Investigating the Period of Switching Roles in Pair Programming in a Primary School

Baichang Zhong[1*], Qiyun Wang[2], Jie Chen[3] and Yi Li[1]

[1]Collaborative Innovation Center for Talent Cultivating Mode in Basic Education, Nanjing Normal University, China // [2]National Institute of Education, Nanyang Technological University, Singapore // [3]Bixi Primary School of Changshu City, China // zhongbc@163.com // qiyun.wang@nie.edu.sg // yunizi19@163.com // yilisd@163.com

[*]Corresponding author

(Submitted April 1, 2016; Revised November 30, 2016; Accepted February 1, 2017)

ABSTRACT

Pair programming (PP) is a useful approach to fostering computational thinking for young students. However, there are many factors impacting on the effectiveness of PP. The period of switching roles between the driver and the navigator is often ignored by researchers. Therefore, this study aimed to explore the impact of the switching period on PP. We conducted a PP experiment in four classes in the sixth grade in a primary school. The results indicated that (a) the semi-free switch was more effective for the learning achievement than the fixed periods, and the preference for adopting the fixed time interval to switch roles existed in previous studies seems to be a kind of prejudice; (b) students who switched roles every 5 minutes and semi-freely were more enjoyable than those who switched roles in every task and in every class session. Moreover, the period of switching roles in every class session decreased students' enjoyment after PP; (c) the frequency of switching roles decreased significantly, but the negotiation between the driver and the navigator became more active with time going in the semi-free class. Implications for teaching are also discussed.

Keywords

Collaborative learning, Primary education, Programmed learning, Pair programming

Introduction

Programming for K-12 can be traced back to the 1960s when Logo programming was firstly introduced as an intellectual thinking educational tool for teaching mathematics (Feurzeig, Papert, & Lawler, 2011). After Logo, the use of programming to teach thinking skills in K-12 was scarcely reported. In recent years, however, there has been renewed interest in introducing programming to K-12 students (Grover & Pea, 2013; Kafai & Burke, 2013). This was aroused by the availability of easy-to-use visual programming languages such as Scratch, Stagecast Creator and Alice, etc.

During programming, students are exposed to computational thinking (CT), a term popularized by Wing (2006). CT involves solving problems, designing systems, and understanding human behaviors, by drawing on the concepts fundamental to computer science (Wing, 2006). Many researchers argue that CT is a fundamental skill for almost everyone in a digital age, not just for computer scientists (National Research Council, 2010; Wing, 2006). More importantly, CT is in line with many 21st century competencies such as creativity, critical thinking, and problem solving (Binkley et al., 2012). Thus, it is not surprising that many educators claim that programming provides an important context and a set of opportunities for K-12 students to develop CT (Kafai & Burke, 2013; Lye & Koh, 2014; Resnick et al., 2009).

This revived interest in programming in K-12 settings suggests a need to consider how CT can be fostered effectively via programming. Studies have showed that students taught with pair programming (PP) often perform better in CT than with solo programming (Lye & Koh, 2014; Werner & Denning, 2009; Werner, Denner, Campe, & Kawamoto, 2012). PP is a practice in which two people work side-by-side at one computer, and closely collaborate to create a program. One is normally called the "driver," who is responsible for using a computer to key in codes. The other is usually known as the "navigator" or "observer/reviewer," who takes the responsibility for observing the driver's work and providing support by pointing errors or offering ideas in solving a problem (Williams & Kessler, 2000).

In view of the usefulness of fostering CT, we have used PP as a pedagogical teaching technique in a primary school for two years. Meanwhile, we have also identified some issues with putting PP into practice. One main issue is about how often the roles (driver and navigator) in a pair should switch from one to the other, since it is very important to switch roles periodically between the driver and the navigator (Williams & Kessler, 2002). In other words, what period should we choose to switch the students' roles in PP practice?

Literature review

Many studies have showed that PP has obvious benefits over solo programming, including PP can (1) significantly improve individual programming skills and promote productivity or program quality (Braught, Eby, & Wahls, 2008; Cliburn, 2003; Hannay, Dybå, Arisholm, & Sjøberg, 2009; Williams & Kessler, 2000); (2) reduce frustration experienced by novice programmers; increase student satisfaction, enjoyment; and foster positive attitudes in programming (Bishop-Clark, Courte, Evans, & Howard, 2006; DeClue, 2003; LeJeune, 2006; McDowell, Werner, Bullock, & Fernald, 2002; Preston, 2005; Werner, Bullock, & Fernald, 2006); (3) increase retention of students (especially for female students) in computer science courses (Li, Plaue, & Kraemer, 2013; McDowell et al., 2006); and (4) better prepare students to work as a team (Cliburn, 2003; Williams & Kessler, 2000).

However, the above benefits do not occur automatically. Some experiments and empirical studies have reported inconclusive or contradictory results (Balijepally, Mahapatra, Nerur, & Price, 2009; Sfetsos, Stamelos, Angelis, & Deligiannis, 2009). This accentuates the need for further studies. Some factors have been identified that influence the effects of PP include:

- Task complexity (Arisholm, Gallis, Dybă, & Sjoberg, 2007; Hannay, Arisholm, Engvik, & Sjøberg, 2010);
- Partners' skills and experiences (Hannay et al., 2010; Lui & Chan, 2006);
- Partners' learning styles (Salleh, Mendes, & Grundy, 2011; Williams et al., 2006); and
- Partners' personalities and temperaments (Hannay et al., 2010; Katira et al., 2004; Sfetsos et al., 2009).
- Social factors (Choi, 2015; Lewis, 2011; Zhong, Wang, & Chen, 2016)

We found that most of these empirical studies were concerned with space factors (task types and pair formations) only, but largely ignoring the time factor (period of switching roles). This one-sided consideration illustrated an incomplete picture of PP.

Many studies conducted in primary schools and higher education environments just described the period of switching roles adopted, but did not state reasons behind, and excluded it from the experiments' variables system. For examples, Bevan, Werner, and McDowell (2002) conducted a PP experiment in a freshman programming class at UCSC, in which students alternated between driving and navigating at intervals of no more than 1 hour. Lewis (2011) conducted a study to investigate differences between PP and collaborative learning in two summer enrichment classes for students entering the sixth grade, in which the students switched their roles every 5 minutes. Mendes, Al-Fakhri, and Luxton-Reilly (2005) conducted a PP experiment at the University of Auckland (NZ) involved 300 second year computer science students. During each of the 90-minute lab sessions pairs swapped roles every 20 minutes, reminded by teaching assistants. Salleh, Mendes, Grundy, and Burch (2010) conducted a formal experiment at the University of Auckland to investigate the influence of personality differences among paired students. Students worked with their partners for an initial period of 30 minutes; and then swapped roles every 15-20 minutes. Salleh, Mendes, and Grundy (2014) conducted five formal experiments at the University of Auckland to investigate the effects of personality composition on PP's effectiveness, where students worked with their partners for an initial period of 30 minutes.

Some studies took place in commercial and industrial environments with professional programmers in a flexible time interval, and investigated the frequency of switching roles. Rostaher and Hericko (2002) reported that the switching frequency correlated with programming experience and that more experienced pairs switched more often. Chong and Hurlbutt (2007) concurred that frequent switch helped the developers to maintain a high level of mutual awareness of each other's action. Plonka, Segal, Sharp, and Linden (2011) also found that most pairs switched roles frequently and that the frequency and fluidity of switching roles indicated a high level of engagement on both developers.

These studies provide insights into the effects of switching, which indicate the switching frequency (namely, the period of switching roles) could impact on PP and should be investigated in more detail. We hypothesize that different periods of switching roles may lead to different results. Therefore, this paper reports on an experimental investigation of the time factor on PP with an expectation of generating better effectiveness in programming.

Research questions

This study aimed firstly to explore the impact of the switching period on PP effectiveness, measured by the learning achievement in PP, and learning attitude including confidence, enjoyment and value toward

programming. Our research purpose was to compare the learning achievement and attitude in different periods of switching roles. Therefore, our first two research questions were:
RQ1: Does the learning achievement vary significantly in various periods of switching roles?

RQ2: Does the learning attitude including programming confidence, enjoyment and value vary significantly in various periods of switching roles?

The second aim of this study was to compare the frequency of switching roles among three learning stages (theme 1, theme 2, and theme 3) in the semi-free switching class, and to generate implications for teaching. Our third research question was:

RQ3: Does the frequency of switching roles increase with the growth of PP time and experience?

Participants

Participants were sixth grade pupils coming from a primary school in China (it is called "school C" in this paper). We selected randomly 4 of 8 classes in the sixth grade and 150 pupils to participate in the experiment (see Table 1). These students took the Scratch introductory course in the 2014 autumn semester and Scratch advanced course in the 2015 spring semester before this study, and had basic knowledge and skills about programming. We conducted a prior knowledge quiz before this experiment, and the result of a one-way ANOVA showed that there was no significant difference among the four classes on the basic programming knowledge and skills, $F(3, 146) = .63$, $p > .05$.

We designed four various periods for Classes A, B, C, and D, respectively:

- Class A rotated roles every 5 minutes. Pairs must switch their roles every 5 minutes during PP.
- Class B rotated roles in every class session. Pairs switched their roles at the beginning of a class session which has about 40 minutes, and 20 to 30 minutes for PP.
- Class C rotated roles in every task. Pairs switched roles once in every two consecutive tasks. Each task needed about 5 to 15 minutes for PP.
- Class D rotated roles in a semi-free form. Pairs switched their roles according to their own needs, and they must switch roles at least once in a class session.

Table 1. Sample characteristics

Class	Period of switching	Class size	Boys	Girls
A	Every 5 minutes	34	14	20
B	Every class session	38	18	20
C	Every task	42	21	21
D	Semi-free	36	15	21
Total	—	150	68	82

Procedure

We conducted a PP teaching experiment in the 2015 autumn semester. The experiment lasted for 13 weeks in all, including 4 weeks for learning basic knowledge and skills of Alice programming, 6 weeks for formal experimental treatment. Also there were a quiz and a pretest at the beginning of the experiment and 3 achievement tests and an attitude survey at the end of the experiment (see Table 2). Every class had an additional session to learn programming each week taught by the same teacher.

Table 2. Schedule of the experiment

Stage	Week	Content
Pre-experiment	1st	Prior knowledge quiz and attitude survey, grouping students and PP guidance.
	2nd	My amusement park (A) (Create new scene, add and delete objects, and save file in Alice programming tool)
	3rd	My amusement park (B) (Modify the appearance and position of objects via keyboard and simple visual codes, respectively)
	4th	My amusement park (C) (Freely design new scenario and animation for my amusement park.)

Experimental treatment	5th 6th 7th 8th 9th 10th	**Theme 1:** skating girl (Control the girl skating with different methods via event or keyboard.) **Theme 2:** exploring the universe (Help the robot move to a big rock, and add new scenario and animation for it when an extraterrestrial is discovered.) **Theme 3:** confused kangaroo (Help the kangaroo to jump smoothly, and add new scenario and animation for it.)
Post-experiment	11th 12th 13th	Achievement test 1 and Attitude survey Achievement test 2 Achievement test 3

In the first week, the participants took part in a prior knowledge quiz to confirm that the four classes had the same level of basic knowledge and skills about programming. We also conducted an attitude survey about programming as a pretest. The participants were given a brief introduction to PP terminology and guidance prior to the Alice programming course.

In the 2nd to 4th weeks, the participants learned the basic knowledge and skills about how to programming in Alice.

In the 5th to 10th weeks, the formal PP was implemented. The participants learned to complete three programming themes in a way of PP. During the PP process, each pair exchanged the role of being a driver or being a navigator according to Table 1. Meanwhile, the teacher also used a three-stage learning progression model called Use-Modify-Create to describe a pattern of engagement (Lee et al., 2011).

In the 11th to 13th weeks, we conducted the posttests including three achievement tests and one attitude survey.

Materials

Alice

The 3D programming language Alice2.4 developed by Carnegie Mellon was used in this curriculum. Alice (http://www.alice.org/) is an easy-to-learn environment which allows users to build 3D virtual worlds.

E-textbook

For this experiment, we developed a school-based curriculum “learning to storytelling by programming” based on the three-dimension framework of computational thinking (Zhong, Wang, & Chen, 2016). For the convenience of improvement, the curriculum was developed into an e-textbook via 3DPageFlip (see Figure 1).

Figure 1. A snapshot of the e-textbook

Handbook for PP

For effectively switching roles in PP, we designed a student handbook, which included pair information, PP guidance, a sheet for the progress of PP (see Table 3), and a task design table. The handbook was delivered before each class session and collected back at the end of each class session. The teacher checked the handbook carefully each time, and approached the students individually if they did not fill the handbook in detail.

Table 3. Sheet for the progress of PP

Switch[a]	Roles	Times (Minutes)	Task Completion	Who Initiated[b]
1	Driver			
	Navigator			
2	Driver			
	Navigator			
3	Driver			
	Navigator			
4	Driver			
	Navigator			
5	Driver			
	Navigator			
6	Driver			
	Navigator			

Note. This sheet must be filled in class by the navigator and calculated for a theme (a duration of two class sessions). [a]There were at least 10 minutes for teaching, and a maximum of 6 switches in a class session. [b]Filling the table cell with "driver," "navigator," or "consensus." The "consensus" means the switch was not individually initiated by the drive or the navigator, but reached a consensus through negotiation between the driver and the navigator.

Measures

A single-factor experiment was employed to examine the impacts of the switching period on PP (see Figure 2). The instruments for the pretest and posttest are shown in Table 4. All students completed the pretest and posttest individually.

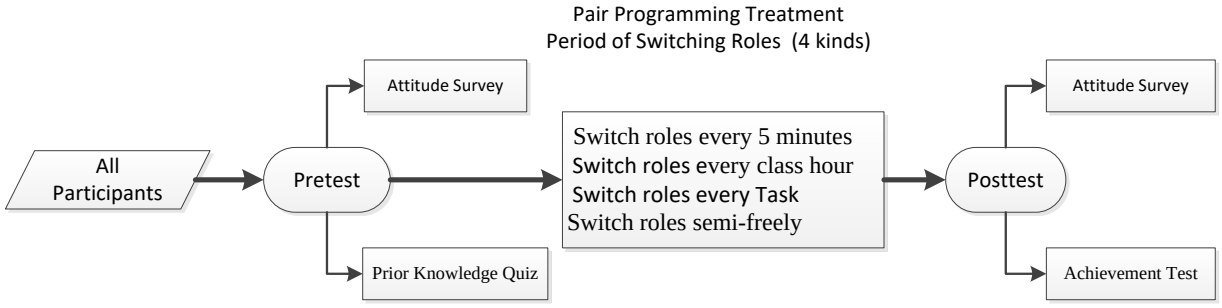

Figure 2. Overview of experimental design

Table 4. Instruments for data collection

Construct		Source	Reliability*	
			Pretest	Posttest
Prior Knowledge (11 questions)		Designed by ourselves	.73	—
Attitude	Confidence (5 questions)	Bishop-Clark et al., 2006	.78	.84
	Enjoyment (5 questions)	Bishop-Clark et al., 2006	.75	.85
	Value (4 questions)	Designed by ourselves	.76	.84
Achievement (5 tasks)		Zhong et al., 2015	—	.86

Note. *The Cronbach's Alpha was used to check the reliability of Attitude Test and Achievement Test, the Kuder-Richardson 20 (KR-20), however, was used in Prior Knowledge Test since it was a 0-1 academic success test.

Prior knowledge quiz

The prior knowledge quiz included 11 multiple choice questions was used to measure the students' understanding of programming concepts, since they all had taken two Scratch courses before the experiment. For instance, one question read "In order to write one piece of code to draw a square with different length of side, we can set the length of side as a: (a) variable, (b) constant, (c) parameter, (d) instruction." The mean scores of the quiz in the four classes had no significant difference. The validity of the quiz was determined by experts' evaluation, and the reliability coefficient of the quiz was .72.

Attitude survey

The attitude survey included three dimensions to assess participants' confidence, enjoyment, and value toward programing. A five-point Likert scale was used, ranging from 5 = strongly agree to 1 = strongly disagree. Table 5 shows the questions used for the confidence dimension, enjoyment dimension, and value dimension.

Table 5. Attitude measures

Type	Task
Confidence (5 questions)	Generally I have felt secure about computer programming.
	I am sure that I could learn programming.
	I have a lot of self-confidence when it comes to programming.
	I am not good at programming.
	I am not the type to do well at programming.
Enjoyment (5 questions)	I like writing computer programs.
	Programming is enjoyable and stimulating.
	Once I start trying to work on a program, I find it hard to stop.
	The challenge of programming problems does not appeal to me.
	Programming is boring.
Value (4 questions)	Programming may improve my computer skills.
	Programming is helpful for studying other subjects' knowledge.
	Programming makes me work in a logical and rational way.
	Programming is helpful for solving the problems in life.

Achievement test

The achievement test was conducted after the PP course to assess the participants' programming knowledge and capability. Each participant finished the achievement test individually. The test tasks consisted of five tasks adopted from a previous study by Zhong, Wang, Chen, and Li (2015) and covered two closed tasks with a defined outcome and a defined process solely, two semi-open tasks with a defined outcome and a undefined process, and one open task with an open outcome and an open process (see Table 6).

Tasks 1 to 4 had the same story context: Two hungry rabbits, one is big, the other is small. They come to a beautiful garden. They see the cauliflowers in the garden . . . (see Figure 3).

Tasks 5 had a subsequent story context: Two hungry rabbits, one is big, the other is small. They come to a beautiful garden with cauliflowers. The small rabbit eats a cauliflower and becomes weaker because she is poisoned. She cries for help . . . (see Figure 4).

Table 6. Introduction to tasks 1 to 5

Type	Task
Closed task	Task1: Make the small rabbit eats off a green cauliflower.
	Task2: The small rabbit does not stop when it eats the cauliflower, please correct it.
Semi-open task	Task3: Make the big rabbit jumps to the front of red cauliflowers.
	Task4: The big rabbit moves to the front of red cauliflowers directly without jumping and swing, please correct it.
Open task	Task 5: Design a scenario to describe what is probably happened after the small rabbit became smaller, and need to fill out a creative design report before working.

The closed tasks and semi-open tasks were worth 5 points each, and the open task was worth 20 points. All together there were 40 points in total. The validity of the test was determined by experts, and the reliability coefficient of the test was .86.

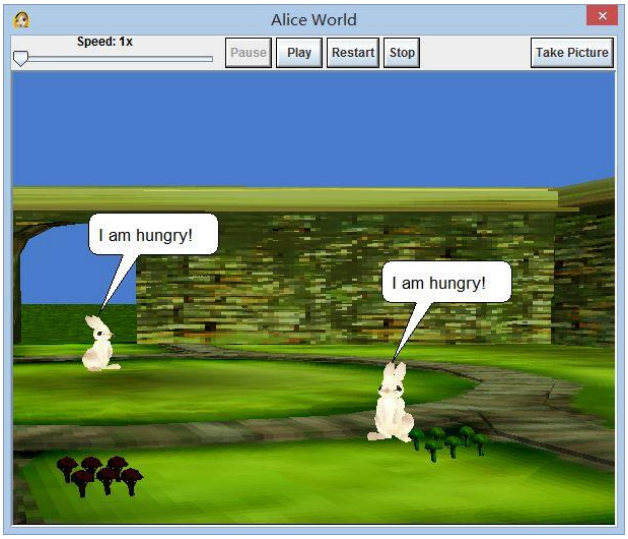

Figure 3. Scenario of rabbits' garden for tasks 1 to 4

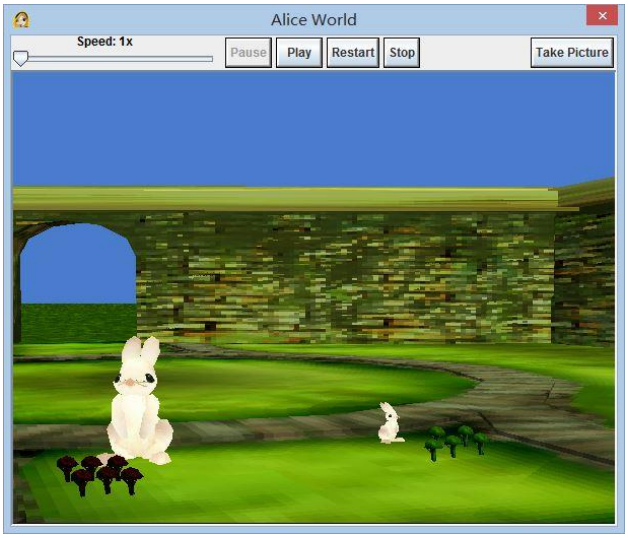

Figure 4. Scenario of rabbits' garden for task 5

All participants were also asked to provide demographic data including gender, class, and group number. The participants' responses were coded to uniquely identify them so that the data could be further compared.

Results

Achievement of programming

The first research question was about whether there was variation in the learning achievement in PP among the different periods of switching roles. To do so, we conducted a one-way ANOVA on the learning achievement, with period as a between-groups factor.

The period of switching roles had an impact on the achievement of student programming. The one-way ANOVA demonstrated that these differences were statistically reliable, $F(3, 146) = 2.97$, $p < .05$, $\eta^2 = .086$ (see Table 7). The post hoc comparisons showed that Class D who switched roles in a semi-free form got higher scores (M = 30.22) than Class A (M = 25.56), Class C (M = 26.71) and Class B (M = 27.26) in the achievement test (see Table 8).

Table 7. Summary for One-Way ANOVA on achievement

Source	*SS*	*df*	*MS*	*F*	*p*	η^2
Between-Subjects	422.29	3	140.76	2.97	.034*	.086
Within-Subjects	6916.54	146	47.37			
Total	7338.83	149				

Note. * $p < .05$.

Table 8. SNK Post Hoc Test on achievement

Class (period of switching roles)	*N*	Subset*	
		1	2
Class A (Every 5 minutes)	34	25.56	
Class C (Every task)	42	26.71	
Class B (Every class session)	38	27.26	
Class D (semi-free)	36		30.22
Sig.		.535	.074

Note. * $p < .05$.

Attitude of programming

The second research question was concerned with whether there was a variation in the learning attitude of programming among the different periods of switching roles. To investigate whether the learning attitude varied, we conducted a multivariate analysis of variance (MANOVA) with three dimensions of learning attitude as dependent variables and the period of switching roles as a between-subjects factor. Table 9 presents the mean scores of attitudes per period.

Table 9. Mean scores of the three attitudes in four periods of switching roles

Class (period of switching roles)	Confidence	Enjoyment	Value	*N*
Class B (Every class session)	19.31(.54)	18.95(.61)	19.90(.55)	38
Class C (Every task)	19.11(.52)	19.90(.58)	19.91(.52)	42
Class D (semi-free)	19.91(.56)	20.75(.62)	20.19(.56)	36
Class A (Every 5 minutes)	20.29(.57)	21.97(.64)	21.41(.58)	34
Whole sample	19.63(3.34)	20.33(3.86)	20.31(3.40)	150

Note. Table entries include means and (standard deviations).

Tending to the predicted direction, the results of MANOVA, Wilks' $\Lambda = .989$, $p < .05$, showed that periods of switching roles resulted with difference in some dimensions of attitude.

To find the difference in detail, we conducted a one-way ANOVA for each dimension of attitude. The results determined that only the mean of enjoyment was significantly different, $F(3, 146) = 4.26$, $p < .01$, $\eta^2 = .093$ (see Table 10). The post hoc comparison with SNK shows that Class A ($M = 21.97$) and Class D ($M = 20.75$) got higher scores than Class C ($M = 19.90$) and Class B ($M = 18.95$) in the dimension of enjoyment (see Table 11).

Table 10. Summary for one-way ANOVA on the three attitudes

Source	Dependent variable	*SS*	*df*	*MS*	*F*	*p*	η^2
Between-Subjects	Confidence	32.67	3	10.89	.974	.407	.020
	Enjoyment	178.10	3	59.37	4.26	.006**	.093
	Value	55.20	3	18.40	1.61	.189	.032
Within-Subjects	Confidence	1632.42	146	11.18			
	Enjoyment	2037.23	146	13.95			
	Value	1665.07	146	11.41			
Total	Confidence	1665.09	149				
	Enjoyment	2215.33	149				
	Value	1720.27	149				

Note. ** $p < .01$.

The reasons why the switch in a semi-free model was more enjoyable than others are rather obvious, since students' right of learning freedom was respected. But why was the short period of 5 minutes still enjoyable? We

interviewed the students in Class A who switched roles every 5 minutes. Many students indicated that the switch was fun, as it looked like a game of carousel pattern, or seesaw.

Table 11. SNK Post Hoc Test for enjoyment (posttest)

Class (period of switching roles)	*N*	Subset*	
		1	2
Class B (Every class session)	38	18.95	
Class C (Every task)	42	19.90	
Class D (semi-free)	36		20.75
Class A (Every 5 minutes)	34		21.97
Sig.		.097	.160

Note. $^{*}p < .05$.

We conducted a supplementary paired-samples T test to reveal the difference of attitude between the pretest and posttest in the four classes. Table 12 shows, overall, the programming confidence, enjoyment and value became significantly more positive after PP than before. Interestingly, however, students who switched roles every class session actually had a decreased enjoyment level after PP (M = 18.95) than before (M = 19.32), which further indicated that the period of switching roles every class session was less enjoyable than others.

Table 12. Difference of attitude between pretest and posttest in the four classes

Attitude	Class	Test	*N*	Mean	*SD*	*t*	*Sig.*	η^2
Confidence	A	Pretest	34	17.24	3.25	-3.78	.001**	.202
		Posttest	34	20.29	2.92			
	B	Pretest	38	17.61	3.37	-2.19	.035*	.059
		Posttest	38	19.32	3.57			
	C	Pretest	42	17.02	3.33	-2.85	.007**	.086
		Posttest	42	19.12	3.56			
	D	Pretest	36	16.44	3.53	-3.96	.000***	.214
		Posttest	36	19.92	3.21			
Enjoyment	A	Pretest	34	19.18	3.17	-3.66	.001**	.175
		Posttest	34	21.97	2.98			
	B	Pretest	38	19.32	3.53	.389	.70	.002
		Posttest	38	18.95	4.63			
	C	Pretest	42	18.45	2.53	-2.42	.020*	.060
		Posttest	42	19.90	3.25			
	D	Pretest	36	16.92	4.75	-4.10	.000***	.168
		Posttest	36	20.75	3.85			
Value	A	Pretest	34	18.74	2.54	-3.90	.000***	.225
		Posttest	34	21.41	2.50			
	B	Pretest	38	17.26	3.18	-3.23	.003**	.119
		Posttest	38	19.89	4.03			
	C	Pretest	42	16.55	3.05	-4.763	.000***	.224
		Posttest	42	19.90	3.28			
	D	Pretest	36	16.50	3.68	-4.27	.000***	.216
		Posttest	36	20.19	3.46			

Note. $^{*}p < .05$; $^{**}p < .01$; $^{***}p < .001$.

Frequency of switching roles

The third research question regarded whether the frequency of switching roles increased with the growth of PP time and experience in the semi-free class. To explore it, we calculated the Class D's switching frequency in each theme of PP. Moreover, we calculated respectively the frequency initiated by the driver, navigator or in a consensus basis according to the records in Table 4. Certainly, we were very concerned about the switching frequency initiated by consensus and its ratio to whole, since the consensus through negotiation indicated a high level of compatibility and experience of PP.

The result (Figure 5), surprisingly, shows that the frequency of switching roles decreased significantly followed by the variations of themes, which meant more experienced pairs switched less. This decrease was confirmed by

classroom observation. Some of the dialogues inferred that the switches were initiated in a simple and crude pattern at the beginning of PP. Those dialogues read, “You do it for a long time, it’s my turn.” “I can do it, let me try!” “You cannot correct it, but I can.” “I said that we should do like this, you wait and see.” Obviously, these dialogues aimed to initiate switch quickly by blaming. However, we heard more discussion in the third theme like “Take it easy, try not to …” “Maybe you can try it again.” “You may correct the error by changing …” which was essentially a type of dialogue called problem-solving oriented, but not switching-roles oriented.

In addition, the switching frequency initiated in a consensus basis and its ratio to whole also somewhat confirmed this decrease. There were more switches initiated by consensus, followed by the variations of themes (Theme 1: 42%, Theme 2: 50%, and Theme 3: 58%).

In conclusion, the switches of driver/navigator roles were reduced with the progress of the PP course. Nevertheless, the partners’ negotiation became more active than before.

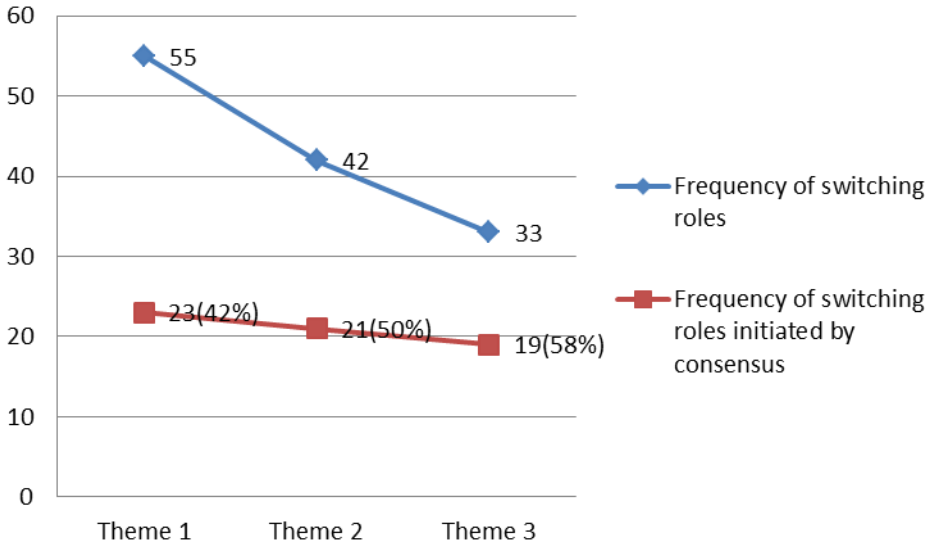

Figure 5. Frequency of switching roles in three themes

Discussion

The period of switching roles in many PP studies generally was a fixed time interval, such as 5 minutes (Lewis, 2011), 20 minutes (Mendes et al., 2005), 15-20 minutes (Salleh et al., 2010), and 30 minutes (Salleh et al., 2014). These studies had a pre-hypothesis that the fixed time interval was the best option. There were, however, no evidence to confirm it in those studies. On the contrary, this study showed that students switching roles in a semi-free form got higher achievement than the fixed time interval from 5 to 30 minutes. The result indicated that the self-directed switch under proper pressure was more effective for the learning achievement in PP. A main reason for this result is that the right of learning freedom was respected in the semi-free switching model.

Regarding the attitude of student programming, overall, the result of this study is consistent with other studies that PP could reduce frustration experienced, increase student enjoyment, and foster positive attitudes towards programming (Bishop-Clark et al., 2006; DeClue, 2003; LeJeune, 2006; McDowell et al., 2002; McDowell et al., 2006; Preston, 2005). However, the period of switching roles every class session actually decreased students’ enjoyment of programming. On the contrary, students who switched roles every 5 minutes or semi-freely were more enjoyable than those who switched roles in every task or in every class session. Some studies conducted in commercial and industrial environments found that more experienced pairs switched more often because frequent switches indicated a high level of engagement of both programmers (Chong & Hurlbutt, 2007; Plonka, et al., 2011; Rostaher & Hericko, 2002). The result of those studies seems plausible, but the result of this study contracts that the frequency of switching roles decreased significantly with time going in the semi-free class. This difference was possibly caused by two factors as follows.

On the one hand, the participants were different. In this study, the participants were pupils who learned to program, but not adults who worked by programming. This difference led to a rigid and awkward style of switching roles indicated in the partners’ dialogues above. As a result, partners, especially the navigators, were

fussed about having an equal opportunity to operate the mouse and keyboard at the beginning of PP. Subsequently, they knew that the goal of PP was to work together to resolve programming problems and when they should switch roles with the progress of the PP course. Obviously, the skilled professional programmers would not switch their roles just for the purpose of getting equal operation time even at the beginning of PP.

On the other hand, we argue that negotiation may be more important than switch. Some previous studies concerned about who initiated the switch. Bryant, Romero, and du Boulay (2005) found that switches were mostly initiated by the driver. Contrary to the literature, Plonka et al. (2011) found that switches were frequently initiated by the navigator. However, none of those studies above focused on a special switch which was not individually initiated by the drive or the navigator, but was reached via negotiation between the driver and the navigator. This kind of switch is crucial for PP because negotiation deals with the essence of collaborative learning.

In task-oriented interactions of collaborative learning, negotiation can occur on three main levels (Allwood, Nivre, & Ahlsén, 1992; Dillenbourg, & Baker, 1996): (1) communication (meaning, signification of utterances, words, etc.); (2) task (problem-solving strategies, methods, solutions, etc.); and (3) management of the interaction on above levels 1 and 2 (coordination, feedback on perception, understanding, and attitudes.). Obviously, the switch via negotiation involves the three levels according to partners' dialogues mentioned above.

Moreover, collaborative learning implies symmetry among partners at the same negotiation level in task-oriented interaction (Dillenbourg & Baker, 1996). This opinion is confirmed by Freudenberg, Romero, and du Boulay (2007), who found that the experienced driver and navigator tended to talk in terms of the same levels of abstraction rather than working at different levels of abstraction (Hazzan & Dubinsky, 2003). As a result, they suggested that rather than the driver and navigator roles being defined by segmenting the problem space according to the level of abstraction, they were more simply defined by the additional physical and cognitive load of typing borne by the driver. That is to say, the more important factor for PP may be not the switch of roles, but the maintenance of the same levels of abstraction which rely on the effective negotiation between the driver and the navigator. The study conducted by Vanhanen and Korpi (2007) also confirmed that the driver/navigator roles were switched only 2-3 times a day, but the partners maintained active communication and hence improved the quality of tasks.

Conclusions

This study showed that the students switching roles in a semi-free form got higher achievement than the fixed time interval from 5 to 30 minutes. The result indicated that the self-directed switch under proper pressure was more effective for the learning achievement in PP than the fixed periods. The preference of tending to adopt a fixed time interval to switch roles existed in previous studies seems like a kind of prejudice.

The results also showed that only the mean to enjoyment was significantly different among the four various periods. Students who switched roles every 5 minutes or semi-freely were more enjoyable than those who switched roles in every task and in every class session. Moreover, the period of switching roles in every class session actually got students' enjoyment decreased after PP.

Contrary to some existing studies, this study showed that the frequency of switching roles decreased significantly with time going in the semi-free class. This difference was possible contributed by two factors. One is that the participants were pupils who were unskilled in programming and immature in collaboration in this study, not experienced adults who worked by programming as in the previous studies. The other is that negotiation is more important than switch, which is often ignored in many PP studies. Although the switch of roles was reduced with the progress of the PP course in this research, but the negotiation between the driver and the navigator became more active than before.

The following recommendations are suggested for teachers to effectively apply PP in practice. One is choosing the semi-free model for achieving the higher learning achievement and more active attitude towards programming. Meanwhile, teachers should foster students' awareness of self-directed switch, and guide students on how and when to switch via negotiation during the process of PP.

Another is providing scaffolding tools for PP, such as the handbook for PP which included pair information, PP guidance, a sheet for the progress of PP (just like Table 3), and a task design table. Especially, the sheet for the progress of PP can help students manage switch in the self-directed pattern.

Finally, teachers should supervise and guide switch and negotiation promptly when a meaningless switch or a lack of negotiation for a long time between the driver and the navigator happens.

Acknowledgements

The research study is sponsored by the project "Collaborative Innovation Center for Talent Cultivating Mode in Basic Education, Nanjing Normal University, China," and the Priority Academic Program Development of Jiangsu Higher Education Institutions in China.

References

Allwood, J., Nivre, J., & Ahlsén, E. (1992). On the semantics and pragmatics of linguistic feedback. *Journal of semantics, 9*(1), 1-26.

Arisholm, E., Gallis, H., Dybă, T., & Sjoberg, D. I. (2007). Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Transactions on Software Engineering, 33*(2), 65-86.

Balijepally, V., Mahapatra, R., Nerur, S., & Price, K. H. (2009). Are two heads better than one for software development? The Productivity paradox of pair programming. *MIS Quarterly, 33*(1), 91-118.

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. In *Proceedings of the 15th Conference on Software Engineering Education and Training* (pp. 100-107). doi:10.1109/CSEE.2002.995202

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining twenty-first century skills. In P. Griffin, B. McGaw, & E. Care (Eds.), *Assessment and teaching of 21st century skills* (pp. 17-66). Dordrecht, Netherlands: Springer.

Bishop-Clark, C., Courte, J., & Howard, E. V. (2006). Programming in pairs with Alice to improve confidence, enjoyment, and achievement. *Journal of Educational Computing Research, 34*(2), 213-228.

Braught, G., Eby, L. M., & Wahls, T. (2008). The Effects of pair-programming on individual programming skill. *ACM SIGCSE Bulletin, 40*(1), 200-204.

Bryant, S., Romero, P., & du Boulay, B. (2005). Pair programming and the re-appropriation of individual tools for collaborative programming. In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work* (pp. 332-333). New York, NY: ACM.

Choi, K. S. (2015). A Comparative analysis of different gender pair combinations in pair programming. *Behaviour & Information Technology, 34*(8), 825-837.

Chong, J., & Hurlbutt, T. (2007). The Social dynamics of pair programming. In *Proceedings of the 29th International Conference on Software Engineering* (pp. 354-363). doi:10.1109/ICSE.2007.87

Cliburn, D. C. (2003). Experiences with pair programming at a small college. *Journal of Computing Sciences in Colleges, 19*(1), 20-29.

DeClue, T. H. (2003). Pair programming and pair trading: Effects on learning and motivation in a CS2 course. *Journal of Computing Sciences in colleges, 18*(5), 49-56.

Dillenbourg, P., & Baker, M. (1996, June). *Negotiation spaces in human-computer collaborative learning*. Paper presented at the International Conference on Cooperative Systems, Juan-les-Pins, France.

Feurzeig, W., Papert, S. A., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487-501.

Freudenberg, S., Romero, P., & du Boulay, B. (2007). "Talking the talk": Is intermediate-level conversation the key to the pair programming success story? In *Proceedings of the AGILE Conference 2007* (pp. 84-91). doi:10.1109/AGILE.2007.1

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A Review of the state of the field. *Educational Researcher, 42*(1), 38-43.

Hannay, J. E., Arisholm, E., Engvik, H., & Sjøberg, D. I. (2010). Effects of personality on pair programming. *IEEE Transactions on Software Engineering, 36*(1), 61-80.

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.

Hazzan, O., & Dubinsky, Y. (2003). Bridging cognitive and social chasms in software development using extreme programming. In M. Marchesi & G. Succi (Eds.), *Extreme Programming and Agile Processes in Software Engineering* (pp. 47-53). Berlin, Germany: Springer.

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan, 95*(1), 61-65.

Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., & Gehringer, E. (2004). On understanding compatibility of student pair programmers. *ACM SIGCSE Bulletin, 36*(1), 7-11.

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

LeJeune, N. F. (2006). Teaching software engineering practices with Extreme Programming. *Journal of Computing Sciences in Colleges, 21*(3), 107-117.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students*? Computer Science Education, 21*(2), 105-134.

Li, Z., Plaue, C., & Kraemer, E. (2013). A Spirit of camaraderie: The Impact of pair programming on retention. In *Proceedings of the 26th Conference on Software Engineering Education and Training* (pp. 209-218). doi:10.1109/CSEET.2013.6595252

Lui, K. M., & Chan, K. C. (2006). Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-computer studies, 64*(9), 915-925.

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.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The Effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin, 34*(1), 38-42.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM, 49*(8), 90-95.

Mendes, E., Al-Fakhri, L. B., & Luxton-Reilly, A. (2005). Investigating pair-programming in a 2nd-year software development and design computer science course. *ACM SIGCSE Bulletin, 37*(3), 296-300.

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: National academies press.

Plonka, L., Segal, J., Sharp, H., & van der Linden, J. (2011). Collaboration in pair programming: Driving and switching. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 43-59). Berlin, Germany: Springer.

Preston, D. (2005). Pair programming as a model of collaborative learning: A Review of the research. *Journal of Computing Sciences in colleges, 20*(4), 39-45.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67.

Rostaher, M., & Hericko, M. (2002). Tracking test first pair programming—An Experiment. In D. Wells & L. Williams (Eds.), *Extreme Programming and Agile Methods — XP/Agile Universe 2002* (pp. 174-184). Berlin, Germany: Springer.

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A Systematic literature review. *IEEE Transactions on Software Engineering, 37*(4), 509-525.

Salleh, N., Mendes, E., & Grundy, J. (2014). Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments. *Empirical Software Engineering, 19*(3), 714-752.

Salleh, N., Mendes, E., Grundy, J., & Burch, G. S. J. (2010). An Empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. In *Proceedings of the 32nd ACM/IEEE International Conference on Software* Engineering (Vol. 1, pp. 577-586). New York, NY: ACM.

Sfetsos, P., Stamelos, I., Angelis, L., & Deligiannis, I. (2009). An Experimental investigation of personality types impact on pair effectiveness in pair programming. *Empirical Software Engineering, 14*(2), 187-226.

Vanhanen, J., & Korpi, H. (2007). Experiences of using pair programming in an agile project. In *Proceedings of the 40th Annual Hawaii International Conference on System* Sciences (pp. 274b-284b). doi:10.1109/HICSS.2007.218

Werner, L., & Denning, J. (2009). Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education, 42*(1), 29-49.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The Fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 215-220). New York, NY: ACM.

Williams, L. A., & Kessler, R. R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM, 43*(5), 108-114.

Williams, L. A., & Kessler, R. R. (2002). *Pair programming illuminated.* Boston, MA: Addison-Wesley Longman Publishing Company.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2015). An Exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research, 53*(4), 562-590.

Zhong, B., Wang, Q., & Chen, J. (2016). The Impact of social factors on pair programming in a primary school. *Computers in Human Behavior, 64*, 423-431.