Experience with a Computer-Assisted Formal Programming Examination

John English
School of Computing and Mathematical Sciences
University of Brighton,
Brighton BN2 4GJ, UK
(+44) 1273 642672
je@brighton.ac.uk

ABSTRACT
This paper describes a web-based system for the online delivery of formal examinations and their automated marking. This system was first used in June 2001 in an end-of-year exam for a first year undergraduate programming course. The outcome of this experiment is also described.

Categories and Subject Descriptors
K.3.1 [**Computers and Education**]: Computer uses in education – *computer-managed instruction.*

General Terms
Management, Measurement, Experimentation, Human Factors.

Keywords
Automated assessment, formal examination, programming.

1. INTRODUCTION
In common with many other institutions, the University of Brighton has experienced a sharp growth in student numbers in recent years. Increased numbers make it more difficult to assess student attainment; if assessments are graded manually, staff must either set fewer assessment tasks or resign themselves to a greatly increased marking load. There are also problems with plagiarism [7]. The University of Brighton has, like many other institutions, been experimenting with a variety of online assessment techniques to alleviate these problems [2,3].

One of the difficulties with teaching programming is finding ways to assess students which assesses practical skills but which also prevents plagiarism. One approach which has proved suitable in other modules is to set individual pieces of work; for example, the author has used this approach with great success on a computer architecture module and an introductory compiler construction module, where the basic problem is the same but the particular data given to each student is randomly generated on an individual basis. However, it is much more difficult to parameterise programming problems in this way. If the basic problem is the same, the solution is essentially the same, with only minor changes being required to accommodate an individual set of data.

ITiCSE'02, June 24-26, 2002, Aarhus, Denmark.

The traditional approach to avoid the potential for plagiarism is to use a formal examination, but it is unfair to expect students to produce working programs in a traditional written examination. A solution which has been used by several others [1,5,6] is to conduct a formal computer-based examination which allows students access to compilers and other tools so that they can develop and test practical solutions to the questions they have been set. This also has the additional advantage that the submissions will be available in a machine-readable format which is amenable to some form of automated marking system.

This paper describes an online programming examination which was taken in summer 2001 by a cohort of 64 students. The exam paper and submission system were web-based using an HTML form, and the marking workload was drastically reduced by using a semi-automated marking system similar to that described by Jackson [4] consisting of an automated marking phase followed by a much faster human moderation of the automatically-generated marks.

2. IMPLEMENTATION
The exam was implemented using a system devised by the author. This consists of a CGI script which generates an HTML form from an XML representation of an exam paper, based loosely on the IMS Question and Test specification [8]. Questions of several different types are supported:

- Free text;
- Multiple choice (choose one right answer out of a set of possibilities);
- Mutiple select (choose all right answers out of a set of possibilities); and
- Multiple match (arrange a set of answers into the correct order).

The order of questions within a section can be randomised, as well as the set of answers for a particular question where there is one. Randomly-chosen values from a specified set can also be used to individualise particular questions if desired.

The HTML form generated by this system includes an applet which transmits the contents of the form to a special-purpose

server application every 30 seconds. This information includes the username, the student's registration number, and the seed value used by the random number generator to construct the exam paper. The server simply stores this data using the username and the current time to generate a unique filename. This guards against potential system crashes, since the last set of data could be used to restore the exam paper for each student to the state it was in within 30 seconds of the crash.

The server application is controlled by a configuration file which specifies the server port number, the name of the XML exam paper, the time allowed, and specific time extensions granted to students with special needs (e.g. dyslexia). For health and safety reasons, it was also a requirement to allow students to take a break from using the computer, so a separate allowance is also included for the permitted break time (and extra break time for individual students where necessary).

The generated HTML exam form also includes JavaScript to obtain the time remaining and the remaining permitted break time from the applet and display it in the browser's status bar. To deal with permitted break time, the applet fetches the time remaining and permitted break time when it is first loaded (from its *init()* method) and its *destroy()* method notifies the server when it is terminated. The server logs these events for each user and uses them to calculate the remaining time whenever the applet is reloaded.

To take a break, a student merely needs to shut down the browser (thereby killing the applet). The server deducts the time between the applet shutting down and restarting from the permitted break time, or if the break time has been exhausted, from the time limit for the exam. When the time limit for the exam expires, no further submissions are allowed; the server stops recording the content of the user's exam form, while at the same time JavaScript in the exam form displays an alert box to notify the student that the exam is over.

3. THE EXAM

The examination described here was an open-book exam conducted in our computer laboratory for a first year undergraduate programming module using Ada 95. It lasted a total of three hours: 2.5 hours online, with a total of 30 minutes permitted break time.

The exam consisted of three sections. Section A contained ten multiple-choice questions worth a total of 30%; section B contained five short free-text practical questions worth a total of 30%; and section C contained two longer free-text practical questions worth 20% each. One of the section C questions was a program with a mixture of syntax and logic errors to be fixed, the other was a package specification which contained the declaration of two functions, for which the corresponding package body needed to be implemented. Examples of questions from each section are shown in Figure 1 above.

To prevent online collusion, the machines that the students were using were logged in to specially-created accounts with no external Internet access, no email account, and no printing facilities. The exam paper was loaded from an authenticating server using Internet Explorer, which remembers passwords for authenticating sites and uses these to log in automatically. This meant that students did not need to know the password for the accounts they were using, so it would not be possible to log into the account from elsewhere.

Section A:

Which of the following Boolean expressions is true if the value of the integer variable A is between 1 and 10 inclusive?

- not (A < 1 or A > 10)
- not (A < 1 and A > 10)
- A > 1 and A < 10
- A > 1 or A < 10

Section B:

Write a loop statement which will scan a string variable called S and put the total number of upper-case vowels (A, E, I, O or U) in a variable called V. Your solution should work for any string, no matter what its length or bounds.

Section C:

The program below (*not shown*) is intended to read in an array of numbers (terminated by anything which is not a valid number), sort them into ascending order and print them out. For example, if the input is:

5 8 3 4 7 #

the output should be

3 4 5 7 8

In its present state, the program does not compile.

You are required to:

- correct the errors so that the program compiles successfully.
- correct the code so that it generates the correct output.

Figure 1: Sample questions

The copies of Internet Explorer were initially set up to point to a form giving the exam rubric and guidance information. The students entered their registration number on this form, in the same way as they would enter their registration number on the answer book for a standard written examination. They also had to fill in the standard paper examination slip giving the exam code, their name and registration number and the desk number (in this case the username that the machine had been logged in under). The administrative requirements were therefore exactly parallel to those for a conventional written exam.

When the students pressed the button to start the exam, the CGI script described above recorded a log entry for the start of the exam and generated an individualised copy of the exam paper.

4. MARKING

After the end of the examination, a marking script was run which regenerated the questions given to each student and marked the submitted answers. The system can mark all types of questions except free-text questions with perfect accuracy, but free-text

questions are more problematical. In the XML form of the exam paper, a free-text question specifies the name of a command to be executed to process a submission. In this particular case, the command would embed the submission into a test program and attempt to compile it. If this was successful, it would run the result several times against supplied sets of test data, using a sandbox to guard against infinite loops, excessive amounts of output and a variety of other possible problems.

The output in each case would be compared against a set of expected results (generated by running a model solution against the test sets). Additional marks might be awarded for particular types of solution, so additional checks were made for significant coding constructions in the submission. For example, consider this Section B question:

> *Given a time as the number of seconds since midnight in an integer variable called Seconds, write a sequence of Ada statements which will store the time as a number of hours, minutes and seconds in three variables called H, M and S respectively.*

A bonus mark was awarded if a division operator was used, and another if a remainder operator was used (**mod** or **rem** in Ada). The lack of a division operator was taken to imply that a relatively inefficient subtraction loop had been used, and similarly the use of a remainder operator was taken to indicate an efficient solution. A mark was then awarded for each set of test data handled correctly for a total of six marks altogether.

If a submission failed to compile, or if none of the test cases produced the correct output, the submission was flagged for manual checking. Correctly processing any of the sets of test data was taken to indicate a partially correct solution which had met some of the required criteria but had failed on the others, and the marks could therefore be assumed to be accurate.

The moderation process involved printing out a complete set of submissions and a breakdown of the automatically-generated marks, and leafing through this looking for messages requesting a manual check. Most submissions required at least one such check, but the time taken to moderate 64 scripts was about two hours, which is estimated to be at least five times faster than the time it would have taken to mark the entire set of submissions by hand. Because this was the first time this system had been used, all questions and their marks were checked during moderation, so the time saving noted here is in fact a very conservative one.

5. RESULTS

The exam was undertaken by a cohort of 64 students in June 2001. The students could use textbooks and course notes during the exam, as well as locally-available online material and development tools. They were also allowed to leave the laboratory at any time, although they were not permitted to take out or bring in any written notes once the exam had begun.

The results of the exam are summarised in the chart given as figure 2, which shows the spread of marks before and after the moderation process. Moderation added an average of 8% to the marks for each submission, raising the average mark from 41% to 49%. The frequency of changes in marks (ranging from 0 to 20%) is shown in figure 3.

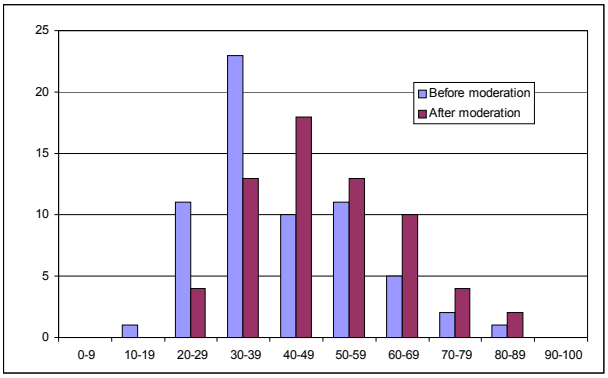

Figure 2: Marks before and after moderation

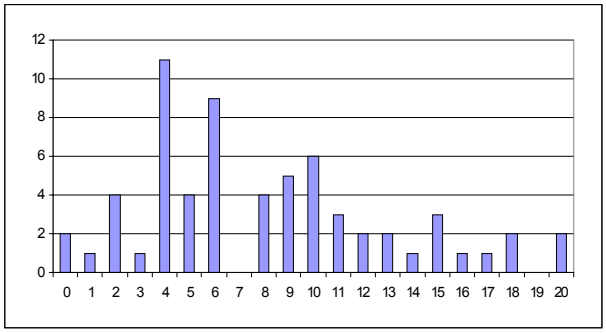

Figure 3: Number of marks added during moderation

The bulk of the problems arose from students not following the instructions in the question sufficiently closely. Common mistakes included:

- inventing (but not declaring) unnecessary variables to hold temporary results;
- attempt to input values into the variables named in the question when not requested to do so;
- producing output when not requested to;
- using different variable names to those in the question.

These errors would result in compilation errors or in no correctly processed test sets. Manual moderation was concerned with factoring out mistakes like these and adding marks that would otherwise have been recorded, while levying an appropriate penalty for the error.

The availability of development tools made it possible for students to test directly whether the answer to a particular question was correct. For example, the answer to the sample Section A question shown in figure 1 could be determined by trying each answer in turn in a simple test program. However, there is inevitably a time penalty for creating such a test program compared to being able to spot the correct answer without the need for external confirmation. In addition, constructing a suitable test program is in itself a good measure of practical programming ability. The fact that students could empirically test their solutions was in important part of the rationale for this exam, and the use of empirical testing even for multiple-choice questions in no way

invalidates this exam as a way of assessing practical programming ability.

Plagiarism was still possible during breaks. Breaks could have been supervised to prevent this, but since the questions were randomised, it was felt that this would hamper students sufficiently from communicating solutions to specific questions. The restriction on bringing in written notes after the start of the exam was felt to be an adequate precaution against plagiarism, and the results seem to bear this out. However, further analysis is planned to determine who took breaks together and whether subsequent submissions show evidence of collusion before the exam is run again in 2002.

6. CONCLUSIONS

This system has only been used once so far, but it has proved to be an extremely successful experiment. In the author's opinion, this examination gave a much more accurate picture of individual practical programming ability within the cohort than other forms of assessment that have been tried in previous years. It also greatly reduced the staff workload by partially automating the marking process.

One huge benefit is that the experiment has yielded a large corpus of data: 2.5 hours work by 64 students sampled at 30 second intervals, or 19200 data points. One possibility is to examine this for evidence of plagiarism, as noted above, which would help to refine the rules for conducting future exams. Other data on the frequency of particular types of error might also prove valuable.

With the success of this experiment, authoring tools to enable others to make use of this system are a prime requirement. Work is currently underway to develop suitable tools and to integrate the exam delivery system with other online assessment tools that have already been developed by the author and his colleagues.

7. REFERENCES

[1] Arnow, D. and Barshay, O. Online Programming Examinations using WebToTeach. Proceedings of ITiCSE '99 (Cracow, June 1999). ACM Press, 21–23.

[2] Davies, P., Hansen, S., Salter, G. and Simpson, K. Online Assessment with Large Classes: Issues, Methodologies and Case Studies. Proceedings of WebNet '99 (Honolulu, October 1999). AACE, 1498–1499.

[3] English, J. and Siviter, P. Experience with an Automatically Assessed Course. Proceedings of ITiCSE 2000 (Helsinki, July 2000). ACM Press, 168–171.

[4] Jackson, D. A Semi-Automated Approach to Online Assessment. Proceedings of ITiCSE 2000 (Helsinki, July 2000). ACM Press, 164–167.

[5] Mason, D.V. and Woit, D. Integrating Technology into Computer Science Examinations. Proceedings of SIGCSE '98 (Atlanta, February 1998). ACM Press, 140–144.

[6] Medley, D.M. Online Finals for CS1 and CS2. Proceedings of ITiCSE '98 (Dublin, August 1998). ACM Press, 178–180.

[7] Ryan, J.J.C.H. Student Plagiarism in an Online World. Prism (December 1998). ASEE. Available online: http://www.asee.org/prism/december

[8] Smythe, C., Shepherd, E., Brewer, L. and Lay, S. IMS Question & Test Interoperability Specification 1.2 (September 2001). Available online: http://www.imsproject.org/question/index.html