The Impact of Computer Augmented Online Learning and Assessment Tool

By: Misook Heo and Anthony Chow

Heo, M. & Chow, A. (2005). *The Impact of Computer Augmented Online Learning and Assessment*. Educational Technology & Society, 8(1), 113-125.

Made available courtesy of International Forum of Educational Technology & Society: http://www.ifets.info/others/

*****Note: Figures may be missing from this format of the document**

Abstract:

The purpose of the study was to investigate the impact of an experimental online learning tool on student performance. By applying cognitive load theory to online learning, the experimental tool used was designed to minimize cognitive load during the instructional and learning process. This tool enabled students to work with programming code that was supplemented with instructor descriptions and feedback, embedded directly within the code while maintaining the original integrity of the coding environment. A sample of 24 online graduate students at a southeastern university were randomly assigned to four groups: Group 1 (Control group), Group 2 (Assessment group: the tool was used to provide feedback on student work), Group 3 (Lecture group: the tool was used to describe examples of code provided in lectures), and Group 4 (Total tool group: the tool was used to provide feedback on student work as well as describe examples of code in lectures). Student learning was measured via analysis of six online quizzes. While provision of tool-facilitated feedback alone did not appear to enhance student learning, the results indicate that students performed best when they had the opportunity to view examples of code facilitated by the tool during the learning process of new material. This implies a carefully designed online learning environment, especially while controlling for and minimizing cognitive load when presenting new information, can enhance that student learning.

Keywords: Online learning, Information technology education, Assessment, Personalized learning, Cognitive load theory

Article:

Introduction

According to the human cognitive architecture, only the information that is attenuated to and processed through adequate rehearsal in the working memory is transferred to the long-term memory, becoming a part of a person's permanent memory (Anderson, 2000). Long-term memory can be used to store schemas of varying degrees of automaticity. The capacity of long-term memory is virtually unlimited, but humans are not directly conscious of long-term memory. Humans are conscious of only the contents of their working memory. Unfortunately, the capacity of working memory is limited to about seven elements at a time (Miller, 1956).

Cognitive Load Theory suggests that instructional design will be improved if better consideration is given to the role and limitations of the working memory (Cooper, 1990). According to Sweller, one of the primary objectives of instruction is to reduce the mental workload of the learner (cognitive load) in working memory. When information is properly processed in working

memory, it is encoded into long-term memory. Knowledge is stored in the form of organized schemata in the long-term memory; this process can free the working memory capacity, as these schemas allow for meaningful encoding and efficient knowledge retrieval for learners allowing processes to occur that otherwise would overburden working memory (Sweller, van Merrienboer, & Paas, 1998). Cognitive load theory postulates that two types of cognitive load affect learners simultaneously: intrinsic and extraneous cognitive load. Intrinsic cognitive load is based on the level of difficulty the learner associates with the information that is presented, and this load cannot be reduced externally through either the instructional design of the material or the instructor. Extraneous cognitive load, on the other hand, is information that is not essential to instruction, which serves to distract learners from the primary information to be learned. When the intrinsic cognitive load is low, the working memory has enough space to handle a large extraneous cognitive load. In this case, the instructional design does not have much of an impact on student learning. When the intrinsic cognitive load is high, on the other hand, not much room remains available in the working memory for extraneous cognitive load. As such, poorly organized information will cause a substantial increase in extraneous cognitive load, using most, if not all, available working memory. In this case, learning does not occur efficiently, if it occurs at all. Extraneous cognitive load therefore is of primary concern for instructors and instructional designers, with the goal being to minimize these distractions as much as possible.

Utilizing worked examples is a primary way to reduce extraneous cognitive load and facilitate student learning, and this is referred to as the worked example effect. Worked examples let learners attend to problem states and associated operators, enabling learners to induce generalized solutions or schemas. In the absence of a schema with worked examples, means-ends analysis is an efficient way of attaining a problem goal. When learners utilize the means-ends analysis, they focus their search on actions that reduce the difference between the current state and the state that is their goal. In this case, the learner attends to the information, negotiating the differences between the current state and the desired state in their working memory until the goal is reached. Learners' extraneous cognitive loads, thus, become high. In contrast, when appropriate worked examples are utilized, learners have nothing else to attend to and their extraneous cognitive loads become low. There can be, however, no guarantee that all worked examples reduce cognitive load in comparison to a means-ends search.

For example, worked examples presented to the student in a non-integrated fashion scatters student attention. This split attention can cause increased cognitive load, impairing the ability of students to learn, such as using code to explain a concept but placing the description and explanation of that code at the end of the sequence instead of in an integrated fashion directly paralleling the code and the discussion of that code. This forces the learner to go back and forth from explanation to original code, placing an extraneous demand on working memory (Cooper, 1990). Therefore, providing worked examples in an integrated format is critical in order to best facilitate learning.

The use of worked examples is a critical component to the learning process in programming courses, as these types of courses often are designed with tightly paired conceptual and pragmatic knowledge. Students gain exposure to fundamental programming techniques and underlying concepts through practice with code examples that they are able to later transform into practical solutions through assignments and small projects (Clear, Haataja, Meyer, Suhonen,

& Varden, 2000; Emory & Tamassia, 2002; Malmi, Korhonen, & Saikkonen, 2002). Instructors, thus, often utilize textbooks and lectures that provide ample code examples that are in the performance context in an effort to facilitate student learning.

In this environment, students often play the role of a self-directed learner while instructors serve as facilitators of personalized learning rather than as broadcasters of knowledge (Clear et al., 2000; Malmi et al., 2002). To better support personalized learning, instructors are also asked to provide personal attention to students and to provide an environment where students can learn in ways that work most effectively for them (VanDeGrift & Anderson, 2002).

One of the primary means for achieving this goal is for instructors to provide accurate and meaningful assessment (Preston & Shackelford, 1999). It is often reported, however, that the task of grading student programs is a laborious process (Jackson & Usher, 1997). When direct contact with students is limited, assessing student work becomes even more difficult (Gayo, Gil, & Álvarez, 2003).

While educators agree that instructors in an online learning environment must spend more time and effort reviewing student work than they would spend in a face-to-face classroom course (Clear et al., 2000; O'Quinn & Corry, 2002; Preston & Shackelford, 1999), the instructor does not always have the luxury of devoting this amount of time. In fact, many instructors contend that the inability to complement their virtual classroom environment with traditional methods dampens their sense of effectiveness (Gayo et al., 2003; O'Quinn & Corry, 2002).

For online learning environments, thus, lectures and code examples tend to be functionally and visually static and remain organized around the delivery media rather than the knowledge representation and learning tasks of the student (Altman, Chen, & Low, 2002; Reed & John, 2003; Zachary & Jensen, 2003). For example, a code example is often followed by additional explanations and descriptions, causing split-attention effect for learners and creating a situation where every student receives the same amount of description for a specific code. Likewise, coding assignments are usually graded with limited feedback in problem solving and programming techniques (Trivedi, Kar, & Patterson-McNeill, 2003). General interaction among the instructor and students is often less frequent than it would be in a face-to-face classroom environment, especially when the course is in programming where the primary mode of communication is text-based (Malmi et al., 2002; Price & Petre, 1997).

The need to reduce cognitive load in online learning environments that are predominately static and text-based represents a significant problem, especially in programming courses. A tool that simultaneously reduces student split-attention but does not cost instructors any additional time and effort in teaching online programming courses is needed. By addressing the problem of split-attention, overall student extraneous cognitive load should be reduced, leading to more available working memory for learning newly introduced information, and potentially increasing overall learning effectiveness. In addition, such a tool could also potentially reduce the overall work load of an online programming instructor through augmenting the process of providing meaningful descriptions and personalized comments to code examples and student work.

Purpose of the Study

The research presented in this paper seeks to investigate the impact of an experimental online learning tool on student performance. An experimental tool, the Online Learning and Assessment Tool (OLAT), was implemented to apply the cognitive load theory to online learning, attempting to minimize cognitive load during the instructional and learning process.

This study is intended to address the primary research question, "Does the use of the OLAT improve student learning?" We have developed three hypotheses addressing the research question tested in this paper:

1. Students receiving tool-facilitated feedback on their work will gain enhanced understanding from their mistakes, thus their test performance over time will improve beyond that of control group students.
2. Students receiving tool-facilitated descriptions in code examples will develop a better understanding of the examples, thus their test performance over time will improve beyond that of control group students.
3. Students receiving both tool-facilitated descriptions and feedback will show the greatest improvement in performance over time.

The OLAT allows the student to view instructor-provided descriptions and/or feedback needed to increase knowledge about a particular section of code or about mistakes that have been made.

It is expected that exposure to the tool-facilitated descriptions will improve student learning by reducing extraneous cognitive loads for students when they are first exposed to new materials. While it is expected that exposure to the tool-facilitated feedback alone will not improve student learning much since students' intrinsic cognitive loads will not be high when dealing with already learned material, it is anticipated that students exposed to both the tool-facilitated descriptions and feedback will achieve the greatest improvement in performance over time.

Methodology

Participants

Each of the 24 graduate students participating in the six-week study attended a southeastern university; a possible selection bias is present in the study as each of the participants was self-registered to the online session of the Advanced Web Applications course. Participants were not monetarily awarded but were rewarded with academic credit for participation in this experimental study. Each participant's age, gender, and academic program were recorded, but kept confidential in accordance with the Institutional Review Board (IRB) guidelines.

Instruments and Materials

Experimental Intervention: The Online Learning And Assessment Tool (OLAT) – as Learning Tool

The OLAT serves as a learning aid by allowing the instructor to tailor descriptions to a specific portion of the code example. The process occurs easily, utilizing simple "point and describe" actions. The instructor simply loads the saved code example to a web browser and clicks on the line requiring detailed description. This mouse click action can be likened to the process of a face-to-face classroom instructor pointing to a portion of code to provide an explanation. A description-ready window appears in which the instructor is able to compose the desired

description. Figure 1 provides a screenshot of this stage. The instructor's code description is now ready to be viewed by students.

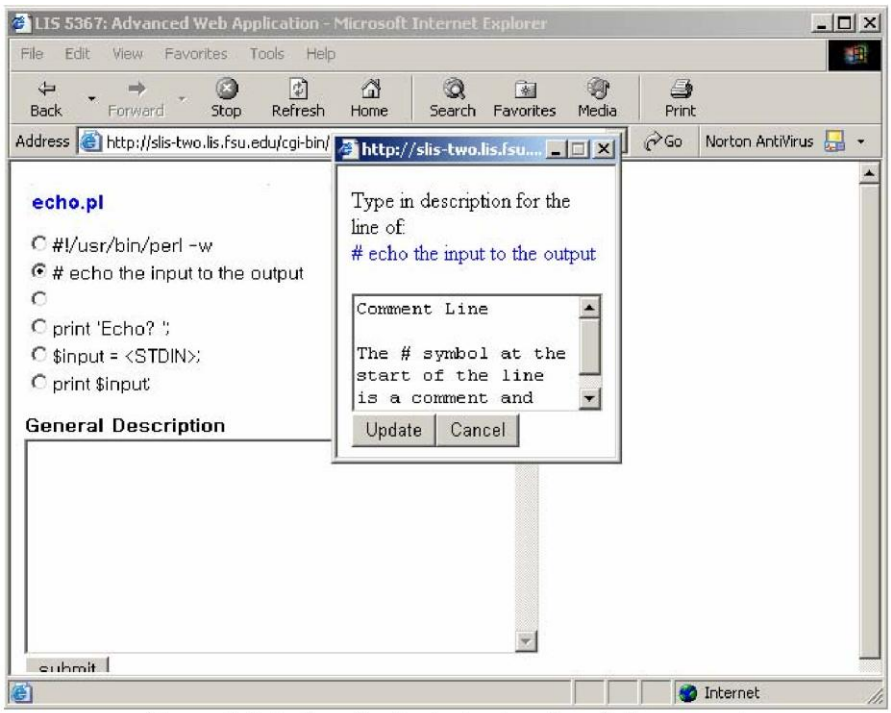

Figure 1. Interface for the instructor: description prompt

The process of reducing cognitive load for the instructor by providing such an efficient means for adding description to meaningful worked examples can be referred to as augmentation, which reduces the load of working memory by removing trivial human tasks, thereby freeing the working memory and enhancing the available capacity to be used for instructional purposes as efficiently as possible.

Once an instructor description has been embedded, students are free to review the code example with or without the embedded descriptions. Students may choose to review the embedded code descriptions by moving the mouse over the color-coded lines (e.g. red-colored lines implicitly indicate that there are embedded descriptions). This process can be likened to the student raising a hand in the face-to-face classroom for further explanation of a specific section of code. Figure 2 provides a screenshot of this stage.

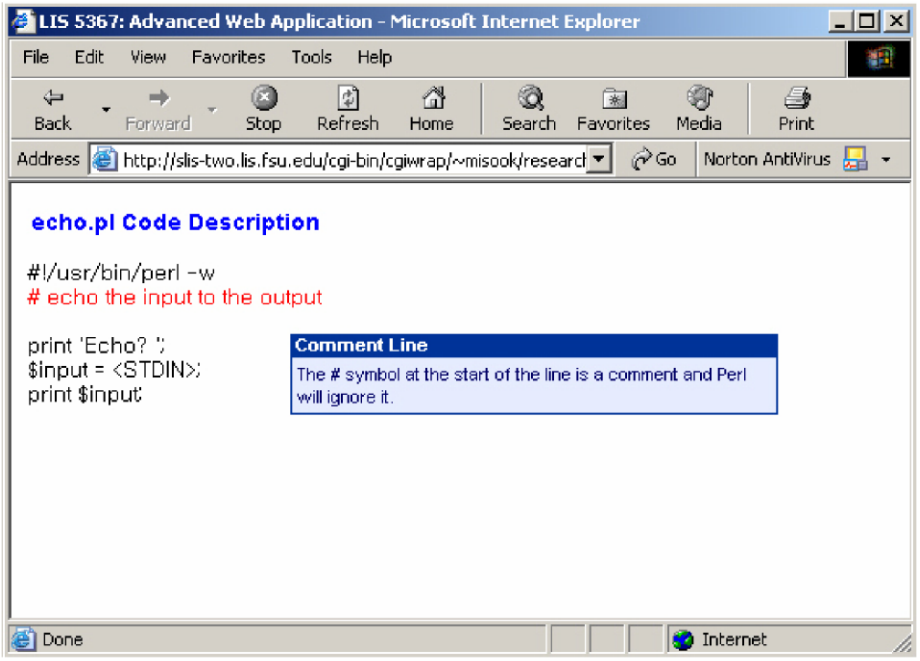

Figure 2. Interface for the student: revealed description

Theoretically, by providing instructor descriptions embedded directly within the worked examples, the OLAT can help protect against the split-attention effect that causes extraneous cognitive load in working memory, and thus support efficient student learning. Further, the OLAT's ability to view code descriptions on demand cultivates a positive environment of self-directed, personalized learning among a diverse student population (Brusilovsky, 2001).

Providing mouse-over activated descriptions within text to preserve the original integrity of the performance environment is not completely new technology. Popular applications such as MS Word provide easy access to such functionality. One of the primary benefits of Web-based education, however, is classroom and platform independence (Atolagbe, Hlupic, & Taylor, 2001; Brusilovsky, 1998). With the advent of platform-independent applications, there are far greater possibilities for creating more useful educational tools (Bridgeman, Goodrich, Kobourov, & Tamassia, 2000). Eliminating the need to rely on students owning specific proprietary software can be seen as taking full advantage of the benefits offered through online learning.

Color-coding for the code lines with embedded description and feedback utilizes the inherent advantages of pre- attentiveness theory. This theory holds that processing occurs automatically for people as they pay visual attention to and process graphical features such as color and size in a pre-attentive fashion. In other words, people see and process size and color differences prior to cognitive processing. Pre-attentive information representation is mentally economical, since the information is rapidly and efficiently processed by the preattentive visual system rather than through cognitive effort (Bartram, 1997).

Experimental Intervention: The Online Learning And Assessment Tool (OLAT) – as Assessment Tool

In face-to-face classroom courses, students may submit their assignment printouts to the instructor. The instructor is then able to read through the submitted code, marking errors or

inserting corrections, comments, or advice in the appropriate portion of the code. Often, the instructor will choose to emphasize feedback with colored ink (Herrmann et al., 2003). Upon receiving the graded assignment, the student is able to directly view the location where feedback is written; it would not be necessary to count line numbers or read the code line by line to interpret the instructor's feedback.

Currently, in most online education environments, student assignments are uploaded to the course server or delivered to the instructor by email. The instructor then reviews the submitted assignment and adds feedback at the end of the assignment file or in a separate email message, which is returned to the student. This practice creates added difficulty since, in addition to evaluating and providing feedback on the code, the instructor must now consider the line number and location of the applicable comment, or must use proprietary software to insert comments within the document itself. This process also presents added difficulty for students, as they must orient themselves to the specific location of comment by counting line numbers and apply consolidated feedback to appropriate sections of the code, or they must possess the necessary proprietary software.

The assessment portion of the OLAT facilitates the process of providing feedback on student code by allowing the instructor to tailor comments to individual student code. The instructor simply accesses the submitted assignment code through a web browser and clicks on the lines that require feedback. This mouse click action can be likened to providing feedback at a specific location within a printed version of the code. On this click action, a comment-ready window appears and the instructor simply types in the appropriate feedback. The process for making and retrieving comments are the same as for providing and retrieving instructor descriptions to worked examples as part of a lecture depicted earlier in Figures 1 and 2. Although the OLAT's learning and assessment tool features serve different purposes, they function in the same manner.

The OLAT, as a platform independent online application for providing instructor feedback, facilitates the assessment process and also relieves the instructor and/or students from the burden of needing to have proprietary software—inserted descriptions occur and are saved directly to the server (the application is Perl/CGI/JavaScript based) through any browser they may use. This augments the process by allowing instructors to skip the time intensive process of downloading a student document, making comments and saving that document to their local desktop, and then having to upload it back to the server and/or emailing it back to the student. The entire transaction occurs online.

The OLAT was embedded into the existing online course infrastructure. Since the tool produces pages with embedded description and feedback that are visible in any Web browser, neither the instructor nor students were asked to install any special software to make use of the tool.

Course Management System

An in-house course management system was used to conduct the study. This system has similar functionality to commercial products such as BlackBoard or WebCT possessing 1) asynchronous components such as posted lectures, threaded-discussion boards, email, announcements, assignment/drop box, and other course materials (syllabus, course calendar, etc.), and 2) synchronous components that consist of text-based chat interactions.

Assessment Measures
There were two types of assessment measures used in the study: quizzes and coding assignments.

Quizzes
Six in-class quizzes were conducted. Quizzes consisted of online multiple-choice and short answer questions intended to address the two main categories of assessment, objective questions and performance based questions (McCracken et al., 2001). Students were asked to respond to quiz questions on a weekly basis after reviewing code examples and instructor feedback. Quizzes were based on relevant course material and were offered not only to evaluate the impact of tool-facilitated material, but also to reinforce student learning and application of course content.

Six questions were asked in each quiz. On each quiz, three questions addressed the previous week's assignment and three questions addressed the current week's examples of code. For each quiz, the majority of questions were standard close-ended multiple choice (objective based questions); each quiz, however, also included a few open-ended, short answer questions (performance based questions). The quizzes were standard HTML, form- based, and conducted online using the course management system's assessment features. Figure 3 shows an example of the quiz questions.

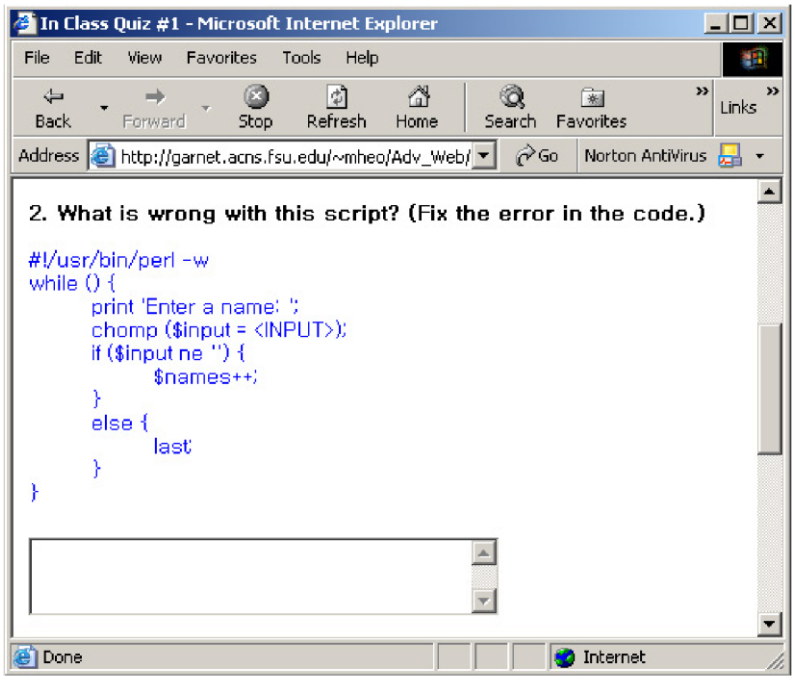

Figure 3. Example Quiz Question

Assignments
Each week after learning new syntax and being exposed to worked examples, students were asked to complete a coding assignment. Student code was submitted through the course Website by uploading a zipped ASCII text file.

Questionnaires

Two questionnaires were administered during the study. The pre-test was a 15-item questionnaire, which collected participant demographic data including age and gender, and previous experience with computers, the Web, programming languages, and online learning. Example pretest questions include: "How many computer language courses did you take so far?" and "How many courses per semester (on average) do you take Web- based distance courses?"

The post-test questionnaire was also a 15-item instrument and collected participants' perspectives on their experience with tool-facilitated descriptions and feedback. The questions were a combination of ordinal scale and seven-point Likert scale. Examples of post-test items include: "Indicate the amount of time you spent studying the lecture slide/audio per week" and "How helpful the descriptions in the code examples for your understanding of concepts of each week's learning material? Choose one between 1 (Not at all) and 7 (Very much)."

Procedure

In order to test our hypotheses, a six-week experimental study was conducted in a Web programming course taught online during a summer session. The study involved 24 graduate students from a southeastern university, all of whom were enrolled in the online course "Advanced Web Applications." Participants were randomly assigned to four experimental groups. Students in each group reviewed lecture material and completed a series of quizzes and programming assignments. The experiment ran from May 2003 through June 2003.

The experimental study consisted of four groups: 1) Control group: the OLAT tool was not used and participants viewed line-number based descriptions and feedback in the traditional manner, as shown in Figure 4; 2) Assessment group: the OLAT tool was only used to provide feedback on student work; 3) Lecture group: the OLAT tool was only used to provide descriptions of examples of code in lectures; and 4) Total tool group: the OLAT tool was used to provide feedback on student work as well as descriptions of examples of code in lectures. Each participant was assigned randomly to one of the four groups.

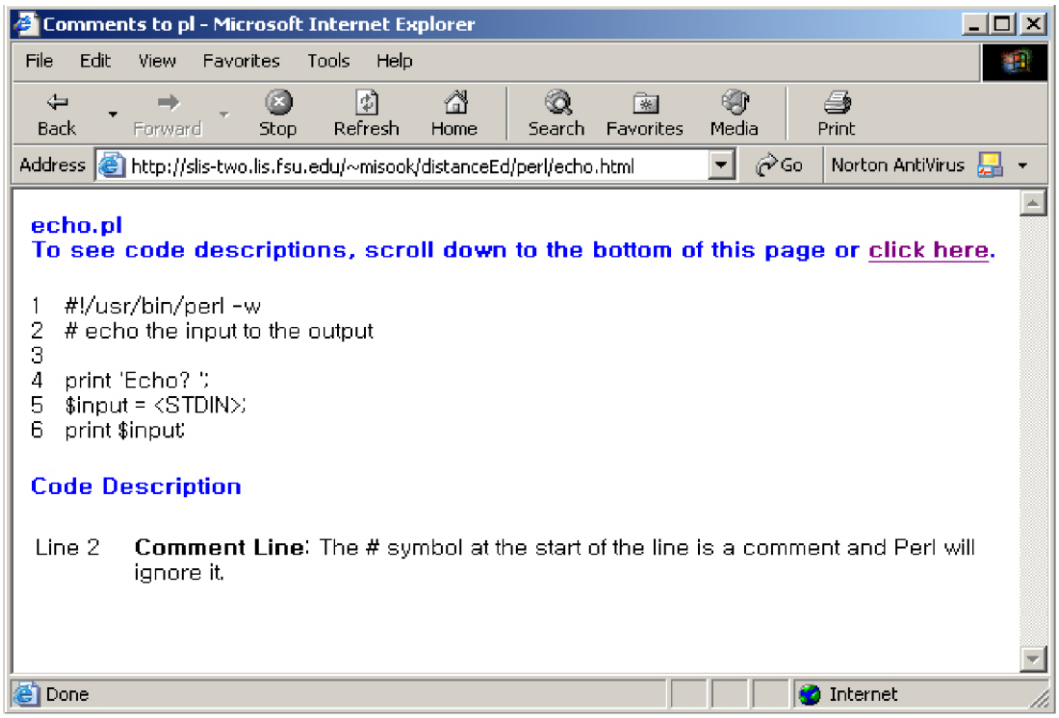

Figure 4. Interface for the student: line numbered description

An initial online pre-test questionnaire was administered to participants to collect demographic and experience data as described above. Each week of the study period, lectures were offered and coding assignments were presented. Each lecture provided one or more examples of code and the instructor reviewed each student assignment within 12 hours of the assignment deadline. Student learning was also supported by various online education methods, including weekly audio lectures with slides, weekly synchronous chat sessions, an asynchronous faculty office discussion forum, and an asynchronous student discussion forum.

During each week of the study participants took an online quiz during the regularly scheduled class time (two hours each week), which included questions from the current week's examples of code as well as the previous week's assignment. No time limit was enforced for any of the quizzes but participants were advised to finish the quiz in 15 minutes. User logs such as access time to the quiz page, IP address, and student ID, were reviewed for student identification. After concluding the pre-test and all six quizzes, participants were asked to provide their perspectives on the examples of code and feedback on their work in a post-test questionnaire. Immediate access to quiz performance was not available due to the fact that there were a few open-ended, short answer questions, which needed to be graded by the instructor. Multiple-choice questions had one correct answer per question. The short answer questions were performance based requiring participants to identify problems and/or provide necessary solutions to coding examples.

Measures

Dependent variables

The number of correct answers and the amount of time taken to complete weekly quizzes served as the study's dependent variables of performance; if tool-facilitated descriptions and/or

feedback aided student learning, student performance on quizzes would improve over time. Time taken to complete quizzes was analyzed to measure the possible trade-off between the number of correct answers and time on task. Follow-up perceptive evaluation results were also collected and analyzed to enrich the data.

Analysis

Analysis of Variance (ANOVA) one-way factorial design was used to measure the participants' performance progress, and the main effects of the OLAT on student learning were analyzed. When the resulting F values were significant, a post-hoc analysis was conducted to investigate the differences among the groups. As a post-hoc test, Tukey's Honestly Significantly Different test was used. Pearson Product Moment Correlation Coefficient was used to analyze linear relationships between the number of correct answers and the time taken to complete quizzes. ANOVA one-way factorial design was also used to analyze the results of the perceptive evaluation questionnaire.

Results

Homogeneity Among Groups

Since no screening process was used to recruit participants, variance among the four experimental groups was analyzed using the Kruskal and Wallis Test to test for potential differences in homogeneity. No significant difference in the amount of experience among the groups on computers, the Web, programming languages, and online learning was found (all asymptotic significance values were greater than the significant level of 0.05).

Analysis of student performance on quizzes completed during the first three-weeks also showed inter-group homogeneity. Student performance on these quizzes could reflect pre-existing knowledge of course subjects, since participants were newly introduced to the course and had not yet become accustomed to the way to view tool-facilitated descriptions and feedback.

Main Test Results

The progress in performance (the change in the number of correct answers from the first three weeks to the last three weeks) for each of the four experimental groups is shown in Table 1. A one-way ANOVA indicated that student performance in the tool-facilitated lecture group significantly increased across questions asking about lecture material ($F(3, 20) = 4.34$, $p = .016$). Analysis also showed that this group had a corresponding positive trend on student learning in overall quiz questions ($F(3, 20) = 2.77$, $p = .069$). The total group also showed an increase in student performance across overall quiz questions.

To determine where the difference occurred, a Tukey post-hoc test was conducted. According to the test, the significant difference in increased performance occurred between the lecture group and the assessment group ($p = .014$). Each group spent about the same amount of time completing quizzes ($F(3, 20) = 1.65$, n.s.), however the lecture group showed more progress than the participants in other groups. Pearson's correlation showed no significant trade-off between the number of correct answers and time taken to complete quizzes.

Table 1. Statistical results of the main-test

Group	Improvement in number of correct answers				Quiz time		N
	Question	Mean[a]		Std. deviation	Mean[b] (seconds)	Std. deviation	
Control	Lecture	-2.00	-.50	1.64	1007	252.40	6
	Assignment		-1.50	2.17			
Assessment	Lecture	-3.17	-1.67	1.51	1083	292.51	6
	Assignment		-1.50	3.02			
Lecture	Lecture	1.33	1.67	1.51	1274	327.04	6
	Assignment		-.33	1.97			
Total tool	Lecture	2.00	.67	2.07	921	270.97	6
	Assignment		1.33	3.50			
Total	Lecture	-.46	.04	2.03	1071	298.95	24
	Assignment		-.50	2.81			

Note. [a]Number of questions asked at quiz #4 through #6 = 18.
(Lecture related questions = 9 and assignment related questions = 9)
[b]Mean quiz time reflects a summed amount.

Role of Experience in Online Education

While programming knowledge in general and programming course experience in a face-to-face classroom environment showed positive correlations toward student performance, .396 (n = 24, p = .056) and .404 (n = 24, p = .051), previous online education experience showed a significant negative correlation. For the question, "How many courses per semester (on average) do you take online?" there was a significant negative correlation of -.726 (n = 24, p = .000) with student performance. In addition, for the question, "How long have you been taking online courses?" there also was a significantly negative correlation of -.593 (n = 24, p = .002) with student performance.

Perceptive Evaluations

In the last week of the study, participants were asked to rate their experience with the examples of code and feedback on their work via a set of survey questions. This post-survey was used to obtain participants' perspectives on Web pages with embedded description and feedback facilitated by the tool. The survey consisted of 15, seven-point Likert scale questions. A one-way ANOVA indicated that participants in the total tool group spent significantly less preparation time than the participants of the control group in studying examples of code ($F(3, 20) = 6.305$, $p = .003$).

Discussion

The purpose of the study was to determine whether the OLAT intervention would improve student learning in an online educational environment. Participants in this study were asked to answer quiz questions after reviewing examples of code and feedback on their work provided with or without the use of the Online Learning and Assessment Tool. Clear performance differences emerged among the four groups. Analyses of experiment data show that descriptions of code examples facilitated by the tool were the most helpful for participants' learning. In contrast, tool-facilitated feedback appeared to be the least helpful for participants' learning.

We believe that these findings suggest that the OLAT intervention facilitated a decrease in the overall extraneous cognitive load associated with students learning new material. When intrinsic

cognitive load is high, such as when faced with processing new material, the available working memory is already severely limited, leaving little room for additional requirements made by extraneous cognitive load (e.g. an online instructional and learning environment). In our study, the intrinsic cognitive load for students was assumed to be relatively high as they were faced with learning new material presented. Given this premise, additional extraneous cognitive load associated with the instructional delivery and environment could potentially increase the overall cognitive load leading to cognitive overload (intrinsic plus extrinsic cognitive load exceeds available working memory), thereby impeding student learning.

Based on the fact that the OLAT lecture group and the OLAT total group (OLAT was used in both lecture and assessment) showed improved performance at higher levels than other groups, we infer that the OLAT intervention led to increased student learning by reducing overall extraneous cognitive load. For the OLAT assessment group, however, which performed significantly lower than the OLAT lecture and total groups, we believe that although extraneous cognitive load may have been reduced by using OLAT during the assessment process, the impact on overall student learning was minimal due to the fact that the tool was not available during the initial learning process.

In addition to the primary findings, we have also determined that while participants with programming language experience showed improved performance on objective testing over time, participants with online education experience showed an overall decrease in test performance over time. While this trend was somewhat surprising, we surmise that students who have more experience with online courses may have established expectations of minimal interaction and personal engagement when reading lectures and assessing assignments. Experienced online students are more likely to typically face a lack of personalized learning and assessment feedback from the instructor. Such learning strategies are problematic in an online programming course because of the iterative, trial-and-error nature of knowledge and skill acquisition involved in becoming proficient in a programming language. Simply put, we believe that more experienced online students, with already preconceived learning and feedback paradigms, took advantage of the experimental tool less frequently and effectively than their less experienced peers.

Based on the findings in our perceptive evaluation, the participants in the total tool group devoted significantly less time on preparation than did the members of the control group in studying examples of code. This difference suggests that participants could obtain the same amount of knowledge, if not more, in less time when the tool- facilitated lectures were provided in both lectures and in assignment feedback.

Implications

The findings of this study support cognitive load theory as applied to instructional design (Sweller et al., 1998). If instruction is delivered in such a way as to effectively reduce extraneous cognitive load, then the necessary working memory will be available for processing and retention of new information. Students who were exposed to the OLAT during the initial presentation of new information in the lecture significantly improved their learning performance over time.

Our study also supports the idea that the worked-example effect is only beneficial if the example actually decreases extraneous cognitive load (Cooper, 1990; Feinberg & Murphy, 2000). When the split-attention effect occurs, the worked-example effect may actually increase cognitive load and impede student learning. Using the OLAT's embedded description/comments arguably helps preserve the integrity of the instructional context—all the information being presented is in the same location, thus protecting students from the split-attention effect. Overall, the findings of this study indicate that student learning can be improved in an online programming environment when unique challenges, such as non-linear characteristics of code paired with associated comments, are carefully considered and mitigated.

Limitations

Although this study provides data supporting cognitive load theory, there are still a number of limitations that should be considered. Although it was strongly recommended that students read all lectures and instructor feedback on their work, it cannot be determined if all students actually read the lectures and feedback comments. Also, we were unable to control for variance in the breadth and time students devoted to this area.

Students were required to participate in a weekly synchronous text-based chat sessions that were supplemented by voluntary asynchronous threaded discussion forums. We acknowledge that discussion and interaction play an important role in online learning and may have a significant impact on student learning; unfortunately, due to constraints of resources and time, we were unable to control for this variable. One point of qualification, however, is that the asynchronous threaded discussions were voluntary and ultimately not utilized to a great extent by students, therefore most likely not accounting for much of the performance variance.

Our findings are only preliminary with a small sample size; extended study with additional participants may need to be conducted to increase the overall strength of our findings. In addition, as student performance was measured only through the use of quiz scores, other measures may need to be used in future studies.

Lastly, the overall amount of time students spent completing quizzes and assignments was not controlled. Students began the quizzes at the same time, although when they finished varied across students. On average, each quiz took approximately 18 minutes to complete. One week was given to complete each assignment.

Conclusion

Although the study described in this paper is still in its early stages of research and development, the results are encouraging and suggest that the OLAT may be successful in reducing cognitive load during the initial instruction and learning process. We remain optimistic that this tool, with additional research, will prove to be beneficial for online programming instruction and student learning.

Continued improvement of the current tool is planned and includes incorporation of a component that allows descriptions or feedback to be provided across multiple lines of code (one comment for multiple lines of code in different areas). In addition, providing the ability to run code

directly within the browser is also planned in order to substantially reduce time required by the instructor to test student code and to reduce students' time to test code examples.

Currently, teaching an online programming course can be a daunting task. The need for easing the burden on behalf of instructors and students is essential to increase the overall effectiveness and efficiency of instruction and learning in virtual space. The OLAT represents an initial attempt to bring together contemporary learning theory and information technology to realize the core vision for online learning—open access, platform independence, self-directed, personalized learning among a diverse student population freed from the limitations of space and time (Brusilovsky, 2001).

References

Altman, E., Chen, Y., & Low, W. C. (2002). Semantic exploration of lecture videos. *Paper presented at the International Multimedia Conference,* December 1-6, 2002, Juan-les-Pins, France.

Anderson, J. R. (2000). *Cognitive Psychology and Its Implications* (5th ed.), New York: Worth Publishers.

Atolagbe, T., Hlupic, V., & Taylor, S. J. E. (2001). Teaching tools and methods: GeNisa: a web-based interactive learning environment for teaching simulation modeling. *Paper presented at the 33rd Conference on Winter Simulation*, December 9-12, 2001, Arlington, VA, USA.

Bartram, L. (1997). Perceptual and interpretative properties of motion for information visualization. *Paper presented at the Workshop on New Paradigms in Information Visualization and Manipulation*, November 10-14, 1997, Las Vegas, USA.

Bridgeman, S., Goodrich, M. T., Kobourov, S. G., & Tamassia, R. (2000). PILOT: An interactive tool for learning and grading. *Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education*, March 7-12, 2000, Austin, TX, USA.

Brusilovsky, P. (1998). Adaptive Educational Systems on the World-Wide-Web: A Review of Available Technologies. *Paper presented at the Workshop "WWW-Based Tutoring" at the 4th International Conference on Intelligent Tutoring Systems*, August 16-19, 1998, San Antonio, TX, USA.

Brusilovsky, P. (2001). WebEx: Learning from examples in a programming course. *Paper presented at the WebNet'01 Conference*, October 23-27, 2001, Orlando, FL, USA.

Clear, T., Haataja, A., Meyer, J., Suhonen, J., & Varden, S. A. (2000). Dimensions of Distance Learning for Computer Education. *Proceedings of the ITiCSE 2000 Working Group Reports*, New York: ACM Press, 101- 110.

Cooper, G. (1990). Cognitive load theory as an aid for instructional design. *Australian Journal of Educational Technology, 6* (2), 108-113.

Emory, D., & Tamassia, R. (2002). Jerpa: A distance-learning environment for introductory Java programming courses. *Paper presented at the 33rd SIGCSE Technical Symposium on Computer Science Education*, February 27-March 3, 2002, Covington, KY, USA.

Feinberg, S., & Murphy, M. (2000). Applying cognitive load theory to the design of Web-based instruction. *Paper presented at the IEEE Professional Communication Society International Professional Communication Conference*, September 24-27, 2000, Cambridge, MA, USA.

Gayo, J. E. L., Gil, J. M. M., & Álvarez, A. M. F. (2003). A generic e-learning multiparadigm programming language system: IDEFIX project. *Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education*, February 19-23, 2003, Reno, NV, USA.

Herrmann, N., Popyack, J. L., Char, B., Zoski, P., Cera, C. D., Lass, R. L., & Nanjappa, A. (2003). Redesigning introductory computer programming using multi-level online modules for a mixed audience. *Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education*, February 19-23, 2003, Reno, NV, USA.

Jackson, D., & Usher, M. (1997). Grading student programs using ASSYST. *Paper presented at the 28th SIGCSE Technical Symposium on Computer Science Education*, February 27-March 1, 1997, San Jose, CA, USA.

Malmi, L., Korhonen, A., & Saikkonen, R. (2002). Experiences in automatic assessment on mass courses and issues for designing virtual courses. *Paper presented at the 7th annual conference on Innovation and technology in computer science education*, June 24-28, 2002, Aarhus, Denmark.

McCracken, M., Wilusz, T., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B., Laxer, C., Thomas, L., & Utting, I. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, New York: ACM Press, 125-180.

Miller, G. (1956). The magic number seven, plus or minus two: some limits of our capacity for processing information. *Psychological Review, 63*, 81-97.

O'Quinn, L., & Corry, M. (2002). Factors that deter faculty from participating in distance education. *Online Journal of Distance Learning Administration, 5* (4), retrieved December 21, 2004 from http://www.westga.edu/~distance/ojdla/winter54/Quinn54.htm.

Preston, J. A., & Shackelford, R. (1999). Improving on-line assessment: An investigation of existing marking methodologies. *Paper presented at the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education*, June 27-30, 1999, Cracow, Poland.

Price, B., & Petre, M. (1997). Teaching programming through paperless assignments: An empirical evaluation of instructor feedback. *Paper presented at the 2nd Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education*, June 1-5, 1997, Uppsala, Sweden.

Reed, D., & John, S. (2003). Web annotator. *Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education*, February 19-23, 2003, Reno, NV, USA.

Sweller, J., Van Merrienboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review, 10* (3), 251-296.

Trivedi, A., Kar, D. C., & Patterson-McNeill, H. (2003). Automatic assignment management and peer evaluation. *The Journal of Computing in Small Colleges, 18* (4), 30-37.

VanDeGrift, T., & Anderson, R. J. (2002). Learning to support the instructor: Classroom assessment tools as discussion frameworks in CS 1. *Paper presented at the 7th annual conference on Innovation and technology in computer science education*, June 24-28, 2002, Aarhus, Denmark.

Zachary, J. L., & Jensen, P. A. (2003). Exploiting value-added content in an online course: Introducing programming concepts via HTML and JavaScript. *Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education*, February 19-23, 2003, Reno, NV, USA.