Enhancing the Learning Experience on Programming-focused Courses via Electronic Assessment Tools (Extended Abstract)

Hans-Wolfgang Loidl Phil Barker Sanusi S. Usman

Department of Computer Science

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh EH144AS, Scotland

H.W.Loidl@hw.ac.uk Phil.Barker@hw.ac.uk ssu30@hw.ac.uk

Developing solid programming skills is the back-bone of student transition in any computer science programme, when it comes to technical skills that need to be acquired by the students. On-line tests and quizzes to test the students progress, which can be done in their own time and are tailored to the requirements of the specific course, are a highly effective mode of providing formative feedback.

We report on our efforts of using electronic exam software to provide formative feedback to students, with a focus on immediate automated feedback. To assess the results we draw on survey data we gather from students on three programming courses, in first, second and fourth year of studies. We contrast a traditional approach, with feedback on coursework submissions, to our new approach of using e-assessment both for summative and formative assessment. In the context of a fourth year course on parallel programming languages we also assess the influence on the style of programming language on the student's learning experience, contrasting main-stream, low-level message passing languages with high-level parallel functional languages. This phase of learning represents a step change of grasping new concepts as supported by modern functional languages, to the incremental learning phase of acquiring new skills within a known language.

This work is part of a larger effort at Heriot-Watt University, to integrate on-line exams, both in the form as formative quizzes and summative class tests, into software development courses of the early years in our under-graduate degree programmes and in new degree programmes that feature a substantial percentage of distance-learning-style material.

This is work-in-progress and we are currently collecting and analysing the survey feedback data from the e-exam. This data will be available in time for a presentation at TFPIE 2017, and for an in-depth analysis in the full version of the paper.

1 Introduction

Teaching courses on programming languages or more widely on programming-centric skills naturally lends itself to an approach of active learning, that is driven by carefully structured programming exercises and offers a learning-by-doing environment. A key component in such an approach is the provision of timely and detailed feedback to the students on their (partial) solutions. Established methods, such as in-lab, verbal feedback by tutors and lab-helpers have their limitations in the depth of feedback that can be given and in the scalability to large class sizes. The nature of the language being taught, either object-oriented or functional, also plays a significant role in how the topic of feedback is approached. In this paper we are reviewing some of the current feedback techniques used in computer science courses (modules) at Heriot-Watt University, and contrast them with approaches to on-line assessment (either formative or summative) that we are starting to use.

Submitted to:
TFPIE 2017

This study is part of a wider initiative on enhancing the feedback given to students, in particular giving more immediate feedback at several points in the course than would be possible without an online framework. In previous surveys, students often expressed their desire for more quiz-style sections, and therefore we are confident that this effort will also improve the overall student experience.

Our rationale for introducing this style of feedback are the following predicted benefits:

- improving the student transition along the development of technical programming skills;
- more regular and immediate feedback on quizzes;
- increasing student engagement and providing checkpoints of learning progress for the students;
- easing the load of staff to do manual marking;
- dealing with exceptionally large class sizes (as we observe them in the current first year for CS);
- and transferring this infrastructure to other campuses;

In this extended abstract we describe the context of these studies (the nature of the courses and profile of the student cohorts), the technology that we are employing for the e-assessment components, and some initial results from early usage of e-assessment in a first year programming course. More studies are ongoing, in particular the usage on a second year programming course and more in-depth results from a parallel programming course that covers both imperative as well as functional programming languages.

2 E-exams

2.1 A Pedagogic Case for the Use of E-exams

The requirements that most strongly influenced our choice of an online assessment system came from its use to reduce the burden of marking summative assessments. These assessments affect a student's progression in their degree programme. At a high level the requirements can be summarised as the need for the system to be *secure, reliable and scalable.* These lead us to consider hosted commercial offerings, and ultimately to trial the BTL Surpass system. Acceptance criteria were written for the trial, listing criteria around security, auditing and checking of questions, pedagogy and operational considerations. A full list of these criteria is available [1]. Once we were convinced that BTL Surpass was satisfactory for our needs we examined the wider pedagogic case for using it in formative and low stakes assessment.

Firstly, there is a subject specific consideration. Much of the material we teach in computer science, especially on the programming courses, spanning object-oriented and functional paradigms, is not suited to paper-based assessment. In our computer science courses we are trying to be practical and focused on developing real-world experiences to make the transition from university into industry as easy as possible. Therefore, we frequently use computer lab and tutorial sessions based around programming. It is important that we look to creating equally authentic means of assessment — a paper based exam is unnatural and counter-productive. Coursework is only a partial answer to this as it is difficult assess whether the student knows what to do and how to do it, as opposed to their ability to find a recipe to follow using services such stack-exchange, and furthermore marking coursework is a heavy burden on teaching staff with typically feedback to students being provided only slowly.

In common with many in Higher Education, we acknowledge a problem with a lack of engagement of students on the courses we offer. Exams and class tests may be the peak time of student engagement,

[1] `https://blogs.pjjk.net/phil/requirements-online-exam-system/`

when students discuss their answers to various questions asked. Online assessment with automatic marking offers the opportunity of giving students immediate feedback at this point of peak engagement rather than weeks later when all students care about is their numerical result. It is possible in Surpass to tag questions by learning objective. This means that we can use continual assessment throughout the course to show students what abilities we expect them to be developing, and allow them and the teaching staff to appraise their progress in the course.

As an international multi-campus university it is important to us that Surpass is scalable beyond a single campus. It has a relatively sophisticated workflow for creating and checking questions and for moderating marks which can be shared among several individuals. We hope this will support teamwork across campuses in setting questions that address course learning objectives, leading to a better shared understanding, in concrete terms, of what is expected from students.

In summary, we hope that continual assessment with BTL Surpass will counter-balance the coursework with secure objective class tests and allow us and the students to check learning progress.

2.2 Exam Authoring in BTL Surpass

The BTL Surpass software for e-exams provides a platform for creating electronic tests and quizzes. These are initially formed as *questions items,* and structured according to the targeted learning outcomes. All question and informative non-question pages are integrated in an item authoring section. The section has a collection of templates for different question types, in addition to other functionalities to provide a well-integrated environment. Templates such as the multiple choices, multiple response, selection from the list, numerical entry etc. are useful for the basic e-exam questions. Additionally, fill-in-the-blank and short essays provide text entry templates, to be marked based on the exact text match. Using appropriate templates, an assessment can be effectively designed for entirely computer marking, a combination of computer and human marking, or entirely human marking.

Depending on the requirements, tests can be created for *learning, formative and summative assessments.* In learning assessments, students can check their progress through instant access to sample solutions, or hints guiding students while answering questions. In the case of formative assessments, the solutions and any other feedback are only accessible for the student upon completion of the test in order to review performance. Whereas the learning and formative assessments are delivered in a web browser, summative assessment uses a *secure client* to deliver tests in a lock-down environment. Results are only released after tests are marked and moderated, with reports generated to show performance statistics.

A notable, important benefit of the software is the capability to assign learning outcomes to questions, such that when feedback is given to students, they could see how well they performed across different learning tasks. This will be helpful in directing attention and additional efforts to areas so identified. Additionally, being an online tool, it could be deployed at several campuses without restriction on concrete setup. Furthermore, the simple design of the interface, and good navigation makes the software easy to operate, which was confirmed in our earlier test runs with students.

3 Context

We will contrast the formative and summative assessment given in three programming-centric courses.

"Software Development 1" which is the first programming language course, using Java, in Year 1, Semester 1 of our under-graduate degrees. Some of our students come to this course with no programming background at all and therefore this course has to teach programming from ground up. We use

Java as an object-oriented language, but take a “basics first” approach, introducing basic programming language constructs, before covering object-oriented constructs in the second half of the course.

“Hardware-Software Interface” is a Year 2, Semester 2 course on low-level and systems programming. It introduces C and ARM Assembler languages, and applies them to systems programming tasks on the Raspberry Pi 2. At this point students have solid (object-oriented) programming background, but they often lack a firm grasp of performance implications of their code. Giving this view, and controlling simple devices wired up to a Raspberry Pi were the main points of motivation for introducing this course.

“Parallel and Distributed Programming” is a Year 4, Semester 2 optional course, for final year students. This course covers a range of different programming technologies, from C+MPI and OpenMP, to parallel functional languages such as Glasgow Parallel Haskell (GpH) and Single Assignment C (SAC). In its spread of programming languages, it is a good test case for picking up advanced programming skills in a very short time span. Since this course is optional it typically attracts our best programmers, which results in a different student profile compared to the two mandatory courses above.

In the bigger picture of our programming language courses, we put a focus on object-oriented programming, and in particular Java, in the first year. The scope is broadened to other paradigms, in particular functional programming and logic programming, as well as scripting and low level programming, in the second year. This phase in particular is challenging since it represents a step change of grasping new concepts (such as higher order functions), compared to gradually adding new, but fairly narrow, mechanisms in a known language (Java). This step change is reflected in the different kinds of questions by the students in these courses, with more focus on concepts as opposed to low-level details. Whereas in the first year, naturally, a lot of questions are on basic understanding of compute structures, and are more amenable for automated feedback, the focus shifts to the main concepts in functional and logic languages, which is more difficult to cover with automatic feedback. The difficulty in low-level languages for most students is to get a firm grasp of the concrete operational aspects of program execution. This is fairly hard to cover through the usual form of (static) textual feedback, but can be effectively addressed using dynamic approaches, such as animated graphics. We already use the latter in the form of the widely used Python Tutor [2] in lab settings, but not embedded in the feedback to students. In the context of functional languages the main benefit of quizzes and short programming tasks is to reinforce the lecture-style presentation, that focuses on the concepts, and demonstrate the practical relevance and benefits in a tightly controlled environment. Based on this approach, we typically get very positive feedback by students on Haskell and SAC as “more powerful” languages with a concrete benefit in programmer productivity.

Another motivation for this study are our current efforts to support more distance-learning material in our courses, which is part of the development of new degree programmes with a big work-based learning component. This setup requires a highly structured nature of the courses, with intermediate check-points for the students to check their learning progress. E-exams and quizzes are one suitable instrument for this purpose, provided that they deliver sufficient feedback on the student’s performance.

4 Preliminary Evaluation

4.1 Conventional delivery in an advanced course on parallel programming

The course “Distributed and Parallel Technologies” covers a range of programming technologies for programming parallel machines, starting from low-level approaches based on message passing (C+MPI) to high-level approaches using functional languages (Glasgow parallel Haskell and Single Assignment C) or

[2] Python Tutor

functionally motivated approaches such as Google's MapReduce framework. By starting with low-level approaches, the students appreciate the ease-of-programming provided by the high-level approaches and this is reflected in their comments from the end-of-term course survey. As practical coursework, the students implement a very simple mathematical computation, which is amenable for data parallelism, first using one of the low-level approaches and then using one of the high-level approaches. This experience provides a powerful message on the relative ease of parallel programming in a functional language that directly translates to a reduced effort in implementation. Both pieces of coursework are summative assessments and we give detailed feedback on the submission, in particular the quality of the parallelisation and of the code in general. Traditionally, the students' performance in the exam is poorer compared to the practical coursework. In part, this is due to the unusual nature of doing coding exercises in an exam setup. But it also reveals gaps in understanding of underlying concepts of the two functional languages used in this course. Therefore, we conjecture that a move to e-exams and the use of quizzes throughout the course would deepen the understanding of these technologies (although, the coverage of six different parallel programming technologies in this course is a strain on the students' ability to absorb new material within one term).

4.2 Using e-exams in software development courses

Evaluation of the use of BTL Surpass for *summative exams* in "Software Development 1" was performed using a questionnaire given to the students, an interview with a small group of students and an interview with the staff involved. The staff interview largely determined that the staff involved felt that the online exam system had performed well but they needed more support in its use and in scheduling and setting up labs for use in exams. The more interesting results concerning the potential of online assessment for learning computer science came from the students.

The student questionnaire was completed by 60 of the 150 students from the Edinburgh campus. These were students who attended a lecture in the subsequent "Software Development 2" course and chose to return the questionnaire, and thus results are available from reasonably large but self-selected sub group of those students who took the exam, probably representing the more conscientious students in the class. Responses to five-point likert scale questions indicated a positive student experience, see Table 1.

Only 3 of the 60 students felt there should be no more online exams. Additionally there was significant strong agreement to the statement "I would like to be able to self-assess for practice while studying"

Table 1: Student feedback on a summative e-exam in "Software Development 1" (five-point likert scale: 1 means strongly disagree, 5 means strongly agree)

	1	2	3	4	5
I was able to demonstrate whether I understood the material being tested.	1	2	12	25	19
I felt adequately prepared to use the software.	1	4	8	24	23
I found the software used to deliver the exam easy to use.	0	2	7	14	37
I was able to enter the answers that I wanted to.	1	1	6	12	40
The feedback from the software about whether my answers had been submitted was adequate.	0	5	12	20	23
The computer lab was a suitable setting for this exam	1	4	10	13	32

(42 strongly agree, 7 agree) and "I would like assessment to be in smaller chunks during the course (i.e. continuous assessment rather than one exam at the end)" (27 strongly agree and 8 agree), albeit with a desire expressed for "immediate automatic feedback on my performance" (44 strongly agree, 10 agree).

Written comments from the questionnaires and comments from the interview (again, a self selected group of 6 conscientious students) shed some light on the reasons for these responses, which are largely in accordance with the rationale for introducing online assessment. The students liked the general ease of use of the system, found it easier and less stressful to work with the online system compared to paper-based exams, specifically mentioning worries about the legibility and speed of their hand-writing, the ability to flag questions to which they wanted to return and the display of a timer and progress monitor. They also pointed to the naturalness of using an online system to assess computer science "We are computer scientists, we like computers".

While the overall feedback from students was overwhelmingly positive, there were some significant exceptions, which largely centred on operational issues regarding network delays in logging on to the system. There was also a perception from the students that results from an online exam should be available immediately whereas in fact they need to be approved by the same exam boards as any paper-based exam.

For "Hardware-Software Interface", this year we are complementing the traditional summative assessments, in form of programming exercises as practical coursework, with a formative e-exam, as a mock-exam, to prepare the students for the unusual setup of coding exercises in an exam. In the long term we are considering to entirely move to this form of assessment, which is more suitable for programming centric courses, as underlined by our experience with the early software development courses.

5 Conclusions

Our experience so far with the e-exam software has been positive, and so was the feedback that we received from the students on their experience. We are now extending the usage to formative assessment as well, which was well received, but needs more studies to come up with concrete conclusions. Concrete lessons learnt so far are:

- Specific coding skills can be effectively tested in an e-exam setting, and better so than in a pen-and-paper exam, leading to a better student experience;
- Immediate feedback can be given on quizzes and e-exams, using the e-exam software that we are using (BTL Surpass); initial comment on the effectiveness of this feedback has been positive, but needs more evaluation;
- Practical coding exercises proved to be crucial in making the argument of ease-of-development in parallel functional languages, as opposed to more main-stream approaches
- In introducing new concepts from functional programming languages, in particular higher-order functions, a balance of lecture-style presentations with practical coding exercise is crucial for capturing the essence of these concepts

In the long term we hope that continual assessment with BTL Surpass will counter-balance the coursework with secure objective class tests and allow us and the students to check learning progress.