Journal of Computing in Higher Education (2022) 34:489–516
https://doi.org/10.1007/s12528-021-09307-w

An exploratory study on fade-in versus fade-out scaffolding for novice programmers in online collaborative programming settings

Lanqin Zheng[1] · **Yuanyi Zhen**[1] · **Jiayu Niu**[1] · **Lu Zhong**[1]

Accepted: 23 December 2021 / Published online: 21 January 2022

Abstract

Programming skills have gained increasing attention in recent years because digital technologies have become an indispensable part of life. However, little is known about the roles of fade-in and fade-out scaffolding in online collaborative programming settings. To close this research gap, the present study aims to examine the roles of fade-in and fade-out scaffolding for novice programmers in online collaborative programming. A total of 90 undergraduate students participated in the exploratory study and were assigned to 15 fade-in groups and 15 fade-out groups. All of the participants completed the same programming task. The findings reveal that fade-in scaffolding can significantly improve collaborative knowledge building, programming skills, metacognitive behaviors, emotions, and collective efficacy. Goal setting, planning, monitoring and control, enacting strategies, and evaluation and reflection are identified as the crucial metacognitive behaviors. The main contribution of this exploratory study is to shed light on how to design and implement scaffolding for novice programmers.

Keywords Fade-in scaffolding · Fade-out scaffolding · Collaborative programming · Programming skills · Metacognitive behaviors · Emotion

Introduction

Programming skills have been widely recognized as one of the most important set of skills for twenty-first century success in many countries (Exter & Ashby, 2019; Wu et al., 2020). The mastery of programming skills can help learners improve their problem-solving abilities (Fessakis et al., 2013) and computational

✉ Lanqin Zheng
bnuzhenglq@bnu.edu.cn

[1] School of Educational Technology, Faculty of Education, Beijing Normal University, No. 19, XinJieKouWai St., HaiDian District, Beijing 100875, People's Republic of China

thinking (Sentance & Csizmadia, 2016). However, many students have difficulties and encounter challenges in programming. For example, Mladenović et al. (2016) found that students have difficulties in problem understanding, language syntax knowledge, and debugging. Mohd Rum et al. (2017) revealed that many students have difficulties in planning programming, testing program output, and evaluating programming solutions. In addition, many students do not like programming and they believe that programming is boring and horrible (Mathrani et al., 2016). Students become disengaged in programming because it is a notoriously difficult subject (Giacaman & De Ruvo, 2018). Furthermore, many students have a superficial understanding of programming concepts (Kunkle & Allen, 2016) and they lack a sufficient amount of hands-on programming experience (Yeomans et al., 2019). Previous studies revealed that scaffolding is an effective strategy for overcoming these challenges in programming (Margulieux & Catrambone, 2021; Zhang et al., 2021). Therefore, it is necessary to provide scaffolding for learners to overcome difficulties during programming.

Previous studies have attempted to provide fixed and static support for novice programmers through the use of technology. For example, Phuong and Shimakawa (2008) developed a collaborative programming environment for programmers, and they found the developed environment enhanced the capabilities of novice programmers. Mohd Rum et al. (2017) developed a metacognitive scaffolding learning environment for novice programmers and found that learners with metacognitive scaffolding outperformed learners without metacognitive scaffolding. However, the use of advanced twenty-first century technology alone cannot guarantee the desired learning outcomes (Bond, 2020). Productive and successful learning requires the thoughtful integration of technology and pedagogy (Janssen et al., 2019). Therefore, this study focuses on how to provide fade-in and fade-out scaffolding for novice programmers in collaborative programming environment. Fade-in scaffolding refers to scaffolding that is gradually introduced and fade-out scaffolding refers to scaffolding that is gradually removed.

Previous studies indicated that learning performance was closely related to the timing of the scaffolding (Shin et al., 2020). There is a debate about the roles of fade-in and fade-out scaffolding. For example, Tawfik et al. (2018) revealed that learners with fade-in scaffolding outperformed those with fade-out scaffolding in ill-structured problem solving. On the contrary, Tullis et al. (2015) found that fading-out scaffolding improved learning more than constant scaffolding in a mouse movement task. In addition, although previous studies have developed systems to provide scaffolding for individual programming (Mohd Rum & Ismail, 2017; Sun & Hsu, 2019), it is challenging to scaffold collaborative programming effectively. Collaborative programming is particularly helpful for novice programmers because it can offer opportunities to establish a shared understanding as well as to code, test, and debug projects along with their peers (Teague & Roe, 2008). However, very few studies have examined the roles of scaffolding fading in the collaborative programming field. Therefore, the aim of this study is to address the research gaps and examine the roles of fade-in versus fade-out scaffolding in collaborative programming.

The remainder of the paper is organized as follows. First, a literature review details the research status of collaborative programming and scaffolding. Next, we

present a study comparing the roles of fade-in versus fade-out scaffolding in a collaborative programming context. Finally, the findings of the present study are summarized and discussed. As we move into the discussion section, there are a set of limitations associated with this study that must be kept in mind.

Literature review

Collaborative programming

Information technology has developed rapidly in recent years, and it requires a large human workforce with programming skills (Lu et al., 2017). Therefore, programming skills have become increasingly important and have become a core objective of undergraduate and graduate programs in engineering fields (Wang & Hwang, 2017). To improve programming skills, collaborative programming has been widely adopted in many schools. Collaborative programming, in which a group of learners work on the same code and complete programming tasks together, is considered an effective pedagogical approach (Nosek, 1998). Collaborative programming aims to improve learners' programming skills through writing code and refining programs with peers (Lu et al., 2017). Furthermore, the benefits of collaborative programming, such as improving programming performance (Wang & Hwang, 2017), building computational thinking competence (Denner et al., 2014), improving problem-solving abilities, and developing a higher level of confidence (Beck & Chizhik, 2013), have been well documented in the literature.

However, most students have many difficulties with collaborative programming. For example, many students are not willing to collaborate with peers during collaborative programming (Wei et al., 2021), and experienced coders demonstrate less enthusiasm about collaborative programming (Bowman et al., 2020). Students often lose confidence and decrease engagement when they encounter difficulties with programming (Mladenović et al., 2016). Furthermore, students have difficulties in establishing a shared understanding through conflict negotiation during collaborative programming (Wu et al., 2019). Therefore, it is necessary to provide scaffolding during collaborative programming. To the best of our knowledge, few studies have investigated how to provide scaffolding to improve collaborative programming skills. Therefore, this study attempts to bridge this gap and examine fade-in versus fade-out scaffolding in online collaborative programming environments.

Scaffolding and the fading paradigm

Scaffolding was proposed on the basis of the zone of proximal development, which is defined as the distance between the actual developmental level and the potential development level (Vygotsky, 1978). Scaffolding is conceptualized as assistance from a more knowledgeable peer or an adult (Wood et al., 1976). It has been found that learning is the most effective within learners' zone of proximal development through scaffolding (Yu & Hu, 2017). One important aspect of scaffolding is

how the scaffolding fades over time (Puntambekar & Hubscher, 2005). There are two types of fading paradigms, namely fade-in and fade-out (Jennings & Muldner, 2020). Traditionally, scaffolding should be fade-out when learners do not need it anymore (Lajoie, 2005). In fade-in scaffolding, scaffolding is gradually introduced, which is aligned with the theories of productive failure proposed by Kapur (2008). Kapur and Bielaczyc (2012) proposed that productive failure includes two phases, namely the exploration phase without scaffolding and the consolidation phase with scaffolding. Productive failure is characterized as scaffolding transitioning from low to high. Kapur (2016) believes that solving problems without scaffolding is a productive exercise and initial failures contribute to future learning.

However, there are differing opinions about the roles of fading scaffolds. For example, Bulu and Pedersen (2010) found that learners with continuous scaffolding outperformed those with fade-out scaffolding. In contrast, Kalyuga and Renkl (2010) believe that learners benefit more when scaffolding fades out. Therefore, it would be very interesting to investigate which type of fading paradigms is more effective. Although Wang et al. (2021) conducted an exploratory case study to investigate how a teacher can support preschoolers' programming, how to fade scaffolding for novice programmers was not investigated in the study. To the best of our knowledge, very few studies have investigated how to provide fade-in and fade-out scaffolding in an online collaborative programming environment. To clarify the roles of fading scaffolding, this study examines the roles of fade-in and fade-out scaffolding for novice programmers in such an environment. In addition, the fade-in and fade-out scaffolding was provided by teachers in this study. Although technology scaffolding excels at organizing knowledge (Sung & Hwang, 2013) and promoting social interactions (Molenaar et al., 2014), it fails to provide an ongoing assessment and personalized guidance (Jennings & Muldner, 2020). Moreover, learners perceive guidance from computers as generic and unresponsive when compared with teacher guidance (Tansomboon et al., 2017). Furthermore, Furberg (2016) revealed that learners still need substantial teacher guidance even in a well-scaffolded learning environment. Teachers can provide explanations and personalized feedback for learners (Yilmaz & Yilmaz, 2020), and a teacher's involvement can promote positive interactions (Kaendler et al., 2015), increase learning engagement (Xu et al., 2020), and foster collaboration (van Leeuwen & Janssen, 2019). Therefore, teachers cannot be replaced because they play a very important role as an educator, a facilitator, and a mentor (Mathrani et al., 2016).

Research questions

The present study aims to examine the roles of fade-in versus fade-out scaffolding on collaborative knowledge building, programming skills, metacognitive behaviors, emotions, and collective efficacy. The following research questions were addressed:

- RQ1: Do the students who learn with fade-in scaffolding build a higher level of collaborative knowledge than those who learn with fade-out scaffolding?

- RQ2: Do the students who learn with fade-in scaffolding demonstrate better programming skills than those who learn with fade-out scaffolding?
- RQ3: Do the students who learn with fade-in scaffolding demonstrate more metacognitive behaviors than those who learn with fade-out scaffolding?
- RQ4: Do the students who learn with fade-in scaffolding have more positive emotions than those who learn with fade-out scaffolding?
- RQ5: Do the students who learn with fade-in scaffolding demonstrate higher collective efficacy than those who learn with fade-out scaffolding?

Methods

Participants

This study was conducted in a higher education context, and the participants came from public universities located in the north of China. A total of 90 undergraduate students who had enrolled in a course of C programming in the first or second year of college voluntarily participated in this study. Power analyses for a-priori sample sizes for t-tests and F-tests indicated a required minimum sample size of N = 84 for an anticipated large effect size, a statistical power level of $\beta = 0.95$, an α-level of 0.05 (Faul et al., 2007). Therefore, the sample size of 90 participants was a statistically fair sample. There were 59 males and 31 females, and the average age of participants was 21 years. All of the participants were assigned into 15 fade-in groups or 15 fade-out groups. There were no significant differences in gender ($X^2 = 3.15$, $p = 0.07 > 0.05$), major ($X^2 = 0.05$, $p = 0.82 > 0.05$), age ($z = 1.26$, $p = 0.08 > 0.05$), or course grades of programming in C ($t = 0.09$, $p = 0.93 > 0.05$) between the fade-in group and fade-out group. Therefore, these two groups are statistically similar to each other in terms of gender, age, major, and course grades of programming in C. There were three students in each group. All participants had prior knowledge of C programming, but they were still novice programmers because they did not have extensive experience in C programming.

Procedure

The exploratory procedure is shown in Fig. 1. Before the exploratory experiment, all of the participants performed a pre-test to examine their level of prior knowledge. There was no significant difference in prior knowledge about C programming between the fade-in and fade-out groups ($t = 0.47$, $p > 0.05$). Next, all participants took part in an online collaborative programming project using the Visual Studio integrated development environment and completed the same task for three hours. The Visual Studio integrated development environment includes an editor, terminal, and text-based chat window, in which participants can collaboratively program and discuss the task with peers. Figure 2 shows a screenshot of the collaborative programming environment. The programming task was to develop a three-player

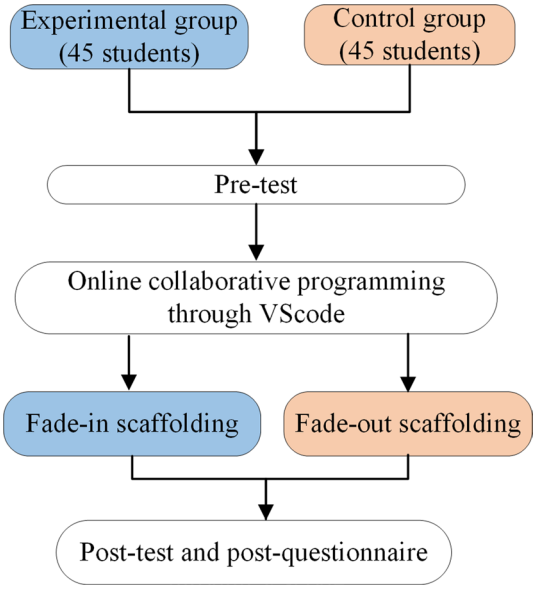

Fig. 1 The diagram of the exploratory experiment design

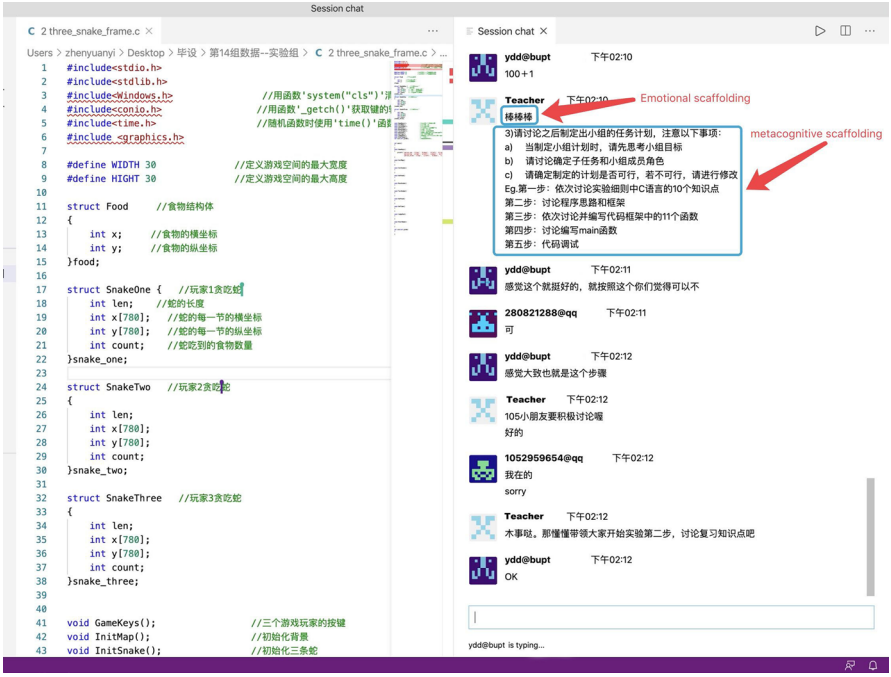

Fig. 2 The collaborative programming environment

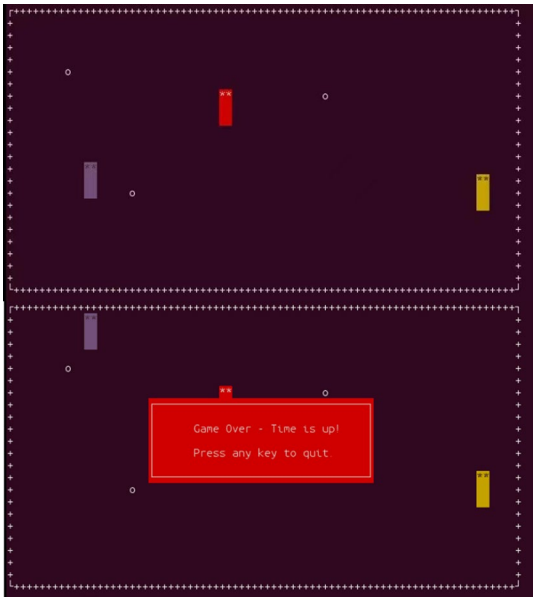

Fig. 3 The screenshot of a three-player hungry snake game

hungry snake game using the C programming language. Figure 3 shows the process and outcome of a three-player hungry snake game.

Table 1 presents the details of the three types of scaffolding in each group. The three types in the fade-in group were gradually introduced and only provided when participants could not solve problems independently. More specifically, scaffolding was not provided if this group met the task requirements on their first attempt, and they had to first struggle to solve problems by themselves. The three types of scaffolding in fade-out group were only provided a single time when certain conditions were met and then removed during programming. After online collaborative programming, all participants took a post-test and answered a questionnaire on collective efficacy.

Instruments

The main measurement tools of this study were the pre-test, post-test, and questionnaire on collective efficacy. Both the pre-test and post-test were adapted from Tan (2017). The aim of the pre-test was to evaluate participants' prior knowledge about the C programming language. It consisted of six multiple-choice questions, four fill-in-the-blank questions, and two programming questions, with a full score of 100. An example multiple-choice question of pre-test is shown in Fig. 5. An example fill-in-the-blank question is "The three types of iteration statements in C programming language include ——, ——, and ——." The two programming questions were "Please write a program to calculate the value of '$100+101+102+\cdots+300$" and "Please write a program that takes 10 integers as input, stores them in a one dimensional

Table 1 Types of scaffolding

Types	Sub-types	Conditions	Examples
Cognitive scaffolding	Concept explanations	When students had misconceptions	Loop control statement "while", "do while", "for": • while (conditional statement) {executed statement;} • do {executed statement;} while (conditional statement) • for (int i = 0; i < 10, i + +) {executed statement;}

	Coding examples	When students don't know how to code	if(map[snake_one.x[0]][snake_one.y[0]] = = 2)
			{
			snake_one.len++;
			snake_one.count++;
			map[snake_one.x[0]][snake_one.y[0]]
			= 1;
			}

	Concept maps	When students are confused about the concepts and their relationships	The example of concept map is shown in Fig. 4

Table 1 (continued)

Types	Sub-types	Conditions	Examples
Meta-cognitive scaffolding	Set goals	• At the beginning of collaborative programming • When group members did not set sub-goals	✓ What is learning goal when you complete the task? If your group have not set goals, please set the learning goal at the beginning ✓ It is suggested that your group can set sub-goal when you program
	Make plans	When group members did make a plan	✓ How to complete the programming task? Please make a detailed plan first ✓ Please design a flow chart to represent the algorithm ✓ It is suggested that your group can assign roles based on abilities ✓ Your group can make a schedule to manage time
	Monitor	• During collaborative programming • When group members make mistakes in programming • When group members did not how to collaborate	✓ How is going? How far is from the target? ✓ There are some errors in programming. Please revise it immediately ✓ Your group has not achieved the goal. Please focus on the task and debugging ✓ Communicate with peers when you have any ideas. You can test and debug together ✓ If you have some conflicts, please express your ideas first, and then analyze the advantages and disadvantages with your peers
	Evaluate and reflect	• At the end of collaborative programming • When group members did not reflect and evaluate	✓ When you complete the task, please evaluate and reflect the group products. If there is any problem, please revise first ✓ It is suggested that your group can reflect the whole collaborative learning process and summarize the advantages and disadvantages

Table 1 (continued)

Types	Sub-types	Conditions	Examples
Emotional scaffolding	Positive	When students express the positive feelings	✓ The discussion atmosphere is so hot! Go ahead and Come on! ✓ This idea is wonderful! Awesome! ✓ It's so great! Good job!
	Negative	When students were sad, anxious, and unconfident	✓ When you have some difficulties, discuss with peers and I believe that you can overcome any difficulties ✓ Don't be anxious. Take it easy ✓ Don't be sad
	Neural	When students did not any progress	✓ I hope your group can be more positive and contribute more ✓ You are the best ones. Be more active
	Confused	When students were confused and don't know how to proceed	✓ If you can think from different perspectives, you will find solutions ✓ Please discuss with your peers to solve the problem

array, and then outputs them in reverse order." The post-test consisted of three multiple-choice questions, six fill-in-the-blank questions about writing code, and one programming item to evaluate what participants had learned during the collaborative programming. The full score of the post-test was also 100. An example multiple-choice question of post-test is shown in Fig. 5. An example fill-in-the-blank questions is "Please write code to determine whether or not two snakes have collided." The programming question was "Please write a program to make a snake move horizontally and increase its body length by 1 unit in every 2 s. After 2 min, end the program automatically." These items were closely related to the structured programming method, functions, and algorithms in C programming, and they can represent a participant's programming skills to a large extent. Furthermore, the differentiation of the pre-test and post-test were 0.30 and 0.52, indicating good differentiation. The difficulty of the pre-test and post-test were 0.75 and 0.59, implying appropriate difficulty. The homogeneity reliability of the pre-test and post-test reached 0.83 and 0.92, indicating good reliability. The validities of the pre-test and post-test were examined and confirmed by two experts who had extensive experience in C programming. The inter-rater agreement of the pre-test and post-test were 0.82 and 0.87, indicating good validity.

Collective efficacy is defined as a group's shared beliefs in its abilities to execute tasks to achieve group goals (Bandura, 1997). The questionnaire of collective efficacy was adapted from Zheng (2017). It includes two types of constructs, one is the belief in a group's abilities to achieve the goal, and the other is the perception of commitment to the group work. The collective efficacy questionnaire consists of 10 items with a 5-point Likert scale. The Cronbach's alpha value of the questionnaire was 0.85, indicting excellent reliability. In addition, the confirmatory factor analysis (CFA) was further conducted to examine the construct validity of the collective efficacy questionnaire. The CFA results revealed that $X^2/df = 1.26$, the root mean square error of approximation (RMSEA) = 0.05, the goodness of fit index (GFI) = 0.95, the comparative fit index (CFI) = 0.98, the incremental fit index (IFI) = 0.98, and the Tucker–Lewis index (TLI) = 0.98. The model fits the data well and satisfied the threshold values according to Hair et al. (2010). Therefore, the construct validity of the questionnaire was confirmed further. Moreover, the results indicated that all of the factor loading values ranged from 0.53 to 0.86, which satisfied the threshold value of 0.5 according to Bagozzi and Yi (1988). The composite reliability (CR) of the two constructs were 0.72 and 0.82, which satisfied the threshold value of 0.60 according to Bagozzi and Yi (1988). The average variance extracted (AVE) values of the two constructs were 0.50 and 0.56, which satisfied the threshold value of 0.50 based on Bagozzi and Yi (1988). Therefore, the collective efficacy questionnaire achieved good convergent validity.

Data analysis methods

This study adopted a knowledge-map-based analysis method to analyze 8913 discussion transcripts to measure collaborative knowledge building level. This analysis method was proposed and validated by Zheng et al. (2015), and it consists of three

steps. The first step is to draw a target knowledge map. The second step is to code discussion transcripts based on predefined rules by two coders. The inter-rater reliability of two coders was 0.79, indicating good reliability. The last step is to calculate collaborative knowledge building level and generate a knowledge graph for each group. The collaborative knowledge building level is equal to the activity quantities of all knowledge nodes in the knowledge graph. The activity quantity of each knowledge node refers to the information entropy of online discussion transcripts, which can be calculated through our analytical tool.

Programming skills were measured through the post-test and group products. The group products were evaluated on the basis of the rubric shown in Table 2. Furthermore, the pre-test, post-test, and group products were analyzed by two experienced teachers who are blinded to experimental condition and had rich experiences in coding procedure and programming knowledge. The inter-rater reliability of the group products was calculated using Kappa statistics and was 0.895, indicating good reliability.

Content analysis and lag-sequential analysis were adopted to analyze the metacognitive behavioral patterns. The two coders are blinded to subject identity and experimental condition to analyze discussion transcripts of all groups based on the coding scheme (Table 3). This coding scheme was based on that of Zheng et al. (2019). The inter-rater reliability was 0.9, indicating excellent reliability. Next, GSEQ 5.1, developed by Quera et al. (2007), was adopted to calculate the behavioral transition through the z-score. The z-score refers to the adjusted residual that can represent whether the behavioral sequence is more significant than expected by chance (Bakeman & Gottman, 1997). If the z-score is larger than 1.96, it indicates that the behavioral sequence is statistically significant (Bakeman & Gottman, 1997). The z-score is sensitive to sample size (Bakeman & Quera, 2011) and this study analyzed 8913 behavior codes to examine the behavioral transition. Moreover, the emotional status was classified as positive, negative, neutral, or confused based on Zheng and Huang (2016). Two coders classified the emotional status of the 30 groups and the inter-rater reliability was 0.9, indicating good reliability.

Results

Analysis of collaborative knowledge building

To examine the difference in collaborative knowledge building, the normality of the distributions was tested using the Kolmogorov–Smirnov test. The result reveals that all data are normally distributed ($p > 0.05$). The homogeneity of variance was also evaluated, and the result indicates that the homogeneity of variance is not violated ($F = 0.177$, $p = 0.678 > 0.05$). Therefore, analysis of covariance (ANCOVA) can be used to examine the differences between the fade-in group and fade-out group by excluding the impact of the pre-test scores. Table 4 shows the results of ANCOVA. The finding reveals that there is a significant difference in collaborative knowledge building between the fade-in and fade-out groups ($F = 5.410$, $p = 0.028 < 0.05$). Furthermore, the fade-in group outperformed the fade-out group. Figures 6 and 7

show the knowledge graphs of one fade-in group and fade-out group. The nodes and edges of the knowledge graphs represent the knowledge and their mutual relationships, respectively. The numbers next to the nodes denote the activation quantities that were calculated using our analytical tool. It is very clear that the fade-in group activated more knowledge nodes and relationships than the fade-out group.

Analysis of programming skills

Programming skills were measured through post-test and group products. The normality distribution and homogeneity of variance were evaluated to examine whether the data could be analyzed using ANCOVA. The Kolmogorov–Smirnov test results reveal that all data sets are normally distributed ($p > 0.05$). In addition, the findings reveal that the homogeneity of variance is not violated for the post-test ($F = 0.006$, $p = 0.939 > 0.05$) and group products ($F = 0.093$, $p = 0.762 > 0.05$). Thus, ANCOVA can be used to examine the differences in the post-test and group products of the fade-in and fade-out groups by excluding the impact of the pre-test scores. Two ANCOVA for the post-test and group product scores were conducted using the pre-test score as the covariant and the fading paradigm as the independent variable, and Table 5 shows the results. The analysis shows that there is a significant difference in the post-test scores of the fade-in and fade-out groups ($F = 10.383$, $p = 0.002 < 0.01$). Moreover, the adjusted means of the fade-in group are significantly higher than those of the fade-out group. Therefore, fade-in scaffolding can significantly improve group learning achievement.

The results also reveal that there is a significant difference in group product scores between the fade-in and fade-out groups ($F = 146.879$, $p = 0.000 < 0.001$). Furthermore, the fade-in groups outperform the fade-out groups in terms of group product score. Figures 8 and 9 show portions of the group products of the fade-in and fade-out groups, respectively.

Analysis of metacognitive behaviors

The lag-sequential analysis method was adopted to analyze metacognitive behavior transitions. Tables 6 and 7 are the adjusted residuals tables for the fade-in and fade-out groups, respectively. In the two tables, the behaviors in the left-most column indicate the starting behaviors and the behaviors in the top-most row indicate the subsequent behaviors. The findings reveal that there are 13 significant behavior sequences for the fade-in group: OG → OG, OG → MP, MP → MP, MP → MC, ES → ES, ES → MC, MC → OG, MC → MP, MC → ES, ER → MC, ER → ER, AM → ER, and AM → AM. There are only seven significant behavior sequences for the fade-out group, namely OG → OG, MP → MP, ES → ES, MC → MP, MC → MC, ER → ER, and AM → AM. Figures 10 shows the behavioral transition diagrams of the fade-in and fade-out groups.

There are several differences between the fade-in group and fade-out group, as shown in Table 8 and Fig. 10. First, the fade-in group showed more behavioral transition than the fade-out group. There are seven significant behavior sequences

that only occurred in the fade-in group, namely OG → MP, MP → MC, ES → MC, MC → OG, MC → ES, ER → MC, and AM → ER. This result indicates that goal setting, planning, monitoring and control, enacting strategies, and evaluation and reflection could be the crucial behaviors resulting in better programming performance. Second, the fade-out groups show more repetitive behavior transitions, such as OG → OG, MP → MP, and so on. Overall, the fade-in groups show a stronger metacognitive behavioral transition than the fade-out group.

Analysis of emotional status

To examine the differences in emotional state of the fade-in and fade-out groups, the independent samples *t*-test was adopted to analyze the data. As shown in Table 9, the results indicate that there is a significant difference in the positive emotion of the fade-in and fade-out groups ($t = 2.59$, $p < 0.05$, $d = 0.94$). There is a large effect size for positive emotion according to Cohen (1988). However, there are no significant differences in negative emotions ($t = 1.92$, $p > 0.05$, $d = 0.54$), neutral emotions ($t = 1.48$, $p > 0.05$, $d = 0.70$), and confused emotions ($t = 0.97$, $p > 0.05$, $d = 0.35$) between the fade-in and fade-out groups. The fade-in group demonstrated less negative, neutral, and confused emotions. These findings reveal that fade-in scaffolding can significantly encourage more positive emotions.

Analysis of collective efficacy

The mean of collective efficacy was 3.98 for the fade-in group and 3.54 for the fade-out group, respectively, indicating the fade-in group outperformed the fade-out group (Table 10). The results of the independent samples *t*-test reveal that there is a significant difference in the collective efficacy between the fade-in and fade-out groups ($t = 3.29$, $p < 0.01$, $d = 0.70$). Therefore, fade-in scaffolding can significantly improve the collective efficacy of the group.

Discussion

This exploratory study examined the roles of fade-in scaffolding and fade-out scaffolding on collaborative knowledge building, programming skills, metacognitive behaviors, emotions, and collective efficacy in an online collaborative programming environment. The results indicate that the learners with fade-in scaffolding significantly outperformed those who learned with fade-out scaffolding in terms of knowledge building, programming skills, metacognitive behaviors, emotions, and collective efficacy. Therefore, fade-in scaffolding may be more useful than fade-out scaffolding for novice programmers.

This study revealed that fade-in scaffolding could significantly improve knowledge building and programming skills. The main reason lay in that the scaffolding was gradually introduced and increased during collaborative programming to engage participants in exploration and solving problems. The cognitive scaffolding

provided in this study was in the form of concept maps, coding examples, and concept explanations. Previous studies revealed that novice programmers faced difficulties in understanding basic programming concepts (Koorsse et al., 2015; Tsai, 2019). Menon and Kovalchick (2020) found that concept maps are helpful for understanding programming concepts through organizing and key concepts and their relationships. Therefore, the concept maps provided in this study (as shown in Fig. 4) demonstrated programming concepts and their relationships to improve the level of collaborative knowledge building. In addition, coding examples can lower the difficulty and improve efficiency for programmers (Yaghmazadeh et al., 2018). Thus, the coding examples provided in this study contributed to improving programming skills. Moreover, concept explanations were also provided for students to clarify their misconceptions. All in all, the fade-in cognitive scaffolding helped improve knowledge building levels and programming skills.

Programmers need metacognition support to understand programming problems, correct programming errors, and test program output (Mohd Rum & Ismail, 2017).

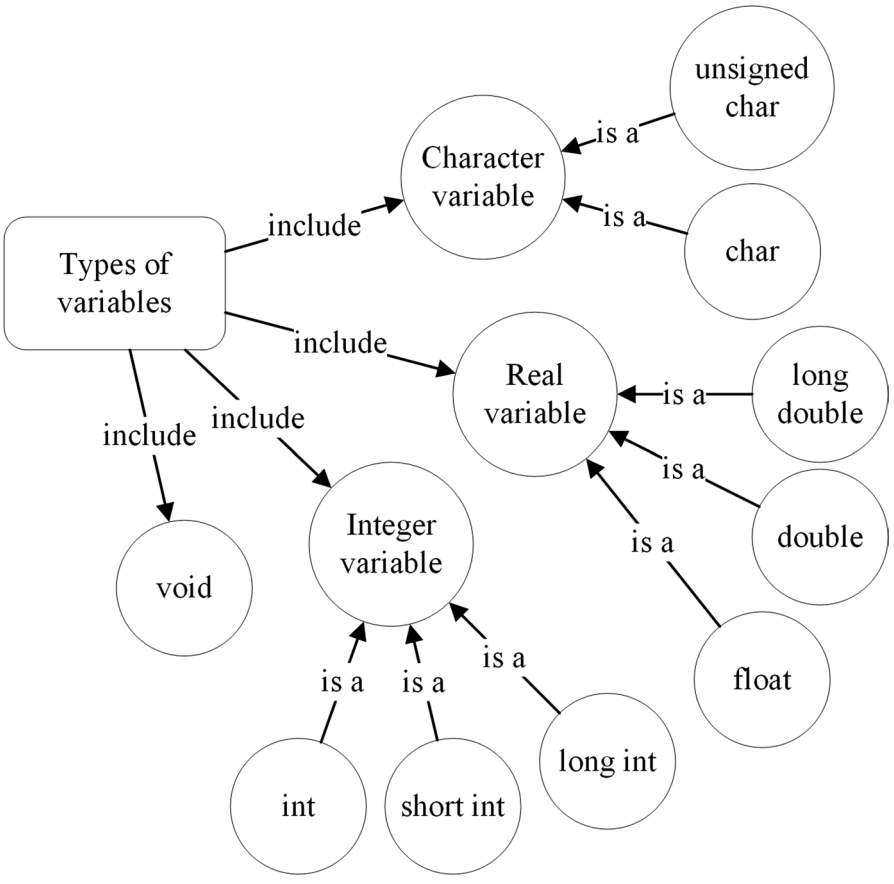

Fig. 4 The example of a concept map

An example multiple-choice question of pre-test:

Which of the following is equal to '!x' in the statement of While(!x)?

(A) x!=0;

(B) x==1;

(C) x!=1;

(D) x==0.

An example multiple-choice question of post-test:

What is the data transfer between the actual parameter variable and formal parameter variable when calling a function?

(A) Address transfer;

(B) pass by value;

(C) transfer from the actual parameter to the formal parameter and then pass back from the formal parameter to the actual parameter;

(D) delivery method specified by the user.

Fig. 5 The example multiple-choice questions of pre-test and post-test

Table 2 Rubric for group product evaluation

Dimensions	Descriptions
Feasibility (30 scores)	• The program logic is clear and reasonable • The program can be executed correctly • The program results meet the requirements of task • The code is easily readable
Completeness (30 scores)	• The header file is complete • The data type is complete • The main function is complete • The functional definition is complete
Correctness (30 scores)	• The header file is correct • The data type is correct • The function is correct • The input and output are correct
Novelty (10 scores)	• The algorithm is innovative • The time complexity is lower than previous program • The space complexity is lower than previous program

The findings of this study demonstrate that the fade-in metacognitive scaffolding significantly increases metacognitive behaviors and behavioral transition. This result corroborates with previous findings that metacognitive scaffolding can offer substantial benefits for online collaborative learning (Kwon et al., 2013). The metacognitive scaffolding provided in this study aimed to help students to set programming goals, make plans, monitor progress, reflect, and evaluate as well as adapt their

Table 3 Coding scheme for metacognitive behaviors

Code	Behaviors	Descriptions
OG	Set goals	Set learning goals and establish task demands
MP	Make plans	Make plans about how to reach the group's goals, including select strategies, set timelines, negotiate the division of labor
MC	Monitor and control	Monitor the collaborative programming processes; claiming (partial) understanding or comprehension failure; detecting errors, and debugging
ER	Evaluate and reflect	Evaluate current solutions; reflecting on the group goals, processes, and outcomes
AM	Adapt metacognition	Make adaptions to learning goals, plans, or strategies

Table 4 Summary of ANCOVA results on collaborative knowledge building

Group	N	Mean	SD	Adjusted mean	Std error	F	η^2
Fade-in group	45	689.18	203.78	699.13	55.83	5.410*	0.167
Fade-out group	45	524.05	232.53	514.10	55.83		

*$p < 0.05$

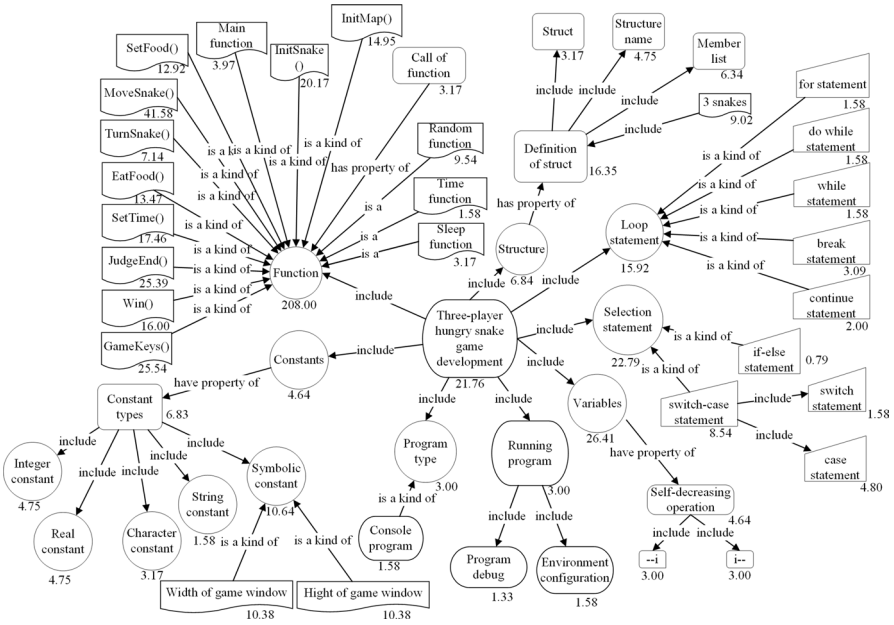

Fig. 6 A knowledge graph of a fade-in group

metacognition. These metacognitive behaviors indeed resulted in better programming performance, on the basis of the findings in this study. Furthermore, the present study revealed that goal setting, planning, monitoring, enacting strategies and evaluating and reflecting could be the crucial metacognitive behaviors. As Mohd Rum et al. (2017) found, outstanding programmers adopt more metacognitive strategies than novice programmers. Therefore, the metacognitive scaffolding elicited more metacognitive behaviors and increased programming skills.

This study revealed that fade-in emotional scaffolding promotes positive emotions and decreases negative, neutral, and confused emotions. The main reason could be that the teachers provided fade-in emotional scaffolding when the learners demonstrated various kinds of emotions. Teachers can help learners to regulate negative, neutral, and confused emotions. For example, when learners felt sad and anxious during programming, the teachers encouraged them to think positively and find help from peers. On the contrary, learners' emotions could not be captured in

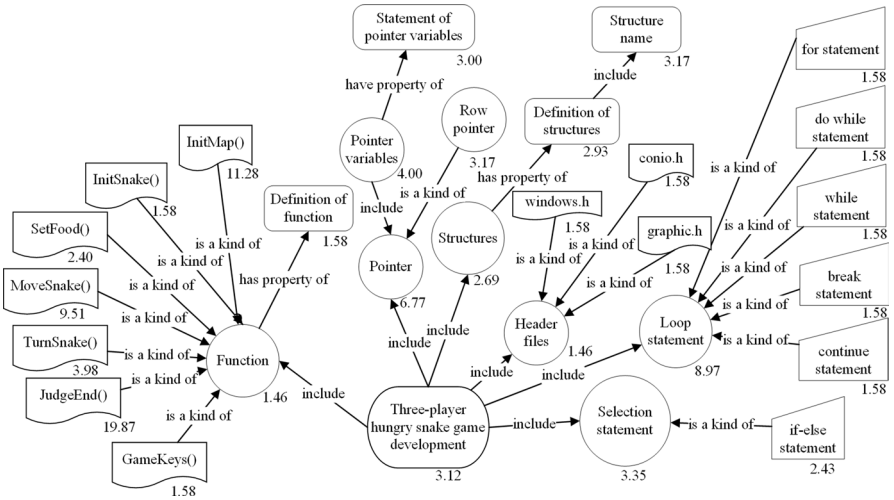

Fig. 7 A knowledge graph of a fade-out group

real time when scaffolding was faded out, which led to more negative, neutral, and confused emotions in the fade-out group. In addition, the present study found that the fade-in group demonstrated higher collective efficacy than the fade-out group. This result could be explained by Bandura's (1997) views, that engaging participants in collaborative learning with scaffolding increases a group's efficacy beliefs.

Regarding the implications for research and practice in collaborative programming, several points can be made according to the results. First, fade-in scaffolding is better than fade-out scaffolding for novice programmers. Furthermore, novices need scaffolding for a long time (Tawfik et al., 2018). The findings reveal that teachers and practitioners should provide a set of scaffolding for novice programmers during programming. Second, this study found that teachers cannot be replaced by tools and learners still need teacher guidance in programming. Even with the aid of technology, teachers still need to make explanations based on analysis results and make decisions based on data (Yilmaz & Yilmaz, 2020). Third, learners need multiple kinds of scaffolding during collaborative programming, including cognitive, metacognitive, and emotional scaffolding. The results of this study indicate that cognitive scaffolding in the forms of concepts maps, coding examples, and concept explanations are very helpful for improving programming skills. In addition, metacognitive scaffolding can elicit more metacognitive behaviors and help novice programmers to plan, monitor, and evaluate during programming. Emotional scaffolding can promote positive emotions and motivate novice programmers to improve their programming skills.

However, this study was constrained by several limitations. First, this study only focused on one online collaborative programming task and the duration of the online collaborative programming was short due to the impact of COVID-19 pandemic. Caution should be exercised when generalizing the findings to other contexts. Second, this study only examined the roles of fade-in and fade-out scaffolding in

Table 5 Summary of ANCOVA result on the post-test and group product

Items	Group	N	Mean	SD	Adjusted mean	Std error	F	η^2
Post-test	Fade-in group	45	62.51	12.16	62.36	2.84	10.383^{**}	0.107
	Fade-out group	45	49.25	24.20	49.41	2.84		
Group product	Fade-in group	45	73.10	5.42	72.96	1.89	146.879^{***}	0.845
	Fade-out group	45	40.33	8.59	40.47	1.89		

$^{**}p < 0.01$, $^{***}p < 0.001$

```
175  void EatFood(struct Snake* snake)
176  {
177      int tail_x = snake->x[snake->len - 1];
178      int tail_y = snake->y[snake->len - 1];
179
180      int dx = tail_x - snake->x[snake->len - 2];
181      int dy = tail_y - snake->y[snake->len - 2];
182
183      int nx = tail_x + dx;
184      int ny = tail_y + dy;
185
186      snake->len++;
187      snake->x[snake->len - 1] = nx;
188      snake->y[snake->len - 1] = ny;
189
190      map[nx][ny] = snake->sid;
191  }
192
193  void SetTime()
194  {
195      int a = 180;
196      system("color 0a");
197      while (a >= 0) {
198          system("cls");
199          printf("%d",a);
200          sleep(1000); a=a-1;
201      }
202      system("pause");
203      printf("游戏结束\n");
204  }
205
206  bool JudgeEnd()
207  {
208      return cnt != 0;
209  }
210
211  void StartGame()
212  {
213      GameKeys();
214      InitMap();
215      InitSnake();
216      dirs[0]['2'] = 0; dirs[0]['8'] = 1; dirs[0]['4'] = 2; dirs[0]['6'] = 3;
217
```

Fig. 8 The fade-in group product

programming performance, metacognitive behaviors, emotions, and collective efficacy for novice programmers.

```
173   void EatFood()
174   {
175       if(map[snake_one.x[0]][snake_one.y[0]]==2)
176       {
177           snake_one.len++;
178           snake_one.count++;
179           map[snake_one.x[0]][snake_one.y[0]] = 1;
180       }
181
182       if(map[snake_two.x[0]][snake_two.y[0]]==2)
183       {
184           snake_two.len++;
185           snake_two.count++;
186           map[snake_two.x[0]][snake_two.y[0]] = 1;
187       }
188
189       if(map[snake_three.x[0]][snake_three.y[0]]==2)
190       {
191           snake_three.len++;
192           snake_three.count++;
193           map[snake_three.x[0]][snake_three.y[0]] = 1;
194       }
195       SetFood();
196   }
197
198   void SetTime()
199   {
200
201   }
202
203   void JudgeEnd()
204   {
205
206
207   void StartGame()
208   {
209
210   }
211
212   int win(int grade)
213   {
214
215   }
```

Fig. 9 The fade-out group product

Table 6 Adjusted residuals of the fade-in group

Starting behavior	Subsequent behavior					
	OG	MP	ES	MC	ER	AM
Orientating goals (OG)	22.10*	5.59*	−10.79	1.12	−0.78	−0.92
Making plans (MP)	1.68	18.64*	−11.78	3.72*	−1.73	−1.29
Enacting strategies (ES)	−10.88	−10.76	6.64*	2.10*	0.90	−1.88
Monitoring and controlling (MC)	3.58*	1.98*	6.22*	−7.15	−2.45	−0.42
Evaluating and reflecting (ER)	−1.54	−2.91	−4.72	4.59*	3.47*	0.83
Adapting metacognition (AM)	−0.91	−1.29	−1.88	−1.65	2.74*	17.79*

*$p < 0.05$

Table 7 Adjusted residuals of the fade-out group

Starting behavior	Subsequent behavior					
	OG	MP	ES	MC	ER	AM
Orientating goals (OG)	21.65*	1.68	−1.72	−0.10	−0.74	−0.19
Making plans (MP)	−0.44	23.63*	−8.53	1.81	−3.53	−0.92
Enacting strategies (ES)	−2.32	−9.77	10.28*	−6.20	−0.33	−1.88
Monitoring and controlling (MC)	1.15	1.98*	−4.65	5.53*	−2.30	−1.25
Evaluating and reflecting (ER)	−0.67	−1.90	−3.94	0.78	7.09*	0.13
Adapting metacognition (AM)	−0.18	−0.92	−1.88	−1.24	0.12	21.38*

*$p < 0.05$

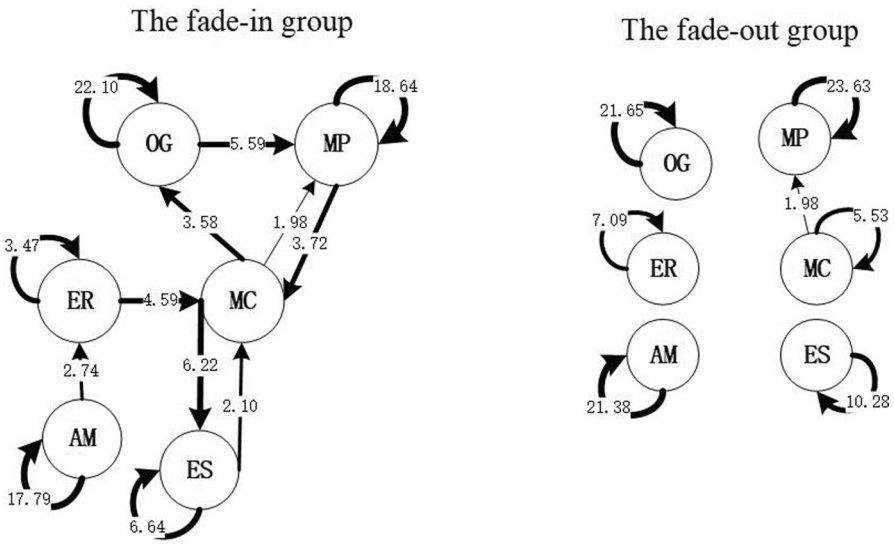

Fig. 10 Behavioral transition diagrams of the fade-in and fade-out groups

Table 8 Significant behavior sequences that occur in the fade-in and fade-out groups

Groups	Significant behavior sequences
Fade-in groups	OG→OG, OG→MP, MP→MP, MP→MC, ES→ES, ES→MC, MC→OG, MC→MP, MC→ES, ER→MC, ER→ER, AM→ER, AM→AM
Fade-out groups	OG→OG, MP→MP, ES→ES, MC→MP, MC→MC, ER→ER, AM→AM

Table 9 Independent samples t-test results of emotional status

Emotion	Group	N	Mean	SD	t	Cohen's d
Positive	Fade-in group	45	307.20	71.51	2.59*	0.94
	Fade-out group	45	236.27	77.88		
Negative	Fade-in group	45	0.60	2.32	1.92	0.54
	Fade-out group	45	2.47	4.27		
Neutral	Fade-in group	45	1.13	1.72	1.48	0.70
Confused	Fade-out group	45	3.93	5.35	0.97	0.35
	Fade-in group	45	19.47	9.15		
	Fade-out group	45	23.27	11.97		

*$p < 0.05$

Table 10 Independent samples t-test results of collective efficacy

Group	N	Mean	SD	t	Cohen's d
Fade-in group	45	3.98	0.54	3.29**	0.70
Fade-out group	45	3.54	0.70		

**$p < 0.01$

Conclusions

This exploratory study sought to examine the roles of fade-in and fade-out scaffolding for novice programmers and obtain a better understanding of the complexity of scaffolding. This study serves to demonstrate the importance of fade-in scaffolding for novice programmers. The present study revealed that the fade-in scaffolding played a very important role in helping novice programmers improve knowledge building and programming skills, foster positive emotions, and increase metacognitive behaviors and collective efficacy. This study also provides a valuable reference for research and practice in collaborative programming.

Future studies should examine the roles of fade-in and fade-out scaffolding when learners complete different programming tasks over a longer period of time.

Moreover, there is a need for further studies to investigate the roles of fade-in and fade-out scaffolding in higher-order skills, transfer skills, and cognitive load through mixed methods to obtain a deep understanding of the nature of fade-in and fade-out scaffolding. In addition, the current scaffolding approach should be extended in instructing student programming processes to contribute to the growing research community.

Acknowledgements This study is funded by the International Joint Research Project of Huiyan International College, Faculty of Education, Beijing Normal University (ICER202101).

Declarations

Conflict of interest The authors have no conflicts of interests.

Consent to participate Informed consent was obtained from all participants included in the study.

References

Bakeman, R., & Gottman, J. M. (1997). *Observing interaction: An introduction to sequential analysis*. Cambridge University Press. https://doi.org/10.1017/CBO9780511527685

Bakeman, R., & Quera, V. (2011). *Sequential analysis and observational methods for the behavioral sciences*. Cambridge University Press.

Bagozzi, R. P., & Yi, Y. (1988). On the evaluation of structural equation models. *Journal of the Academy of Marketing Science, 16*(1), 74–94. https://doi.org/10.1007/BF02723327

Bandura, A. (1997). *Self-efficacy: The exercise of control*. Freeman.

Beck, L., & Chizhik, A. (2013). Cooperative learning instructional methods for CS1: Design, implementation, and evaluation. *ACM Transactions on Computing Education, 13*(3), 10–21. https://doi.org/10.1145/2492686

Bond, M., Buntins, K., Bedenlier, S., Zawacki-Richter, O., & Kerres, M. (2020). Mapping research in student engagement and educational technology in higher education: A systematic evidence map. *International Journal of Educational Technology in Higher Education, 17*, 2. https://doi.org/10.1186/s41239-019-0176-8

Bowman, N. A., Jarratt, L., Culver, K. C., & Segre, A. M. (2020). Pair programming in perspective: Effects on persistence, achievement, and equity in computer science. *Journal of Research on Educational Effectiveness, 13*(4), 731–758. https://doi.org/10.1080/19345747.2020.1799464

Bulu, S. T., & Pedersen, S. (2010). Scaffolding middle school students' content knowledge and ill-structured problem solving in a problem-based hypermedia learning environment. *Educational Technology Research & Development, 58*(5), 507–529. https://doi.org/10.1007/s11423-010-9150-9

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Lawrence Earlbaum Associates.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education, 46*(3), 277–296. https://doi.org/10.1080/15391523.2014.888272

Exter, M. E., & Ashby, I. (2019). Preparing today's educational software developers: Voices from the field. *Journal of Computing in Higher Education, 31*(3), 472–494. https://doi.org/10.1007/s12528-018-9198-9

Faul, F., Erdfelder, E., Lang, A. G., & Buchner, A. (2007). G* Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior Research Methods, 39*(2), 175–191. https://doi.org/10.3758/BF03193146

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87–97. https://doi.org/10.1016/j.compedu.2012.11.016

Furberg, A. (2016). Teacher support in computer-supported lab work: Bridging the gap between lab experiments and students' conceptual understanding. *International Journal of Computer-Supported Collaborative Learning, 11*(1), 89–113. https://doi.org/10.1007/s11412-016-9229-3

Giacaman, N., & De Ruvo, G. (2018). Bridging theory and practice in programming lectures with active classroom programmer. *IEEE Transactions on Education, 61*(3), 177–186. https://doi.org/10.1109/TE.2018.2819969

Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2010). *Multivariate data analysis*. Prentice Hall.

Kaendler, C., Wiedmann, M., Rummel, N., & Spada, H. (2015). Teacher competencies for the implementation of collaborative learning in the classroom: A framework and research review. *Educational Psychology Review, 27*(3), 505–536. https://doi.org/10.1007/s10648-014-9288-9

Kalyuga, S., & Renkl, A. (2010). Expertise reversal effect and its instructional implications: Introduction to the special issue. *Instructional Science, 38*(3), 209–215. https://doi.org/10.1007/s11251-009-9102-0

Kapur, M. (2008). Productive failure. *Cognition and Instruction, 26*(3), 379–424. https://doi.org/10.1080/07370000802212669

Kapur, M. (2016). Examining productive failure, productive success, unproductive failure, and unproductive success in learning. *Educational Psychologist, 51*(2), 289–299. https://doi.org/10.1080/00461520.2016.1155457

Kapur, M., & Bielaczyc, K. (2012). Designing for productive failure. *Journal of the Learning Sciences, 21*(1), 45–83. https://doi.org/10.1080/10508406.2011.591717

Koorsse, M., Cilliers, C., & Calitz, A. (2015). Programming assistance tools to support the learning of IT programming in South African secondary schools. *Computers & Education, 82*, 162–178. https://doi.org/10.1016/j.compedu.2014.11.020

Kwon, K., Hong, R. Y., & Laffey, J. M. (2013). The educational impact of metacognitive group coordination in computer-supported collaborative learning. *Computers in Human Behavior, 29*(4), 1271–1281. https://doi.org/10.1016/j.chb.2013.01.003

Kunkle, W. M., & Allen, R. B. (2016). The impact of different teaching approaches and languages on student learning of introductory programming concepts. *ACM Transactions on Computing Education, 16*(1), 1–26. https://doi.org/10.1145/2785807

Lajoie, S. P. (2005). Extending the scaffolding metaphor. *Instructional Science, 33*(5–6), 541–557. https://doi.org/10.1007/s11251-005-1279-2

Lu, O. H., Huang, J. C., Huang, A. Y., & Yang, S. J. (2017). Applying learning analytics for improving students engagement and learning outcomes in an MOOCs enabled collaborative programming course. *Interactive Learning Environments, 25*(2), 220–234. https://doi.org/10.1080/10494820.2016.1278391

Janssen, N., Knoef, M., & Lazonder, A. W. (2019). Technological and pedagogical support for preservice teachers' lesson planning. *Technology, Pedagogy and Education, 28*(1), 115–128. https://doi.org/10.1080/1475939X.2019.1569554

Jennings, J., & Muldner, K. (2020). Assistance that fades in improves learning better than assistance that fades out. *Instructional Science, 48*(4), 371–394. https://doi.org/10.1007/s11251-020-09520-7

Margulieux, L. E., & Catrambone, R. (2021). Scaffolding problem solving with learners' own self explanations of subgoals. *Journal of Computing in Higher Education, 33*, 499–523. https://doi.org/10.1007/s12528-021-09275-1

Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). PlayIT: Game based learning approach for teaching programming concepts. *Educational Technology & Society, 19*(2), 5–17.

Menon, P., & Kovalchick, L. (2020). Using a concept map to represent the composition of knowledge in an introductory programming course. *Information Systems Education Journal, 18*(3), 4–17.

Mladenović, S., Krpan, D., & Mladenović, M. (2016). Using games to help novices embrace programming: From elementary to higher education. *The International Journal of Engineering Education, 32*(1), 521–531.

Mohd Rum, S. N., & Ismail, M. A. (2017). Metacognitive support accelerates computer assisted learning for novice programmers. *Educational Technology & Society, 20*(3), 170–181.

Molenaar, I., Sleegers, P., & van Boxtel, C. (2014). Metacognitive scaffolding during collaborative learning: A promising combination. *Metacognition and Learning, 9*(3), 309–332. https://doi.org/10.1007/s11409-014-9118-y

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM, 41*(3), 105–108. https://doi.org/10.1145/272287.272333

Phuong, D. T. D., & Shimakawa, H. (2008). Collaborative learning environment to improve novice programmer with convincing opinions. *WSEAS Transactions on Advances in Engineering Education, 5*(9), 635–644.

Puntambekar, S., & Hubscher, R. (2005). Tools for scaffolding students in a complex learning environment: What have we gained and what have we missed? *Educational Psychologist, 40*(1), 1–12. https://doi.org/10.1207/s15326985ep4001_1

Quera, V., Bakeman, R., & Gnisci, A. (2007). Observer agreement for event sequences: Methods and software for sequence alignment and reliability estimates. *Behavior Research Methods, 39*(1), 39–49. https://doi.org/10.3758/BF03192842

Sentance, S., & Csizmadia, A. (2016). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies, 22*(2), 469–495. https://doi.org/10.1007/s10639-016-9482-0

Shin, Y., Kim, D., & Song, D. (2020). Types and timing of scaffolding to promote meaningful peer interaction and increase learning performance in computer-supported collaborative learning environments. *Journal of Educational Computing Research, 58*(3), 640–661. https://doi.org/10.1177/0735633119877134

Sun, J. C. Y., & Hsu, K. Y. C. (2019). A smart eye-tracking feedback scaffolding approach to improving students' learning self-efficacy and performance in a C programming course. *Computers in Human Behavior, 95*, 66–72. https://doi.org/10.1016/j.chb.2019.01.036

Sung, H. Y., & Hwang, G. J. (2013). A collaborative game-based learning approach to improving students' learning performance in science courses. *Computers & Education, 63*, 43–51. https://doi.org/10.1016/j.compedu.2012.11.019

Tan, H. Q. (2017). *Programming in C* (5th ed.). Tsinghua University Press.

Tansomboon, C., Gerard, L. F., Vitale, J. M., & Linn, M. C. (2017). Designing automated guidance to promote productive revision of science explanations. *International Journal of Artificial Intelligence in Education, 27*(4), 729–757. https://doi.org/10.1007/s40593-017-0145-0

Tawfik, A. A., Law, V., Ge, X., Xing, W., & Kim, K. (2018). The effect of sustained vs. faded scaffolding on students' argumentation in ill-structured problem solving. *Computers in Human Behavior, 87*, 436–449. https://doi.org/10.1016/j.chb.2018.01.035

Teague, D., & Roe, P. (2008). Collaborative learning-towards a solution for novice programmers. In *Proceedings of the tenth Australasian computing education conference in conferences in research and practice in information technology-CRPIT volume 78* (pp. 147–153). Australian Computer Society. https://eprints.qut.edu.au/17818/1/c17818.pdf

Tsai, C. Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior, 95*, 224–232. https://doi.org/10.1016/j.chb.2018.11.038

Tullis, J., Goldstone, R., & Hanson, A. (2015). Scheduling scaffolding: The extent and arrangement of assistance during training impacts test performance. *Journal of Motor Behavior, 47*, 1–11. https://doi.org/10.1080/00222895.2015.1008686

van Leeuwen, A., & Janssen, J. (2019). A systematic review of teacher guidance during collaborative learning in primary and secondary education. *Educational Research Review, 27*, 71–89. https://doi.org/10.1016/j.edurev.2019.02.001

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.

Wang, X. C., Choi, Y., Benson, K., Eggleston, C., & Weber, D. (2021). Teacher's role in fostering preschoolers' computational thinking: An exploratory case study. *Early Education and Development, 36*(1), 26–48. https://doi.org/10.1080/10409289.2020.1759012

Wang, X. M., & Hwang, G. J. (2017). A problem posing-based practicing strategy for facilitating students' computer programming skills in the team-based learning mode. *Educational Technology Research and Development, 65*(6), 1655–1671. https://doi.org/10.1007/s11423-017-9551-0

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

Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry, 17*(2), 89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x

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

Wu, L., Looi, C. K., Multisilta, J., How, M. L., Choi, H., Hsu, T. C., & Tuomi, P. (2020). Teacher's perceptions and readiness to teach coding skills: A comparative study between Finland, Mainland China, Singapore, Taiwan, and South Korea. *The Asia-Pacific Education Researcher, 29*(1), 21–34. https://doi.org/10.1007/s40299-019-00485-x

Xu, B., Chen, N. S., & Chen, G. (2020). Effects of teacher role on student engagement in WeChat-Based online discussion learning. *Computers & Education*. https://doi.org/10.1016/j.compedu.2020.103956

Yaghmazadeh, N., Wang, X., & Dillig, I. (2018). Automated migration of hierarchical data to relational tables using programming-by-example. *Proceedings of the VLDB Endowment, 11*(5), 580–593.

Yeomans, L., Zschaler, S., & Coate, K. (2019). Transformative and troublesome? Students' and professional programmers' perspectives on difficult concepts in programming. *ACM Transactions on Computing Education, 19*(3), 1–27. https://doi.org/10.1145/3283071

Yilmaz, F. G. K., & Yilmaz, R. (2020). Student opinions about personalized recommendation and feedback based on learning analytics. *Technology, Knowledge and Learning, 25*, 753–768. https://doi.org/10.1007/s10758-020-09460-8

Yu, S., & Hu, G. (2017). Can higher-proficiency L2 learners benefit from working with lower-proficiency partners in peer feedback? *Teaching in Higher Education, 22*(2), 178–192. https://doi.org/10.1080/13562517.2016.1221806

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*. https://doi.org/10.1080/10494820.2021.1943687

Zheng, L. (2017). *Knowledge building and regulation in computer-supported collaborative learning*. Springer.

Zheng, L., & Huang, R. (2016). The effects of sentiments and co-regulation on group performance in computer-supported collaborative learning. *The Internet and Higher Education, 28*, 59–67. https://doi.org/10.1016/j.iheduc.2015.10.001

Zheng, L., Huang, R., Hwang, G.-J., & Yang, K. (2015). Measuring knowledge elaboration based on a computer-assisted knowledge map analytical approach to collaborative learning. *Educational Technology & Society, 18*(1), 321–336.

Zheng, L., Li, X., Zhang, X., & Sun, W. (2019). The effects of group metacognitive scaffolding on group metacognitive behaviors, group performance, and cognitive load in computer-supported collaborative learning. *The Internet and Higher Education, 42*, 13–24. https://doi.org/10.1016/j.iheduc.2019.03.002

Lanqin Zheng currently works as an associate professor at the Faculty of Education in Beijing Normal University. Her research interests include computer supported collaborative learning, learning analytics, personalized learning, and AIED.

Yuanyi Zhen received the M.S. degree in educational technology from the Faculty of Education in Beijing Normal University.

Jiayu Niu is a master student at the Faculty of Education in Beijing Normal University. Her research interests focus on computer supported collaborative learning.

Lu Zhong is a master student at the Faculty of Education in Beijing Normal University. Her research interests focus on computer supported collaborative learning.