



Understanding undergraduates' computational thinking processes: Evidence from an integrated analysis of discourse in pair programming

Ruijie Zhou^{1,2} · Yangyang Li^{1,2} · Xiuling He^{1,2} · Chunlian Jiang³ · Jing Fang^{1,2} · Yue Li^{1,2}

Received: 3 November 2023 / Accepted: 19 February 2024

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2024

Abstract

Computational thinking (CT), as one of the key skills in the twenty-first century, has been integrated into educational programming as an important learning goal. This study aims to explore CT processes involved in pair programming with the support of visual flow design. Thirty freshmen participated, working in pairs to solve two programming problems. Their discourses were recorded, transcribed, and coded based on a CT framework encompassing cognitive, practical, and social perspectives. Both quantitative and qualitative methods were applied to analyze the data. In particular, Epistemic Network Analysis (ENA) was applied to explore the patterns of their CT processes. The findings revealed that social perspectives emerged the most frequently in all pairs' discourses. The high-level groups (HLGs) focused more on practical and social perspectives whereas the low-level groups (LLGs) emphasized more on cognitive perspectives. The ENA networks revealed that social perspectives mostly centered around cognitive perspectives for all pairs with CT process patterns in HLGs crossing the three perspectives more frequently. In addition, HLGs exhibited a more complicated and developmental trend in solving the two problems, while LLGs displayed a relatively similar CT pattern. The current study provides insights into the design and implementation of collaborative learning activities in educational programming.

Keywords Computational thinking · Discourse analysis · Educational programming · Pair programming

1 Introduction

Computational thinking (CT) has become an essential application skill and mode of thinking that students from primary to college levels must possess (Tang et al., 2020; Zhang & Nouri, 2019) in the internet era. *CT involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental*

Extended author information available on the last page of the article

to computer science (Wing, 2006, p.33). It is generally developed through computer programming education (Lye & Koh, 2014; Tikva & Tambouris, 2021).

Researchers have studied CT processes from cognitive, practical, and social perspectives. First, it involves the use of a set of cognitive skills including abstraction, algorithms, and systemic thinking that is developed and utilized in problem solving (Lai & Wong, 2022; Wing, 2011). Second, it involves the practice of breaking down complex problems into smaller parts, modeling them, designing algorithms, coding-testing-and-debugging, and running the program to obtain the solutions to the problems (Wing, 2006). Third, CT is developed in a social environment with guidance and help from teachers or peers (Vygotsky & Cole, 1978), and its characteristics extend beyond computational knowledge and skills (Wu et al., 2019). Brennan and Resnick (2012) proposed a three-dimension model (e.g., computational concepts, practice, and perspectives) to describe the CT processes of young people (8–17 years old). Their model was validated within a Scratch environment, a visual programming environment empowering novice programmers to conceptualize and create their stories and games. This model offers valuable insights for describing CT processes in the context of educational programming. However, whether the model is applicable in a broader educational programming context needs further investigation. The current study will extend the framework to further explore the interactions among the cognitive, practical, and social perspectives in an educational programming environment at the university level.

Collaborative programming, including pair programming, has been shown to have a positive impact on the development of CT skills (Echeverría et al., 2019; Lai & Wong, 2022; Wei et al., 2021). This approach presents considerable advantages in algorithm design and code review (Wang et al., 2012), contributing to improvements in students' academic achievements and CT (Lai & Wong, 2022; Wei et al., 2021). How the cognitive, practical, and social perspectives in pair programming interact with each other to make a better performance, in particular, the differences between high-level groups and low-level groups, will be studied in the current research.

Both qualitative and quantitative methods are used to describe students' CT processes. Qualitative methods were used to identify a specific or several distinct CT processes a particular group of students has. Quantitative methods were used to investigate the frequencies of various CT processes in complex problem-solving processes and to compare the differences in behavior patterns of different groups of students. Considering the complex and evolving nature of CT, there is a need for analytical methodologies that uncover the interactivity, interdependence, and temporal process features (Rolim et al., 2019). In recent years, Epistemic Network Analysis (ENA) has been used to capture the temporal and dynamic characteristics of problem-solving processes (Siddiq & Scherer, 2017). As a method of learning analytics, ENA enables both quantitative and qualitative analysis of CT processes, given its inherent potential to reveal interactivity and temporal processes in collective problem solving (Swiecki et al., 2020). The current study will use ENA together with the traditional qualitative and quantitative methods to reveal the mechanisms of CT processes in order to gain a deeper understanding of complex problem solving in programming (Israel et al., 2015).

In summary, this study aims to integrate ENA with the traditional qualitative and quantitative methods to study the CT processes of learners during pair programming

from three perspectives and further explore the differences in the CT processes of groups at different performance levels. This study will help us to gain a better understanding of the interaction of CT processes involved in pair programming and the differences between groups at different performance levels, which will help us to provide better guidance to students with difficulties in educational programming.

2 Literature review

This section presents an overview of the relevant literature on CT, CT skills in educational programming, pair programming, and models for describing CT processes.

2.1 Computational thinking

The concept of computational thinking originated from computer science and has been extended to science, technology, engineering, and mathematics (STEM) areas (Mohaghegh & McCauley, 2016). Papert (1980) seemed to be the first to use the term CT, indicating that the use of computers could enhance thinking capabilities and change the pattern of knowledge acquisition. Papert (1993, 1996) further developed the idea emphasizing that CT allowed children to construct meaningful objects through computers. To respond to the need for the development of digital technologies and computational sciences and their wide applications in various fields, CT is a fundamental skill not only for computer scientists (Wing, 2006). As the definition in Wing's (2011) work, indicating that *CT is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent* (p.1). After that, many definitions have been proposed (CSTA & ISTE, 2011; Grover et al., 2015; Selby & Woollard, 2013; Weintrop et al., 2016). Those definitions converge on the consensus that CT is a way of thinking for problem solving (Lodi, 2020). The concept of CT has been extended to refer to problem-solving processes involving some core principles from computer science (e.g. abstraction and algorithm design, decomposition, etc.) (Mohaghegh & McCauley, 2016). CT was rooted in computer science, which now has become an important component of 21st-century skills (Davies et al., 2011; Mohaghegh & McCauley, 2016; Wing, 2006, 2008). CT, the skills to conceptualize and draw abstractions from data sets, and model problems computationally, is taken as one of the ten future work skills (Davies et al., 2011). CT is also listed as one of the learning objectives of STEM-related courses (International Society for Technology in Education (ISTE), 2015; National Research Council, 2012). For example, in *Framework for K-12 Science Education* (National Research Council, 2012), Using mathematics and CT is one of the eight scientific and engineering practices for science education in the USA. In the current study, we shall create a collaborative problem-solving environment in programming to develop the CT skills of undergraduate students. That is to say, CT reflects problem-solving processes in programming, echoing the definition proposed by Wing (2011). We thus intend to employ this concept, framing CT as the thinking process of problem solving in the context of educational programming.

A variety of methods have been used to measure CT in the context of general education and subject domains. In the context of general education, CT tests (Dagienė & Futschek, 2008; Dagienė & Stupurienė, 2016) and self-reported questionnaires and scales (Korkmaz et al., 2017; Yağcı, 2019) were used. For example, Bebras Contest, a competition for pupils to apply CT to the formulation of problem solutions (Dagienė & Futschek, 2008; Dagienė & Stupurienė, 2016). Korkmaz et al., (2017) developed a scale as a summative tool to determine the levels of computational thinking skills (CTS) of students. In the subject domains, mainly in computer sciences, CT tests (Román-González, et al., 2017) and programming tasks/activities (Lui et al., 2020; Werner et al., 2012) were generally used. For example, in the study of Román-González et al. (2017), a Computational Thinking Test (CTt) with 28 multiple-choice items was designed to measure the basic programming abilities of students at 5th to 10th grade levels. In the study of Werner et al. (2012), Fairy Assessment as Alice Program with three tasks was developed to measure algorithmic thinking, abstraction and modeling aspects of students at the ages of 10–14 years. With the development of IT technologies, digital portfolios were used to capture students' development of CT skills by documenting their learning trajectories in creating an electronic textile mural project (Lui et al., 2020). In the current study, our primary focus is on CT processes within programming. In particular, we shall use two programming tasks to measure undergraduate students' CT skills with their discourses and behaviors recorded using screen recording software.

2.2 CT skills in educational programming

Programming provides a conducive learning environment for students' development of CT skills for its unique advantages in logic operations, algorithm design, and reasoning application (García-Peñalvo & Mendes, 2018; Hsu et al., 2018). Therefore, many educators in computer sciences consider programming as the most suitable subject for fostering students' CT (e.g., Buitrago Flórez et al., 2017; García-Peñalvo & Mendes, 2018). Diverse approaches have been proposed to facilitate programming and CT skills (Bers et al., 2014), such as robot-based programming (Atmatzidou & Demetriadis, 2016; Bers et al., 2014; Harvey, 1997) and algorithm-visualization programming (Brennan & Resnick, 2012; Hundhausen et al., 2002). Text-based programming is primarily a way of programming to improve CT (Bai et al., 2021; Sun & Zhou, 2023), and some existing studies explored ways to facilitate educational programming (Cheah, 2020; Dale & Weems, 2014; Fang et al., 2022). Dale and Weems (2014) divided programming into the problem-solving phase and the implementation phase. Student needs to first identify the concept of programming and data structure to generate solutions by developing algorithms in the problem-solving phase, which contributes to the coding process in the implementation phase (Cheah, 2020). The problem-solving phase was further classified into problem identification and flow definition stages, and the implementation phase was further classified into coding and testing stages. The PFCT (problem identification, flow definition, coding, and testing) approach was commonly used to teach programming (Budny et al., 2002), and its effectiveness in fostering students' CT skills has been demonstrated

in the studies of Buitrago Flórez et al. (2017) and Fang et al. (2022). Another useful model for fostering CT in programming is the cognitive apprenticeship (Collins et al., 1991), an instructional strategy that has also been successfully used in programming teaching (Lee, 2011). It comprises modeling, coaching and scaffolding, articulation, reflection, and exploration (Hopcan et al., 2022). Via cognitive apprenticeship, the application of metacognitive skills can reveal cognitive processes and promote reflection during programming (Collins et al., 1991; Plonka et al., 2015). We shall combine the cognitive apprenticeship model with the PFCT approach to design a four-stage process, including problem identification, visual flow design, coding, testing and debugging, and sharing and reflecting.

Flow design determines the solution to the problem and reveals the CT process during programming. Visual flow design scaffolding plays a vital role in algorithm design, bridging the gap between concepts and practice (Charntaweekhun & Wangsiripitak, 2006; Nassi & Shneiderman, 1973; Threekunprapa & Yasri, 2020; Zhang et al., 2021). This scaffolding can help students better understand programming terms, and represent programming logic visually to simulate the dynamic process of programming from a global perspective (Salleh, et al., 2018; Zhang et al., 2021). Consequently, this promotes problem-solving skills and enables participants to break down complex problems into sub-problems, develop algorithms for each sub-problem, and then integrate them into a program (Fang, et al., 2022). Visual flow design will be used in this study to support problem decomposition and formulate solutions for complex programming problems, indirectly supporting students' CT development.

2.3 Pair programming

Pair programming, derived from cooperative learning theory, is a learning method in educational programming that involves two individuals working together to design algorithms, write code, run tests, and develop software for solving problems (Wei et al., 2021). In pair programming, each member plays a distinct role. One acts as the "driver" controlling the process of program design and is responsible for program development and implementation. The other acts as the "navigator" who monitors the driver to identify problems, defines possible strategies, and checks for errors (Hopcan et al., 2022). Often their roles might switch at a specific time when completing a part of the project (Tsan et al., 2020).

Previous studies have demonstrated that pair programming offers great advantages in checking code errors (Wang, et al., 2012), improving problem-solving quality (Demir & Seferoglu, 2021), enhancing students' CT abilities (Wei et al., 2021), and increasing self-efficiency (Hannay et al., 2009). Pair programming allows participants to freely express themselves by connecting, colliding, and integrating different ideas. This leads to a deeper reflection on the knowledge and enhances high-level thinking abilities (Asunda, 2018; Chiu, 2020). Consequently, pair programming plays a positive role in solving problems through social interactions that help learners enhance their CT abilities. While collaborative learning is often employed for its benefits, it is argued that positive learning outcomes are not always guaranteed

(Kreijns et al., 2003; Li et al., 2023). Specifically, the particular methods within collaborative learning that result in superior academic performance are worthy of deeper exploration. Existing studies (e.g., Wei et al., 2021), highlighting the positive relationship between pair programming and CT outcomes, primarily focused on the outcomes of CT and not on the processes that lead to them. Therefore, it is necessary to explore the relationship between CT processes involved in pair programming and academic performance.

2.4 Models for describing CT processes

In educational programming, several models have been proposed to describe CT processes. One of the most widely used theoretical frameworks of CT was developed by Brennan and Resnick (2012). In their models, there were three dimensions of CT: computational concepts, computational practices, and computational perspectives. Computational concepts refer to the concepts employed in programming, which might include sequences, loops, parallelism, events, conditionals, operators, and data. Computational practices refer to the practices designers develop as they program, for example, being incremental and iterative, testing and debugging, reusing and remixing, abstracting and modularizing, etc. Computational perspectives refer to the perspectives programmers form about the world around them and about themselves such as expressing, connecting and questioning. The model proposed by Brennan and Resnick (2012) was widely used (Sáez-López et al., 2016; Wu et al., 2019; Zhong et al., 2016). On the other hand, Chao (2016) used the log data recorded during students' computational problem-solving processes and described their activities in a sequence of designing, composing, and testing processes. Students' activities in the designing process were represented as computational design, which included problem decomposition, abutment composition, and nesting composition. Students' activities in the composing and testing processes were represented as computational practice, which includes sequence, selection, simple iteration, nested iteration, and testing. Students' performance was measured from two aspects (i.e., goal attainment and program size). These two models have notably prioritized programming practices. In the latter model, computational design is linked with the process of abstracting and modularizing within the computational practice featured in the former model. Furthermore, the latter model's practice demonstrates considerable consistency with the computational concepts and practices of the former. Nevertheless, a critical divergence is highlighted by the distinction that the former model accentuates the perspectives within programming, while the latter distinctly underscores programming performance.

However, these models focused on the cognitive and practical perspectives of CT, and more researchers believed that social perspectives should be considered for CT. Kafai (2016) considered CT to social perspective should be reframed as computational participation, indicating moving beyond tools and code to community and context. Lai and Wong's (2022) meta-analysis showed the essence of CT skills in collaborative computational problem-solving from cognitive, social, and affective perspectives. Considering the context of connecting with others, the

three-dimensional CT framework by Brennan and Resnick (2012) was utilized in a collaborative programming environment to explore the CT processes from cognitive, practical and social perspectives.

2.5 Research questions

This study aims to investigate the CT processes of students engaged in pair programming and to describe the differences in CT processes between different academic performance groups, in particular between high-level groups (HLGs) and low-level groups (LLGs). Specifically, this study aims to address the following research questions:

- RQ1: What is the frequency distribution of CT processes involved in pair programming, and to what extent do the processes differ between HLGs and LLGs?
- RQ2: What is the relationship among CT processes involved in pair programming among all groups, and to what extent do the relationships differ between HLGs and LLGs?
- RQ3: How do the CT processes differ between HLGs and LLGs in different problem-solving scenarios during pair programming?

3 Method

3.1 Research context

The study was carried out in the course entitled "C Programming Language Experiment" in the Spring of 2022, lasting for three weeks at a university in central China. The objectives of the course include: (a) To master the basic knowledge of C programming language and the basic methods for computer software design; (b) To learn the basic thinking skills of using the computer to solve real-world problems and abilities to test and modify programs; and (c) To develop students' CT abilities. Implementing this programming course is critical for fostering CT skills—a premise that aligns with our study's goal of exploring CT processes within programming. This study was conducted in the middle of the course after four sessions that provided an overview of the basic C programming environment and programming knowledge.

3.2 Participants

Thirty first-year undergraduate students (15 females and 15 males) majoring in Artificial Intelligence and Data Science volunteered to participate in the study. Their participation in this course is compulsory, aligning with the specialized training goal aimed at enhancing CT skills. Before this study, they had already completed two college mathematics courses (i.e., calculus and linear algebra) and a course entitled Computational Thinking). They were divided into 15 groups after taking into consideration

their performance in the three courses, and their performance in the first four sessions of the course. Each student's score ranged from 0 to 400, with a set value of 100 attributed to three prerequisite courses and the prior performance in the course. In order to ensure homogeneity between pairs, two teaching assistants computed the scores, subsequently categorizing students based on the cumulative score of their CT skills. The fifteen groups were coded as G1, G2, ..., and G15.

3.3 Learning environment

The study was conducted in a computer lab. Each student was equipped with one computer that could provide Xiaoya, a teaching and learning platform. In addition, three learning tools including ProcessOn, Code::Blocks, and EV screen recording were also provided to them.

Xiaoya (<http://www.ai-augmented.com/>) was independently developed by the research team to provide intelligent teaching and learning services for teachers and students in the university (He et al., 2023; Zhang et al., 2023). The teacher could upload all the course materials to Xiaoya and manage coursework and assignments submitted by his/her students. In this study, one member of a group submitted their flowchart and program code to Xiaoya after they finished their programming tasks. In addition, students were required to review other groups' programming work and reflect on their experience in pair programming and collaborative learning in Xiaoya as part of their coursework.

ProcessOn (<https://www.processon.com/>) is an online platform that can be used to draw flowcharts. It includes almost all the flowblock components that students might use in its toolbar. Students could just drag them into the working area to construct their flowchart in their programming processes.

Code::Blocks is a software that provides learners with an environment for programming. Students can do coding, testing, debugging, and finally use the program to solve the problems in their assignment. It has been used by the software designers (Soto & Figueroa, 2018).

EV screen recording, which can be downloaded freely from <https://www.ieway.cn/>, is a software that can be used to record students' discourse and track their behaviors in pair programming. This kind of data will be used to analyze the CT processes of each group.

In summary, the platform Xiaoya, along with the first two tools (i.e., ProcessOn and Code::Blocks) provides support for students to accomplish CT tasks. Moreover, the EV screen recording feature contributes to documenting the participants' CT processes in the programming tasks.

3.4 Project tasks and procedure

Participants worked in pairs to write programs for solving the two tasks (i.e., the Diet Problem and the Kaprekar Problem) shown in Fig. 1. In the programming process, they need to use their knowledge of loop nesting and comparison sorting algorithms, respectively. Such practical application fosters the development of their CT skills.

The current study lasted for three weeks. In the first week, students were taught the principles of pair programming and motivated to engage in interactive communication, including equal dialogue, being open-minded, attentive listening, mutual respect, and encouraging others. The trainer also held a question-and-answer session to ensure that all students understood the requirements. Each pair was also given 15 min to practice.

In the second and third weeks, each pair worked on one of the tasks shown in Fig. 1 following a four-stage process (Fang et al., 2022; Hopcan et al., 2022). The four stages are:

- a) Problem identification. Students read the problem and determined what was the input and what was the output.
- b) Visual flow design. Students decomposed the problem into several sub-problems and used ProcessOn to draw their flowchart to visualize the algorithm for solving the problem.
- c) Coding, testing, and debugging. Students coded based on the flowchart drawn in the previous stage and tested the code. If the code could not run properly, a debugging process would be conducted to refine the code until the problem was resolved. This stage was done in Code::Blocks.
- d) Share and reflect. After completing the task, students were guided to do peer evaluation and select exemplary flow designs and program codes for sharing. Finally, they also needed to reflect on their thinking to consolidate their learning.

Each group needed to submit their flowchart, their programs, and reflections to Xiaoya at the end of the 2nd to 4th stages, respectively. Throughout the whole process, the teacher and her teaching assistants acted as guides, offering individual support to learners who encountered challenges.

4 Data collection and analysis

The collected data encompass group performance data and the discourse data captured via EV screen recording.

Diet Problem:

A girl is restricted to 900 calories per meal because she is on a diet. The main food is a 160-calorie noodle, and the side food includes a 40-calorie orange, a 50-calorie watermelon, one side food, and the total number of servings is no more than 10. The output of the program is the number of servings of noodles, oranges, watermelon, and vegetable and an 80-calorie vegetable. Program the girl to calculate how to choose the food for a meal so that the total number of calories is 900 and the total number of servings is at least one main food and ables that the girl eats each day, and all possible solutions in the order "number of servings of noodles, oranges, watermelon, and vegetable.

Kaprekar Problem:

For any four-digit number, as long as the digits in each digit are not identical, there is a pattern as follows:

- [1] arrange the four digits that make up the four-digit number from largest to smallest to obtain the largest four-digit number made up of these four digits;
- [2] arrange the four digits that make up the four-digit number from smallest to largest to obtain the smallest four-digit number made up of these four digits (if the four digits contain zeroes, the smallest four-digit number obtained is less than four digits);
- [3] find the difference between these two numbers to obtain a new four-digit number (the higher digit zeroes are retained). Repeating the above process, the final result is always 6174, a number known as the Kabrek constant. Write a function to verify the above Kaprekar operation. You are required to first enter a four-digit number and then output the result of each step of the operation until you finally output 6174.

Fig. 1 The two tasks

Group performance was assessed based on the flowchart and program code they submitted to Xiaoya. The flowchart was scored in the following four aspects (Charntaweekhun & Wangsiripitak, 2006; Su et al., 2022):

- (a) Integrity. It refers to the completeness of the flowchart structure which should include the essential parts of a flowchart (McCormick & Ross, 1990; Xiao & Yu, 2017), such as begin-end symbols, additional symbols, and concepts fundamental to a viable problem solution (Nassi & Shneiderman, 1973).
- (b) Rationality. It refers to two aspects: 1) the appropriateness of selected programming symbols (e.g. process symbol, flow line, decision symbol, connector symbol, start & stop symbol, input symbol, and subroutine symbol) for the given problem, and 2) successful execution of the flow to obtain the correct solution (Charntaweekhun & Wangsiripitak, 2006).
- (c) Normalization. It refers to two aspects: 1) legibility of the content and clear presentation of the execution process. Supplementary annotations are allowed to improve clarity, and 2) the flow line is to be connected chronologically from left to right and from top to bottom with connecting lines designed to minimize overlap (Charntaweekhun & Wangsiripitak, 2006).
- (d) Creativity. It signifies the ability to generate diverse and innovative designs for problem solving using new variables and perspectives (Smith & Browne, 1993).

Through the discussion with the expert with more than 20 years of teaching experience, the total score was set at 40, with scores of 8, 16, 8, and 8 for each dimension based on their importance levels.

The program code was scored in the following aspects (Fang et al., 2022; Olsen et al., 2020):

- (a) Accuracy. It refers to the absence of grammatical mistakes in syntax and logic for problem-solving procedures (Fang et al., 2022; Rahman & Nordin, 2007).
- (b) Identifiers naming. It refers to the reasonable naming of the variables or functions that adhere to established naming conventions (Fang et al., 2022).
- (c) Coding style. It refers to the readability and expressiveness of the code, including code indent and code comments (Charntaweekhun & Wangsiripitak, 2006; Fang et al., 2022).
- (d) Execution. It refers to the successful execution of the program and careful evaluation and verification of the output to ensure that it adheres to the specifications outlined in the question prompt of the problem (Olsen et al., 2020).

Similarly, the total score was set at 60, with scores of 20, 10, 10, and 20 for each dimension after the various facts weighing.

The total score for each group was the sum of the scores they received for their flowchart and program code. These scores were then used to classify the groups into high-level groups (HLGs), low-level groups (LLGs), and intermediate-level groups, in a ratio of 2:2:6 (Sun et al., 2022; Zhang et al., 2022).

The scoring was conducted by peer groups and checked by the instructor who taught this course. The peer groups were randomly assigned and required to follow the assessment criteria outlined above to provide feedback to the author group. The author group then assessed the reasonableness of the scores they received. The teacher carefully monitored the scoring process and resolved any conflicts between the author group and the peer group. A final score was determined for each group. We evaluated the participants' satisfaction rate with the scoring procedures after the experiment. The acceptability measure stood at 4.85 out of a possible 5, indicating that the majority of participants held a favorable view of their scoring experiences.

Each pair's working process and their discourse were captured using EV screen recording. Each group engaged in solving one problem was taken as a discourse dataset, therefore there was a total of 30 datasets. Their discourse was recorded, transcribed, and analyzed using NVivo and ENA to describe and compare the CT processes of HLGs and LLGs involved in pair programming.

An integrated analysis approach was used to analyze the discourse datasets from both quantitative and qualitative perspectives (Fig. 2). The integrated analysis method allows us to holistically investigate and understand discourses in collaborative learning (Ouyang et al., 2023). It consisted of three layers, including CT categories (quantitative), CT patterns (both quantitative and qualitative), as well as the microlevel sequence of CT (qualitative) involved in pair programming. The first layer describes CT processes from a statistical perspective, while the second layer portrays chronologically linked CT processes via ENA. The third layer, examined qualitatively, pinpoints specific or multiple unique CT processes evident within student pairs.

In the first layer, the frequencies of CT processes representing cognitive, practical, and social processes were taken by each group and all the groups were counted. The data analysis was conducted in two steps. The first step was to segment each dataset into different utterances using Nvivo and then code based on the coding scheme shown in Table 1. In the initial coding phase, two discourse datasets (one for Problem 1 and one for Problem 2) were randomly selected and coded. The coding scheme of Brennan and Resnick (2012) was thoroughly reviewed, discussed, and modified to ensure its appropriateness and consistency. We considered the categories of computational perspectives of Brennan and Resnick (2012) from the social perspectives within the context of pair programming, redefined the code of Abstracting & modularizing based on the visual

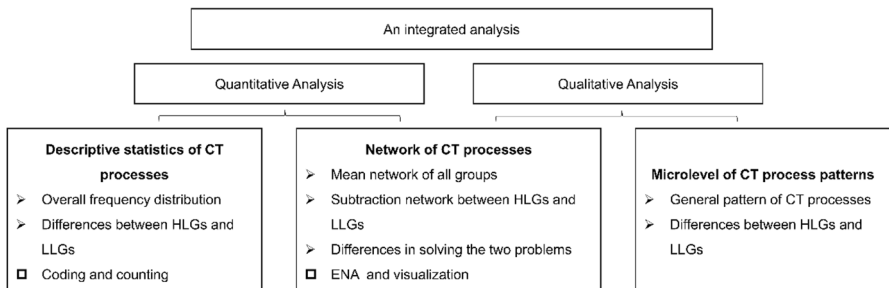


Fig. 2 The analytical framework

flow design stage, and struck out several codes (e.g., parallelism, events, reusing and remixing) that were not found in the current study. After reaching a consensus, the first and second authors independently coded 10% of the entire dataset based on the modified coding scheme. The Cohen's kappa between the two coders was 0.83. The discrepancies were thoroughly discussed and resolved. After that, the first author continued coding the remaining data. The examples included in the last column of Table 1 are examples translated from Chinese. G1-P1-36' indicates that the transcription was from Group 1 in solving the first problem starting at 36'.

In the second layer, this study investigated the network of CT processes of all the groups and a comparison of the CT process networks of different performance groups by examining the epistemic structure of code co-occurrence. ENA was utilized to demonstrate the accumulation of connections between codes and represent them in dynamic network models (Shaffer et al., 2016). To capture, visualize, and quantitatively compare patterns of learning activities across various conditions (Csanadi et al., 2018), an ENA Webkit (epistemicnetwork.org) (Marquart et al., 2018) was utilized. ENA plots were demonstrated to identify and quantify the chronological associations among CT processes in the network model. There are three essential concepts in ENA, i.e., code, unit of analysis, and stanza. Code pertained to a set of conceptual elements, and their interactions served as the focus of analysis. The unit of analysis pertained to the objects of ENA, such as gender groups, achievement groups, and so forth. Lastly, stanza referred to the scope of co-occurrence of cognitive elements. The co-occurrence data was represented by ENA as an adjacency matrix and visualized through normalization, dimension reduction, and singular value decomposition to reveal the relationship among the cognitive elements over a two-dimensional space. In the current study, the three dimensions of CT were codes, groups and a combination of several groups at high- or low-performing levels as the unit of analysis, and four discourses as a stanza window, respectively.

In the third layer, three episodes were identified and analyzed qualitatively to demonstrate the microlevel attributes of CT process patterns as in the study of Borreguero Zuloaga and De Marco (2021). The first episode was selected to demonstrate a general pattern of CT processes for all the groups, whereas the last two episodes were selected to demonstrate the patterns of CT processes of high- and low-performing groups, respectively.

5 Results

The report of the results includes four sections. The first section deals with the categorization of HLGs and LLGs based on their total scores obtained in flowchart designing and program coding. The second section presents the frequency distribution of CT processes and the differences between HLGs and LLGs, which are the answers to the first research question. The third section reports the ENA results and qualitative analysis of three episodes, which are the answers to the second research question. The last section reports the differences in CT process patterns between HLGs and LLGs when solving the two problems, which are the answers to the third research question.

Table 1 The coding scheme and examples

Dimensions	Codes	Description	Illustrative Example
<i>Cognitive perspectives</i>			
Sequences	Cog1	A series of individual steps/instructions that can be executed by the computer	Subsequently, the system will output a list of all possible solutions at this step (G1-P1-36')
Loops	Cog2	A mechanism for running the same sequence multiple times	To exit out of the loop after the final iteration (G1-P1-23')
Conditionals	Cog3	Decisions were made based on certain conditions, which allows different outcomes under different conditions	If the condition is not satisfied, just end it (G13-P1-33')
Operators	Cog4	Numeric and string manipulations are performed on mathematical, logical, and string expressions	The sum is A multiplied by one hundred sixty plus B multiplied by forty. (G2-P1-34')
Data	Cog5	Data involves storing, retrieving, and updating values	At the outset, a is assigned a value of 1, while all others are assigned a value of 0 (G9-P1-10')
<i>Practical perspectives</i>			
Iteration	Pra1	Students make another try by modifying the initial design based on feedback, their experiences, and new ideas in solving a problem	Let's go through the problem again. At the outset, a is set to 1 while all others are set to 0. We can move on from these initial values (G2-P1-38')
Testing & debugging	Pra2	Students run the code they developed, check the appropriateness of the results obtained, try to figure out the errors they might make if the results were not appropriate, and finally modify their previous code	Let's run it and see what happens (G4-P2-35')
Abstracting & modularizing	Pra3	Students determine the appropriate type of algorithms based on their conceptualization of the problem and build something large by putting together collections of smaller parts	We can try out this idea. It seems to require two functions, one 'up' to sort from largest to smallest, and one 'down' to sort from smallest to largest, so we need searching and sorting algorithms (G7-P2-12')
<i>Social perspectives</i>			
Expressing	Soc1	Students express their ideas and/or views on problem-solving approaches, flow design, etc	If we want to achieve the same result with the actual parameter, we must add a pointer and operate on it (G2-P2-23')
Questioning	Soc2	Students discuss the affordances and limitations of algorithms and the reality of the world	Pls think it over, do you think that a header file is needed for the function in this problem (G13-P2-43')?
Connecting	Soc3	Students collaborate on seeking and providing help, appreciating other's ideas, encouraging each other, etc	Uh, what's this equal to (G4-P2-25')?

5.1 Group performance in flowchart designing and program coding

The total scores were used to classify all the groups into HLGs, LLGs, and intermediate-level groups. The range of the total score is from 0 to 200 in the two problems solving. Based on the ratio of the number of groups mentioned above, groups 1 (total score: 198), 4 (total score: 186), and 6 (total score: 180) were classified as HLGs, while groups 3 (total score: 95), 13 (total score: 95), and 12 (total score: 90) were classified as LLGs. A comparison between the HLGs and LLGs was conducted to determine the differences in CT processes involved in programming between HLGs and LLGs.

5.2 Frequency distribution of CT processes revealed from discourses in pair programming

5.2.1 The overall frequency distribution of CT processes

Table 2 presents the numbers and percentages of different CT processes in the discourses of all the groups. A total of 6,790 utterances were recorded with social perspectives accounting for the largest proportion (58.95%), followed by cognitive perspectives (34.42%), and practical perspectives the least (6.63%). Most of the discourses (93.37%) were related to social and cognitive perspectives. In the cognitive perspective, *data*, *conditionals*, and *operators* emerged as the most frequent codes, covering 11.69%, 7.10%, and 6.82% of total utterances, respectively. From the practical perspective, *abstracting & modularizing* was the most frequent (2.78%) while *testing & debugging* was the least (1.74%) of total utterances. From the social perspective, *connecting* and *expressing* were the most frequent, accounting for 31.34% and 23.25% of the total utterances, respectively.

Similar patterns of CT processes in discourses were observed in solving the two problems, their frequencies from the highest to the lowest were in the order of social, cognitive, and practical perspectives. As expected several discrepancies happened in the cognitive perspective due to the differences in the requirement of the two problems. Aforementioned students needed to use their knowledge of loop nesting in solving the Diet Problem, discourses in the conditional, data, and operators categories were the most frequent. While they needed to use their knowledge of comparison sorting in solving the Kaprekar Problem, discourses in the data, operators, and sequences were the most frequent.

5.2.2 Differences in the frequency distribution of CT processes between HLGs and LLGs

Table 3 displays the numbers and percentages of different CT processes in discourses of HLGs and LLGs. A significant difference was found in the distribution of CT processes between HLGs and LLGs ($\chi^2=76.79$, $df=2$, $p<0.001$). The HLGs had higher percentages of discourses in social (60.67% vs. 56.46%, $Z=2.59$, $p<0.05$) and

Table 2 Number (%) of different CT processes in discourses of all groups

Dimensions	Code	Diet Problem No. (%)	Kaprekar Problem No. (%)	Total No. (%)
<i>Cognitive perspectives</i>				
Sequences	Cog1	117 (3.92)	211 (5.54)	328 (4.83)
Loops	Cog2	109 (3.65)	161 (4.23)	270 (3.98)
Conditionals	Cog3	292 (9.79)	190 (4.99)	482 (7.10)
Operators	Cog4	203 (6.80)	260 (6.83)	463 (6.82)
Data	Cog5	265 (8.88)	529 (13.90)	794 (11.69)
Sub-total		986 (33.05)	1351 (35.49)	2337 (34.42)
<i>Practical perspectives</i>				
Iteration	Pra1	46 (1.54)	97 (2.55)	143 (2.11)
Testing & debugging	Pra2	40 (1.34)	78 (2.05)	118 (1.74)
Abstracting & modularizing	Pra3	110 (3.69)	79 (2.08)	189 (2.78)
Sub-total		196 (6.57)	254 (6.67)	450 (6.63)
<i>Social perspectives</i>				
Expressing	Soc1	654 (21.92)	925 (24.30)	1579 (23.25)
Questioning	Soc2	180 (6.03)	116 (3.05)	296 (4.36)
Connecting	Soc3	967 (32.42)	1161 (30.50)	2128 (31.34)
Sub-total		1801 (60.38)	2202 (57.84)	4003 (58.95)
Total		2983 (100)	3807 (100)	6790 (100)

practical perspectives (8.56% vs. 2.92%, $Z=7.21$, $p<0.001$) than the LLGs. whereas, the LLGs (40.62%) had a higher percentage of discourses in cognitive perspective than the HLGs (30.77%) ($Z=-6.23$, $p<0.001$). The above comparison results revealed that (a) the HLGs tended to solve problems more collaboratively; (b) the HLGs took the practical perspectives important and talked more about it; and (c) the LLGs made more effort to the identification of computational concepts, which is the premise of forming flow design ideas and completing programming tasks correctly.

5.3 Network of CT processes in pair programming

5.3.1 Mean network of all groups

A mean network was generated by accumulating all group discourses. It shows not only the CT processes categories identified in all group discourses, but also the connections (co-occurrence relationship) between categories. The resulting network model was then projected onto a two-dimensional graph, using X and Y axes to capture the significant characteristics of the network (Gašević, et al., 2019; Sun, et al., 2022). Figure 3 shows the mean network model of all group discourses projected. We can observe that the CT processes related to cognitive perspectives are mostly on the right side (e.g., Cog1, Cog2, Cog3, Cog4), with those related to social perspectives clustered on the left side (e.g., Soc1, Soc2, Soc3). CT processes related to practical perspectives are distributed across the X -axis.

Table 3 No. (%) of CT processes between HLGs and LLGs

Dimensions	High-Level Groups (HLGs)		Low-Level Groups (LLGs)	
	No	%	No	%
<i>Cognitive perspectives</i>				
Cog1	96	4.89	120	7.01
Cog2	53	2.70	74	4.32
Cog3	120	6.11	157	9.18
Cog4	138	7.03	126	7.36
Cog5	197	10.04	218	12.74
Total	604	30.77	695	40.62
<i>Practical perspectives</i>				
Pra1	52	2.65	4	0.23
Pra2	42	2.14	16	0.94
Pra3	74	3.77	30	1.75
Total	168	8.56	50	2.92
<i>Social perspectives</i>				
Per1	399	20.33	387	22.62
Per2	101	5.15	94	5.49
Per3	691	35.20	485	28.35
Total	1191	60.67	966	56.46

As shown in Table 3, the biggest five nodes are Soc3, Soc1, Cog5, Cog3, and Cog4, which are consistent with the frequencies presented in Table 3.

The co-occurrence between *connecting* (Soc3) and *expressing* (Soc1) is the most significant, which seems to suggest that the two students in each pair tried to understand each other through expressing and connecting. Similarly, the connection between *data* (Cog5) and *connecting* (Soc3) and between *data* (Cog5) and *expressing* (Soc1) is the second most significant which seems to reveal that the pairs tried to make sense of the data in the problems through expressing and connecting. Finally, the connections between two social perspectives (Soc3 and Soc1) and two cognitive perspectives (Cog3 and Cog4) were also important for them to decide what computational knowledge to use for solving the problem. Among the connections between practical perspectives and the other two perspectives, the connection between *abstracting & modularizing* (Pra3) and *connecting* (Soc3) and *expressing* (Soc1) is the highest, which reveals that the pairs discussed more in the abstracting, decomposing, and modularizing processes. The connections among other CT process categories are very faint.

Visual flow design in computational problem-solving entails a process of problem decomposition and algorithm design, the core of which is abstraction and modularization concerning computational practices. In this regard, below (Episode 1) shows G4's discourse in solving the Diet Problem in the visual flow design stage.

S1: It can be implemented with loop nesting.

S2: Well, let's draw the process.

S1: In the beginning, a is set to 1 while all others are set to 0.

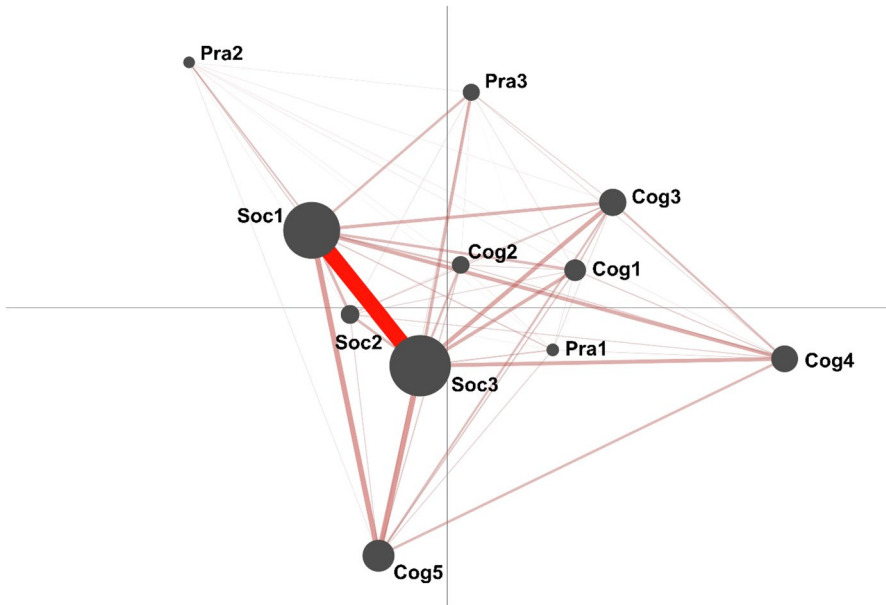


Fig. 3 Mean network of all groups

S2: And then can we judge?

S1: Yes, and then judge whether a is less than or equal to 5.

S2: If it is not less than or equal to 5, it is meaningless to output.

S1: Well, if it is, it will start to enter the loop.

S2: When will they be output?

S1: Continue to judge that they meet this condition, and they will output.

S2: If they are not satisfied, where will they go back?

S1: Go around to this position.

The navigator, S1, first proposed the use of loop nesting (Cog2) to solve problems, and then S2, as the driver, agreed and suggested seeking the path of loop nesting through flow design (Pra3). Next, S1 proceeded to define and initialize variables (Cog5), and then S2 asked for the next step (Soc3). S1 replied positively and gave steps of conditionals (Cog3), after that S2 expressed opinions on judgment results (Soc1). S1 agreed and expressed opinions on loops (Cog2), and S2 subsequently asked for instructions on sequence (Cog1). S1 answered the questions on conditionals (Cog3) and S2 asked for the end of the loops. S1 answered based on the scaffolding and ultimately solved the problem of the return point.

5.3.2 Differences in the network of CT processes between HLGs and LLGs

ENA was applied to further compare the connections and interdependence of CT processes between the HLGs and LLGs in coding for solving the two problems. Figure 4 shows the subtraction network of CT processes between the HLGs and LLGs,

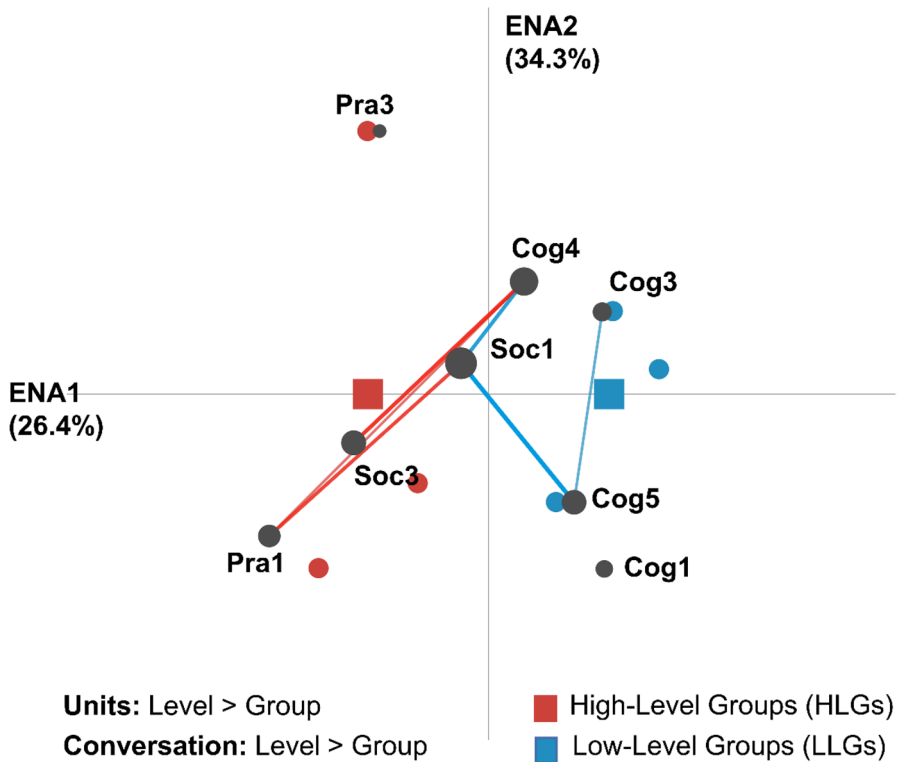


Fig. 4 ENA networks of computational thinking for the HLGs (red) and LLGs (blue)

showing the differences in the patterns of CT process connections between the two clusters. The minimum edge weight was set as 0.13 to strike out the low correlation between CT processes. The HLGs are represented by the red dots and the LLGs by the blue dots. The red square represents the centroid (mean position of the projected points) of HLGs, and the blue square represents the centroid of LLGs. The horizontal axis of the ENA space (ENA1) depicted CT processes as the right side with the computational concepts (e.g., Cog1, Cog3, Cog4, Cog5) and the left side with the computational practices (e.g., Pra1, Pra3) and perspectives (e.g., Soc3).

Mann Whitney U test was used to compare the distribution of projection points between HLGs and LLGs in the ENA space (Swiecki et al., 2020). A significant difference (MDN = -1.50, $N = 3$, $U = 9.00$, $p = 0.10$, $r = -1.00$) was found on the horizontal axis (ENA1) of the ENA space at Alpha = 0.05 level. The subtraction network showed that the strong correlation in the HLGs primarily reflected practical and social perspectives, while those in the LLGs mainly reflected cognitive perspectives.

Moreover, the discourse of the HLGs revealed higher correlations across the three dimensions, especially between *operators* (Cog4) and *connecting* (Soc4) and *iteration* (Pra1), and between *iteration* (Pra1) and *expressing* (Soc1), which seems to suggest that HLGs employed a more systematic approach, discussing more in theoretical construction and practical testing. In contrast, the discourse of the LLGs exhibited stronger

associations between two cognitive perspectives (Cog3-Cog5), and between social and cognitive perspectives (Soc1-Cog4/Cog5). The connections between practical perspectives and others are very faint. It seems to reveal that LLGs committed to solving problems through communicating more on concept recognition. The connection coefficients of the six lines in the ENA networks for both HLGs and LLGs are presented in Table 4.

To further reveal the differences in CT process patterns between HLGs and LLGs, two sets of transcriptional examples were selected for analysis.

The following transcriptional example (Episode 2) illustrates the problem-solving process of group 4 (one of the HLGs) during the coding, testing, and debugging stage for the Kaprekar Problem. S1 served as the navigator and suggested running the program (Pra2), then S2 acted as the driver and compiled the program successfully (Pra2). However, S1 found errors in the running result (Soc3), while S2 believed there was an error in the sorting (Soc1). The pair then proceeded to debug the program step by step after S1 suggested (Pra2). Subsequently, S2 discovered problems with the code (Soc1), while S1 suggested finding problems through the flowchart (Pra3). When S2 expressed confusion about loops (Cog2), S1 provided a viewpoint (Soc1). After S2 made modifications to the code and provided explanations (Pra2), and S1 added conditionals (Cog3), the program was finally executed successfully (Pra2).

S1: Let's try it.

S2: Wow, that's right. Come in and have a look.

S1: Ah, what is this?

S2: Wrong sort.

S1: Let's debug.

[Note] Students debug the program step by step

S2: The first three are all right, but they didn't jump out

S1: Let's see that if it is 6174, it will jump out

S2: Where will it jump from?

S1: Jump here as a semicolon.

S2: If it is satisfied, exit completely.

S1: If not, continue looping.

S2: Wow, we did it.

Episode 2 demonstrated that when program errors emerged during pair programming, the students relied on flow design scaffolding to debug and successfully

Table 4 Connection coefficients of the ENA networks of HLGs and LLGs

Connection	HLGs	LLGs
Cog4-Soc3 (Operators-Connecting)	0.28	0.09
Cog4-Pra1 (Operators-Iteration)	0.13	-
Pra1-Soc1 (Iteration-Expressing)	0.19	0.01
Cog3-Cog5 (Conditionals-Data)	0.17	0.28
Cog4-Soc1 (Operators-Expressing)	0.17	0.28
Cog5-Soc1 (Data-Expressing)	0.26	0.43

HLGs = High-Level Groups, LLGs = Low-Level Groups

identify and solve problems. Working collaboratively, they listened to each other's opinions and reached a consensus, demonstrating effective communication.

The following transcriptional example (Episode 3) depicts the problem-solving process of group 14 (one of the LLGs) during the coding, testing, and debugging stage of the Kaprekar Problem. S1 acted as the navigator and suggested running the program (Pra2), while S2 acted as the driver, and ran the program unsuccessfully (Pra2). Subsequently, S1 suggested using loops for the output array (Soc1) and S2 replied passively (Soc3). When S1 suggested modifying the program (Soc1), S2 became disheartened by the presence of bugs (Soc1). Although S1 later discovered problems in the function header files (Soc1), S2 questioned her (Soc2). Despite S1 making suggestions after answering questions (Soc3), S2 still ignored the errors and ran the program again (Pra2). Unfortunately, the program still failed to run successfully.

S1: Let's run it.

S2: It is wrong.

S1: It has to use a loop for the output array.

S2: Everything is troublesome after learning the loops.

S1: Try to comment on the previous one.

S2: Why was it always wrong?

S1: Wait, we didn't have a function header file.

S2: Why did I use a header file?

S1: It's necessary to use the header file for maximum function. Try it first.

S2: Why could annotated things still appear in the wrong places? No matter if click again, it will run directly

S1: If not, continue looping.

S2: Wrong again.

[Note] Student failed to run the program until the end of the task

It appeared that the pair in Episode 3 relied on trial and error instead of following conventional debugging operations to identify problems at different levels. However, they were unable to come to a consensus and make effective corrections until after completing the task, ultimately leading to the program's failure.

5.4 Differences in CT patterns between HLGs and LLGs in solving the two problems

ENA was further applied to draw the subtraction network between HLGs and LLGs in pair programming for solving the Diet Problem (Fig. 5(a)) and the Kaprekar Problem (Fig. 5(b)).

Network(a) and Network(b) as shown in Fig. 5, independently reveal the CT projected points and their means in ENA space while solving two problems. Similar to Fig. 4, set the minimum edge weight to 0.13, and the HLGs are represented by red dots in the ENA network, while the LLGs are represented by blue dots. The red square represents the centroid of the HLGs, while the blue square represents the centroid of the LLGs. The horizontal axis (ENA1) of the ENA space depicted CT

as the right side with cognitive perspectives (e.g., Cog3, Cog4, Cog5), and the left side with practical perspectives (e.g., Pra1, Pra2, Pra3) during the Kaprekar Problem solving. Whereas in the Diet Problem solving, the same axis was relatively weak in depicting practical perspectives, The vertical axis (ENA2) of the ENA space distributed CT as the upward side with social perspectives (e.g., Soc1, Soc2, Soc3).

The Mann–Whitney U test was used to compare the distribution difference of projection points between HLGs and LLGs in ENA space in solving the two problems. The Alpha=0.05 level of the horizontal axis (ENA1) in the ENA space showed a statistically significant difference both in the Diet Problem (MDN=-1.54, N=3, U=9.00, p=0.10, r=-1.00) and Kaprekar Problem (MDN=-1.94, N=3, U=9.00, p=0.10, r=-1.00).

In the Diet Problem, the discourse of the HLGs revealed higher correlations between social and cognitive perspectives (Soc3-Cog3/Cog5) while LLGs showed a close relationship between two cognitive perspectives (Cog3-Cog4), and between cognitive and social perspectives (Cog4-Soc2). It showed that HLGs committed to discussing the concepts of conditionals and data to solve problems while LLGs argued more about how to operate. In the Kaprekar Problem, the discourse of the HLGs revealed higher correlations among the three dimensions, especially between practical and cognitive perspectives (Pra1-Cog4/Cog5), and between social and practical perspectives (Soc1-Pra1/Pra2/Pra3, Soc3-Pra2), and between cognitive and social perspectives (Cog4-Soc3). While LLGs merely showed a close relationship between cognitive and social perspectives (Cog4-Soc1, Cog5-Soc3). Compared to the Diet Problem stage, the CT processes displayed a more systematic and holistic connection in the subtraction network between HLGs and LLGs in solving the Kaprekar Problem. with complicated and developmental CT process patterns. Specifically, the discourses of HLGs expanded more on practical perspectives, with complicated and developmental CT process patterns. Meanwhile, LLGs showed some differences in social perspective, mainly reflected in the strengthened

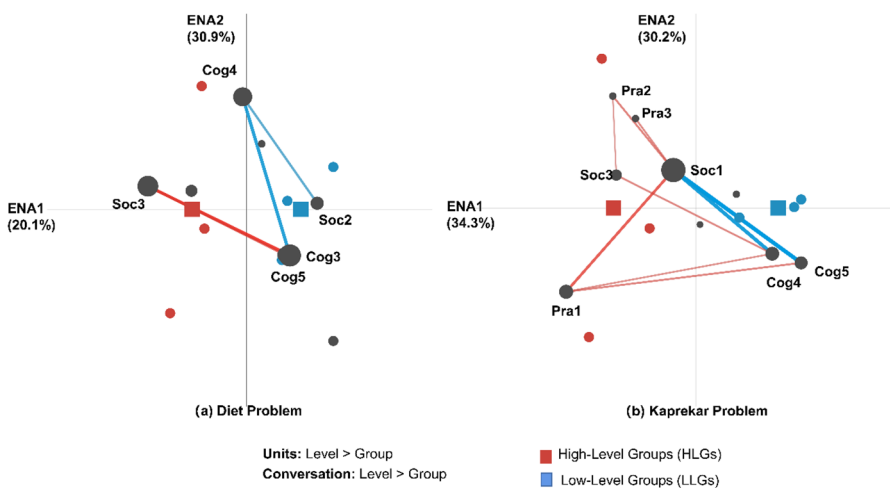


Fig. 5 Subtraction networks of CT between the HLGs (red) and LLGs (blue) in solving the two problems

relationships with connecting and expressing, and the weakened connection with questioning, and their CT process patterns were relatively similar and fixed. The connection coefficients of these ENA networks are shown in Table 5.

To further explain the difference thoroughly in the CT patterns between HLGs and LLGs in pair programming, ENA was used to show the centroid of the CT process between two groups in the ENA spaces from Diet Problem to Kaprekar Problem (see Fig. 6).

The centroid of the ENA network model refers to the median value of the network and depends on the arithmetic mean value of the edge weight of the ENA network model. In Fig. 6, the red dot represents the network centroid of HLGs, while the blue dot represents the network centroid of the LLGs. In the ENA network space, the centroid on the right side indicates stronger and more connection on the right while the centroid on the left side indicates stronger and more connection on the left.

It can be seen from Fig. 6 that the centroids of HLGs are mainly on the left side of the ENA network space, revealing more connections with practical and social perspectives. The centroids of LLGs are mainly distributed on the right side, showing more connections with cognitive perspectives. As shown in Fig. 6, development paths and CT process patterns showed differences between HLGs and LLGs from

Table 5 Connection coefficients of the ENA networks in the HLGs and LLGs in solving the two problems

Task stages	Connection	HLGs	LLGs
CT Processes for Solving the Diet Problem			
	Cog3-Soc3 (Conditionals-Connecting)	0.32	0.13
	Cog5-Soc3 (Data-Connecting)	0.25	0.11
	Cog4-Soc2 (Operators-Questioning)	0.01	0.14
	Cog3-Cog4 (Conditionals-Operators)	0.12	0.32
CT processes for solving the Kaprekar Problem			
	Cog5-Pra1 (Data-Iteration)	0.21	0.02
	Pra1-Soc1 (Iteration-Expressing)	0.28	0.01
	Pra1-Cog4 (Iteration-Operators)	0.17	-
	Pra2-Soc1 (Testing & Debugging-Expressing)	0.21	0.01
	Pra2-Soc3 (Testing & Debugging-Connecting)	0.17	0.03
	Pra3-Soc1 (Abstracting & modularizing-Expressing)	0.18	-
	Cog4-Soc3 (Operators-Connecting)	0.22	0.03
	Cog4-Soc1 (Operators-Expressing)	0.13	0.49
	Cog5-Soc3 (Data-Connecting)	0.18	0.58

HLGs = High-Level Groups, LLGs = Low-Level Groups

the Diet Problem stage to the Kaprekar Problem stage. HLGs began to maintain active communication and ended with debugging and testing. On the contrary, LLGs were committed to the elaboration of the computational concept but lacked practical means to solve problems.

6 Discussion

The study collected students' discourse data and employed an integrated analysis approach from both quantitative and qualitative perspectives to reveal CT processes in pair programming.

Firstly, statistical analysis was conducted to show the frequencies and distribution of CT framework elements. On the whole, pairs interacted more frequently around cognitive and social perspectives, leading to idea exchange and the formation of computational perspectives. This finding is consistent with previous studies (Hopcan et al., 2022; Satratzemi et al., 2021). Peer interaction was reflected in help-seeking and feedback (López-Pellisa et al., 2021), encouragement (Meier et al., 2007), and error review (Satratzemi et al., 2021), highlighting the importance of collaborative problem solving. Comparing HLGs and LLGs, significant differences in the frequencies and distribution of practical and social perspectives emerged, with HLGs performing higher than LLGs. This aligns with previous research demonstrating that application courses emphasize problem solving and skill development through practice-based learning (Sankaranarayanan et al., 2022). Collaborative interaction, involving task factors and social attributes (Soller & Lesgold, 2007), aims to solve problems and develop skills (Swiecki et al., 2020). Through communication, members exchange information based on tasks, coordinate behaviors and ideas, and construct knowledge (Yücel & Usluel, 2016). Externalizing ideas with words can further stimulate partners to rethink their ideas, identify reasoning problems (Kolloffel et al., 2011), and promote thinking development. Moreover, both the HLGs and LLGs focused on the identification of concepts, which is a fundamental step in problem solving. However, there were significant differences in the frequencies and distribution of cognitive perspectives, with LLGs having a higher proportion than HLGs. Similar results can be found in Zhang et al. (2022), showing that LLGs tend to spend more time on basic issues during collaboration, hindering their ability to transition to higher-level activities, and leading to low academic performance.

Second, the ENA network was utilized to reveal CT process patterns of all groups and differences between HLGs and LLGs. The mean network diagram illustrated CT process patterns in all groups, revealing that concepts related to conditionals, operators, and data emerged frequently from communication and the expression of perspectives, which are crucial to solving programming problems. Moreover, pairs frequently used flow design scaffolding to sort out problem-solving models formed by computational concepts and support clear communication in a schematic form. This finding is consistent with previous studies (Ibrahim et al., 2018; Zhang et al., 2021) indicating that visual flow design could help learners decompose tasks and sharpen their programming problem-solving skills. However, the process was closely related to connecting, indicating that pair communication mainly focused on

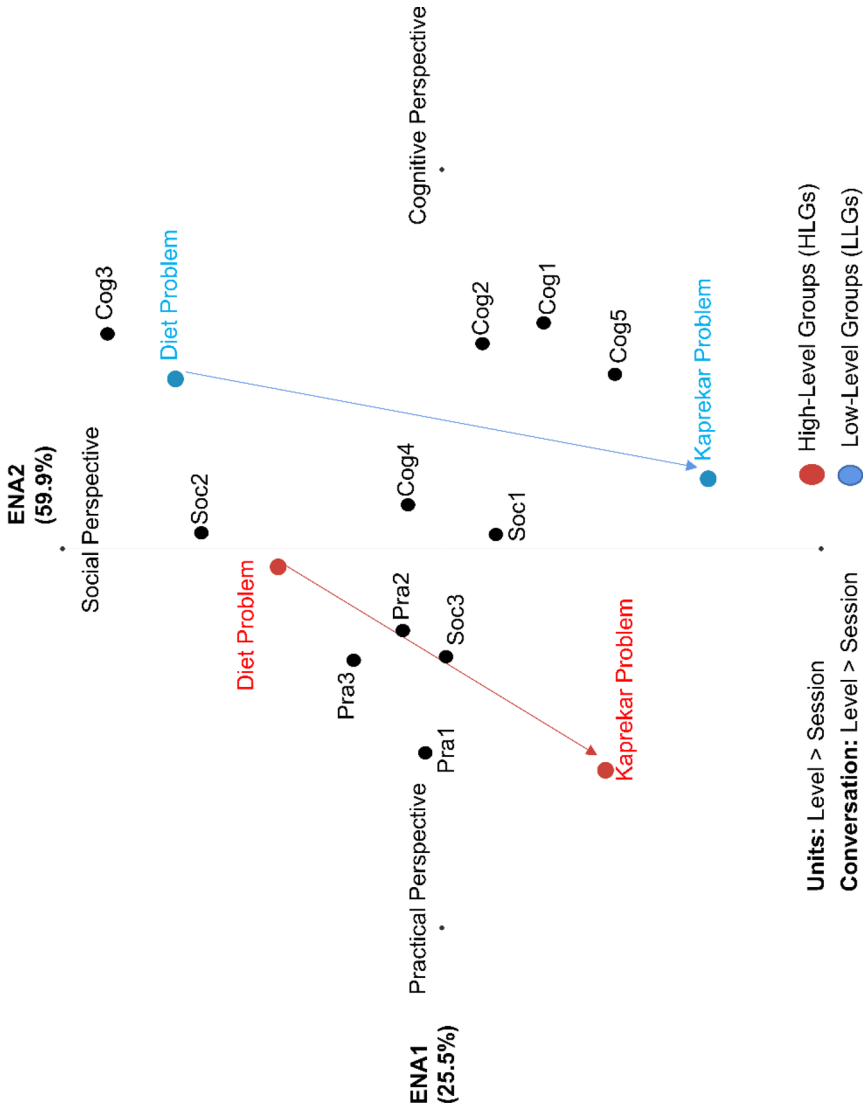


Fig. 6 ENA network centroids of the HLGs and LLGs from Diet Problem to Kaprekar Problem

flow design. Despite pre-class skill training, students still experienced difficulties in selecting and generating flowcharts for constructing problem-solving models. Progressive learning strategies are required to master this skill, as noted in previous research (Zhang et al., 2021). The subtraction network of all groups revealed differences in CT patterns between HLGs and LLGs. Our findings suggested a positive correlation between the comprehensive CT process patterns and high-quality programming problem-solving outcomes. HLGs had stronger connections among cognitive, practical, and social perspectives, while LLGs only had stronger connections between cognitive and social perspectives. Similar to the results of the previous study (Wu et al., 2019), HLGs exhibited a more systematic pattern of concept identification, model construction, and iterative testing in pair communication, while LLGs focused mainly on cognitive perspectives with little co-occurrence of multiple sub-dimensions of CT. The reason might be that HLGs made work plans at the beginning of solving problems and viewed programming stages as a whole. In contrast, LLGs regarded programming stages as separate ones and lacked practical ability in idea integration, code review, and iterative testing. Moreover, a positive collaborative learning atmosphere is conducive to idea formation and problem solving (Martin & Collie, 2019). The original discourse data that HLGs tended to establish a positive collaborative learning atmosphere through equal dialogue, while LLGs struggled to form a consensus regarding the key to solving the problem. As not all students have the same learning abilities in group learning, creating an incentivizing learning atmosphere is essential (Supena et al., 2021).

Third, the collaborative problem-solving process is characterized by periodicity (Swiecki et al., 2020). By analyzing the differences in CT between HLGs and LLGs during different programming tasks using ENA, it may be possible to identify the reasons for the differences in collaborative performance (Zhang et al., 2022). Compared to the Diet Problem, the differences in CT process patterns in practical perspectives were more prominent between HLGs and LLGs, with slight differences in the connection of cognitive perspectives codes in the Kaprekar Problem. One possible explanation is that case-based reflection promoted concept learning better (Sankaranarayanan et al., 2022). After the first pairing, students were asked to reflect on their learning activity. In the second pairing, HLGs directed more attention towards testing, debugging, and iteration after realizing the importance of practical testing in problem solving, while LLGs strengthened the interaction to reach an agreement rather than mutual questioning. Therefore, teachers should guide learners to reflect on the learning process after corresponding collaborative learning tasks, to check the deficiencies and improve learning efficiency. Moreover, although both HLGs and LLGs exhibited differences in the two problem-solving activities, the different characteristics indicate their transformed CT process patterns. This is in line with the conclusions of the existing literature, that is, HLGs adopted different CT patterns in pair programming for the two problem-solving activities, with complicated and developmental CT processes (Zhang et al., 2023), featuring multiple dimensions of CT more closely related and interactive. LLGs maintained relatively fixed CT patterns with little co-occurrence of multiple dimensions of CT (Xu et al., 2020). The reason might be attributed to the different meta-cognitive abilities between HLGs and LLGs. HLGs concentrated on self-regulation through a more

integral perspective, making them better at monitoring the problem-solving process. Conversely, LLGs focused on a single aspect of reflection, lacking overall self-regulation ability. Echoing this finding, reflection is formed from a deep learning approach (Ghanizadeh, 2017). Students with higher academic performance ponder thoroughly, reflecting on the purpose and nature through problem-solving, rather than simplistically handling tasks. Incorporating reflective learning strategies into learning activities can help students enhance academic performance and encourage deep learning (Tsingos et al., 2015).

7 Conclusion

This research provides valuable insights into undergraduates' CT processes in pair programming problem solving. The CT process patterns of HLGs and LLGs were analyzed in detail, revealing the relationship between CT and academic performance, and explaining differences at different problem-solving stages. This work is essential in providing a theoretical explanation of CT processes that may lead to good performance in programming. Methodologically, this study has incorporated an integrated analysis approach from both quantitative and qualitative perspectives into group discourse analysis. This approach has enriched the understanding of CT processes and their interaction relationship from three dimensions, providing a comprehensive analysis of students' CT development in pair programming problem solving. In addition, the research results have implications for educational practice. The findings of CT process patterns provide a reference for educators in programming education to design interventions aimed at enhancing students' CT.

Although this study provides insight into problem-solving and CT processes in pair programming, there are some limitations. Firstly, the study was conducted in the context of a university programming course, where the specific characteristics and difficulties of the programming tasks and personalized guidance could have influenced the CT mode. Therefore, the results of this study may only apply to similar research backgrounds. Secondly, the small sample size of only 30 students (15 pairs) may have led to the omission of some key CT patterns. In addition, although students were grouped based on prior knowledge, other personality characteristics such as gender, attitude, and cognitive style would inevitably affect CT process patterns. Finally, the analysis of students' CT based on discourse data is insufficient. The use of behavior data from flow design and coding as an additional data source could provide a more comprehensive evaluation of CT. However, the collaborative editing platform employed in this study limited our access to such behavior data. Future research could enhance credibility and expand the research results by strengthening the investigation of specific subject tasks and student group characteristics. It is also essential to conduct research in multiple programming courses and expand the sample size to increase the validity of the findings. Moreover, collecting other data sources, including discourse data, behavior data, and eye movement data, would enable a comprehensive analysis of the nature of students' CT processes in programming learning.

Acknowledgements This research was supported by National Natural Science Foundation of China (NSFC) for the Project “A Study on the Perception and Attribution Analysis of Learners’ Higher-Order Thinking Activities” (No.: 62177023), and the project of the Faculty of Artificial Intelligence In Education of Central China Normal University (No.: 2022XY014). We are grateful for the support from NSFC and Central China Normal University. Any opinions expressed herein are those of the authors and do not necessarily represent the funds’ views. We thank the teacher and students for their participation.

Author contribution Ruijie Zhou contributed the central idea, performed the research, analyzed most of the data, and wrote the initial draft of the paper. Yangyang Li contributed to refining the ideas and carrying out additional analyses. Xiuling He developed the idea for the study, formed overall research objectives, and provided an implementable environment for experiments. Chunlian Jiang contributed to refining the ideas and finalizing this paper. Jing Fang contributed to refining the ideas. Yue Li contributed to refining the ideas.

Data availability The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Ethics statement All procedures performed in studies involving human participants were in accordance with ethical standards. The study was approved by the Social Sciences and Humanities Research Ethics Committee of Central China Normal University.

Conflict of interest The authors declare no conflicts of interest.

References

- Asunda, P. A. (2018). Infusing computer science in engineering and technology education: An integrated STEM perspective. *The Journal of Technology Studies*, 44(1), 2–13. Retrieved March 9, 2024, from <https://www.jstor.org/stable/26730725>
- 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>
- Bai, H., Wang, X., & Zhao, L. (2021). Effects of the problem-oriented learning model on middle school students’ computational thinking skills in a python course. *Frontiers in Psychology*, 12. <https://doi.org/10.3389/fpsyg.2021.771221>
- Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, 72, 145–157. <https://doi.org/10.1016/j.compedu.2013.10.020>
- Borreguero Zuloaga, M., & De Marco, A. (2021). The role of immersion and non-immersion contexts in L2 acquisition: A study based on the analysis of interactional discourse markers. *Corpus Pragmatics*, 5(1), 121–151. <https://doi.org/10.1007/s41701-020-00093-x>
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, 1, 25.
- Budny, D., Lund, L., Vipperman, J., & Patzer, J. L. I. I. I. (2002). Four steps to teaching C programming. *32nd Annual Frontiers in Education*, 2, F1G-18-F1G-22. <https://doi.org/10.1109/FIE.2002.1158140>
- Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation’s way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, 87(4), 834–860. <https://doi.org/10.3102/0034654317710096>

- Chao, P.-Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education*, 95, 202–215. <https://doi.org/10.1016/j.compedu.2016.01.010>
- Charntaweekhun, K., & Wangsiripitak, S. (2006). Visual programming using flowchart. *2006 International Symposium on Communications and Information Technologies* (pp. 1062–1065). <https://doi.org/10.1109/ISCIT.2006.339940>
- Cheah, C. S. (2020). Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology*, 12(2), ep272. <https://doi.org/10.30935/cedtech/8247>
- Chiu, C.-F. (2020). Facilitating K-12 teachers in ceating apps by visual programming and project-based learning. *International Journal of Emerging Technologies in Learning (iJET)*, 15(01), 103–118. <https://doi.org/10.3991/ijet.v15i01.11013>
- Collins, A., Brown, J. S., & Holum, A. (1991). Cognitive apprenticeship: Making thinking visible. *American Educator*, 15(3), 6–11.
- Csanadi, A., Eagan, B., Kollar, I., Shaffer, D. W., & Fischer, F. (2018). When coding-and-counting is not enough: Using epistemic network analysis (ENA) to analyze verbal data in CSCL research. *International Journal of Computer-Supported Collaborative Learning*, 13(4), 419–438. <https://doi.org/10.1007/s11412-018-9292-z>
- CSTA, & ISTE. (2011). *Operational definition of computational thinking for K–12 education*. Retrieved February 14, 2023, from <http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf>
- Dagienė, V., & Futschek, G. (2008). Bebras international contest on informatics and computer literacy: Criteria for good tasks. In R. T. Mittermeir & M. M. Sysło (Eds.), *Informatics Education—Supporting Computational Thinking* (Vol. 5090, pp. 19–30). Springer. https://doi.org/10.1007/978-3-540-69924-8_2
- Dagienė, V., & Stupurienė, G. (2016). Bebras—A sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education*, 15(1), 25–44. <https://doi.org/10.15388/infedu.2016.02>
- Dale, N. B., & Weems, C. (2014). *Programming and problem solving with C++ (sixth edition)*. Jones & Bartlett Publishers.
- Davies, A., Fidler, D., & Gorbis, M. (2011). *Future Work Skills 2020*. Institute for the Future for University of Phoenix Research Institute.
- 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(1), 41–70. <https://doi.org/10.1177/0735633120949787>
- Echeverría, L., Cobos, R., & Morales, M. (2019). Improving the students computational thinking skills with collaborative learning techniques. *IEEE Revista Iberoamericana De Tecnologías Del Aprendizaje*, 14(4), 196–206. <https://doi.org/10.1109/RITA.2019.2952299>
- Fang, J.-W., Shao, D., Hwang, G.-J., & Chang, S.-C. (2022). From critique to computational thinking: A peer-assessment-supported problem identification, flow definition, coding, and testing approach for computer programming instruction. *Journal of Educational Computing Research*, 60(5), 1301–1324. <https://doi.org/10.1177/07356331211060470>
- García-Peñalvo, F. J., & Mendes, A. J. (2018). Exploring the computational thinking effects in pre-university education. *Computers in Human Behavior*, 80, 407–411. <https://doi.org/10.1016/j.chb.2017.12.005>
- Gašević, D., Joksimović, S., Eagan, B. R., & Shaffer, D. W. (2019). SENS: Network analytics to combine social and cognitive perspectives of collaborative learning. *Computers in Human Behavior*, 92, 562–577. <https://doi.org/10.1016/j.chb.2018.07.003>
- Ghanizadeh, A. (2017). The interplay between reflective thinking, critical thinking, self-monitoring, and academic achievement in higher education. *Higher Education*, 74(1), 101–114. <https://doi.org/10.1007/s10734-016-0031-y>
- Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2), 199–237. <https://doi.org/10.1080/08993408.2015.1033142>
- Hannay, J. E., Dybå, T., Arisholm, E., & Sjøberg, D. I. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7), 1110–1122. <https://doi.org/10.1016/j.infsof.2009.02.001>

- Harvey, B. (1997). *Computer science logo style: Symbolic computing* (Vol. 1). MIT Press. Retrieved March 9, 2024, from https://sc.panda321.com/extdomains/books.google.com/books/about/Computer_Science_Logo_Style_Symbolic_com.html?hl=zh-CN&id=BmuqURW0G5UC
- He, X., Fang, J., Cheng, H. N. H., Men, Q., & Li, Y. (2023). Investigating online learners' knowledge structure patterns by concept maps: A clustering analysis approach. *Education and Information Technologies*. <https://doi.org/10.1007/s10639-023-11633-8>
- Hopcan, S., Polat, E., & Albayrak, E. (2022). Collaborative behavior patterns of students in programming instruction. *Journal of Educational Computing Research*, 60(4), 1035–1062. <https://doi.org/10.1177/07356331211062260>
- Hsu, T.-C., Chang, S.-C., & Hung, Y.-T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126, 296–310. <https://doi.org/10.1016/j.compedu.2018.07.004>
- Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3), 259–290. <https://doi.org/10.1006/jvlc.2002.0237>
- Ibrahim, N., Saifuzzin, N. F. S., Seman, A. A., Wahab, N. A., & Osman, A. (2018). Flowchart discovery game for basic programming course (FlowGame). *Journal of Applied and Fundamental Sciences*, 10(1S), 1109–1122. <https://doi.org/10.4314/jfas.v10i1s.81>
- Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, 82, 263–279. <https://doi.org/10.1016/j.compedu.2014.11.022>
- ISTE. (2015). *CT Leadership toolkit*. Retrieved March 9, 2024, from <https://www.iste.org/standards/iste-standards-for-computational-thinking>
- Kafai, Y. B. (2016). From computational thinking to computational participation in K–12 education. *Communications of the ACM*, 59(8), 26–27. <https://doi.org/10.1145/2955114>
- Kolloffel, B., Eysink, T. H. S., & de Jong, T. (2011). Comparing the effects of representational tools in collaborative and individual inquiry learning. *International Journal of Computer-Supported Collaborative Learning*, 6(2), 223–251. <https://doi.org/10.1007/s11412-011-9110-3>
- 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>
- Kreijns, K., Kirschner, P. A., & Jochems, W. (2003). Identifying the pitfalls for social interaction in computer-supported collaborative learning environments: A review of the research. *Computers in Human Behavior*, 19(3), 335–353. [https://doi.org/10.1016/S0747-5632\(02\)00057-2](https://doi.org/10.1016/S0747-5632(02)00057-2)
- Lai, X., & Wong, G. K. (2022). Collaborative versus individual problem solving in computational thinking through programming: A meta-analysis. *British Journal of Educational Technology*, 53(1), 150–170. <https://doi.org/10.1111/bjet.13157>
- Lee, Y.-J. (2011). Empowering teachers to create educational software: A constructivist approach utilizing Etoys, pair programming and cognitive apprenticeship. *Computers & Education*, 56(2), 527–538. <https://doi.org/10.1016/j.compedu.2010.09.018>
- Li, W., Liu, C.-Y., & Tseng, J. C. R. (2023). Effects of the interaction between metacognition teaching and students' learning achievement on students' computational thinking, critical thinking, and metacognition in collaborative programming learning. *Education and Information Technologies*, 28(10), 12919–12943. <https://doi.org/10.1007/s10639-023-11671-2>
- Lodi, M. (2020). Informatical Thinking. *OLYMPIADS IN INFORMATICS*, 113–132. <https://doi.org/10.15388/foi.2020.09>
- López-Pellisa, T., Rotger, N., & Rodríguez-Gallego, F. (2021). Collaborative writing at work: Peer feedback in a blended learning environment. *Education and Information Technologies*, 26(1), 1293–1310. <https://doi.org/10.1007/s10639-020-10312-2>
- Lui, D., Walker, J. T., Hanna, S., Kafai, Y. B., Fields, D., & Jayathirtha, G. (2020). Communicating computational concepts and practices within high school students' portfolios of making electronic textiles. *Interactive Learning Environments*, 28(3), 284–301. <https://doi.org/10.1080/10494820.2019.1612446>
- 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>
- Marquart, C. L., Hinojosa, C., Swiecki, Z., Eagan, B., & Shaffer, D. W. (2018). *Epistemic network analysis [Software] Version 1.6. 0*. [Computer software]. epistemicnetwork.org.

- Martin, A. J., & Collie, R. J. (2019). Teacher–student relationships and students’ engagement in high school: Does the number of negative and positive relationships with teachers matter? *Journal of Educational Psychology*, *111*(5), 861–876. <https://doi.org/10.1037/edu0000317>
- McCormick, D., & Ross, S. M. (1990). Effects of computer access and flowcharting on students’ attitudes and performance in learning computer programming. *Journal of Educational Computing Research*, *6*(2), 203–213. <https://doi.org/10.2190/E3DQ-YN2T-7U0V-JQ5N>
- Meier, A., Spada, H., & Rummel, N. (2007). A rating scheme for assessing the quality of computer-supported collaboration processes. *International Journal of Computer-Supported Collaborative Learning*, *2*(1), 63–86. <https://doi.org/10.1007/s11412-006-9005-x>
- Mohaghegh, M., & McCauley, M. (2016). Computational thinking: The skill set of the 21st century. *International Journal of Computer Science and Information Technologies*, *7*(3), 1524–1530. Retrieved March 9, 2024, from <http://www.ijcsit.com/docs/Volume%207/vol7issue3/ijcsit20160703104.pdf>
- Nassi, I., & Shneiderman, B. (1973). Flowchart techniques for structured programming. *ACM SIGPLAN Notices*, *8*(8), 12–26. <https://doi.org/10.1145/953349.953350>
- National Research Council. (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. National Academies Press.
- Olsen, J. K., Sharma, K., Rummel, N., & Aleven, V. (2020). Temporal analysis of multimodal data to predict collaborative learning outcomes. *British Journal of Educational Technology*, *51*(5), 1527–1547. <https://doi.org/10.1111/bjet.12982>
- Ouyang, F., Tang, Z., Cheng, M., & Chen, Z. (2023). Using an integrated discourse analysis approach to analyze a group’s collaborative argumentation. *Thinking Skills and Creativity*, *47*, 101227. <https://doi.org/10.1016/j.tsc.2022.101227>
- Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
- Papert, S. (1993). *The children’s machine: Rethinking school in the age of the computer*. Basic Books.
- 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>
- Plonka, L., Sharp, H., van der Linden, J., & Dittrich, Y. (2015). Knowledge transfer in pair programming: An in-depth analysis. *International Journal of Human-Computer Studies*, *73*, 66–78. <https://doi.org/10.1016/j.ijhcs.2014.09.001>
- Rahman, K., & Nordin, M. J. (2007). *A review on the static analysis approach in the automated programming assessment systems*. National Conference on Software Engineering and Computer Systems.
- Rolim, V., Ferreira, R., Lins, R. D., & Gásevíc, D. (2019). A network-based analytic approach to uncovering the relationship between social and cognitive presences in communities of inquiry. *The Internet and Higher Education*, *42*, 53–65. <https://doi.org/10.1016/j.iheduc.2019.05.001>
- 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>
- Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools. *Computers & Education*, *97*, 129–141. <https://doi.org/10.1016/j.compedu.2016.03.003>
- Salleh, S. M., Shukur, Z., & Judi, H. M. (2018). Scaffolding model for efficient programming learning based on cognitive load theory. *International Journal of Pure and Applied Mathematics*, *118*(7), 77–83. Retrieved March 9, 2024, from <https://acadpubl.eu/jsi/2018-118-7-9/articles/7/10.pdf>
- Sankaranarayanan, S., Kandimalla, S. R., Bogart, C., Murray, R. C., Hilton, M., Sakr, M. F., & Rose, C. P. (2022). Collaborative programming for work-relevant learning: Comparing programming practice with example-based reflection for student learning and transfer task performance. *IEEE Transactions on Learning Technologies*, *15*(5), 594–604. <https://doi.org/10.1109/TLT.2022.3169121>
- Satratzemi, M., Xinogalos, S., Tsompanoudi, D., & Karamitopoulos, L. (2021). A two-year evaluation of distributed pair programming assignments by undergraduate students. In T. Tsiatsos, S. Demetriadis, A. Mikropoulos, & V. Dagdilelis (Eds.), *Research on E-Learning and ICT in Education* (pp. 35–57). Springer International Publishing. https://doi.org/10.1007/978-3-030-64363-8_3
- Selby, C., & Woollard, J. (2013). Computational thinking: The developing definition. *18th Annual Conference on Innovation and Technology in Computer Science Education*. Retrieved March 9, 2024, from <https://eprints.soton.ac.uk/356481/>
- Shaffer, D. W., Collier, W., & Ruis, A. R. (2016). A tutorial on epistemic network analysis: Analyzing the structure of connections in cognitive, social, and interaction data. *Journal of Learning Analytics*, *3*(3), 3. <https://doi.org/10.18608/jla.2016.33.3>







- Siddiq, F., & Scherer, R. (2017). Revealing the processes of students' interaction with a novel collaborative problem solving task: An in-depth analysis of think-aloud protocols. *Computers in Human Behavior*, 76, 509–525. <https://doi.org/10.1016/j.chb.2017.08.007>
- Smith, G. F., & Browne, G. J. (1993). Conceptual foundations of design problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(5), 1209–1219. <https://doi.org/10.1109/21.260655>
- Soller, A., & Lesgold, A. (2007). Modeling the process of collaborative learning. In H. U. Hoppe, H. Ogata, & A. Soller (Eds.), *The Role of Technology in CSCL* (pp. 63–86). Springer US. https://doi.org/10.1007/978-0-387-71136-2_5
- Soto, M. S., & Figueroa, I. (2018). Heuristic evaluation of code::blocks as a tool for first year programming courses. *37th International Conference of the Chilean Computer Science Society (SCCC)*, 1–8. <https://doi.org/10.1109/SCCC.2018.8705158>
- Su, Q., Zhang, W., Wang, H., & Li, H. (2022). Research on project-based learning of information technology curriculum for cultivating senior high school students' computational thinking. *China Academic Journal Electronic Publishing House*, 43(8), 109–115+122. <https://doi.org/10.13811/j.cnki.eer.2022.08.014>
- Sun, L., & Zhou, L. (2023). Does text-based programming improve K-12 students' CT skills? Evidence from a meta-analysis and synthesis of qualitative data in educational contexts. *Thinking Skills and Creativity*, 49, 101340. <https://doi.org/10.1016/j.tsc.2023.101340>
- Sun, M., Wang, M., Wegerif, R., & Peng, J. (2022). How do students generate ideas together in scientific creativity tasks through computer-based mind mapping? *Computers & Education*, 176, 104359. <https://doi.org/10.1016/j.compedu.2021.104359>
- Supena, I., Darmuki, A., & Hariyadi, A. (2021). The influence of 4C (constructive, critical, creativity, collaborative) learning model on students' learning outcomes. *International Journal of Instruction*, 14(3), 873–892. <https://doi.org/10.29333/iji.2021.14351a>
- Swiecki, Z., Ruis, A. R., Farrell, C., & Shaffer, D. W. (2020). Assessing individual contributions to collaborative problem solving: A network analysis approach. *Computers in Human Behavior*, 104, 105876. <https://doi.org/10.1016/j.chb.2019.01.009>
- 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>
- Threekunprapa, A., & Yasri, P. (2020). Unplugged coding using flowblocks for promoting computational thinking and programming among secondary school students. *International Journal of Instruction*, 13(3), 207–222. <https://doi.org/10.29333/iji.2020.13314a>
- Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Computers & Education*, 162, 104083. <https://doi.org/10.1016/j.compedu.2020.104083>
- Tsan, J., Vandenberg, J., Zakaria, Z., Wiggins, J. B., Webber, A. R., Bradbury, A., Lynch, C., Wiebe, E., & Boyer, K. E. (2020). A comparison of two pair programming configurations for upper elementary students. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 346–352. <https://doi.org/10.1145/3328778.3366941>
- Tsingos, C., Bosnic-Anticevich, S., & Smith, L. (2015). Learning styles and approaches: Can reflective strategies encourage deep learning? *Currents in Pharmacy Teaching and Learning*, 7(4), 492–504. <https://doi.org/10.1016/j.cptl.2015.04.006>
- Vygotsky, L. S., & Cole, M. (1978). *Mind in society: Development of higher psychological processes*. Harvard University Press. Retrieved March 9, 2024, from https://xs.zidianzhan.net/books/about/Mind_in_Society.html?hl=zh-CN&id=RxjjUefze_oC
- Wang, Y., Li, H., Feng, Y., Jiang, Y., & Liu, Y. (2012). Assessment of programming language learning based on peer code review model: Implementation and experience report. *Computers & Education*, 59(2), 412–422. <https://doi.org/10.1016/j.compedu.2012.01.007>
- Wei, X., Lin, L., Meng, N., Tan, W., & Kong, S. C. (2021). The effectiveness of partial pair programming on elementary school students' computational thinking skills and self-efficacy. *Computers & Education*, 160, 104023. <https://doi.org/10.1016/j.compedu.2020.104023>
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127–147. <https://doi.org/10.1007/s10956-015-9581-5>
- Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 215–220. <https://doi.org/10.1145/2157136.2157200>

- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. <https://doi.org/10.1098/rsta.2008.0118>
- Wing, J. M. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine*, 6, 20–23. Retrieved March 9, 2024, from http://link.cs.cmu.edu/files/11-399_The_Link_Newsletter-3.pdf
- Wu, B., Hu, Y., Ruis, A. R., & Wang, M. (2019). Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning*, 35(3), 421–434. <https://doi.org/10.1111/jcal.12348>
- Xiao, M., & Yu, X. (2017). A model of cultivating computational thinking based on visual programming. *International Conference of Educational Innovation through Technology (EITT)*, 2017, 75–80. <https://doi.org/10.1109/EITT.2017.26>
- Xu, H., Huang, D., Leng, J., & Xu, X. (2020). Investigating the developmental trajectory of critical thinking in online discourse among college students: An epistemic network analysis. *The Interdisciplinarity of the Learning Sciences*, 1, 509–512. Retrieved March 9, 2024, from <https://repository.isls.org/handle/1/6681>.
- Yağcı, M. (2019). A valid and reliable tool for examining computational thinking skills. *Education and Information Technologies*, 24(1), 929–951. <https://doi.org/10.1007/s10639-018-9801-8>
- Yücel, Ü. A., & Usluel, Y. K. (2016). Knowledge building and the quantity, content and quality of the interaction and participation of students in an online collaborative learning environment. *Computers & Education*, 97, 31–48. <https://doi.org/10.1016/j.compedu.2016.02.015>
- Zhang, J.-H., Meng, B., Zou, L.-C., Zhu, Y., & Hwang, G.-J. (2021). Progressive flowchart development scaffolding to improve university students' computational thinking and programming self-efficacy. *Interactive Learning Environments*, 31(6), 3792–3809. <https://doi.org/10.1080/10494820.2021.1943687>
- Zhang, L., & 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>
- Zhang, S., Gao, Q., Sun, M., Cai, Z., Li, H., Tang, Y., & Liu, Q. (2022). Understanding student teachers' collaborative problem solving: Insights from an epistemic network analysis (ENA). *Computers & Education*, 183, 104485. <https://doi.org/10.1016/j.compedu.2022.104485>
- Zhang, S., Li, H., Wen, Y., Zhang, Y., Guo, T., & He, X. (2023). Exploration of a group assessment model to foster student teachers' critical thinking. *Thinking Skills and Creativity*, 47, 101239. <https://doi.org/10.1016/j.tsc.2023.101239>
- Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, 53(4), 562–590. <https://doi.org/10.1177/0735633115608444>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Ruijie Zhou^{1,2}  · Yangyang Li^{1,2}  · Xiuling He^{1,2}  · Chunlian Jiang³  ·
Jing Fang^{1,2}  · Yue Li^{1,2} 

✉ Xiuling He
xlhe@ccnu.edu.cn

✉ Chunlian Jiang
cljiang@um.edu.mo

Ruijie Zhou
zhourijie@mails.ccnu.edu.cn

Yangyang Li
necellyy@ccnu.edu.cn

Jing Fang
fangjing@ccnu.edu.cn

Yue Li
li_yue@mails.ccnu.edu.cn

¹ National Engineering Research Center of Educational Big Data, Central China Normal University, Wuhan 430079, China

² National Engineering Research Center for E-Learning, Central China Normal University, Wuhan 430079, China

³ Faculty of Education, University of Macau, Macao, China