Web-based programmed instruction: evidence of rule-governed learning

Henry H. Emurian *

Information Systems Department, College of Engineering and Information Technology, UMBC 1000
Hilltop Circle, Baltimore, MD 21250, USA
Available online 8 April 2004

Abstract

Seventeen graduate students in two classes worked on a web-based programmed instruction tutoring system as the first technical exercise in a Java™ programming course. The system taught a simple Java applet to display a text string in a browser window on the world wide web. Students completed tests of near transfer and far transfer before and after using the tutor and again after a lecture on the material. The results showed that performance improved over pre-tutor baseline on all assessments, to include the far transfer test, which required integrating information in the tutor into a rule to apply to solve a novel programming problem not explicitly taught in the tutor. Software self-efficacy also increased across four assessment occasions. These data show that programmed instruction can produce problem solving skills and can foster student confidence, based upon the documented mastery of fundamental material in a technical domain. An investigative approach that follows systematic replication, rather than null hypothesis refutation, may be best suited to assess the impact and dependability of competency-based instructional systems.

© 2004 Elsevier Ltd. All rights reserved.

Keywords: Programmed instruction; Rule-governed behavior; Java training; Software self-efficacy

1. Introduction

At the end of a chapter, a textbook in Java™ may present terminology to learn, self-review questions to answer, and coding projects to perform (e.g., Deitel & Deitel,
The information necessary for students to complete those exercises, assigned or discretionary, is contained within the chapter. It is assumed, perhaps, that students possess the study skills to reach a criterion of mastery as evidenced by completion of the study aids. That assumption may not be warranted because not all students possess equivalent skill in self-regulated behavior (Morin & Latham, 2000; Schunk, 2000; Skinner, 1968; Veenman, Prins, & Elshout, 2002; Young, 1996; Zimmerman, 1994). This is supported by the observation that when students are simply exposed to information and asked to perform optimally, they may not always do so because there is often no specific external referent of mastery (Locke, Chah, Harrison, & Lustgarten, 1989; Locke & Latham, 1990, 2002). That is, students are not informed about how to study and how to identify when they have achieved a performance criterion or steady state with the knowledge domain.

Programmed instruction may be helpful when students are trying to learn a new knowledge domain because it provides study discipline, as structured rehearsal, as well as the necessary referent of achievement. The essential design features of programmed instruction are synthesized as follows: (1) comprehensibility of each frame of information; (2) tested effectiveness of a set of frames; (3) sequential frames; (4) self-correcting tests; (5) encouragement for learning; (6) diagnosis of misunderstandings; (7) adaptations to errors by hints, prompts, and suggestions; (8) learner constructed responses based on recall; (9) immediate feedback; (10) successive approximations to a terminal objective; and (11) learner-paced progress (Scriven, 1969; Skinner, 1958; Vargas & Vargas, 1991). A program of study consists of a learner’s interaction with many instructional frames designed to promote the cumulative achievement of a demonstrable set of competencies for the individual learner (e.g., Anger, Rohlman, Reed, Lundeen, & Eckerman, 2001; Bitzer, Braunfield, & Lichtenberger, 1962; Coulson, 1962; Holland, 1960; Holland & Skinner, 1961; Kritch & Bostow, 1998). Our current work reflects the application of this instructional technology to a technical knowledge domain involving understanding and using symbols in a programming language. Greer (2002) presents a synthesis of behavior principles applied to teaching strategies in which programmed instruction is but one tactic that may be adopted within that general framework, and its adoption here is a reflection of the context and objectives of our teaching.

The purpose of the present paper, then, is to support and extend our previous work on a programmed instruction tutoring system for learning Java (Emurian, 2004; Emurian & Durham, 2001, 2002, 2003; Emurian, Wang, & Durham, 2003) by presenting evidence that students can acquire general rules of Java programming and can apply those rules to problems not covered explicitly in the tutor. In a traditional formulation of learning, this is a transfer of training problem (Barnett & Ceci, 2002). In behavior analysis this is a rule-governed problem (Hayes, 1989). The two approaches seek a common outcome, and this outcome is evaluated in the present work.

The importance of providing this evidence is to be understood in terms of addressing criticism directed to “rote memorization” in science and mathematics education (e.g., Bransford, Brown, & Cocking, 2000), although programmed instruction has been demonstrated to be effective in mastering general concepts such as conductance (Deterline, 1962). Moreover, there is increasing criticism of
premature constructivism in K-12 education, as shown by Project Follow Through (1996) and by the ongoing New York City Open Logical Debate on Mathematics Education Reform (Braams & Carson, ND). There is interplay, then, between the mastery of details and the mastery of problem-solving strategies, and our tutoring system is intended to teach both facts and rules. What has yet to be investigated, however, is the extent of rule-governed performance that results from using the programmed instruction tutoring system.

The design of the web-based tutoring system1 and examples of the tutor stages and interfaces are presented elsewhere, along with data showing performance during a typical 3-h learning session by undergraduate and graduate students (Emurian, 2004; Emurian & Durham, 2003; Emurian, Hu, Wang, & Durham, 2000; Emurian et al., 2003). The same ten-row applet, consisting of up to 32 items of code, has been used throughout this series of investigations, and the prior work may be consulted to view the code. The code is also displayed in the introductory instructions to learners in the tutor that is available on the Web. It is also presented in Appendix B, which is explained below.

The tutoring system has been used for several semesters as the first technical training exercise in a graduate and undergraduate Java course. The course content and instructional style are directed to information systems majors, who generally have less interest and need for computer programming skills than do some other majors such as computer science. How the use of the tutor fits into our general pedagogical framework for these students is discussed in our previous work.

In our prior studies, assessment of learning was based in part upon a series of multiple-choice tests that were presented after each frame in the tutor. This approach is similar to Crowder (1962), who also used multiple-choice tests embedded within programmed instruction as a test of mastery of a frame. In the present tutor, tests were based on facts pertaining to a particular item of code that constituted a component of the Java applet to display a text string in a browser window on the world wide web. Learning was also demonstrated by accurate input of the code, by recall, in a successive approximation to mastering the complete program in incremental steps. Since each individual frame often contained both facts about a particular item together with generalizable information, or a rule about the item, it is important to know the extent to which students who use the tutor also acquire general rules of programming that can be applied to problems not explicitly taught by the tutor. Such evidence, should it exist, would be a beneficial by-product of progressing through the tutor experience.

Against that background, this study tested the students’ competency to answer questions about Java code when the answer required generalizing and integrating information presented in the individual frames. This was accomplished by administering an assessment exercise on three separate occasions, when the first assessment (i.e., baseline) occurred prior to using the tutor. Additionally, as a

---

1 The tutoring system is freely accessible on the web: URL http://nasa1.ifsm.umbc.edu/learnJava/v6/front/. The source code is freely available by contacting the author.
further test of the effectiveness of the tutor, the multiple-choice test that was presented as a component of learning each of the 10 rows of code was also administered at the same time as the rule test. This approach yielded two different baseline assessments of knowledge prior to the use of the programmed instruction tutoring system.

2. Method

2.1. Participants

Participants were students enrolled in two sections of a graduate course entitled “Graphical User Interface Systems Using Java.” One section met for seven weeks in the summer of 2002 (S), and the other section met for 14 weeks in the fall of 2002 (F). Both sections met 14 times, and each class lasted 2.5 h, except for the first class, which lasted 3 h. For the S class, there were five males and five females (median age = 29.5, range = 24–64). For the F class, there were two males and five females (median age = 28, range = 24–36). The prerequisite for the course was one prior programming course in any procedural or object-oriented language.

2.2. Materials

This study used the Java programmed instruction tutoring system, and the most recent version is explained elsewhere (Emurian, 2004; Emurian & Durham, 2003). The tutoring system teaches a simple Java applet that displays a text string in a browser window on the world wide web. All information contained in the tutoring system, to include instructions, frames, and tests, is available in text documents from the author.

There are 32 frames of information that explain the 21 atomic units of code in the applet being learned, and some units, such as the semi-colon, were duplicated. Below are three of the atomic units of code. Presented for each unit is the information in the frame that pertains to the generative rule that is required to answer question number four in the rule test questions, which are presented in Appendix A. The frame itself has much more information in it than just the rule, and that other information is needed to pass a multiple-choice test on the item of code explained in the frame.

1. **public**
   The public keyword is known as an access modifier in Java because it determines the rules by which a class, method, or variable can be accessed.

2. **void**
   The term void is used here to indicate that the method, in this case the init() method that will follow the term void in this program, does not return any value to the code that invoked or started the method.

3. **init()**
   Inside the Applet class, the init() method has no statements in it. When the programmer uses the init() method in a program and adds statements to it, that is
called overriding the init() method. The general form of the init() method is as follows:

```java
public void init() {
    a line of Java code;
    a line of Java code;
}
```

Also notice that a method has a name that begins with a lower case letter, and it has an argument list inside the parentheses. You know that init() is a method because of those properties. There are some special methods with names that begin with capital letters, and you will learn about these special methods later.

### 2.3. Procedure

Similar to our previous work, the self-report measures below are Likert-type scales, since they were not formed by item-analysis procedures or the method of summative ratings (Aiken, 2000). The scale statements were selected by their face validity, and the rating choices are similar to those found in this literature.

Prior to using the tutor, each student completed a pre-tutor questionnaire that presented two rating scales. The first 5-point rating scale assessed the student’s prior experience with Java, where the scale anchors were 1 = No experience. (I am a novice in Java.) to 5 = Extensive experience. (I am an expert in Java.). The second 5-point rating scale assessed the student’s confidence in being able to use each of the 21 unique Java items to write a Java computer program. The statement to be rated was as follows: “How confident are you that you can use the following symbol to write a Java program”? The scale anchors were 1 = Not at all confident. I do not know how to use the symbol. to 5 = Totally confident. I know how to use the symbol. This second scale was used as the measure of software self-efficacy (SSE). Although Bandura (1997, p. 382) distinguished between non-descript “confidence” and self-efficacy, the reference was to general athletic functioning. The present use is related to programming confidence using a specific instance of Java code.

The students also completed two multiple-choice tests as a pre-tutor baseline. The first test, the rule test (see Appendix A), consisted of four problems, and each problem solution required a synthesis and extension (i.e., “transfer”) of information presented within the individual item frames. The specific information required to answer the rule questions, however, was not presented in the frames. Unlike the multiple-choice tests embedded within the tutor, there was no immediate feedback for an answer and no requirement to repeat a frame-test cycle until the answer selected was correct.

The second test, the row test (see Appendix B), consisted of ten problems, and these problems were exactly the same as were presented on the first pass of the row-by-row tutor interface. In that later interface, if a student could not integrate the previously learned item information to answer a question based on a row of code, there was an optional access to a frame that provided that information. This second test had to be passed correctly within the tutor, but as administered separately for a baseline, this requirement did not apply.
At the conclusion of the 3 h allotted to the tutoring system or whenever a student finished the tutor prior to that time, a post-tutor questionnaire was completed. This questionnaire repeated the above SSE assessment, and three additional 5-point rating scales were also presented. The first scale assessed the student’s overall reaction to the tutor, where the scale anchors were 1 = Totally negative. I did not like the tutor. to 5 = Totally positive. I liked the tutor. The second scale assessed the student’s opinion of the extent to which the tutor helped to learn Java, where the scale anchors were 1 = Totally negative. The tutor did not help me to learn Java. to 5 = Totally positive. The tutor did help me to learn Java. The third scale assessed the usability of the tutor interfaces, where the scale anchors were 1 = Totally negative. The tutor was difficult to use. to 5 = Totally positive. The tutor was easy to use. The rule and row multiple-choice tests were also administered again. The students were then dismissed from the class, and the tutor continued to be available for those students who were motivated to access the tutor outside of class.

During the immediately succeeding class period, which occurred two days later for S and seven days later for F, the instructor discussed the applet code with the students using a lecture format. The students entered the code into a UNIXTM text editor at the time the items were presented and discussed on the board. This repetition of instruction using a different medium was a deliberate tactic to optimize learning. The world wide web directory tree and HTML file were also presented and discussed. The students then compiled the Java code and ran the applet in a browser by accessing the HTML file as a URL on the Web. To foster a collaborative learning environment, the students were encouraged to help each other and to seek help from the instructor as needed. Anecdotal observation suggested that this was the first time that most students had run an applet on the Web.

3. Results

A Kruskal–Wallis “ANOVA by ranks” test was used because of the small sample sizes (Maxwell & Delaney, 2000, p. 703). The test statistic is based on a $\chi^2$ distribution. The self-reported Java experience by the students was as follows for S (median = 1, range = 1–2) and for F (median = 2, range = 1–2). A Kruskal–Wallis comparison between the classes was not significant ($\chi^2 = 0.71, p > 0.10$).

In the S class, all 10 students completed all stages of the tutor by the end of the 3-h period. In the F class, five of the seven students completed all stages. One student was working on the final program interface, in which the entire program was input as a serial stream. The second student was working on the Java item interface. That student’s progress was uncharacteristically slow, and error data were not generated. Accordingly, sixteen of the seventeen students completed all tutor stages that contained instructional frames and corresponding multiple-choice tests. The tutor was available outside of class, but the data collection was limited to the performance during class. All performance measures were automatically recorded by the tutor code, which was written in Java.
Figs. 1 and 2 present boxplots for software self-efficacy over the four assessment occasions for the S and F classes, respectively. For S, the figure shows graphically that the median value progressively increased over the four successive assessment occasions. For F, the median reached asymptote on the post-tutor assessment and remained at that value thereafter. The figure also shows that the median for the F class was higher than the median for the S class across each assessment occasion. Cronbach’s zs for the four successive assessments of software self-efficacy were as follows for S: 0.97, 0.97, 0.98, and 0.98, respectively. For F, the outcomes were as follows: 0.97, 0.97, 0.99, and 0.85, respectively.

To assess the magnitude of the changes over successive occasions, a “difference score, Dij” was computed for each student (Di = 1, n) for the three sets of differences (Dj = 1, 3) obtained over the four successive assessment occasions. For S, a comparison of the D1 and D2 values was significant ($\chi^2 = 12.22, p < 0.01$, Bonferroni corrected), but the comparison of D2 and D3 values was not significant.

Fig. 1. Boxplots of software self-efficacy ratings over the four assessment occasions for the S class.

Fig. 2. Boxplots of software self-efficacy ratings over the four assessment occasions for the F class. Circles are outliers, and triangles are extreme values.
A test of linear trend across the four assessment occasions, computed by comparing all students’ regression slopes with a population of zeros (Maxwell & Delaney, 2000, p. 580), was significant ($\chi^2 = 16.52, p < 0.01$). For F, a comparison of D1 and D2 values was significant ($\chi^2 = 10.52, p < 0.01$, Bonferroni corrected), but the comparison of D2 and D3 values was not significant ($\chi^2 = 2.71, p > 0.10$). Because one student dropped the course soon after the post-applet assessment, only six sets of scores were available for D3 in the F class. For both S and F classes, the greatest improvement in SSE ratings occurred between the pre-tutor and post-tutor assessments.

Fig. 3 presents total correct rule test answers over the three assessment occasions for all students in the S and F classes, respectively. The data are sorted in ascending order using the pre-tutor outcomes. The student number identifier on the sorted data is retained for comparison on subsequent figures. The bar connects the pre-tutor total with the post-applet total, and the magnitude of the bar is a visual representation of the magnitude of the change for each student.

For both classes, the median number of correct answers was zero for the pre-tutor assessment, but one student in each class answered all four questions correctly prior to using the tutor (S-10 in S and S-7 in F). The figure also shows graphically that the...
most pronounced improvement occurred between the pre-tutor and post-tutor assessments. A Kruskal–Wallis comparison of the two sets of differences between the two successive pairs of assessment occasions (Di1 and Di2) for all students was significant for S ($\chi^2 = 9.30, p < 0.01$) and for F ($\chi^2 = 5.20, p < 0.05$). On the post-tutor assessment, all but three students answered all four questions correctly (S-1 and S-7 in S and S-1 in F). On the post-applet assessment, all but three students answered all questions correctly (S-1 and S-7 in S and S-1 in F). The data also show that two students did not show any improvement in performance until the post-applet assessment (S-1 in S and S-1 in F). The small gains noted by S-1 in F are related to the fact that this student did not complete the item interface during the 3-h period and showed the lowest rule and row test performance over assessment occasions.

Fig. 4 presents total correct row test answers over the three assessment occasions for all students in the S and F classes, respectively. The data are sorted in ascending order using the pre-tutor outcomes. For both classes, the median number of correct answers during the pre-tutor assessment exceeded zero, reaching three for the S class and seven for the F class. The figure also shows graphically that on the post-tutor assessment, performance reached the ceiling of 10 correct answers for all but three students (S-1 and S-6 in S and S-1 in F). During the post-applet assessment, the median number of correct answers was 10 for both classes, although there were

![Fig. 4. Total correct row test answers over the three assessment occasions for all students in the S and F classes, respectively. The data are sorted in ascending order using the pre-tutor outcomes. See text for explanation.](image-url)
several students with less than perfect performance and three instances of a decline in performance (S-4 and S-8 in S and S-5 in F). A Kruskal–Wallis comparison of the two sets of differences between the two successive pairs of assessment occasions (Di1 and Di2) for all students was significant for S ($\chi^2 = 15.00, p < 0.01$) and marginally significant for F ($\chi^2 = 3.40, p < 0.07$). However, a test of linear trend in total correct answers over successive assessment occasions was significant for S ($\chi^2 = 16.45, p < 0.01$) and for F ($\chi^2 = 8.83, p < 0.01$).

Fig. 5 presents boxplots of total tutor errors for the S and F classes. Total tutor errors consisted of the sum of input and test errors across all interfaces from the item familiarity interface through the program interface. The figure shows that the median number of errors was higher for S in comparison to F. However, a Kruskal–Wallis test between the classes was not significant ($\chi^2 = 2.65, p > 0.10$). The figure shows a wide range of total errors that students emitted in completing the tutor. Pearson correlations between total tutor errors by all students and pre-tutor correct row test answers ($r = -0.01$) was not significant ($p > 0.05$). The Pearson correlation between total tutor errors and pre-tutor correct rule test answers ($r = -0.49$) was marginally significant ($p < 0.06$).

Fig. 6 presents boxplots of the total number of programming courses that the students reported taking prior to this course for the S and F classes. The data for one student in the F class was not usable. The figure shows that the median number of courses taken was higher for the F class in comparison to the S class. A Kruskal–Wallis test between the classes was significant ($\chi^2 = 4.90, p < 0.03$). Pearson correlations between the number of programming courses previously taken and total tutor errors by all students ($r = -0.04$), pre-tutor correct rule test answers ($r = -0.11$), and pre-tutor correct row test answers ($r = -0.32$) were not significant ($p > 0.05$).

Fig. 7 presents boxplots of self-reported ratings for students in the S and F classes for the following scales: overall evaluation of the tutor, effectiveness of the tutor in learning Java, and usability of the tutor interfaces. The median value for all six plots is five, the scale ceiling. Kruskal–Wallis comparisons of ratings showed no difference between the S and F classes on any scale. A comparison between all 17 ratings on a...
scale with a “baseline” population of seventeen fours was significant for overall ($\chi^2 = 10.77, p < 0.01$), learning effectiveness ($\chi^2 = 7.19, p < 0.01$), and usability ($\chi^2 = 13.47, p < 0.01$).

4. Discussion

In the following discussion, we use the functional classification of verbal behavior originally provided by Skinner (1957). We refer to the work of Catania (1998), who
summarized many of the important features of that difficult account. We acknowledge as well the potential contributions of an emerging relational frame theory account of language and cognition to guide future research in the presentation of textual information that potentiates the development of rule-governed behavior and equivalence classes (Hayes, Barnes-Holmes, & Roche, 2001a).

This study extended our previous classroom applications of a programmed instruction tutor to show that the tutoring system produced a verbal repertoire more complex than simple intraverbal performances (Catania, 1998, p. 417), which may be acquired by rote memorization and sustained over time with self-echoic and self-intraverbal behavior (Catania, 1998, p. 411). By “more complex” is meant the evidence of generative rule-governed behavior in which a solution to a novel problem could be identified without prior training or instruction on that specific problem or the rule to solve it. In addition to this far transfer performance improvement, with pre-tutor testing as a baseline, students also showed improvement on the near transfer assessment of the objectives of each row of code. All students who completed the tutor did so within the framework of their tested understanding of the 32 items and 10 rows of Java code and of their recall capacity to construct the final program as an error-free serial stream. These near and far transfer effects are similar to those reported by Tudor and Bostow (1991) and Kritch and Bostow (1998), although the post-tutor assessments of application performances in that research were demonstrated in a between-group design.

Self-reported ratings of attitude have been collected in other programmed instruction research (e.g., Kritch & Bostow, 1998), and such autoclitic verbal behavior (Catania, 1998, p. 407), which is based upon the students’ description of the strength of their own behavior, was found useful here as well. Software self-efficacy reports increased over baseline for both S and F classes. The largest change occurred between the pre-tutor and post-tutor assessments for both classes, and the F class students generally showed higher ratings. This observation is consistent with the self-reported number of prior programming courses taken, where the F class reported taking more courses. Additionally, the F class showed fewer median errors on the tutor in comparison to the S class, plausibly assuming that failure to reject the null hypothesis was a Type II error. Taken together, these observations help to reveal the different histories of the students in those two classes as they relate to tutor learning performance leading to a common competency outcome.

The self-reports of the effectiveness of the tutor on the overall, learning, and usability scales all showed a median of five for all students in both classes. The median of five is somewhat higher than the range of the mean ratings across groups (3.43–2.54) reported by Kritch and Bostow (1998) for self-reports of their students’ attitudes about the programmed instruction material on a similar 5-point rating scale. The outliers and extreme value (i.e., low ratings) shown in Fig. 7 by one student in the present study were reported by a student who had eight prior programming courses and previous Java experience. The Java tutor is best suited for novitiate learners, perhaps, who respond favorably to the imposed study discipline and corresponding learning outcome provided by the tutor. This is consistent with the opinion expressed by Deitel and Deitel (1999, p. 38): “All learners initially learn
how to program by mimicking what other programmers have done before them.” The Java tutor is an attempt to bring rigor to that latter approach to learning, and students are at least not unappreciative of the process and outcome.

The analysis of rule-governed behavior plays an important role in behavior analysis because of the prevalence of such stimulus control in so much of our actions in everyday life (Hayes, 1989; Dillenburger, O’Reilly, & Keenan, 1997). A behavioral account of the development of competency in answering the rule questions benefits from an analysis of joint control and rule following as presented by Lowenkron (1999). This researcher provided a behavioral account of the following rule: “When the cake has risen, remove it from the oven.” The account of a listener’s initially hearing the rule through the subsequent removal of the cake, at a later point in time, was based on a consideration of the memory function, the recognition function, and the response function, all operationalized in terms of joint self-echoic and naming tact (Catania, 1998, p. 427) stimulus control, terminating with a performance occasioned by a command, i.e., mand (Catania, 1998, p. 419).

As an example, in the present analysis the rule needed to answer the fourth question in the rule test is as follows:

If the access modifier is public AND the return type is void AND the method name is all lowercase letters AND the method argument list () is empty AND the body of Java statements is within braces {}

Then the method may be the one to override a method in the Applet

This rule was not explicitly taught in the tutor. Some of the identifiers were taught item by item (e.g., public and void), but the method taught was the init() method, which is contained in the Applet class, not the stop() method contained in the correct answer. The general form of the init() method was presented in the tutor, and it was stated that there were other types of special purpose methods that begin with a capital letter. Each item taught by the tutor, then, may be considered to be a rule, such as “If a method returns no value, insert the void keyword between the access modifier and the method name.”

How is the concatenated rule constructed by several separate small rules to influence the selection response at a later time? Self-echoic or self-intraverbal behavior likely did not apply as a memory function during the first test occasion, at least, and the recognition of the correct test alternative as the event specified in the rule could come into play only after the tutor had been used or only by prior experience. It was the case, however, that the form of the init() method was displayed in the tutor for examination, leading to the potential for a type of autoclitic pointing (Lowenkron, 1999). Furthermore, if the statement in the “Then” clause is taken as a mand, where the correct alternative is at least a tact, this occasions the autoclitic self-selection (i.e., “pointing”) response of choosing the correct answer. The account, of course, does require the assumption of conditioned reinforcers to maintain these performances, as suggested by Lowenkron (1999). Related considerations include the prior strength of
the individual elements of the concatenated autoclitic frame “IF..., THEN” acquired during progress through the individual tutor frames (Skinner, 1957, p. 312). This interpretation comes from Vaughan (1989), who provided an account of the history of rule-governed behavior in behavior analysis.

Additionally, a relational frame theory account suggests that both init() and stop() became members of a hierarchical relational frame\(^2\) within the scope of override by a process of combinatorial entailment (Hayes et al., 2001b, p. 30). Since the stop() method was not presented in the tutor, its membership in this hierarchy was plausibly mediated by a rule that related init() and stop() as members of an equivalence class (Carpentier, Smeets, & Barnes-Holmes, 2003; Healy, Barnes-Holmes, & Smeets, 2000; Sidman, 2000). We regard the emergence of an accurate test performance (i.e., selection) by a learner who has never been explicitly trained on a particular instance of a class as non-trivial, although obviously simplistic within the present example. A technology of individualized instruction requires a research-based understanding of the parameters of that acquisition process in furtherance of optimizing the learning outcomes that accrue from a student’s interactions with an automated instructional system.

A student’s knowledge that was transferred to a concatenated rule was acquired by reading the frames. Observing the problems initially in the baseline assessment perhaps occasioned extra attention to the information required to solve them. Kritch and Bostow (1998) reported that their subjects in a “passive” reading condition achieved a mean of more than 56% correct answers on a 34-item post-test requiring the production of authoring language code. Reading instructional text, then, is obviously associated with learning and retention, although the latter investigators also showed that the requirement for constructed responses during learning enhanced post-tutor performance in comparison to reading the frames only. When an instructional intervention produces both retention and transfer, Mayer (2002) refers to such an outcome as “meaningful learning” (p. 3).

The application of behavior analytic principles to account for a rule-governed performance, as demonstrated by Lowenkron (1999) and discussed by Vaughan (1989) and others (e.g., Kerr & Keenan, 1997), may prove helpful in interpreting a learning experience for students using programmed instruction. For example, Crosbie and Kelly (1994) and Kelly and Crosbie (1997) suggested that a delay interval between successive programmed instruction frames might have impacted subsequently enhanced test performance by providing an opportunity for rehearsing the presented information, a type of self-echoic or self-intraverbal behavior. A similar account may be applied to the study by Kritch and Bostow (1998) in which the response was a written or silent production.

A behavior steady state that indicates a learner’s readiness to transition from one instructional frame to another, determined by accurate test performance or an accurate production, could be based upon the learn unit formulation of Greer and McDonough (1999). Rehearsal and strengthening of behavior during the course of

\(^2\) The term “frame” in this context differs from a frame of information in the tutor.
learning, however, are rarely, if ever, formal considerations in programmed instruction, although structured practice and overlearning are often studied in other fields (e.g., Salas & Cannon-Bowers, 2001; Swezey & Llaneras, 1997). Tudor and Bostow (1991) did mention repetition in their program, but details were not provided. Hopefully, continued research in rule-governed behavior (e.g., Joyce & Chase, 1990; Mace, 1994; Schmitt, 1998) will ultimately impact instructional design by improving the effectiveness of a teaching tactic such as programmed instruction by suggesting the history required to provide optimal generative use of rules in novel situations (cf. Anderson, Conrad, & Corbett, 1989; Anderson, Corbett, Koedinger, & Pelletier, 1995).

The design of the programmed instruction tutoring system was based upon the learn unit formulation of Greer and McDonough (1999). A terminal performance was first specified, and a series of progressively more demanding learn units was presented to achieve that outcome for all students in an actual classroom context. This tactic differs from conventional research that is based on between-group comparisons that typically seek information on relative effect sizes across groups in a single experiment. In a discussion of single-group training evaluation, for example, Sackett and Mullen (1993) argued that differential performance outcomes observed in between-group studies might not be as important to an organization as is the confidence that a learner has been exposed to an instructional experience that assures the achievement of a target level of performance. Our model for producing cumulative knowledge in this domain follows this research approach, rather than one that requires the use of “control groups” and null hypothesis refutation in a single experiment, rarely if ever replicated, as the tool of persuasion required to adopt and to improve upon what works for all of our students.

This approach is consistent with the goals of a behavior analysis application, and the series of observations with the Java tutor, to include the present data, constitute systematic replications (Sidman, 1960) that increase our confidence that the tutor can dependably do what is asserted over many different student groups. The replications are systematic in that the tutor interfaces have evolved since the initial introduction of this instructional technology into our classrooms, and the tutor has proved beneficial over many different classes of undergraduate and graduate students. This focus on replicating classroom learning, using a web-based tutoring system, is also consistent with Mayer’s (2002, p. 14) context approach to research on instructional methods and with Tennyson’s (1999) consideration of the goals of automated instructional systems.

We accepted the following constraints in this work. First, we intended to use undergraduate and graduate students in the classroom rather than research subjects in the laboratory. We selected high external validity for our work. Second, we were limited to the time available, which was a 3-h class. Although the tutor was available outside the classroom, we were not able or willing to require completion of the tutor in a time frame that would insure temporal continuity with what had been completed during class. Accordingly, data collection ended at the conclusion of the class. Third, our primary objective was to insure that all learning objectives in the tutor were achievable, eventually, by all students, without regard to the time required to
complete the tutor. For that reason, we consider the tests of near and far transfer performances that are sub-optimal as revealing deficiencies in the tutor design rather than being criteria for potential rejection or adoption of an instructional tactic based on comparisons of measures of central tendency among groups. To the extent that those performances are important as a learning objective, they should be built into the tutor at some future point, perhaps following the guidelines for teaching for meaningful learning suggested by Mayer (2002). We accepted these constraints because of our experience and commitment that the tutor does provide a more uniform background preparation among students for studying Java than would have occurred if we had assigned pages of a textbook or lecture notes for the students to study.

The learn unit size in the present tutor changed over successive stages in the tutor, beginning with a simple *echoic* response, such that the constructed performance became larger and larger within each three-term contingency unit, until the student could produce the final Java applet program. The fact that students showed transfer of learning from the elemental frames indicates that the final recitation of the program was achieved under more complex controlling conditions than the simple production of an *intraverbal* series of items. Tudor and Bostow (1991) also emphasized the importance of a close correspondence between the terminal performance and the requirements of learning during programmed instruction. Finally, the design of the present tutoring system followed the definition of the elements of programmed instruction presented previously.

The definition of programmed instruction should continue to evolve as a function of its effectiveness and of our progressive understanding of the principles and parameters of learning and how to apply them to achieve meaningful learning in automated instructional systems. Our obvious preference is the learn unit formulation by Greer and McDonough (1999) as a unifying framework. Too often, perhaps, limiting the design of frames and tests of learning to canonical archetypes, perhaps important in a historical perspective of behavior analysis, has inhibited the growth, dissemination, and use of this instructional technology. From Pressey (1927) to Skinner (1958), the intent was to provide landmarks for future automated instructional systems that would reflect ever more complex repertoires, to include abstraction (Holland, 1960) and general problem solving skills (Peel, 1967).

In that latter regard, a potential area of development is the use of a human “expert,” in a vocal or written interaction with a learner, to evaluate the learner’s understanding of material when the subject matter is too complex for automated assessment of mastery. The use of the Keller method (Austin, 2000; Keller, 1968; Martin, Pear, & Martin, 2002), modified so that proctor evaluators may suggest that a student repeat a set of programmed instruction frames, is a promising avenue to explore. An *interteaching* tactic (Boyce & Hineline, 2002), requiring a mutually informed verbal interaction between students who are paired to converse, may also prove helpful even when applied to learning technical material. The integration of automated instructional systems with other teaching tactics warrants ongoing consideration, implementation, and evaluation.
A search, however, on the Journal of Applied Behavior Analysis web site\(^3\) on the terms “programmed instruction” yielded only nine articles from 1969 through 2002, and some of those involved disadvantaged learners. Using PsycINFO, an identical search over the same time interval returned 161 items, and 59 of them were circulated in dissertation abstracts. These are puzzling outcomes in light of the positive impact of the Java programmed instruction tutoring system on our students and in light of unanswered questions about the application of this approach in the classroom. It has been our anecdotal experience that many students who are highly motivated to learn about information technology but who lack an extensive history in the use of computers are sometimes intimidated and demoralized by the sudden demands to learn a new symbol set and mode of communication, especially in the object-oriented programming paradigm. These students appear to benefit most from a series of learning experiences giving them practice with the formal properties of the symbols (“learning to type”) and gradually leading them to master a simple Java program. This positive initial experience – this structured history – clearly carries over into a willingness by these students to continue their study of information technology with confidence. But the research stream in programmed instruction seems dry.

It is our view that programmed instruction is uniquely applicable to technology education tactics and deserving of continued investigation. We would benefit from research that would help us to determine the optimal number of repetitions of a frame and to determine when a learner should transition from one learn unit to the next. We would also benefit from research that would help us to engineer tutor frames to achieve the most efficient learning and transfer, while acknowledging the methodological challenges in this important area of work (Barnett & Ceci, 2002). Finally, a shift away from between-group comparisons of instructional approaches, using null hypothesis testing, to a competency model that assures an identical achievement outcome across all learners (Sackett & Mullen, 1993), could also stimulate useful research in this area.

Most of the computer-based tutoring systems that have been extensively researched and even applied in the K-12 classroom and beyond rarely, if ever, mention programmed instruction or behavior analysis as a scholarly context (e.g., Anderson et al., 1995; Brock, 1997; Hall & Hughes, 2000; LeLouche, 1998; Shute & Psotka, 1996; Suppes & Morningstar, 1969; Szabo & Montgomerie, 1992). Behavior principles are lawful, however, even when researchers in other fields of education do not name (i.e., tact) that orderliness in nature with them. To allow programmed instruction to participate more actively and prescriptively in the ever-increasing number of practical computer-based tutoring systems and to occasion more research enthusiasm for this important area of behavior analysis and application, it will be essential to nurture and advocate this approach in a developmental way that makes it visible and appealing by virtue of its consequences. As stated by Critchfield (2002), “A branch of science, no matter how effective, serves little purpose if most investigators, practitioners, and policy makers fail to take notice” (p. 424).

\(^3\) http://www.envmed.rochester.edu/wwwrap/behavior/jeab/jeabindx.htm.
Acknowledgements

The author is indebted to John Goodall for his helpful comments on the manuscript. Thanks also to the IFSM 613 students who helped by taking the course and by filling out all those questionnaires.

Appendix A

This is a questionnaire to assess your current understanding of Java. The purpose of the questionnaire is to help us to offer you the most effective tools to learn Java. Your serious attention to these questions and your sincere answers will help us to accomplish that. These questionnaires in no way are related to your grade in a course at UMBC.

Please circle the best answer for the multiple-choice questions.

1. Which of the following lines most likely would be used to reference Frame.class, which is a class file 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;
   e. Not ready to answer.

2. Which of the following lines most likely would be used to construct an instance of a Button class?
   a. myButton = new Button.class (“Hello”);
   b. myButton = new Button (“Hello”);
   c. myButton = button.class (“Hello”);
   d. myButton = Button (“Hello”);
   e. Not ready to answer.

3. Which of the following lines most likely would be used to add a Button object to a container?
   a. Add (an instance name);
   b. Add (a class name);
   c. add (a class name);
   d. add (an instance name);
   e. Not ready to answer.

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}
   e. Not ready to answer.
Appendix B

The below is the Java program that you will learn or have learned, and it is organized into 10 rows of code. Answer the 10 questions below as best you can at this point in your learning. Please circle your choice of answer for each of the 10 multiple-choice questions.

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: }

1. What is the overall objective of the code in Row 1?
   a. Reference the import utilities.
   b. Create a shorthand notation to reference the built-in Applet class.
   c. Import all available java class files.
   d. Include a file named java.applet.
   e. Not ready to answer.

2. What is the overall objective of the code in Row 2?
   a. Create a shorthand notation to reference the built-in Label class.
   b. Create a shorthand notation to reference the built-in label class.
   c. Copy the Abstract Windowing Toolkit.Label directory.
   d. The objective is to import the awt.label file.
   e. Not ready to answer.

3. What is the overall objective of the code in Row 3?
   a. Name a class, MyProgram, that will be a superclass of the Applet class.
   b. Name a class, myProgram, that will be a subclass of the Applet class.
   c. Override the extends Applet modifiers.
   d. Name a class, MyProgram, that will be a subclass of the Applet class.
   e. Not ready to answer.

4. What is the overall objective of the code in Row 4?
   a. Construct an instance of myLabel.
   b. Construct an instance of Label.
   c. Declare myLabel as a potential instance of the Label class.
   d. Declare Label as a potential instance of the myLabel class.
   e. Not ready to answer.

5. What is the overall objective of the code in Row 5?
   a. Insert the init() class in this program.
   b. Write a method that returns a value.
c. Write the init() method in this program. The method will not return a value.
d. The objective is to hide the init() method.
e. Not ready to answer

6. What is the overall objective of the code in Row 6?
a. Construct a new instance, myLabel, of the Label class.
b. Construct a new instance, myLabel, of the label class.
c. Display immediately the text string in the browser window.
d. Construct a new instance, myLabel, of the myLabel class.
e. Not ready to answer.

7. What is the overall objective of the code in Row 7?
a. Add to the default brightness of the Label object.
b. Install myLabel in the Applet container.
c. Concatenate the myLabel and Label objects.
d. Concatenate the String values of myLabel.
e. Not ready to answer.

8. What is the overall objective of the code in Row 8?
a. Apply the myLabel method to setVisible().
b. Make myLabel invisible with a boolean argument.
c. Modify the properties of the setVisible() method.
d. Make myLabel visible to the user if it was invisible.
e. Not ready to answer.

9. What is the overall objective of the code in Row 9?
a. Close the group of statements in the init() method.
b. Close the group of statements in the class definition.
c. Insert a comment marker.
d. It is an end-of-line marker.
e. Not ready to answer.

10. What is the overall objective of the code in Row 10?
a. Start the flow of control in a new Thread.
b. Close the group of statements in the init() method.
c. Close the group of statements in the MyProgram class definition.
d. Block out the import lines.
e. Not ready to answer.

References


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.


