Experiences with Online Programming Examinations

Monica Farrow and Peter King
School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh EH14 4AS

Abstract

An online programming examination was used to assess undergraduates who had learnt Java for two terms. The advantages are that the students are assessed on what they have to do in practice, rather than on their theoretical knowledge, and that the marking load on the examiner is drastically reduced. The style of question used and the results of the examination are analysed.

1 Introduction

In the last ten years, since the release of Java on the unsuspecting world, object oriented programming has become the paradigm of choice for many initial programming courses. The question of how to best introduce objects leads to (at least) two approaches. The traditional approach was to introduce programming with a procedural paradigm and gradually introduce the idea of objects as data structures with associated operations packaged with them. The second approach is often known as objects-first and uses a specialised environment to introduce objects independently of one another, with no obvious programming support, gradually increasing the amount of programming students do to both use and construct their own objects. After a few years using the traditional approach, Heriot-Watt has moved to the objects-first approach, using the BlueJ environment.

Assessing programming skills is notoriously difficult under examination conditions. Few programmers write code that is correct first time, and to expect them to produce such code under time pressure even for small examples is unrealistic. In normal programming practice, a programmer will design an algorithm, code it, and then iterate through a compile and test cycle. Typically the compiler will be used to catch simple syntax errors such as missing

semicolons, mismatched parentheses, etc. When marking a conventionally written examination, the examiner is having to guess whether the syntax that the student has written is wrong because of a simple slip, or because the student doesn't understand the language. In addition, the examiner has to decide whether or not the student's code meets the specification asked for. This is not always obvious.

An online examination would allow the student to use the mode of working that is familiar when developing code. However, the time to design and code any significant application would exceed the time available for a reasonable length examination. It was decided to experiment with an online programming examination in which students would be given significant amounts of source code and asked to correct it if necessary and to complete it to some specification.

2 BlueJ

BlueJ[4] is a programming environment developed by Deakin University, Melbourne, Australia and the University of Kent, England. It supports object oriented programming in Java, and is freely available as a download from the Internet. An increasing number of textbooks either use it directly or refer to it as a mechanism for executing programming exercises. In addition to providing a text editor (specialised to Java to some extent), it also allows the user to compile the resulting code and to instantiate objects, independently of any particular program. Of course, the objects so instantiated are executing as part of the BlueJ environment, but to the user they appear to exist independently.

In addition to these features, the environment allows extensions to be provided using a plug in mechanism. These extensions provide support for additional amounts of UML, aspects of coding style checking, logging user behaviour, etc. Of particular interest to us is the Submitter extension, that allows a student to send a project to another person, or to a program. This can submit an arbitrary set of files from the BlueJ project to a marking system and display the output from the marking system to the student.

3 JUnit

JUnit[2] is a tool developed by Kent Beck to support the Extreme Programming methodology of continuous system integration with automated regression testing. Essentially it allows the user to write a Java class (or classes)

containing methods that test aspects of the system being constructed. A *test suite* is a single Java class with an identifier that ends with `Test`, containing a number of test methods. Conventionally, each test method within a suite tests a single aspect of the system being constructed, and each test should be as far as possible independent of other tests. JUnit is now packaged with BlueJ, so that students can develop code in the extreme programming style, although we have not been active in encouraging this style with our elementary programming classes.

4 Examination Structure

With the availability of these tools, we decided to run an examination at which the students would have to correct compilation errors and enhance existing software. The examination would be marked entirely automatically, by using scripts to compile the student's code and to run JUnit tests on the resulting classes. Marks would be awarded for correcting the compilation errors, and also for each test that was successfully passed.

The approach taken involved providing partial source code, which the student could then alter or extend. This approach has several benefits:

1. The supplied code can be used to provide examples of Java syntax, rather than compelling the student to remember this exactly. In addition to reducing the time spent typing in straightforward program code, this also introduces an element of code-reading into the examination.

2. It continues the approach used in the weekly laboratory exercises, which were taken from the textbook 'Objects First with Java'[1]

3. JUnit testing is more likely to succeed because the correct class names or method signatures have been provided.

It was not practicable to alter the lab to prevent the students using the Internet, email or their previous work. Even if had been practicable to isolate the lab from the Internet, the departmental intranet was still required for the JUnit testing. The questions were therefore written so that students would not need to look up documentation or previous exercises. Some useful examples of code were provided within the supplied methods, and some documentation on relevant Java library methods were given within the instructions. Access to the standard online documentation for Java was available in any case. The students were trusted not to email each other, and the invigilators did not see any such activity.

The students were supplied with four BlueJ projects. Within each project, the source code for one or more classes were provided, containing errors or omissions within the methods. An accompanying document gave an introduction and a list of the required tasks for each project.

4.1 Marking Mechanism

At any stage, the students could submit a project for assessment. The project is then run against one or more JUnit test suites. The student can view the result of these tests to discover how many tests were successful, and to obtain details of compilation and test errors. There is no limit to the number of times that a project can be submitted.

The marks obtained for each project are determined by multiplying the number of successful tests by an integer scale factor which is associated with the corresponding test suite. Additionally, marks can be obtained for successful compilation. The student's overall mark is calculated by summing the maximum mark obtained for each test suite. The student can view their marks at any stage.

Submissions are stored in the lecturer's workspace. There is a folder for each student, and within that, folders for each submission. A log file lists whether project compilation was successful or not, and the number of successful tests for each test suite. Each submission folder contains the source code and a text file containing a report on compilation and testing. This allows the student's whole submission history to be reconstructed for examination audit or appeal processes.

After the exam is over, the lecturer runs a Perl script to extract the marks for each student. The Perl script outputs a text file containing a line for each student in a format suitable for further processing. This line contains the final mark, the overall time taken, and, for each project, the number of attempts made, the success of the compilation, and the marks obtained from testing.

5 Questions

Suitable questions for this style of examination are different from those in more conventional examinations. The marking mechanism means that no account can be taken of coding style or documentation. The only factor that can influence the mark is the success or otherwise of an objective test.

It is essential that the questions are phrased in an unambiguous manner such that correctly answering the question will successfully pass all the JUnit

tests associated with that question. One question used in a previous year's examination asked students to complete the implementation of a `toString` method. This turned out to be a poor question, because it is difficult to test such a method without providing such a precise format to the student that the question is almost trivial.

Each question consists of a list of tasks to be completed. The associated BlueJ project contains incomplete source code for one or more classes. The projects provided were

1. A very simple class holding details of the name and year of a student, with a constructor and a `toString()` method. This class contained a few compilation errors such as incorrect spelling or missing keywords, and a couple of logic errors in the instance variable assignment and `String` concatenation.

   The code was tested by constructing an object and checking the output of the `toString` method.

2. A class representing a number of silver coins. Some methods were provided and other methods had skeleton implementations for the student to complete. This tested the use of simple expressions and the use of conditional statements.

   The code was tested by constructing several objects with different properties, and checking the `String` or numeric output of the methods.

3. Two classes modelling different employee grades. They extend the same abstract class which holds the common attributes of an employee. Students were required to implement an abstract method for both subclasses.

   The code was tested by constructing objects of each class, and checking the numerical output of a method. With this question, it would have been possible for the student to alter the supplied classes so that inheritance was not used. However, if they had done this, it would demonstrate at least that they understood which components to remove!

4. A class which displays a GUI containing textfields and buttons. Students were required to fill in the body of the `actionPerformed` method, setting the contents of a textfield depending on the contents of two other textfields.

   The code was tested by initialising the input textfields and generating the `Event` object corresponding to clicking the button. This will trigger

a call on the `actionPerformed` method and the test can then inspect the result textfield.

The execution of a test like this requires a little cunning on the part of the system. If the web server on which all the tests were run was to execute the tests naively, potentially thousands of windows would be opened and manipulated by its X display manager during the running of the exam, potentially overloading the server. In addition, the main screen associated with the server would have many windows appearing and disappearing! To avoid this, a dummy display was used, by running the X virtual frame buffer server[5]. The testing script then opened any windows specified by programs under test on the virtual frame buffer, effectively ignoring the visual output.

6 Analysis of Results

53 students sat the exam. The maximum mark was 30, and this mark was obtained by 15 students. Most of the students taking this module are in their first year studying for a degree in (or with) computer science or in their second year studying for a degree in information technology. The information technology degree is designed to be suitable for students who are less able mathematically. The frequency of the different scores in the exam is shown in Figure 1. The most obvious feature of this histogram is the two-humped appearance, suggesting two distinct groups of student ability, as observed by other authors[3].

21 out of 38 computer science students (55%) achieved a score of over 70%. A significant number of computer science students and almost all the information technology students have scores of less than 40%. It does raise the question as to whether this module is suitable for the information technology students, and highlights the divide between students who are coping with programming and those who are not.

6.1 Sample Exam

To give students a chance to familiarise themselves with the submission system and the style of questions, a sample exam of similar questions was made available one month before the final exam. A laboratory class was dedicated to helping students use the submission system. When the class was over, the questions and marking system remained available for further practice.

37 students, from the class of 53, attended the lab class. Of these, eight continued to use the system during the following week. In the week immedi-

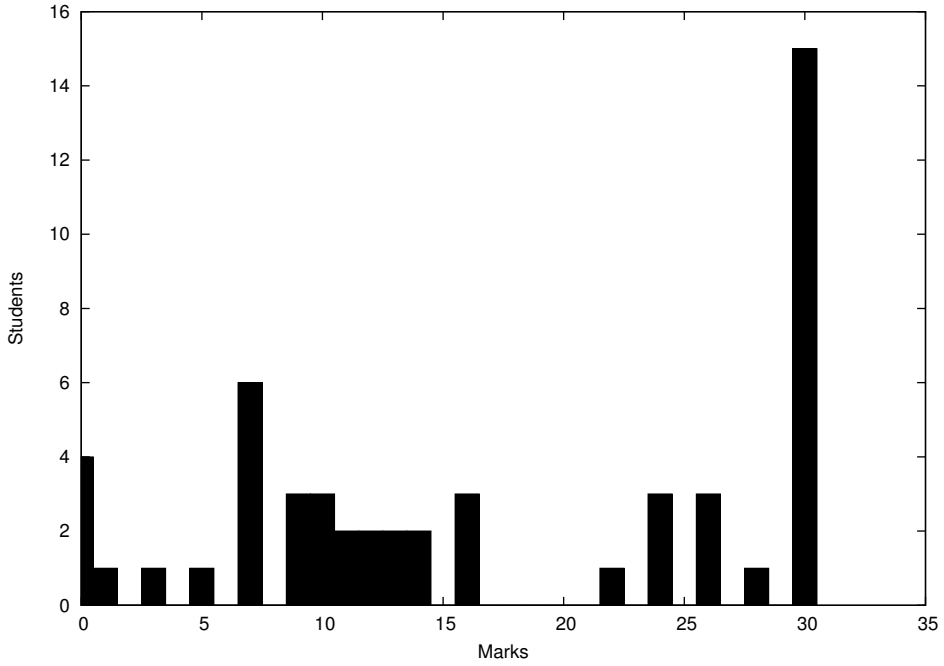

Figure 1: Frequency of Final Scores

ately before the exam, six students used the system for a significant amount of time. Four other students made single submissions.

Figure 2 shows the students' percentage in the sample examination compared to their percentage in the actual examination. In the sample exam, only four students got full marks. This is possibly due to minor problems with one of the questions, or possibly because the system was new to them. In general, it can be seen that students improved their performance in the final exam over that in the sample exam.

6.2 Overall time taken

Figure 3 shows the total number of marks achieved against the time taken.

The time taken was measured between first and last submissions and so is slightly shorter than the actual time taken by each student. However, the first question was very straightforward, and it is likely that most students either corrected it within a few minutes, or submitted it almost immediately to see the result. Ideally the student should work on the project and test it within BlueJ, but observations within the lab show that many students do try a submission fairly quickly, in order to see what errors the test suite shows up.

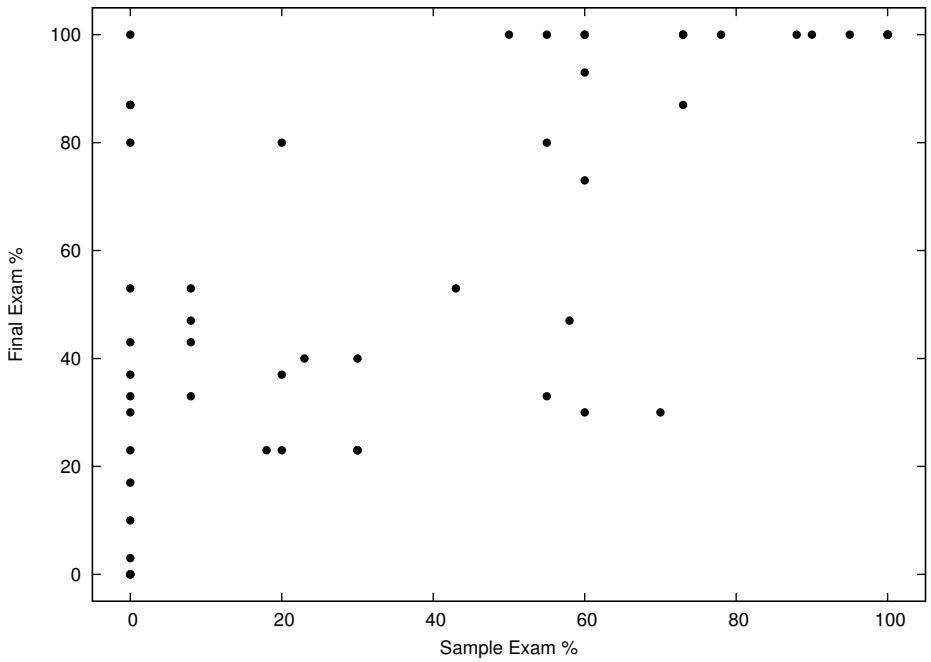

Figure 2: Final Exam % against Sample exam %

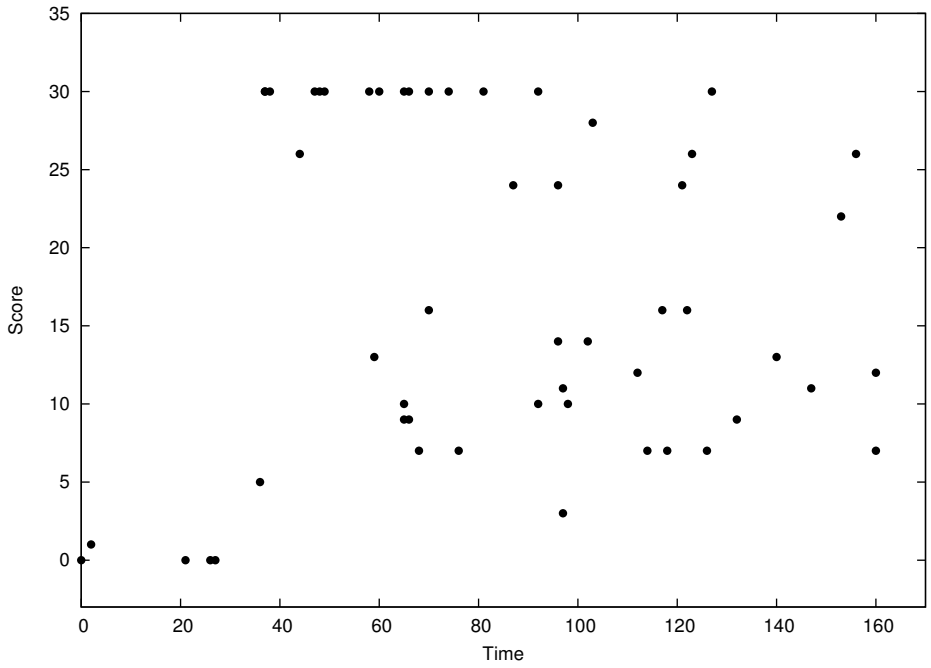

Figure 3: Total marks against Time Taken

For all the marks obtained, there is quite a spread of time taken for that mark to be achieved. It can be seen that there is no general relationship between time taken and marks obtained. Some students spent over 2 and a half hours at the exam and still obtained fewer than 10 marks. Students who obtained full marks took between 40 minutes and 2 hours to do so. The second author of this paper, an experienced Java programmer, took about 30 minutes to do the exam when he tackled it as an unseen exercise, so 40 minutes is approaching the lower limit of time to be expected among novice programmers.

The number of attempts is for the project as a whole, rather than for the individual alterations required, or the individual tests. Figures for attempts at individual tests would not have been relevant, since it is likely that a student would submit to check one particular alteration, and in this case they would inevitably get errors in tests for other alterations which they had not yet considered.

6.3 Attempts made

The number of attempts made by each student was also recorded. It is interesting to see the relationship between the total number of attempts made on a question and the number of marks achieved in that question. This relationship may indicate the difficulty of the question, or be related to the number of separate tasks involved in that questions.

Almost everyone completed question one successfully after very few attempts.

6.3.1 Silver coins - question 2

Figure 4 shows the number of students who made that number of attempts on question 2, and the number of marks they obtained for that question. For example, there were 5 students who made 3 attempts on the question and obtained full marks. Similarly, 5 students made 4 attempts in total, but only obtained partial marks. A single student made 4 attempts but received no marks for the question. This question contained a number of distinct tasks which may account for the long tail in the number of submissions.

6.3.2 Employees - question 3

Figure 5 shows the number of students who made that number of attempts on question 3, and the number of marks they obtained for that question. For this

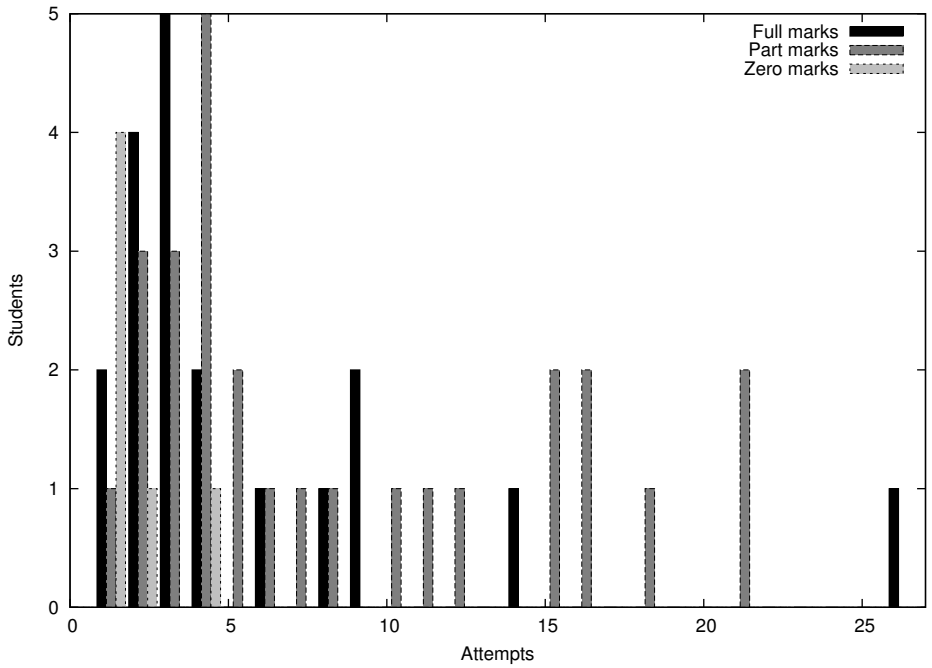

Figure 4: Silver coins - Marks against Attempts

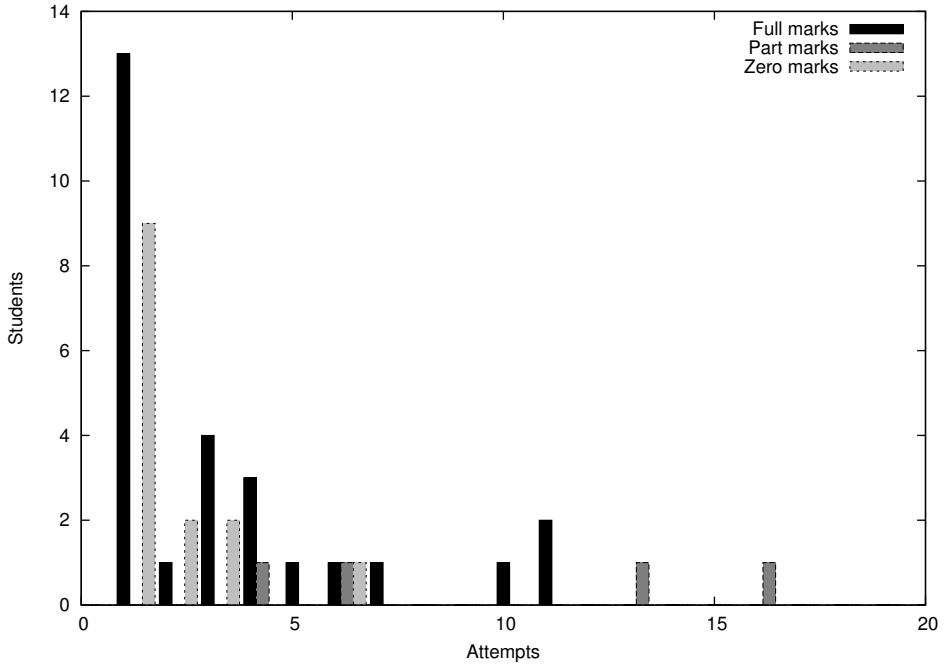

Figure 5: Employees - Marks against Attempts

question, it made no sense to submit an answer until both the constructors and the implementations of the abstract method had been completed.

Here the vast majority of students achieved full marks after only a single attempt although there is a significant tail of students who took multiple attempts. Perhaps surprisingly, there is also a significant number of students who received no marks at all and only made a single attempt on the question.

6.3.3 GUI - question 4

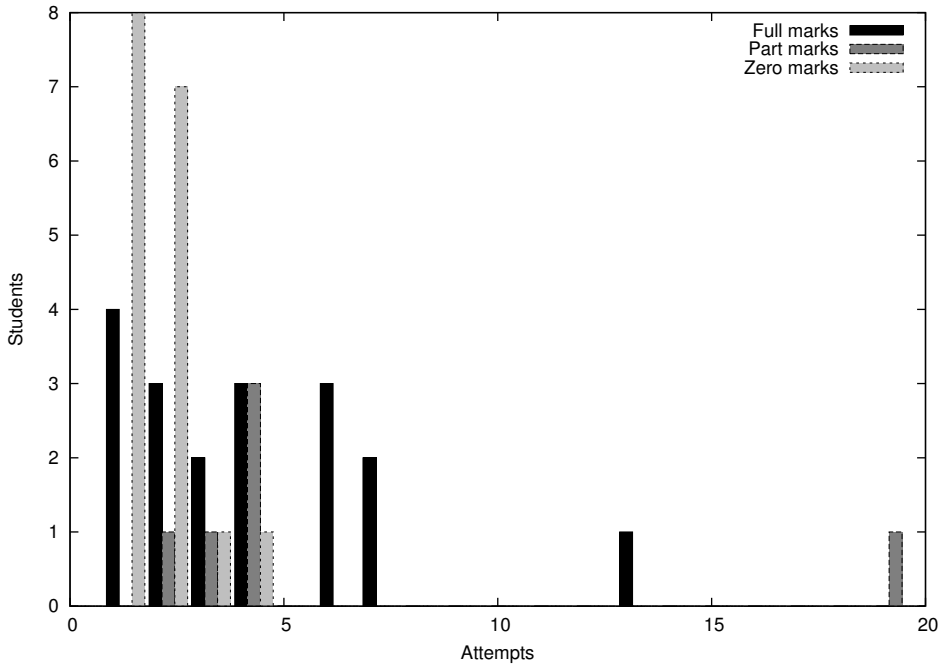

Figure 6: GUI - Marks against Attempts

Figure 6 shows the number of students who made that number of attempts on question 4, and the number of marks they obtained for that question.

For this question the number of students receiving no marks at all was higher. Most of these students gave up after one or two attempts. For those students who were successful, the number of attempts ranged between 1 and 19.

7 Conclusions

The use of online examinations for programming can be considered a successful experiment. In addition to the benefits for the student of having a

familiar environment and the usual support tools for programming at hand, there is a large benefit for the examiner since the marking is all carried out automatically and the results are available as soon as the examination is over. The student can also obtain his or her mark at any time during the examination. Since every submission is recorded, it is possible to perform later analyses of the detailed responses to individual questions.

The intention of this online examination was to provide an environment closer to real-life programming than a purely written examination. It tests the ability of the student to eventually achieve correct code. An informal comparison of the students' results in the online exam compared with their performance in the other conventional exams in the same diet showed a correlation between their conventional exam mark and the online mark of around 0.8, indicating that the online exam is testing something different, but it is not discriminating unfairly against good students.

It was hoped that the removal of the time limit would result in a more relaxed atmosphere during the examination. It was noted by the invigilators that no-one asked to leave the room for a comfort break during the entire period. This is very unusual, and can be interpreted as proof that the online examination was a less stressful and more engaging experience than a traditional examination.

Although our initial examination took some weeks to prepare, most of the effort was involved in modification of scripts and design of how to store the submitted code. The actual preparation of the questions was not significantly more difficult than preparing good questions for a conventional written exam.

The authors are greatly encouraged by the success of the experiment and look forward to developing more demanding question types for advanced classes.

References

[1] D.J. Barnes and M. Kölling. *Objects First with Java*. Prentice Hall, 3rd edition, 2006.

[2] Kent Beck and Erich Gamma. Junit. Website: `http://www.junit.org`. Last visited 21 August 2006.

[3] Saeed Dehnadi and Richard Bornat. The camel has two humps. Working paper, School of Computing, University of Middlesex, February 2006.

[4] Michael Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. The BlueJ system and its pedagogy. *Computer Science Education*, 13(4):249 – 268, December 2003.

[5] XFree86. *Xvfb virtual framebuffer X server for X Version 11.* Unix manual page.