Laboratory Exams in First Programming Courses

Quintin Cutts
Department of Computing Science
University of Glasgow
quintin@dcs.gla.ac.uk

David Barnes
Computing Laboratory
University of Kent
D.J.Barnes@kent.ac.uk

Peter Bibby
Computing & Electronic Technology
University of Bolton
P.Bibby@bolton.ac.uk

Jim Bown
Complex Systems
University of Abertay, Dundee
J.Bown@abertay.ac.uk

Vicky Bush
Multi Media & Computing
University of Gloucestershire
vbush@glos.ac.uk

Phil Campbell
Computing
London South Bank University
campbep@lsbu.ac.uk

Sally Fincher
Computing Laboratory
University of Kent
S.A.Fincher@kent.ac.uk

Stephan Jamieson
Department of Computer Science
Durham University
stephan.jamieson@durham.ac.uk

Tony Jenkins
School of Computing
University of Leeds
tony@comp.leeds.ac.uk

Michael Jones
Computing
Bournemouth University
mwjones@bournemouth.ac.uk

Dimitar Kazakov
Department of Computer Science
University of York
kazakov@cs.york.ac.uk

Thomas Lancaster
Department of Computing
University of Central England
Thomas.Lancaster@uce.ac.uk

Mark Ratcliffe
Computer Science Department
University of Wales, Aberystwyth
mbr@aber.ac.uk

Monika Seisenberger
Department of Computer Science
University of Wales, Swansea
M.Seisenberger@swansea.ac.uk

Dermot Shinners-Kennedy
Department of Computer Science
University of Limerick, Ireland
dermot.shinners-kennedy@ul.ie

Carole Wagstaff
School of Computing
University of Teesside
C.A.Wagstaff@tees.ac.uk

Linda White
School of Computing & Technology
University of Sunderland
white.holmes@sunderland.ac.uk

Chris Whyley
Department of Computer Science
University of Wales, Swansea
C.J.Whyley@swansea.ac.uk

Disciplinary Commons Web Page: http://www.cs.kent.ac.uk/~saf/dc

ABSTRACT

The use of laboratory examinations to test students' practical programming skills is becoming more common in first programming courses, in particular to counter plagiarism and increase validity. In this paper, we outline and compare 7 such examination techniques used by members of the Disciplinary Commons project. The reliability, validity and scalability of the exams are assessed, highlighting the appropriateness of some methods for particular environments. Implicit costs as well as reported benefits are given.

Keywords

Introductory Programming, Scholarship of Teaching and Learning (SoTL), Laboratory Examination, Summative Assessment

1. INTRODUCTION

The *Disciplinary Commons* is a project whereby teachers come together to share and document their practice through the production of course portfolios. In the academic year 2005/6, 18 teachers of introductory programming courses in different institutions met together every four weeks to discuss and document their teaching.

During the discussion of assessment practices, it became clear that a range of laboratory-based exam-style assessments of programming skills were being used by different members of the group. Whilst papers in the literature report on isolated examples of laboratory programming exams [1, 2], the *Commons* represents a unique opportunity to draw out common and divergent factors in the design of such assessment methods.

Across higher and further education, laboratory examinations are increasingly being used to assess students. There are two strong impetuses for this change: practical skills are assessed under exam conditions without fear of plagiarism or contract cheating, increasing reliability; and there is a general view that an exam requiring a problem to be solved, coded and debugged at a machine is a more valid assessment of programming skills than the more traditional written exams.

Providing a valid and reliable examination environment at a reasonable cost is however not so straightforward, and each of the examination models presented in this paper represents a trade-off in this respect.

The paper proceeds to define essential characteristics of a laboratory examination, before outlining the format and rationale of the seven *Commons* exam models. The similarities and differences of the models are then explored with respect to reliability, validity and scalability.

2. CATEGORISING ASSESSMENTS

In order to qualify as a *laboratory examination,* an assessment must contain some elements of what are commonly considered as *exam conditions*. For example, students are not permitted to confer with one another verbally or electronically; there is a time limit; the students may or may not have seen the question paper in advance, and may or may not have access to books, notes etc. While such characteristics are common to most coursework, they are rigorously enforced in a laboratory exam.

Of the 16 institutions taking part in the *Commons* project, 7 use a laboratory examination as defined above and 9 do not.

Figure 1 gives a breakdown of some of the key characteristics of these examinations. The meanings of each characteristic are as follows:

Students gives the number of students currently being assessed, or if a range is given these are known minimum and maximum class sizes used with this method.

Frequency indicates how many times the exam format is used in a run of the course to which it relates. */yr*, */smstr* indicates how long the course is.

Sessions indicates how many sittings are required in order to complete one run of the exam. Multiple sessions often reuse the existing timetabled weekly laboratory slots, whereas a single session with all students may use a specifically timetabled slot. A requirement for multiple sessions is typically due to the size of the laboratory or timetabling constraints that may not permit the whole class to be gathered in the lab for a significant period of time.

Versions indicates how many different versions of the exam are required for a single run. Multiple versions are typically required to ensure fairness in the presence of multiple sessions: students in a later session may otherwise have an advantage over students in an earlier session because they may get access to the exam before their session.

Length is the length in minutes of the exam.

Open Book indicates whether students have access to paper-based materials such as text books or their notes.

Unseen indicates whether the students have seen the exam question(s) in advance of their exam session.

Individual indicates whether students are permitted to ask for assistance from peers or tutors, or to work in groups.

Networked indicates whether the machines used by the students are connected to each other and the web. In all cases, even though the machines may be connected, invigilators ensure that students are not communicating with each other electronically.

All the exams studied here make use of invigilators – either the existing tutors, or professional invigilators – in order to ensure that the predetermined exam conditions are maintained.

3. FORMAT AND RATIONALE

An overview of the format of each exam is now provided, along with the underlying rationale for using the exam. These should be read in conjunction with Figure 1, which gives details of the results obtained in the examinations, and of the highlights of each particular method as identified by that method's 'owner'. Note that a strict analysis of marking/evaluation techniques has not been made at this stage.

The online, open-book **Aberystwyth** exam format is held every 4 weeks or so. Full access to the Internet enables students to consult their own notes, the notes of their lecturer and the Java API. Students are not permitted to seek help in any form: messenger software, posting on forums, and open folders for other students to use are banned.

Prior to this examination, online multiple choice tests were used with similar frequency, principally to enable fast and plentiful feedback. These proved popular, but were deemed to favour students with particular learning styles and to not

Institution	Aberystwyth	UCE	Durham	Glasgow	Leeds	Sunderland	Swansea
Students	120	160	80	170 – 480	5 – 200	100	5
Frequency	4/yr	2/smstr	2/yr	2/yr	1/crse	3/yr	1 resit only
Sessions	1	2, same day	5	Up to 24	1	4	1
Versions	1	6	1 basic scenario & 5 derived tests	1	1	6 scen. + 3 tests per scen per year	1
Length	120	90	110	110	~180	75	120
Open Book	Y	Y	Y	Limited	Y	Y	Y
Unseen	Y	Y	Y	N	Y	Y	Y
Individual	Y	Y	Tutor avail., costs marks	Y	Y	Y	Y
Networked	Y	Y	Y, no msg-ing, e-mail	N	Limited	Y	Y
Results	Radical improvement over previous MCQ test – smaller std dev and tail, and end of course MCQ much better answered.	First run this year. Low average mark with a third passing, but largely caused by many students being absent.	About 10% of the students used the help on offer, and were then able to proceed. This significantly reduced visible stress levels (tears) in some students	Some rote learning takes place (see Sec. 3), but range of practical skills clearly assessed. Better marks than in written exam.	Results match the perform-ance of students on formative assess-ment.	60/40 % on coursework/exam, significant minority show big discrepancy between the two.	As expected. Half the students realise the severity of the exam and revise seriously, the rest fail.
Highlights	Submission & marking on-line, maximises feedback. The exam tests comprehension not memory.	Provision of early feedback to both staff and students on progress. 2nd test based on coursework. Security of exam paramount.	Orthogonality in test design, allowing students stuck at one point to succeed still elsewhere. Provision of assistance, costing marks, reduces stress levels.	Particular skills assessed – coding, testing, debugging – design tested elsewhere. One version of exam only reqd. across multiple sittings	Matching intended learning outcomes with assess-ment.	Students appreciate the *validity* of the exam. Rapid feedback. Tests design, coding and testing skills.	The practical nature of the assessment has strong effect on student attitudes and hence performance.

Figure 1: Categorising the 7 *Commons* laboratory examinations

necessarily be a valid assessment of programming, particular coding, skills.

The **University of Central England (UCE)** model is used to assess 50% of the course, using two equally weighted in-class tests. Each run is split into 2 sessions held on the same morning and marked in the lab. Hence the principal criteria for the exam are that it tests the appropriate skills and can be marked swiftly. Multiple versions of the test are developed to ensure that students in the same session cannot cheat by looking at another's screen, and that one student cannot assist a student in a later session.

The aims are (a) to demonstrate that students can produce simple software, not just answer MCQs and (b) to act against plagiarism and contract cheating.

The **Durham** model uses the four regular weekly lab slots. To handle the problem of some lab groups being unfairly advantaged over others, the model makes use of a single central scenario, with individual tests crafted for each group addressing different aspects of the scenario, but all of similar complexity. Uniquely in this study, students can ask for help from tutors, but any assistance given is recorded and taken into account during marking.

The principal motives behind the design are to reduce student stress levels created by a practical exam, and to ensure that if students are stuck at one point in an exam, they are able to show their knowledge and skill in another part: the components of the exam are designed to be at least partially orthogonal to one another.

The **Glasgow** model was designed at a time of very high student numbers (ah, happy days!) and like Durham uses the existing weekly lab slots to avoid timetabling problems. Uniquely, the students see the question 10 days prior to the first session, and use this time to develop a solution, with or without external assistance, although not from tutors. In such an open scenario, only one version of the exam is required. Students may bring nothing into the exam, and with the question in front of them only, they must develop their solution on the machine. They can access a language reference manual and the other help-features of the programming environment, but are otherwise entirely isolated from the network.

It is accepted that this exam is in no way a reliable test of problem solving skills. However, problems are used with solutions that are believed to be too large to memorise. Instead, students are expected to be able to remember the outline of their solution and then the exam measures their ability to code, test and debug this solution. A separate written exam tests problem solving skills – although whether that is a valid exam is another matter.

The **Leeds** model requires all students to take the exam simultaneously, although possibly in different locations, and hence need not worry about fairness issues deriving from separate sessions. Students are given "about 3 hours", although this is not enforced rigorously: the rubric says "We will ask you to stop when you are not making any progress". Provision of tutor assistance was tried once, but some students' hands were continually in the air, so this was dropped.

The rationale for this model is very straightforward – "It seems senseless to assess programming in any other way than asking students to program – we use this method so that they can't cheat."

Sunderland has four tutorial groups in its course. For each group, an examination scenario and three tests are developed to be used across three examination points during the year. Additional scenarios are also developed for practice and for 'referred' students. The seemingly high load of developing this setup – 6 scenarios and 18 tests – is amortised over the years, and in fact a core of 8 scenarios and tests are reused in various forms.

This method is used because it is seen as the most reliable way to prove that a student has reached a certain level of programming ability.

Finally, **Swansea's** model is only used for the resit exam, in order to be able to allow for bad performance in either coursework or final examinations. As such students pick 2 from two practical questions and one theoretical question. The numbers are low, and so only one session is required, simplifying the arrangements appropriately.

The most significant discovery with this format is the effect it has on the resitting students. Many such candidates tend to leave revision too late – but with a programming exam in store for them, many appear to realise that they must extensively practice their programming skills on a machine, on their own, and hence a higher proportion than usual pass the exam.

4. COMPARING MODELS

All of the models examined here are in use and are therefore sufficiently reliable and valid (within their stated contexts) to have satisfied their institution's quality assurance processes. Nonetheless, they display a number of differences and these will be explored now.

4.1 Three major formats

Single session, unseen. Aberystwyth, Leeds and Swansea are all able to bring their students together for a single examination session, and are therefore able to use a format similar to a traditional written exam, only based in the lab. This is clearly the simplest format.

Multiple sessions, unseen. UCE, Durham and Sunderland maintain reliability and fairness across multiple sessions by requiring separate versions of the exam for each session. This presents a potentially significant overhead in setting the exam.

Multiple sessions, seen. Glasgow's unusual format assesses only a subset of its course's stated learning outcomes – those associated with the use of a programming environment to code, test and debug a program. The benefit is that only one paper is needed across multiple sessions.

4.2 Reliability

A reliable examination framework ensures that each student is examined in the same way.

One aspect of a reliable exam is that students are unable to cheat. Most of the methods appear to model the exam conditions found in traditional open or closed-book written exams as closely as possible. Hence invigilators are used in all cases to uphold the relevant regulations. In particular it is the invigilators who ensure in networked labs that the students are not communicating – although Leeds and Glasgow take this a step further by disabling virtual communication methods.

UCE uniquely takes this a step further by acknowledging the potential for students' eyes to stray to a neighbour's screen. Unlike in a traditional written exam, this problem of overlooking one's neighbour is countered by using multiple versions of the exam within a single session and ensuring that adjacent students are taking different versions of the exam.

A second aspect of reliability is that each student should face questions of the same complexity and examining the same material. This is obviously an issue when multiple versions of an exam must be created to be used in multiple sessions. A detailed analysis of different versions of questions is beyond the scope of this paper – but some general approaches emerge: Durham develops an overarching scenario common to all versions of a particular exam, from which individual, distinct and equally-challenging programming tasks are derived; Sunderland develops a different scenario for each lab group, and then derives a sequence of programming tasks from each scenario to be used by the group throughout the year – again these need to be of comparable complexity; and UCE develop entirely independent versions – the reliability comes from the fact that the same basic abstractions underlie the solution to each problem.

4.3 Validity

Real world programmers have access to a wealth of resources – from their own completed programs, to notes, text books, discussion boards, FAQ sites, other programmers and so on. Whilst all the models here permit at least some access to reference material, increasing the validity of the assessment, only Durham's model permits access to tutor support – i.e. other programmers. This comes at a cost since tutor assistance is taken account of in assigning marks. This may of course effectively correspond to the industry situation – if a programmer needs to ask for help regularly, they are unlikely to be promoted so quickly.

Another consideration is whether the problems used in the exam assessments match the problems used in the students' skill development phase – the formatively-assessed coursework exercises. A full analysis is again beyond the scope of the paper, although UCE at least bases one exam on aspects of the students' coursework.

4.4 Scalability

Some of the methods presented here do not scale well in the face of increasing class sizes. Where multiple versions of exams are required, there comes a point when the number of versions required will become too great. For those institutions currently using a single session format, as long as enough machines can be commandeered at one time (or in back-to-back sessions) for all candidates, there is no problem. Once multiple sessions are required, however, one of the more complex methods will be required.

5. SUMMARY

We recognise the appropriateness of using laboratory exams for at least part of our assessment, in supporting increased validity and reliability. Furthermore, our students recognise this too – for example, they are quite vociferous about the plagiarists in their midst.

There is clearly potential for further analysis in order to fully understand these assessments, e.g. comparisons of the methods used to evaluate students' exam performance, and of their performance on the lab exam(s) against other assessments in the same course.

The methods analysed here, however, have highlighted a number of beneficial aspects: improved student attitudes and hence study habits; provision of reliable feedback to both staff and students early and often; assessment of the full range of programming skills; testing of comprehension not memory; separation of the skill development phase from the skill testing phase; and the potential to reduce exam stress levels.

6. REFERENCES

[1] Califf, M.E. and Goodwin, M. Testing Skills and Knowledge: Introducing a Laboratory Exam in CS1, SIGCSE Bulletin, 34:1 (2002), 217-221.

[2] Chamillard, A.T. and Jointer, J.K. Using Lab Practica to Evaluate Programming Ability. SIGCSE Bulletin, 33:1 (2001), 159-163.