Development of an Instructional Model based on Constructivism for Fostering Computational Thinking

Sook-Young Choi

Abstract: Background/Objectives: The purpose of this study is to design an instructional model based on constructivism to enhance Computational Thinking (CT) and to investigate the effect of the model on CT in Java programming class of college students. Methods/Statistical analysis: In order to analyze the effect, a CT evaluation rubric for the learners' program artifacts and questionnaires to measure learners' perceptions and effectiveness of CT, programming, and problem solving were developed. Two evaluators assessed the midterm and final program artifacts submitted by the learners with the evaluation rubric. A pre and post surveys were conducted using the questionnaires to examine changes in leaners' CT, programming learning, and problem solving abilities. A paired t-test was used for the pre and post survey.Findings: The analysis results showed that the CT assessment score of the final program artifacts were slightly higher than the midtermprogram artifacts.In addition, the survey analysis revealed that learners had improved their scores on the perception and effectiveness of CT, programming, and problem solving in the post survey compared to the pre survey. The instructional model based on constructivism confirmed that it helped students to learn CT and programming positively. In particular, it was found that reflecting on the problem solving process based on CT after completing a program coding could contribute to the learner's more clearly internalization of the learning experience of the CT concepts and thus foster problem solving ability using CT. Improvements/Applications: It is necessary to modify and supplement the instructional model and the assessment model for CT proposed in this study and analyze the effect by applying it in other programming classes in the future.

Keywords: Computational thinking, Problem solving, Programming, Constructivism, Reflection

I. INTRODUCTION

Since computers are used in almost every area of our world, the performance of computing technology and SW has a great impact on not only companies but also national competitiveness. In particular, technologies such as artificial intelligence, the Internet of things, and big data, which are core technologies of the Fourth Industrial Revolution that will fundamentally change human life and society, are all based on software. Accordingly, countries around the world recognize the importance of software education and prepare a national curriculum to strengthen software education [1]. In Korea, SW education has been emphasized through the software education operating guidelines and the revised curriculum in 2015, and software education has been introduced and implemented in middle schools since 2018.

The key goal of software education is to nurture people with computational thinking (CT), the ability to think given problems from the perspective of computing and solve them efficiently.

CT was first used by S.Papert (1991), but it was the paper published by Professor J. Wing (2006) that people actually got attention. Wing (2006) argued that "CT is the thinking ability that everyone in the twenty-first century should have" [2]. The concepts and components of CT are slightly different depending on the institution and the scholar [3], but they can be defined as "the ability to solve everyday life problems creatively and efficiently using the basic concepts and principles of computer science, and computing systems". The training for CT can be done primarily in programming classes [3]. But, most programming classes are often taught with a focus on program coding rather than focusing on developing problem solving skills from the CT perspective. Recently, some studies on CT-based instructional design and application in K-12 have been carried out, but they have not deviated significantly from existing programming teaching methods or strategies.

In this study, we use a constructivism-based learning strategy to foster CT in college programming classes. Think-pair-share, pair programming, and reflection learning are used as learning strategies. After developing an instructional model based on these learning strategies, it was applied to the class and analyzed the effect.

II. RELATED WORKS

2.1. CT and Programming

Wing (2008) defined "computational thinking is a thought process that can analyze problems and format them in a form that can be effectively implemented by computers."[4]. Abstraction and automation are introduced as sub-components of CT. After that, CSTA, BBC, Google for Education in the UK, etc., defined the concept of CT and the operational definition of sub-elements in detail. TABLE 1 is a revised version of the table presented by Lee (2017) [5].

Programming classes can be considered as a common way to promote CT. Studies are being conducted on programming classes to enhance CT at home and abroad. Lye & Koh (2014) analyzed studies to enhance CT through programming based

Revised Manuscript Received on January 03, 2019.

Sook Young Choi, Corresponding author, Department of Information Security, Woosuk University, Samre-up, Jeonbuk, South Korea.

on the three-dimensional framework developed by Brenda & Resnick [3]. Kim (2015) designed and applied a learner-oriented class model for developing CT to college students and analyzed their effects [6].

TABLE 1. Computational Thinking Components

Researchers	ISTE & CSTA	BBC	Brennan & Resnick	KERIS
Computational Thinking Components	·Data collection ·Data Analysis ·Data Representation ·Problem decomposition ·Abstraction ·Algorithm & procedure ·Automation ·Simulation .Parallelism	·Decomposition ·Pattern recognition ·Abstraction ·Algorithms	·Computational Concepts -Variable, Loop ·Computational Practices - Being incremental &iterative, Testing &debugging, Reusing, Abstracting) ·Computational perspective - Expressing & questing about the technology world	·Data collection ·Data Analysis . Structuring ·Abstraction - Decomposition -Modeling - Algorithm ·Automation - Coding - Simulation ·Generalization

Romero, Lepage, and & Lille (2017) introduced CT in creative programming classes using Scratch in universities [7]. Research by Fessakis et al. (2013) showed that the visualization of programming codes helped K-12 students in terms of CT [8]. Brenda & Resnick (2012) proposed three dimensions of CT as a CT model for Scratch classes; computational concepts, computational practices, and computational perspectives [9]. Based on this framework, several studies have assessed CT. Hoover and his colleagues (2016) conducted a study to assess students' CTs through game programs designed by students [10].

Looking at the above studies, most studies focus on programming learning methods simply to move to the stage of improving learner's thinking, rather than on a variety of teaching strategies to help students experience the process of solving problems using CT to improve their thinking skills

2.2. CT Education from a Constructivist Perspective

Studies have been conducted to approach CT from the perspective of constructivism. Learning from constructivism is centered on learner activity and learners are leading learning. In other words, learning is not done as a transfer of knowledge by the teacher, but rather as a learner's own reconstruction of meaning [11]. From a constructivist perspective, teachers play a role in encouraging learners and promoting learning [12]. As an instructional strategy based on this constructivism, this study considered think-pair-share, pairing programming and reflection learning.

2.2.1. Think-pair-share and Pair programming

Think-pair-share (TPS) is an active learning strategy in which students participate to solve a problem [13]. This method allows students to think individually about any topic or problem first, and then have time to discuss their partner and what they think. Finally, let the students in the whole class present their thoughts and opinions. This technique can promote students' participation in learning and thinking, and can increase the interest of learning because they can get immediate feedback from their mates.

Pair programming technique is that two people use one computer to collaborate [14]. In other words, two learners sit together to collaborate on problem analysis, algorithm design, coding and debugging. One of the two learners plays the role of driver and the other plays the role of navigator. A driver is mainly responsible for coding using keyboards and mouse, and a navigator searches for strategies and tactics for problem solving or finds errors in the program. Through interactions, they can create new ideas for problem solving or complement each other's problems. There are many advantages such as being able to more concreate their knowledge in the process of discussing each other and to raise learning motivation.

In this study, we use the above two strategies. First, we use think-pair-share techniques to allow learners to think about how to solve problems and to discuss their thought with their mates. Then we use the pair programming to write program on their computer with my partner.

2.2.2. Reflective learning

In learning, reflection means rethinking what students have experienced, which means looking back on the learning process or connecting current experience to the existing learning contents. Through this process, they can approach better problem solving processes at the next opportunity by recognizing the deficiencies in problem solving and considering the direction of its improvement. Moreover, the reflection process includes the withdrawal process, so the withdrawal can be made more quickly and accurately as the concept once drawn is left with a more solid concept or integrated with the previous knowledge in the brain, thereby enhancing the neural circuit.

There are studies that apply this reflection learning to programming learning [15]. Lye & Koh (2014) argues that through reflection, learners can develop computational practice and computational perspectives because it requires them to reflect on their programming process [3]. This study encourages learners to develop their CT by looking back at the problem solving process based on CT through reflection.

III. RESEARCH METHODS AND PROCEDURES

3.1. An Instructional Model based on Constructivism for Fostering CT

This study designed an instructional model to enhance CT in a university programming class. The instructional model was designed based on CT and constructivism. It considers the programming process as a problem solving process and approach CT from the point of view of problem solving strategy. It also uses think-pair-share, pair programming and reflection learning, as a teaching strategy to help programming and CT. Fig 1 shows this learning model.

Because programming processes require cognitive and metacognitive aspects, learners generally find programming difficult. Especially, novice programmers generally lack several traits compared with experts. They make mistakes such as the grammar error, translation error, and strategic error during the programming process. The strategic error is in the process of planning a detailed procedure for achieving a goal.

In particular, in the case of novice programmers, there are many strategic difficulties in how to solve problems. In this study, we approach CT as a problem solving strategy. In other words, students analyze the problem, decompose the problem, find the core idea of the problem solving through the abstraction process, and find out if there is a pattern that is repeated and used. This process is associated with CT components, but it can be difficult for learners to do this alone. Therefore, in this study, we support scaffolding by providing learners with time to think first at each stage of problem solving from a constructivist point of view and then discussing what they think with their colleagues from a constructivist point of view.

3.1.1 Think-pair-share and Pair Programming Learning

This study employed an instructional model based on think-pair-share andpair programming. However, the model is a modified type from original think-pair-share and pair programming. The think-pair-share technique was used in the phase of problem definition, problem solving strategy, and algorithm design phase in problem solving process and pair programming was at the actual coding phase. The problem solving process is as follows. When a problem is given by a teacher, the first step is to define the problem. The learners will read and understand the problem at this stage. The next step is to set up a strategy to solve the problem, to break down the problem, to remove unnecessary parts, and to pick the core part of the problem. The core parts of the problem are designed as modules and the relevance between modules is displayed using UML, etc. Next is the algorithm design phase to consider the logical order for problem solving. This process finds patterns that are repeated. The algorithms are constructed based on these modules and patterns. These steps use think-pair-share. In other words, learners first define problems individually, then plan problem solving based on CT, and then discuss what they think with their peers, and then take steps to correct or supplement what they have considered through questions or feedback. By building a strategy based on CT and discussing it with a partner before coding on a computer, it can enhance learners' CT for problem solving.

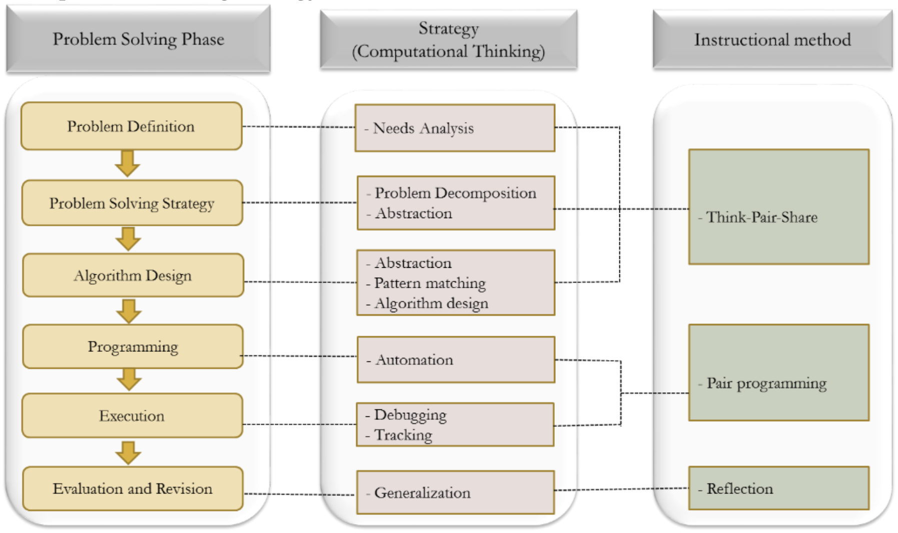

Fig 1. Instructional Model based on Constructivism Fostering CT

The next step is to create a program using the actual programming language as an implementation step. If an error is found during coding, locate the cause and correct it. This step is carried out using pair programming. Then, one of two students write program and then has to ask appropriate questions, to respond to immediate feedback from their partner, and revise the program. While students were engaged in the pair programming activity, the instructor provided appropriate feedback and asked questions in order to guide their discussions. Since pair programming requires individuals to perform operations involving metacognition, it could foster students' self-monitoring and improve their problem solving performance.

3.1.2 Reflection

In this study, after completing a program coding, evaluate the program, revise it, each learner will have time to reflect on the problem solving process. By looking back at the problem solving process, he/she considers the CT elements approached in the problem solving strategy stage and the algorithm design stage, and think about what they meant to solve the problem. In addition, he/she considers what difficulties there were in what part. It will be required to explain how he/she solved those difficulties. Through this process, learners are encouraged to develop CT by making their learning experiences more clearly internalized. In order to induce students' reflection, this study provided students with questions that were already written, so that students could simply describe them according to the question.

3.2 Research subjects and research design

The study was conducted on 28 students who took the Java programming class at X University, which consisted of students in the second and third grades. The rest of the students, except for the two, took the C language as a prerequisite, and the class was designed based on the learning model proposed in this study. Three experts consulted to design a class based on the model. In addition, after the class was designed by the learning model, it was reviewed by experts to verify its validity. Students were asked to submit two program artifacts during the semester. One program artifact was to be submitted at the mid-term and the other was the end of the semester.

Students were given an explanation of the course progress and CT at the beginning of the semester. From the 1st week to the 7th week, the classes were conducted according to the existing teaching methods. In order to provide classes based on the instructional model developed in the study, students were given an explanation of the instructional strategies in detail at the 8th week. After the explanation, the classes actually went from the 9th to the 14th week.

3.3 Research analysis method

The following analytical tools were used to analyze the effects of the instructional model proposed in this study on students' CT. First, we made a CT evaluation rubric for the students' project outputs and assessed the outputs using it. The CT evaluation rubric was made by modifying the evaluation rubrics presented in the study [9] to fit the Java programming language and was checked by content validity through three experts. Two project outputs were assessed by the developed evaluation rubric.

We also developed questionnaires to measure learners' perceptions and effectiveness of CT, programming, and problem solving before and after programming classes based on the learning model. The questions were consisted of questions such as the degree of awareness of CT as a problem-solving strategy, the usefulness of CT in a class model, the ability to solve problems through CT, and the ability to solve problems through programming. The created questions were modified through discussion with computer engineering experts and education experts. Students were also asked to answer a narrative questionnaire at the end of the semester.

IV. RESULT ANALYSIS

4.1 CT Assessment of the program artifacts

We assessed the CT of students by applying the rubric developed in this study to the program artifacts from the programming stage of the students. Two evaluators applied the evaluation rubric to the midterm and final program artifacts submitted by the students. The reliability of the evaluation rubric was verified by the inter-rater reliability, and the inter-rater reliability was confirmed by the consistency of the opinions of the two evaluators.

Consistency was calculated by Cronbach's α. The value of Cronbach's α was 0.71. TABLE 2 shows the average scores of evaluators on two projects. As shown in the Table 3, we were able to see that the average score value for the final program artifacts was elevated above the average score for the mid-term program artifacts.

TABLE 2. Average score by evaluators

Domain	M	SD	Max	Min
Intermediate	55.9	8.6	93	30
final	64.7	6.9	98	38

4.2 Questionnaire analysis results

Two surveys were conducted before and after class using the questionnaires developed to examine changes in students' CT, programming learning, and problem solving abilities. A paired t-test was performed for the survey analysis. TABLE 3 shows the analysis results of pre-post surveys. It shows that the post-survey scores all increased in the areas of computing power, programming, and problem solving. The significant level also shows a statistically significant value. The results showed that the instructional model designed in this study had a positive effect on students' computational thinking, programming skills, and problem solving abilities.

TABLE 3. Pre-post questionnaire analysis result

Domain	Survey	M	SD	t	p
Computational thinking	pre	3.57	0.83	.927	0.018
	post	3.78	0.61		
Programming	pre	3.41	1.32	-1.55	0.034

	post	3.83	0.98		
Problem Solving	pre	3.52	0.75	-1.88	0.003
	post	3.89	0.69		

4.3Analysis of narrative questionnaires

The students' answers to the narrative questionnaires were analyzed using NVivo 10, a qualitative data analysis tool. Through the tool, we analyzed how the class based on the instructional model proposed in this study had an effect on the students. According to the analysis result, the most frequent ones in terms of effectiveness are as shown in Figure 2. As shown in the Figure 2, the students answered that their classes were the most helpful for understanding CT concepts, problem solving, programming, etc.

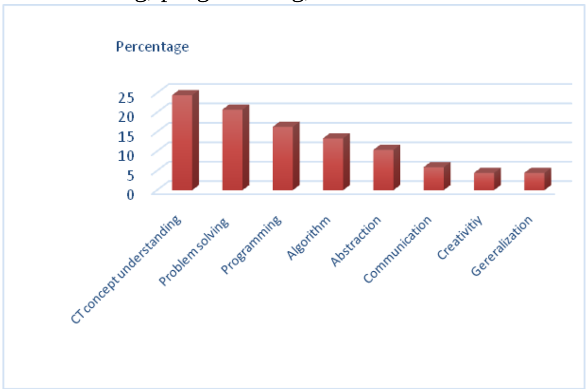

Fig 2. Narrative survey analysis

V. CONCLUSION

In this study, we designed a CT instructional model based on constructivism for the Java programming class of the university and applied it to the class to verify its effect. The instructional model supports think-pair-share, pair programming, and reflective learning based on constructivism to approach CT and boost CT as a problem solving strategy. The results of the study showed that the instructional model proposed in this study positively affected students' CT, programming, and problem solving abilities. In particular, each student could better understand the CT concepts as a learning strategy through discussions with peer learners, and have an in-depth understanding and application of problem solving strategies using the CT concepts. Pair programming allows students to develop their understanding, program writing skills, and analytical skills of programs. Also, since the problem-solving process can be reconsidered through reflection, more meaningful learning can be done about the CT concepts and their application. It is necessary to revise and supplement the instructional model proposed in this study as a future research project. In addition, it is necessary to evaluate the effectiveness of the model more accurately through the study on the evaluation method that can effectively evaluate CT.

REFERENCES

1. Zhong B, Wang Q, Chen J, Li Y. An exploration of three-dimensional integrated assessment for computational thinking. Journal of Educational Computing Research. 2016 Oct; 53(4):562-590
2. Wing JM. Computational thinking. Communications of the ACM. 2006 Mar; 49 (3): 33-35
3. LyeSY, Koh JHL. Review on teaching and learning of computational thinking through programming. Computers in Human Behavior. 2014Dec; 41:51-61
4. Wing JM. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London: Mathematical, Physical and Engineering Sciences*. 2008 Oct;*366*(1881): 3717–3725.
5. Lee CH. Development of Real-Life Problem Solving Model (CT-RLPS Model) based on Computational Thinking for Software Education. Journal of Korean Practical Arts Education. 2017 Sept; 30(3): 33-57. Available from http://www.earticle.net/Article.aspx?sn=283400
6. Kim SH. Effects of Teaching and Learning Strategies of Learner-Centered Learning for Improving Computational Thinking. Journal of the Korean Association of Information Education. 2015 Sep; 19(3): 323-332. Available from http://www.earticle.net/Article.aspx?sn=254358
7. Romero M, Lepage A, Lille B. Computational thinking development through creative programming in higher education. International Journal of Educational Technology in Higher Education. 2017 Dec;14(42): 1-15
8. Fessakis G, Gouli E, Mavroudi E. Programming solving by 5-6 years old kindergarten children in a computer programming environment: A case study. Computer & Education. 2013 Apr; 63: 87-97
9. Brennan K, & Resnick M. Using artifact-based interviews to study the development of computational thinking in interactive media design. Paper presented at annual American Educational Research Association Meeting. 2012Vancouver, Canada.
10. Hoover AK, Barnes J, Fatehi B, Moreno-León J, Puttick G, Tucker-Raymond E, Harteveld C. Assessing computational thinking in students' game designs. In Proceedings of the 2016 annual symposium on computer-human interaction in play companion extended abstracts. 2016 Oct; 173–179
11. Papert S, Harel, I. Constructionism. Norwood. NJ: Ablex Publishing; 1991.
12. Sharma S, Bansal D. Constructivism as paradigm for teaching and learning. International Journal of Physical Education, Sports and Health. 2017; 4(5): 209-212
13. Lyman F. Think-Pair-Share: An Ending Teaching Technique. MAA-CIE Cooperative News. 1987 Oct; 1:1-2.
14. Williams L, Kessler RR, Cunningham W, Jeffries R. Strengthening the Case for Pair Programming. IEEE Software. 2000 Jul; 17(4): 19-25, DOI:10.1109/52.854064
15. Zimmerman BJ, Tsikalas KE. Can Computer-Based Learning Environments (CBLEs) Be Used as Self-Regulatory Tools to Enhance Learning?. Educational Psychologist. 2005; 40(4): 267-271.Available from https://www.learntechlib.org/p/73427