

# Constructive Interaction on Collaborative Programming: Case Study for Grade 6 Students Group

Sayaka Tohyama<sup>1</sup>(✉), Yoshiaki Matsuzawa<sup>2</sup>, Shohei Yokoyama<sup>1</sup>,  
Tepei Koguchi<sup>1</sup>, and Yugo Takeuchi<sup>1</sup>

<sup>1</sup> Shizuoka University, 3-5-1 Johoku, Naka-ku,  
Hamamatsu-shi, Shizuoka, Japan  
toyama.sayaka@shizuoka.ac.jp

<sup>2</sup> Aoyama Gakuin University, 5-10-1 Fuchinobe, Chuo-ku,  
Sagamihara-shi, Kanagawa, Japan

**Abstract.** Recent learning sciences have revealed some of the mechanisms of how people learn through interactions in collaborative educational settings. In this research, we tried to capture the nature of constructive interaction by in-depth qualitative analysis of the discourse in a programming learning environment. The analyzed group was comprised of three female students, all in the sixth grade, who engaged in making an animation using Scratch. However, they had trouble with their object modelling during the task. Through their problem-solving procedure, the students attempted externalizations of their solution ideas, and these interactions promoted their understanding of the problem through the iterative process of each individual. Working collaboratively, the three students used various procedures to solve their shared object-modelling problems.

**Keywords:** Collaborative learning · Programming · Computational thinking  
K–12 · Constructive interaction

## 1 Introduction

Computing education with “Computational Thinking” [1] is not only growing as a research field but is also being addressed as a political and common issue all over the world [2]. To develop computing competencies during the early stage of citizens’ lives, many countries start compulsory programming education in grades K–12. They have been discussing what should be taught [3, 4] as Computational Thinking at this level for all citizens living in a 21st century knowledge-based society.

There is a consensus between researchers that the movement of computing education is a revival of the 1980s programming education conducted using Logo [5]. The origin of programming education with Logo by Papert, who coined the term “computational thinking” [6], was not primarily intended to develop programming skills but to open a new method of learning mathematics through programming. By preparing situated environments, children could construct their ideas by directly operating them in a situated world [7]. Kay expanded the application of the idea from mathematics to

various other disciplines [8]—he called it “dynamic media”—in which programming is considered basic literacy in a computing society where citizens are using computers as meta media [9]. This dream was inherited to the latest programming environments for kids, Scratch, which is the direct successor of Squeak and Logo.

Many works exploring the effects of programming education with Logo were done by cognitive science researchers in the ‘80s and ‘90s. At that time, synonyms for computational thinking included powerful thinking or higher order problem-solving skills, with the key issue being whether programming experiences developed such skills. While we assume “programming” to be a cyclic process comprised of modelling, planning, coding, and the evaluation (debugging) process, it is considered a complex, ill-structured task. As expert programmers can integrate the knowledge of generic algorithm-construction and that of programming language [10], they are assumed to possess high cognitive skills. Actually, they showed a higher level of generic problem-solving skills, such as the decomposition or inferring of problems [11]. Accordingly, programming was expected to develop one’s cognitive skills.

Despite the limited numbers, there are a few works that show evidence of developing transferrable competencies through programming. Lawler succeeded in illustrating the development process of a 6-year-old child’s cognitive strategy for calculation through Logo programming experiences [12], albeit within the limitations of single-subject research. Clements and Gullo conducted an experimental study between a CAI and a programming group. The results supported that the programming experiences developed students’ creative-thinking, reflectivity, and cognitive skills [13].

However, many reports have appeared to show some results that contradict the expectation of programming education. In particular, Pea and Kurland [14, 15] criticized developing transferrable cognitive skills by programming, based on their results. Webb et al. tackled analyzing the problem-solving strategies in a group programming process [16, 17]. Pair of children (aged 11–14) learned programming using BASIC. Although the results were not negative, they did not succeed in finding clear evidence of advancing children’s planning skills.

This issue in programming education has been controversial since the 1980s, as discussed above; however, there has been remarkably little research conducted after 2000. Consequently, we remain at the 1980s level of discussion in the cognitive study of programming education, despite the improvement of the programming environment [18] and studies from cognitive science and the “learning sciences”.

From these points of view, firstly, we suggest the use of Scratch. Scratch, is a visualized programming environment that is broadly used by practitioners and researchers in programming workshops. Scratch may better enhance students’ focus on higher level problem-solving than the text coding. Secondly, we focused on collaborative learning. For this reason, the handbook of collaborative learning was published [19], and PISA started an assessment of collaborative problem solving in 2015 [20]. From the trend of collaborative learning, Constructive Interaction (CI) [21] is a key reason for our choice of a collaborative setting in this study. Not only is the CI analysis method capable of revealing an iterative, progressive problem-solving process, but participants deepened their own understandings when CI occurred in their discussions. Miyake pointed out that CI is well produced if the participants externalize their own

understandings and the depths of their understandings differed. In this paper, the key suggestion is that there are levels of understanding, and the difference in these levels helps the participants deepen their understanding.

We intend to contribute to the pursuit of a modern version of Webb's work [16, 17] using the viewpoints of Miyake's work [21]. Webb's paper discussed concerns about familiarity with and the students' typing skills as reasons for why they could not observe quality interactions. Higher-level problem-solving interactions can be expected if we add CI points of view.

Toward the goal of clarifying the mechanism for problem solving in programming, we attempted a qualitative analysis of collaborative programming that enhances the externalization of the participants' different levels of understanding using Scratch.

## 2 Method

### 2.1 Programming Workshop

We held a one-day collaborative programming workshop for elementary school students. The participants were 16 sixth-grade students who responded to the request for participation issued at Hamamatsu Elementary School, which is attached to the Faculty of Education, Shizuoka University (8 boys, 8 girls; 4 of the students had no programming experience). We conducted the workshop at the school, on August 9, 2016, using iPads. We installed the app "Pyonkee" on the iPads and distributed one iPad to each student.

Based on constructionism, we designed the workshop so as to encourage students to express their creative ideas in their own ways. We avoided a training design that focuses solely on fostering accurate and impressive coding skills.

To encourage collaborative programming, we asked the students to form teams and to produce a single work from each team. We allowed the students to form their teams themselves. The students formed five three-person, unisex teams.

Table 1 shows the flow of the workshop. We first conducted a preliminary questionnaire survey (four-point scale) to ascertain the students' programming experience, their impressions of programming, and their attitudes toward collaborative learning. Next, each team produced a storyboard design sheet for the program to externalize their work designs, and the teams presented their diagrams to each other. The team members then wrote a program to implement what they envisaged on their team's storyboard design sheet. Finally, the teams presented their programs to each other and completed a feedback questionnaire. The feedback questionnaire was identical in content to the preliminary questionnaire (see Table 1).

We recorded the students' activities in the workshop using a video camera and audio recorder. During the programming process, we projected the screen content of the students' iPads onto a projector using Apple TV and recorded the projected images using a video camera.

**Table 1.** Timetable of the workshop

Time	Contents
10:00–10:50	Preliminary questionnaire, guidance
11:00–12:00	Drawing storyboard design sheet
13:00–15:15	Programming
15:15–15:45	Presentation
15:45–15:55	Feedback questionnaire

## 2.2 Programming Environment: Pyonkee

Figure 1 shows Pyonkee’s operation screen. It works in almost the same manner as Scratch. Scratch is an environment for object-oriented programming in which users can issue motion instructions to objects (an example of an object is the character at the top right of Fig. 1). Pyonkee allows users to create a range of works, including moving picture shows and shooter games. In Pyonkee, the protagonist and his/her environment (the ground, the sky, etc.) are treated as “objects.” Multiple objects can move correspondingly. If we use two objects, they will impact each other using “message-passing.” The programs are written by selecting pre-prepared blocks of code and arranging them into as many combinations as desired.

Pyonkee also allows users to create objects as backgrounds (the stage). In Pyonkee, the entire stage is treated as a single object.

**Fig. 1.** Pyonkee

### 2.3 Analytical Sample

Of the children who participated in the workshop, we focused on one of the five groups. There were three girls in this group (X, Y, and Z). In the questionnaires, these three girls exhibited a different trend than those of the other teams. Specifically, the girls' average score for the question, "Does programming feature in your daily life?" improved by 1.5 points in the feedback survey compared to the preliminary survey (change in the overall mean score for this question was 0.2 points).

These girls had participated in a previous programming workshop that we conducted. X conducted most of the programming operations on the iPad without any suggestions from the supervisors or the other girls (Y and Z).

### 2.4 Analytical Method

To analyze the girls' activity qualitatively, we transcribed their discourse and operations on the iPad. We split the discourse into 236 utterances made by mouth, and we wrote each utterance into lines of the transcript.

The girls' programming progress culminated in the completion of their program according to their storyboard design sheet (see Fig. 2). They told the story of a climbing experience during school camp. The storyboard design sheet comprised three situations: the protagonist started climbing, it started to rain mid-climb, and the protagonist slipped on muddy tracks while climbing.

### 2.5 Focused Situation

We focused on the third situation (the protagonist slipped) because it required using the message-passing technique to depict the protagonist, rain, and ground simultaneously. The girls took about 40 min to complete their program.



Fig. 2. Storyboard design sheet designed by X, Y and Z

### 2.6 Levels


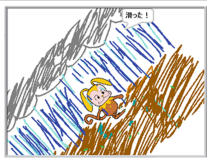



We defined "levels" to depict how the girls' programming completion levels raised on the third situation qualitatively. To distinguish each level, we checked the existence of objects, separation between each object, and appropriateness of rotations of each

object. Five levels were defined, and a higher number of levels indicate a higher state of program completion. Level 1 showed only the protagonist; Level 2 included all of the objects but not appropriate rotation of the rain and the ground because they were not separated from the protagonist; Level 3 did not include the ground; Level 4 had all of the objects, but the ground was not separated from the protagonist; and Level 5 showed the third situation perfectly.

**2.7 Phases**

According to time series, we divided the girls’ transcript into “phases” to summarize their activities. We separated the phases based on which level was focused on by each girl. We did not consider whether their program had been completed or not because we focused on the girls’ viewpoints. The transcript was split into 27 phases.

**Table 2.** Levels of the girls’ programming (✓ means the object exists, +means correct rotation of the object, - means incorrect rotation of the object)

Levels	execution results	protagonist	ground	rain	protagonist	ground	rain
1		✓			+		
2		✓	✓	✓	+	-	-
3		✓		✓	+		+
4		✓	✓	✓	+	-	+
5		✓	✓	✓	+	+	+

### 3 Results

#### 3.1 Summary

First of all, the girls began trying the command of rotation (see Level 1 in Table 2). Then, they depicted the protagonist, the ground, and the rain, as if these three were one object, and gave commands such as “turn 30 degrees.” As a result, they failed to get the protagonist to turn independently from the background (see Level 2 in Table 2). The girls discussed why the protagonist and the ground turned, and eventually they were able to separate the protagonist from the background.

In the activity, the girls showed different trends. X initially remained at Level 1, and then moved between levels 1 and 5. Y was alongside X from Phases 8 to 20. Z mainly focused on Level 5 and did not say much. We will show these trends in Fig. 3.

#### 3.2 Detailed Analysis

Referring to Table 2, we analyzed the utterances and programming actions of X, Y, and Z and plotted the results of this analysis onto a graph (see Fig. 3). We divided the above phases into four scenes according to their activity trends to create an outline. The orange circles in Fig. 3 indicate that programming was executed in that phase.

##### Scene 1: X struggled by herself (Phases 1–6)

X was separated from Y and Z. Y and Z repeatedly explained the desired program motion (Level 5) as depicted in their storyboard design sheet. X attempted to implement their ideas using Pyonkee. However, she failed to make a complete program for the desired motion, and Y and Z conveyed their opinions.

##### Scene 2: X and Y shared the problem (Phases 7–12)

X raised an issue with Y in the form of a question, saying “Hey Y? There’s something that doesn’t work.” She then made the following Level 5 utterance: “We should not put the background in when the protagonist is climbing; the background should only be there when the protagonist is slipping” (Phase 9, Level 5). Y responded: “So I guess it should just be the protagonist that moves?” (Phase 10, Level 5). X then restated her utterance, saying, “We should only put the background in here (when the protagonist is slipping)” (Phase 12, Level 5). However, these ideas were not incorporated into the program during this scene.

Meanwhile, Z was focusing more on the background than on the protagonist. She reminded the others about the background, saying, “I want the background to be rain” (Phase 11, Level 3).

##### Scene 3: X and Y compromised (Phases 13–20)

Despite Z’s reminder, X modified the program in such a way that the protagonist alone slipped (no ground). This was not the result that even X intended. Y again pointed out that the execution result differed from the storyboard design sheet. Then, Y joined X in the iPad operation, and they collaborated to erase the ground from the protagonist object (Phases 14–18).

After some discussion between X and Y—with X around Level 2 and Y around Level 5—they edited the program so that the protagonist slipped along the ground in the rain (Level 3). Even though the program was incomplete, X and Y expressed satisfaction with this result to Z, saying, “Not bad, is it?” (Phases 19 and 20).

**Scene 4: Clarified the Goal (Phase 21–27)**

Z, who was acting as a monitor from Phases 1 to 21, pointed out that there was still a discrepancy between the program and the storyboard design sheet: “When the protagonist slips, there needs to be ground there; without any ground, I cannot understand that the protagonist has slipped on a muddy track!” In response, X drew ground beneath the protagonist, but the ground also turned along with the protagonist. Y disagreed with the results because the ground should be flat. In an attempt to correct this fault, X erased the ground beneath the protagonist. Then, Z intervened again, saying “Now there’s no ground... you got rid of the ground? Why did you do that?” In response, X drew the ground as a background, which means the protagonist was separated from the ground. Consequently, they completed a program that matched their storyboard design sheet.

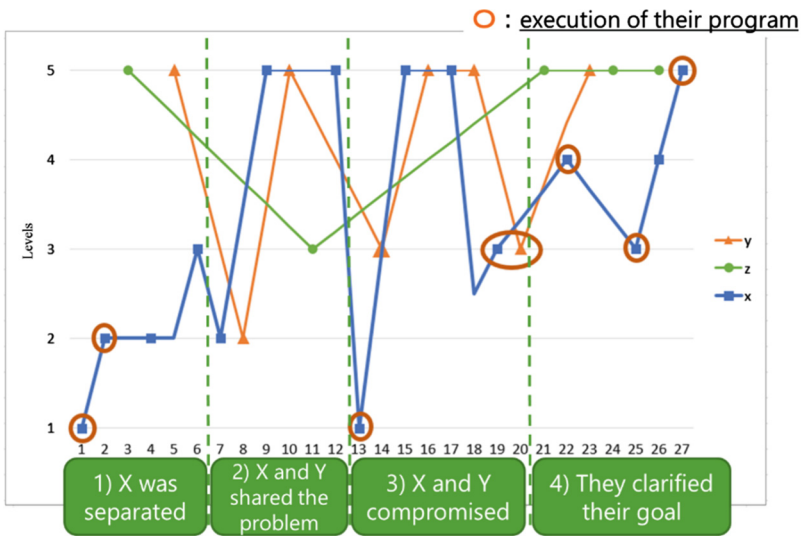


Fig. 3. Levels of the girls' viewpoints in each phase

**4 Discussion**

In conclusion, CI was observed because the completeness of the program was eventually getting higher through the discussion between the task-doer (X) and the monitors (Y, Z). The girls gradually noticed their problems by observing the execution results, solved their problems by discussing them with each other, and eventually raised the program's completion level. These problems were not identified by the task-doer (X) but by the monitors (Y and Z).



As we showed in the example of the discussion, collaborative programming is effective in avoiding the downsizing of goals to match what the learners can do. A statement such as, “There we are. I think it’s complete now,” which we observed in the third scene, is far from rare in a Constructionism-based workshop. In that case, collaborators (often monitors) deny downsizing because the monitor still strives to complete the program as planned. The monitor then contributes to raising their program’s completion level.

Programming has the potential to foster in children a tenacious learning attitude. However, such a learning attitude would never take root if the learner gives up, as in the above example, and sets their sights on a lower goal. From that point of view, collaborative programming has the potential to elevate the activity of programming and to foster children’s creativity. Based on this perspective, the process analysis that we conducted in this study might serve as a measure for assessing children’s collaborative programming processes.

## 5 Conclusion and Future Directions

According to our analysis, it is possible to identify the challenges and difficulties children face in programming. The girls encountered difficulties even though they were only distinguishing (not coding) the objects in our analysis. Furthermore, even after separating the ground and rain from the protagonist, the girls still had to undergo a number of trial-and-error iterations. We think that the reason that the girls raised their completion level of their program is that they continued their endeavor from the viewpoint of asking, “Why has it gone wrong?”

In the example, there were a number of problems that needed to be resolved in order to complete the program. Most often, the person who identified these problems was not the one who engaged directly in the problem-solving operations (who in this case was X), but it was rather the person who closely observed the problem-solver’s work (in this case, Y and Z). According to Shirouzu et al. [22], monitors are better at observing the task objectively than is a task-doer. Kent argued that pair programming improves productivity [23]. This is probably because the strategy of having a task-doer and monitor exchange opinions from their respective positions works effectively.

As the mechanism of CI, a single person notices ambiguous points in his/her thinking when another person asks questions. These questions encourage the task-doer to reconsider the matter in order to resolve such ambiguity, and it encourages him or her to achieve a deeper understanding. In a sense, programming has the potential to enhance the mechanism of CI because it requires the externalization of one’s thinking.

A limitation of this study is that the analysis focused only on one team. In the future, we intend to create an assessment method for evaluating teams that produce different kinds of work and to use this method for evaluating the workshop.

**Acknowledgments.** This study received funding from Shizuoka University’s Takayanagi Memorial Future Technology Creation Fund and a JSPS Grant-in-Aid for Scientific Research (KAKENHI; No. 26280129, 17K17786). In addition, we were able to implement the study owing to the support of Koji Tominaga, a teacher at Hamamatsu Elementary School, attached to the Faculty of Education, Shizuoka University. We would like to express our sincere appreciation to him.

## References

1. Wing, J.: Computational thinking and thinking about computing. *Philos. Trans. Roy. Soc. A* **366**, 3717–3725 (2008)
2. Bocconi, S., Chiocciariello, A., Dettori, G., Ferrari, A., Engelhardt, K.: Developing computational thinking in compulsory education-implications for policy and practice (No. JRC104188). Joint Research Centre (Seville site) (2016)
3. Lye, S., Koh, J.: Review on teaching and learning of computational thinking through programming: what is next for K-12? *Comput. Hum. Behav.* **41**, 51–61 (2014)
4. Barr, V., Stephenson, C.: Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? *ACM Inroads* **2**(1), 48–54 (2011)
5. Tedre, M., Denning, P.: The long quest for computational thinking. In: *Proceedings of the 16th Koli Calling Conference on Computing Education Research*, pp. 120–129 (2016)
6. Papert, S.: *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York (1980)
7. Papert, S.: Situating Constructionism. In: Harel, I., Papert, S. (ed.) *Constructionism*, pp. 1–12 (1991)
8. Papert, S.: Perestroika and epistemological politics. In: Harel, I., Papert, S. (eds.) *Constructionism*, pp. 13–28 (1991)
9. Kay, A., Goldberg, A.: Personal dynamic media. *Computer* **10**(3), 31–41 (1977)
10. Nickerson, R.S.: Computer programming as a vehicle for teaching thinking skills. *Think. J. Philos. Child.* **4**, 42–48 (1982)
11. Hayes-Roth, B., Hayes-Roth, F.A.: A cognitive model of planning. *Cogn. Sci.* **3**, 275–310 (1979)
12. Lawler, R.W.: The progressive construction of mind. *Cogn. Sci.* **5**, 1–30 (1981)
13. Clements, D.H., Gullo, D.F.: Effects of computer programming on young children's cognition. *J. Educ. Psychol.* **76**(6), 1051–1058 (1984)
14. Pea, R.: Logo programming and problem solving. Paper Presented at Symposium of American Educational Research Association (1983)
15. Pea, R., Kurland, D.: On the cognitive effects of learning computer programming. *New Ideas Psychol.* **2**(2), 137–168 (1984)
16. Webb, N.M.: Microcomputer learning in small groups: cognitive requirements and group processes. *J. Educ. Psychol.* **76**, 1076–1088 (1984)
17. Webb, N.M., Ender, P., Lewis, S.: Problem-solving strategies and group processes in small groups learning computer programming. *Am. Educ. Res. J.* **23**(2), 243–261 (1986)
18. Brennen, K., Resnick, M.: New frameworks for studying and assessing the development of computational thinking. In: *Annual Meeting of the American Educational Research Association* (2012)
19. Hmelo-Silver, C.E., Chinn, C.A., Chan, C.K.K., O'donnell, A., (eds.): *The International Handbook of Collaborative Learning*. Routledge, New York (2013)
20. OECD: *PISA 2015 Collaborative Problem Solving Framework*. OECD Publishing (2013)
21. Miyake, N.: Constructive interaction and the iterative process of understanding. *Cogn. Sci.* **10**(2), 151–177 (1986)
22. Shirouzu, H., Miyake, N., Masukawa, H.: Cognitively active externalization for situated reflection. *Cogn. Sci.* **26**(4), 469–501 (2002)
23. Beck, K., Andres, C.: *Extreme Programming Explained: Embrace Change*, 2nd edn. Addison-Wesley Professional, Boston (2004)