Scientific Research Publishing

Open Journal of Social Sciences, 2026, 14(3), 609-629
https://www.scirp.org/journal/jss
ISSN Online: 2327-5960
ISSN Print: 2327-5952

A Study on the Design and Practice of Pair-Programming Instruction for Junior Secondary Students Aimed at Developing Computational Thinking

Xiulin Ma[1,2], Tianwen Su[1], Fei Wu[1*]

[1]School of Education Science, Xinjiang Normal University, Urumqi, China
[2]School of Educational Technology, Beijing Normal University, Beijing, China
Email: *1626351246@qq.com

How to cite this paper: Ma, X. L., Su, T. W., & Wu, F. (2026). A Study on the Design and Practice of Pair-Programming Instruction for Junior Secondary Students Aimed at Developing Computational Thinking. *Open Journal of Social Sciences, 14,* 609-629.
https://doi.org/10.4236/jss.2026.143033

Received: February 5, 2026
Accepted: March 27, 2026
Published: March 30, 2026

Abstract

With the promulgation and implementation of China's Education Modernisation 2035 and the Compulsory Education Information Technology Curriculum Standards (2022 Edition), computational thinking has become an essential core competency for Chinese students in the digital age. However, information technology teaching in junior secondary schools across the northwest region faces numerous practical challenges, including inadequate cultivation of computational thinking, inefficient collaborative learning, and superficial learning. This study aims to investigate the impact of Pair-Programming (P-P) for secondary school pupils on the development of computational thinking, self-efficacy, and collaborative abilities, thereby providing new strategies and theoretical support for information technology teaching. Based on social constructivism, the zone of proximal development, and cognitive load theory, this study employed a quasi-experimental design. Three eighth-grade classes from municipal junior high schools in Northwest China were randomly assigned to independent, paired, and group collaborative programming groups. A 12-week instructional intervention was implemented. Utilizing a mixed-methods research paradigm combined with the Computational Thinking Scale (CTS) and other methodologies, the study systematically evaluated the paired programming strategy and its multidimensional impact on student development. Findings indicate: 1) P-P has demonstrated a positive impact on three core dimensions of junior high students' computational thinking development, programming self-efficacy, and collaborative skills in terms of overall effectiveness. 2) Different pairing strategies exert markedly distinct effects on computational thinking and collaborative skills. Pairing arrangements should holistically consider three dimensions-programming proficiency, learning styles, and social

characteristics—with proximal gradient pairing yielding optimal outcomes.

Keywords

Pair-Programming (P-P), Computational Thinking, Self-Efficacy, Collaborative Skills, Quasi-Experimental Study

1. Introduction

1.1. Background

1) Computational thinking has become one of the fundamental competencies for citizens in a digital society

Computational thinking is defined as the mental activities individuals engage in when applying computer science concepts and methods to solve problems, including abstraction, decomposition, modeling, and algorithm construction (Li et al., 2022a). Against the backdrop of continuously evolving information technology, computational thinking has become a key subject-specific core competency in the new information technology curriculum standards and is recognized as one of the essential skills students must possess in the 21st century. The Ministry of Education emphasizes that the development of information and network technologies, coupled with contemporary societal demands, urgently requires enhancing the comprehensive quality of the populace to cultivate innovative talent. UNESCO also notes that computational thinking has become a core competency for learners' flourishing development. Strengthening its educational research has emerged as a mainstream trend.

2) Information technology courses have become the primary platform for cultivating computational thinking

Information technology courses serve as the primary platform for cultivating computational thinking. Having evolved from a specialized field in higher education computer science to a fundamental literacy in the digital age, enhancing students' computational thinking abilities has become an inevitable choice for the development of information technology education in primary and secondary schools (Lye & Koh, 2014). The High School Information Technology Curriculum Standards require guiding students to apply computational thinking to solve problems (Li & Zhao, 2016), while the Compulsory Education Information Technology Curriculum Standards (2022 Edition) further designate computational thinking as a core objective. Currently, many frontline teachers in secondary education are integrating computational thinking cultivation into programming instruction. The author's school is also actively engaged in related practices.

3) Challenges in Cultivating Computational Thinking

However, in the municipal junior high schools of Northwest China where the author is based, information technology education faces numerous practical challenges that fall short of the aforementioned requirements. First, large class sizes

and insufficient teaching staff make it difficult for teachers to provide timely and personalized guidance. Second, collaborative learning is often organized inefficiently. In traditional group programming activities, the "one person operates while others observe" phenomenon is widespread. Most classrooms suffer from "free-riding" issues, with some students not participating in coding or logical discussions throughout the entire session. Third, students' programming abilities exhibit a "bell-shaped distribution." Students with weak foundations have never independently completed a full programming project and may even show significant resistance to programming. Intermediate-level students can only mimic teacher examples and lack the ability to solve problems independently. High-ability students grasp programming concepts and principles clearly but lack deep challenges. Fourth, computational thinking cultivation remains superficial. Analysis of student programming projects reveals that over 90% closely resemble teacher examples, substituting "copy-paste" for deep understanding. Students generally lack problem decomposition awareness, abstract thinking, and algorithm optimization skills.

4) The quality of collaboration within traditional teams is not high

To address these issues, some researchers have attempted to improve through "project-based group programming." However, this approach has revealed new challenges in practice: significant skill disparities among team members lead to uneven collaboration, high communication and coordination costs, and difficulty in measuring individual contributions. The "one-on-one" collaboration strategy of Pair-Programming (P-P) offers a targeted solution to these challenges: Its core advantage lies in the ability to directly solve these problems. By assigning distinct "driver-navigator" roles and implementing a 15 - 20 minute rotation mechanism, it prevents free-riding. One-on-one interaction reduces communication overhead while forcing explicit thinking. Furthermore, "proximal gradient pairing" (high-to-medium or medium-to-low ability combinations) based on the Zone of Proximal Development theory allows high-ability students to consolidate knowledge through teaching while providing personalized support to medium- and low-ability students, thereby preventing communication barriers caused by excessive skill gaps.

1.2. Research Problem

This study aims to provide new theoretical foundations and practical models for P-P instruction, while offering effective teaching strategies for cultivating computational thinking skills in programming education.

Based on the above context, this research focuses on the realities of junior high school information technology education in Northwest China. Centered on "P-P for junior high students oriented toward computational thinking development," it employs a 12-week instructional intervention and quasi-experimental design to address the following three core questions, thereby providing theoretical support and practical strategies for regional programming education:

1) How does P-P instruction promote computational thinking development

among junior high students, and what impact does it have on their programming self-efficacy?

2) What effect does P-P instruction have on the development of collaborative skills among junior high students?

3) Which type of pair organization strategy yields the most significant growth effects for students?

2. Literature Review and Define the Core Concept

2.1. Definition of the Core Concepts

1) Computational Thinking

Computational thinking emphasizes leveraging computer science concepts to address problems across disciplines in computable ways. The problem-solving process involves mental activities such as abstract thinking, problem decomposition, model building, and algorithm design. From its early development to gaining widespread societal recognition, the precise definition of computational thinking has remained a focal point of academic debate. It was not until Professor Zhou Yizhen provided a relatively systematic definition of computational thinking, arguing that the computational thinking employed by computer scientists targets tasks such as problem-solving, system design, and understanding human behavior. This thinking process is grounded in fundamental concepts of computer science and utilizes a broad range of intellectual tools from various computer science domains (Wing, 2006).

This study focuses on programming education. Therefore, computational thinking in this research primarily addresses algorithmic thinking while also incorporating concepts such as abstraction and modeling.

2) Pair-Programming

Pair-Programming (P-P) refers to a collaborative development approach where two programmers work together on the same programming task at a single computer (Zhong & Wang, 2018), dividing roles as "driver" and "navigator" to complete the task. The driver is responsible for writing code and executing operations, while the navigator oversees logic, troubleshoots errors, and provides strategic advice. When encountering difficulties, both parties discuss solutions together. The roles can freely switch based on task requirements and rotate periodically, but both parties should always participate equally and share the programming outcomes.

P-P is a specific form of collaborative learning. However, its one-on-one structure helps mitigate free-riding to a certain extent.

2.2. Research Status

1) Research on Computational Thinking Development

a) Research on the Concept and Implications of Computational Thinking

Zhou Yizhen first defined computational thinking in 2006 as "a way of thinking that applies fundamental concepts of computer science to solve problems, design

systems, and understand human behavior." She described this as "thinking from a computer scientist's perspective," elevating it beyond a purely computer science conceptual framework.

The theoretical substance of computational thinking continues to deepen alongside educational contexts and technological advancements. Luo et al. (2019), by reviewing definitions from domestic and international sources, proposed a "three-dimensional perspective" for computational thinking: from a comprehensive perspective, it represents interdisciplinary thinking abilities; from a professional perspective, it is closely linked to programming literacy; and from a functional perspective, it focuses on specific application skills such as abstraction and algorithms. This classification provides a clear theoretical framework for cultivating computational thinking in primary and secondary schools.

Additionally, other scholars have supplemented the definition of computational thinking with multidimensional perspectives based on their disciplinary backgrounds and interpretations. Li Feng posits that computational thinking is a psychological tool for adapting to an information-driven society, possessing the proto-scientific characteristics of technology. It represents the ability to solve problems using information technology and constitutes a rational interaction process between internal mental information systems and external natural information systems. Bai & Gu (2019) localized Zhou Yizhen's definition, asserting that computational thinking constitutes essential digital literacy for K-12 students. It manifests through four core competencies: "decomposing complex problems into sub-problems (decomposition), extracting key features while ignoring irrelevant information (abstraction), designing step-by-step solutions (algorithm), and verifying solution effectiveness (evaluation)".

Following the 2022 release of the Compulsory Education Information Technology Curriculum Standards, scholars interpreted computational thinking in alignment with these standards, emphasizing practical applicability in educational settings and refining definitions to better fit classroom contexts. Li Feng posits that computational thinking within the new curriculum framework constitutes "cognitive activities where individuals apply computer science methodologies to abstract, decompose, model, and design algorithms during problem-solving." This requires the ability to simulate and validate solutions, reflect on and optimize outcomes, and transfer skills to other domains, establishing it as one of the four core competencies in information technology education (Li et al., 2022a).

b) Research on Computational Thinking Cultivation and Evaluation

In practical application research, scholars worldwide focus on three areas: computational thinking cultivation methods, instructional model development, and support system design. Current studies primarily target K-12 education, emphasizing the close connection between computational thinking and 21st-century core competencies such as innovation, critical thinking, and problem-solving skills. Cultivation models include visual programming and project-based learning. Regarding instructional model development, domestic scholars have created tar-

geted teaching frameworks based on computational thinking characteristics, such as those centered on light gaming and problem-guided approaches (Zeng, 2015; Bao et al., 2015). Domestic research on computational thinking cultivation strategies has formed a multi-pathway system encompassing inquiry-based, task-driven, blended, and other methodologies. Development of computational thinking support systems remains in its infancy, with existing outcomes primarily consisting of online education platforms and gamified learning systems.

Additionally, in computational thinking assessment research, scholars both domestically and internationally have evolved from single-test approaches to comprehensive, multi-dimensional evaluation systems emphasizing formative assessment and multidimensional analysis of competency demonstration.

2) Research on Pair-Programming

a) Research on the Concept and Essence of Pair-Programming

The essence of P-P lies in "two developers sharing a single computer to collaboratively complete the entire programming workflow—from requirements analysis, design, and coding to testing." Its core characteristics are real-time collaboration and mutual review, fundamentally enhancing development quality and efficiency through dual-person interaction. The concept of P-P was first introduced among the 12 Extreme Programming (XP) practices (Beck, 1999). As a core XP practice, scholar Zhang (2008) was the first to systematically introduce this concept in China. Zhong Bochang et al. define P-P in robotics education as "a collaborative model where two learners jointly complete robot assembly and programming tasks." Its core lies in achieving complementary skills through role division, preventing free-riding, and enhancing both collaboration and programming abilities (Zhong & Wang, 2018).

b) Current Research Status at the Application Level of Pair-Programming

Compared to traditional group collaboration models, P-P effectively mitigates free-riding. It establishes distinct "driver-navigator" roles: the driver handles coding operations while the navigator reviews logic and offers suggestions. Roles rotate every 15 - 20 minutes or after completing key tasks, preventing role stagnation and ensuring both participants remain deeply engaged throughout.

P-P strengthens students' retention and comprehension of computer science concepts while significantly enhancing computational thinking and reasoning skills. Furthermore, this approach effectively reduces learners' apprehension toward programming, boosts confidence and self-efficacy during learning, and facilitates deep programming engagement by enabling meaning construction through social interaction—helping students transcend individual learning limitations.

2.3. Theoretical Foundations

1) Social Construction Theory

Social construction theory posits that knowledge is not passively received but actively constructed through social interaction. Learners collaboratively build understanding and meaning of concepts by communicating, negotiating, and dia-

loguing with others. Language serves not only as a tool for communication but also as a medium for thought. When learners express their ideas and exchange them with others, they not only foster social interaction but also advance their own cognitive development. Learners can progressively acquire knowledge and skills while constructing their identity through interaction with more experienced members. Social constructivist theory offers a crucial perspective for P-P instruction: successful pairing enables two members to think collaboratively, making their tacit knowledge explicit and forming shared understanding. This mode of knowledge construction aligns closely with the social learning emphasized by social constructivism.

2) Collaborative Learning Theory

Collaborative learning theory originates from social constructivism, centering on learners constructing knowledge through collaborative interaction. It emphasizes active interdependence, individual responsibility, and group engagement. This theory provides a practical foundation for P-P, as the pairing model fulfills the core elements of collaborative learning. It facilitates novice growth through "cognitive apprenticeship"-style guidance while satisfying students' needs for autonomy and competence, thereby stimulating intrinsic motivation. The approach requires balancing individual accountability with group objectives, enhancing interaction efficiency through task design.

3. Research Scheme and Design

3.1. Choice of Research Object

This study selected eighth-grade students from three classes at Middle School B in City A, Western China, with 42 students per class. The pre-test analysis of ability differences across the three classes is presented in Table 1, consistent with the quasi-experimental research paradigm.

Table 1. Pre-test variability analysis results for various abilities across three classes.

	Class (Mean ± Standard Deviation)			F	p
	Experimental Class (n = 42)	Control Class A (n = 42)	Control Class B (n = 42)		
Computational Thinking	3.24 ± 0.40	3.09 ± 0.44	3.22 ± 0.51	1.308	0.274
Bebras International Test Questions	23.10 ± 9.50	23.41 ± 9.52	22.67 ± 8.68	0.068	0.934
Self-efficacy	2.96 ± 0.56	2.84 ± 0.63	2.90 ± 0.61	0.410	0.664
Collaborative skills	2.67 ± 1.00	2.61 ± 0.86	2.72 ± 0.98	0.143	0.867

*$p < 0.05$, **$p < 0.01$.

The rationale for selecting eighth-grade students is twofold: on one hand, they possess foundational computer operation skills, making them suitable for programming instruction; on the other hand, eighth grade marks a critical period for

abstract thinking development and represents a pivotal opportunity for cultivating computational thinking. After obtaining informed consent from the school, students, and parents, three classes were randomly assigned to three teaching conditions: paired programming, independent programming, and group programming. They were designated as the experimental class, control class A, and control class B, respectively.

3.2. Research Framework Design

Based on the research subjects and objectives, this study designed the research framework shown in Figure 1. The core of the framework is to explore the impact of different programming learning modes (paired programming, independent programming, and group collaborative programming) on the development of computational thinking among junior high school students, while also examining changes in programming self-efficacy and collaborative abilities.

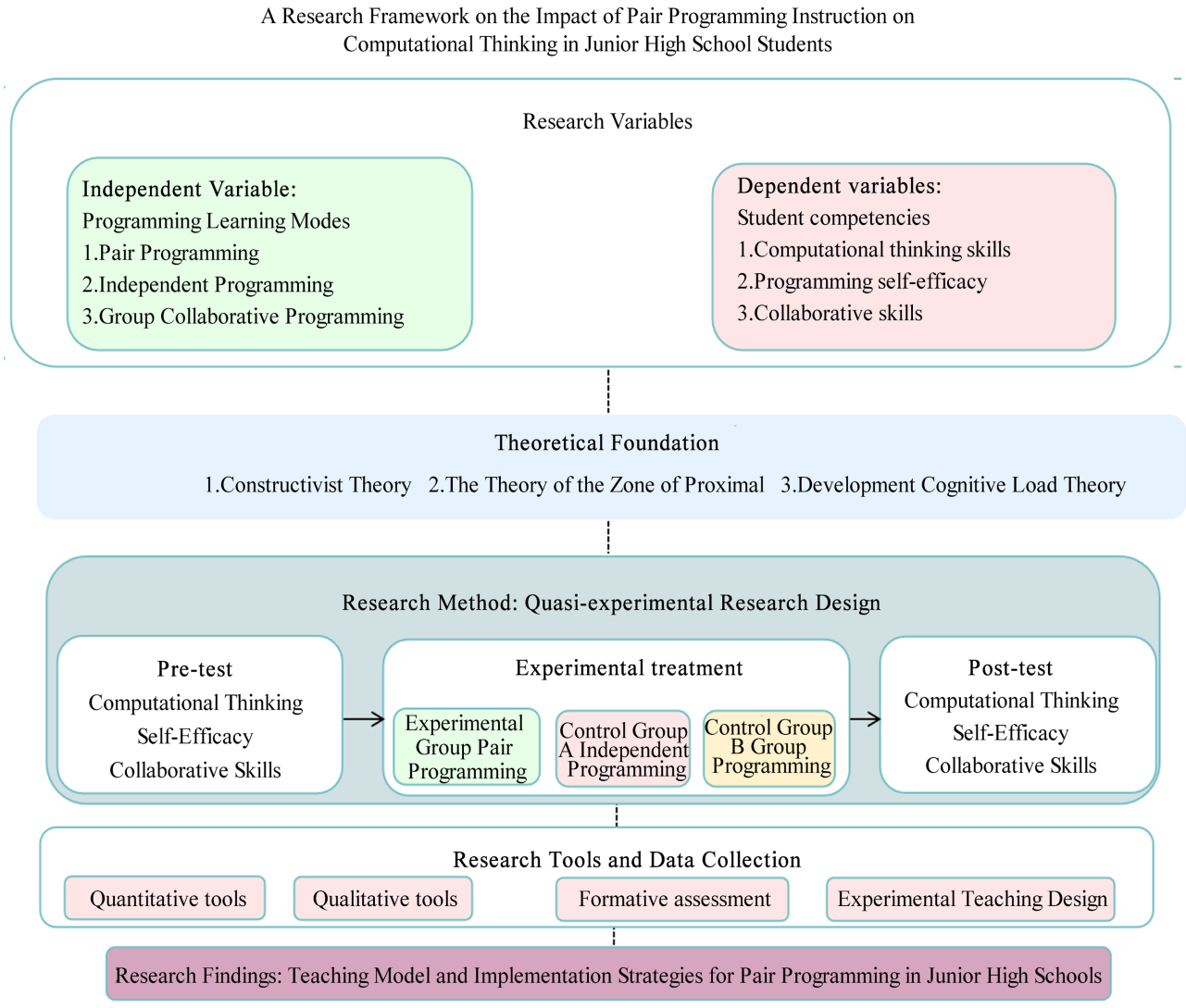

Figure 1. Experimental research framework.

3.3. Character Pairing Mechanism

1) Selection and Basis of Pairing Strategies

According to Vygotsky's Zone of Proximal Development theory, heterogeneous pairing is more conducive to the growth of weaker students, as higher-ability learners can create an ideal learning environment for lower-ability learners. However, Webb's research found that when the ability gap is too large, communication and understanding become hindered, resulting in collaboration outcomes falling short of expectations. Building on these findings, scholars proposed "adjacent gradient pairing," which matches students at adjacent proficiency levels. This approach creates developmental opportunities while ensuring effective communication. Further validation of this strategy showed that high-ability students reinforce their knowledge through teaching, medium-ability students receive targeted support, and low-ability students avoid over-reliance while gaining appropriate guidance (Li, 2018).

Beyond ability levels, research indicates that other non-cognitive factors such as learning styles significantly influence pairing outcomes. Pairing programming requires integration with periodic assessments and dynamic adjustment mechanisms to optimize learning effectiveness.

2) Role Rotation Strategy

To prevent role stagnation, role rotation employs a four-trigger mechanism: rotating roles every 15 - 20 minutes, and additionally after completing minor tasks or key functions. Emergency rotation is initiated when the current driver encounters a significant technical roadblock they cannot overcome independently. Furthermore, instructors guide timely role rotations based on observed pairing dynamics.

In summary, the paired programming groups employed heterogeneous pairing, matching students with differing but not overly disparate foundational levels (high-medium, medium-low). The experimental class comprised 42 students, forming 21 pairs. Foundational level categorization was based on pre-test scores and teacher assessments, dividing students into high, medium, and low tiers. Once formed, pairings remained relatively stable, with adjustments made only under exceptional circumstances (e.g., prolonged absences) during the experiment.

To validate the impact of different pairing methods on learning outcomes, pairs 1 through 9 adopted a high-medium pairing model, pairs 10 through 18 adopted a medium-low pairing model, and pairs 19 through 21 adopted a high-low pairing model.

3.4 Research Tools

1) Teaching and Research Platform for Pair-Programming

During the experimental teaching process, the "Mind+ Visual Programming Software"—a domestically developed youth programming software under the DFRobot platform—was selected based on comprehensive considerations including difficulty level and interface design.

2) Computational Thinking Assessment Tools

This study primarily aims to cultivate computational thinking among junior high school students, making the measurement of computational thinking proficiency a critical indicator for evaluating research outcomes. To this end, the Computational Thinking Scale (CTS), the Beras International Computational Thinking Challenge, and the Programming Self-Efficacy Scale were selected as assessment tools. Relevant research tools are detailed in Table 2.

Table 2. Computational thinking measurement tools.

Measuring Tools	Tool Introduction
Computational Thinking Scale—CTS Scale	The 29-item Computational Thinking Scale (CTS) developed by Korkmaz et al. comprises five core dimensions: Creativity (8 items), Algorithmic Thinking (6 items), Teamwork (4 items), Critical Thinking (5 items), and Problem Solving (6 items). It employs a five-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree).
Beras International Computational Thinking Challenge	The Bebras test comprises multiple real-world scenario-based problems covering computational thinking dimensions such as algorithms and data structures, logical reasoning, and problem decomposition. This study selected test questions appropriate for junior high school students to assess changes in their computational thinking abilities through pre- and post-experiment evaluations. The chosen questions feature moderate difficulty and evenly cover multiple key dimensions of computational thinking, aiming to evaluate students' computational thinking skills through authentic real-world problem scenarios.

During their development and application, the CTS Scale, Beras International Computational Thinking Challenge, and Programming Self-Efficacy Scale underwent rigorous validity testing and have been widely adopted in academic circles, demonstrating sound validity. Therefore, this paper will not elaborate further on this matter.

3) Learning Effectiveness Measurement Tool

Table 3. Learning outcomes measurement tools.

Measuring Tools	Tool Introduction
Programming Self-Efficacy Scale	The Computer Programming Self-Efficacy Scale (CPSES) developed by Kukul, Gökçearslan, and Günbatar: This scale is designed to assess students' self-efficacy in computer programming learning, specifically their confidence in their ability to complete programming tasks. The CPSES comprises 32 items, and this study selected 15 items adapted to the characteristics of junior high school students. These items are scored using a five-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree).

Continued

Classroom Observation Record Form	This study aims to systematically collect objective data on student collaborative behaviors during classroom instruction to evaluate the impact of P-P on the quality of student collaboration, learning engagement, and problem-solving abilities. The observation framework encompasses five dimensions: task allocation and role execution, learning engagement and participation, problem-solving and collaborative efficiency, P-P-specific phenomena, and overall observation evaluation.
Semi-Structured Interview Guide	Through face-to-face discussions, gain a deeper understanding of students' collaborative efforts during paired programming, the challenges they encounter, their problem-solving strategies, their comprehension of programming tasks, and their reflections. The focus will be on five key areas: collaborative experiences, encountered difficulties, problem-solving approaches, understanding of programming tasks, and reflections and summaries.

To ensure the scientific rigor and accuracy of the evaluation of teaching and research effectiveness, the author drew upon the latest findings in computational thinking assessment, adopted established measurement scales, and developed additional evaluation tools. The scientific validity and reliability of these tools were verified through reliability and validity analyses. Relevant research tools are presented in Table 3.

4. Teaching Practice and Effectiveness Analysis

4.1. Teaching Practice and Key Steps

This study designed a 12-week programming curriculum based on the Compulsory Education Information Technology Curriculum Standards (2022 Edition) and the characteristics of junior high school students. The curriculum balances foundational knowledge, computational thinking development, and practical applications, with progressively increasing difficulty. The specific curriculum design is shown in Table 4. The experiment spanned 12 weeks, divided into a preparation phase (3 weeks), a learning phase (8 weeks), and a post-test phase (1 week).

Table 4. Experimental schedule.

Stage	Weekly	Main Content
Preparation Phase	1	1. Completed Beras International Computational Thinking Challenge Paper A 2. Completed Computational Thinking CTS Test 3. Completed Programming Self-Efficacy Scale
	2	1. Complete the "Programming Proficiency Assessment," "Learning Style Questionnaire," and "Social Characteristics Evaluation" 2. Form pairs and groups according to the strategy

Continued

	3	1. Teach students the fundamentals of P-P, emphasizing the roles and responsibilities of partners 2. Explain the basic operations and usage of the graphical programming platform
Learning Stage	4	Understanding Artificial Intelligence: Fundamental Concepts and Applications of AI, Intelligent Voice Assistants That Greet You
	5	Smart Waste Sorting Assistant: Application of Artificial Intelligence in Waste Classification, Conditional Statements, and Use of Loop Structures
	6	Smart Weather Forecasting Assistant: Acquisition and Processing of Weather Data, Application of Variables and Data Processing
	7	Smart Health Monitoring Assistant: An App for Health Monitoring, Variable and List Functions, Data Collection and Analysis
	8	Intelligent Traffic Signal Controller: Control Logic and Integrated Applications for Traffic Signals
	9	Intelligent Voice Translation Assistant: Fundamental Principles of Voice Translation, Application of String Processing and Conditional Judgment
	10	Smart Voice Unlocker: The Simple Principle and Comprehensive Applications of Voiceprint Recognition
	11	Smart Campus Navigation Assistant: Fundamentals of Navigation, Applications in Map Creation and Path Planning
Post-Test Phase	12	1. Completed Beras International Computational Thinking Challenge Paper B 2. Completed Computational Thinking CTS Test 3. Completed Programming Self-Efficacy Questionnaire 4. Student Interview

1) Practical Schedule

In accordance with quasi-experimental research requirements using the "experimental class-control class" design, the experimental and control classes should exhibit no significant differences in all aspects except for their differing perspectives on "paired programming." To ensure internal validity of the experiment, all three groups utilized identical teaching content, learning tasks, 40-minute instructional periods, and evaluation criteria, with instruction delivered by the same teacher. The teaching process was supported by the Mind+ graphical programming platform.

2) Principles Guiding Instructional Content Design

a) Spiral Progression: Core concepts reappear at varying depths and complexities across different stages, forming a spiral learning path.

b) Task-Driven Approach: Each theme is driven by concrete tasks and projects,

emphasizing real-world problem-solving.

c) Contextual Relevance: Programming tasks connect to students' daily lives and subject learning, enhancing meaningful engagement.

d) Differentiated Adaptation: Each theme offers foundational, intermediate, and challenging tasks to accommodate students at varying proficiency levels.

e) Computational Thinking Focus: Clearly defines computational thinking elements cultivated in each unit to ensure cognitive skill development.

4.2. Data Analysis of Practical Outcomes

1) The Impact of Pair-Programming Instruction on Computational Thinking Development in Junior High School Students

a) Analyzing the Effectiveness of Pair-Programming from the Perspective of Computational Thinking Ability Enhancement among Different Student Types

At the conclusion of the study, computational thinking abilities at the cognitive level were assessed using the Computational Thinking Scale (CTS), while problem-solving abilities at the behavioral level were evaluated through Bebras problems. Independent samples t-tests were conducted on students from different groups, with the results presented in Table 5.

Table 5. Comparative analysis results of computational thinking post-test data.

Independent Samples t-Test Analysis Results (Group Comparison)					
Evaluation Dimensions	Experimental Class (n = 42) M ± S	Control Class A (n = 42) M ± S	Control Class B (n = 42) M ± S	Experimental Class vs. Control Class A (T-value/*p*-value)	Experimental Class vs. Control Class B (T-value/*p*-value)
Computational Thinking (Cognitive) CTS	3.81 ± 0.69	3.49 ± 0.51	3.52 ± 0.45	2.417/0.018*	2.315/0.024*
Computational Thinking (Behavior) Bebras	30.36 ± 7.60	25.85 ± 11.45	26.63 ± 8.71	2.106/0.039*	2.101/0.039*

*$p < 0.05$, **$p < 0.01$.

As shown in Table 5, the post-test CTS score for the experimental class (paired programming) was 3.81 ± 0.69, significantly higher than that of the independent programming control class A (3.49 ± 0.51) and the small-group collaborative programming control class B (3.52 ± 0.45). The t-test results between the experimental class and both control classes reached statistical significance (t = 2.417, $p < 0.018 < 0.05$ and t = 2.315, $p = 0.024 < 0.05$). $p = 0.018 < 0.05$ and (t = 2.315, $p = 0.024 < 0.05$), respectively). In the Bebras test, the experimental class scored 30.36 ± 7.60, also significantly higher than control class A (25.85 ± 11.45) and control class B (26.63 ± 8.71). The t-test results between the experimental class and both control classes reached statistical significance (t = 2.106, $p = 0.039 < 0.05$ and (t = 2.101, $p = 0.039 < 0.05$)), indicating that paired programming outper-

forms traditional programming instruction in both cognitive understanding and practical application of computational thinking.

b) Growth Analysis of Students' Computational Thinking Skills

The growth analysis of students' computational thinking skills is primarily demonstrated through pre- and post-test changes among students in Class 3. The data analysis results are shown in Table 6.

Table 6. Comparative analysis of computational thinking measurements.

Paired t-Test for Comparative Analysis (Pre- and Post-Test Comparison)						
Measurement Dimensions	Class Grouping	Pre-Test (M ± SD)	Post-Test (M ± SD)	Difference (Pre-Test - Post-Test)	T-Value	*p*-Value
Computational Thinking Skills (CTS)	Experimental Class	3.24 ± 0.40	3.81 ± 0.69	−0.57	−4.167	0.000**
	Control Class A	3.09 ± 0.44	3.49 ± 0.51	−0.40	−3.753	0.001**
	Control Class B	3.22 ± 0.51	3.52 ± 0.45	−0.30	−3.082	0.004**
Computational Thinking Competency (Bebras)	Experimental Class	23.10 ± 9.50	30.36 ± 7.60	−7.26	−4.216	0.000**
	Control Class A	23.41 ± 9.52	25.85 ± 11.45	−2.44	−1.156	0.255
	Control Class B	22.67 ± 8.68	26.63 ± 8.71	−3.95	−2.146	0.038*

*$p < 0.05$, **$p < 0.01$.

As shown in Table 6, the posttest-pretest difference in CTS scores for the experimental class was 0.57 (t = −4.167, $p < 0.001$), indicating a significantly greater improvement than that observed in Control Class A (posttest-pretest difference of 0.40) and Control Class B (posttest-pretest difference of 0.30). In the Bebras test, the experimental class achieved a post-test minus pre-test difference of 7.26 (t = −4.216, $p < 0.001$), while control class B recorded a difference of 3.95 (t = −2.146, $p = 0.038$), while the improvement in control class A did not reach statistical significance ($p = 0.255 > 0.05$). This indicates that paired programming has a particularly pronounced effect on enhancing computational thinking behavioral application skills.

2) Analysis of the Value of Pair-Programming Instruction in Cultivating Self-Efficacy among Junior High School Students

a) Demonstrating the Effectiveness of Paired Programming Through Post-Test Differences in Self-Efficacy Data among Diverse Student Groups

Table 7. Comparative analysis of post-test data on self-efficacy.

Independent Samples t-Test Analysis Results (Group Comparison)					
Evaluation Dimensions	Experimental Class (n = 42) M ± S	Control Class A (n = 42) M ± S	Control Class B (n = 42) M ± S	Experimental Class vs. Control Class A (T-Value/*p*-Value)	Experimental Class vs. Control Class B (T-Value/*p*-Value)
Programming Self-Efficacy	3.51 ± 0.43	3.18 ± 0.71	3.28 ± 0.54	2.560/0.013*	2.218/0.029*

*$p < 0.05$, **$p < 0.01$.

At the conclusion of the study, the Programming Self-Efficacy Scale was administered to measure the impact of different instructional models on middle school students' programming self-efficacy. The results of the data analysis are presented in Table 7.

Table 7 shows that the post-test score for programming self-efficacy in the experimental class was 3.51 ± 0.43, significantly higher than that of Control Class A (3.18 ± 0.71) and Control Class B (3.28 ± 0.54). Furthermore, the t-test results between the experimental class and both control classes reached statistical significance (t = 2.560, $p = 0.013 < 0.05$) and (t = 2.218, $p = 0.029 < 0.05$).

b) Growth Analysis of Students' Programming Self-Efficacy

Analysis of experimental data revealed changes in students' programming self-efficacy before and after the intervention. Self-efficacy data results are presented in Table 8.

Table 8. Comparative analysis of programming self-efficacy measurement data.

Paired t-Test Analysis Results (Pre- and Post-Test Comparison)						
Measurement Dimensions	Class Grouping	Pre-Test (M ± S)	Post-Test (M ± S)	Difference (Pre-Test - Post-Test)	T-Value	*p*-Value
Programming Self-Efficacy	Experimental Class	2.96 ± 0.56	3.51 ± 0.43	−0.55	−5.040	0.000**
	Control Class A	2.84 ± 0.63	3.18 ± 0.71	−0.34	−4.595	0.000**
	Control Class B	2.90 ± 0.61	3.28 ± 0.54	−0.37	−2.977	0.005**

*$p < 0.05$, **$p < 0.01$.

Table 8 shows that the experimental group's self-efficacy increased by 0.55 points (t = −5.040, $p < 0.001$), which is 62% higher than control class A (0.34 points) and 49% higher than control class B (0.37 points). This indicates that the learning environment created by paired programming has a stronger promotional effect on enhancing self-efficacy.

3) Exploring the Impact of Pair-Programming Instruction on Collaborative Skills Development Among Junior High School Students

a) Analyzing the Value of Pair-Programming from the Perspective of Collaborative Skills Enhancement Among Different Student Types

As a vital component of 21st-century core competencies, collaborative skills were a key focus of this study regarding the effects of P-P on their development. Data analysis results are presented in Table 9.

Table 9 shows that the experimental class achieved a collaborative ability score of 3.56 ± 0.73, exhibiting a highly significant difference compared to control class A (3.00 ± 0.84) (t = 3.272, $p = 0.002 < 0.01$). This indicates that paired programming significantly enhances collaborative abilities, with effects notably more pronounced than those observed in control class B (3.20 ± 0.93, t = 2.02, $p = 0.05$). $p = 0.002 < 0.01$), indicating that P-P significantly enhances collaborative skills. This score was also significantly higher than that of control class B (3.20 ± 0.93, t = 2.020, $p = 0.047 < 0.05$), suggesting that the deep collaboration fostered in P-P

cultivates students' collaborative skills more effectively than the division-of-labor approach in traditional group work.

Table 9. Comparative analysis of collaborative capability measurement data.

Independent Samples t-Test Analysis Results (Group Comparison)					
Evaluation Dimensions	Experimental Class (n = 42) M ± S	Control Class A (n = 42) M ± S	Control Class B (n = 42) M ± S	Experimental Class vs. Control Class A (T-Value/*p*-Value)	Experimental Class vs. Control Class B (T-Value/*p*-Value）
Collaborative Skills	3.56 ± 0.73	3.00 ± 0.84	3.20 ± 0.93	3.272/0.002**	2.020/0.047*

*$p < 0.05$, **$p < 0.01$.

b) Growth Analysis of Students' Programming Self-Efficacy

The difference between post-test and pre-test scores for collaborative ability in the experimental Class was 0.90, with a t-value of 4.077 ($p < 0.001$), indicating a significant increase. In contrast, the improvement in control class A did not reach statistical significance ($p = 0.069 > 0.05$). Although control class B achieved statistical significance ($p = 0.069 < 0.05$), the effect size was smaller than that of the experimental Class. $p < 0.001$), indicating a significant level. In contrast, the improvement in Control Class A did not reach statistical significance ($p = 0.069 > 0.05$). Although Control Class B achieved statistical significance ($p = 0.026 < 0.05$), the improvement was only 0.48 points. These findings suggest that paired programming is a more effective method for cultivating students' collaborative abilities.

4) Qualitative Analysis Based on Classroom Observations and Interviews

a) Classroom Observation Findings

Students in the experimental class demonstrated clear awareness of "driver-navigator" roles during task assignments, rotating roles every 15 - 20 minutes on average. Their interaction frequency and depth significantly exceeded those of the control class. The incidence of "free-riding" decreased compared to the control class, as each student had distinct and irreplaceable role responsibilities, greatly reducing passive reliance on others. P-P significantly enhanced solution diversity and quality, demonstrating clear advantages in error handling. When encountering programming errors, the experimental group's pairs resolved issues in less time. Their pairs typically identified problems at an earlier stage, whereas the control class often began systematic troubleshooting only after multiple failed runs.

Within the experimental class, different pairing methods also yielded notable differences in students' computational thinking, self-efficacy, and collaborative skills, which will be discussed in detail later.

5) Analysis of Student Interview Findings

Interview data indicates that most students believe P-P helped them gain a deeper understanding of programming concepts. This enhanced comprehension was achieved primarily through two mechanisms: first, by explaining their thought

processes to peers, students were compelled to externalize tacit knowledge, thereby deepening their own understanding; second, by listening to peers' explanations, students encountered diverse ways of thinking and problem-solving strategies. P-P fostered positive attitudes toward collaborative learning among students. The vast majority of students indicated that after experiencing P-P, they were more willing to collaborate with peers to solve problems. Many students mentioned that P-P increased their interest in and intrinsic motivation for learning information technology courses. This heightened motivation stemmed partly from the enjoyment of learning through social interaction and partly from the sense of accomplishment gained from successfully solving complex problems.

4.3. Conclusions and Discussion

Based on teaching practice outcomes and data analysis, paired programming has demonstrated overall effectiveness in promoting three core dimensions among junior high school students: computational thinking (both cognitive and behavioral aspects), programming self-efficacy, and collaborative skills.

Simultaneously, observations from the implementation process reveal that factors such as different pairing arrangements, varying teacher intervention styles during paired programming, and learners' study habits significantly influence final outcomes. The depth and breadth of its impact on student development are closely tied to the scientific rigor of pairing arrangements and their alignment with students' cognitive styles.

1) Differentiated Learning Outcomes Reflected in the Grade Gap Between Partners

Practical implementation of paired programming instruction confirms that pairing students with excessive grade disparities is unsuitable, while pairing students with perfectly balanced grades also yields suboptimal results. Conversely, pairs composed of "high-medium" and "medium-lower-medium" level students demonstrate the most effective collaborative outcomes.

During the teaching process, Groups 4, 7, and 9 adopted the "high-medium" pairing approach, while Groups 3, 6, and 8 were formed by "medium-to-below-medium" students. Within these programming pairs, the two students collaborated effectively: First, the slightly stronger student often took on the role of "navigator," while the other served as a collaborator. Since their knowledge levels were not vastly different, their interaction frequency was high. Second, they readily accepted each other's perspectives and approaches during programming. After a period of collaborative coding, they developed strong synergy, maintaining mutual recognition even when switching roles. Third, this pairing method aligns with the "heterogeneous grouping" principle in collaborative learning theory, facilitating effective cooperation.

Teaching practice further confirms that pairing students with vastly different academic levels often leads to the "high-achieving student monopolizing all work while the weaker student becomes a bystander" phenomenon. Under such cir-

cumstances, the weaker student gains little improvement and may suffer psychological harm, undermining the original purpose of paired programming and collaborative learning. Additionally, when pairing students with identical academic levels, two notable phenomena emerge: First, while they may collaborate effectively to complete programming tasks, the collaborative efficacy remains low—failing to achieve the educational goal of "mutual support." Second, they may become competitive, each working independently, thus missing the collaborative learning objective of paired programming. In summary, this "homogeneous grouping" model typically fails to fulfill the core principle of collaborative learning—the educational philosophy of enabling weaker students to overcome their weaknesses through cooperation.

Based on the "proximal gradient pairing" theory, this study focuses on two heterogeneous pairing methods: high-medium and medium-low. By comparing the potential effects of high-low pairing, it clarifies the moderating role of pairing methods on learning outcomes. In high-medium pairings, high-ability students internalized knowledge by explaining programming logic to their medium-ability peers, while medium-ability students received targeted support within their Zone of Proximal Development. Both groups achieved synergistic improvement in algorithmic thinking and problem-solving skills—a finding consistent with Denner et al.'s research on cognitive complementarity in P-P. Medium-low pairings also demonstrated positive effects. Medium-ability students significantly enhanced their sense of responsibility and self-confidence while serving as "navigators." Low-ability students, guided progressively by their peers, reduced reliance on teachers, gradually overcame "copy-paste" learning patterns, and effectively alleviated previous programming-related frustrations.

In summary, the quality and frequency of collaborative interactions in the high-low pairing group were markedly lower than those in the high-medium and medium-low pairing groups. High-low pairings often suffer from collaborative imbalance due to excessive ability gaps. Low-ability students tend to become passive observers with minimal substantive participation, while high-ability students experience learning fatigue from frequent waiting and repetitive explanations. This validates Webb's "ability gap threshold theory"—when the ability disparity between partners exceeds a reasonable range, communication efficiency drops significantly, and collaborative outcomes become less effective than those achieved in near-gradient pairings.

2) Differences in Learning Outcomes Resulting from Pairing Partners' Cognitive Style Variability

Scholar Li (2022b) argues that the effectiveness of P-P is not solely determined by ability levels; non-ability factors such as learning styles and partnership dynamics are equally crucial. In this study, the author also examined learning outcome variations stemming from these characteristics.

Prior to formal teaching implementation, the author expressed concern about whether "field-independent" students could adapt to the paired programming learning environment. This stemmed from observations that in traditional collab-

orative learning settings involving groups of 3 - 4 students, field-independent learners often struggled to integrate into teams and tended to become passive observers in collaborative processes. However, teaching practice demonstrated that in paired programming, field-independent students actively engaged in programming activities and collaborated effectively with their partners. Thus, the learning environment constructed around P-P principles provides an excellent setting and opportunity for cultivating the collaborative skills of field-independent learners.

Additionally, teaching practice confirmed that during the pairing process in P-P, the cognitive style gap between partners should not be too large. While adhering to the principle of "heterogeneous grouping," it is crucial to maintain "moderate gradation and appropriate differences." Directly pairing field-independent and field-dependent students often leads to mutual distrust, resulting in low collaboration effectiveness and undermining the original objectives of organizing P-P instruction.

3) Summary

In summary, the effectiveness of P-P largely depends on the scientific matching of partners. While programming ability was the primary consideration during initial pairings in this study, subsequent research suggests that adjustments should also appropriately account for the influence of students' learning styles and social characteristics. Some empirical studies suggest optimal pairing weightings of: programming ability (50%), learning style (30%), and social characteristics (20%). This weighting distribution was also referenced in the second and third iterations organized by the author (Demir & Seferolu, 2021). Teaching practice confirms that, based on collected data regarding students' programming ability, learning style, and social dimensions, pairing parameters can be calculated for each student using these weighting indicators. Subsequently, student pairings are arranged according to the proximity gradient matching principle, using these pairing parameter values as the foundation.

Additionally, differences in programming ability are the primary factor influencing pairing effectiveness (Denner et al., 2014). Students' cognitive styles and social characteristics also exert a certain degree of influence. However, the influence of cognitive styles and social characteristics is not fixed across different age groups and backgrounds. Therefore, during the pairing process for programming, it is essential to adjust and holistically consider the impact of each factor based on specific circumstances: For research subjects with diverse backgrounds, the weighting values of the three factors may need to be dynamically adjusted as needed to calculate targeted, reasonable, and effective pairing parameters, thereby providing valuable reference data for student pairings.

5. Summary and Reflection

5.1. Research Findings and Innovations

1) Exploration and Establishment of a Stable and Rational Multidimensional Pairing Mechanism

Based on the theory and empirical research of the Zone of Proximal Development, this study constructs a multidimensional pairing mechanism tailored for junior high school students. It comprehensively considers programming ability (50%), cognitive style (30%), and social characteristics (20%). Initial pairings prioritize programming ability, with subsequent adjustments dynamically increasing the weight of other dimensions based on student adaptation. The "proximal gradient pairing" strategy prevents communication barriers and dependency caused by excessive ability gaps while ensuring students of varying levels receive tailored support. Additionally, tailored to 40-minute class periods, the system incorporates four role-rotation methods: fixed time, task completion, challenge resolution, and teacher guidance. Each transition emphasizes the driver summarizing progress and the navigator clarifying focus areas to reach consensus, ensuring seamless role shifts without disrupting learning momentum or outcomes.

2) Establishing a Systematic Pair-Programming Instructional Strategy Framework

This study integrates constructivism, the Zone of Proximal Development, and cognitive load theories to develop a systematic P-P instructional strategy tailored to the realities of western Chinese junior high schools and student characteristics. It constructs a four-tiered, spiraling task structure—foundational, analytical, design-oriented, and innovative—alongside a "microtask-main task" model. This approach aligns with junior high students' cognitive patterns while addressing their weak programming foundations. Three-tiered strategies—code annotation markers, accountability explanation mechanisms, and interlocking evaluation systems—prevent free-riding; the "flowchart exchange" strategy and "sandwich feedback method" (acknowledging strengths first, raising questions second, concluding with suggestions) visualize abstract computational thinking and alleviate communication shyness; A motivation-building strategy is developed across four dimensions: tiered challenges, autonomy in choice, real-world scenario design, and "valuable bug sharing." This approach aligns with the psychological characteristics of students in western regions, enhancing learning motivation and resilience to form a targeted and actionable teaching plan.

5.2. Research Limitations

Although this study achieved significant results regarding the impact of P-P on middle school students' computational thinking, programming self-efficacy, and collaborative abilities, limitations remain: First, the sample was confined to a single region, was limited in scale, and featured a homogeneous cultural background, potentially affecting the generalizability of conclusions; Second, the short intervention period only validated short-term effects, with long-term impacts requiring further tracking. Third, the reliance on quantitative data and interviews limited deep insights into individual student learning experiences and collaborative processes, leaving room for improvement in integrating quantitative and qualitative analyses.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

Bai, X. M., & Gu, X. Q. (2019). Construction and Application of Computational Thinking Assessment Tools for K-12 Students. *Chinese Journal of Educational Technology, No. 10,* 83-90.

Bao, Y., Meng, F., & Zhang, Y. (2015). A "Stepwise" Guided Model for Cultivating Computational Thinking. *Journal of Educational Technology Research, 36,* 87-92+99.

Beck, K. (1999). *Extreme Programming.* Daedalos Consulting.

Demir, Ö., & Seferoglu, S. S. (2021). The Effect of Determining Pair Programming Groups According to Various Individual Difference Variables on Group Compatibility, Flow, and Coding Performance. *Journal of Educational Computing Research, 59,* 41-70. https://doi.org/10.1177/0735633120949787

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair Programming: Under What Conditions Is It Advantageous for Middle School Students? *Journal of Research on Technology in Education, 46,* 277-296. https://doi.org/10.1080/15391523.2014.888272

Li, F., & Zhao, J. (2016). Revision of High School Information Technology Curriculum Standards: Philosophy and Content. *Chinese Journal of Educational Technology, No. 12,* 4-9.

Li, F., Li, D. M., Wei, X. Y., & Zhu, S. (2022a). Developing Key Competencies to Enhance Digital Literacy and Skills: Content Design and Implementation Suggestions for the Compulsory Education Information Technology Curriculum Standards (2022 Edition). *Journal of Teacher Education, 9,* 55-62.

Li, T. T. (2018). *The Effect of Pairing Methods on Chinese College Students' Participation in Peer Peer-Review of English Writing.* Master's Thesis, Shandong University.

Li, T. T., Hao, Q., Wen, Y., Su, X., Fang, Y., & Liu, J. (2022b). The Effect of Paired Programming Based on Learning Styles and Partnerships on Computational Thinking in Elementary School Students. *Journal of Distance Education, 40,* 105-112.

Luo, H. F., Liu, J., & Luo, Y. (2019). Essential Mental Competencies in the AI Era: Computational Thinking. *Modern Educational Technology, 29,* 26-33.

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

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

Zeng, X. (2015). Preliminary Exploration of a "Light Game" Teaching Model Based on Computational Thinking Cultivation. *Vocational Education Forum, No. 11,* 79-82.

Zhang, M. (2008). Research and Extension of Pair Programming. *Computer Systems Applications, No. 4,* 62-64+68.

Zhong, B. C., & Wang, Y. X. (2018). The Effectiveness of Paired Learning Models in Robotics Education. *Modern Distance Education Research, No. 3,* 66-74.