

An Empirical Investigation of Visual Computer Programming Language Effects on HBCU Students' Problem-Solving Capabilities

Mike F. Unuakhalu, Ed.D.
Kentucky State University

Abstract

This study was conducted to explore how Historically Black Colleges and University (HBCU) students exposed to an integrated object-oriented programming instruction with transfer training activities in everyday tasks could enhance their problem solving abilities considering the digital divide factor, which places the African American and other minorities far behind whites in terms of computer and Internet usage. Besides shedding some new light on the link between learning programming languages and developing problem solving skills, this study also provides initial research on a population that has previously not received sufficient focus. The results of the study, after an eight-week treatment, did not show any support of improved problem solving ability from instruction in computer programming between both treatment groups. The paper concludes with a discussion of the implications and directions for future research.

1. Introduction

According to the Final Report of the Association for Computing Machinery (ACM) K-12 Task Force Curriculum Committee of the Computer Science Teachers Association (CSTA, 2003), problem solving and critical thinking should be a central focus of computer science and programming instruction. Additionally, among its position statements, the National Council of Teachers of Mathematics (NCTM, 1989) had indicated that problem solving, with its demand for linguistic competence in discerning mathematical relationships, be emphasized in curricula more than arithmetic procedures and algorithmic routines. Artists, philosophers, designers, and scientists in all disciplines are united in the intensely creative activity of problem solving. To achieve this objective, particularly for students who are educationally and digitally disadvantaged, computer science teachers are presented with challenges and opportunities (Cardelle-Elawar, 1995). The (NCTM, 1995) also maintains its commitment to equity in mathematics education for all students, including under-represented minority students. For students with limited programming competence, it might be difficult to determine what the aforementioned goals mean in practical terms. Simply changing the textbooks or providing programming instruction that is grounded in problem solving for the students may not be enough (Kaplan & Patino, 1996).

Many schools' computer science curricula include computer programming particularly in their introductory computing courses. One reason given for the inclusion is that programming helps students to think creatively and deliberately, as well as develop sound judgments in order to understand and change with their environment. This perception, supported by many educators (Choi & Repman, 1993; Dalton & Goodrum, 1991; Kimmel, Deek, & Kimmel, 2003; Lee & Thompson, 1997; Norris & Jackson, 1992; Robin, Rountree, & Rountree, 2003;), was based on the speculated link between learning programming languages and development of problem solving skills, as well as the recommendation of the NCTM. The findings concerning the impact of computer programming experience on problem solving are mixed (Lee & Thompson, 1997; Liao & Bright, 1991; Palumbo & Reed, 1991). Furthermore, the literature suggests that the majority of the programming and problem solving studies were conducted in predominantly white colleges. There is no empirical research available that focused on Historically Black Colleges and Universities (HBCUs). Additionally, the current study provides critical insight regarding the opinions of a population that statistically falls within the diminutive end of digital divide (Buzzetto-More & Sweat-Guy, 2006).

Research that has specifically addressed the development of thinking skills that closely parallel the programming environment has yielded the most conclusive results supporting a positive link between programming and problem-solving skills. For example, Liao and Bright's (1991) meta-analysis addressed the use of computer programming at various levels (elementary, high schools, and colleges) of students' cognitive outcomes. They found that students who received programming instruction scored higher on various cognitive ability tests than those who did not. Likewise, a number of studies were conducted by Reed and his colleagues to investigate if programming instruction using BASIC and Logo would enhance students' problem solving skills (Palumbo & Reed 1991; Reed & Palumbo, 1988; Reed, Palumbo, & Stolar, 1987). They found that students' problem solving skills increased significantly after systematic exposure to BASIC programming instruction and that both BASIC and Logo programming languages were equally effective in increasing students' problem solving scores.

In their comparative study, Choi and Repman (1993) researched the relationship between problem solving and two programming languages – Pascal and FORTRAN in 58 college-age novice programming students. Results showed that while learning to program significantly improved problem solving abilities of college level students, there was no significant difference between the Pascal and the FORTRAN group. Dalton and Goodrum (1991) examined the effects of systematic programming and problem solving instruction on problem solving skills and attitudes. One hundred and 72 students were exposed to one of four computer programming and problem solving treatments for 20 weeks. Results showed significant problem solving skills improvements when programming was integrated with systematic problem solving instruction, but computer programming alone was not as effective when not integrated. Dalton and Goodrum maintained that most literature supporting the effects of programming on problem solving has been non-empirical.

In their study, Deek, Kimmel, and McHugh (1998) reinforce the contention that problem solving is a possible foundation for teaching programming. It describes a first year problem solving-based computer science course which introduces language features only in the context of the students' solutions to specific problems. Results showed that students in the problem solving group generally rated their own abilities and confidence slightly more highly and achieved a significantly better grade for the course than the control group students. In another study, Kimmel, Deek, and Kimmel (2003) found that students' skills at solving engineering graphics problems improve as a result of implementing a structured approach towards developing a solution plan.

While some studies show a link between programming and problem solving, other studies do not support the use of programming as a means of developing problem-solving skills, or as a means of accelerating the cognitive development of young students (Clements & Gullo, 1984; Jonassen & Reeves, 1996; Pea & Kurland, 1984; VanLengen & Maddux, 1990). In the research conducted by Haberman and Averbuch (2002), they found that students showed negative transfer effects from logic programming to procedural programming when evaluating incorrectly designed recursive programs. This author, in a previous study found no significant difference on general problem solving ability. Only one of the four problem-solving skills (Analysis of Attributes) measured showed an improvement in performance favoring the integrated programming group (Unuakhalu, 2008).

1.1 The Digital Divide

Digital inequality is one of the most critical issues in the knowledge economy. According to Morris (2001), the digital divide is a gap between people with access to

computer technology and those who do not have such technology. The African-American community and other minority ethnic groups are far behind whites in terms of computer and Internet usage.

Many proponents of HBCUs are concerned about the existence of a digital divide between predominantly white colleges and HBCUs. Furthermore, Williams, Goldstein, and Goldstein (2001) noted the continued growth in historical pressures on HBCUs as they are expected to stretch their limited resources to offer access to widening student population and prepare students for their first occupation with minimal corporate training. The HBCU stakeholders that are comprised of students, state governments, trustees, parents, and donors hold the universities responsible for high risk education populations, as well as funding and accreditation.

Johnsson (2001) asserts that it is difficult to make a dent in the digital divide when many of Chicago's public high schools do not encourage gifted students with a knack for programming to develop their mathematics skills – a must for a career in computer science. Gilbert et al. (2008) reported that progress regarding the digital divide has been made in closing the achievement gap between white students and minorities in mathematics achievement. They maintained that there is still much work to be done in narrowing this gap.

1.2 Computer Programming and Everyday Tasks

People have always performed many operations they encountered in their everyday lives with little effort. They have become so versed in performing some operations that they do not stop to think that there is a set of clear, concise, and finite steps they can follow in order to solve these everyday problems. When we have a task that needs specific operations with which we are not familiar, we may use a recipe prepared by someone else. If we follow the directions of the recipe as accurately as possible, we expect to get the desired results. Everyday examples can be performed using task structures similar to the repetitive task structures of the programming language being learned. For example, when learning subscripted arrays in programming, the task structure in the example of playing a simple card game would parallel the task structure of programming with subscripted arrays. Another example involves driving a car around the block several times to find a place to park. The search continues until that elusive spot is found. This everyday task would parallel the same sequence of functions that a computer program would employ when it encounters a loop in a program (Zak, 2001).

The purpose of teaching programming is to teach problem solving as well as programming. Linn and Dalbey (1989) propose a "chain of cognitive accomplishments" to describe the sequence of learning programming: (a) learn the

language features, (b) learn to design programs to solve problems, and (c) learn problem solving skills applicable to other formal systems. To learn the primitive features of a programming language is just the first component in this link. The ultimate goal for learning programming in this model is to learn to solve problems not only in one programming language, but also in other formal systems. The formal systems could be a new programming language, a database management system, a computer controlled monitoring device, etc.

Robin, Rountree, and Rountree (2003) reported possible reasons for students not gaining the cognitive skills that are necessary and beneficial in novel situations are: (a) students have not moved far enough along the chain of cognitive accomplishments to achieve the type of transfer necessary for a certain novel skill or situation; (b) students have learned ineffective or inefficient procedures for solving; and (c) students have deviated from the chain of cognitive accomplishments and, therefore, have not moved to the more complex cognitive operations. Linn and Dalbey's (1989) argument parallels many cognitive psychologists claim that the programming process will help students realize that problem solving is a process of understanding a goal, making a plan, building a solution one piece at a time, and checking the results (Clements, 1986; Pea & Kurland, 1984; Palumbo & Palumbo, 1993; Robin, Rountree & Rountree, 2003)

The difficulty that minority students often encounter in introductory computer programming classes, and their poor performance on subsequent assessments of their learning programming, may be unrelated to their potential for learning and understanding programming concepts and procedures. Rather, many of these minority students are likely to be forced into a pattern of failure simply because they have not yet learned how to seek information, plan strategies, monitor an ongoing performance, evaluate the process, and revise their previous strategies (Kaplan & Patino, 1996; Lee & Thompson, 1997).

It is fitting, therefore, that if we are to consider the digital divide factor, NCTM's and CSTA's recommendations for mathematics and computer science, respectively, for everyone seriously, educators need to have some specific adaptations in instructional methods for minority students, particularly for those in Historically Black Colleges and Universities (HBCUs). In general, these adaptations should include programming instructions integrated with real life everyday problems. It is expected that these everyday examples can be performed using task structures similar to the repetitive task structures of the programming language the students have learned. Toward this end, the author has developed a method for teaching programming instruction embedded with everyday tasks to enhance problem solving

capabilities of students irrespective of their digital divide sensitivity

1.3 Purpose of the Study

This study investigated the effectiveness of a visual programming instruction embedded with everyday tasks on undergraduate HBCU students' problem solving skills. There is no research that has examined the programming and problem-solving link in a HBCU environment. Therefore, the author is interested in employing the use of an effective learning tool and methodology that comprises of decomposing, planning, executing, error identification, and debugging activities that are utilized when writing programs and designing solutions to everyday tasks that will improve students' problem solving skills. The tool is also sensitive to the digital divide factor. In an effort to carry out the study's purpose, the following specific research questions that are consistent with general predictions derived from research literature were addressed: (a) Do students need programming instruction integrated with everyday tasks and activities to enhance their programming skills in solving problems? and (b) What effect does programming instruction integrated with everyday tasks and activities have on problem solving capabilities of HBCU students?

The programming and problem-solving link has not been examined using visual programming language embedded with everyday tasks nor has it been examined in HBCUs. Thus, the author developed the following hypotheses:

1. There will be difference in the composite scores on a posttest measuring problem-solving ability between students of the programming plus everyday tasks and students of the programming-only groups.
2. There will be difference in the composite scores on a posttest measuring computer programming competence ability between students of the programming plus everyday tasks and students of the programming-only groups.

2. Methodology

2.1 Participants

Forty students enrolled in either of two sections of a semester long college level computer science programming course titled "Programming in Visual BASIC" in a small southeastern HBCU participated in this study. There were 26 freshmen, eight sophomores, two juniors, and four seniors. Subjects were largely freshmen who had reported little or no knowledge of the subject matter. There were 18 African Americans, 15 Hispanics, one White, and six claiming other ethnicity groups. Thirty-two subjects had no computer programming experience. Eight had some limited experience in a programming language other than Visual BASIC. All the participants had adequate access to a

computer. Twenty-four males and 16 females were included in the study. The mean age of the subjects was 19 and they were assigned to one of two intact groups.

2.2 Treatments

The participants in this study were configured from each section of the Visual BASIC course into two instructional treatments: (a) programming plus everyday tasks (N=19) and (b) programming only (N=21). The “programming plus everyday tasks” group received a series of four print-based instructional modules on Visual BASIC programming and noncomputer everyday problems during a regularly scheduled class period for eight weeks. For the programming-only group, the four modules were designed in a fashion parallel to that of the programming plus everyday task group, except for the lack of provision of noncomputer everyday problems.

2.3 Instrumentation

The pre- and posttest problem solving skills were composed of two sections of the Watson-Glaser Critical Thinking Appraisal (Deduction and Interpretation) and two sections of Ross Test of Higher Cognitive Processes (Analysis of Relevant and Irrelevant Information, and Analysis of Attributes). This 61-item problem-solving instrument used in this study, as adapted by Reed and Palumbo (1998), has been found to be related to some of the kinds of problem solving activities specifically involved in learning a programming language. Reed and Palumbo maintained that these instruments measure any changes in programming-related, problem solving skills. Several researchers (Choi & Repman, 1993; Palumbo, 1991; Palumbo & Reed, 1991; Reed, Palumbo, & Stolar, 1988; Unuakhalu, 2008) have used this problem-solving instrument since its development. The programming skills of the participants were assessed by a programming competency measure.

The *Watson-Glaser Critical Thinking Appraisal* instrument provides scores for five sections of 16 items each. These measures parallel the processes of problem solving and computer programming (Widmer & Parker, 1985). Test-retest reliability ($r = .75$), the reliability for the alternate forms A and B was .75, and internal consistency coefficients range from .69 to .89 (Watson & Glaser, 1980). These findings are sufficient enough to warrant the use of two subscales (i.e., Deduction and Interpretation) of the test in this research study. The Deduction section is composed of 16 problems that determine whether certain conclusions necessarily follow from information in given statements or problems. In other words, deduction is the ability to see how a specific conclusion was arrived at from a given rule. The Interpretation section has 16 problems that ask students to weigh evidence and decide if generalizations or conclusions based on the given data are warranted.

The *Ross Test of Higher Cognitive Processes* reliability estimates range from .92 and .94 (Ross & Ross, 1976). The Ross Test consists of eight subtests, each keyed to specific types of cognitive processing. Two of the eight sections or subtests from the Ross Test of Higher Cognitive Processes utilized for this study were: (1) the Analysis of Relevant and Irrelevant Information, which consists of 14 mathematical problems that may or may not contain sufficient information to be solved or that may contain irrelevant or extraneous information not essential to solution of the problem. This section measures a student’s ability to analyze data and identify critical information or the lack of the same. (2) Analysis of Attributes that presents groups of similar figures that have a variety of features or attributes. This 10-item section measures a student’s capability to analyze figures, determine their critical elements, formulate hypotheses as to which attributes are necessary for set membership, and use these hypotheses in a decision making process of identifying set members from a group of new figures.

The *Programming Competency Test* contains definitions of terms and commands, applications of specific commands and procedures, and questions that address general competency on the particular skills being addressed in the curriculum. The test was assessed for validity by a panel of computer educators consisting of university instructors who assessed whether or not the test instrument addressed the computer skills taught as described in their respective curricula. After review and evaluation, the panel decided that the programming content test possessed a strong content validity. Additionally, the performance measures internal reliability ($r = .73$) was determined using the Cronbach’s alpha method.

The Short Answer segment of the criterion reference test was scored by three experienced faculty who have taught this course numerous times. This method was deemed necessary to remove any grading or scoring inconsistencies due to its subjective nature. The three raters scored each student’s test individually in order to test the inter-rater reliability of the scoring system without knowing the student’s name or the treatment condition. Each question was assigned a specific value and the maximum achievable score for the Short Answer component was 31 points.

2.4 Procedures

The subjects met twice a week and the duration of the study was eight weeks for a total of 16 sessions. Prior to the onset of the treatment at the first meeting, subjects in the two treatment groups were asked to complete a demographic and computer experience questionnaire after which they were administered a problem solving pretest instrument and a programming competency test. They were subsequently given instruction either in computer programming language integrated with everyday task activities or in a non-integrated computer programming language. During the

eighth week of their respective treatments, subjects in both groups responded to the same problem-solving instrument again and completed a programming competency test covering information taught during that treatment period.

2.4.1 Programming Plus Everyday Tasks Group Curriculum

The instructional material for the programming plus transfer group included the following concepts: (1) problem solving and computer programming language syntax/semantics, (2) VBASIC introduction and designing applications, (3) variables, constants, and calculations (4) looping, selection, and sequence control structures, and (5) arguments and parameters in functions and procedures. The reasoning acquired by the subjects from being exposed to these computer programming concepts could be applied to the solution of other kinds of problems in a noncomputer domain.

Subjects in the programming plus everyday task group were provided with a series of four print-based instructional modules on Visual BASIC programming and noncomputer everyday tasks. Each of the four modules was presented to the subjects every two weeks (four instructional sessions) during a regularly scheduled class period. Subjects received Visual BASIC instruction integrated with everyday tasks and activities that systematically guided them through formal problem solving heuristic tasks of decomposing, planning, executing, error identification, and error debugging while solving Visual BASIC and noncomputer everyday problems. The procedure required when changing an automobile's flat tire, baking a cake, or washing a dirty car are some examples of the everyday tasks utilized in this study.

During the decomposing phase, the students broke down a complex problem into smaller manageable elements. One element is information pertaining to input, a second element is calculation or processing information, and a third is output information. The planning activity was designed to facilitate the students' ability to organize and sequence the decomposed information and elements in a finite number of steps using tasks, objects, and events. Each identified task is matched with a corresponding object and event, which are subsequently used by the subjects to design a user interface. The executing activity was designed to help subjects translate the planned solution into Visual BASIC syntax and execute the program or solution on the computer, but in the case of noncomputer everyday problems, the students were expected to write and implement the detailed solution in simple English statements. Once the objects are created, the subjects then concentrate on writing the specific instructions in Visual BASIC instructing each object how to respond when clicked, double-clicked, scrolled, and so on. The error identification activity revolves around students' ability to describe the discrepancy between planned and actual output,

locate errors, and explain what was wrong, if anything. The debugging step was designed to encourage students to correct identified errors and analyze the results for reliability and efficiency. It was expected that subjects would be able to make modifications to previous steps in the formal problem solving heuristic activities, if necessary, at any point in the process until the problem is solved.

Each instructional module was designed to build upon and follow the information and concepts presented by previous modules. Specifically, each instructional module contains a brief introduction of the objective of the lesson, a review of previously learned commands, new programming commands, and a demonstration of programming with these new commands. Each module presented the learners with two problems to solve using the new commands followed by homework assignment, and a quiz. For the noncomputer short everyday problems, such as changing a flat tire or washing a dirty car, students were expected to complete them using the same problem solving strategies learned in Visual BASIC programming. In addition the students were given printed description of possible solution to the problem as a self-check.

2.4.2 Programming-Only Group Curriculum

Instruction for the programming-only group, unlike the programming plus everyday task group, emphasized programming skills acquired by students who were provided the opportunity to apply their acquired knowledge in the design, development, and debugging of Visual BASIC object-oriented programs only. The subjects were not exposed to noncomputer everyday tasks and algorithms. However, they received the same set of programming problems and examples as the programming plus everyday task group.

2.5 Design of Study

The research design for this study was pretest-posttest quasi-experimental. In this procedure, Treatment and Time Exposure were the independent variables with problem solving skills and program competency tests as the dependent variables. Equivalence of the two groups in this study was assessed by comparing data available on students' test, as well as comparison of scores on the problem solving pretest administered to all students at the beginning of the class.

3. Results

3.1 Biographical Descriptive Statistics

Table 1 presents results from the demographic data obtained from the questionnaire completed by the subjects.

3.2 Performance Measures Descriptive Information

Pre- and posttest mean scores and standard deviations by treatment for the problem solving instrument are presented

in Table 2. They were obtained from the subjects' performance on the problem-solving instrument. Descriptive information on the pre- and posttest mean scores, standard deviations, and sizes by treatment for the programming competency test are also shown in Table 2.

3.3 Problem Solving Measures Analysis

A comparison of the programming plus every day task and programming-only groups on the pretest cumulative total, as indicated in Table 3, shows no significant difference between the two groups in problem solving.

The resulting two factor (Treatment by Exposure Time) ANOVA with repeated measures on one variable (Exposure Time) with problem solving instrument as the dependent measure did not show significant difference between the programming plus everyday tasks and the programming-

only groups. The test for the (Exposure Time by Treatment) interaction was, $F(1,38)=2.62$, $p=0.11$. The results did not support the hypothesis that computer programming instruction embedded with everyday tasks would improve problem solving ability. However, both groups' problem-solving scores significantly increased from pretest to posttest (see Table 2 for means and standard deviations).

3.4 Programming Competency Measures Analysis

A comparison of the programming plus every day task and programming-only groups on cumulative total of the pretest, as indicated in Table 4, shows no significant difference between both groups in programming competence.

The two factor (Treatment by Exposure Time) ANOVA with repeated measures on one variable (Exposure Time) with programming competency as the dependent measure

Table 1
Participant's Demographic Information by Group

	Programming Plus Everyday Task (N = 19)	Programming Only (N = 21)	Total (N = 40)
Sex			
Male	9 (47.3%)	15 (71.4%)	24 (60.0%)
Female	10 (52.6%)	6 (28.6%)	16 (40.0%)
Age			
17-22	14 (73.7%)	16 (76.2%)	30 (75.0%)
23-28	5 (26.3%)	4 (19.1%)	9 (22.5%)
Over 28	0 (0.0%)	1 (4.8%)	1 (2.5%)
Ethnicity			
White	0 (0.0%)	1 (4.8%)	1 (2.5%)
African American	0 (0.0%)	18 (85.7%)	18 (45.0%)
Hispanic	14 (73.7%)	1 (4.8%)	15 (37.5%)
Other	5 (26.3%)	1 (4.8%)	6 (15.0%)
Classification			
Freshman	18 (94.7%)	8 (47.6%)	26 (70.0%)
Sophomore	1 (5.3%)	7 (38.1%)	8 (22.5%)
Junior	0 (0.0%)	2 (9.5%)	2 (5.0%)
Senior	0 (0.0%)	4 (14.3%)	4 (7.5%)
Major			
Computer Science	0 (0.0%)	8 (38.1%)	8 (20.0%)
Information Tech.	19 (100.0%)	1 (4.8%)	20 (50.0%)
Drafting Tech.	0 (0.0%)	4 (19.1%)	4 (10.0%)
Criminal Justice	0 (0.0%)	6 (28.6%)	6 (15.0%)
Other	0 (0.0%)	2 (9.5%)	2 (5.0%)

Table 2

Means and Standard Deviations for Subjects' Problem Solving and Program Competency Measures by Treatment Group

Measure	Programming plus Everyday Task			Programming Only	
		Pretest	Posttest	Pretest	Posttest
Problem Solving Achievement					
Total	M	31.05	37.95	32.81	36.43
	SD	6.57	4.65	7.21	6.33
Program Competency Achievement					
Total	M	38.53	81.22	34.92	58.89
	SD	6.36	11.60	12.47	13.97

Table 3

A Comparison of Problem Solving Instrument Pretest Mean For Both Treatment Groups

Measure	Treatment Group	Exposure Time	M	SD	t-value	*p
Problem Solving Achievement						
Total	PE	Pretest	31.05	6.57	-1.23	0.45
	PO	Pretest	32.81	7.21		

*p < .05

PE = Programming plus Everyday Task

PO = Programming Only

showed a significant difference between the programming plus everyday tasks and the programming-only groups. The test for the (Exposure Time by Treatment) interaction was $F(1,38)=25.46$, $p=0.01$. The results support the hypothesis that computer programming instructions embedded with everyday tasks would improve the subjects' programming competency ability. Furthermore, both groups programming competency scores significantly increased from pretest to posttest. (See Table 2 for means and standard deviations.)

4. Discussion

The results of this study did not show a significant overall treatment effect for computer programming instruction integrated with everyday tasks training for improved problem solving. This finding is not in support of previous body of research that provides evidence that interventions, such as the programming instruction used in this experiment, enhance the learners' problem solving abilities. This current study also produced an unpredicted but

interesting finding, which was the impressive programming content knowledge gains made by the students in spite of the fact that the study was not initially designed to help develop and enhance students' programming skills relative to their problem solving capabilities. This important unexpected significant gain may be attributed to the quality and quantity of programming instructions and accompanying strategy that models how to apply programming skills to other problems rather than teaching the language and only later applying it to solving problems (Bishop-Clark, 1998; Choi & Repman, 1993; Dalton & Goodrum, 1991; Jonassen & Reeves, 1996; Kimme & Deek, 2003; Lee & Thompson, 1997; Palumbo & Reed, 1991). Consistent with the literature, the most interesting result related to the programming outcomes is that strategies fostered through the computer and noncomputer problem solving materials apparently helped students learn programming. However, the limited number of student

Table 4

A Comparison of Pretest Mean On the Problem Competence Instrument for Students in Programming Plus Everyday Task and Programming-Only Groups

Measure	Treatment Group	Exposure Time	M	SD	t-value	*p
Program Competency Achievement						
Total	PE	Pretest	38.53	6.36	0.39	0.70
	PO	Pretest	34.92	12.47		

*p < .05

PE = Programming plus Everyday Task

PO = Programming Only

practice sessions and the inability to ensure that practice happened within the context of programming instruction itself, due to time constraint, could be a reason for failure of transfer to other subject areas in this study. It is also possible that the gains found from the treatment may not have been complex cognitive gains, but memorization of the task, which might suggest students' ability to construct correct solutions, but may not have been able to explain how and why they constructed the solutions. Given the short duration of the intervention used in this study, it may have prevented sufficient mastery of Visual BASIC programming.

The problem solving literature suggests the importance of designing programming activities that encourage the mindful application of problem solving strategies, such as planning, simplification, and modeling (Dalton & Goodrum, 1991; Haberman, 2004; Jonassen & Reeves, 1996; Palumbo, 1991; Kimmel, Deek, & Kimmel, 2003; VanLengen & Maddux, 1990). In this study, problem solving skills did not significantly improve with integrated programming language instruction. One possible reason could be because of the low socioeconomic status of many of the students, thus requiring them to engage in full-time employment while studying (William, Goldstein, & Goldstein, 2002). This may have had an adverse effect on the students' study habit and performance.

The correlation between programming performance and posttest means for the programming plus everyday task group was more significant; indicating that those students who completely mastered programming integrated with everyday tasks instructions solved problems more effectively. This show of sufficient mastery of the visual programming language taught could be explained by the students' interactions with the content of the assigned computer programming and everyday tasks activities that provided an ideal practice medium for the problem solving transfer strategies (planning, decomposing, coding, and debugging) taught.

Furthermore, many of the students in this study have Spanish and French ethnic origin. From the beginning of the year, they had had the opportunity to work in small groups and enroll in an English proficiency course. It was interesting to see how the aforementioned dynamics helped to develop these students' English language skills and confidence. The students demonstrated their willingness and ability to discuss computer and programming during the class session in spite of their ethnic differences. The literature also noted that working on the computer tends to increase the quantity and quality of student discussion (Brenner, 1998).

4.1 Contributions

The process of learning is under constant examination in efforts to design instruction that will foster knowledge transfer across content areas. This study sheds some new light on the link between learning programming languages and developing problem solving skills. Additionally, it provides initial research on a population that has previously not received sufficient focus. Using pre- and posttest assessments, it was found that this study's method for facilitating problem solving skills of HBCU students by utilizing programming embedded with everyday tasks instructions enabled them to become more successful programmers than problem solvers. This study's lack of significant increase in problem solving skills and show of unexpected but impressive programming content knowledge is important to the education field because educators need to see the potential that computers and technology have in enhancing the programming skills of HBCU students.

This paper builds on the findings of a number of similar studies that have been conducted at majority institutions only. Although no such studies were conducted at HBCUs, this study was designed to explore the relationship between learning a specific programming language and enhanced problem-solving capabilities, thus providing valuable information that will help researchers, instruction designers,

and practitioners in making curriculum decisions regarding computer usage in HBCUs.

4.2 Limitations of This Study

Several possible explanations can be offered for the lack of significant increases in the problem solving ability for the programming plus everyday task group. The eight week short duration of the intervention used in this study may have prevented sufficient mastery of Visual BASIC programming. For the programming plus everyday task group, the long and arduous process involved in teaching Visual BASIC instruction embedded with problem transfer strategies in the course of eight weeks may not have been possible. This amount of time may not offer the potential for improvement of problem-solving ability with programming instruction (Fredricksen, 1984; Lee & Thompson, 1997; Palumbo, 1990; VanLengen, 1990).

The programming plus everyday task treatment in this study did not appear to promote the development of problem solving as measured by the instrument used. Problem solving strategies taught in programming requires students to use deductive and inductive reasoning to arrive at a representation of the problem. The students then had to evaluate their representation of the problem and select alternatives to solve the problem. The results of this study suggest that strategies taught in the programming context did not transfer sufficiently to influence the total scores on the problem solving test instrument. It could be the case that the measure was too broad in scope and general, thus requiring an extremely strong treatment for more than one semester. The result of this study indicates that the programming treatment integrated with problem transfer strategies of a particular type had a positive effect on learning programming skills.

Another limitation of this study was the lack of a large sample size. Students' participation was limited to 40 students who completed the study. With such limited size, it is difficult to generalize the results of this study to a larger population. A large-scale study would be necessary before robust and generalizable conclusions could be drawn.

4.3 Summary and Future Research

While the results of this paper suggest that the overall problem solving ability were not significantly better for the programming plus everyday tasks group, current research efforts in this area have indicated some potential for using computer programming as a tool for improving problem solving ability. Although there was no significant evidence of a significant programming and problem solving link between both treatment groups, an unexpected but significant difference on programming competence skills

favoring the programming plus everyday task group was identified.

In light of the results of this study and in an effort to expand and use it as an inspiration for further studies, the following possibilities are suggested:

1. Research is needed that addresses other problem solving skills, which can be taught in these types of computer environments. The current study showed that the treatment was not effective in increasing the overall problem solving measured.
2. Future studies need to lengthen the treatment to at least a semester to allow for sufficient coverage of the decision and logical components of the programming language and problem transfer strategies. The current study involved an intensive eight-week programming session.
3. Since the results from this experiment may only be applied to Visual BASIC programming language, it would be beneficial to conduct further research with other visual programming languages, such as C++, C#, Java, Alice, etc., embedded with everyday tasks activities
4. A more in-depth analysis of advanced students versus beginning students and use of programming needs to be addressed in order to use object-oriented programming integrated with transfer training instruction effectively.
5. The results of this study indicate that there is need for more research in this area, particularly in minority institutions. These findings can help inspire a large-scale more in-depth research study.

References

- Computer Science Teachers Association _CSTA). (2003). *A model curriculum for K-12 computer science: ACM K-12 task force curriculum committee final report*. New York: Association for Computing Machinery
- Bishop-Clark, C. (1998). Comparing understanding of programming design concepts using Visual BASIC and traditional BASIC. *Journal of Educational Computing Research*, 18(1), 37-47.
- Buzzetto-More, N., & Sweat-Guy, R. (2006). Incorporating the hybrid learning model into minority education at a historically black university. *Journal of Information Technology Education*, 5, 153-164.
- Cardelle-Elawar, M. (1995). Effects of metacognitive instruction on low achievers in mathematics problems. *Teaching & Teacher Education*, 11(1), 81-95.
- Choi, W. S., & Repman, J. (1993). Effects of Pascal and FORTRAN programming on the problem-solving abilities of college students. *Journal of Research on Computing in Education*, 25(3), 291-302.

- Clements, D. H., & Gullo, D. F. (1984). Effects of programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051-1058.
- Dalton, D. W., & Goodrum, D. A. (1991). The effects of computer programming on problem solving skills and attitudes. *Journal of Educational Computing Research, 7*(4), 483-506.
- Deek, F. P., Kimmel, H., & McHugh, J. A. (1998). Pedagogical changes in the delivery of the first-course in computer science: Problem solving, then programming. *Journal of Engineering Education, 87*, 313-320.
- Frederiksen, N. (1984). Implications of cognitive theory for instruction in problem solving. *Review of Educational Research, 54*(3), 363-407.
- Gilbert, J., Arbuthnot, K., Hood, S., Grant, M., West, M., McMillan, Y., Cross, V. Williams, P., & Eugene, W. (2008). Teaching algebra using culturally relevant virtual instructors. *The International Journal of Virtual Reality, 7*(1), 21-30.
- Haberman, B. (2004). How learning logic programming affects recursion comprehension. *Computer Science Education, 14*(1) 37-53.
- Haberman, B., & Averbuch, H. (2002). The case of base cases: Why are they so difficult to recognize? – Student difficulties with recursion. *SIGSCE Bulletin, 34*(3), 84-88.
- Johnsson, J. (2001). A push for minorities to cross digital divide. *Crain's Chicago Business, 24*(34), 6.
- Jonassen, D. H., & Reeves, T. C. (1996). Learning with technology: Using computers as cognitive tools. In Jonassen, D. H., (Ed.) *Handbook of Research for Educational Communications and Technology*. New York: MacMillan Publishing, 693-719.
- Kaplan, R., & Patino, R. (1996). The effects of a communicative approach on the mathematical problem solving proficiency of language minority students. Paper presented at the Annual Meeting of the Northeastern Educational Research Association, Ellenville, NY.
- Kimmel, S., Deek, F., & Kimmel, H. (2003). Using a problem-solving heuristic to teach engineering graphics. *International Journal of Mechanical Engineering Education, 32*(2), 135-146.
- Lee, M. C., & Thompson, A. (1997). Guided instruction in Logo programming and the development of cognitive monitoring strategies among college students. *Journal of Educational Computing Research, 16*(2), 125-144.
- Liao, Y., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta analysis. *Journal of Educational Computing Research, 7*(3), 1991, 252-268.
- Linn, M. C., & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 57-81), Norwood, NJ: Ablex.
- Morris, K. (2001). Seeking digital equity. *Black Voices Quarterly, 2*(4), 32-34.
- National Council of Teachers of Mathematics [NCTM]. (1989). *Standards*. Reston, VA.: National Council of Teachers of Mathematics.
- National Council of Teachers of Mathematics [NCTM]. (1995). *Curriculum and Evaluation Standards for School Mathematics*. Reston, VA.: National Council of Teachers of Mathematics.
- Norris, C., Jackson, L., & Poirot, J. (1992). The effect of computer science instruction on critical thinking and mental alertness. *Journal of Research on Computing in Education, 24*(3), 329-337.
- Palumbo, D. B. (1990). Programming language/problem solving research: A review of relevant issues. *Review of Educational Research, 60*, 65-89.
- Palumbo, D. B., & Reed, W. M. (1991). *The effect of BASIC programming language instruction on high school students' problem solving ability and computer anxiety*. Paper presented at the Annual Conference of the Eastern Educational Research Association, Clearwater, Florida.
- Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*(2), 137-168.
- Reed, W. M., Palumbo, D.B., & Stolar, A. L. (1987). The comparative effects of BASIC and Logo instruction on problem-solving skills. *Computers in the Schools, 4*(3), 50-58.
- Reed, W. M., & Palumbo, D. B. (1988). The effects of BASIC programming language on problem solving skills and computer anxiety. *Computer in the Schools, 4*(3), 91-104.
- Ross, J. D., & Ross, C. M. (1976). *Ross test of higher cognitive processes: Manual*. Novato, CA: Academic Therapy Publications.
- Rountree, N., Rountree, J., & Robins, A. (2003). Identifying the danger zones: Predictors of success and failure in a CS1 course. *Inroads (The SIGSCE Bulletin), 34*, 121-124.
- Unuakhalu, M. (2008). Enhancing problem solving capabilities using object-oriented programming language. *Journal of Educational Computer Systems, 37*(2), 121-137.
- Watson, G., & Glaser, E.M. (1980). *Watson Glaser critical thinking appraisal manual*. San Antonio, TX: The Psychological Corporation.
- Widmer, C. C., & Parker, J. (1985). A study of characteristics of student programmers. *Educational Technology, 25*(10), 47-50.
- Williams, K., Goldstein, D., & Goldstein, J. (2002). Improving the study habits of Minority students through web-based courses. *TechTrends, 46*(2), 21-27.
- VanLengen, C. A., & Maddux, C. D. (1990). Does instructional in computer programming improve problem

solving ability? *Journal of Information Systems Education*, 2(2).

Zak, D. (2001). *Programming with Microsoft Visual BASIC: Enhanced edition*. Boston: Course Technology and Thompson Learning Publisher.

Author Information

Mike F. Unuakhalu, Ed.D.

Assistant Professor
Computer & Technical Sciences Department
Kentucky State University
mike.unuakhalu@kysu.edu