Strategic Applications of Distance Learning Technologies

Mahbubur Rahman Syed
Minnesota State University, Mankato, USA

INFORMATION SCIENCE REFERENCE
Hershey · New York

Director of Editorial Content: Kristin Klinger
Managing Development Editor: Kristin M. Roth
Senior Managing Editor: Jennifer Neidig
Managing Editor: Jamie Snavely
Assistant Managing Editors: Carole Coulson
Copy Editor: Alana Bubnis
Typesetter: Carole Coulson
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

and in the United Kingdom by
Information Science Reference (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanbookstore.com

Library of Congress Cataloging-in-Publication Data

Strategic applications of distance learning technologies / Mahbubur Rahman Syed, editor.

p. cm.

Summary: "This collection of advanced research incorporates global challenges and opportunities of technology integration while outlining strategies for distance learning within developing countries"--Provided by publisher.

Includes bibliographical references and index.

ISBN 978-1-59904-480-4 (hardcover) -- ISBN 978-1-59904-482-8

1. Distance education--Computer-assisted instruction. 2. Educational technology. I. Syed, Mahbubur Rahman, 1952-

LC5803.C65S77 2008

371.35'8--dc22

2008007624

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter XIV
A Web-Based Tutor for Java™: Evidence of Meaningful Learning

Henry H. Emurian
University of Maryland, Baltimore County, USA

ABSTRACT

Students in a graduate class and an undergraduate class in Information Systems completed a Web-based programmed instruction tutor that taught a simple Java applet as the first technical training exercise in a computer programming course. The tutor is a competency-based instructional system for individualized distance learning. When a student completes the tutor, the student has achieved a targeted level of understanding the code and has written the code correctly from memory. Before and after using the tutor in the present study, students completed a software self-efficacy questionnaire and a test of the application of general Java principles (far transfer of learning). After completing the tutor, students in both classes showed increases in software self-efficacy and in correct answers on the test of general principles. These findings contribute to the stream of formative evaluations of the tutoring system. They show the capacity of the Web-based tutor to generate meaningful learning (i.e., understanding of concepts) at the level of the individual student.

INTRODUCTION

This chapter presents a continuation of a series of formative evaluations to assess and to enhance the instructional effectiveness of an automated and individualized distance learning system that is intended to assist Information Systems students in beginning their study of Java™. We previously reported our progress in the development of this tutoring system, which teaches a simple Java applet, and its application as the first technical training exercise for students in a computer programming course (Emurian, 2004; Emurian & Durham, 2001, 2002; Emurian, Hu, Wang, & Durham, 2000). The purpose of the tutor is to provide each and every student with a documented and identical level of elementary knowledge and skill. The tutoring system has been demonstrably effective in promoting technical skill and in cultivating self-confidence in beginning students

by giving them a successful learning experience that motivates their further study of Java using textbooks, lectures, laboratory demonstrations, independent problem solving, and the like. The tutoring system is targeted to Information Systems majors, whose primary interests may lie in systems analysis and design, database, decision support systems, knowledge management, and information resource management, but who would benefit from acquiring elementary skill in an object-oriented programming language such as Java.

One of the challenges of developing an automated distance learning system, however, is to craft the instructional experience so that students acquire the capability to solve problems not explicitly taught or encountered in the system itself. When students are able to apply knowledge successfully to new situations, they are said to be demonstrating meaningful learning (Mayer, 2002) as opposed to reciting facts acquired by rote memorization. These two outcomes reflect the opposite endpoints on a generality-specificity dimension of skill (Novick, 1990). Generalizable rules, which may be the essence of meaningful learning, can be acquired by direct instruction and rehearsal or by induction, when many different situations are encountered that exhibit the general rule (Kudadjie-Gyamfi & Rachlin, 2002). The former tactic is consistent with our instructional system design, which is competency-based and intended to insure that all students reach the same level of knowledge and skill with the applet being taught.

The theory supporting the development of the tutoring system is a behavior-analytic model based upon the learn unit formulation of Greer and McDonough (1999) as applied to programmed instruction for technology education (Greer, 2002). The interactive tutoring system interfaces that are presented to the learner reflect the systematic increase in the size of the learn units from simple symbol production (i.e., learning to type) to writing and understanding the entire program. The stream of work leading up to the present study constitutes a series of replications over which the system was refined to ensure that mastery would occur for all learners. This approach is similar to instructional material improvements suggested by formative evaluation (Harley, Seals, & Rosson, 1998) and by design-based research (Hoadley, 2004), and how this strategy contrasts with between-subjects evaluations is discussed in subsequent paragraphs.

The purpose of the present study is to show that students who complete the tutor do acquire general rules that are applicable to programming problems not explicitly addressed in the tutor itself. The study extends our prior investigation (Emurian, 2005) by increasing the number of testable rules to 10 and by supporting the reliability of the outcomes over two different groups of students. This research approach constitutes systematic replication (Sidman, 1960), which is an alternative to null hypothesis testing and intended to validate externally this particular instructional system design rather than to test hypotheses regarding effect sizes across alternative designs. This methodology falls within the scope of an outcomes assessment model of evaluating teaching effectiveness (Fox & Hackerman, 2003), and, in the present situation, the teacher is the Web-based tutoring system.

Interpretative surveys of the scientific literature in far transfer effects of learning continue to show the advantages of explicitly teaching generalizable principles and rules rather than expecting such knowledge to develop implicitly or abstractly as a byproduct of memorizing facts (Barnett & Ceci, 2002) or by pure discovery learning (Mayer, 2004). For example, it is likely more efficient to teach students the rule to begin the name of a Java class with a capital letter than to expect students to discover such a rule inductively by studying many different programs and by trying to discern commonalities among them. In fact, a combination of teaching rules with examples might be optimal for meaningful learning, and

our approach to the design of the tutoring system is based on that latter assumption. The study to follow, then, assesses the extent of rule-governed knowledge before and after students have used the tutoring system.

METHOD

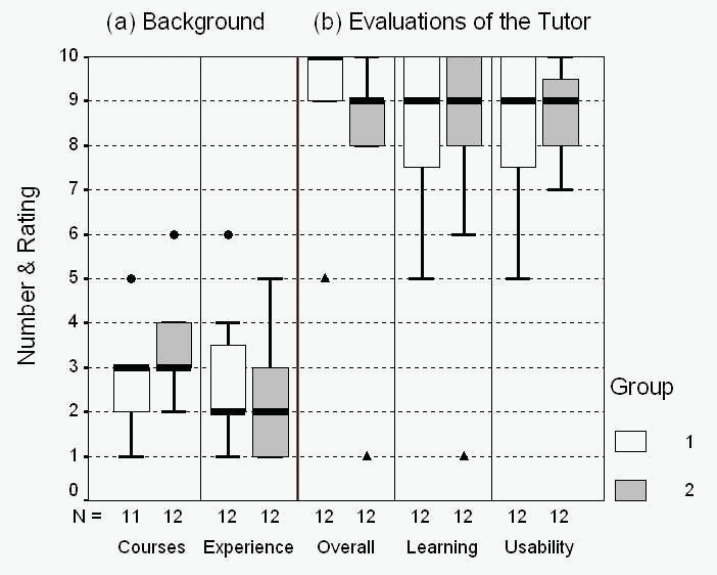
Subjects

Two groups of students participated as subjects in this investigation. Prior to using the tutor, the subjects completed a questionnaire that collected information on gender, age, number of prior programming courses taken, and experience with Java. The anchors for the latter rating scale were as follows: *1 = No experience. I am a novice in Java.* to *10 = Extensive experience. I am an expert in Java.* The subjects in Group 1 were eight female and four male graduate students in Information Systems, who were enrolled in a course (Summer 2003) entitled Graphical User Interface Systems Using Java. The subjects in Group 2 were two female and 10 male undergraduate students enrolled in the same course at a later time (Spring 2004). For Group 1, the median age was 27 years (range = 21 to 49), and for Group 2, the median age was 22 years (range = 20 to 26). A Kruskal-Wallis test[1] showed a significant difference between the two groups in age ($\chi^2 = 5.68$, df = 1, $p < 0.02$). Figure 1(a) presents boxplots of the number of programming courses that the students had previously taken and their experience with Java[2]. The median number of courses taken was three for both groups, although the interquartile range was graphically higher for Group 2. The median reported Java experience was two for both groups. Differences between the groups were not significant on either measure.

Materials

The tutoring system teaches a simple Java applet, which is a program that is downloaded from a server and run in a browser. The program, pre-

Figure 1. (a) Boxplots of the number of programming courses that the students had previously taken and their experience with Java. (b) Boxplots of overall evaluation, teaching effectiveness, and usability ratings. Circles are outliers, and triangles are extreme values.



sented in Appendix A, displays a text string in a browser window. The tutoring system is freely available on the Web[3]. It is intended for students who are not proficient computer programmers and who may lack confidence in their ability to write and to understand a program that works. The Applet code is organized into 32 items and 10 lines (i.e., rows). The student is taught the meaning of each item and the meaning of each line. The student must pass a multiple-choice test on each program component; studying continues until each test is passed correctly. The tutor exits when the student can write the entire program without an error. The content of the tutor, to include the tests embedded within the tutor interfaces, is freely available as a document[4].

The seven stages of the tutoring system, from basic instructions and code examples to the construction of the code from memory, are presented in Emurian (2004) and Emurian, Wang, and Durham (2003). In brief, the tutoring system is an interactive system that combines teaching, assessment of competency, and rehearsal within a single framework. The design of the system is based upon programmed instruction, which takes a learner through a series of experiences from simple mastery of the form of symbols to writing and understanding a complete program. This design reflects the application of behavior principles to designing teaching strategies for technology education (Greer, 2002). These principles are uniquely applicable to individualized instruction.

Information is delivered to the student in a frame. A frame consists of (1) the presentation of the Java symbol (e.g., *import*) to be learned, within the proper context; (2) a textual display of information explaining the symbol's meaning and use; (3) a multiple-choice test on the meaning of the symbol; and (4) an input field for typing the symbol by recall. If the student makes an error during steps 3 or 4, the tutor resets to step 1, and that cycle repeats until the input is correct. When the input in step 4 is correct, the student progresses to the next frame. The functional properties of a frame constitute a learn unit (Greer & McDonough, 1999) when the student progresses from one frame to the next, determined by accurate performance.

Next is presented the explanation for the ninth of the 32 item frames in the tutoring system. It explains the meaning of the *MyProgram* item, which is a subclass of the Applet class in the program.

MyProgram Explanation

The term *MyProgram* is the name of the class that you are writing. Your Java program is a class. The name is an arbitrary alphanumeric string. *MyProgram* is not the name of an instance of this class. It is the name of the class. It is important that you begin to distinguish the name of a class from the names of particular instances of that class that are created later. This distinction will become clearer as you progress through the tutor. Notice that the name of the class begins with a capital letter. That is a convention in Java. The name of a class begins with a capital letter. That is an important rule to know.

The text file that contains the Java program for the *MyProgram* class must have exactly the same name, together with "dot java" at the end. The file for your program would be named *MyProgram.java*. The name of the text file must match exactly the name of the class. That is an important rule to know.

The Java text file, which is the source program, will be compiled with *javac MyProgram.java*. The result of compiling the program is a class file named *MyProgram.class*, which will be located in your directory.

The ten rule-based multiple-choice questions are presented in Appendix B. For Group 1, only the first five questions were available. For Group 2, the number of questions was increased to 10. The correct solution to each question required the application of a general principle that was

presented in the explanatory information. None of the items to solve these questions appeared verbatim in the explanations or in the multiple-choice tests that were embedded in the frames. This eliminated rote memorization as a reason for correct solutions, should they be observed at all. For each rule-based question, the student rated his or her confidence that the correct answer was selected. The ordinal scale anchors were 1 and 10, where *1 = Not confident* and *10 = Totally confident*. A 10-point rating scale was adopted to increase the sensitivity of the scale in order to detect changes in ratings over three successive assessment occasions.

Figure 2 presents the running applet as it appears within the Mozilla browser. Figure 3 presents the display of the 12th Java symbol (i.e., item) to be learned, which is an opening brace symbol ({). In Figure 3, the { would appear when the learner selects the enabled *Show Java* button. In the browser, the symbol is blue to set it apart from the previous symbols that have been learned. Figure 4 shows the symbol's explanatory information that is presented when the user selects the enabled *Explain* button. Figure 5 shows the multiple-choice test for this particular opening brace in the program. Figure 6 shows the input field for the brace. Figure 7 shows the interface for the line-by-line input. In this example, the code displayed in the third line (i.e., Row 3) is ready to be assessed when the learner presses the keyboard *Enter* key. This interface functions similarly to the items learning interface, but the unit of learning is a line of code, and there are two iterations through the interface. The second iteration has no tests associated with it. Figure 8 shows the final interface in the tutor, where the learner enters the entire program. The input format previously enforced by the tutor is relaxed for this stage of learning compared to the previous tutor stages, and the input is evaluated as a stream of characters. The remaining instructional component of the tutoring system, along with the instructions to the learner, may be observed by running the tutor on the Web.

Figure 2. The applet as it appears running in a browser on the Web

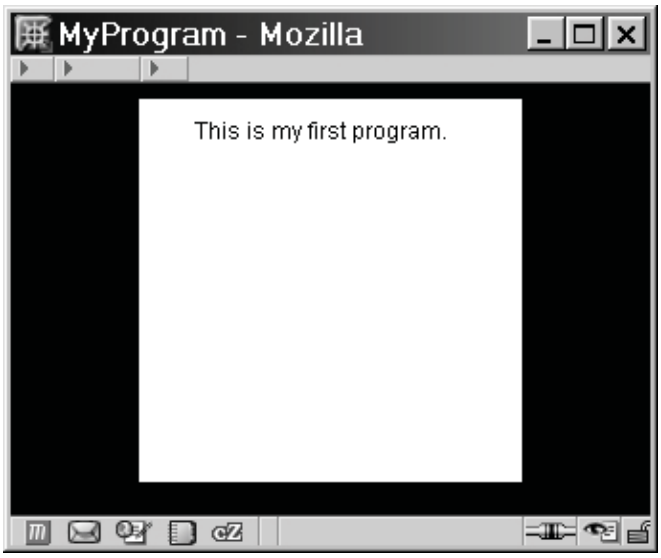


Figure 3. The display of the Java symbol to be learned, which is the opening brace in this example. The brace, as are all symbols first observed for learning, is colored blue to distinguish it from the symbols that already have been learned in the sequence of presentations. The symbol to be learned is presented when the user selects the enabled Show Java button.

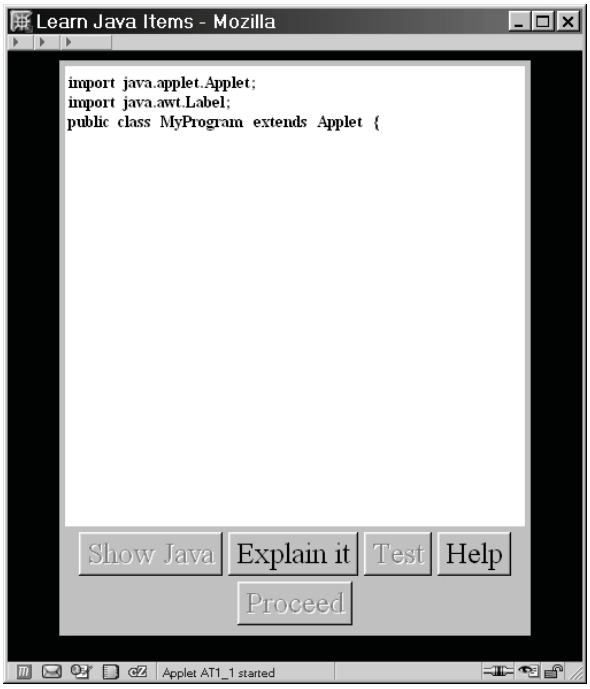


Figure 4. The explanatory information for the brace. It is presented when the user selects the enabled Explain button.

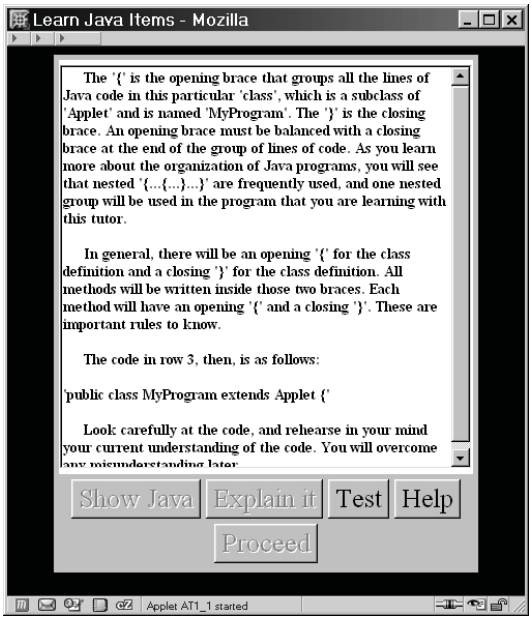


Figure 5. The multiple-choice test for this particular opening brace in the program. It is presented when the user selects the enabled Test button.

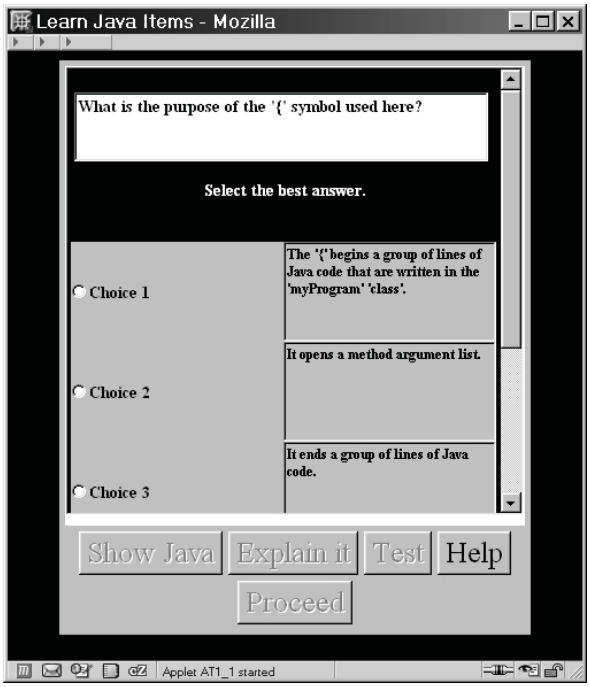


Figure 6. The input field for the symbol. The empty field is presented at the location of the symbol in the tutor code, and its width matches the width of the characters to be entered. This view is presented automatically when the user selects the correct test choice. The user types the symbol from memory into the input field and then hits the Enter key on the keyboard. If the input is correct, the symbol is locked into position in the display, and it appears colored black.

Figure 7. An example of the line-by-line interface with input in the third row. This interface is functionally similar to the items learning interface, but the unit of learning is a line of code, not an item of code.

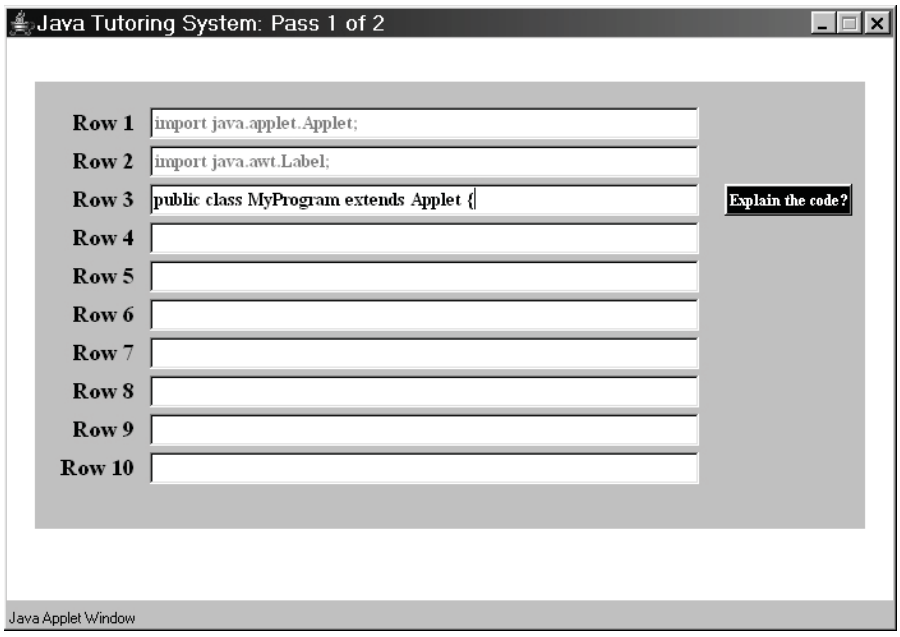
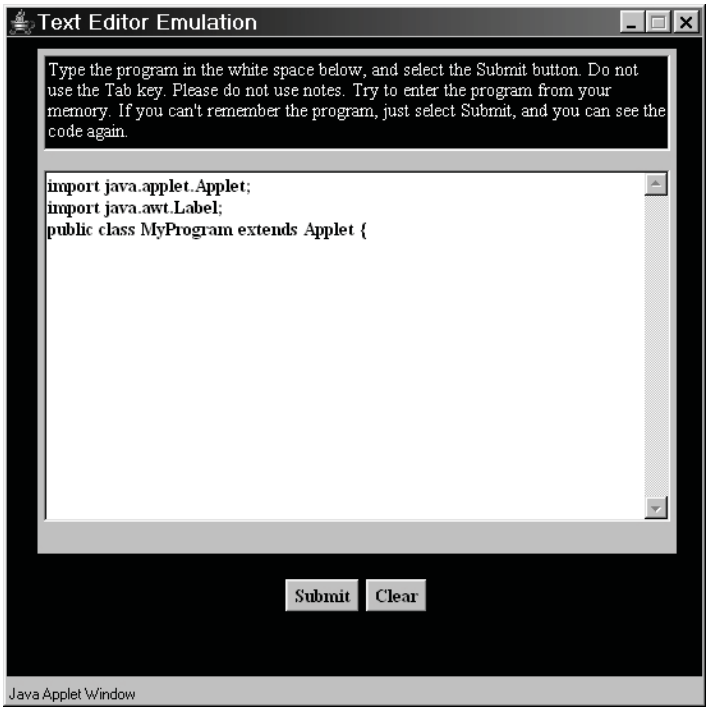


Figure 8. The final stage in the tutor in which the learner enters the entire program. If the input contains an error, the correct code is displayed. After the code view is closed, the learner again attempts to enter the program into a cleared input field. This cycle repeats until the program is entered correctly.



Procedure

During the first scheduled class, students used the Web-based tutor. Although the students accessed the tutor during a class located in a PC lab, the system is designed as an individualized distance learning system. All students completed all stages in the tutor within the three-hour class period. This means that all students in both groups left the first class period being able to write the Java code from memory and with no error. The students had also studied the frames until they accurately could input all items and lines and correctly answer all multiple-choice test questions.

Prior to using the tutor, students completed a questionnaire that assessed software self-efficacy. For each of the 21 unique items of code in the program, the student rated his or her confidence in being able to use the symbol, where *1 = Not confident* and *10 = Totally confident*. This was the measure of software self-efficacy, based on the original work by Bandura (1977) and the subsequent adoption of this approach by researchers in information technology education (Potosky, 2002; Torkzadeh & Van Dyke, 2002). Students also completed the five (Group 1) or 10 (Group 2) rule-based questions, to include the rating of confidence in the accuracy of the answer selected.

After the students completed the tutor, they repeated the software self-efficacy ratings and rule-based questions. The students then rated the overall quality of the tutor, the effectiveness of the tutor in learning Java, and the usability of the tutor interfaces. Each of these scales was a 10-point ordinal scale, where *1 = Poor quality* and *10 = Best quality*.

During the second class period, the author repeated the teaching of the Applet, but this time, a lecture and discussion format was used. This was part of the formative evaluation, and insights gained here are applicable to refining the tutoring system. For Group 1, the second class was two days after the first one. For Group 2, the second class was one week after the first one. These different delays were attributable to the days designated for the classes to meet. During the second class, the author wrote the program on the board and discussed each item and line of code. The students simultaneously entered the program into a UNIX™ text editor. At the completion of the lecture, the students were taught how to compile the program into bytecode. Additionally, the UNIX directory tree and file protections were taught. The HTML file then was taught, and the students ran the Applet on the Web. The students then repeated the questionnaire assessing software self-efficacy and rule-based learning. All of this instructional material, however, was also presented on the Web as part of the ancillary material supporting the individualized tutor.

RESULTS

Figure 9 presents boxplots of total correct rule-based answers by all students across the three assessment occasions: pre-tutor, post-tutor, and post-lecture. The figure shows graphically that the median value increased over the three occasions, and a Friedman test was significant for Group 1 ($\chi^2 = 19.58$, df = 2, $p < 0.001$) and for Group 2 ($\chi^2 = 10.21$, df = 2, $p < 0.001$). The figure also shows that the most pronounced increase occurred between the pre-tutor and post-tutor occasions, in comparison to the post-tutor and post-lecture occasions. For Group 1, a comparison of the means of the differences for all 12 students between pre-tutor and post-tutor totals (Mean = 2.3) with post-tutor and post-lecture totals (Mean = 0.3) was significant t(17.5) for unequal variances = 4.24, $p = 0.001$. For Group 2, a comparison of the means of the differences for all 12 students between pre-tutor and post-tutor totals (Mean = 1.5) with post-tutor and post-lecture totals (Mean = 0.7) was not significant, t(16.9) for unequal variances = 1.12, $p > 0.10$).

Figure 9. Boxplots of total correct rule-based answers by all students across the three assessment occasions: pre-tutor, post-tutor, and post-lecture. The triangle is an extreme value.

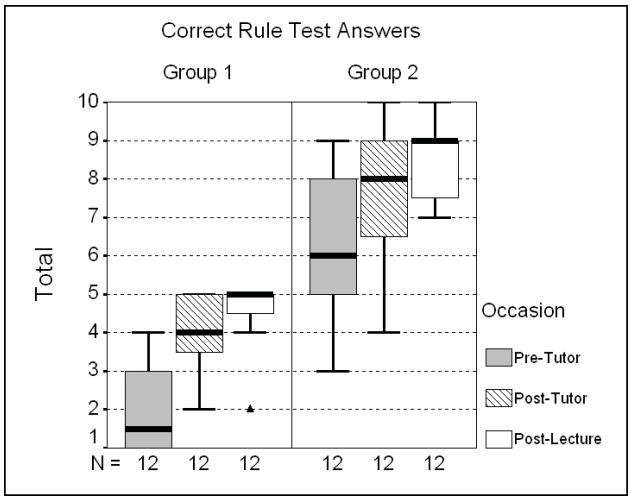


Figure 10. Boxplots of median confidence ratings for Right and Wrong answers for pre-tutor (Pre), post-tutor (Post), and post-lecture (Lecture) assessment occasions. Data for the Lecture occasion were not obtained for one student in Group 1. Circles are outliers, and triangles are extreme values.

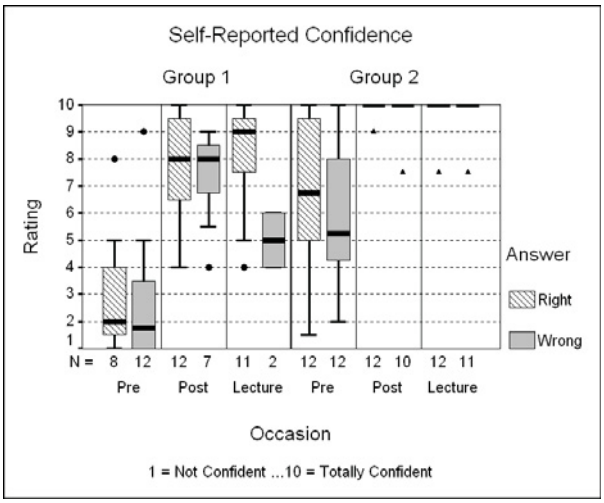


Figure 10 presents boxplots of median confidence ratings for Right (R) and Wrong (W) answers for pre-tutor (Pre), post-tutor (Post), and post-lecture (Lecture) assessment occasions for Group 1 and Group 2. Values in the boxplots are based upon the collection of median ratings for each student for R and W answers across the three occasions. For Group 1, one student did not report data for the post-lecture assessment, and $n = 11$.

For Group 1, K-W comparisons between R and W ratings were not significant within pre-tutor ($\chi^2 = 1.07$, df = 1, $p > 0.10$) and post-tutor occasions ($\chi^2 = 0.16$, df = 1, $p > 0.10$). Accordingly, data were combined for R and W within each occasion, and a K-W comparison between pre-tutor and post-

tutor ratings was significant (χ^2 = 19.68, df = 1, p < 0.001). Too few medians were present in the W category for the post-lecture assessment for a meaningful comparison using those data.

For Group 2, K-W comparisons between R and W ratings were not significant within pre-tutor (χ^2 = 0.62, df = 1, p > 0.10) and post-tutor occasions (χ^2 = 0.11, df = 1, p > 0.10). Accordingly, data were combined for R and W within each occasion, and a K-W comparison between pre-tutor and post-tutor ratings was significant (χ^2 = 23.27, df = 1, p < 0.001). A comparison with the post-lecture outcomes was not undertaken because a rating ceiling (i.e., median = 10) was graphically apparent on the post-tutor occasion.

These ratings show the students' insensitivity in monitoring their own learning. Since the null hypothesis of no difference in confidence ratings between R and W answers could not be rejected for Group 1 and Group 2, the learners did not know, perhaps, that their learning was incomplete. Since self-regulation of learning is an important skill (Veenman, Prins, & Elshout, 2002; Young, 1996; Zimmerman, 1994), how to achieve this outcome within the context of the present tutoring system warrants consideration as this teaching technology continues to mature.

Figure 11 presents boxplots of self-reports of software self-efficacy by all students across the three assessment occasions: pre-tutor (Cronbach's alpha = 0.97 for Group 1 and 0.97 for Group 2, post-tutor (Cronbach's alpha = 0.98 for Group 1 and 0.98 for Group 2), and post-lecture (Cronbach's alpha = 0.98 for Group 1 and 0.98 for Group 2). For Group 1, the figure shows graphically that the median value increased over the three occasions, and a Friedman test was significant (χ^2 = 21.38, df = 2, p < 0.001). The figure also shows that the most pronounced increase occurred between the pre-tutor and post-tutor occasions compared to the post-tutor and post-lecture occasions. A comparison of the means of the differences for all 12 students between pre-tutor and post-tutor ratings (Mean = 5.2) with post-tutor and post-lecture ratings (Mean = 1.0) was significant, t(17.3) for unequal variances = 4.80, p < 0.001.

For Group 2, Figure 11 shows graphically that the median value increased between the pre-tutor and post-tutor occasions, and it reached the ceiling on that latter occasion. A Friedman test

Figure 11. Boxplots of self-reports of software self-efficacy by all students across the three assessment occasions. Circles are outliers, and triangles are extreme values.

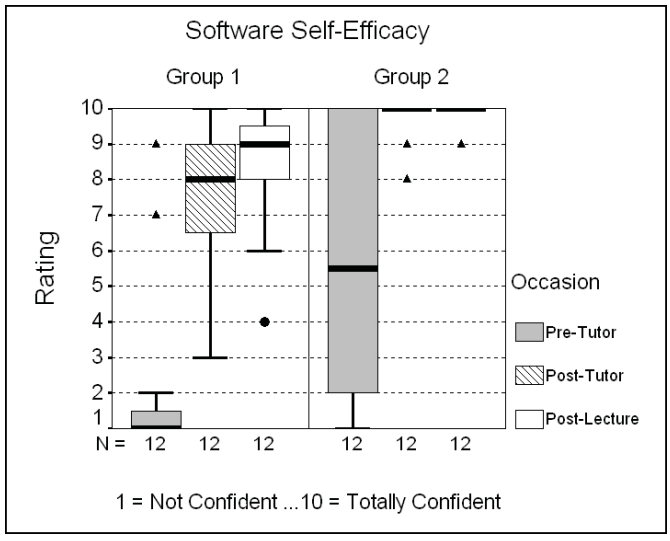
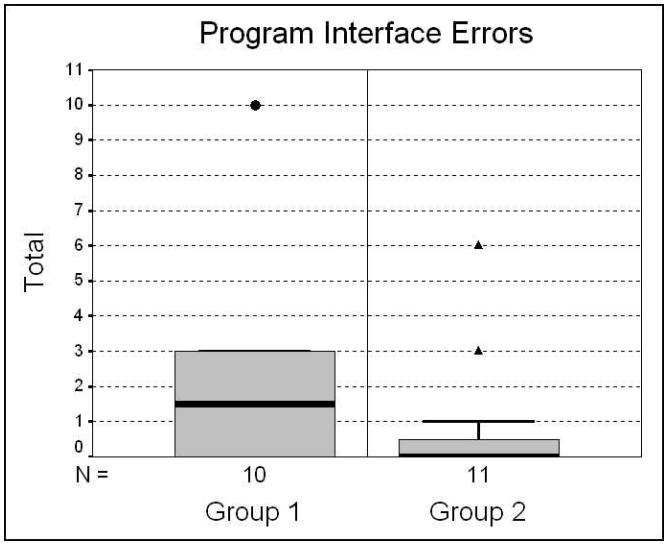


Figure 12. Boxplots of errors on the program interface requiring the input of the entire Java program. A single error reflected any and all errors that were present when the code was submitted for evaluation. An error, then, reflected an incorrect attempt to write the code correctly. The data records for two students in Group 1 and one student in Group 2 were not usable. Circles are outliers, and triangles are extreme values.



was significant ($\chi^2 = 15.44$, df = 2, $p < 0.001$). A comparison of the means of the differences, for all 12 subjects between pre-tutor and post-tutor ratings (Mean = 4.2) with post-tutor and post-lecture ratings (Mean = 0.1) was significant, t(11.1) for unequal variances = 3.96, p = 0.002.

Figure 12 presents boxplots of errors on the final tutor interface, the program interface, which required writing the Java code (see Figure 8). When the student submitted the code for evaluation, one or more errors anywhere in the code produced a pop-up window displaying the correct code. It was that event that was counted as an error. When that window was closed by the student, the program interface was cleared, and the student again entered the code from memory. The tutor is designed as a series of applets, and the performance data are generated and saved automatically for each student. The data for two students in Group 1 and one student in Group 2 were corrupted, thereby reducing the number of students to 10 and 11, respectively, for this analysis. Figure 12 shows that the median errors for students in Group 1 was higher than for Group 2, and a K-W test showed that the difference between the groups was marginally significant ($\chi^2 = 2.82$, df = 1, $p < 0.10$).

Figure 1(b) shows post-tutor ratings on the Overall, Learning Effectiveness, and Usability scales. For both groups, all medians were nine or higher, with Group 1 showing a median of 10 for the Overall rating. A K-W test showed that the difference between Group 1 and Group 2 ratings on the Overall scale was significant ($\chi^2 = 4.67$, df = 1, $p < 0.05$). The extreme value (i.e., rating = 1) present for the Overall and Learning Effectiveness scales was reported by a computer science major who had a different history of computer programming compared to the other students. The tutor is best received by students with minimal

background in programming, and more advanced students, even those with no experience in Java, perhaps should be excused from using such a tutor as the one presented here. Otherwise, the data show that the tutor is well received by graduate and undergraduate students who are taking this course.

DISCUSSION

The results of this study provide further evidence of the effectiveness of programmed instruction presented as an individualized distance learning system in technology education. Students with varying backgrounds all achieved a common outcome of mastering a simple Java applet as the first technical exercise in a programming course. Students not only achieved tested mastery on the syntax and semantics of the code and the capability to recite the code, but they also demonstrated meaningful learning as evidenced by improvements on rule-based questions that required far transfer knowledge. Although studying and memorizing code may well be valuable to achieving facility using a new set of symbols, at the very least the evidence here, which indicates that frames of information and corresponding tests of understanding and retention may be crafted to promote the acquisition of problem-solving skills, supports the more general usefulness of the programmed instruction tutoring system in promoting cognitive skill development.

The present study, then, extends the generality of our previous investigation (Emurian, 2005) to a situation that involved more rule questions with two new groups of students. This shows the reliability of the outcomes over systematic replications under actual classroom conditions. The merits of this design-based research tactic for studying learning in context, which involves students in a classroom, have been discussed recently in a special issue of *Educational Psychologist* (Special Issue, 2004).

The results are consistent with recommendations for evaluating the effectiveness of science, technology, engineering, and mathematics (STEM) instruction: "Provide experiences for students to develop functional understanding. These [exemplary] programs place emphasis on students' understanding of science concepts and ability to apply these concepts to new situations" (McCray, DeHaan & Schuck, 2003, p. 30). Having demonstrated the far transfer consequences of using the tutoring system as an experience that fosters understanding, research in this area can focus on the refinement of the instructional information based upon a relational-frame theory of cognition (Hayes, Fox, Gifford, Wilson, Barnes-Holmes & Healy, 2001) that might be anticipated to potentiate such meaningful learning outcomes of programmed instruction.

For example, answering rule-question number eight correctly required a combinatorial entailment of the information presented in the corresponding explanation in the tutor (Hayes et al., 2001). How to optimize such frames to enhance functional understanding is an important consideration for future research in this area of individualized instructional design applicable to distance learning technologies. In that regard, designers of prose for textbook learning recommend the use of signaling, adjunct questions, and advance organizers as techniques for enhancing a learner's understanding of the material (Mayer, 2002). Unfortunately, perhaps, the research leading to such a recommendation typically is based only on single and time-limited episodes of studying across various experimental treatments. That strategy reveals little about the value of actual studying behavior and resulting understanding, because the process of studying until mastery has occurred is one hallmark of a high-achieving, self-regulated student (Eilam & Aharon, 2003; Pintrich, 2003).

How to make a learning process leading to mastery available to all students should be the goal of instructional design. Equally unfortu-

nate, moreover, is the entrenched elitism of our educational system that continues to undervalue teaching effectiveness.

At present, faculty members are likely to face significant disincentives to learn new teaching approaches and reformulate an introductory course: it requires a large investment of time, it is a distraction from the focus on research, and their investment may not be rewarded. (McCray, DeHaan & Schuck, 2003, p. 51)

There, then, are barriers to implementing effective instruction that is directed toward the achievement of the individual learner.

The importance of the ratings of software self-efficacy is to be understood in terms of the impact of the learning experience on the students' motivation to continue their studies. Given the frequently expressed concern that women and minority groups avoid or withdraw from STEM disciplines (Emurian, 2004), it is encouraging to observe that at least some instructional tactics, such as programmed instruction, may be helpful to provide both skill and confidence in students who initially are interested in a discipline, but whose lack of preparation may sometimes result in demoralization to the point of avoiding or withdrawing from continued study. The essence of effective automated tutoring is to provide a set of experiences that gives all students the skill and confidence to manage their own learning effectively without regard to content and without the continued support of a tutoring system.

The ratings of confidence in the rule-based answers, however, present a more complex interpretative challenge. There was insufficient evidence to suggest that students within either group were able to distinguish between answers that were correct and answers that were incorrect, at least based upon the students' confidence ratings. This was evident, moreover, even within the context of a general improvement in performance between pre-tutor and post-tutor occasions. As noted previously, self-regulation of learning (learning to learn) is an important skill, and these data suggest that without feedback for correct and incorrect answers, learners may not be able to recognize the extent of their own competencies and deficiencies. Although an experimental analysis of rule-governed performance may require obtaining information under the present set of conditions, a classroom application requires feedback (i.e., knowledge of results) for a student's performance errors, as was the case with the assessment tools that were embedded within the tutor itself (Locke, Chah, Harrison, & Lustgarten, 1989; Locke & Lantham, 2002). As indicated by McCray et al. (2003), STEM students need to know when they do not understand, and they need the experiences that lead to understanding a body of material at a deep level. These confidence ratings, then, may reveal a shortcoming in the tutor design that was not evident by a simple tally of errors.

In our series of studies, we make an attempt to document individual differences among subjects and groups only to show the generality of the tutoring system to have a positive impact on learners with different backgrounds, preparatory experiences, and evaluations of the tutor. In the present study, for example, the graduate students as a group were somewhat older than the undergraduates, but on measures of previous courses taken and Java experience, the undergraduates and graduates were not found to differ. Furthermore, there were more female students in the graduate course than in the undergraduate course. The number of errors made on the program interface was larger for the graduate students than for the undergraduates, and the graduate students gave higher overall ratings to the tutoring system compared to the undergraduates. Despite these differences, all students exited the tutoring system with the same level of knowledge and skill, at least with respect to the targeted learning objectives, because the system was designed to achieve that outcome by individualized distance learning. Had the system been designed to require the understanding of

the concepts tested by the rule-based questions that were administered, all students would have exited the tutor with a history of answering all rule-based questions accurately.

Fostering constructive metacognitive processes and supporting individual differences in ability constitute the foundation of effective automated tutoring (Cuevas, Fiore, Bowers & Salas, 2004). In addition to programmed instruction, technology education also would benefit from a consideration of the use of direct instruction (Watkins & Slocum, 2004), precision teaching (Chiesa & Robertson, 2000), and interteaching (Boyce & Hineline, 2002), as those tactics facilitate task mastery at the level of the individual learner. Moreover, the learning history generated by the programmed instruction tutoring system is intended to free our students from future dependence on such automated instruction. With the ultimate goal of producing self-directed learners, our overall strategy in the course is gradually to withdraw structured support over the semester while teaching students to seek out information that is required to solve increasingly more difficult programming problems. Other teaching tactics, such as the Online Problem-Based Model of Learning Java (Tsang & Chan, 2004), have an important role to play in providing subsequent experiences that promote such self-directed learning. Our entire course, not just the Java tutor, is freely available on the Web[5], and our approach can be observed, experienced, and adopted by others.

As discussed elsewhere (Emurian & Durham, 2003), much of the literature in teaching computer programming addresses this challenge as though the skill of computer programming requires a unique teaching technology. The research also is directed toward groups of students rather than to the individual learner. Our approach is different. We assume that learning to write computer programs falls within the scope of training in general (Salas & Cannon-Bowers, 2001) and rule-governed learning in particular (Hayes, 1989). We also assume that a teaching technology only can be rationally developed and applied when it is directed toward the achievement of a criterion of mastery by each and every student.

In that latter regard, research and interventions that are based on null hypothesis refutations of average performance differences between and among treatment groups by definition accept at least some deficient student performance as an outcome. It is encouraging, then, to see the emergence of more achievement-oriented research based on pre-training and post-training comparisons in one group of learners (Potosky, 2002; Torkzadeh & Van Dyke, 2002) in contrast to between-subjects comparisons. As indicated by Sackett and Mullen (1993), it may be more important to an organization to know that an instructional intervention will be successful for all learners than it is to know that average performances between and among groups show differences that are statistically significant. Null hypothesis research typically is focused on effect size differences across treatment conditions, and that approach has recently been characterized by one education scholar as an "old fashioned experimental 'horse-race' design" (Mayer, 2004, p. 16).

Although some proponents of education research continue to advocate that latter model of randomized designs (Towne & Hilton, 2004), how a shift away from null hypothesis evaluation toward alternative methodologies might be undertaken has been discussed elsewhere (Emurian, 2005). In fact, the process of systematic replication used here (Sidman, 1960) is consistent with Design Principle 5 presented in the Shavelson and Towne (2002) report: "Replication and generalization strengthen and clarify the limits of scientific conjectures and theories" (p. 70). Once a behavioral teaching strategy has been demonstrated to be effective for all students, experimental evaluations then might be undertaken to determine the optimal parameters of the instructional system, but only after the initial design has been demonstrated to be effective at the level of the individual learner.

It is an unfortunate irony, however, that educational research sometimes is perceived as less valuable than other areas of research, at least within the social sciences (Sternberg & Lyon, 2002). The irony comes from the obvious importance of education and of knowing and applying the conditions that will help students to learn best throughout the life span. In that regard, research in how best to teach computer programming typically is characterized by comparing average performance among several groups of learners, where each group is exposed to a somewhat different instructional condition. The literature is filled with scores of such studies ranging from early evaluations of conditional constructions (Sime, Green, & Guest, 1973) through LOGO programming (Pea & Kurland, 1984) to levels of graphical support in teaching UNIX™(Sohn & Doane, 1997), classroom tactics for teaching computer programming (Mayer, 1988), and structured exercises and guided exploration in learning the Hypercard™ program (Wiedenbeck & Zila, 1997). That strategy inherently is flawed, because it tacitly accepts the outcome that not all students will achieve mastery even in the group with the best average performance. When it comes to learning Java, the focus on the individual learner is acknowledged by textbook authors such as Deitel and Deitel (1999): "All learners initially learn how to program by mimicking what other programmers have done before them" (p. 38). As distance-learning technologies continue to mature, so will the requirement for the design of individualized instruction rather than group-oriented instruction suitable for implementation with such technologies.

The motivation for the strategy of between-subjects comparisons, of course, is to determine the best instructional tactic, but the outcomes of the research are useful practically only when the time and resources for instructional delivery or for studying are constrained for all students. Such constraints have nothing to do with the process of learning. To adopt the best teaching strategy should mean to respect the right of all students to be given the opportunity to achieve mastery, where opportunity is redefined to mean sustained exposure to the proper conditions of learning to include being taught learning strategies (Namlu, 2003) until achievement has been attained at the level of the individual student. Programmed instruction is a promising tool to foster technical mastery and meaningful learning for students who would benefit from a set of principled interactive instructions that assures the achievement of competency, confidence, and meaningful learning.

The limitations of this work are attributable to it being a series of formative evaluations, rather than an experiment in the traditional sense, although we have argued against such a traditional research methodology. Our interest is effect efficiency and dependability for a common learning outcome for all students rather than effect size differences among alternative instructional treatments producing variable learning outcomes across students. In that regard, the components and parameters of the tutoring system have not been independently validated or isolated for effectiveness. For example, it is not obvious that all learners benefit from the introductory interfaces that only provide facility with the symbols, independent of their meaning. It also is not obvious how much repetition should be built into the tutor. In its current version, for example, the tutor requires two iterations through the line-by-line interface and only one correct input of the entire applet program. Data supporting those parameters are lacking. The choice was made heuristically by our efforts to make the tutor fit into a three-hour period for most students. Since overlearning and repetition are important to retention of a skill (Durham & Emurian, 1998; Swezey & Llaneras, 1997), determining the optimal set of parameters will require further experimental evaluation.

Can students achieve the same outcome with other instructional approaches? Almost certainly, some can. Can students simply study a manual

or textbook and come away with equivalent understanding and skill? Quite possibly, many students can. However, when a teacher wants to be certain that all students achieve a targeted level of performance with a Java applet, the Web-based programmed instruction tutoring system is dependable in producing that outcome, and it is generally well received by students. The individualized tutor will continue to play an important role in our offerings to Information Systems majors, whether as a component in a face-to-face course or within our online distance-education programs.

REFERENCES

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review, 84*, 191-215.

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn? A taxonomy for far transfer. *Psychological Bulletin, 128*, 612-637.

Boyce, T. E., & Hineline, P. N. (2002). Interteaching: A strategy for enhancing the user-friendliness of behavioral arrangements in the college classroom. *The Behavior Analyst, 25*, 215-226.

Chiesa, M., & Robertson, A. (2000). Precision teaching and fluency training: Making math easier for pupils and teachers. *Educational Psychology in Practice, 16*(3), 297-310.

Cuevas, H. M., Fiore, S. M., Bowers, C. A., & Salas, E. (2004). Fostering constructive cognitive and metacognitive activity in computer-based complex task training environments. *Computers in Human Behavior, 20*(2), 225-241.

Deitel, H. M., & Deitel, P. J. (1999). *Java™ How to program* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

Durham, A. G., & Emurian, H. H. (1998). Learning and retention with a menu and a command line interface. *Computers in Human Behavior, 14*, 597-620.

Eilam, B., & Aharon, I. (2003). Student's planning in the process of self-regulated learning. *Contemporary Educational Psychology, 28*(3), 304-334.

Emurian, H. H. (2004). A programmed instruction tutoring system for Java™: Consideration of learning performance and software self-efficacy. *Computers in Human Behavior, 20*(3), 423-459.

Emurian, H. H. (2005). Web-based programmed instruction: Evidence of rule-governed learning. *Computers in Human Behavior, 21*, 893-915.

Emurian, H. H., & Durham, A. G. (2001, May 20-23). A personalized system of instruction for teaching Java. In M. Khosrow-Pour (Ed.), *Managing information Technology in a Global Economy, 2001 Information Resources Management Association International Conference* (pp. 155-160), Toronto, Ontario, Canada. Hershey, PA: Idea Group Publishing.

Emurian, H. H., & Durham, A. G. (2002, May 19-22). Enhanced learning on a programmed instruction tutoring system for JAVA. In M. Khosrow-Pour (Ed.), *Issues & Trends of Information Technology Management in Contemporary Organizations, 2002 Information Resources Management Association International Conference* (pp. 205-208), Seattle, Washington, USA. Hershey, PA: Idea Group Publishing.

Emurian, H. H., & Durham, A. G. (2003). Computer-based tutoring systems: A behavioral approach. In J. A. Jacko, & A. Sears (Eds.), *Handbook of human-computer interaction* (pp. 677-697). Mahwah, NJ: Lawrence Erlbaum & Associates.

Emurian, H. H., Hu, X., Wang, J., & Durham, A. G. (2000). Learning Java: A programmed

instruction approach using applets. *Computers in Human Behavior, 16*, 395-422.

Emurian, H. H., Wang, J., & Durham, A. G. (2003). Analysis of learner performance on a tutoring system for Java. In T. McGill (Ed.), *Current issues in IT education* (pp. 46-76). Hershey, PA: IRM Press.

Fox, M. A., & Hackerman, N. (2003). *Evaluating and improving undergraduate teaching in science, technology, engineering, and mathematics*. Washington, DC: The National Academies Press.

Greer, R. D. (2002). *Designing teaching strategies: An applied behavior analysis systems approach*. New York: Academic Press.

Greer, R. D., & McDonough, S. H. (1999). Is the learn unit a fundamental measure of pedagogy? *The Behavior Analyst, 22*, 5-16.

Harley, H. D., Seals, C. D., & Rosson, M. B. (1998, Winter). A formative evaluation of scenario-based tools for learning object-oriented design. *ACM Crossroads*, 1-5. Retrieved April 6, 2004, from http://www.acm.org/crossroads.xrds5-1/eval.html

Hayes, S. C. (1989). *Rule-governed behavior: Cognition, contingencies, and instructional control*. New York: Plenum Press.

Hayes, S. C., Fox, E., Gifford, E. V., Wilson, K. G., Barnes-Holmes, D., & Healy, O. (2001). Derived relational responding as learned behavior. In S. C. Hayes, D. Barnes-Holmes & B. Roche (Eds.), *Relational frame theory: A post-skinnerian account of human language and cognition*. New York: Kluwer Academic.

Hoadley, C. M. (2004). Methodological alignment in design-based research. *Educational Psychologist, 39*(4), 203-212.

Kudadjie-Gyamfi, E., & Rachlin, H. (2002). Rule-governed versus contingency-governed behavior in a self-control task: Effects of changes in contingencies. *Behavioral Processes, 57*(1), 29-35.

Locke, E. A., Chah, D., Harrison, S., & Lustgarten, N. (1989). Separating the effects of goal specificity from goal level. *Organizational Behavior and Human Performance, 43*, 270-287.

Locke, E. A., & Latham, G. P. (2002). Building a practically useful theory of goal setting and task motivation. *American Psychologist, 57*(9), 705-717.

Maxwell, S. E., & Delaney, H. D. (2000). *Designing experiments and analyzing data*. Mahwah, NJ: Lawrence Erlbaum Associates.

Mayer, R. E. (Ed.). (1988). *Teaching and learning computer programming*. Hillsdale, NJ: Erlbaum.

Mayer, R. E. (2002). *The promise of educational psychology. Volume II. Teaching for meaningful learning*. Upper Saddle River, NJ: Pearson Education.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction. *American Psychologist, 59*(1), 14-19.

McCray, R. A., DeHaan, R. L., & Schuck, J. A. (Eds.). (2003). *Improving undergraduate instruction in science, technology, engineering, and mathematics*. Washington, DC: The National Academies Press.

Namlu, A. G. (2003). The effect of learning strategy on computer anxiety. *Computers in Human Behavior, 19*, 565-578.

Novick, L. R. (1990). Representational transfer in problem solving. *Psychological Science, 1*, 128-132.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137-168.

Pintrich, P. R. (2003). A motivational science perspective on the role of student motivation in learning and teaching contexts. *Journal of Educational Psychology, 95*(4), 667-686.

Potosky, D. (2002). A field study of computer efficacy beliefs as an outcome of training: The role of computer playfulness, computer knowledge, and performance during training. *Computers in Human Behavior, 18*, 214-255.

Sackett, P. R., & Mullen, E. J. (1993). Beyond formal experimental design: Towards an expanded view of the training evaluation process. *Personnel Psychology, 46*, 613-627.

Salas, E., & Cannon-Bowers, J. A. (2001). The science of training: A decade of progress. *Annual Review of Psychology, 52*, 471-499.

Shavelson, R. J., & Towne, L. (Eds.). (2002). *Scientific research in education*. Washington, DC: The National Academies Press.

Sidman, M. (1960). *Tactics of scientific research.* New York: Basic Books.

Sime, M. E., Green, T. R. G., & Guest, D. J. (1973). Psychological evaluation of two conditional constructions used in computer languages. *International Journal of Man-Machine Studies, 5*, 105-113.

Sohn, Y. W., & Doane, S. M. (1997). Cognitive constraints on computer problem-solving skills. *Journal of Experimental Psychology: Applied, 3*, 288-312.

Special Issue. (2004). Design-based research methods for studying learning in context. *Educational Psychologist, 39*(4), 199-260.

Sternberg, R. J., & Lyon, G. R. (2002). Making a difference to education: Will psychology pass up the chance? *Monitor on Psychology, 33*, 76. Retrieved April 6, 2004, from http://www.apa.org/monitor/julaug02/difference.html

Swezey, R. W., & Llaneras, R. E. (1997). Models in training and instruction. In G. Salvendy (Ed.), *Handbook of human factors and ergonomics* (pp. 514-577). New York: Wiley.

Torkzadeh, G., & Van Dyke, T. P. (2002). Effects of training on Internet self-efficacy and computer user attitudes. *Computers in Human Behavior, 18*, 479-494.

Towne, L., & Hilton, M. (Eds.). (2004). *Implementing randomized field trials in education*. Washington, DC: The National Academies Press.

Tsang, A. C. W., & Chan, N. (2004). An online problem-based model for the learning of Java. *Journal of Electronic Commerce in Organizations, 2*(2), 55-64.

Veenman, M. V. J., Prins, F. J., & Elshout, J. J. (2002). Initial inductive learning in a complex computer simulated environment: The role of metacognitive skills and intellectual ability. *Computers in Human Behavior, 18*, 327-241.

Watkins, C. L., & Slocum, T. A. (2004). The components of direct instruction. *Journal of Direct Instruction, 3*(2), 75-110.

Wiedenbeck, S., & Zila, P. L. (1997). Hands-on practice in learning to use software: A comparison of exercise, exploration, and combined formats. *ACM Transactions on Computer-Human Interaction, 4*(2), 169-196.

Young, J. D. (1996). The effect of self-regulated learning strategies on performance in learner controlled computer-based instruction. *Educational Technology Research and Development, 44*, 17-27.

Zimmerman, B. J. (1994). Dimensions of academic self-regulation: A conceptual framework for education. In D. H. Schunk, & B. J. Zimmerman (Eds.), *Self-regulation of learning and performance* (pp. 3-21). Hillsdale, NJ: Erlbaum.

ENDNOTES

[1] Unless otherwise noted, the Kruskal-Wallis (K-W) ANOVA by ranks and Friedman tests were used in this investigation, because they are conservative non-parametric tests that are best applied to ordinal and ratio data with small sample sizes (Maxwell & Delaney, 2000). The tests are based upon a chi-square (χ^2) distribution.

[2] Data for number of prior programming courses taken were missing for one student in Group 1.

[3] http://nasa1.ifsm.umbc.edu/learnJava/tutorLinks/TutorLinks.html

[4] http://nasa1.ifsm.umbc.edu/learnJava/savetext/TutorContent.pdf

[5] http://nasa1.ifsm.umbc.edu/IFSM413_613/

APPENDIX A

Following is the Java program that was taught by the tutoring system. The code is arbitrarily organized into 10 lines or rows. The method in Row 8, not needed with the default FlowLayout manager, was presented only to teach the application of a method on an object.

```
Row 1:    import java.applet.Applet;
Row 2:    import java.awt.Label;
Row 3:    public class MyProgram extends Applet {
Row 4:    Label myLabel;
Row 5:    public void init() {
Row 6:    myLabel = new Label("This is my first program.");
Row 7:    add(myLabel);
Row 8:    myLabel.setVisible(true);
Row 9:    }
Row 10:   }
```

APPENDIX B

Rule-Based Questions

1. Which of the following lines most likely would be used to create a shorthand notation for the Frame class, which is built in to Java?

a. import java.awt.frame;
b. import java.awt.Frame.class;
c. import java.awt.Frame;
d. import java.awt.frame.class;

2. Which of the following lines most likely would be used to construct an instance of the Button class?

a. MyButton = new Button(“Hello”);
b. myButton = new Button(“Hello”);
c. myButton = button.class(“Hello”);
d. MyButton = Button(“Hello”);

3. Which of the following lines most likely would be used to add a Checkbox object to a container?

a. Add(myCheckBox);
b. Add(Checkbox);
c. add(Checkbox);
d. add(myCheckBox);

4. Which of the following lines most likely overrides a method that is contained in the Applet class?

a. public void stop(){ lines of Java code here }
b. public void Stop{} { lines of Java code here }
c. Public void Stop() (lines of Java code here)
d. Public void stop() { lines of Java code here }

5. Which of the following sequences is correct?

a. declare a TextField object, construct a TextField object, add a TextField object to a container
b. construct a TextField object, declare a TextField object, add a TextField object to a container
c. declare a TextField object, add a TextField object to a container, construct a TextField object
d. add a TextField object to a container, declare a TextField object, construct a TextField object

6. Given the line, **public class MyTextArea extends TextArea {**, which of the following statements is correct?

a. TextArea is a subclass of MyTextArea.
b. MyTextArea is a superclass of the extends class.
c. TextArea is a superclass of MyTextArea.
d. MyTextArea is a subclass of the Text class.

7. Which one of the lines below declares myJFrame as a potential instance of the JFrame class?

a. myJFrame extends JFrame.
b. JFrame myJFrame:
c. myJFrame JFrame;
d. JFrame myJFrame;

8. Given the following code: **public class MyJFrame extends JFrame { …**
which one of the below would be the name of the Java file that contains this program?

a. MyJFrame.Java
b. MYJFrame.java
c. MyJFrame.java
d. MyJFrame.doc

9. Which of the following lines most likely would add a JTextField object to a JPanel object?

a. JPanel.add(JTextField);
b. JPanel.add(myJTextField);
c. myJPanel.add(JTextField);
d. myJPanel2.add(myJTextField2);

10. A Java Applet program has two methods written in the class. The methods are not nested. What is the minimum number of braces, { and } added together, that are needed for this program?

a. 9
b. 6
c. 3
d. 4

This work was previously published in International Journal of Distance Education Technologies, Vol. 4, Issue 2, edited by T. K. Shih, pp. 10-30, copyright 2006 by IGI Publishing, formerly known as Idea Group Publishing (an imprint of IGI Global).