A VIRTUAL LEARNING ENVIRONMENT FOR INTRODUCTORY PROGRAMMING

Ian Allison
The Nottingham Trent University
Burton Street
Nottingham
ian.allison@ntu.ac.uk
http://dcm.ntu.ac.uk/5_staff/staff_ian.htm

Paul Orton
The Nottingham Trent University
Burton Street
Nottingham
paul.orton@ntu.ac.uk
http://dcm.ntu.ac.uk/5_staff/staff_pao.htm

Heather Powell
The Nottingham Trent University
Burton Street
Nottingham
heather.powell@ntu.ac.uk
http://dcm.ntu.ac.uk/5_staff/staff_hmp.htm

ABSTRACT

Teaching of initial programming is a significant pedagogical problem for computing departments. It is shown that by understanding the changing characteristics of computing students helps to identify their learning approaches and requirements. These findings are used to explain the rationale for the development and use of a virtual learning environment to support the learning of introductory programming.

Keywords

Virtual learning environment; teaching programming.

1. INTRODUCTION

The teaching of the initial programming language is a significant pedagogical problem for computing departments. With increasing numbers of computing students in the UK, the teaching of this core skill has become more difficult. A higher proportion of students are selecting to study computing without a natural aptitude for programming.

One approach adopted to address this problem in some curriculum areas is computer-aided learning (CAL). A variety of CAL tools have been applied to support teaching in computing. In part the emphasis for this has been on reducing class contact. However, the reduction of contact time is only one aspect of CAL tools. CAL has been shown to provide valuable support in many contexts [e.g. 13].

A virtual-learning environment (VLE), known as Idyll, has been developed by NTU in response to the problems outlined above. Lessons are drawn from previous experience with CAL tools to develop Idyll. The purpose of this paper is to show how Idyll can help to overcome a number of problems of teaching and learning programming. The paper initially outlines the background to this initiative and the use of computer-aided learning tools in computing. We then discuss the new approach and show how it addresses the problems outlined.

3nd Annual Conference on the Teaching of Computing, Loughborough University

2. BACKGROUND

2.1 Changes in the computing academic environment

Computing was a niche discipline over a decade ago, with many new students gaining their initial experience in programming prior to starting their degree. These students were “motivated to learn about a subject they found interesting” [7]. Today’s students have quite a different background with home PCs often seen as an entertainment vehicle rather than just a tool for dedicated hobbyists. The IT skills shortage and resulting lucrative job market entice many school leavers into undertaking computing degrees. The consequence of these changes is that a diverse set of students enters university to study computing [7]. Many of these students are not equipped with an appropriate problem-solving aptitude for learning programming. So, computing courses have wastage rates that are high compared to other disciplines.

Over a similar time period, industry standard languages were introduced to the curriculum instead of simplified teaching languages such as Pascal. There was pressure to change because Pascal was “suitable only for small self-contained programs that have only trivial interactions with their environment” [10]. Since the early 90’s C++ has been the core programming language taught at NTU. Arguably, this change has achieved the desired objective of assisting graduates to obtain jobs in the employment market. However, current languages and programming environments complicate the learning

process for novice programmers. These environments place an additional burden upon the learner to understand not just the language and concepts of programming, but also the idiosyncrasies of the development environment. All of this places a cognitive overload on those who do not have the natural aptitude for programming.

Student feedback at NTU gives very high ratings for all aspects of the teaching of introductory programming modules. It seems quite natural then to assume that the problem is simply that C++ is intrinsically too hard or the students are not capable of using it. Some universities have adopted alternative languages like Java or Visual Basic in an attempt to overcome problems with language complexity [e.g. 2], but this does not solve the problem. It is our view that understanding the different approaches and motivation for learning programming is the key. By adapting to the needs of the students we can support those with less natural ability to learn.

2.2 Student Motivation and Learning

Student motivation is a complex issue and influences the individual's approach to learning. It is widely assumed that computing students are extrinsically motivated. However, it is argued that students' motivation is a product of their value of success and their expectation of success [6]. The value of success is equivalent to the student's own extrinsic motivation factors, such as a well-paid job or high marks. The problem in programming is that the expectation to succeed wanes with early disappointments and difficulties. For students to expect to succeed they must see that they have control over their own success.

An active role in the learning process encourages learning. An active learner is not necessarily independent, but one that has more involvement with the subject matter. Figure 1 shows that an active learner will selectively apply the knowledge gained and evaluate ideas through applying them to a problem. Motivation to learn will increase through successful application of the techniques learnt.

By their natural aptitude for learning to program students can be classified in order to ensure teaching resources are targeted appropriately. For students who can be classified as strugglers [7] our role as motivator is essential. The tools that we use need to support student learning by helping students to expect to succeed [6]. The struggler's "confidence, anxiety and other affective states…can change as a result of experience" [1]. So the objective of this project is to address the constraints on learning amongst this group of students. "After all, the art of pedagogy is, as I see it, the craft of helping other people to teach themselves even those things which they, at some time and for any reason, may not be inclined to learn, in spite of their desire to do so." [14].

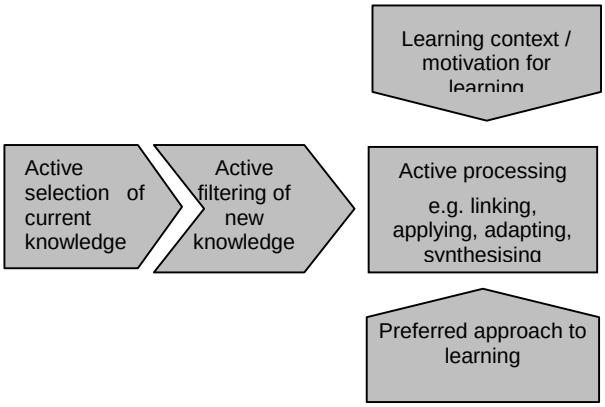

Figure 1 Active student learning adapted from [3]

3. CAL TOOLS FOR PROGRAMMING

Computer-aided learning has many potential benefits but also many potential drawbacks that developers need to be aware of. The perception that drove the Teaching and Learning Technology Programme[1] in the early 1990s was that content could just as easily be delivered through a computer interface as via a lecture, thereby saving lecturer time. Moreover, the learning would be student-centred because it would take place at the student's own pace and at their convenience. Whilst some of this is true, there are several factors that favour a more imaginative use of computers for teaching than mere content dissemination. One is the relatively large amount of time that it takes to develop presentational material for CAL. Another is the need to motivate students. Just giving learners access to a CAL package does not guarantee that they will use it. Learners that benefit most from this approach are those in work who use CD material to improve work-related skills. A human facilitator is still needed to initiate the learning and maintain momentum for most undergraduate learners.

There are two other important factors. One is that to achieve the most thorough level of understanding in any subject, learners need to apply what they have learnt. This is where the potential of CAL can be most fully realised. For maximum efficacy in terms of the quality of learning, highly interactive environments are needed. These environments allow students to try things out and learn by experimenting. These are known as simulations and can take a number of different forms. Whilst initially

[1] TLTP 75m programme funded by HEFCE, SHEFC, HEFCW and DENI from 1992 to 1996

expensive to produce, they are by their nature relatively reusable so ultimately can be more cost-effective than presentational CAL.

The final factor concerns the use of CAL for weaker students. We have noted that CAL is student-centred, supporting individual variation in the pace of learning. It is also a powerful medium for allowing concepts that abler students might understand from written text to be brought to life. This is achieved through graphical demonstration and the use of concrete examples and natural semantics. This enables students with weaker logical and linguistic skills to grasp more complex concepts.

The teaching of computing is a special case with respect to CAL. The medium and the message are inextricably linked. This is most strikingly apparent when programming is considered. It is perhaps for this reason that CAL for programming has not been fully exploited. Any programming environment already provides the learner with a highly interactive environment where students can try things out and learn by experimenting. So what else is necessary? A programming environment on its own has no pedagogic input, being effectively an environment that allows learning by discovery. Supporters of discovery learning, such as [11] and [12], might welcome this as giving the best option for complete understanding. However, research in other environments has shown that some pedagogic guidance is beneficial [4]. The main focus for pedagogic input to programming environments has been to manage the learning by scheduling the student in which problems to attempt next and in assessing the student's answers. The main benefits for students of these facilities is the immediate feedback received and the removal of some of the effort and motivation required for them to work through the course of examples completely independently. These advantages combined with the benefits to the tutor of much reduced marking load and some protection against plagiarism make for a powerful package.

However, previous work along these lines makes no attempt at addressing the difficulties learners experience with the subject matter itself. One reason for this is that it is very difficult to incorporate a pedagogic component into what is effectively a very complex simulation [5]. The system has so many possible states that to achieve a reasonable level of tutor feedback would require an AI engine to cope with the complexity. Some free-standing CAL packages have been developed to aid students in understanding aspects of programming e.g. the graphical demonstration of linked lists, but the use of virtual environments to help in the teaching of C++ has been limited to mirroring the pattern of worked examples conventionally used, which in turn has been dictated by the characteristics of conventional C and C++ programming environments.

The approach presented in this paper is able to address the difficulty of the subject matter because the virtual environment developed allows complete flexibility in terms of examples students are able to try. It has been possible therefore to reassess the ordering of the sequence in which students practice and assimilate the components of the language. This has enabled progression through the material to be organised such that it is in small increments as opposed to the conventional approach which requires students to understand a lot of syntax and several programming concepts just to get started.

4. A VIRTUAL LEARNING ENVIRONMENT

4.1 A Rationale for a Virtual Learning Environment for Programming

The term virtual learning environment has been coined to reflect a broader concept than either CAL or content-based internet browser systems. Virtual learning environments bring together learning opportunities, assessment and feedback, management of student progressions, and interconnectivity with university-wide managed-learning environment [MLE] [9]. Learning environments need to be multi-layered to be effective [8]. Idyll is just one strand of the overall VLE/MLE in the computing domain at NTU. Aspects of content and broader university facilities are provided by an intranet system connected into the university student information systems. The Idyll VLE system is specifically aimed at the students on the introductory programming modules.

One view of learning environments is that they are "a coping strategy to deliver government targets within current resources" [8]. Particularly with larger numbers of students where several members of staff are involved on a single course the problem of managing the learning environment becomes much more complex. Monitoring student performance and identifying weaker students at an early stage is problematic. A system that integrates student monitoring, the delivery and testing of exercises is essential. If it also gives rapid feedback and help to the students then tutors can concentrate on giving good support to those who most need it.

A pitfall with the introduction of VLEs, as with CAL, is the danger of focusing on the benefits to the tutor. The only way learning environments can make "a real difference has to be by allowing better learning

to take place by providing learners with more choice and flexibility which in turn means better content and better interactions" [8]. By encouraging students to realise that they have control over their own success motivates them to reach higher standards and become independent learners. So, the VLE outlined here is designed to provide students with the motivation to learn by rewarding incremental understanding. It is designed to boost student confidence at the very critical early stage. By providing instant checking, feedback and monitoring of results it facilitates better nurturing of struggling students.

4.2 Idyll: an VLE for Learning Programming in C++

The Idyll VLE provides a simplified, streamlined and more intuitive introduction to programming. It does this by reducing the complexity of the task to a single problem-solving exercise. The cognitive load of using a standard development environment is removed because the tool acts as an interface to the Visual C++ development environment. Idyll has the flexibility of learning normally associated with CAL tools. The system provides positive reinforcement of the learning through constant feedback and provides incentives through a rating system. All of this is done within an integrated environment that allows access to taught content and provides tutors with information to allow the management of students' progression.

The predominant approach in C++ textbooks is to start with terminal type interfaces that use Cin and Cout classes for the input and output. Executable programs are produced. In PC Windows environments these applications run in DOS windows. Since DOS applications were never designed to run under the powerful new operating systems they cannot easily access Windows graphics or many of the powerful API functions. From the student's point of view it appears that they are being taught skills that are out of date that do not let them access many of the facilities that they aspire to use. Idyll provides a more gratifying experience by enabling the early use of powerful Windows facilities. With teaching of terminal or DOS type programs there are questions about whether a consistent message is received by the student. Currently, nearly all students are brought up on Windows interfaces, where input means point and click and output means a lot more than a character string.

A large proportion of programming today is not about producing stand-alone programs from scratch but rather about writing code fragments or dlls that link or communicate with a mass of proprietary software. In essence, the emphasis has shifted towards programming that is largely about writing functions or fragments of code. The MS Visual C++ development studio allows you to develop Windows applications but does not require you to write the Winmain wrapping parts. Programmers can simply add functions or code fragments.

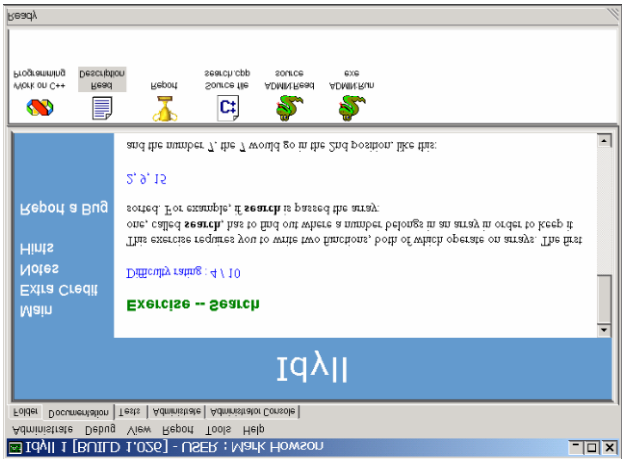

Figure 2. A problem exercise

Idyll uses this facility to isolate the problem to be tackled to a fragment of code in a dll. It issues exercises in the form of problems with a function specification (see figure 2). The exercises typically have some correct and some missing information. There are links to relevant teaching material for the particular problem. The student simply edits and compiles the program. The system automatically runs the compiler environment. Idyll detects that a new dll has been compiled and then runs tests. Before running the code, Idyll checks automatically to see if the parameters and return types are correct. The code is then called with a whole range of appropriate test data. If the code crashes, Idyll traps the exception and suggests possible causes. Idyll automatically times out when code is trapped in a loop.

Similar applications, such as Coursemaster, use a template to check the program 'correctness' and style. Idyll does not attempt to look at style issues, leaving this to the tutor. Idyll, however, rigorously checks the results giving appropriate feedback.This testing of students' programs is an important aspect of the learning process, as most students have an immature approach to testing their own code. It rates the solutions and records a mark before providing a new challenge. It will give the students their current rating statistics on the work to date.

Idyll maintains a database to keep track on students, look after their work their achievements and monitor attendance. It is intended that students should see it as a helpmate rather than a taskmaster. The database keeps all the student's attempted

solutions. This information is processed and accessible to the lecturer but, importantly, the results it are also accessible to the individual to give feedback on performance. Students are also able to view peer group statistics.

By taking this approach, students will be able to start solving significant problems without being distracted by I/O issues or testing strategies. Students should be able to gain confidence with the use of the most basic concepts (variables, assignment conditional branching and loops). Ultimately it is intended that the system will allow a smooth transition to classes.

5. EVALUATION AND FUTURE WORK

The basic ideology has been widely discussed by staff and students and is very well received. Implementation of this tool is at an early stage but initial feedback has been received from students. A focus group was set up consisting of students of different abilities. Stronger students who wish to explore the wider environment approved of the system, but felt constrained. The less-confident students were very positive with the additional support. A pilot-study of selected students will be conducted during the coming year.

Further consideration needs to be given to the types of problems that Idyll will use. It is expected that a significant number of exercises will be built up to provide students with a good choice as they develop confidence to tackle the initial programming concepts. A further development to enable distance learning is required. One such aspect would be intelligent tutoring of a student. Idyll does not currently provide this support, but it is designed for use within a traditional tutor-centred environment. The role of the teacher remains critical to the success of a Virtual Learning Environment.

6. CONCLUSION

Idyll provides a computer based environment that is both effective and improves motivation to enable the learning of initial programming skills by students who do not have a natural aptitude for it. A learning tool for programming should have integral syntax checking, test programs thoroughly, and provide objective assessment of other style criteria [5]. Idyll addresses all these requirements.

7. ACKNOWLEDGEMENTS

This paper would not have been possible without the contributions from the whole of the Idyll project team. We gratefully acknowledge the support of the LTSN-ICS development fund.

8. REFERENCES

[1] Baldwin,L.P., Eldabi, T. and Macredie, R.D., *Learning Strategies and Individual Learner Differences; their role in the learning of programming*, In: procs of the 6th UKAIS conf., 649-659 (2001).

[2] Bergner, K. and Huber, F. *Systems Development with Java: Experiences from a practical project course in software engineering*, In: Procs of 8th Int. Workshop on Software technology and Engineering Practice, IEEE Computer Society, 382-389 (1997)

[3] Brown, G and Atkins, M *Effective Teaching in Higher Education*, Methuen (1988).

[4] Duffy, T. M. and David, H., J., *Constructivism and the Technology of Instruction*, Lawrence Erlbaum Associates (1992).

[5] Jackson, D. *Computer-based teaching of programming: can it be done?* In: Hart, J. and Smith,M.(eds)Innovations in Computing Teaching 2: Improving the Quality of Teaching and Learning, SEDA paper 91.

[6] Jenkins, T. *Teaching Programming- A journey from teacher to motivator,* In Procs of 2nd Annual Conf. of the LTSN Centre for Information and Computer Sciences, (2001).

[7] Jenkins, T. and Davy, J., *Dealing with Diversity in Introductory Programming,* In Procs of 1st Annual Conf. of the LTSN Centre for Information and Computer Sciences, 81-87 (2000).

[8] JISC workshop on Managed Learning Environments final report. http://www.jisc.ac.uk/pub00/mle/final_rep.html (2000)

[9] JISC, Introducing Managed Learning Environments, http://www.jisc.ac.uk/mle/ (2001)

[10] Kernighan, B.W. Why Pascal is not my favorite programming language http://www.lysator.liu.se/c/bwk-on-pascal.html (1981).

[11] Papert, S., *Mindstorms: Children, computers and powerful ideas*, Basic Books (1980).

[12] Piaget, J., *Structuralism*, Basic Books (1970).

[13] Powell, H and Palmer-Brown, D., Computer-Aided Learning in Computing, *Innovation*, **1**,47-50, (1996).

[14] Wankowski, J., Increasing students' power for self-teaching in Raaheim et.al. (eds.) *Helping Students to Learn,* SRHE & OU Press (1991).