
Siu Cheung KONG and Bowen LIU

Abstract—Computational Thinking (CT) concepts and practices involve higher-order cognitive skills. Higher-order cognitive skills can be assessed using a performance-based assessment of programming processes. This study presents EasyCode, a performance-based assessment platform for developing CT in students. Distinct from existing methods that rely on the static code analysis of students’ artifacts, EasyCode conducts a dynamic code analysis, enabling the assessment of CT practices throughout the programming process. EasyCode aids student learning by setting performance tasks and providing an online judgment service, which tests the functionality of students’ submitted code and provides immediate feedback. For students with needs, EasyCode provides a support mode with tailor-made toolboxes and decomposed subtasks to help them develop CT concepts and practices. The performance tasks on EasyCode are categorized into three difficulty levels to provide a progressing ladder for students. A sample CT performance task is provided to illustrate how EasyCode facilitates students’ learning of CT concepts and practices.

Index Terms—Computational Concepts, Computational Practices, Computational Thinking, Performance-Based Assessment, Platform

I. INTRODUCTION

Computational Thinking (CT) is a type of analytical thinking that uses fundamental computing concepts and practices to solve problems, design systems and understand human behavior [1]. It is recognized as a skill that can benefit everyone in all walks of life. Researchers and educators agree that kindergarten to grade twelve (K-12) learners should learn CT [2]. Much effort has been devoted to incorporating CT courses into K-12 education curricula. Fuelled by the availability of visual programming environments, CT courses are designed in the content of programming activities that expose students to different dimensions of CT, such as computational concepts (e.g., loops and conditionals) and computational practices (e.g., testing and debugging). Although teaching CT in the context of programming is advantageous because programming is closely related to computation, assessing the learning of CT in programming environments is still a challenge. Reliable and effective assessment is necessary for widely incorporating CT courses into K-12 school education at scale.

CT assessment plays an important role in K-12 CT education. It provides students with feedback throughout the learning process, and, if effective, informs students how well they understood a concept and mastered a skill. It also provides teachers with information on the extent to which learning goals are being achieved.

Nevertheless, CT assessment is challenging. CT not only involves concepts of computer programming, but also higher-order cognitive skills such as testing and debugging, being incremental and iterative, and reusing and remixing [3]. Thus, a single assessment method may not be sufficient to achieve a comprehensive understanding of students’ CT development. Although much effort has been devoted to developing tools and methods for accessing CT, researchers have yet to reach a consensus on a standard assessment [4]. Previous research on the assessment of CT concepts and practices can be grouped into two types: (1) traditional assessment using standardized paper and pencil tests with task-based and multiple choice questions and (2) performance assessment based on static code analysis [5] that relies on assessing learning outcomes by programming artifacts. Some studies [6–8] use both methods to capture a holistic view of students’ understanding of CT. The traditional assessment is designed to test students’ CT concepts, whereas the performance-based assessment is used to evaluate students’ programming artifacts.

Performance assessment based on static code analysis is widely used in CT assessment because of the convenience of collecting students’ artifacts at scale, the accessibility of static analysis, and the availability of automatic assessing tools. However, static code analysis on students’ artifacts cannot maximize the capacity of performance assessment in evaluating the higher-order cognitive skills of CT. Static analysis does not examine the functionality of the codes. Therefore, it does not accurately show whether students constructed CT concepts in their codes. Some CT practices, such as being incremental and iterative, are only observable in the programming process. A static analysis of the final programming products does not show whether students experienced the CT practices. In addition, although existing automatic assessing tools can efficiently conduct static analysis in a very short time, they only provide feedbacks after students finish their coding. They cannot provide support during the programming process.

To address the existing inadequacies of the static analyses of learners’ programming artifacts, this study designs and creates a performance assessment platform that can dynamically assess students programming code and provide them with timely feedback throughout their programming process. This platform can help students develop CT concepts and practices and can allow researchers and teachers to design and administer performance tasks according to their own curriculum. It also provides them with a feasible and convenient way to observe students’ problem-solving process in programming activities.

This paper is organized as follows: First, the authors review the current research on CT assessment. It also examines how performance-based assessment can be used to support students’ CT development through the learning process. Second, the authors present a design of the EasyCode platform, describing its system architecture, main features, and the function of each component. Third, the authors focus on how to design a performance task that can facilitate students’ learning of CT concepts and practices. 

Received July 23, 2020, Accepted August 19, 2020, Published online December 6, 2020.

Siu Cheung Kong, is with the Education University of Hong Kong. He is now with the Department of Mathematics and Information Technology (e-mail: skong@eduhk.hk).

Bowen Liu is with the Centre for Learning, Teaching and Technology, the Education University of Hong Kong.

This work is under Creative Commons CC-BY-NC 3.0 license. For more information, see https://creativecommons.org/licenses/by-nc-nd/3.0/.
in the programming process and help them develop their CT concepts and practices. An example of a CT performance task is provided to illustrate how the platform facilitates the learning of CT concepts and practices. Fourth, the authors summarize the paper and provide recommendations for further work and improvements to the platform.

II. RELATED WORK

A. Assessment of Computational Thinking

Since Jeannette Wing’s seminal work on CT in 2006 [1], research on K-12 CT education has grown. An important task in K-12 CT education is to assess students’ CT development. Brennan and Resnick [3] proposed a theoretical framework for assessing CT in three dimensions: CT concepts, CT practices, and CT perspectives. In this study, we focus on finding an effective and efficient tool for assessing CT concepts and CT practices. Traditional multiple choice open-ended essay tests are often used to assess CT concepts and practices. For example, Meerbaum-Salant et al. [9] developed an instrument to measure students’ understanding of programming concepts in Scratch throughout the intervention. Grover et al. [10] developed a paper-and-pencil instrument to assess students’ understanding of specific computing concepts and CT skills such as debugging. Jenson and Droumeva [11] designed a test to evaluate students’ CT concepts by assessing their knowledge of computer science concepts. Román-González et al. [12] developed a CT test composed of 10 items that evaluate four computational concepts, and they verified that CT is correlated with reasoning and problem-solving abilities. Portfolio assessment, which analyzes students’ learning outputs, has also been widely adopted for assessing CT [13–15]. Many CT learning activities have been designed for a programming context. The artifacts produced by students in the learning process are systematically collected and graded according to scoring rubrics [16]. For example, Dr Scratch [5] is a Web-based automatic analysis tool for Scratch programming projects. Code Master [17] is a Web application that provides grading and assessing services for App Inventor [18] and Snap! [19] programming artifacts. These automatic assessment tools relieve teachers from the labor-intensive assessment of students’ artifacts and thus improve efficiency. However, they may not accurately reflect students’ achievements in CT learning, because they perform static code analysis on programming project outcomes and evaluate CT concepts based on the frequency of the programming blocks that students used in the learning environment. They do not examine the functionality of the artifact and cannot identify whether the corresponding CT concepts were properly used in the programming tasks. In addition, CT is essentially a higher-order cognitive activity, and CT practices can only be observed in the programming process, not in the final programming artifact. For example, CT practices such as incremental and iterative progress can only be evaluated by investigating the series of programming tasks that students perform during a programming activity.

B. Performance-based Assessment to Support CT Development

Many CT learning activities take the form of problem-based and project-based tasks, wherein students develop CT skills while finding solutions to well-defined problem spaces. In the learning process, the learning status of students varies because of differences in their prior knowledge, learning pace, and learning abilities. Some students may need help in the learning process, whereas others may be able to move on without support. Providing formative feedback to students during the learning process helps to address this learner diversity. It encourages students to reflect on their learning progression and provides opportunities for them to make progress. A performance-based assessment platform is a useful tool for providing formative feedback. Koh et al. [20] designed a real-time CT assessment tool that provides teachers with information to make effective instructional decisions. The assessment system is embedded in the Scalable Game Design Arcade and offers real-time CT pattern analysis outputs on programming projects. The analysis results are made into visuals that give teachers a high-level sense of the progression both entire class and individual student. With this system, teachers can provide timely support to students in need. Choi et al. [21] designed a puzzle-based program for algorithm learning and found that it was effective in improving students’ CT competency. The designed puzzles enabled students to apply algorithm design skills to solve tailored-made real-world problems. Students were required to explain their algorithms. The algorithms were then verified and evaluated by teachers during the learning process. Fronza et al. [22] used agile software engineering methods to teach CT and designed CT learning activities that followed the general procedure of software development. They used programming project analyses and artifact-based interviews to assess students’ performance at multiple points across the learning process. The planned assessment was augmented by unplanned assessment based on the students’ progress to provide the necessary support. Teachers regularly interviewed students about their learning and guided them through their understanding of CT concepts. Vallance and Towndrow [23] employed a transforming pedagogy for CT. They pointed out that the concept of feedback is missing from Brennan and Resnick’s assessment framework and that it is an essential characteristic for CT learning. In their study, teachers utilized the Welsh Baccalaureate Essential Skills assessment to provide formative feedback and help students reflect on their learning experience after each task.

To sum up the key points of the research studies on CT development, providing formative feedback at the right time during the learning process is important and, in general real-time feedback is preferred. However, in all previous learning studies, teachers have provided most of the formative feedback, but in practice limited time and resources means that teachers are not able to care for the learning needs of all students. Without the support of automatic tools, it is difficult to implement a large-scale performance-based assessment of CT learning. In addition, well-defined tasks are needed to help students develop CT concepts and practices in the problem-solving process. Therefore, in this study, we designed a performance-based assessment platform to provide real-time feedback to students as they solve well-defined programming tasks. One of the aims of building this platform is not to compete with existing block-based programming environments, but to provide a complementary platform that focuses on fundamental CT concepts and practices across platforms to aid in the assessment and development of CT concepts and practices among students inexperienced in CT.

III. DESIGN OF THE PERFORMANCE-BASED ASSESSMENT PLATFORM

EasyCode

Performance-based CT assessment requires a performance assessment platform that provides well-defined CT tasks, real-time feedback, and practical ways for researchers and practitioners to observe students’ performance. Although numerous block-based programming environments are available for K-12 students, no platform allows researchers and practitioners to design and administer programming tasks for assessment or provides automatic judgment of those tasks. Therefore, we developed the block-based EasyCode platform. It provides tasks and judgment services online, to help develop students’ CT concepts and practices. The EasyCode platform contains a Learning Analytics Agent that captures the programming process and stores programming artifacts throughout the task. The
EasyCode platform supports students’ CT development by setting performance tasks and a series of progressive subtasks.

A. System Architecture of the EasyCode Platform

The system architecture of the EasyCode platform is shown in Fig. 1. The platform contains four components: Interface Agent, Judgment Agent, Learning Analytics Agent, and CT Performance Task Database.

The Interface Agent controls user interaction through a block-based visual programming environment with a code submission entry. It is the front end of the platform. Fig. 2 shows a screen capture of the interface. Users submit their solutions to the server. The submitted solution is then executed by the Judgment Agent and tested on predefined test cases in the system. Cases that fail the test are presented along with the output generated by the users’ programming code. Users are allowed to test their solutions on custom test cases.

The Judgment Agent provides the online judgment service. It executes the submitted code for the predefined test cases in each task. The submitted code is first converted into a textual programming language such as JavaScript or Python before execution. To prevent malicious code entry, the conversion is conducted on the server side rather than the client side. The submitted code is executed in a Docker container for security and for preventing accidental crashes caused by programming errors. The executed result is then compared with the expected output and feedback is generated for the user.

The Learning Analytics Agent records user operations in the programming environment and stores both intermediate and final programming artifacts. The Learning Analytics Agent backs up every user submission to the Judgment Agent. When users test or debug their code using the self-defined test cases, the Learning Analytics Agent records both the code and the test cases. In addition, user operations during the programming process (such as drag, add/delete, connect/disconnect blocks) are all captured in the user logs stored by the Agent.

The CT Performance Task Database stores open-ended programming tasks, solutions, and related test cases. The database is designed to store all of the feasible answers to programming tasks to enable the automatic assessment of student submissions. Researchers and practitioners can design their own programming tasks according to their CT curriculum and provide test cases for those tasks. An example of a CT performance task is given in this study. The EasyCode platform also provides labels for task designers to mark their tasks to indicate the CT concepts and practices that are needed to complete the task.

The workflow of the platform is presented in Fig. 3. The Interface Agent fetches tasks from the CT Performance Task Database and display them to the students. Students can write codes in the programming environment provided in the Interface Agent and submit their codes to the Judgment Agent. The Judgment Agent executes the submitted codes and returns the execution result to the Interface Agent. The Interface Agent will display the result to students so that they can revise their code accordingly. In the students’ programming process, the Learning Analytics Agent will record student’s operations in the programming environment, self-defined test cases and intermediate coding products.

B. Key Features of the EasyCode Platform

EasyCode has two distinctive features that the existing online block-based programming environment and assessment tools do not have: a support mode and dynamic code analysis.

1) Support Mode

Some students may find it difficult to determine which blocks should be used to design a feasible solution to a problem, or they may encounter difficulties picking up useful blocks from the programming environment to solve the problem. EasyCode provides students with a support mode for each CT performance task. As shown in Fig. 2 and Fig. 4, the user can switch between “normal mode” and “support mode” by clicking a button on the interface. In the normal mode, a standard toolbox with all usable blocks is provided. In the support mode, a tailor-made toolbox is provided that contains only the blocks required for a feasible solution, thus providing hints to students by eliminating unnecessary blocks and helping them focus on the algorithm design. The support mode also provides a progressive ladder to help students develop CT, whereby a challenging task is decomposed into a series of subtasks to guide students through the solution process. Students progress by accomplishing the subtasks one by one.

2) Dynamic Code Analysis:

Existing automatic assessment tools mainly conduct a static code analysis of students’ final programming outcomes. By measuring the block codes’ frequency, the tools make inferences about whether students have achieved a suitable understanding of certain CT
concepts. Although frequency information can assess some attributes of students’ understanding of CT concepts, static code analysis cannot determine whether the blocks that have been used were properly applied to formulate the solution. For example, using a “while-do” block does not necessarily mean that a student has mastered the “loop” concept if the block has been applied incorrectly in an endless loop. Static code analysis is also easily misled by codes that contain a lot of unrelated blocks that do not function properly. The EasyCode platform conducts dynamic code analysis in which the submitted code is executed on the server to test its functionality. The output generated by the submitted code is compared with the pre-defined expected output of the task to judge the correctness of the coding. This online judgment service provides clear goals for students in testing and debugging their codes.

![User interface of EasyCode’s support mode and tailor-made toolbox.](image)

**Fig. 4.** User interface of EasyCode’s support mode and tailor-made toolbox.

### C. Design of CT Performance Tasks

A successful performance-based assessment requires well-designed performance tasks. A well-designed performance task usually has three criteria: real-world context, sufficient complexity, and clear performance criteria. A well-designed performance task should be like or similar to a real-world task and should require similar cognitive skills. The task should be complex enough to require effort from students. The performance criteria should be clear so that students can set goals for problem-solving. Careful task design is needed to fulfill these criteria. Task designers need to formulate the tasks in a concrete context that is familiar to students. The task description should be precise and concise so that students can set clear goals for the programming tasks. Task designers can manage task complexity on the EasyCode platform by designing appropriate subtasks and toolboxes for the support mode. In terms of performance criteria, students’ main objective is to work out a feasible solution and submit it to the EasyCode platform for online judgment. The online judgment service can provide timely feedback, but it can only determine the correctness of the coding. That alone is not sufficient, as CT involves more than correct coding. The platform is designed to provide performance-based assessment of both CT concepts and CT practices. CT concepts can be assessed by observing whether students properly applied the correct programming blocks and functions in solving the programming tasks. Such assessments can be made in a single task. However, assessing CT practices requires the continuous evaluation of a series of tasks. The EasyCode platform achieves this by studying the solving process of a series of subtasks.

Task difficulty management is important for performance assessment because task difficulty is closely related to students’ performance. Previous research suggests that the relationship between task difficulty and performance is an inverted-U, which means that students’ performance first increases then decreases as students experience increased difficulty [27,28]. The increase in task difficulty at the beginning can cause a corresponding increase in performance [29]. However, when the task difficulty reaches a certain level, it begins to impair students’ performance [30]. To ensure that each student is challenged but capable of progressing, we categorized the performance tasks into three levels of difficulty. They are defined according to the CT concepts required for the tasks. Level 1 tasks are the least difficult. They can be solved using a single CT concept. They assess fundamental CT concepts. For example, the Level 1 task “Sum of Two Numbers” requires students to write a program that can add two numbers. Students should have the knowledge of mathematic operators such as addition, subtraction, division, and multiplication. They accomplish the task using an “add” block. Level 1 tasks also assess students’ testing and debugging skills with the help of the online judgment service. Students receive error messages from the Judgment Agent if their programs do not work. They will need to modify their codes accordingly. When students submit appropriate test cases, they will be informed that their programs work and are fully tested and debugged. In this case, students will have a record of successful practices of testing and debugging at Level 1. Another goal of the Level 1 tasks is to familiarize students with the programming environment and functions on the EasyCode platform. Students who are beginners in programming learn about fundamental CT concepts by working on Level 1 tasks. Students who are users of other block-based programming environments such as Scratch and App Inventor can easily transfer their programming knowledge and quickly familiarize themselves with EasyCode toolboxes and functions by accomplishing Level 1 tasks. For example, the Level 1 task “Calculating the remainder” teaches students how to use the modulo operation. The modulo operation is provided via a “the remainder of” block on the EasyCode platform. Users familiar with blocks for the modulo operation from other programming environments can easily transfer their knowledge to EasyCode by working on this task.

Level 2 tasks require users to apply two CT concepts to produce a solution for a programming task. For example, the task “Swap” requires users to exchange the value of two variables. Students need the CT concepts of variable and assignment, which entail resetting the value stored in one variable to another variable. The related CT practices required in this task include designing a simple algorithm of swap as well as testing and debugging the program with a test case showing the result of swap. A level 2 task can also be grounded in an operation on data structure. Students must know data structures, such as list, which is a collection of items, and how to operate on them. The Level 2 task “Print a non-empty list of numbers” assesses students’ CT concepts of a list that holds items with index and their use of a loop to traverse the list using its index to print all the numbers in the list. Students need to understand that a list is a sequential data container and that items in the list can be retrieved with index. Students need to demonstrate the CT practices in the task, such as designing an algorithm for performing the task and testing and debugging the program to complete the task. The list in this task is constrained to be non-empty. Without this constraint, the difficulty of the task would be too high for a Level 2 task because students would need an extra CT concept, the conditionals, to solve it. Students need to employ a conditional statement to determine whether the list is empty and display an empty list notice if the list does not contain an item.

Level 3 tasks require users to apply more than two CT concepts to solve a programming task. Level 3 tasks are relatively complex and require the use of multiple CT concepts. Students need to understand the programming task and devise a plan to solve the problem. For example, the Level 3 task “Prime number or composite number” asks
students to develop an algorithm to determine whether a given positive integer is a prime number or a composite number. Prime numbers have two factors while composite numbers have more than two factors [24]. Students need to find the factors of the given positive integer, store the factors in a list, and make decisions based on the length of the list. This level 3 task is a difficult task because it contains multiple computational subtasks with each subtask involving several CT concepts. This platform provides support mode for difficult tasks to support learner diversity. In the support mode, the platform provides a suggested plan. It decomposes the task into subtasks to help the user through the programming task. For example, the Level 3 task “Prime number or composite number” in the support mode advises the students to divide the task into three subtasks. The programming codes in subtask can be reused to construct the solution for the next subtask. In the next section, we will discuss this Level 3 task in detail and illustrate how students can develop CT concepts and practices using the support mode.

IV. EXAMPLE CT PERFORMANCE TASK: PRIME NUMBER OR COMPOSITE NUMBER

Many curriculum and assessment tools have been developed for K-12 CT education [4], [10], [25]. In this section, we present an example of a CT performance task that develops CT as students learn a primary mathematics topic. This section builds on the work of Kong [26]. In this learning task, students are asked to develop an algorithm to determine whether a given positive integer is a prime number or a composite number. The EasyCode platform is well suited for this kind of task for three reasons. First, the learning task is situated in a solid computing context that primary school students are familiar with through their mathematics learning. Second, deciding whether a number is prime or not requires several computational subtasks, suggesting that the task has sufficient complexity. Finally, the task has a clear and programmable goal that enables us to develop corresponding performance subtasks for the support mode.

This task is formulated in the EasyCode platform for presentation to students. It is a complex task for students. The EasyCode platform sets up a progressive ladder to help students tackle the task in the support mode. In the support mode, the original task is decomposed into three subtasks. The task description and the required CT concepts and practices are listed in Table I. The decomposition follows the key steps in the algorithm for deciding whether a given positive integer is a prime number or a composite number. The design of the subtasks reduces the complexity of the original task and provides a clear road map for students to solve the problem.

In the first subtask “Is it a factor?”, students need to demonstrate knowledge of CT concepts such as conditionals, modulo operator and comparison operator, which are needed to determine whether a number is the factor of another number. The modulo operator is needed to build a block to calculate the remainder in the division. If students know the corresponding concepts in the mathematics curriculum, they can use the modulo operator to solve the subtask.

The concept of comparison is needed to compare whether the calculated remainder is zero. The concept of conditionals is required to generate the output text based on the comparison result. Students also need to demonstrate the practice of testing and debugging using the online judgment service on the EasyCode platform. After constructing the code, students test their programs then submit the solution to the platform. The program should function properly under two situations: (1) A is a factor of B and (2) A is not a factor of B. Students need to design test cases, which are recorded in the Learning Analytics Agent of the EasyCode platform. Researchers and practitioners can then examine whether students considered all situations by examining the submitted test cases. After students submit their codes to the server, the Judgment Agent informs the students whether their code correctly determines whether A is a factor of B. If the submitted code fails the pre-defined test case on the platform, the failed case, the expected output, and the user’s output are fed back on the interface. Students can then debug their code accordingly.

The second subtask “Find all factors of a positive integer” is a subsequent task that requires students to find all of the factors of a given positive number. In this task, students need to demonstrate the concept of loops using a “for” block to enumerate all numbers that are less than or equal to the given number A. Students will need to know the concept of lists because the task requires storing factors in a list. The second subtask is closely related to the first task because deciding whether a number is a factor of another number is also a core part. The solution to the second subtask can be achieved by adding loops and list operations based on the solution to the first subtask. Students therefore experience incremental and iterative programming as well as reusing and remixing.

The final subtask “Prime/composite?” is the undecomposed original task from the normal mode. In this task, students need to write a complete program to decide whether a number is prime or composite. This task may be challenging if students do not have the results from the previous two subtasks to solve it directly. Students need to know the concept of factors and how to calculate a number’s factors before deciding whether the number is prime. However, if students have solved the two previous subtasks, they may have an easier time with the final subtask. The final subtask allows students to demonstrate certain CT practices, such as being incremental and iterative and reusing and remixing. As shown in Fig. 5, they can also build their solution by reusing part of the code they generated to calculate the factor list in the second subtask. Students can then use a conditional block to judge the length of the factor list and determine whether the given number is a prime number or a composite number.

V. SUMMARY AND FUTURE WORK

In this study, we present a performance-based assessment platform called EasyCode for developing CT concepts and CT practices. The EasyCode platform is equipped with a block-based visual programming environment and a Learning Analytics Agent that captures students’ programming processes. The EasyCode platform also provides an online judgment service that gives students timely feedback on their performance in solving tasks or subtasks of a given programming activity. With EasyCode, researchers and practitioners can design and administer tasks according to their curriculum. An example on how to develop a CT performance task based on a mathematics curriculum is provided. It shows how CT performance tasks can be based on a context familiar to students. In the future, we will expand the CT performance task database. The platform presented in this study is still a prototype. Students’ experiences on our platform remain unknown. A user study will be conducted to address this limitation. We will improve the user experience of the platform accordingly.

CT practices are an important aspect of the assessment of CT development. CT practices are best evaluated through high-level programming skills such as testing and debugging, being incremental and iterative and reusing and remixing. Therefore, to assess CT practices, we need to study students’ programming processes. The EasyCode platform keeps a log of user operations in the programming environment with the Learning Analytics Agent and uses them as a basis for assessing CT practices and higher-order cognitive skills. Future work will involve the development of further analysis tools for extracting learning features from the user logs in the Learning Analytics Agent.
Analytics Agent. Visualization of the learning trajectory of students using EasyCode will also be explored in future research.

REFERENCES


Fig. 1. System architecture of the EasyCode platform

### TABLE I

<table>
<thead>
<tr>
<th>Task Order</th>
<th>Subtask Name</th>
<th>Brief Description of the Task</th>
<th>Required CT concepts</th>
<th>Required CT practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is it a factor?</td>
<td>Given a positive integer A and a positive integer B, design a program to determine whether A is a factor of B.</td>
<td>Conditionals, operators (modulo, comparison)</td>
<td>Testing and debugging</td>
</tr>
<tr>
<td>2</td>
<td>Find all factors of a given positive integer</td>
<td>Given a positive integer A, design a program to calculate a list that stores all of the factors of A</td>
<td>Conditionals, operators (modulo, comparison), loops, data (list)</td>
<td>Testing and debugging, being incremental and iterative, reusing and remixing</td>
</tr>
<tr>
<td>3</td>
<td>Prime / composite?</td>
<td>Given a positive integer A, design a program to determine whether A is a prime number or a composite number according to its number of factors.</td>
<td>Conditionals, operators (modulo, comparison), loops, data (list)</td>
<td>Testing and debugging, being incremental and iterative, reusing and remixing</td>
</tr>
</tbody>
</table>

Fig. 5. Example of an alpha-test user’s submitted solutions and their reuse of codes.
KONG Siu Cheung currently is Professor of the Department of Mathematics and Information Technology (MIT); and Director of Centre for Learning, Teaching and Technology (LTTC), the Education University of Hong Kong. Prof. Kong holds a doctorate from the Department of Computer Science of the City University of Hong Kong. Prof. Kong has produced over 230 academic publications in the areas of pedagogy in the digital classroom and online learning; policy on technology-transformed education and professional development of teachers for learner-centered learning; and computational thinking education.

Bowen Liu obtained his Ph.D. from the Education University of Hong Kong. He received his bachelor degree from Jinan University and studied in Aalborg University Denmark as an exchange student. He has broad research interest in image processing, computer vision and learning analytics.