Id: 1860106

Participatory Educational Research (PER)
Vol.13(3), pp. 119-141, May 2026
Available online at http://www.perjournal.com
ISSN: 2148-6123
http://dx.doi.org/10.17275/per.26.37.13.3

The Effect of Problem-Solving-Based Programming Training on Computational Thinking Skills in the Context of Programming and Reasoning

Servet Kılıç *
Computer Technologies Department, Vocational High School of Technical Sciences, Ordu University, Ordu, Türkiye
ORCID: 0000-0002-1687-3231

Seyfullah Gökoğlu
Department of Computer Technology & Information Systems, Faculty of Science, Bartın University, Bartın, Türkiye
ORCID: 0000-0003-0074-7692

Article history

Received:
09.11.2025

Received in revised form:
22.01.2026

Accepted:
11.02.2026

Key words:
computational thinking; programming; reasoning skills; problem-solving-based programming

Computational thinking (CT), which is based on the effective use of information processing tools in problem solving, can be developed through programming activities. Because programming is a complex structure requiring the use of strategic information (debugging, algorithmic thinking, evaluation, and others.) as well as conceptual knowledge (decisions, loops, operators, variables, arrays, and so on), novice students may experience difficulties in solving certain problems even after learning the basic concepts. This can limit the development of CT. Creating algorithmic structures through reasoning based on fundamental concepts depends on changing the focus of the training. This study aims to examine the effect of problem-solving-based programming training (PSbPT) on CT development in the context of programming performance and reasoning skills. PSbPT is a structure based on the use of CT components depending on the problem-solving stages. 40 students studying in the computer technology department of a state university underwent a 14-week programming training program. Participants were randomly assigned to groups in an experimental design with pre-test and post-test control groups. During the training process, students were presented with complex problem scenarios that integrated reasoning and programming and were based on CT components. The results showed that the programming performance and reasoning skills of the experimental group students who received PSbPT were statistically significantly improved compared to the control group. PSbPT was more effective than traditional programming training in terms of developing students' CT skills. The impact of the adopted PSbPT approach on programming, reasoning, and CT skills was discussed separately and holistically.

* Correspondency: servetkilic@odu.edu.tr

Introduction

Problem-solving is a skill that allows individuals to categorize objects into appropriate classifications, reach new conclusions based on events or rules, reason, and manage their own lives successfully (Dunbar, 2008). Students should develop problem-solving skills to succeed in their courses and daily lives. Reasoning, a sub-skill of problem-solving, means determining the causes of the encountered problems and making inferences about the solutions using inductive or deductive methods (Barbey & Barsalou, 2009). Reasoning is essential for success in daily and academic life. For example, a doctor can diagnose a disease by reasoning about the symptoms observed in a patient. A civil engineer can design durable and functional buildings by reasoning from data such as landforms, the condition of surrounding buildings, and weather conditions. Students need reasoning skills to succeed in the courses they are responsible for during their education.

Reasoning is recognized worldwide as an essential reference for investigating students' educational achievement. Various exams are organized for students at different levels by organizations such as the Organisation for Economic Co-Operation and Development (OECD) and the International Association for the Evaluation of Educational Achievement (IEA). Programme for International Student Assessment (PISA), Trends in International Mathematics and Science Study (TIMSS), and Progress in International Reading Literacy Study (PIRLS) are examinations conducted to periodically assess the educational achievements of countries at the international level. These exams are also organized for computer science (CS) fields in recent years and include tests based on without a computer coding, algorithms, reasoning, and higher-order thinking skills. BEBRAS is an international initiative that operates at all levels of education, from preschool to higher education (BEBRAS, 2022). BEBRAS is a test based on CT components, including breaking complex tasks into simpler components, algorithm design, pattern recognition, generalization, and abstraction. BEBRAS test includes reasoning-based questions that address these skills.

CT is a kind of analytical thinking that uses CS concepts to solve problems, design systems, and understand human behavior (Wing, 2006). ISTE and CSTA (2011) describe CT as solving problems using computers or other information processing tools, automating solutions through algorithmic thinking, identifying possible solutions, and transferring solutions to other problems by analyzing, applying, and generalizing. These definitions indicate that CT is a problem-solving approach related to various skills (see Fig. 1). According to Zhang and Nouri (2019), CT is a skill at the intersection of other concepts. The development of CT skills is related to programming (Lye & Koh, 2014; Ma et al., 2021; Mouza et al., 2020), reasoning (Boom et al., 2018; Marinus et al., 2018), and problem-solving activities (Román-González et al., 2017; Shute et al., 2017).

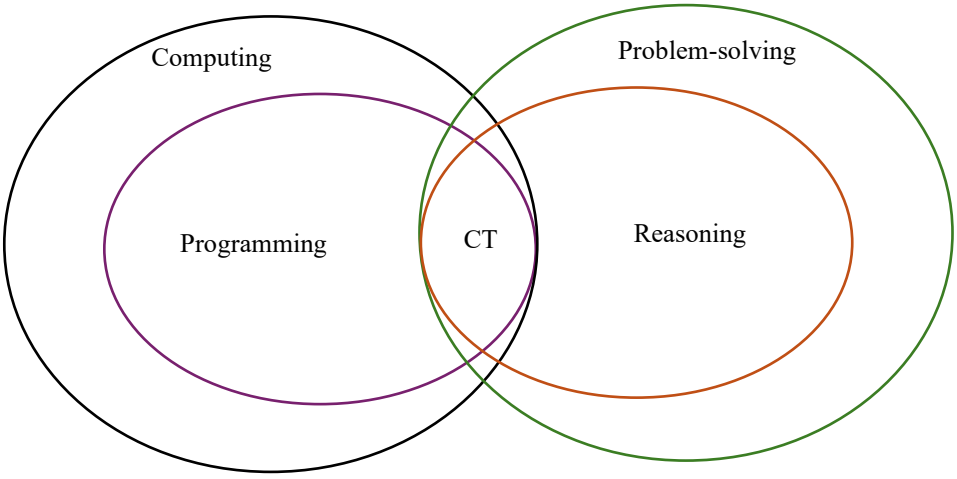

Figure 1. Concepts associated with CT

Programming can be used as a practical method to develop reasoning skills (Alferes et al., 1996; Clement & Falmagne, 1986). Conducting problem-solving-based educational activities within the framework of CT can be an effective method for programming learning (Çakıroğlu & Kiliç, 2023; Kılıç & Çakıroğlu, 2023). However, CT skills developed through programming learning may not be reflected in problem-solving processes in the context of reasoning (Shen et al., 2022). Therefore, designing an educational environment where reasoning and programming scenarios can be integrated can facilitate programming learning and contribute to developing students' CT skills.

Programming is a complex knowledge that requires applying skills such as analysis, planning, hypothesis generation, and problem-solving through algorithms created by combining various syntax structures (variables, loops, decision structures, operators, and alike) (Nickerson, 1983). Three types of knowledge stand out in programming learning: conceptual, semantic, and strategic (Oliver, 1993). Conceptual knowledge refers to syntax structures. Semantic knowledge involves creating algorithms. Strategic knowledge deals with solving problems. Students new to programming have difficulty using this knowledge in solving complex problems even if they learn the syntax structures of programming (Gökoğlu & Kilic, 2023). Garner (2003) stated that when teachers explain programming, students understand the topics well, but they struggle to solve problems when programming individually. Strategic knowledge depends on conceptual and semantic knowledge, and problem-solving skills are required to solve problems that require strategic knowledge (Lahtinen et al., 2005). Considering the difficulties of programming, educators might consider adopting methods where students can learn programming by focusing on problem-solving approaches instead of teaching only concepts (Kwon, 2017). In this way, the focus of programming instruction can be shifted from simple coding to problem-solving. This approach may help reduce emotional and cognitive barriers to programming in students' minds can be eliminated (Chang, 2014; Tom, 2015).

In this study, university students received PSbPT. We examined the development of programming performance, reasoning skills, and CT components. The research hypotheses are listed below:

RQ1: How does PSbPT affect programming performance?

RQ2: How does PSbPT affect reasoning skills?

RQ3: How does PSbPT affect CT components in the context of programming and reasoning?

Theoretical background

Computational thinking

Papert (1996) emphasized that we can effectively use computers to develop thinking skills. This emphasis created the basis of today's understanding of CT. Wing (2006) defined CT as a fundamental skill such as reading, writing, and arithmetic. The introduction of CT as a fundamental skill has increased the interest in CT research. The increase in CT research has led to different views on the definition, scope, development, and assessment of CT. The definitions of CT commonly include the effective use of computers in problem-solving (ISTE, 2016; Wing, 2006). We can use computers as a supplementary tool for performing mathematical and logical operations. In addition to the contribution of computers, learners need logical questioning and critical thinking to solve problems. In the definitions of CT, researchers talk about formulating the proposed solutions and transferring these solutions to different problems (ISTE & CSTA, 2011). Based on these definitions, using systematic approaches for problem-solving would be useful. Algorithm creation and problem-solving techniques are systematic approaches that can be used to solve problems that may arise in the information processing process (Aho, 2012).

Researchers associate CT with different problem-solving skills. These skills include abstraction, decomposition, data collection, data analysis, pattern recognition, conceptualization, data representation, mathematical reasoning, algorithm, parallelization, automation, modeling and simulation, testing, debugging, and generalization (Kalelioğlu et al., 2016). Decomposition makes complex problems more understandable by breaking them into smaller parts (Csizmadia et al., 2015). Abstraction is extracting the necessary information from complex information structures (Curzon et al., 2014). Algorithmic thinking is identifying and sequencing the necessary steps for solving problems (Csizmadia et al., 2015; Curzon et al., 2014). Evaluation checks the functionality and correctness of the algorithms developed and the codes written (Schneider & Gersting, 2018; Yadav et al., 2018). Testing and debugging are essential for evaluation (Kılıç et al., 2021). Generalization is applying a method suitable for solving one problem to solve different problems (Barr et al., 2011; Curzon et al., 2014). Automating algorithmic structures or code blocks using them in different parts of the same software is also called generalization (I. Lee et al., 2011).

Different approaches are used to develop CT-related skills. Weinberg (2013) stated that without a computer activity, general programming, robot programming, or STEM applications can develop CT. Since it is considered to be directly related to CS, programming activities are used more in research on CT development (Atmatzidou & Demetriadis, 2016; Lye & Koh, 2014; Rodríguez-Martínez et al., 2020; Wang et al., 2021). Table 1 shows the skills associated with CT and the assessment methods used.

Table 1. CT-related components and assessment methods

Research	Component	Assessment Method
Asbell-Clarke et al. (2021)	Decomposition, abstraction, pattern recognition, algorithm design	BEBRAS tests
del Olmo-Muñoz et al. (2020)	Decomposition, abstraction, algorithmic thinking, evaluation, generalization	BEBRAS tests, Programming test
Mouza et al. (2020)	CT concepts (algorithms, loops, conditionals, variables and so forth.)	Programming test
Rodríguez-Martínez et al. (2020)	CT concepts (sequences, loops, events, conditionals)	Programming test
Sun et al. (2022)	Decomposition, abstraction, algorithmic thinking, evaluation, generalization	BEBRAS tests
Wu and Su (2021)	Decomposition, pattern recognition, abstraction, algorithm design	Programming test
Kuo and Hsu (2020)	Decomposition, abstraction, pattern recognition	BEBRAS tests

As seen in Table 1, programming or reasoning tests are generally used to assess CT. BEBRAS tests, which are applied within the scope of annual events in 59 countries that are members of the organization, are recommended for measuring CT development (Román-González et al., 2018; Tang et al., 2020). The questions in the BEBRAS tests are structured according to decomposition, abstraction, algorithmic thinking, evaluation, and generalization skills. Yükseltürk and Altıok (2022) emphasized that the systematic use of CT components in the solution process of a problem will contribute to programming. Bayman and Mayer (1988) stated that developing skills such as decomposition, evaluation, and debugging is essential for effective programming. These skills are considered basic skills related to CT (Angeli et al., 2016; Csizmadia et al., 2015).

Problem-solving-based programming training

Programming includes conceptual knowledge of syntax structures, semantic knowledge that includes algorithmic structures, and strategic knowledge that deals with problem-solving processes (Bayman & Mayer, 1988; Oliver, 1993). In programming, various strategic methods should be used to overcome the difficulties encountered in problem-solving processes. Providing students with strategies on where and how to use which thinking skill can play a facilitating role in the problem-solving process. Problem-solving approaches structured within the CT framework can affect the development of programming skills (Atmatzidou & Demetriadis, 2016; Bundy, 2007). Kalelioğlu et al. (2016) stated that there are similarities between problem-solving steps and CT skills. They proposed a framework for using CT skills in problem-solving (Fig. 2).

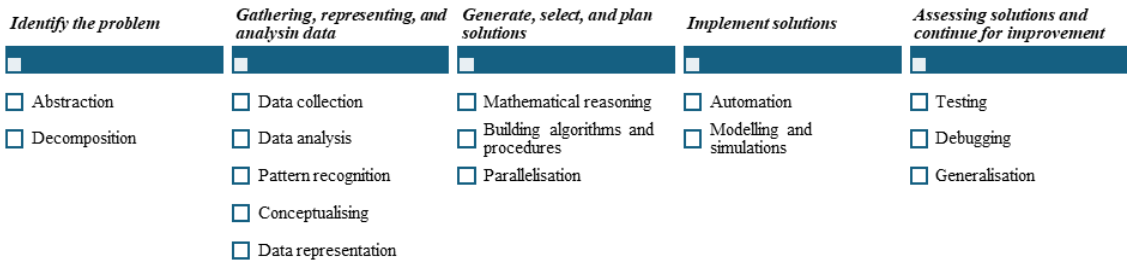

Figure 2. Framework for CT as a problem-solving process

Fig. 2 was obtained by categorizing the skills in the operational definition of CT (ISTE &

CSTA, 2011), which emerged from analyzing CT-related articles (Kalelioğlu et al., 2016). In this study, we performed PSbPT by modifying the framework proposed by Kalelioğlu et al. (2016). In the training process, we examined the development of CT components that support the problem-solving steps shown in Table 2.

Table 2. Relationship between problem-solving and CT components

Problem-Solving Steps	CT Components
Defining and simplifying the problem	Decomposition
Collecting, presenting, and analyzing information for problems	Abstraction
Create solution steps	Algorithmic Thinking
Implementing solution steps	Generalization (automatization)
Evaluating the Solution	Evaluation (testing, debugging)

As shown in Table 2, we hierarchically mapped the problem-solving steps to the CT components in this study. Such mapping approaches are used to solve problems in programming training (e.g., general programming and robotic programming). Kılıç and Çakıroğlu (2023) evaluated the CT development of teachers receiving robotic programming training. They structured robotic task problems hierarchically based on decomposition, abstraction, algorithmic thinking, generalization, and debugging skills. Similarly, Üçgül (2022) proposed robotic programming problems that require the hierarchical use of logical questioning, decomposition, abstraction, algorithmic thinking, debugging, and generalization skills. Yükseltürk and Altıok (2022) provided examples of how decomposition, abstraction, algorithm, automation, and generalization skills can be used to solve traditional programming problems.

Methodology

Research model

This study employed a pretest-posttest control group experimental design to examine the effect of problem-solving-based programming training (PSbPT) on students' computational thinking development. A true experimental design with pre-test and post-test control groups was created by randomly assigning students to two equal groups (Büyüköztürk et al., 2016). The independent variable was the instructional method, with two levels: (a) problem-solving-based programming training (experimental group) and (b) traditional programming training (control group). The dependent variables were (1) programming performance, (2) reasoning skills, and (3) computational thinking components (decomposition, abstraction, algorithmic thinking, generalization, and evaluation).

Participants were 40 vocational school students enrolled in the Computer Technologies department. Due to the limited number of computers available in the laboratory for individual use, students were randomly assigned to either the experimental (n = 20) or control (n = 20) group using a simple randomization procedure. Specifically, all participant names were listed alphabetically and assigned random numbers using a computer-generated randomization sequence. The first 20 students were assigned to the experimental group, and the remaining 20 to the control group. This procedure ensured that each participant had an equal probability of being assigned to either group. Groups were balanced regarding gender (4 females, 16 males in each group) and prior programming experience (minimal to no prior C# knowledge). Students were not given any information about whether they would be in an experimental or control group.

Procedure

This research was conducted over a 14-week academic semester. The experimental procedure consisted of four distinct phases: pretest administration, intervention implementation, posttest administration, and data analysis. Both groups received 1080 minutes of programming training (90 minutes per week). Prior to the intervention, all participants completed two assessment instruments: (1) a programming performance test measuring C# syntax knowledge and visual programming skills, and (2) a BEBRAS-based reasoning skills test measuring computational thinking components. Tests were administered in a computer laboratory setting during regular class hours. No significant differences were found between groups at pretest (see Results section).

The control group received traditional programming instruction following a conventional sequence: (1) theoretical explanation of concepts by the instructor, (2) demonstration of sample applications, (3) individual problem-solving by students, and (4) instructor feedback on solutions. Instruction covered basic C# concepts (variables, operators, decision structures, loops, arrays, methods) and visual programming tools (Button, TextBox, ComboBox, Label, MaskedTextBox, CheckBox, ListBox, RadioButton, ProgressBar, Timer, Chart). The experimental group received problem-solving-based programming training structured around computational thinking components. The instructional process followed five steps mapped to CT components (see Table 2): (1) Complex, real-life programming scenarios were presented (secure password application, states of water, quiz application, cake-making, garden planning, personnel management system). (2) Students used structured problem-solving forms to analyze problems through decomposition (breaking into sub-problems) and abstraction (identifying relevant information). (3) Students developed algorithms using algorithmic thinking skills. (4) Students coded solutions using generalization principles. (5) Students tested and debugged their solutions.

At the beginning of the training, the experimental group received explicit instruction on CT components and their application to problem-solving. The instructor modeled CT-based problem-solving using sample applications. Students then solved problems individually using structured forms, followed by class discussions of different solution approaches.

Table 3. Comparison of instructional approaches

Aspect	Control Group (Traditional)	Experimental Group (PSbPT
Problem presentation	Standard programming exercises	Complex, real-life scenarios
Conceptual framework	Syntax-focused	CT component-focused
Problem-solving process	Unstructured	Structured forms (decomposition → abstraction → algorithm → generalization → evaluation)
Instructor role	Lecturer and demonstrator	Facilitator and modeler
Student activity	Individual coding	CT-based analysis + coding + discussion
Feedback	Individual error correction	Collective solution discussion

Upon completion of the intervention, both groups completed the same programming performance test and BEBRAS reasoning skills test used at pretest. Tests were administered under identical conditions to ensure comparability.

The same instructor delivered instruction to both groups. The instructor held a position in the Computer Technologies department, had ten years of computer science teaching experience, and was actively engaged in computer science education research. This consistency controlled for instructor effects across conditions.

During the 12-week training period, both groups were presented with six different problem situations related to daily life that needed using basic programming concepts and visual programming tools. The problems titled secure password application, states of water, quiz, cake-making, garden planning, and personnel management system were asked at two-week intervals. The students in the experimental group used the structured problem-solving form to solve the given problem situations within the framework of CT components. Kılıç and Çakıroğlu (2023) used a similar form to evaluate CT development through robotic programming. After the students filled in the forms, the instructor asked a few randomly selected students to share their answers with the other students. Thus, a discussion environment was created, and different solutions for the problems were presented. Fig. 3 exemplifies a sample problem situation. The problem is related to array and random structures in programming. Students should use visual programming tools such as Label, TextBox, and Button to solve this problem.

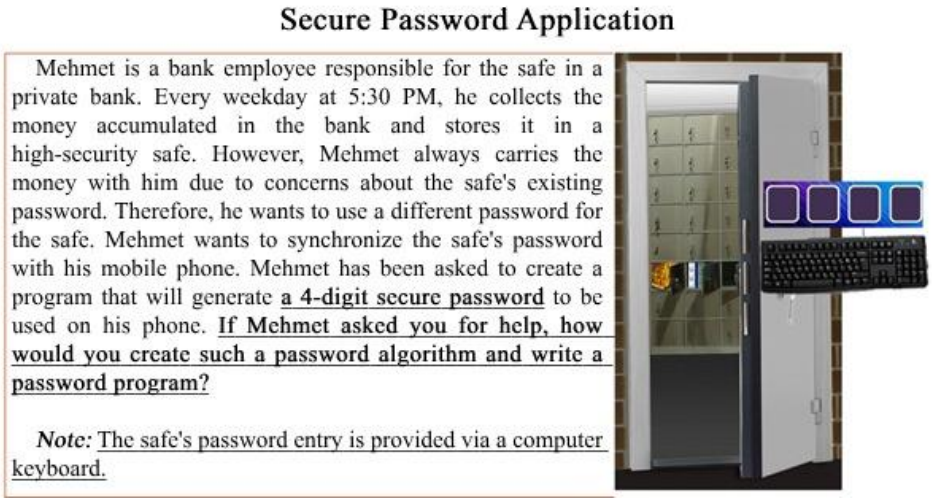

Figure 3. Secure password application

Fig. 4 exemplifies another application. The cake-making application requires using decision structures (if-else, switch-case) and loop (for, while) concepts in programming. Students should use visual programming tools such as ProgressBar, Timer, and Chart to solve this problem.

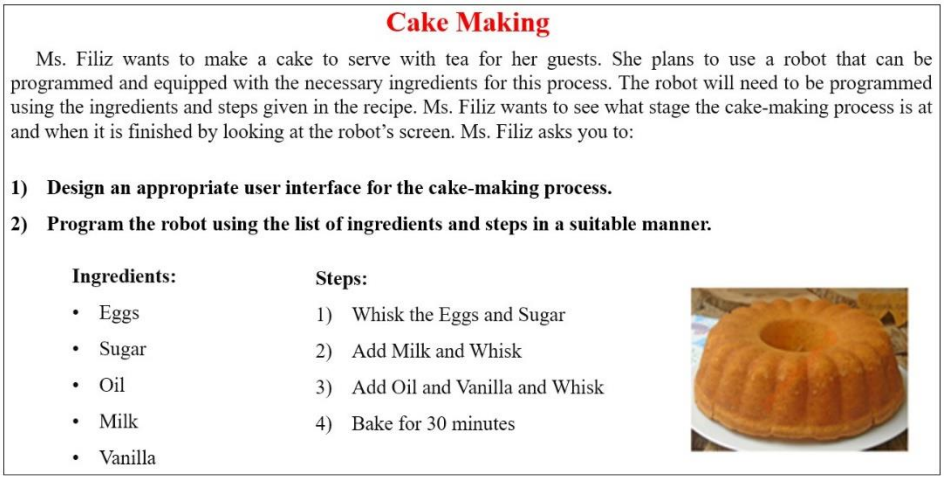

Figure 4. Cake-making application

Fig. 5 and Fig. 6 exemplify the statements the students wrote on the problem-solving form concerning a problem situation within the decomposition framework, abstraction, and algorithmic thinking skills. Decomposition and abstraction are the primary stages necessary to address the problem situation. In Fig. 5, the student decomposed the given problem into specific parts such as questions and answers, correct and incorrect checks, calculation of score, and time. He identified the necessary visual tools, variables, and programming structures in the abstraction stage.

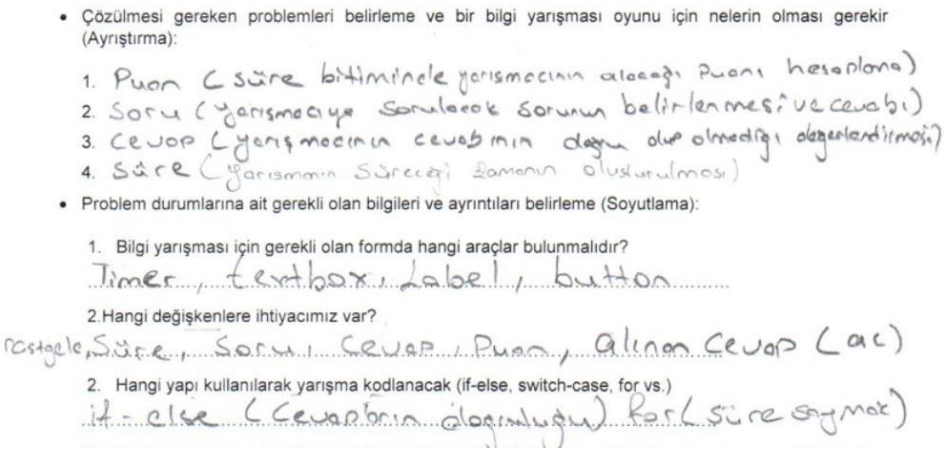

Figure 5. Problem-solving form sample (decomposition and abstraction)

Fig. 6 shows the solution to the given problem within the algorithmic thinking. In Fig. 6, the student designed a form and wrote an algorithm for solving the problem. In this part of the problem-solving form, students designed the program interface they will create within the framework of Visual Programming and wrote the solution steps. The algorithm in Fig. 6 listed information about the program's functioning.

Figure 6. Problem-solving form sample (algorithmic thinking)

Participants

The study participants consisted of 40 students studying in the Computer Technologies department of a vocational school of a state university in Türkiye. The students were randomly divided into groups. There are four girls and sixteen boys in each group. The age range of the students was between 20 and 25. Before the study, a form was applied to the students to learn their demographic information and programming experiences. Most of the students did not know about C# programming language and visual programming tools.

Instruments and item design

This study employed a two-dimensional assessment approach to measure computational thinking development: (1) Contextual application of CT and (2) underlying CT components. Programming Performance measured through the Programming Test, assessing students' ability to apply CT components in actual coding tasks (C# syntax and visual programming). Reasoning skills measured through BEBRAS tests, assessing students' ability to apply CT components in abstract, computer-free reasoning tasks. These two dimensions represent "contextual applications" of computational thinking—programming performance reflects CT in a coding context, while reasoning skills reflect CT in a logical problem-solving context. Both tests were designed to measure five core CT components: decomposition, abstraction, algorithmic thinking, generalization, and evaluation. Table 4 presents the mapping of individual test items to specific CT components. This mapping allows for the calculation of *CT composite scores* for each dimension (programming-based CT and reasoning-based CT), representing the sum of scores across all CT components within each test. Thus, while programming performance and reasoning skills are presented as separate outcomes, they serve as indicators of CT development in different contexts. The CT component analysis (Table 10 and 11) represents a more granular examination of which specific CT skills improved within each context.

Table 4. Relationship of questions with CT components

Instrument	Decomposition	Abstraction	Algorithmic Thinking	Generalization	Evaluation
Programming test	Q4, Q5	Q1, Q2, Q3, Q9	Q3, Q5, Q6, Q7, Q8	Q1, Q9	Q6, Q7, Q8
BEBRAS tests	Q3, Q8	Q1, Q6	Q2, Q3, Q4, Q5, Q6, Q7 Q8, Q9, Q10	Q4, Q5, Q10	Q1, Q4, Q5, Q9

Programming test items

The programming test, developed by the researchers to assess students' programming performance, consists of nine multiple-choice and open-ended questions focusing on basic syntax and visual programming tools. The content validity of the test was established using the Delphi survey technique guidelines proposed by Hasson et al. (2000). Three researchers specializing in computational thinking (CT) analyzed the items to ensure they accurately measured the intended CT components. Based on this expert panel, the Content Validity Index (CVI) was calculated at 0.92, exceeding the standard 0.80 threshold for excellent content validity (Polit & Beck, 2006). The instrument's reliability was evaluated using internal consistency and inter-rater agreement. The Cronbach's alpha coefficient for the programming test was 0.84, indicating high internal consistency for educational measurement (Nunnally & Bernstein, 1994). Furthermore, to ensure the objective scoring of the open-ended questions, two independent researchers scored the responses. The resulting Cohen's Kappa coefficient was 0.81, representing "almost perfect" agreement between the raters (Landis & Koch, 1977). The open-ended questions in the programming test were evaluated using a structured scoring rubric. This rubric assigned partial points based on three main criteria: (1) correct use of programming syntax (e.g., variables, loops, decision structures), (2) the logical flow and efficiency of the proposed algorithm, and (3) the accuracy of the expected output. For instance, if a student provided a correct logical structure but had minor syntax errors, they received partial credit to reflect their strategic knowledge. The scoring process was conducted independently by two researchers, and any discrepancies were resolved through consensus. Fig. 7 shows a sample question in the programming test. Students should find which of these

problem situations are solved by using loops. This question measures the students' abstraction and generalization skills.

Which of the program examples below can be written more concisely using a repetitive (loop) structure? Indicate with an 'X' in the Approval column if applicable. Otherwise, leave it blank.

Problems	Approval
Program that writes numbers from 1 to 20 line by line	
Program that prints the larger of 2 numbers entered from the keyboard	
Program that calculates the sum of 3 numbers entered	
Program that calculates the factorial of the entered number	
Program that calculates a person's age given their birthdate	
Program that prints all even numbers from 0 to 100	
Program that determines if the entered number is odd or even	
Program that generates 5 values randomly between 1 and 10	

Figure 7. Sample programming question about loop

According to the algorithm given below, what will be the Toplam Komisyon if the Fiyat value entered is 40 TL?

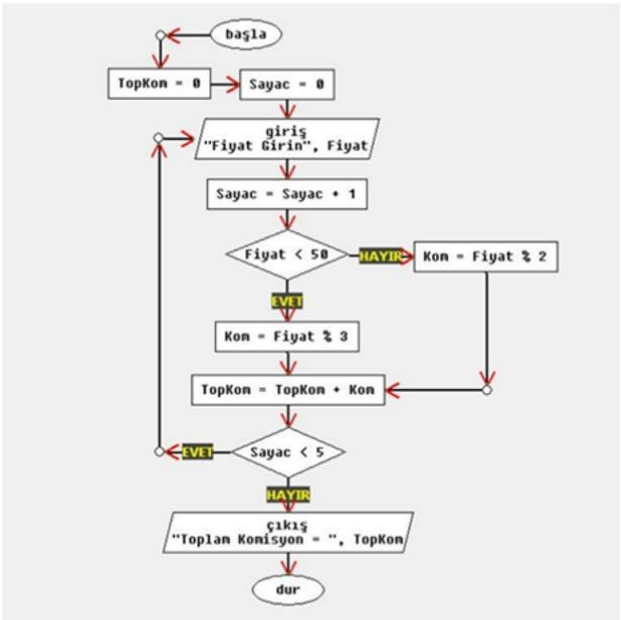

Figure 8. Sample programming question about algorithms

Fig. 8 tests the students' algorithm reading skills. This question also measures students' algorithmic thinking and evaluation skills in the context of CT. Students can obtain 40 points from the nine questions in the programming test. The maximum score for each question varies between 3-6 points. Some questions have only one correct answer (see Fig. 8). Some have more than one, but only correct answers are scored (see Fig. 7).

BEBRAS test items

To assess reasoning skills, 10 multiple-choice and open-ended questions were selected from the official BEBRAS international task pool, varying in difficulty. The BEBRAS tests are internationally recognized for their validity in measuring CT components, including decomposition, abstraction, and algorithmic thinking. For the current study sample (N = 40), the internal reliability of the test was calculated using the Kuder-Richardson 20 (KR-20) formula, yielding a value of 0.78. This value is considered acceptable and consistent with reliability scores reported in similar CT assessment literature (Kuder & Richardson, 1937). Construct validity was further supported by the hierarchical mapping of each question to specific CT skills, as detailed in Table 3. While some questions aim to measure a single CT skill, others measure more than one CT component. Fig. 9 shows sample BEBRAS tests questions.

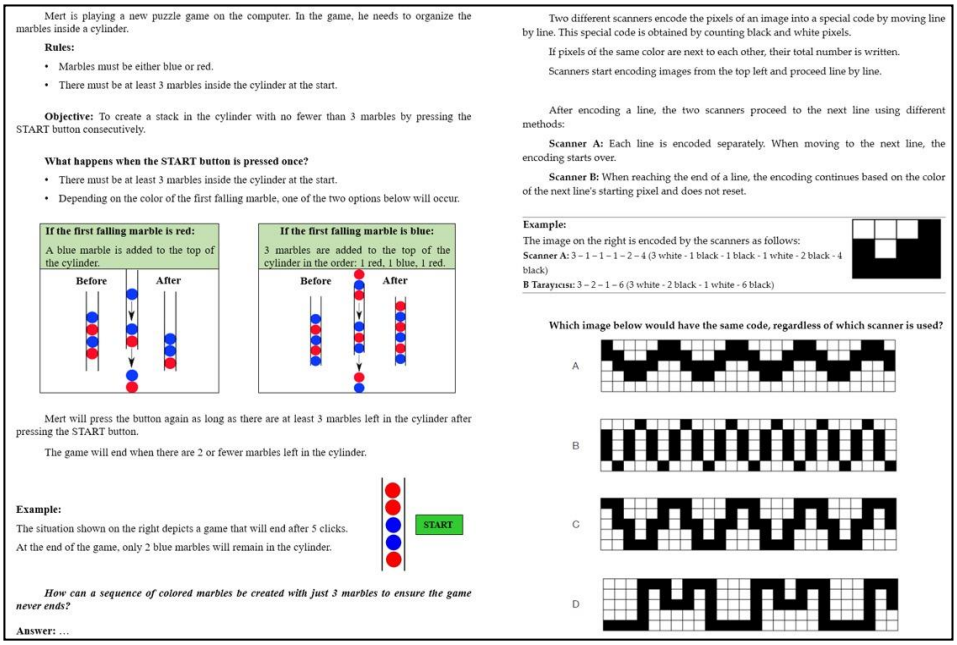

Figure 9. Sample BEBRAS questions related to reasoning skills

The question on the left side in Fig. 9 shows a puzzle game. Students should sort marbles according to the given rules. This question also measures students' decomposition and algorithmic thinking skills in the context of CT (see Table 4). In the question on the right side in Fig. 9, students should find out which images in the options will have the same code when encoded by both scanners.

Data analysis

Prior to hypothesis testing, assumptions for parametric analyses were examined. The Shapiro-Wilk test was used to assess normality due to the small sample size ($n < 50$). Results indicated that pretest and posttest scores for both programming performance and reasoning skills were normally distributed for both experimental and control groups (all $p > .05$). Levene's test was conducted to examine homogeneity of variances. Where Levene's test indicated unequal variances ($p < .05$), Welch's t-test with adjusted degrees of freedom was used instead of the standard independent samples t-test.

Descriptive statistical analyses characterized students' programming performance and

reasoning skill levels. Independent samples t-tests were used to compare group differences at pretest and posttest. Paired samples t-tests examined within-group changes from pretest to posttest. Cohen's d effect sizes were calculated for all significant findings to indicate practical significance (Cohen, 1988): small effect (d = 0.2), medium effect (d = 0.5), and large effect (d = 0.8). All analyses were conducted using SPSS version 23.0, with significance level set at $\alpha = 0.05$.

Results

Considering the research problems, this section presents the findings obtained from the tests (pretest, posttest) for the compared domains for each group. First, the findings regarding the effect of PSbPT on programming and reasoning skills were presented. Then, the results regarding the development of CT components in both contexts were shared.

Programming performance and reasoning skills

Table 5 shows the pretest results for students' programming performance and reasoning skills. The mean score of the students' programming performance pretest was 13.15, and the mean score of the students' reasoning skills pretest was 2.35. The programming performance pretest score of the control group (M=14.40, SD=5.519) was higher than the experimental group (M=11.90, SD=5.046). The reasoning skills pretest score of the experimental group (M=2.60, SD=1.535) was higher than the control group (M=2.10, SD=1.119).

Table 5. Pretest statistics of programming performance and reasoning skills in each group

Skills	Group	N	M	SD	SEM
Programming performance	Control	20	14.40	5.519	1.234
	Experimental	20	11.90	5.046	1.128
	Total	40	13.15	5.371	0.849
Reasoning skills	Control	20	2.10	1.119	0.250
	Experimental	20	2.60	1.535	0.343
	Total	40	2.35	1.350	0.213

M=Mean, SD=standard deviation, SEM=standard error of mean

The independent samples t-test results in Table 6 show no significant difference between the experimental and control groups regarding programming performance (t=1.495, p=0.143) and reasoning skills (t=1.117, p=0.247) before the intervention. This data indicates that the experimental and control groups were equivalent before the intervention. Therefore, we compared the posttest scores of the experimental and control groups to reveal the effect of PSbPT on programming and reasoning skills.

Table 6. Independent samples t-test of pretest programming performance and reasoning skills

Test	Group	Md	*t*	df	p	*d*	%95 CI
Programming performance	Experimental - Control	2.50	1.495	38	0.143	0.47	-0.885, 5.885
Reasoning skills	Experimental - Control	0.50	1.117	38	0.247	0.35	-0.360, 1.360

Md=Mean difference, CI=Confidence Interval, LB=lower-bound, UP=upper-bound

Table 7 shows the posttest results of the students in the sample for programming performance and reasoning skills. The mean posttest score for programming performance was 22.12, and the score for reasoning skills was 3.97. Contrary to the pretest results, the programming performance posttest score of the experimental group (M=26.00, SD=4.877) was higher than the control group (M=18.25, SD=5.820). Similar to the pretest results, the reasoning skills posttest score of the experimental group (M=5.15, SD=1.460) was higher than the control

group (M=2.80, SD=1.321).

Table 7. Posttest statistics of programming performance and reasoning skills in each group

Test	Group	N	M	SD	SEM
Programming performance	Control	20	18.25	5.820	1.301
	Experimental	20	26.00	4.877	1.090
	Total	40	22.12	6.595	1.042
Reasoning skills	Control	20	2.80	1.321	0.295
	Experimental	20	5.15	1.460	0.326
	Total	40	3.97	1.818	0.287

We analyzed pretest and posttest scores with paired samples t-test to check students' programming performance and reasoning skills development. Table 8 shows that the programming performance and reasoning skills of the students in the experimental and control groups who received programming training, even though with different methods, improved significantly. The posttest scores of the students in the experimental group who received programming training based on problem-solving were significantly higher than their pretest scores with a mean difference of 14.10 (SD=5.505) (t=11.454, p=0.000). The programming performance posttest scores of the control group students who traditionally received programming training differed significantly from their pretest scores with a mean difference of 3.85 (SD=2.230) (t=7.718, p=0.000). The results show that the programming performance of the students who received programming training in both ways differed significantly. The reasoning skill posttest scores of the students in the experimental group who received programming training based on problem-solving were significantly higher than the pretest scores with a mean difference of 2.55 (SD=1.637) (t=6.964, p=0.000). The reasoning skills posttest scores of the students in the control group who traditionally received programming training differed significantly from the pretest scores with a mean difference of 0.70 (SD=1.174) (t=2.666, p=0.015). The results show that programming training based on problem-solving improves reasoning skills more than traditional programming training.

Table 8. Paired samples t-test results in each group

Test	Group	N	Post-Pre M	SD	*t*	df	*d*	p
Programming performance	Control	20	3.85	2.230	7.718***	19	1.73	0.000
	Experimental	20	14.10	5.505	11.454***	19	2.56	0.000
Reasoning skills	Control	20	0.70	1.174	2.666*	19	0.60	0.015
	Experimental	20	2.55	1.637	6.964***	19	1.56	0.000

***p < .001, p < .05 (2-tailed)*

In order to investigate the effect of problem-solving-based programming training on the development of programming performance and reasoning skills further, we compared the mean differences between the groups. Table 9 shows the independent samples t-test results on the posttest scores. According to Table 9, the experimental group students' programming performance posttest scores (t = 4.564, p = 0.000) and reasoning skills posttest scores (t=5.334, p=0.000) differed significantly compared to the control group.

Table 9. Independent samples t-test of posttest programming performance

Test	Group	Md	*t*	df	p	*d*	%95 CI
Programming performance	Experimental - Control	7.75	4.564***	38	0.000	1.44	4.308, 11.191
Reasoning skills	Experimental - Control	2.35	5.334***	38	0.000	1.68	1.458, 3.241

Table 10 compares the average difference between the pretest and posttest scores of the experimental and control groups. The analysis revealed that the difference between the two

groups was statistically significant for both programming performance (t=7.717, p=0.000) and reasoning skills (t=4.106, p=0.000). Further analysis of the students' programming performance and reasoning skills before and after the intervention showed that problem-solving-based programming training was more effective than traditional programming training. Students who participated in problem-solving-based programming training showed better programming performance and improved reasoning skills than those who received traditional programming training. Therefore, problem-solving-based programming training can significantly enhance students' programming performances and reasoning skills compared to traditional programming training.

Table 10. Independent samples t-test of post-pretest programming performance and reasoning skills

Test	Group	Md	*t*	df	P	*d*	%95 CI
Programming performance	Experimental - Control	10.25	7.717***	25.076	0.000	2.44	7.514, 12.985
Reasoning Skills	Experimental - Control	1.85	4.106***	38	0.000	1.30	0.937, 2.762

a: Welch's t-test used due to violation of homogeneity of variance assumption (Levene's test, $p < .05$). Degrees of freedom adjusted using Welch-Satterthwaite equation.

CT component developments

Table 11 shows the differences between students' CT skills before the intervention. The t-test results show no significant difference between the experimental and control groups regarding programming performance (t=1.65, p=0.127) and reasoning skills (t=0.817, p=0.419) before the intervention. However, the posttest results show that PSbPT created a significant difference in CT skills for programming performance (t=5.468, p=0.000) and reasoning skills (t=4.992, p=0.000). This result indicates that PSbPT is more effective in developing CT skills than traditional programming training.

Table 11. Independent samples t-test results of CT skills

	Test	Md	*t*	df	p	*d*	%95 CI
Programming performance	Pre	1.65	1.560	38	0.127	0.49	-0.491, 3.791
(Experimental –Control)	Post	6.1	5.468***	38	0.000	1.73	3.841, 8.358
Reasoning skills	Pre	0.80	0.817	38	0.419	0.26	-1.181, 2.781
(Experimental – Control)	Post	4.00	4.992***	38	0.000	1.58	2.377, 5.622

***.001 (2-tailed)

Table 12 shows the results of the paired samples t-test conducted within the framework of CT posttest and pretest score differences within the groups in the context of programming performance and reasoning skills of PSbPT. When we examined the paired samples t-test results, we found that the total CT difference scores of the students in the experimental (t=12.700, p=0.000) and control groups (t=5.534, p=0.000) differed significantly in programming performance. When we analyzed the students' scores in terms of reasoning skills, we observed that there was no significant difference in the CT difference scores of the students in the experimental group only (t=4.595, p=0.000).

Table 12. Paired samples t-test results in each group

Skill	Group	N	Post-Pre M	sd	*t*	df	*d*	p
Programming performance	Control	20	7.00	5.656	5.534***	19	1.24	0.000
	Experimental	20	27.15	9.560	12.700***	19	2.84	0.000
Reasoning skills	Control	20	0.85	3.013	1.216	19	0.28	0.222
	Experimental	20	3.90	3.796	4.595***	19	1.03	0.000

***.001 (2-tailed)

Discussions

Programming performance and reasoning skills development

This study investigated the effect of PSbPT on students' CT development in the context of programming performance and reasoning skills. In this context, we first examined students' programming performance and reasoning skills development. The participants were randomly divided into experimental and control groups within the framework of the pretest-posttest control group experimental design. The control group received programming training without any intervention. The experimental group received PSbPT. Results showed a significant improvement in the students in both groups in the context of developing programming performance and reasoning skills. Since the groups are given programming training, albeit with different methods, a positive development is expected in their programming performance, reasoning and problem-solving skills (Erol & Çırak, 2022; Olsson & Granberg, 2022). Programming requires strategic knowledge as well as conceptual knowledge. Therefore, students use reasoning skills to solve programming problems (Alferes et al., 1996; Clement & Falmagne, 1986). Falkner and Palmer (2009) state that programming activities support students' reasoning skills.

When we compared the students' posttest scores and the differences between the pretest and posttest scores, we observed a significant difference in favor of the experimental group in terms of both programming performance and reasoning skills. While no intervention was made to the students in the control group to solve complex problems during the training process, the experimental group students used a problem-solving form. This form aimed for the students to analyze the problems more planned, systematically, and strategically. In studies using similar forms, it was emphasized that students' strategic knowledge also improved during the solution of programming problems (Kılıç & Çakıroğlu, 2023; Üçgül, 2022; Yükseltürk & Altıok, 2022). Using CT components to solve programming problems helps students develop solution strategies. We think this strategy development is why the improvement in the students' reasoning skills in the experimental group differed significantly from those in the control group. Reasoning is a skill that requires strategic knowledge and is more prominent in complex problem situations. While students do not have much difficulty solving programming problems that require syntax knowledge, they have difficulty solving complex problems that require strategic knowledge and skills (Hoc et al., 1990; Sajaniemi & Navarro-Prieto, 2005). Students who deal with complex problems based on programming within the framework of CT components need to use their reasoning skills more. Accordingly, exposing students to problem-solving processes can be an effective method to develop their reasoning skills.

CT components development

We examined the students' CT development within the framework of the components measured by the programming test and BEBRAS reasoning tests. In the pretests conducted before the intervention, we observed no significant difference between the experimental and control groups regarding CT. These results show that the CT skills of the students in both groups were close to each other. After the intervention, when we compared the pretest and posttest score differences within the groups, we saw that students' CT skills in the context of programming improved significantly in both groups. Programming effectively ensures CT development (Lee et al., 2014; Lye & Koh, 2014). The strategic knowledge required for programming includes high-level thinking skills. Therefore, regardless of the methods used, students who are doing programming have to use these skills. For example, solutions to programming problems are algorithmically transferred to the compiler through coding. In other words, algorithmic thinking is a skill inherent in programming that enables the creation of meaningful structures in problem-solving (Linn & Dalbey, 1985). Syntax errors encountered during the coding process and logical errors that occur during the execution of the code require the use of debugging skills. Testing the correctness of the designed algorithms is a process that requires evaluation skills. In this respect, it is essential to expose students to more complex programming problems and to ensure a higher level of development of CT skills. When we compared the posttest scores of the students between the groups, we can say that the results met this expectation. Within the context of this study, PSbPT appeared effective in supporting students' CT development in the context of programming. The students in the experimental group had the opportunity to practice CT components in complex problems within the framework of the problem-solving form throughout the training. Students divided the problems into parts through decomposition. Through abstraction, they created the structures and algorithms required for each part. They combined the parts through algorithmic design and created a product by practicing through coding. To test the accuracy of the product, they passed it through an evaluation filter and looked for ways to transfer the obtained solutions to different problems. When the differences between the posttest and pretest within the groups were examined, the reasoning-based CT developments of the control group did not differ much. This situation supports the inferences mentioned above. It was observed that the students in the control group, who were trained with the traditional programming method, generally started writing code without much reasoning about the problem. These students are looking for solutions to the errors they encounter during the coding process. Therefore, instead of giving priority to programming problems, students give more priority to errors in the syntax structure of programming. This situation also reveals the reasons for the view in the literature that "even if students know programming syntax structures, they often have difficulty solving complex problems (Aparicio et al., 2018; Gökoğlu & Kılıç, 2023)."

When we examined the CT development of students based on reasoning within and between groups, we observed that the results were significant only in favor of the experimental group. The CT development of the students in the control group in programming performance differed from their CT development in reasoning skills. Although both skills seem related, CT skills acquired in the programming context may not be transferred to solving other problems in the reasoning context (Shen et al., 2022). While some skills in CT may be effective due to the nature of programming, different skills may be more effective in problems based on reasoning and related to different fields. For example, in daily life problems and STEM (Science, Technology, Engineering, and Mathematics) applications, more general CT skills such as Creativity, Algorithmic thinking, Cooperativity, Critical thinking, and Problem-

solving are emphasized (ISTE, 2016; Korkmaz et al., 2017). Swaid (2015) also emphasized skills such as data collection, retrieval, and visualization for STEM. These two contexts were addressed together through PSbPT applied to the experimental group. Through reasoning-based complex programming scenarios, students used their reasoning skills in programming problems. CT and reasoning skills are interrelated. It is crucial to create instructional environments that address both skills together.

The findings of this study should be interpreted within the context of several limitations. First, the sample consisted of 40 vocational school students from a single institution in Türkiye, limiting the generalizability of results to other educational contexts, age groups, or cultural settings. Second, the intervention was conducted within a specific course (C# Programming) over one academic semester; longer-term effects or transfer to other programming languages remain unexplored. Third, while random assignment was used, the small sample size may have limited statistical power to detect smaller effects. Therefore, the strong effects observed should be replicated in larger, more diverse samples before drawing broad conclusions about PSbPT's efficacy. Future research should examine the effectiveness of this approach across different educational levels (e.g., K-12, undergraduate) and programming contexts.

Conclusions and suggestions

Within the context of this study, PSbPT demonstrated greater effectiveness than traditional programming training for developing vocational school students' programming performance, reasoning skills, and computational thinking. These findings suggest that structured problem-solving approaches incorporating explicit CT components may offer advantages over concept-focused instruction, particularly for students facing difficulties at the strategic knowledge level of programming. Therefore, educators need help teaching these concepts and students' understanding of these skills (Yadav et al., 2017). Results may provide some practical suggestions for educators who teach programming and CT. While CT components are primarily theoretical in the studies, there needs to be more certainty about how to use these components in practice. In this study, using CT components in solving programming problems in 12 weeks provides essential information about these components and how to use them in practice. Educators also have significant responsibilities within the PSbPT framework. Educators should include complex programming scenarios based on reasoning when designing questions that measure programming-related outcomes. The reasoning skills developed in this way can contribute to a better understanding of topics in other disciplines.

While this study focused on university-level vocational students, the PSbPT framework may warrant investigation with younger learners who might require additional scaffolding to understand and apply CT components. Furthermore, this study examined five specific CT components; future research might explore additional components such as parallelization or automation. Replication studies with larger, more diverse samples across different educational contexts would strengthen the evidence base for PSbPT's effectiveness.

Statements and declaration

The authors have no relevant financial or non-financial interests to disclose. The authors have no competing interests to declare that are relevant to the content of this article. During the preparation of this work the author(s) used Grammarly to improve writing and grammar. After using this tool, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Declarations

Funding: *The research did not receive any funding support*
Ethics Statements: *This research was conducted in accordance with the permission granted by the Ordu University Social and Human Sciences Research Ethics Committee with decision number 2022-125 dated 01/06/2022.*
Conflict of Interest: *The authors declare that they have no conflict of interest. Authors not involved in any editorial board or guest editorial process that could influence the review or publication of this work.*
Informed Consent: *All participants were provided with the necessary information for the study following the ethics committee's decision, and their consent was obtained.*
Data availability: *The data collected and analysed in the study can be obtained from the corresponding author upon request*

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/COMJNL/BXS074

Alferes, J. J., Pereira, L. M., & Przymusinski, T. C. (1996). Belief revision in non-monotonic reasoning and logic programming. *Fundamenta Informaticae*, *28*(1–2), 1–22. https://doi.org/10.3233/FI-1996-281201

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Educational Technology & Society*, *16*(3), 47–57.

Aparicio, J. T., Aparicio, M., & Costa, C. J. (2018). A virtual robot to support programming learning. *50th International Symposium on Robotics*, 139–142. https://ieeexplore.ieee.org/document/8470585

Asbell-Clarke, J., Rowe, E., Almeda, V., Edwards, T., Bardar, E., Gasca, S., Baker, R. S., & Scruggs, R. (2021). The development of students' computational thinking practices in elementary- and middle-school classes using the learning game, Zoombinis. *Computers in Human Behavior*, *115*, 106587. https://doi.org/10.1016/J.CHB.2020.106587

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, *75*, 661–670. https://doi.org/10.1016/J.ROBOT.2015.10.008

Barbey, A. K., & Barsalou, L. W. (2009). Reasoning and problem solving: Models. *Encyclopedia of Neuroscience*, 35–43. https://doi.org/10.1016/B978-008045046-9.00435-6

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, *38*(6), 20–23.

Bayman, P., & Mayer, R. E. (1988). Using conceptual models to teach basic computer programming. *Journal of Educational Psychology*, *80*(3), 291–298. https://doi.org/10.1037/0022-0663.80.3.291

BEBRAS. (2022). *Task examples*. https://www.bebras.org/examples.html

Boom, K. D., Bower, M., Arguel, A., Siemon, J., & Scholkmann, A. (2018). Relationship between computational thinking and a measure of intelligence as a general problem-solving ability. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 206–211. https://doi.org/10.1145/3197091.3197104

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, *1*(2).

Büyüköztürk, Ş., Çakmak, E. K., Özcan, Ö.E., Karadeniz, Ş. ve Demirel, F. (2016). Scientific research methods (21 st Edition). Ankara: Pegem Academy.

Çakıroğlu, Ü., & Kiliç, S. (2023). Assessing teachers' PCK to teach computational thinking via robotic programming. *Interactive Learning Environments*, *31*(2), 818–835. https://doi.org/10.1080/10494820.2020.1811734

Chang, C. K. (2014). Effects of using Alice and Scratch in an introductory programming course for corrective instruction. *Journal of Educational Computing Research*, *51*(2), 185–204. https://doi.org/10.2190/EC.51.2.C

Clement, C. A., & Falmagne, R. J. (1986). Logical reasoning, world knowledge, and mental imagery: Interconnections in cognitive processes. *Memory & Cognition*, *14*(4), 299–307. https://doi.org/10.3758/BF03202507/METRICS

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking—A guide for teachers*.

Curzon, P., Dorling, M., Ng, T., Selby, C., & Woollard, J. (2014). *Developing computational thinking in the classroom: A framework*.

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education*, *150*, 103832. https://doi.org/10.1016/J.COMPEDU.2020.103832

Erol, O., & Çırak, N. S. (2022). The effect of a programming tool scratch on the problem-solving skills of middle school students. *Education and Information Technologies*, *27*(3), 4065-4086. https://doi.org/10.1007/s10639-021-10776-w

Falkner, K., & Palmer, E. (2009). Developing authentic problem solving skills in introductory computing classes. *ACM SIGCSE Bulletin*, *41*(1), 4–8. https://doi.org/10.1145/1539024.1508871

Garner, S. (2003). Learning resources and tools to aid novices learn programming. *2003 Informing Science + IT Education Conference*, *3*. https://doi.org/10.28945/2613

Gökoğlu, S., & Kilic, S. (2023). Programming learning and teaching of pre-service computer science teachers: Challenges, concerns, and solutions. *E-Learning and Digital Media*, *20*(5), 498–518. https://doi.org/10.1177/20427530221117331

Hasson, F., Keeney, S., & McKenna, H. (2000). Research guidelines for the Delphi survey technique. *Journal of Advanced Nursing*, *32*(4), 1008–1015. https://doi.org/10.1046/J.1365-2648.2000.T01-1-01567.X

Hoc, J. M., Green, T. R. G., Samurçay, R., & Gilmore, D. (Eds.). (1990). *Psychology of programming*. Academic Press.

ISTE. (2016). *ISTE standarts for students*. https://iste.org/standards/students

ISTE & CSTA. (2011). *Computational thinking in K–12 education leadership toolkit*.

Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic J. Modern Computing*, *4*(3), 583–596.

Kılıç, S., & Çakıroğlu, Ü. (2023). Design, implementation, and evaluation of a professional development program for teachers to teach computational thinking via robotics. *Technology, Knowledge and Learning*, *28*(4), 1539–1569. https://doi.org/10.1007/S10758-022-09629-3/METRICS

Kılıç, S., Gökoğlu, S., & Öztürk, M. (2021). A valid and reliable scale for developing programming-oriented computational thinking. *Journal of Educational Computing Research*, *59*(2), 257–286. https://doi.org/10.1177/0735633120964402

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, *72*, 558–569. https://doi.org/10.1016/J.CHB.2017.01.005

Kuder, G. F., & Richardson, M. W. (1937). The theory of the estimation of test reliability. *Psychometrika, 2*(3), 151–160. https://doi.org/10.1007/BF02288391

Kuo, W. C., & Hsu, T. C. (2020). Learning computational thinking without a computer: How computational participation happens in a computational thinking board game. *Asia-Pacific Education Researcher*, *29*(1), 67–83. https://doi.org/10.1007/S40299-019-00479-9/FIGURES/8

Kwon, K. (2017). Novice programmer's misconception of programming reflected on problem-solving plans. *International Journal of Computer Science Education in Schools*, *1*(4), 14–24. https://doi.org/10.21585/IJCSES.V1I4.19

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, *37*(3), 14–18. https://doi.org/10.1145/1151954.1067453

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics, 33*(1), 159–174. https://doi.org/10.2307/2529310

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32–37. https://doi.org/10.1145/1929887.1929902

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child-Computer Interaction*, *2*(1), 26–33. https://doi.org/10.1016/j.ijcci.2014.06.003

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, *20*(4), 191–206. https://doi.org/10.1207/s15326985ep2004_4

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/J.CHB.2014.09.012

Ma, H., Zhao, M., Wang, H., Wan, X., Cavanaugh, T. W., & Liu, J. (2021). Promoting pupils' computational thinking skills and self-efficacy: A problem-solving instructional approach. *Educational Technology Research and Development*, *69*(3), 1599–1616. https://doi.org/10.1007/S11423-021-10016-5/TABLES/9

Marinus, E., Powell, Z., Thornton, R., McArthur, G., & Crain, S. (2018). Unravelling the cognition of coding in 3-to-6-year olds: The development of an assessment tool and the relation between coding ability and cognitive compiling of syntax in natural language. *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, *18*, 133–141. https://doi.org/10.1145/3230977.3230984

Mouza, C., Pan, Y. C., Yang, H., & Pollock, L. (2020). A multiyear investigation of student computational thinking concepts, practices, and perspectives in an after-school computing program. *Journal of Educational Computing Research*, *58*(5), 1029–1056. https://doi.org/10.1177/0735633120905605

Nickerson, R. S. (1983). Computer programming as a vehicle for teaching thinking skills. *Thinking: The Journal of Philosophy for Children*, *4*(3/4), 42–48. https://doi.org/10.5840/THINKING19834310

Nunnally, J. C., & Bernstein, I. R. (1994). *Psychometric theory* (3rd ed.). McGraw-Hill.

Oliver, R. (1993). Measuring hierarchical levels of programming knowledge. *Journal of Educational Computing Research*, *9*(3), 299–312. https://doi.org/10.2190/0LGX-M45X-2WBK-B7A6

Olsson, J., & Granberg, C. (2022). Teacher-student interaction supporting students' creative mathematical reasoning during problem solving using Scratch. *Mathematical Thinking and Learning*, 1-28. https://doi.org/10.1080/10986065.2022.2105567

Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, *1*(1), 95–123. https://doi.org/10.1007/BF00191473/METRICS

Polit, D. F., & Beck, C. T. (2006). The content validity index: Are you sure you know what's being reported? Critique and recommendations. *Research in Nursing & Health, 29*(5), 489–497. https://doi.org/10.1002/nur.20147

Rodríguez-Martínez, J. A., González-Calero, J. A., & Sáez-López, J. M. (2020). Computational thinking and mathematics using Scratch: An experiment with sixth-grade students. *Interactive Learning Environments*, *28*(3), 316–327. https://doi.org/10.1080/10494820.2019.1612448

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/J.CHB.2016.08.047

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction*, *18*, 47–58. https://doi.org/10.1016/J.IJCCI.2018.06.004

Sajaniemi, J., & Navarro-Prieto, R. (2005). *Roles of variables in experts' programming knowledge*. Annual Workshop of the Psychology of Programming Interest Group. https://www.semanticscholar.org/paper/Roles-of-Variables-in-Experts%27-Programming-Sajaniemi-Navarro-Prieto/80c9135a123c0801d393b5a189cec15ee6a315d5

Schneider, G. M., & Gersting, J. (2018). *Invitation to computer science*. Cengage Learning.

Shen, J., Chen, G., Barth-Cohen, L., Jiang, S., & Eltoukhy, M. (2022). Connecting computational thinking in everyday reasoning and programming for elementary school students. *Journal of Research on Technology in Education*, *54*(2), 205–225. https://doi.org/10.1080/15391523.2020.1834474

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/J.EDUREV.2017.09.003

Sun, L., Hu, L., & Zhou, D. (2022). Programming attitudes predict computational thinking: Analysis of differences in gender and programming experience. *Computers & Education*, *181*, 104457. https://doi.org/10.1016/J.COMPEDU.2022.104457

Swaid, S. I. (2015). Bringing computational thinking to STEM education. *Procedia Manufacturing*, *3*, 3657–3662. https://doi.org/10.1016/J.PROMFG.2015.07.761

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798. https://doi.org/10.1016/J.COMPEDU.2019.103798

Tom, M. (2015). Five CS framework: A student-centered approach for teaching programming courses to students with diverse disciplinary background. *Journal of Learning Design*, *8*(1), 21–37. https://doi.org/10.5204/JLD.V8I1.193

Üçgül, M. (2022). Educational robots and computational thinking. In Y. Gülbahar (Ed.), *From computational thinking to programming* (5th ed., pp. 295–318). Pegem Akademi.

Wang, X. C., Choi, Y., Benson, K., Eggleston, C., & Weber, D. (2021). Teacher's role in fostering preschoolers' computational thinking: An exploratory case study. *Early Education and Development*, *32*(1), 26–48. https://doi.org/10.1080/10409289.2020.1759012

Weinberg, A. E. (2013). *Computational thinking: An investigation of the existing scholarship and research* [Doctoral dissertation]. Colorado State University.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wu, S. Y., & Su, Y. S. (2021). Visual programming environments and computational thinking performance of fifth- and sixth-grade students. *Journal of Educational Computing Research*, *59*(6), 1075–1092. https://doi.org/10.1177/0735633120988807

Yadav, A., Krist, C., Good, J., & Caeli, E. N. (2018). Computational thinking in elementary classrooms: Measuring teacher understanding of computational ideas for teaching science. *Computer Science Education*, *28*(4), 371–400. https://doi.org/10.1080/08993408.2018.1560550

Yadav, A., Stephenson, C., & Hong, H. (2017). Computational thinking for teacher education. *Communications of the ACM*, *60*(4), 55–62. https://doi.org/10.1145/2994591

Yükseltürk, E., & Altıok, S. (2022). Block-based programming. In Y. Gülbahar (Ed.), *From computational thinking to programming* (pp. 241–266). Pegem Academy.

Zhang, L. C., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607. https://doi.org/10.1016/J.COMPEDU.2019.103607