US-China Education Review, ISSN 1548-6613
January 2011, Vol. 8, No. 1, 89-97

Learning to Program in KPL Through Guided Collaboration

Sheng-Che Hsiao, Janet Mei-Chuen Lin, Jiin-Cherng Kang
National Taiwan Normal University, Taipei, Taiwan, China

A quasi-experiment was conducted at an elementary school to investigate if guided collaboration would facilitate programming learning of 6 graders. Sixty-six students of two intact classes learned to program in KPL (kid's programming language) for 18 weeks during the experiment. One class was randomly assigned to the control group (i.e., free-collaboration) and the other to the experimental group (i.e., guided-collaboration). Students in both groups formed heterogeneous teams of 3 persons. The guided-collaboration teams in experimental group were provided with a worksheet for every programming task. The worksheet contained a set of task-specific guiding questions to guide students through the problem-solving process in a systematic and disciplined manner. An analysis of test scores showed that the experimental group significantly outperformed the control group in the achievement tests, suggesting that the guiding questions were useful in enhancing students' comprehension of programming concepts and developing their programming skills. A comparison of the amount of time spent at each problem-solving step showed that the experimental group spent significantly more time than the control group at the analysis/design/coding step as well as the reflection step. Conversely, the control group spent considerably more time at the debugging step. With the help of the guiding questions, students in the experimental group were also more able to conduct meaningful discussions.

Keywords: collaborative learning, KPL programming, programming instruction, kid's programming

Introduction

Collaborative learning focuses on collaboration between teammates to maximize their learning outcomes and it has become popular in various educational contexts (Ellison, Boykin, Tyler, & Dillihunt, 2005; Gokhale, 1995; D. W. Johnson, R. T. Johnson, & Holubec, 1998; D. W. Johnson, R. T. Johnson, & Smith, 1998, pp. 699-708). Collaborative learning has also been used in schools at all levels to teach programming. According to Webb (1984, pp. 1076-1088), for example, learning computer programming could be accomplished successfully in group settings and different profiles of student characteristics and experiences in the group predicted different computing outcomes. Researchers have also shown that students who practiced "pair programming" produced better programs and were more likely to complete programming courses successfully (McDowell, Werner, Bullock, & Fernald, 2002; Williams & Kessler, 2000; Williams, Kessler, Cunningham, & Jeffries, 2000; Williams, Wiebe, YANG, Ferzli, & Miller, 2002). In LIN, LI, HO and LI's study (2007), 6th

Sheng-Che Hsiao, Ph.D., Graduate Institute of Information and Computer Education, National Taiwan Normal University.
Janet Mei-Chuen Lin, professor, Graduate Institute of Information and Computer Education, National Taiwan Normal University.
Jiin-Cherng Kang, Graduate Institute of Information and Computer Education, National Taiwan Normal University.

graders formed teams of 3 to learn logo (MSW logo) programming, and it was found that those teams that were provided a worksheet containing guiding questions for each practice problem performed significantly better in the achievement tests than those teams that did not use the guiding worksheets.

This study expanded on LIN, LI, HO and LI's study by using the guided-collaboration strategy to teach a procedural language—KPL (kid's programming language). Our goal was to investigate how the guiding questions should be designed to enable students to write procedural programs in a systematic and disciplined manner and to find out if the learning benefits observed in LIN, LI, HO and LI's study could be repeated in this study. Specifically, the research questions to be answered were:

(1) Would the performance of the experimental group (provided with guiding worksheets) be different from that of the control group?

(2) Would the amount of time spent at each problem-solving step by students of the experimental group be different from that by the control group?

(3) Would the teamwork behavior of the experimental group be different from that of the control group?

Method

In order to answer the above research questions, an experiment was conducted in the spring semester, 2008, at a public elementary school in Taichung, Taiwan. It adopted a mixed-method design, collecting both quantitative and qualitative data from various sources. The participants, the experimental procedures and the research instruments used in this study are described below.

Participants

Two 6th-grade classes, each with 33 students, participated in this study. One class was randomly assigned to the experimental group, and the other to the control group. To ensure that the two groups were comparable with respect to students' intellectual abilities, all participants took the Raven's SPM (standard progressive matrices) test, a widely used non-verbal intelligence test, prior to the experiment. Students' SPM test scores were analyzed using independent-sample t-test, and the results showed no difference between the two groups ($t=-0.88$, $p>0.05$). Each class then formed 11 heterogeneous teams of 3 students. Team formation was based on students' SPM scores. In other words, each team was comprised of a high achiever, a medium achiever and a low achiever.

The Experimental Procedure

As Table 1 shows, the experiment lasted 18 weeks from February to June 2008. Both groups of students learned KPL programming in two 40-minute classes each week. Each first class period was a lecture on KPL programming constructs, and the second class period was used for hands-on programming exercises.

Table 1

The Experimental Procedure

Week	Guided collaboration	Free collaboration
0	Raven's SPM test/Team formation	
1-6	Instruction on collaborative skills/Chapter 1-5 (basic KPL syntax)	
7	Midterm exam	
8-17	Chapter 6-9 (selection and repetition)	
18	Final exam	

On the first week of classes, the instructor introduced collaborative skills, including how to communicate with teammates, the right way to resolve conflicts, how to share workload, etc., and let students practice those skills. The students then started to learn KPL programming. A midterm exam was given on the 7th week and the final exam on the 18th week.

Research Instruments

Learning material. The learning material, modified from KPL's manual (Schwartz, 2006), was provided to students as a handout that contained 9 chapters. As Table 2 shows, the programming constructs that students learned included variables and data types, drawing instructions, input/output statements, arithmetic expressions, selection structures and repetition structures. Students were introduced to the syntax, the semantics and some of the pragmatics of each feature, followed by examples and exercise problems.

Table 2

Learning Material

The midterm exam		The final exam	
Chapter	Content	Chapter	Content
1	What's programming?	6	Input/Output statements
2	What's KLP?	7	Arithmetic expressions
3	Basic programming concepts	8	Selection structures
4	Variables and data types	9	Repetition structures
5	Drawing with KPL		

Achievement tests. Both the midterm exam and the final exam comprised a written part and a hands-on programming part. The midterm exam covered chapters 1 to 5, and the final exam chapters 6 to 9. Written tests contained multiple-choice and blank-filling questions. For a hands-on programming test, students were required to write two complete KPL programs from scratch. Partial credit was given for programs which, though not totally correct, did implement some of the required features.

Guiding worksheets. Computer programming involves the problem-solving phase and the implementation phase (Costelloe, 2004). Polya (1971) provided general heuristics for solving problems of all kinds. He identified 4 basic principles of problem-solving—understanding the problem, devising a plan, carrying out the plan and looking back. In programming, the principle of "carry out the plan" involves translating a solution into a programming language, testing the program to make sure it produces correct results and debugging the program if errors occur.

Combining Polya's principles and the steps specific to problem-solving through programming, we divided the programming process into 5 steps: analysis, design, coding, debugging and reflection. These 5 steps served as the basic structure for organizing the guiding questions for each practice problem that students were asked to solve in class. It was intended that the guiding questions would lead students toward solution in a systematic manner and that team discussions would have proper focus at each problem-solving step. Eight guiding worksheets were designed and used by the experimental group during the experiment. Each team in the experimental group was given a worksheet for each practice problem. Students were required to answer all guiding questions on a worksheet through team discussions, and turn the completed worksheet in for grading. To assure that the control group received the equivalent information, the instructor described the problem-solving steps to students of the control group orally and taught them how to analyze a problem, design

a solution, etc..

To show how a guiding worksheet looks like, we present an example in Figure 1. The problem to be solved in this example is to display 20 multiples of 9, as shown in Figure 1, using a FOR loop.

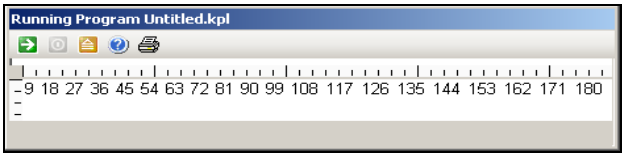

Figure 1. Display 20 multiples of 9.

To help students understand the problem, 7 guiding questions were provided for the "analysis" step. Four of the 7 questions are listed below:

(1) According to the problem statement, how many numbers have to be shown on screen? ( );

(2) According to the problem statement, the numbers shown on screen are multiples of ( );

(3) The first number shown on screen is 9 × ( )=( );

(4) The last number shown on screen is 9 × ( )=( ).

There were 6 guiding questions for the "design" step. They prompted students to start with the simplest case and proceed to the more and more complicated cases, until students finally realized that they had better use a loop instead of brute-force for displaying more than a few numbers. Five of the guiding questions are listed below:

(1) How would you write the program if only one number (i.e., 9) is to be shown on screen?

Print ( )

(2) Is there another way of showing only one number (i.e., 9) on screen?

Print (9 ×)

(3) How would you write the program if there are two numbers (i.e., 9 and 18) to be shown on screen?

Print (9 ×) Print (9 ×)

(4) How would you write the program if there are three numbers (i.e., 9, 18 and 27) to be shown on screen?

Print (9 ×) Print (9 ×) Print (9 ×)

(5) If a million multiples of 9 are to be shown on screen, do we need to write a million "Print" statements then? If not, can you think of a feature in KPL that would allow us to repeat a piece of code many times?

At the "coding" step, a program skeleton, as shown in Figure 2, was provided. The guiding questions were used to help students fill the blanks in the program skeleton.

```
Define i As (①)
For i = (②) to (③)
   Print (④)
Next
```

Figure 2. Program skeleton.

(1) A FOR loop needs a control variable to count the number of times the loop will be executed, what data type (blank ①) should that control variable be declared?

(2) In order to display 20 multiples of 9 from the first to the last, what should you fill in blank ② and ③ respectively?

(3) Look back at the guiding questions at the "design" step. What is the arithmetic expression that should

be filled in blank ④?

The students were then asked to write down the complete program on the worksheet and check its correctness before entering the code for execution.

At the "debugging" step, the guiding questions tried to help students determine possible causes of errors. Students were asked to compare the result produced by their program with the correct output. If errors occurred, they could use the guiding questions to identify if their erroneous output matched any of the cases provided. They then had to describe what they did to correct the error(s).

At the "reflection" step, the guiding questions prompted students to propose other ways of solving the problem. For example, students were asked if it was possible to use addition instead of multiplication in the FOR loop. If so, how would they write the program? Which of the two programs did they think was better and why?

Classroom observation tools. Students' teamwork behavior during problem-solving was observed using multiple tools, including the observation sheets used by the two observers, 3 digital cameras and screen capture software. The video clips that recorded students' problem-solving activities were used to retrieve the amount of time each team spent at each programming step. The data were verified against the data obtained from the screen capture software. In order to analyze students' teamwork behavior quantitatively, the teamwork behavior was classified into different types, as to be explained later, and the frequency of occurrence of each type of behavior was derived for further analyses.

Results and Discussions

Achievement Tests

As already mentioned, the participants took the midterm and the final exams on the 7th and 18th weeks. The total score for each exam was 100 points, including a written part (78 points) and a programming part (22 points). As Table 3 shows, the experimental group and control group had a mean score of 63.67 and 48.94 respectively; the difference between them reached statistical significance (t=2.75, p=0.008). Further analyses of the written part and the programming part separately showed that the experimental group (mean: 50.61) performed significantly better than the control group (mean: 39.90) in the written part (t=2.79, p=0.007). However, the two groups did not differ significantly in their programming scores.

Table 3

T-test Result for Midterm Test Scores

Midterm exam	Groups	N	Mean	Std. Dev.	t	p
Written part	Experimental	33	50.61	16.21	2.79	0.007**
	Control	31	39.90	14.28		
Programming part	Experimental	33	13.06	8.80	1.77	0.08
	Control	31	9.03	9.36		
Total score	Experimental	33	63.67	25.43	2.75	0.008**
	Control	31	48.94	22.60		

Note. **p<0.01.

Table 4 presents the results of the analysis of the final test scores. The experimental group (mean: 54.30) scored higher than the control group (mean: 36.87), and the difference was statistically significant (t=2.89, p=0.005). Further analyses showed that there were significant differences between the two groups in both the

written part (t=2.20, p=0.032) and the programming part (t=3.80, p=0.000).

Table 4

T-test Result for Final Test Scores

Final exam	Groups	N	Mean	Std. Dev.	t	p
Written part	Experimental	33	42.33	20.08	2.20	0.032*
	Control	31	31.77	18.25		
Programming part	Experimental	33	11.97	9.03	3.80	0.000**
	Control	31	4.53	6.03		
Total score	Experimental	33	54.30	25.43	2.89	0.005**
	Control	31	36.87	22.60		

Notes. *p<0.05; **p<0.01.

Based on the above analyses, we can reasonably conclude that the guiding worksheets did benefit the experimental group both in terms of enhancing students' comprehension of the programming concepts, as evidenced by their higher scores in the two written tests, and developing their programming skills, as evidenced by their higher scores in the final programming test. As to the reason why the two groups did not show significant difference in the midterm programming test, a possible explanation is that the two programs that students wrote in the midterm programming test covered only basic KPL features (i.e., variables, arithmetic expressions and drawing instructions). They may be too simple to show the benefits of treatment. When the programs involve more complicated features (i.e., selection and repetition structures), such as those students were asked to write in the final programming test, the benefits brought about by the guiding worksheets become more apparent.

Time Spent at Each Problem-Solving Step

As mentioned previously, video cameras and screen capture software were used to record students' problem-solving activities. Each team was recorded at least 4 times during the experiment. The amount of time each team spent at each programming step was retrieved from the video clips and verified against the data obtained from the screen capture software. Since the experimental teams needed to spend extra time to answer guiding questions on the worksheet, the average time that the experimental group spent on solving a problem was usually longer than that of the control group. To adjust for this factor, we compared the percentages of time, rather than the lengths of time, spent at each problem-solving step by the two groups.

Another adjustment made during data analysis was that the analysis, design and coding steps were treated as a composite step when we computed the length of time a team spent at each programming step. The reason was that the teams in the control group typically developed programs in an ad-hoc manner since they were not required to follow the sequence of steps rigidly. Thus, the time they spent on each step of analysis, design and coding could not be clearly distinguished. To have a fair basis for comparison, the 5 problem-solving steps therefore were compressed into 3, namely, "analysis-design-coding", "debugging" and "reflection". The results are shown in Table 5.

As can be seen, a significant difference was found between the experimental group and the control group in each of the 3 steps, all at 99.9% confidence level. The experimental group spent a larger percentage of time at the "analysis-design-coding" step (69%) and the reflection step (18%) than the control group (50% vs. 14%). Conversely, the control group spent much more time on debugging step than the experimental group

(36% vs. 12%).

Table 5

Chi-square Test of Time Spent at Each Problem-solving Step

Steps	Groups	Percentage of time (%)	Chi-square	p
Analysis/Design/Coding	Experimental	69	176.58	0.000^{***}
	Control	50		
Debugging	Experimental	12	362.10	0.000^{***}
	Control	36		
Reflection	Experimental	18	16.72	0.000^{***}
	Control	14		

Notes. $^{***}p<0.001$.

These results indicate that the use of guiding worksheets did succeed in "forcing" teams in the experimental group to spend more time on understanding the problem, designing the solution and coding, which enabled them to produce programs with fewer errors, thus less time was needed for debugging. An opposite situation occurred with the control group. That is, teams in the control group were able to produce programs more quickly, but they had to spend more time finding and correcting errors in their programs.

As the data also show, the use of guiding worksheets led the experimental group to spend more time on reflection. Clements and Meredith (1992) had pointed out that students do not automatically transfer knowledge gained in one situation to another. Therefore, questions that cause students to reflect on what they are doing are instrumental. In view of the importance of reflection in improving learning, this may at least partly explain why the experimental group was able to outscore the control group in the achievement tests.

Teamwork Behavior

In order to analyze students' teamwork behavior quantitatively, we identified 5 types of teamwork behavior, as described below, and the frequency of occurrence of each type of behavior was counted for further analyses.

(1) Problem-solving-related discussions: This type of behavior refers to discussions conducted between team members regarding how to solve the programming problem at hand. For example, "I think we need to declare a variable here" or "Why don't we use a FOR loop instead?";

(2) Problem-solving-unrelated discussions: This type of behavior refers to discussions that do not have anything to do with solving the programming problem at hand. For example, "I'd like to play computer games" or "I went to a movie last weekend";

(3) Looking information up in the handout: This type of behavior refers to one or more team members looking for information in the handout containing the learning material. It may be to check the syntax of a statement, to find a similar problem that appeared before, and so forth;

(4) Seeking help: This means the team members seek help from the instructor or students of other teams to resolve problems;

(5) Other: All other behaviors, such as dozing, day-dreaming or playing.

The total number of occurrences of each type of teamwork behavior was counted by examining the video clips closely. It was divided by the total length of time (in minutes) to derive a frequency-per-minute count for each type of teamwork behavior exhibited in the two groups. The frequencies were then compared using a t-test.

The results are shown in Table 6.

Table 6

T-test Results for Frequencies of Teamwork Behavior

Behaviors	Groups	Frequency	Std. Dev.	t	p
Problem-solving-related discussion	Experimental	0.6871	0.2714	5.7254	0.000***
	Control	0.3129	0.1253		
Problem-solving-unrelated discussion	Experimental	0.1814	0.1071	2.5656	0.014*
	Control	0.1020	0.0928		
Looking information up in the handout	Experimental	0.0600	0.0721	-2.3815	0.022*
	Control	0.1252	0.1024		
Seeking help	Experimental	0.0578	0.0366	-2.4936	0.017*
	Control	0.1	0.07		

Notes. *p<0.05; ***p<0.001.

It is worth noting that the experimental group conducted problem-solving-related discussions more than twice as frequently as the control group. Apparently, the guiding worksheet was effective in promoting structured and meaningful discussions. It is also interesting to find that the experimental group conducted problem-solving-unrelated discussions more often than the control group. This may be viewed as a side-effect of the more frequent discussions conducted by the experimental group. As to the fact that the control group exhibited the behavior of "looking information up in the handout" and "seeking help" twice as often as the experimental group, the difference may also be attributed to the frequency of discussions, because if most of the problems could be solved through team discussions, it would be less necessary to search for information in the handout or seeking outside help.

Conclusion

In this research, we conducted an experiment at an elementary school to investigate if guided collaboration would facilitate programming learning of 6th graders. The guided-collaboration teams were provided with a worksheet for every programming task. The worksheet contained a set of task-specific guiding questions organized into 5 sections—analysis, design, coding, debugging and reflection, which were intended to guide students through the problem-solving process in a systematic and disciplined manner. The results showed that the experimental group significantly outperformed the control group in the achievement tests. A comparison of the amount of time spent at each problem-solving step showed that the experimental group spent significantly more time than the control group at the analysis/design/coding steps as well as the reflection step. Conversely, the control group spent considerably more time than the experimental group at the debugging step. With the help of the guiding questions, students in the experimental group were also more able to conduct meaningful discussions. In all, this study confirms and extends LIN, LI, HO and LI's (2007) findings that worksheets containing carefully designed guiding questions are useful in enhancing students' comprehension of programming concepts and developing their programming skills.

References

Clements, D. H., & Meredith, J. S. (1992). *Research on logo: Effects and efficacy*. Retrieved Janauary 11, 2009, from http://el.media.mit. edu/logo-foundation/pubs/papers/research_logo.html

Costelloe, E. (2004). *Teaching programming the state of the art*. [PDF document]. Retrieved Novemer 12, 2008, from

https://www.cs.tcd.ie/crite/ publications/sources/programmingv1.pdf

Ellison, C. M., Boykin, A. W., Tyler, K. M., & Dillihunt, M. (2005). Examining classroom learning preferences among elementary school students. *Social Behavior and Personality, 33*(7), 699-708.

Gokhale, A. A. (1995). Collaborative learning enhances critical thinking. *Journal of Technology Education, 7*(1), 22-30.

Johnson, D. W., Johnson, R. T., & Holubec, E. J. (1998). *Cooperation in the classroom* (7th ed.). Edina, Minn: Interaction Book Company Edina.

Johnson, D. W., Johnson, R. T., & Smith, K. A. (1998). Coperative learning returns to college. *Change, 30*(4), 26-35.

LIN, J. M. C., LI, Y. L., HO, R. G., & LI, C. C. (2007). Effects of guided collaboration on sixth graders' performance. Paper presented at *the 37th ASEE/IEEE Frontiers in Education Conference*.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. Paper presented at the proceedings of *the 33rd SIGCSE Technical Symposium on Computer Science Education*.

Polya, G. (1971). *How to solve it: A new aspect of mathematical method*. Princeton, N.J.: Princeton University Press.

Schwartz, J. (2006). *Beginning programming with phrogram*. Retrieved September 30, 2007, from http://phrogram.com/files/folders/phrogram_documentation/entry556.aspx

Webb, N. M. (1984). Microcomputer learning in small groups: Cognitive requirements and group processes. *Journal of Educational Psychology, 76*, 1076-1088.

Williams, L., & Kessler, B. (2000). The effects of "pair-pressure" and "pair-learning" on software engineering education. Paper presented at the proceedings of *the 13th Conference on Software Engineering Education & Training*.

Williams, L. A., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software, 17*(4), 19-25.

Williams, L. A., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education, 12*(3), 197-212.