Teaching and Learning Programming: A Constructivist Approach

Miwha Lee
Busan National University of Education

ABSTRACT

This study examined the cognitive consequences of constructivist teaching practices on the acquisition and transfer of programming with respect to the design of an instructional context that would encourage students to engage in reflective thought; the cognitive consequences of learning in the constructivist context; and the relation between the social and the individual in the teaching and learning process of programming. Students worked on a variety of programming and design problems in constructivist instructional contexts. The results indicated that between-group differences over repeated measures consistently favored students in reflective instruction. Rather than simple differences on measures, the pattern of mean differences over time conformed to a chain of cognitive consequences regulating the acquisition of programming. The implications of the study and suggestions for future research were discussed.

Keywords : Programming instruction, Constructivist teaching

프로그래밍 교수-학습에 대한 구성주의 접근

이미화
부산교육대학교 컴퓨터교육학과

요 약

본 연구는 프로그래밍 교육에 구성주의 교수-학습을 적용하여 프로그래밍 습득 및 전이에 미치는 영향을 분석해 보는 데 목적이 있다. 이에 따라 프로그래밍 학습을 위한 구성주의 기반 교수 환경을 설계하고 이러한 환경에서 실제로 수업을 실시하였다. 본 연구의 목적 및 선행 연구에 기초하여 고안된 연구 도구를 사용하여 프로그래밍 학습 및 태도에 미치는 효과 및 관련 변인들을 측정 분석하였다. 분석 결과 구성주의 기반 프로그래밍 교수-학습 과정에서 연구 대상의 학년 및 교수 환경에 따라 유의미한 효과가 있었으며 긍정적인 영향을 미치는 것으로 나타났다. 본 연구의 결과에 대한 시사점 및 후속 연구에 대한 제언을 제시하였다.

키워드 : 프로그래밍 교수-학습, 구성주의

논문투고: 2012-09-20
논문심사: 2012-09-20
심사완료: 2012-09-24

1. Introduction

1.1. Need for the Study

Although the theoretical tradition initiated by Dewey underscores the importance of reflection to education, it is silent about fruitful targets for reflection in specific domains. Hence, this study relied on previous research about programming to determine what might be important to target for constructivist teaching and learning[21]. First, studies of expert and novice programmers suggest that experts take a design perspective on programming, that is, they think about the goals of the program and potential relations among chunks of code to accomplish their goals, whereas novice programmers tend to see programming as equivalent to writing code[1][17][25].

Second, novice programmers often identify programming actions at the level of individual programming statements; they fail to "chunk" program code in relation to the goal it is intended to accomplish[15]; e.g., the programming structure is flat rather than hierarchical.

Third, and related to the first two points, students often treat programs as local constructions without thinking much about their connections to other related programs or ideas. This tendency highlights the importance of "bridging instruction" to help students develop relations among programming constructs[16].

Fourth, the very interactivity of programming can lead students to chains of action-reaction, rather than mindful consideration of the consequences of their activity[7].

These research findings collectively suggest fruitful targets for reflection because more mindful consideration of any or all of them may have a powerful impact on learning. The research findings guided the development of constructivist teaching practices, each of which was intended to promote more mindful consideration of a programming language[5][26]. To this end, the present study instigated four general classroom practices, each designed to focus on a key target for constructivist teaching and learning in the context of programming, as follows: (a) dialogue about design principles and real audiences for program to promote reflection about programming as design rather than as simply writing code, (b) provision of programming templates to exemplify and promote reflection about key principles of programming design, like modularity, (c) summarization of the action of programming code to promote reflection about the action of chunks of code rather than lines of code, and (d) compare-contrast questions to help students reflect dialogically about relations among different programs or chunks of code, or about relations between programming and everyday experiences.

This study attempted to design a program of constructivist instruction by considering previous research about programming. Taken as a whole, the research on programming supports the need to promote reflection about programming activity, especially because the very interactivity of computer programming can disguise the virtues of considered action. To counteract this tendency, the present study aims to design constructivist instruction to foster reflection in the teaching and learning process of a programming language.

1.2. Purpose of the Study

The purposes of this study were threefold:

First, to design an instructional context that would encourage students to engage in reflective thought;

Second, to examine the cognitive consequences of learning in the constructivist context, with the focus on the relation between reflection and the development of both simpler and more complex forms of programming knowledge;

Third, to analyze the relation between the social, here represented by a set of communally shared design practices, and the individual in the teaching and learning process of programming.

2. Designing Constructivist Teaching

2.1. Student Designers

Students reflect about design, both at the level of the semantics of individual commands and at the schematic and strategic levels of program design, by engaging in teacher-led dialogue about it (see [4][11][13]). To stimulate reflection about the semantics of programming commands, the instructor might ask questions like:

> If you were making up a programming language, how would you do this? Why? What are some other ways that you might do this? What should the units be? Why? Why not? Which one should we use?

These questions promote student dialogue about the assumptions and design principles of the language; students' consensual choices are occasionally incorporated as conventions to reflect these conventions. To promote consideration of design strategies at the program level, students' programs are tested periodically by having other students use them. Student observation of users' reactions to their programs is intended to prompt consideration of alternative designs and help students develop a language of design[14][18].

2.2. Programming Templates

Programming templates are provided to students in the constructivist context. Each programming template exemplifies a major programming construct, such as breaking problems down by writing separate subprograms (i.e., problem decomposition) or using procedures to package collections of commands (i.e., program modularity). The rationale for this approach is to objectify and make visible in code important programming practices and to make it easier for students to compare their practices with those of more skilled programmers. The programming templates are also intended to serve as source analogs for programming solutions. In addition to problem decomposition and modularization, templates are written for programming conventions about variables, conditionals, list processing, looping, and debugging.

2.3. Summarization

To promote reflection about chunks of code rather than individual lines of code, students summarize the actions of their procedures and what they have accomplished each day. Summarization-skill training is modeled after that of Palinscar and Brown[19] and Day[8], beginning with short passages of text and proceeding to program code. The rationale for this component of constructivist teaching is that, by summarizing procedures, students would reflect on the program's purpose, thus bridging the programming actions to the larger context of their use, and by summarizing their accomplishments for the day, students would similarly reflect on the connection between purpose and structure.

2.4. Compare-Contrast Questions

Students are asked to compare what they are doing to other things that they know about or to articulate similarities and differences among their procedures. The rationale is to help students develop problem schemata and to bridge from their knowledge of a programming language to related contexts. The questions are also designed to help students distinguish necessary commands from superfluous commands when they design a program.

3. Method

3.1 Subjects

The participants in the study were students at two grades: twenty-four third-grade and twenty-six sixth-grade students from an elementary school located in a metropolitan school district. All the subjects had some previous experience with computers at or outside of school in a variety of subject areas. The subjects were rank ordered within each grade on the general composite score and assigned to instructional contexts via a stratified random sampling procedure. They remained in the same instructional contexts throughout the study.

3.2 Procedure

Prior to instruction, all participants were interviewed about their previous experience with computers, both in the school and at home. Three tasks measuring verbal and spatial working memory were administered. Scores obtained on the three measures of working memory were standardized within each grade; a sum of these three measures was restandardized to form a general working-memory variable. This working-memory standard score was then combined with a reading comprehension score to form a general composite of working memory and reading comprehension. The construction of this blocking variable, the composite of working memory and prior achievement, drew from previous studies that indicated strong relations among programming learning, prior achievement in school measured by reading skill, and individual differences in working memory[12][25].

The participants were rank ordered within each grade on the general composite score and assigned randomly by alternate ranks to one of two instructional conditions: the inquiry-based and reflective context. Students in both instructional contexts worked for a total of eighteen 40-minute instructional sessions spanning three months of the school calendar. The order of the meeting times was counterbalanced between instructional conditions at each grade to avoid teacher-practice effects. Learning was assessed periodically throughout the course of instruction and immediately following the end of instruction.

3.3 Description of Instruction

The programming topics ranged from a simple introduction to graphics command and the design and production of a movie to more complex programming constructs, facing more complex challenges of design, like those involved in creating interactive programs as well as games or tutorials, with increasing orchestration of skills required over the course of instruction[9]. The primary difference between the two instructional contexts was the four practices designed to increase students' opportunities for reflection, described previously: (a) teaching from a design perspective, (b) student summaries of procedures and daily activity, (c) availability of programming templates, and (d) emphasis on compare-contrast questions that bridged between different programming episodes or between programming and other activities. In addition to these four practices, ways to objectify and make visible particular skills were also introduced throughout the course of instruction. These included computer programs and role-playing skits to help students reflect about the semantics of move and turn commands, the semantics of looping, the steps involved in debugging, and the operation of variables in programs.

3.4 Overview of Measures

As indicated previously, this study aimed to examine student acquisition of four levels of programming knowledge, transfer of the knowledge, and the relation between socially constituted beliefs and individual performance. Table 1 displays the wide range of constructs and measures employed in this study.

<Table 1> Summary of Constructs and Measures

Theoretical Construct	*Measure*
Cognitive resources	Spatial matrix task
	Sentence span
	Working memory
Programming knowledge	
Syntactic	Command recall
Semantic	Move-turn mastery
	Programming constructs mastery
Schematic	Command triads
	Procedure triads

	Cued recall
Strategic	Graphics design problems
	Debugging problems
Transfer: Specific skills	Error detection
	Summarization
Transfer: General skills	Integrating old-new information
	Planning
	Pattern induction
Attitude & Beliefs	Attitudes toward programming
	Programming preferences

3.5 Research Design and Data Analysis

The study employed a randomized block design[3]. The primary analytic technique used was the analysis of covariance (ANCOVA), with each student's standardized score on the blocking variable, the composite of working memory and reading ability, serving as the covariate. The between-subjects factors were instructional condition and grade. Separate analyses were conducted within grade whenever incommensurate measurement made between-grade comparisons fruitless.

4. Results and Discussion

Four sets of results are presented comparing student learning between instructional contexts and grades. First, students' short-term acquisition of levels of programming knowledge is examined. Second, students' long-term acquisition of programming knowledge is presented. Third, transfer of knowledge is then presented, followed by the examination of relations between socially constituted beliefs and individual performance.

4.1 Short-Term Acquisition of Knowledge

Table 2 displays means and standard deviations by instructional context and grade for measures of (a) working memory, (b) recall of commands: syntactic level of knowledge, (c) mastery of turn and move commands: semantic level of knowledge, and (d) each of two measures of strategic knowledge: solving graphics design problems and debugging.

<Table 2> Means and standard deviations of measures of short-term acquisition of levels of programming knowledge

	Grade 3				*Grade 6*			
	Instructional Context				*Instructional Context*			
	Inquiry		*Reflective*		*Inquiry*		*Reflective*	
	M	*SD*	*M*	*SD*	*M*	*SD*	*M*	*SD*
Working memory	12.5	7.8	11.4	5.0	33.6	23.7	34.2	21.1
Syntactic	8.5	5.4	8.5	4.1	13.8	4.3	14.2	3.6
Semantic	23.1	7.5	33.5	10.9	37.4	13.7	50.1	4.6
Strategic								
Design	2.5	1.7	3.6	1.5	2.2	1.5	5.4	2.3
Debugging	3.2	1.9	3.8	1.9	3.9	2.1	6.8	2.7

The results of ANCOVA applied to the measure of working memory indicated grade-related differences, *F* (l, 45) = 26.94, *p* < .01, and the blocking variable (composed of working memory and achievement scores) accounted, as expected, for a significant portion of the within-group regression, *F* (1, 45) = 19.01, *p* < .01. There were no differences due to instructional conditions at either grade. For the measure of command recall, the results indicated no differences between instructional conditions, but the difference between grades was significant; *F* (1, 45) = 20.64, *p* < .01.

The results of the analysis applied to the measure of the mastery of move and turn commands indicated significant differences between instructional conditions, *F* (1, 45) = 18.84, *p* < .01, but not between grades. The results of the analysis of the composite measure of problem-solving suggested an interaction between grade and instructional condition, *F* (1, 45) = 5.24, *p* < .05. Separate analyses within each grade indicated no reliable difference between instructional conditions at grade three, but a reliable difference between instructional conditions at grade six, *F* (1, 23) = 17.48, *p* < .01. The difference between grades on the composite measure of problem-solving performance was significant, *t* = 2.24, *p* < .01.

In summary, no differences were found between reflective and inquiry instruction for the acquisition of syntactic knowledge. Students participating in either form of instruction learned about syntax, although sixth-grade students learned more commands on average than did third-grade students. However, students in the reflective context at both grade levels learned more about the semantics of these commands, but sixth-grade students were on average no better than third-grade students. Students in the reflective context demonstrated superior problem-solving skills (strategic knowledge) at grade six but not at grade three. Differences in working-memory resources were evident between grades. The results show a form of discriminant validity. First, as expected, the reflective intervention made no difference for students' recall of commands. Second, consistent with the chain-of-consequences hypothesis, the highest level of between-group differences was centered at the semantic level in the third grade and the strategic level in the sixth grade.

4.2 Long-Term Acquisition of Knowledge

Table 3 displays means and standard deviations for the long-term acquisition of four levels of programming knowledge: syntax, semantics, schematic, and strategic.

<Table 3> Means and standard deviations of measures of long-term acquisition of levels of programming knowledge

	Grade 3				*Grade 6*			
	Instructional Context				*Instructional Context*			
	Inquiry		*Reflective*		*Inquiry*		*Reflective*	
	M	*SD*	*M*	*SD*	*M*	*SD*	*M*	*SD*
Syntactic	19.8	4.2	21.0	5.2	36.7	11.1	39.9	8.7
Semantic	34.1	11.5	38.9	9.5	49.3	8.2	51.5	4.2
Schematic	13.0	5.3	15.9	4.5	21.6	9.1	22.3	4.1
Strategic								
Design	1.3	0.9	5.8	4.0	3.5	2.1	9.2	5.7
Debugging	2.5	2.3	6.0	1.8	6.2	4.5	18.0	8.1

Syntactic knowledge. The results of ANCOVA applied to a composite variable of the simple recall of commands across time indicated no reliable differences between instructional methods, but reliable differences between grades: *F* (1, 45) = 5.46, $p < .05$. The blocking variable accounted for a significant portion of the within-groups regression, *F* (l, 45)= 6.50, $p < .05$.

Semantic knowledge. The results of the analysis indicated no significant differences between inquiry and reflective forms of instruction at either grade level. The blocking variable at both grades accounted for a significant percentage of the within-group regression for this measure: *F* (1, 21) = 8.02, $P < .01$ at grade three and *F* (l, 23) = 13.19, $P < .01$ at grade six.

Schematic knowledge. Students' schematic knowledge was represented by a composite standard score consisting of the sum of standard scores within each grade for the postinstructional administration of command triads, procedure triads, and the cued recall of commands fitted by the ordered tree algorithm. The results suggested reliable differences between instructional conditions at the third grade, *F* (1, 21) = 10.68, $p < .01$, but not at the sixth grade. The blocking variable did not account for a significant portion of the within-groups regression at the third grade, but it did at the sixth grade, *F* (l, 23) = 5.79, $p < .01$.

Strategic knowledge. Strategic knowledge was assessed by designing a graphic program and by debugging. Different forms of these measures were administered to each grade. The results of the analysis of the measure of program design suggest reliable differences between instructional conditions at each grade level: *F* (1, 21)= 13.99, $p < .01$, at grade three and *F* (1, 23) = 11.64, $p < .01$, at grade six. The blocking variable was unrelated to performance on this measure. The results of the analysis of the measure of debugging also indicated reliable differences between instructional conditions at each grade level: *F* (1, 21) = 18.68, $p < .01$, at grade three and *F* (1, 23)= 23.87, $P < .01$, at grade six. The blocking variable accounted for a significant portion of the within-group regression: *F* (l, 21) = 5.18, $p < .05$, at grade three and *F* (l, 23) = 6.08, $p < .05$, at grade six.

In summary, after a prolonged period of teaching and learning, there was no difference in student learning between instructional contexts with respect to either syntax or semantics, although sixth graders learned more on average about both forms of knowledge than did the third graders. Students in the third grade participating in the reflective context developed more schematic knowledge than their counterparts participating in the inquiry context. No such advantage was found for sixth-grade students. Students participating in the reflective context at both grade levels were better able to put their knowledge to use. The pattern of results conformed to the cognitive-chain hypothesis. At the sixth grade, the benefits of reflection were evident only at the top of the chain, strategic knowledge, whereas for younger students, the benefits of reflection started at the schematic level.

4.3 Transfer

Student performance on measures of transfer by grade and instructional condition is shown in Table 4. The ANCOVA results of the measure of error detection indicate reliable differences between instructional conditions at each grade: *F* (1, 21) = 14.46, *p* < .01, at grade three, and *F* (l, 23) = 4.84, *p* < .05, at grade six. The correlation between student performance on debugging and error detection was substantial at each grade: r (22) = .77, *p* < .05, and r (24) = .60, *p* < .05, respectively. The results for the transfer measure of summarization also suggests reliable differences between instructional conditions: *F* (l, 21) = 25.22, *p* < .01, for grade three and *F* (l, 23) = 4.64, *p* < .05, for grade six. The composite blocking variable did not account for a significant portion of the within-groups regression. As expected, contexts at either grade level for the more general transfer measures of planning or inducing numeric and spatial patterns. In summary, the pattern of results for the measures of transfer conformed to that of other studies of programming: When general skills are taught and assessed specifically, transfer is evident[20][23].

<Table 4> Means and standard deviations for measures of transfer, attitude, and beliefs

	Grade 3				*Grade 6*			
	Instructional Context				*Instructional Context*			
	Inquiry		*Reflective*		*Inquiry*		*Reflective*	
	M	*SD*	*M*	*SD*	*M*	*SD*	*M*	*SD*
Error detection	4.5	3.9	11.1	4.5	9.6	7.1	13.9	3.1
Summarization	6.5	3.1	11.2	2.7	10.4	4.5	13.9	3.5
Planning								
Violations	4.1	2.4	3.0	1.4	2.6	2.5	1.9	1.6
Path length	21.3	1.5	21.6	1.7	19.9	2.8	20.5	3.2
Patterns								
Numeric	5.7	2.5	6.8	2.3	12.4	2.9	12.8	3.6
Spatial	1.1	0.8	1.4	0.8	1.6	0.8	1.9	1.0
Attitude	65.3	8.4	59.8	12.9	58.8	15.9	63.3	13.9
Beliefs								
Debugging	-	-	-	-	10.4	2.3	12.4	2.6
Problem heuristics	-	-	-	-	12.8	2.4	16.8	2.7
Program design	-	-	-	-	11.4	2.9	14.8	2.2
Program preference	0.8	0.7	1.7	0.7	1.2	0.8	1.7	0.5

4.4 Attitude and Beliefs

The means and standard deviations for postinstructional measures of attitude toward and beliefs about programming are shown in Table 4.

Attitude. No differences between grades or instructional conditions were detected for student attitude toward their programming experiences, with the mean judgment centered around the agree point of the scale.

Beliefs. The measure of programming preference suggests a higher endorsement of modular programming practices, using procedures and subprocedures, in the reflective group at each grade level, although the difference was reliable only at the third grade *F* (l, 21) = 11.08, *p* < .01. The analyses of students' beliefs about programming practices were confined to the sixth grade. Students in the reflective condition were more likely to report agreement with statements reflecting ideal practices about debugging, problem-solving heuristics, and program design; debugging, *t* (24) = 2.08, *p* < .05, problem-solving heuristics, *t* (24) = 4.00, *p* < .01, and program design, *t* (24) = 3.23, *p* < .01.

5. Conclusion

This study examined the cognitive consequences of constructivist teaching practices on the acquisition and transfer of programming. Students worked on a variety of programming and design problems in either an inquiry or reflective instructional context. The inquiry context represented former, research-tested best practices in which teachers elicited predictions, asked leading questions, and assisted students when they encountered programming impasses. The reflective context was designed to improve on these practices by providing explicit encouragement of a design stance where students assumed roles as potential designers as well as actual roles as the designers of their own programs[22]. General instructional methods like the adoption of a student-designer approach and the use of programming templates were intended to promote the growth of reflection across a wide range of computer programming activities. Other instructional methods made specific elements of programming more visible as objects for reflection. In each instructional context and grade, the study assessed students' learning of (a) multiple forms of programming knowledge, (b) transferable components of learning, and (c) students' attitude toward and beliefs about their learning experiences. At each grade level, consistent and persistent differences between the inquiry and reflective instructional contexts were found. However, rather than simple mean differences between instructional contexts, the results on most measures conformed to a pattern suggested by a chain of cognitive consequences[2][17].

The pattern of mean differences between instructional contexts among the measures of the four levels of programming knowledge supports a chain of cognitive consequences, in which learning of lower levels of knowledge supports the acquisition of higher levels of competence[10][15]. In this study, lower levels of knowledge were embedded within higher levels of knowledge, as suggested by theories of cognitive apprenticeship that caution against learning skills in isolation[6], so the claim is not that one level must be mastered before learning anything about the next. Instead, the claim of the theory as applied to this study is that lower levels of learning are consolidated before higher levels are consolidated.

The results suggest that students in the reflective context at both grade levels transferred some components of learning to related contexts. Transfer of specific skills like detecting errors is feasible to the extent to which acquisition and transfer tasks share mental units in common[24]. The results obtained generally support this line of work: Students in the reflective context received more effective instruction about skills like summarization and program debugging, and they transferred these skills more readily than did their counterparts. Transfer of general skills was often the lodestone of earlier research about the utility of learning programming languages, but more than a decade of research suggests otherwise. While important and interesting findings have been revealed, the study needs to be replicated. In addition, Future research employing other types of approaches to programming instruction may be worth further investigation.

References

[1] Ambrose, S., Bridges, M. A., DiPietro, M. C. Lovett, M. C., & Norman, M. K. (2010). *How learning works*. San Francisco, CA: Jossey-Bass.

[2] Anderson, J. R. (2009). *Cognitive psychology and its implications*. New York: Worth.

[3] Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences*, 2, 141-178.

[4] Carter, C., & Foley, B. J. (2011) *Constructing the self in a digital world*. Cambridge University Press.

[5] Clements, D. H. (1990). Metacomponential development in a programming environment.

Journal of Educational Psychology, 82, 141-149.

[6] Collins, A., Brown, J. S. & Newman, E. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, learning, and instruction.* (pp.453-494). Hillsdale, NJ: LEA.

[7] Cope, P., & Simmons, M. (1994). Some effects of limited feedback on performance and problem solving strategy. *Journal of Educational Psychology,* 86, 368-379J.

[8] Day, J. D. (1986). Teaching summarization skills. *Cognition and Instruction,* 3, 193-210.

[9] Dettelman, D., & Sternberg, R. J. (1993). *Transfer on trial: Intelligence, cognition, and instruction.* Norwood, NJ: Ablex.

[10] Fay, A. L., & Mayer, R. E. (1987). Children's naive conceptions and confusions about graphics commands. *Journal of Educational Psychology,* 79, 254-268.

[11] Harel, I. (1991). *Children designers: Inter-disciplinary constructions for learning and knowing.* Norwood, NJ: Ablex.

[12] Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension: Individual differences in working memory. *Psychological Review,* 99, 122-149.

[13] Khine, M. S., & Saleh, I. M. (2011). *Models and modeling: Cognitive tools for scientific enquiry.* New York: Springer-Verlag.

[14] Lawson, B. (2005). *How designers think*. Boston: Butterworth Architecture.

[15] Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher,* 14, 14-16, 25-29.

[16] Littlefield, J., Delclos, V., Bransford, J., Clayton, K., & Franks, J. (1989). Some prerequisites for teaching thinking: Methodological issues. *Cognition and Instruction,* 6, 331-366.

[17] Mayer, R. E. (1992). Teaching for transfer of problem-solving skills to computer programming. In E. DeCorte, M. C. Linn, H. Mandl, & L. Verschaffel (Eds.), *Computer-based learning environments and problem solving* (pp.193-206). New York: Springer-Verlag.

[18] Olson, D. R., & Astington, J. W. (1993). Thinking about thinking: Learning how to take statements and hold beliefs. *Educational Psychologist,* 28, 7-23.

[19] Palinscar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and monitor-ing activities. *Cognition and Instruction,* 1, 117-175.

[20] Robins, A. (1996). Transfer in cognition. *Connection Science*, 8, 185-203.

[21] Rountree, N., Rountree, J., & Robins, A. (2002). Identifying the danger zones. *Inroads*, 34, 121-124.

[22] Schon, D. A. (1990). *Educating the reflective practitioner.* San Francisco, CA: Jossey-Bass.

[23] Siegler, R. S. (2004). *Children's thinking.* Englewood Cliffs, NJ: Prentice-Hall.

[24] Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

[25] Soloway, E., & Spohrer, J. (1988) *Studying the novice programmer.* New York: Psychology Press.

[26] Swan, K. (1989). Programming and the teaching and learning of problem solving. *Journal of Artificial Intelligence in Education,* 1, 73-92.

저 자 소 개

이 미 화
미국 위스콘신대학교 석사 및 박사
미국 위스콘신대학교 연구교수
캐나다 멀티미디어연구소 객원교수
호주 멀티미디어교육연구원 연구교수
호주 원격교육센터 연구원
부산교육대학교 컴퓨터교육학과 교수
E-mail : mlee@bnue.ac.kr