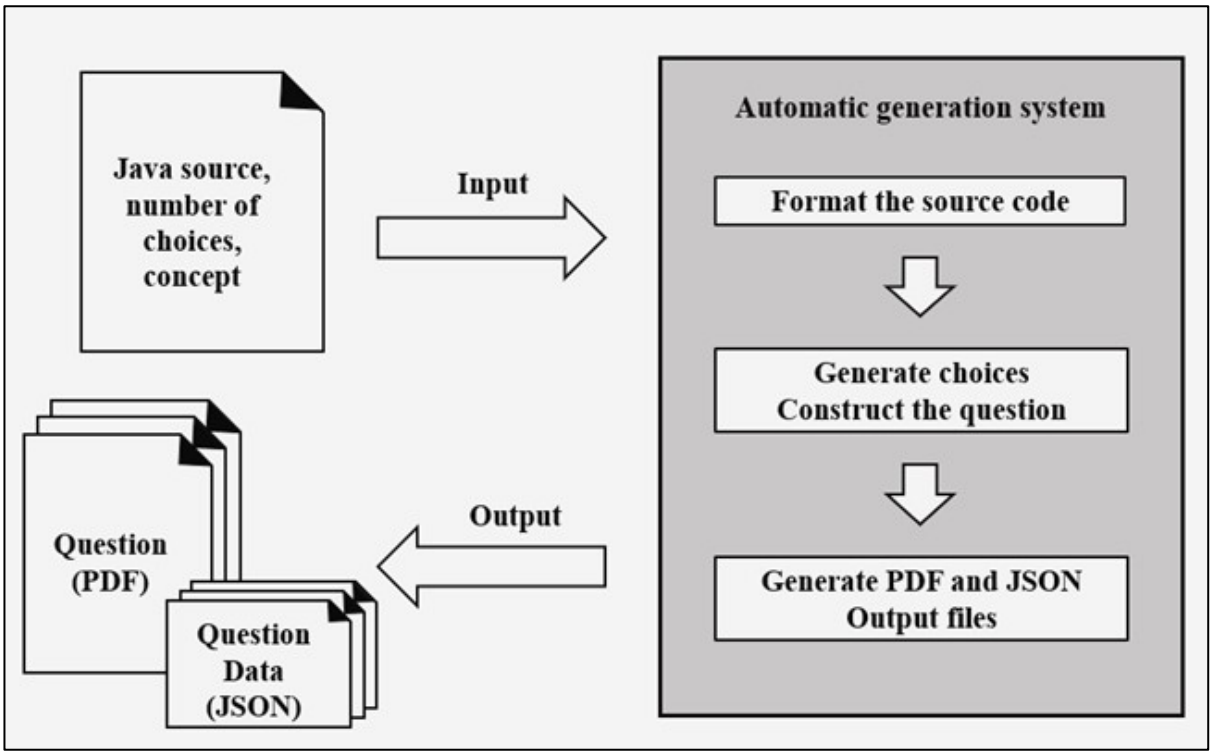

Figure 2: System overview

Input Process

The system uses Java source code and study items as inputs. First, the system determines whether the source code is available for each study item. This verification is performed to avoid cases in which the source code is not available for certain learning items. For example, if no method or constructor is available in the source code for a study item that asks a question regarding a parameter, then the source code cannot be used because a question cannot be generated.

By specifying the directory in which the source code is saved, one can determine whether all internal source codes can be used for the learning item. In this case, a JSON file that shows the mapping between source codes and training items is saved in the directory. This file is rewritten whenever the source code in the directory is added or changed.

The source code used for question generation is formatted after selection to ensure the uniformity of whitespace and line breaks in the source code. Here, the indentation and white space in the source code are standardised.

Generation Process

A well-formed source code is used to perform different generation processes for the different study items. The concepts in the source code are obtained using parsing and regular expressions. Among the concepts obtained, those similar to the target concept are used as candidates for incorrect answer choices. Subsequently, one of the candidates is selected randomly and the source code is shaded or modified such that it becomes grammatically incorrect to generate incorrect answer choices. For questions that use class or object diagrams, the source code is converted such that it can be expressed in these formats; subsequently, shading and text changes are performed.

Figure 3 shows an example of generating alternatives that use the source code. The source code is shaded to indicate the class names, and the source code is modified such that the class names are in lowercase for incorrect choices. In the class diagram, the order in which instance variables are described varies. The generated incorrect answer choices are stored in the list of all choices, and the correct answer choice is inserted at the position of the correct answer number in the list to complete the choices. Using the number of choices, correct answer number, question text, question field, and the source code from which the question is generated, an object representing the question is constructed.

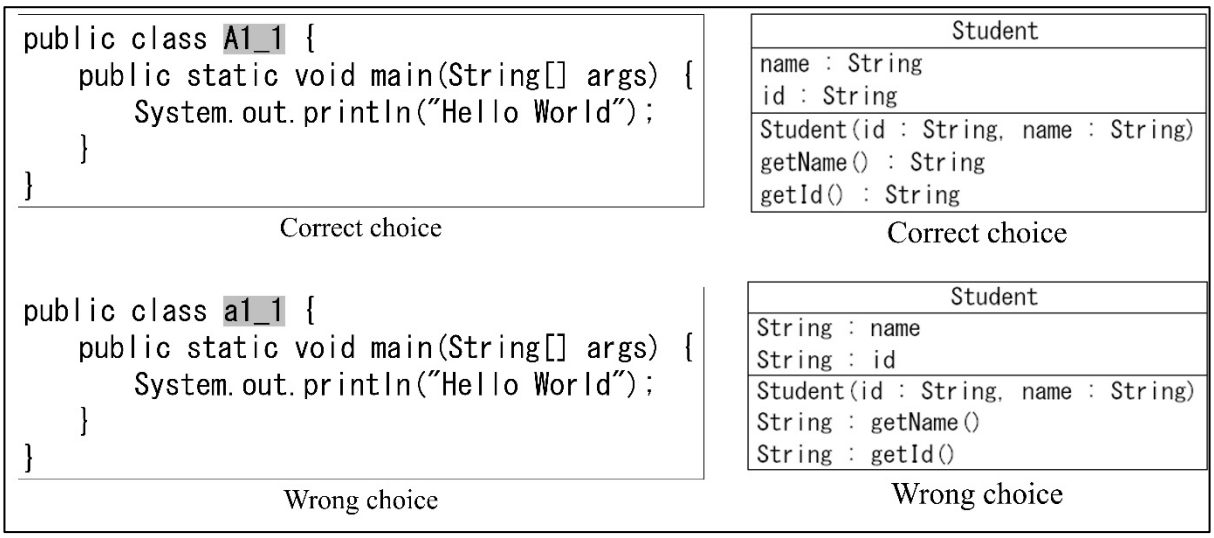

Figure 3: Example of generating choices using source code

Output Process

Based on the question information constructed in the question-generation section, PDF and JSON files containing the question's attached information are generated. Because the content of each study item is different, drawing is implemented for the text, class diagrams, object

diagrams, and source code; subsequently, they are combined to draw the question. Figure 4 shows the selection of the correct class diagram that can be created from the source code.

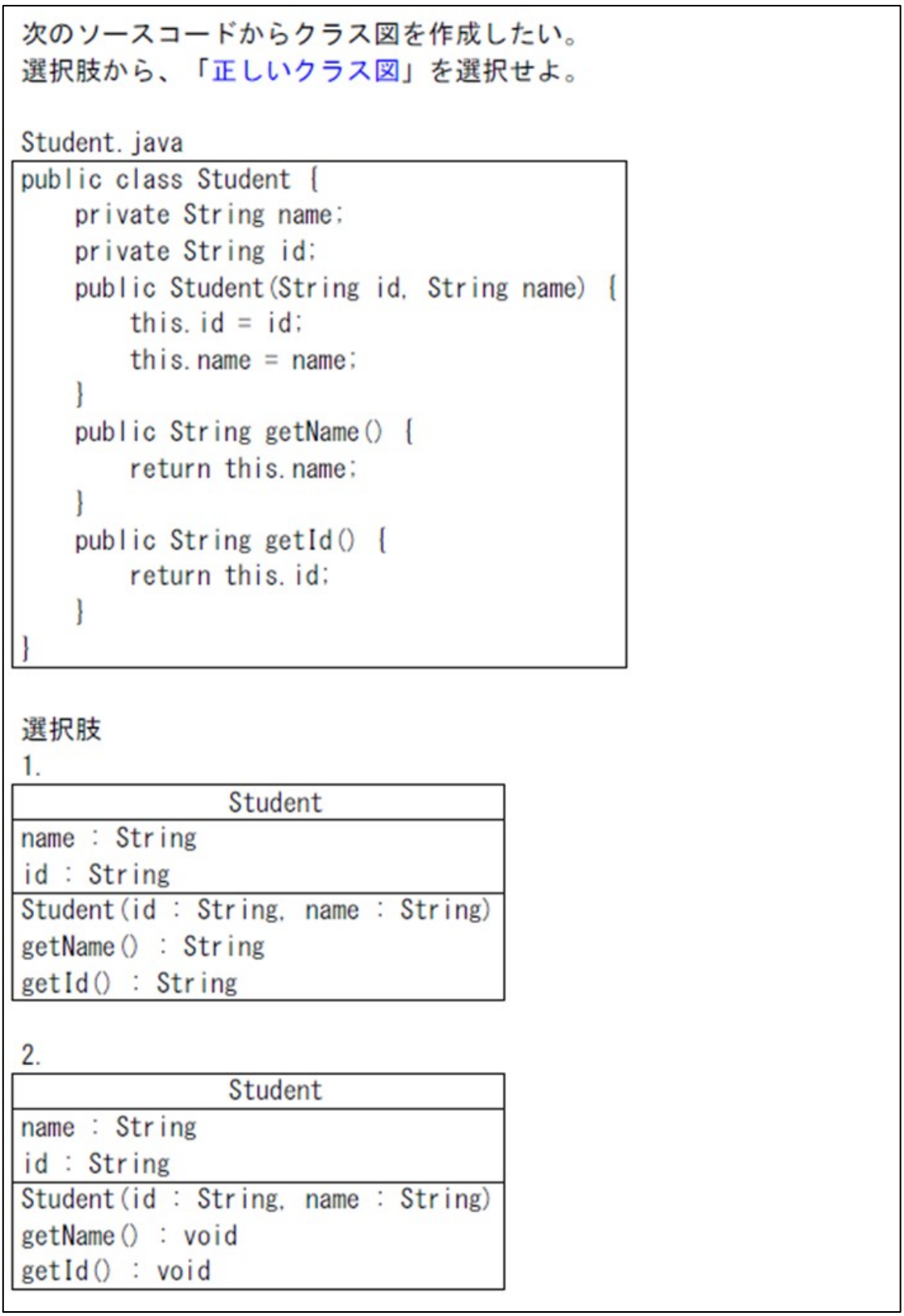

Figure 4: Example of question generated by proposed system

The JSON file contains information regarding the question field, the source code that generated the question, the number of choices, the number of correct answer choices, the source code for the wrong answer choices, and all other changes to the questions. This file is intended for automating the scoring of exercises and regenerating the same questions.

Evaluation

To confirm that the developed system can generate sufficient questions for the exercises, several source codes were used to generate questions. The number of questions generated was calculated for each of the available study items using the following formula, where 'n' is the number of choices, 'a' the number of candidates for the correct choice, and 'b' the number of candidates for the incorrect choice. Questions with the same options in different orders were regarded as separate questions.

$$Number\ of\ questions = n! \times a \times {}_bC_{n-1}$$

The source code used for the generation was created during a programming course at the Tokyo Denki University. Only two-choice questions were posed in this evaluation.

Each of the five source codes was used to generate questions. Table 1 lists the training items and the number of questions that can be generated. As shown, multiple questions can be generated automatically from a single-source code. The number of questions generated increased significantly with the number of concepts in the source code.

The source code used in this experiment constituted a small proportion of the source code created in the lecture—more source codes will be created in practice. Therefore, the number of questions that can be generated increases with the number of source codes created. This implies that a sufficient number of questions can be prepared even when different exercises are provided in each lecture.

Source code	Number of study items	Number of questions
A1_1.java	12	92
Student.java	28	454
Decoration.java	29	1128
Castable.java	7	50
Cup.java	34	5414

Table 2: Result of generating questions from source code

Conclusion

A system that automatically generates programming exercises using source code was proposed in this study. Because this system can generate many practice questions, it is expected to reduce the cost of creating questions. The questions created were reviewed to determine the appropriateness of the question text and options. Several questions appeared to be inappropriate for presentation to students; therefore, they were modified to render them suitable for presentation. However, owing to the numerous questions generated by the system, only the necessary questions to be used in the exercises were determined. To prevent the generation of questions with similar content or completely different difficulty levels, one

must filter the questions by adding conditions at the time of generation, or implement a system that can select questions to be used from those generated.

Furthermore, data obtained from the exercises must be examined to determine whether the exercises in the question format promote a conceptual understanding of programming and thus realise the ability to create programs. Such data are currently being acquired and analysed.

Acknowledgements

This study was supported by JSPS KAKENHI (grant number: JP21K02809). We would like to thank Editage (www.editage.jp) for English language editing.

References

Funaoi, H., Akiyama, M., & Hirashima, T. (2010). Automatic generation of distracters and their comments for a multiple-choice question through a problem solution process. *The IEICE Transactions, J93–D*, 292–302.

Gagné, R. (1985). The Conditions of Learning and Theory of Instruction. (4th Ed.). Wadsworth Pub Co.

Hidano, T. (1972). Psychological Research Methods 7 Test I. University of Tokyo Press.

Ichimura, S., Kajinami, T., & Hirano, H. (2013). A trial to support understanding a programming exercise class. *IPSJ Journal, 54*, 2518-2527.

Makihara, E., Fujiwara, K., Igaki, H., Yoshida, N., & Iida, H. (2016). Pockets: An exploratory programming support environment for introductory programming exercises. *IPSJ Journal, 57*, 236-247.

Ministry of Education, Culture, Sports, Science and Technology. (n.d.). Information Edition: Explanation of the Courses of Study for Senior High Schools (Notification in 2018). https://www.mext.go.jp/content/000166115.pdf

Ministry of Education, Culture, Sports, Science and Technology. (n.d.). Overview Document on Elementary Programming Education. https://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2019/05/21/1416331_001.pdf

Nagataki, H., Itoh, R., Ooshita, F., Kakugawa, H., & Masuzawa, T. (2008). A fault injection method for generating error-correction exercises in algorithm learning. *IPSJ Journal, 49*, 3366-3376.

Taguchi, H., Itoga, H., Mouri, K., Yamamoto, T., & Shimakawa, H. (2007). Programming training of students according to individual understanding and attitude. *IPSJ Journal, 48*, 958-968.

Tsumori, S., & Kaijiri, K. (2009). A method for automatic generation of multiple-choice questions adapted to students' understanding. *Transactions of Japanese Society for Information and Systems in Education, 26*, 240-251.

Contact email: 18aj073@ms.dendai.ac.jp