2nd Annual Conference on the Aiming for Excellence in STEM Learning and Teaching

Improving the student experience on computer programming courses with a holistic approach to learning design and technology

Paul Neve paul@kingston.ac.uk **David Livingstone** d.livingstone@kingston.ac.uk **Gordon Hunter** g.hunter@kingston.ac.uk **Graham Alsop** graham@kingston.ac.uk

Learning Technology Research Group
Kingston University
Penrhyn Road
Kingston Upon Thames
http://ltrg.kingston.ac.uk

Abstract

The NoobLab online learning environment for computer programming, first presented at the HEA STEM Conference 2012, has since been deployed on a first year undergraduate module. This paper discusses how the introduction of the environment had an impact on the student experience. A holistic approach was taken to the development of learning content and the learning environment. Conventional lectures were minimised; instead, students were encouraged to spend the majority of their time engaging in practical exploration. The use of the environment for summative assessments allowed for a "zero marking time" paradigm in many cases. Students reacted positively to this, and the detachment that arises when there is a lengthy delay between submission of work and feedback was greatly reduced. An improvement in student outcomes was evident, and a highly significant correlation was found between students' final marks and time spent within the environment. These positive results have led to the environment being deployed in several other programming modules. The environment is now expected to become the primary delivery tool for teaching introductory programming in the School of Computing and Information Systems at Kingston University.

Keywords
Computer programming, e-assessment, course delivery, pedagogy, distance learning

1. Introduction

An international study of first year university computer science students (McCracken et al. 2001) concluded that "students do not know how to program at the conclusion of their introductory courses", based on the fact that only 21% of students across all the institutions involved could pass a standard test. Students use term such as "difficult" and "boring" to describe programming (Jenkins 2002), and do not acquire the knowledge and skills from introductory teaching that the learning outcomes aspire to.

A common approach to course delivery is to combine lectures with practical workshops, which makes pedagogic sense, and in one study was found to be almost as effective as certain constructivist approaches to delivery (Poindexter 2003). This also makes sense logistically, as large cohorts can be lectured to, and then broken into smaller sized groups for the practicals.

One problem with this approach is the passivity of students during lectures. One study showed that, neurologically speaking, students react the same way to a lecture as they do to television (Manzur 2012, Poh et al. 2010), with brain wave patterns that show very little activity. In contrast, during other activities e.g. "homework", the student's brain was highly active. One might conclude that the active brain patterns involved deep learning modes; the "flatline" of a lecture might be symptomatic of surface learning. Indeed, their attendance at a lecture might lull a student into a false sense of security. After a lecture. if then confronted with an inability to understand a subsequent task they conclude that it is too "difficult" or that they are not capable enough.

The NoobLab online learning environment was created to mitigate such issues (Neve & Livingstone 2012). On one module, NoobLab's introduction led to a holistic approach to learning design and technology where each aspect informed and influenced the other. This paper discusses how this module was developed and the learning environment integrated, the results that occurred when delivered, and how these results are influencing course design and delivery across the programming syllabus.

2. The Practical Programming Module

Practical Programming is a second-semester, first year module delivered separately to Computer Science (CS) and Information Systems (IS) students. It is designed to follow an introductory module, *Programming Essentials*, seeks to go beyond the basics and help students become creative programmers. Its aims are: To develop students' enthusiasm for practical programming; to enhance students' experience with programming environments; and, to develop students' confidence in their ability to write programs.

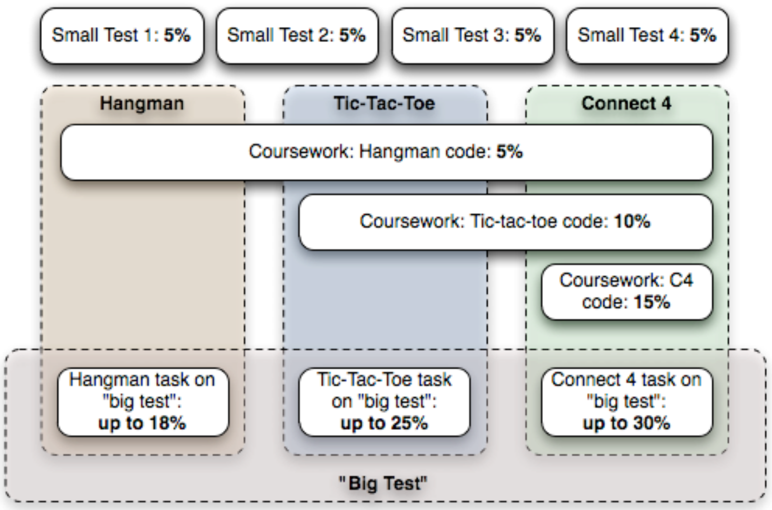
These aims were not being met on the IS instance of the module, where almost half failed to achieve a passing grade in 2011. A new approach was welcomed, and the module was selected to serve as a pilot for a NoobLab-centred redesign.

2.1 Course, assessment and environment design

The IS cohort had differing programming ability levels therefore the possibilities for flexible, self-paced learning offered by the environment were compelling. The hope was that the technology would facilitate a constructivist approach to teaching, with the emphasis on practical exploration and experimentation. It was seen as important that the redesign fostered certain key ideas from the module's aims: *enthusiasm*, *experience* and *confidence*.

The decision was made to begin with the artefacts that the students would be expected to produce and work backward. Three overarching games were selected for this purpose – Hangman, Tic-Tac-Toe and Connect 4. They were chosen to present an increasingly challenging task and be something the students could get some satisfaction from creating. Where possible, activities and assessment components were related to these games so as to retain a constructivist learning perspective (Gance 2002). It was important to avoid abstract problems with which students struggle to engage (e.g. "calculate compound interest").

The vision was that a student would interact with a variety of different media and activities in a single session. These included static text and diagrams, multiple choice and quick answer questions, exemplar code, HTML excerpts, or practical challenges where the student was expected to compose code to be evaluated against test criteria (a "NoobLab test"). Where there was no pre-existing support within the environment, new functionality was added as and when required. As a result, the capabilities of the environment quickly grew alongside the course content. The evolution of the delivery environment and course design informed each other. By the time of the first in-class session, the content and environment had combined to deliver an integrated experience, where the student never needed to leave the environment. Content was exclusively delivered by the environment.



Small Test 1: 5%
Small Test 2: 5%
Small Test 3: 5%
Small Test 4: 5%
Hangman
Tic-Tac-Toe
Connect 4
Coursework: Hangman code: 5%
Coursework: Tic-tac-toe code: 10%
Coursework: C4 code: 15%
Hangman task on "big test": up to 18%
Tic-Tac-Toe task on "big test": up to 25%
Connect 4 task on "big test": up to 30%
"Big Test"

Figure 1: Breakdown of assessment components on Practical Programming

Assessment was divided into components, see Figure 1. Regular "small tests" served a dual purpose – to reward regular effort with marks towards their total, and to avoid disengagement until shortly before final assessment. The main components of assessment were derived from the games, with some marks available for the game code submissions, but the majority were awarded for a "Big Test" during which students were asked to make live changes to alter the behaviour of their code.

2.2 Course delivery

Two in-class sessions took take place on the same day, both of two hours' duration. Lecture-style teaching was minimised; instead, the tutor introduced the opening session each week with a review of the previous week's activities. The material for the current week was then introduced, after which students would embark upon a self-paced exploration of the material. The tutor would then step back and act as a "guide on the side", offering commentary or opinions, giving assistance and feedback, or addressing the group as a whole if something interesting came up. This allowed students to explore and learn at their own pace, unencumbered by class schedules or tutor agendas, while maintaining a support structure. The first six weeks focussed on guided material, after which students were then expected to concentrate on developing their games programs.

Most of the activities designed to support the games activities were delivered as NoobLab tests. Thus, students could get immediate feedback from the environment. "Small Tests" were also expressed this way, meaning that for some summative assessment tasks, students received immediate feedback.

3. Results

Based on the 46 students who completed over half of the summative tasks, 78% passed. It would be unfair to conclude that the change of approach and introduction of the environment was responsible for this improvement: there was also a change from Ruby to

Javascript, and a new lead tutor to consider. However, there was a significant correlation (R = 0.749, p < 0.001) between time spent in the environment and final mark.

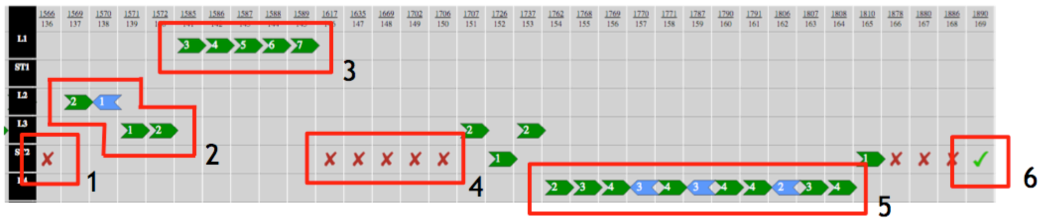


Figure 2: The "Rosetta Stone" Pattern

The potential for employing usage patterns to emulate the impromptu feedback that might arise from a tutor was established previously (Neve & Livingstone 2012). Other patterns influenced pedagogy and course design. One of these became informally referred to as the "Rosetta Stone" (RS) pattern, where the same collection of learning materials and/or activities prompted the same "Eureka moment" in several students. When the RS pattern was unexpected, it proved illuminating to re-examine the material involved. Figure 2 shows a student attempting a Small Test and their associated use of learning materials. Their first solution to the test (box 1) did not pass. The student then reviewed material from several weeks previously (boxes 2 and 3) before failing again (box 4). Only after viewing material from a future week (box 5) could they pass (box 6). Upon examining the upcoming content, it was evident the "recap" of the previous week contained therein was more comprehensible than the original material.

Feedback was positive. Students were given a series of positive statements about aspects of the modules and the learning environment. They were asked to assign a value between 1 (disagreement) and 6 (agreement). For each assertion a "satisfaction index" was calculated, derived from the overall total for all students' values expressed as a percentage of the maximum possible score based on the number of responses. The environment itself scored highly, with satisfaction indexes at 90% or more for related assertions. The environment also appeared positively in qualitative feedback. Students also appreciated the new approach and structure. The concept of Big and Small Tests scored highly – 94% and 96% respectively, and the flexible assessment approach via the games scored 89%.

The tutors' experience was also positive. Two staff and a helper were allocated to the sessions, but often one of them was able to leave before the end – a considerable amount of the support required by students was supplied directly by the environment. The often onerous task of marking students' work was reduced – by using NoobLab tests to frame many of the summative activities, a student who received what came to be known as "the green box of success" could be immediately assigned full marks. Many near misses could also be immediately assigned based on a quick look at the NoobLab stats, with only submissions at the weaker end of the spectrum requiring a human to assess whether some credit was justified. In most cases, students received their marks for a Small Test the same day.

4. Conclusion and future work

Programming is a discipline that must take place at a computer, yet there is often a reluctance to embrace technology in its teaching. When learning programming, face-to-face time is arguably less important than face-to-*screen* time. Nevertheless, we see teaching approaches using long lectures and students are then expected to translate these into the capability to compose and understand code. One student gave an analogy: a lecture on programming is like being shown a collection of ingredients, then being expected to cook a complex meal with them (Proulx 2000). Yet the opposite is no improvement – one cannot simply fling students in at the deep end, place them in front of an editor and compiler and expect them to create original programmatic compositions.

The NoobLab learning environment was successfully deployed on a module, and met with good results with respect to student outcomes and overall feedback. Yet it is important to note that this success was not solely due to the learning environment itself. It is common to hear the phrase "the tail wagging the dog" among learning technologists, where "tail" and "dog" each translates to either pedagogy or technology. Which is the tail and which is the dog varies depending on the learning technologist! However, the success was in no small part due to the holistic approach that was taken to both course design and learning environment enhancement. Ultimately, there was no tail or dog – both the technology and the pedagogy were aspects of the same thing, and this is crucial in a face-to-screen-based discipline such as programming.

As a result of the success of this pilot, the environment is being used in other modules and is now being used to deliver material ranging from pseudocode-based foundational exercises through to object oriented Java programming. In 2013/14, NoobLab will be the primary delivery mechanism and laboratory environment for programming within the critical first year of the various computing degree programmes.

5. References

Gance, S., 2002. Are Constructivism and Computer-Based Learning Environments Incompatible? *Journal of the Association for History and Computing*.

Jenkins, T., 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. pp. 53–58.

Manzur, E., 2012. The scientific approach to teaching: Research as a basis for course design. In ALT-C 2012: A Confrontation With Reality. Manchester University, 11-13 September 2013.

McCracken, M. et al., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education*. ITiCSE-WGR '01. New York, NY, USA: ACM, pp. 125–180.

Neve, P. & Livingstone, D., 2012. Developing Virtual Programming Laboratories to Inform the Pedagogy of Programming. In 1st Annual Conference on the Aiming for Excellence in STEM Learning and Teaching. Imperial College, London.

Poh, M.-Z., Swenson, N.C. & Picard, R.W., 2010. A wearable sensor for unobtrusive, long-term assessment of electrodermal activity. *IEEE transactions on bio-medical engineering*, 57(5), pp.1243–1252.

Poindexter, S., 2003. Assessing active alternatives for teaching programming. *Journal of Information Technology Education*, 2, pp.257–265.

Proulx, V.K., 2000. Programming patterns and design patterns in the introductory computer science course. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*. SIGCSE '00. New York, NY, USA: ACM, pp. 80–84.