Java-based E-Learning and E-Assessment with Encryption

Authors:

Zenon Harley, Skythink Solutions, zenon@skythink.com
Eric Harley, Department of Computer Science, Ryerson University, Toronto, Ontario, eharley@ryerson.ca

Abstract *— We have developed an e-learning and e-assessment tool for a first year university course on computer programming. The tool presents information and questions to the student and provides immediate feedback to the student regarding correctness and score. An answer may be a word or a sentence or a complete computer program involving several files. When a program is required, the answer is open-ended in the sense that the contents of the program are only checked in terms of functionality. That is, the program must meet the functional specifications given in the question. The results are packaged for the instructor in a specially-formatted file that contains the questions as they were presented to the student, the student's answers, and an automated assessment. The tool is platform-independent, since it is written in Java with no particular operating system dependencies. It is capable of shuffling questions, presenting a random subset of a group of questions, and awarding partial credit for repeated tries or case mismatches (where case is important). The tool includes security features to improve evaluation integrity.*

Index Terms *— e-learning, e-assessment, computer programming, encryption.*

INTRODUCTION

Computers are widely used to assist in teaching, learning and assessment, just as they are in common use for many other facets of modern life. Perhaps the most common computer based assessment is the type that tallies the responses to multiple choice questions and provides statistics relating to the quality and hardness of each individual question [7]. This type of assessment and computer assistance is very important and relevant given the large size of classes in many universities and the possibility of designing questions that test higher levels of understanding beyond memorization of basic facts [7]. For many years, computers have also been used to guide students through course materials, present tests and do assessments. Many articles on this topic can be found in the Electronic Journal for e-Learning, for example [3]. For a recent review of on-line testing and a discussion of Learning Management Systems, please see [2].

Although the World Wide Web (WWW) is a common vehicle for e-assessment and e-learning, we decided to build a desktop application because it offered greater flexibility and power. In particular, we wanted to use the desktop's full-featured Java compiler, allowing us to interactively compile and evaluate students' code. Moreover, the off-line nature of this approach makes it more suitable for a formal examination setting, in light of the difficulty involved in restricting access to on-line resources.

We have used our tool in two different but complementary scenarios:

- Student-driven e-learning: The student uses the tool to gauge his or her own mastery of concepts taught in class. This form of self-assessment allows the student to learn on any computer and at any pace. Results are submitted to the instructor to measure class progress and participation. This type of assessment is termed *formative* in the literature [1].
- Controlled e-assessment: The tool, which will have become familiar to the student during e-learning, is used in a controlled environment (i.e., supervised and off-line) to administer a test or an exam. The results submitted are used for grading. This type of assessment is termed *summative* in the literature [1].

With our tool, the instructor is able to present two fundamental question formats to the student:

- Text-based: This general format is amenable to various popular types of stimulus-response questions, including multiple choice, true/false or fill-in-the-blank. Text-based questions are convenient for testing concepts as well as snippets of a few lines of code. For our purposes, the text-based questions were of the short answer type, where the question was either based on information provided on the screen or based on a required reading from the course text.
- Code-based: In this format, the student is asked to submit a full Java program. The tool can provide the student with one or more “skeleton” files that contain the outline of the solution. In preparing an answer, the student is free to use any editor or integrated development environment (IDE). The answer submitted by the student is compiled and evaluated automatically by the software.

In order to ensure the integrity of the results submitted, we included a layer of cryptographic security. Using a public/private key system, the results file is encrypted in such a way that only the instructor can read its contents. The decrypted data contains not only the evaluation, but also every question that was presented to the student and every answer provided. This is done with the aim of resolving any dispute or correcting any error in the automatic evaluation system.

IMPLEMENTATION

This section describes the basic structure and some of the implementation details of the e-learning and e-testing tool.

Wizard

We named the application which presents the user interface and the questions to the user "wizard", because it conforms to the common paradigm of presenting a sequence of pages that can be navigated in either the forward or the reverse direction. This format is convenient because it allows students to skip questions and return to them later. We developed the wizard in Java, which is an object-oriented programming language designed by Sun Microsystems Inc. to be computer platform independent. The only requirement for running the wizard is that the computer have the Java Virtual Machine installed, which is available free of charge for both Windows and Unix-type systems.

The graphical user interface of the wizard is built with Swing, a lightweight Java library that provides a consistent experience regardless of the computer platform. The wizard is distributed to students as a Java Archive (or JAR) file, which encapsulates the entire program in a single convenient package. The overall design goal was to provide a straightforward and intuitive interface for students.

Upon launching the wizard, the main interface is shown along with navigation buttons such as "Next" and "Previous". The first page (not shown here) displays introductory remarks and collects the user's name and student number.. The second page (also not shown here) provides more detailed instructions as might be useful, for example, to indicate which sections of the book to read or to lay out rules for an examination setting.

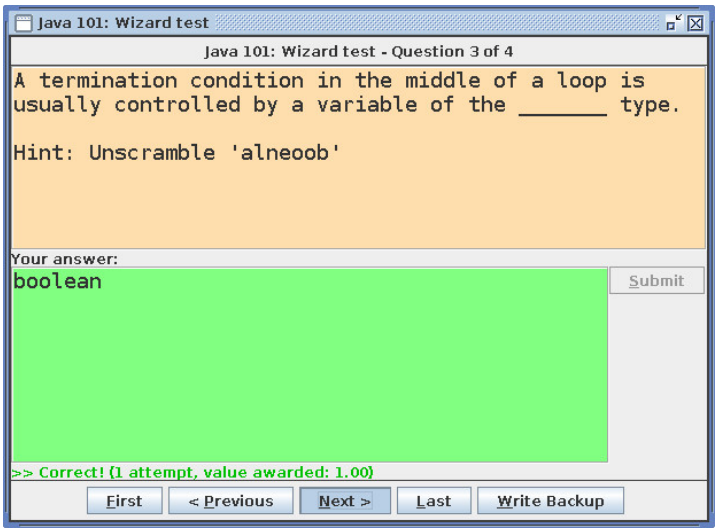

FIGURE 1
EXAMPLE OF A TEXT-BASED QUESTION WITH A $SCRAMBLE HINT AND A CORRECT RESPONSE FROM STUDENT.

After the two introductory pages, the question pages follow. The two fundamental question types, text-based and code-based, have different user interface needs. For each text-based question, the student is simply presented with a question or with information to be learned followed by a question. Space is provided for an answer (see Figure 1). Upon pressing "Submit", the wizard automatically evaluates the answer and provides visual feedback. If the answer is correct, the answer space is filled with green, as shown in Figure 1. If the answer is incorrect, an error message appears in red.

Code-based questions are administered using a similar interface, with some modifications, as shown in Figure 2.. There is an additional button, "Write Files", which writes one or more files to the current directory. The files are typically Java files. One file might be a complete driver file (with the main method), and another file might be a skeleton of the desired code containing "TODO" comments and instructions for the student regarding the functionality of what to fill in. The driver would typically contain example inputs and examples of desired outputs, along with some inputs for which the output is not given. After editing the files to complete the exercise, the student would typically compile and

run the application independently of the wizard to check that the outputs match the desired example output and that the other outputs for which the desired values are not given match what the student would expect. If there are mismatches, then the student would do another round of editing and testing. When the student is satisfied with the correctness of the program, the student presses "Submit". The wizard then compiles and executes the Java files and captures the text output. Then, as in the case of the text-based questions, it immediately evaluates the answer and provides visual feedback.

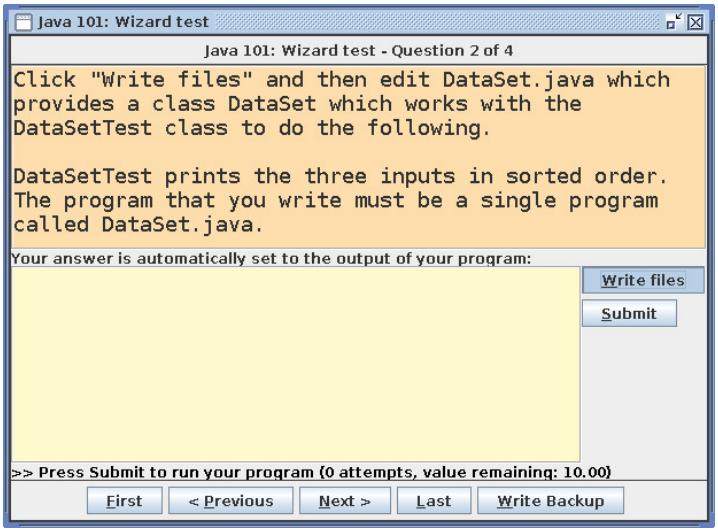

FIGURE 2
EXAMPLE OF A CODE-BASED QUESTION FOR WHICH TWO FILES WOULD BE PROVIDED TO THE STUDENT, ONE COMPLETE DRIVER AND ONE SKELETON.

Students are permitted to retry any number of times after answering a question incorrectly. However, the points available for that question declines with each failed attempt. We found that a compounding decline of 50% per attempt was a good balance between rewarding accuracy and encouraging experimentation.

The purpose of the final page of the wizard is to produce a results file, which then must be submitted to the instructor. The results file contains all of the questions presented as well as all the answers submitted, including the incorrect ones. It is a simple text file, however, it is encrypted with a public key. The private key, known only to the instructor, is able to see the decrypted results .

Question Repository

Questions and answers can be typed directly into the wizard, as part of the Java code, or for simplicity they can be placed in a separate text file (the question repository) which is parsed by another program and loaded into the wizard. Just as the wizard was designed for ease of use for students, the question repository was designed for ease of use for instructors. In particular, we designed it for simplicity in creating and editing questions. The question repository, therefore, is a single specially-structured text file which contains all the instructions, questions and answers which are to be loaded into the wizard by a separate program. In addition to specifying the questions and answers, the repository also provides a means of enabling special features. Special features are enabled with flags prefixed with the $ symbol.

Flags such as $scramble and $first provide hints to the student to reduce the difficulty or ambiguity of a question. Other flags such as $shuffle and $trim are used for introducing variety in a examination setting when it's undesirable for all students to have identical tests. The shuffle and trim operations can be used multiple times in the same repository to randomize and select specified numbers of questions from the subset of questions listed since the previous shuffle or trim. Thus, a random subset of questions can be chosen from each topic area. A detailed description of the optional flags are listed in Table 1.

Flags	Applies to	Description
$scramble	Text-based	Instructs the wizard to provide a scrambled hint to the user. The characters of the answer string are shuffled. See Figure 1 for an example.
$first	Text-based	Reveals the first character and number of characters in the answer.
$code	Code-based	Indicates a code-based question. See Figure 2 for an example.

$weight=N	Any	By default, all questions are scored out of 1. However, a custom weight (N) can be specified. Figure 2 shows a code question which has been given a weight of 10.
$shuffle	Any	Randomizes the order of all the questions specified to that point. When used multiple times, this feature shuffles groups of questions.
$trim=N	Any	Culls questions down to a specified number (N). This is used to select a subset from a larger pool of questions.
$backup=N	Any	Specifies the backup interval. A backup file is automatically written after every N questions.

TABLE 1
OPTIONAL FLAGS IN THE QUESTION REPOSITORY

EXPERIENCE

We have used the wizard as an aid in the learning and teaching of Compuer Science I to a small group of 30 first-year computer science students, most of whom were repeating the course after having had difficulty in the first term. Features were added during the term based on instructor and student feedback. In fact, at the end of each reading assignment and test the wizard presented an opportunity for the student to type in comments and suggestions, which were added to the results file. The comments were almost always positive, suggesting that the students enjoyed using the wizard and found it a valuable tool for learning. The comments were also often constructive. For example, initially the wizard did not save into the results file a submitted program if it did not compile. Students commented that they thought such programs should be included in the results file so that the instructor could possibly give partial marks. We therefore modified the wizard to save all submitted code, whether it compiled or not and whether it produced the correct answer or not. Similarly, multiple attempts on short answer questions are all saved. This gives professor and student a chance later to review fully what occurred during the exam and then to discuss it. For an early test, partial credit was manually given to code that did not compile, but later in the term when students were more used to compiling and understood that compilation is a rudimentary requirement of code, this qualitative assessment was removed. In fact, on the final exam, answers to coding questions had to meet all of the test case requirements in order to receive any credit. There were questions based on each of the chapters covered from the text, so that the difficulty of the questions ranged from very easy to somewhat advanced.

The purpose of encrypting the final results file which the students submitted was to remove the temptation of sharing the files of results among students. At the beginning of each assignment, there was a statement to the effect that students were expected to work alone. The results file (not shown) contains the encrypted (unreadable) results, followed by plain text showing the student's name and score. The name of the student and the score are also part of the encrypted text. In one case of blatant cheating, a student submitted another student's results, changing only the name in the plain text portion. The mismatch in names between the encrypted results and the plain text was caught by eye after decryption. After this unfortunate experience, the decryption program was modified to automatically check for illicit changes. A warning was added on the last page indicating that changing the plain text before submission could result in a charge of academic misconduct, since the object is not to entrap cheaters, but rather to arrange test environments so that cheating does not seem like a viable option.

The use of the wizard for reading assignments allows students to work at their own pace. Nevertheless, since the course had a normal term schedule, deadlines were set for submitting the results files. Assignments were provided and the result files were submitted on the university-wide Blackboard system. The first lab tests did not feature randomization of order of questions and randomization of selection of subsets of questions, and despite the best efforts of the invigilator, there was evidence of cheating. This problem was corrected with the introduction and use of the trim and shuffle tags described above, so that on the final exam, the eyes of the students remained focussed on their own screens. Some students adopted the strategy of restarting the exam if things were not going well, since each time the wizard was started a different exam was initiated. The time limit of the exam (three hours) set a limit on this form of experimentation. One or two students did two exams simultaneously. In the first test, some students accidentally closed the wizard before reaching the last page where the file of results is saved. To correct for this problem, a SAVE button was added to the wizard, which when pressed would save the work up to that point. This did not fix the problem entirely, however, since some students forgot to press SAVE regularly. Finally, the feature of automatic saves after N presses of the SUBMIT button was added to the wizard, where N defaults to 2.

CONCLUSION

We have presented an e-learning and e-assessment tool which we developed for a first year programming course at Ryerson University. First year programming lends itself well to quantitative testing since the ideas and programs required are quite simple, requiring little imagination. The quantitative approach would be less applicable to upper year courses in software engineering, where more *craft* is involved in the design of solutions and user interfaces [4]. The quatitative testing has a formative phase where the wizard, loaded with a reading assignment, is given to the students as homework. The students do the reading and the associated questions and submit the result file. Since this is unsupervised, students typically practice with the wizard until they get a perfect mark. The quatitative testing also has a summative phase where the students do the same or a similar assignment in a supervised lab. This provides a direct measure of at least one aspect ("detailed knowledge in an area of the discipline") of one of the undergraduate degree level expectations expressed by the Ontario Council of Academic Vice-Presidents [5]. Indeed, one of the focuses of the latter article is the need to be able to *measure* academic equivalencies, given the current global nature of higher education. Computer aided assessment such as that presented in this article is one way to address this need.

At the time of writing, the wizard has been in use for one term, and the results have been very satisfying to both instructor and students. We plan to continue its use (next on a class size exceeding 100) and to add more features. It currently offers four of the eight features used as criteria in [2] for assessment of Learning Management Systems: multiple question types, random items, feedback, proctored tests. We believe that our wizard is unusual if not unique in its ability to present and test coding questions, and in its ability to encrypt the final answer report in a way that discourages simple copy-paste cheating. Sites such as the Uva Online judge [8], also provide an environment for presenting a programming question, submitting code and having it checked against specifications, but our system differs in that it can provide skeleton files, driver files and input files to the student, and the testing is done on the student's machine. Our system is missing the following four features: support for multimedia elements, support of equations, statistical test analysis, conformance to a standard (such as QTI [6]) that facilitates sharing of question and assessment results among Learning Management Systems. We expect to add the ability to present images and sound (for example, text to speech reading of the information and questions) in the near future.

ACKNOWLEDGEMENT

This work was supported by the Department of Computer Science, Ryerson University.

REFERENCES

[1] Boston, C, "The Concept of Formative Assessment", *ERIC Clearinghouse on Assessment and Evaluation*, College Park, MD, http://www.ericdigests.org/2003-3/concept.htm, 2002. Last visited 24/06/2010.

[2] Costagliola, G, Fucella, V, "Online testing, current issues and future trends", *Journal of e-Learning and Knowledge Society*, 5, No 3, 2009, 79-90.

[3] Graff, M, "Cognitive Style and Attitudes Towards Using Online Learning and Assessment Methods", *Electronic Journal of e-Learning*, 1, No 1, 2003, 21-28.

[4] Hamlet, D, Maybee, J, *The Enginering of Software*, Addison Wesley, 2001.

[5] Ontario Council of Academic Vice-Presidents, "OCAV Undergraduate Degree Level Expectations", http://www.cou.on.ca/content/objects/Undergrad%20Degree%20Expectations%20FINALen1.pdf, last visited 24/06/2010.

[6] Smythe, C, Shepherd, E, Brewer, L, Lay, S, "IMS Question & Test Interoperability: An Overview", http://www.imsglobal.org/question/qtiv1p2/imsqti_oviewv1p2.html, last visited 24/06/2010.

[7] Woodford, K, Bancroft, P, "Using multiple choice questions effectively in Information Technology education", in Atkinson, R, McBeath, C, Jonas-Dwyer, D, Phillips, R. (Eds), *Beyond the comfort zone:Proceedings of the 21st ASCILITE Conference*, Perth, 2004, 948-955, http://www.ascilite.org.au/conferences/perth04/procs/woodford.html, last visited 24/06/2010.

[8] Uva Online Judge, http://uva.onlinejudge.org/index.php?option=com_comprofiler. Last visited 24/06/2010.