Developing virtual programming laboratories to inform the pedagogy of programming

Mr Paul Neve & Mr David Livingstone  
Kingston University  
Penrhyn Road, Kingston upon Thames, KT1 2EE  
paul@kingston.ac.uk  d.livingstone@kingston.ac.uk

Abstract
Traditional lectures are considered inadequate as a sole means of teaching computer programming, and are most commonly combined with practical workshops designed to encourage students to learn by practical experimentation. Delivering these can be difficult when faced with large cohort sizes and flexible modes of study, with deleterious effects on the crucial “learning loop” that exists between student and tutor in an ideal workshop session. NoobLab is an online environment for the delivery of computer programming workshops. The environment provides a seamless, integrated analogue of the framing lectures, the practical workshop environment, as well as delivering the feedback component of the learning loop. The software is “aware” of a workshop’s learning objectives, providing feedback both on-demand and based on “observation” of the student’s activities. Additionally, by controlling the environment where students engage in practical tasks, students’ progress and the patterns they generate as they solve a problem can be monitored. These patterns can be correlated with other metrics and provide empirical data for investigating pedagogic issues around the teaching and learning of programming.

Keywords
Programming, e-assessment, course delivery, pedagogy, distance learning

1. Introduction

Pedagogic approaches to computer programming that reduce traditional lectures to a minimum have existed for decades (Daly et al. 1979). More recently, Schank (2001) argues that in this subject area even hard-copy textbooks eclipse lectures as a learning tool – the textbook offers a random-access capability a linear lecture does not. Underpinning such literature is an acceptance of the fact that programming is a highly practical subject and the best way to learn is by hands-on experimentation. This is evident in some of the constructivist approaches to teaching programming, where the onus is on the student to explore and the tutor’s role more a “guide on the side” (Wulf 2005). Nevertheless, it is still common for programming courses to adopt an approach where conventional lectures frame material that reinforced by practical “workshop” sessions. There is research that supports this hybrid of the traditional and modern “active” pedagogy (Poindexter 2003), and even Wulf concedes the need for short lectures to provide an overall structure and framework for the students’ experimentation. Without these, if simply invited to go off and explore a new subject on their own, one can easily imagine students feeling “lost” and not knowing where to start.
Consider an ideal workshop session. A student will be presented with a practical task designed to reinforce prior learning and encourage new learning via experimentation. At any given point, the student may request assistance of the tutor. There is also the possibility that the tutor might choose to volunteer information based on observation – for example, a tutor might see a student about to venture down a common “blind alley”, prompting the tutor to interrupt the student and point them in an alternative direction. Any feedback will given by the tutor is usually informed by knowledge of where an activity fits into a wider context (e.g. needs of a module or even entire course of study). A tutor might also tailor their feedback based on knowledge of a student’s ability, personality or any number of “soft” factors. Equally, a tutor might choose to phrase their feedback in a way that directs the student to explore a learning path they may not have discovered on their own. It also bears remembering that this ability to influence is not only one way – e.g. a student’s response might alter the tutor’s perception of the their ability level, which would in turn alter the tutor’s future responses.

This two way cycle of influence has similarities to double-loop learning, defined by Argyris (2002) as learning where “errors are corrected by changing the governing values and then the actions”. In our case, the word “error” is perhaps inappropriate so we instead use the term “learning loop” to describe the continuous, reinforcing dynamic between student and tutor during a programming workshop; the loop repeats an indeterminate number of times during a workshop and involves a partnership of influence between the two parties. This partnership constantly reinforces and informs the process of learning by practical, active experimentation.

Unfortunately, it is very difficult to successfully implement this learning loop for programming given realities in many British HE institutions. Many institutions struggle to make appropriate equipment available – programming is a specialist activity that is not always adequately served by central computing services designed to provide commodity IT (i.e. basic office-type applications) and little more. Large cohort sizes can make it difficult to keep the student to tutor ratio at an acceptable level. Academic helpers are often drafted to supplement tutor(s), but this impacts the quality of feedback and particularly the ability to provide contextually relevant feedback – an academic helper might be a talented programmer and able to express principles clearly, but may not know where a given task fits into the wider context of the students’ learning.

Finally, the increasing demand for distance learning (BBC 2010; Scottish Government 2011) presents different challenges. Such a delivery mode and the flexible schedules implied make it extremely difficult to provide real-time access to tutors, largely ruling out the interactive dynamic of the learning loop. Technical issues are compounded, and students may be expected to create a development environment on their own equipment – so even before they learn a single line of code, they are faced with a non-trivial configuration exercise.
2. Software Tools for Programming Workshop Delivery

There are tools that use virtualisation to deliver ICT-related practical workshops, including WLab (Neve et al. 2010), V-NetLab (Krishna et al. 2005) and Xen Worlds (Anderson et al. 2009). These tools could be used to deliver programming workshops: a tutor could create a virtual machine (VM) image that includes an operating system, an appropriate development environment for a given programming course, plus any sample source code or other material required. Students could then access this virtual machine via a standard web browser, circumventing some of the issues previously described. Unfortunately, virtualisation makes it difficult to go beyond the technical logistics of delivery and provide the feedback required for an automated implementation of the learning loop. The main issue is that it is difficult if not impossible for to determine the internal state of the VM from outside its confines – a virtual machine is essentially a sealed black box. In WLab, for example, students have to manually indicate completion of a task - the environment has no ability to determine the student’s activity and provide automatic feedback.

Projects such as RoboProf (C. Daly & Horgan 2004), CourseMarker (Higgins et al. 2005) and ASAP (Douce et al. 2005) have focused specifically on providing automated feedback for programming tasks. As such, they do to some degree they provide an analogue of the learning loop, but they fall short in a number of respects. Firstly, the expectation is that the student will compose their code using an external application, e.g. a text editor or IDE. Students must then supply their solution to the tool in suitable form – so, for example, they may need to submit the compiled bytecode for a Java program. The extreme position adopted by Prieto-Blázquez et al. (2009) when discussing their solution is that this is a good thing – “one of the things programming students must be able to do is to carry out compiler installations”. However, when teaching beginners who may struggle with very basic programming constructs, demanding that they install and configure a fully-fledged development environment can be a highly demoralising additional complexity at a crucially early point. Secondly, these tools only have “sight” of snapshots of a student’s work at the point they choose to submit. The student’s activity between these snapshots is invisible to the tool, thus it is impossible for the tool to provide an analogue of the unsolicited, improptu feedback in the learning loop.

3. Nooblab – Automating the Learning Loop

NoobLab represents an initial attempt to implement the learning loop in an online environment. The first iteration (Neve & Livingstone 2011) was created as a tool to bootstrap programming naifs to an appropriate level prior to them undertaking an MSc module in Java. The key objectives were

- To present the framing material for a practical programming workshop task alongside a facility for students to compose, edit and run the workshop code

- To provide dynamic feedback based on the student’s activities as they complete a workshop task, and provide an analogue of the learning loop.

In one task the student was asked to construct an IF statement to detect equality of two variables. If they made the common mistake of using a single rather than a double equals, the environment popped up additional material explaining their mistake – thus providing an analogue of an impromptu feedback in our learning loop. However, this first iteration was written explicitly for a single online course - the framing material was hand-crafted HTML and the various conditions which prompt feedback were hard coded in Javascript. The environment and content were tightly coupled and one useless without the other.
The second iteration of NoobLab decoupled environment and content, providing a way for new content to be authored without modification of the environment. HTML was used as a markup language to describe content, the assumption being that those likely to be teaching programming are likely to have some grasp of HTML already, which can be supplemented with any number of editors that exist. While there are a number of pre-existing markup languages that can be used to represent educational content, notably CNXML (Radaelli-Sanchez & Hendricks 2009) and eLML (Fisler et al. 2005), these were rejected primarily because of the learning curve entailed for potential content authors to learn a new markup language (editors do exist for both but are rudimentary at best).

Overlaid on top of the HTML standard were a number of new, environment-specific conventions that make use of standard HTML attributes and elements to specify interactive content, for example, small pieces of exemplar code that the student can be invited to examine, run and amend, or perhaps a quick “quiz” question. Within the NoobLab environment, any elements that fulfil these conventions are rendered as interactive content. Figure 2 gives an example of the HTML for an inline multiple choice quiz question.

The environment also provides the ability to set “exercises” within a workshop activity, where students receive feedback on their code. These exercises can be specified using simple rulesets based on the program’s final output, or using explicit Javascript code designed to evaluate the student’s efforts in more depth. Again, these are specified within HTML <div> elements using the class attribute to specify parameters. Impromptu feedback can be specified in the same way using a similar ruleset or again using more explicit code for additional control.
Try deleting the second quote from the line of code. What happens?

Nothing - the program works fine.

Not true! The program will produce no output, and instead you'll get an error complaining about an unterminated string literal.

The program doesn't work at all.

Correct! The error refers to an unterminated string literal.

The program works and prints the message but also produces an error.

Not true! The program doesn't actually print the message - just an error about an unterminated string literal.

Figure 2 - HTML for a multiple-choice “quiz” question

The second iteration of NoobLab is now being used as the primary tool for course delivery on the first year undergraduate module at Kingston University, Practical Programming. The ethos of this module is to try to bring an element of fun to programming and dissuade students from the perception that programming is “boring” or “difficult”. Therefore, the vast majority of the students’ time is spent in the NoobLab environment, where they are encouraged to examine, modify and compose program code. Errors are presented as a good thing because they help the student to learn.

As a result, the tutor’s role becomes free-flow. Each week, a Wulf-style framing lecture summarises the previous week’s activities and introduces the new material, taking no more than 15-20 minutes to highlight pertinent aspects of the content before inviting students to undertake the practical tasks. While the tutor remains present, the primary “guide on the side” becomes the environment itself. The tutor remains present to supplement the environment, and to bring interesting developments from individual students (e.g. errors, innovative approaches) to the group as a whole. Such “interesting developments” might then be factored into the learning material itself, potentially becoming impromptu feedback supplied by the environment on future instances of the module.

Anecdotal feedback from students during sessions is positive, and particularly that of repeat students who are able to directly compare and contrast with the previous incarnation of the module. In particular, students appreciate the single, seamless presentation of framing learning content alongside the practical environment, and the fact that they can work at a pace that suits them yet still receive the same quality of teaching as others. Usage statistics combined with the visible lack of Facebook and other distractions on students’ screens during workshops seem to indicate that a good level of engagement is occurring. Meanwhile, the support provided by the environment has meant that a single tutor and postgraduate helper can accommodate a group approaching 70 in size.
4. Future work and the Pedagogy of Programming

Jenkins (2002) noted that the purpose of an introductory programming course should be to teach how to program and not to teach a specific language. In order to simplify the development of a browser-based development environment, NoobLab has used Javascript as a base language thus far, albeit with a small number of extensions to support the introduction of a “console” with user input and output. Javascript also makes sense from a pedagogic perspective as an introduction to programming in that many of the constructs and much of the syntax is similar to other high level languages. When students progress to other modules that specifically deal with (for example) Java they should find much of it familiar. However, if the NoobLab environment is to have utility beyond the beginner, it will need to be able to support languages other than Javascript and some preliminary work involving Java has already taken place.

The environment is also seen as a tool for investigating the pedagogy of programming and particularly the efficiency of course and material design. As a student works through the content, a considerable amount of usage information is logged; not only simple navigation, but activity within quiz questions, the code window and attempts to run code and solve exercises.

<table>
<thead>
<tr>
<th>Index</th>
<th>Timestamp</th>
<th>Action</th>
<th>Location</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>09:59:00 2012/02/06</td>
<td>RunStart</td>
<td>CI1152B:1:7</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>09:59:03 2012/02/06</td>
<td>RunUserInput</td>
<td>CI1152B:1:7</td>
<td>Paul</td>
</tr>
<tr>
<td>49</td>
<td>09:59:03 2012/02/06</td>
<td>RunSuccess</td>
<td>CI1152B:1:7</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>09:59:15 2012/02/06</td>
<td>TestStart</td>
<td>CI1152B:1:7:1stIf</td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td>09:59:15 2012/02/06</td>
<td>TestFailed</td>
<td>CI1152B:1:7</td>
<td>2/3</td>
</tr>
<tr>
<td>52</td>
<td>10:01:29 2012/02/06</td>
<td>RunStart</td>
<td>CI1152B:1:7</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>10:01:31 2012/02/06</td>
<td>RunUserInput</td>
<td>CI1152B:1:7</td>
<td>PAUL</td>
</tr>
<tr>
<td>54</td>
<td>10:01:31 2012/02/06</td>
<td>RunSuccess</td>
<td>CI1152B:1:7</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>10:01:32 2012/02/06</td>
<td>TestStart</td>
<td>CI1152B:1:7:1stIf</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>10:01:33 2012/02/06</td>
<td>TestPassed</td>
<td>CI1152B:1:7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3 - usage logging

Figure 4 shows an excerpt from a student’s usage logs, and in this instance we can see the process of a student attempting to complete an exercise. (Their code is also captured, but not shown here for reasons of brevity). The lines at indexes 47-59 show a student running their program and (mistakenly) believing that it fulfils the exercise criteria and responds to the user input “Paul”. Lines 50-51 show their work passing 2 out of 3 test criteria (the failure in this instance being a requirement for case insensitivity). Lines 52-56 show a subsequent modification which succeeds in passing the test. Note the Data field in line 52. This is a difference index based on Levenshtein (1966), and indicates the level of changes that have occurred from one run to another.

As students generate this usage detail, the expectation is that common patterns will emerge. One such pattern on Practical Programming exposed a common misconception where a number of students were declaring multiple variables unnecessarily to perform simple arithmetic operations. This common “signature” exhibited by students exposed a shortcoming in the material involving the declaration and use of variables. This led to the tutor modifying the material accordingly – another example of the learning loop in practice.

Interestingly, the authors have recently become aware of Haverbeke (2007), which quite independently and coincidentally used an almost identical approach in their online materials by introducing an artificial console into Javascript.
We can predict other, generic signatures across all workshops. A pattern of RunStart-Error-RunStart-Error-RunStart-Error with only minor difference indexes between them might indicate a student who is stuck, and is desperately making minor cosmetic changes without truly understanding what the error or changes mean. A similar pattern followed by success but with a large difference index might indicate possible plagiarism.

Longer term, the wish is to correlate common signatures with performance metrics. A signature that is common across poorly performing students might be flagged in future instances of the module as one requiring tutor intervention. Signatures might also exhibit themselves across students but specific to activities – so an activity that routinely resulted in the students generating a high number of errors or test failures might be flagged as one that requires examination and re-evaluation.

5. Conclusion

The teaching of computer programming is often seen as a difficult proposition by both tutor and student alike. The wide range of literature covers an equally wide range of different approaches, yet perhaps Gantenbein (1989) stated it most precisely:

“Anyone with a reasonable intelligence and some grasp of basic logical and mathematic concepts can learn to program; what is required is a way to demystify the programming process and help students to understand it, analyse their work, and most importantly gain the confidence in themselves that will allow them to learn the skills they need to become proficient.”

The NoobLab environment provides a tool where students can acquire confidence and proficiency without the fear factor, in a setting they find comfortable. By providing a seamless blending of framing material and development environment, that provides feedback to the student both on demand and spontaneously, we have created an online analogue not only of the learning loop inherent in a well-crafted programming workshop, but also of the framing lecture that sets the scene for this workshop. Perhaps more importantly, it provides a platform for the study of the pedagogy of programming, the continuous improvement of and reflection upon course design, and increasing student retention and satisfaction.

6. References


