A Self-Practice Online Tool for Teaching and Learning Computational Skills in Engineering Curricula

Abbas El-Zein, School of Civil Engineering, University of Sydney, Australia, aelzein@usyd.edu.au,
Tim Langrish, School of Chemical Engineering, University of Sydney, Australia, tim.langrish@usyd.edu.au
Nigel Balaam, School of Civil Engineering, University of Sydney, Australia, nbalaam@civil.usyd.edu.au

***Abstract* - Computer programming is a skill that engineering students are expected to acquire in their undergraduate studies. Many engineering schools and faculties have moved towards including engineering programming as part of a first-year course taught to large engineering classes. This approach is effective in rationalizing resources and improving the cost-effectiveness of course delivery. In addition, it can lead to wholesale quality improvements in teaching and learning. However, the size of classes and the large variety of student backgrounds can lead to difficulties in achieving the required learning outcomes. Flexible learning has been shown to be potentially effective in addressing such issues. We describe the design and development of a WebCT-based self-practice online tool (SPOT) to support student learning of computer programming. The tool is divided into three components which focus on three aspects of computer programming with increasing levels of difficulty: a. computer programming syntax, b. understanding the way computer programs work and c. writing computer programs. We present the way the tool is integrated into the overall learning flow of the course and its role in course assessment. Finally, we discuss statistics of usage and usefulness in achieving learning outcomes, drawn from a survey of students and make specific recommendations concerning the implementation and development of such tools.**

Index Terms – e-learning, computer programming, mathematics, large classes

INTRODUCTION

A new paradigm of online education has spawned a rich literature on the effectiveness and efficiency of various forms of electronic teaching tools, from full online courses [e.g., 1] to web-assisted, lecture-based courses [e.g., 2,3]. The ability of these modes of teaching and learning to achieve desired engineering learning outcomes and their efficiency in achieving that aim remain open questions. Evidence points to an improvement in learning efficiency, although students with access to online resources are not necessarily more likely to achieve learning outcomes [4-6]. Although the number of distance courses has risen significantly over the last decade, mixed modes of delivery, with face-to-face settings supported by online tools, remain the dominant form of online learning on campus. There is clearly a need in the literature for greater exploration of flexible modes of learning including e-tools, when teaching computational skills to engineering students.

Programming skills are now deemed essential in most engineering schools. Both structured languages, such as FORTRAN and C, and computational tools such as MATLAB, have been used in engineering curricula. Hodge and Steele [7] surveyed engineering programs in the US and found that FORTRAN had lost its dominance and computational tool were increasingly employed by educators because of the trend towards integrating various computational functions in a single environment. At the Faculty of Engineering of University of Sydney, MATLAB was adopted in an introductory computational course (ENGG1801) for first-year engineering students for two reasons: a. its ability to integrate programming with matrix operations and graphics and b. the relative simplicity of its programming tools which offer the possibility of introducing students to fundamental programming concepts without requiring them to grapple with other aspects of structured programming such as dimensioning and compilation. However, the development of programming skills by first-year engineering students has proved to be a complex task, especially in large 500+ student classes, and a small but significant proportion of students (20%) failed to perform satisfactorily.

This paper discusses the design, development and implementation of an e-learning tool into ENGG1801 and offers a student-centered model for integrating e-learning with other course resources, including face-to-face interaction. The aim of this integration is to increase the number of students who achieve the required learning outcomes and reduce the percentage of students who fail the course. While other methods for improving learning outcomes have been suggested in the literature (e.g., a crash course preceding the main course as described by Christensen et al [8]), e-learning remains more attractive because of its potential cost-effectiveness in terms of student time and financial expenditure.

CURRICULAR CONTEXT

ENGG1801 is made of two components which run in parallel: Computer-Aided Design (CAD) with SolidWorks and programming using MATLAB. The first component

occupies around 40% of the course, while the second accounts for 60%. These percentages reflect the division of hours of lectures and lab sessions, as well as assessment weights. In this paper, we focus on the programming part of the course and will not discuss the CAD component. ENGG1801 is aimed at first-year civil, mechanical and chemical engineering students. The number of students enrolled in the course have increased, from 450 in 2004 to 550 in 2007.

The programming part of the course aims to develop students skills at writing simple computer programs that can solve simple mathematical and engineering problems. By the end of the course, students are expected to be able to write sequential programs using the following families of commands: input and output, conditional structures such as "if" and "case", loop structures such as "for" and "while", modular structures such as "functions" and "subroutines" and, finally, graphic functions intrinsic to MATLAB. Although MATLAB is used in teaching, course instructors make it clear to students that the purpose of the course is not to teach MATLAB per se, but programming more generally. Skills and programming concepts used in one sequential programming language are still valid in another, with minimal adjustment, in the same way that driving skills acquired with one car brand are transmissible to another. Students are given one hour of programming lecture per week, after which they attend a computer lab session, with around 50 students in each session, where they are asked to solve a programming problem, with help from tutors.

A number of issues arose in the first two deliveries of the course in 2004 and 2005. The first issue was related to tutor-student contact. Although three tutors were allocated for each MATLAB programming session, with a ratio of 16 students per tutor, some students clearly felt they needed more tutorial support. Given budgetary constraints, it was impossible to reduce this ratio. Instead, an additional tutorial session for programming, called a clinic session, was introduced in 2006 and was run by the lecturers, rather than the tutors. Attendance was voluntary and open to all students who needed extra support. In addition, tutors were asked to provide more pro-active guidance to students at the beginning of each session.

A second issue was related to programming quizzes. Three quizzes were given during the semester. Given the large number of students, a quiz system, introduced in 2004 and followed in 2005, had students sitting their quizzes during their lab sessions, on specially designated weeks. Tutors invigilated and marked papers, immediately after students finished writing their answers on the computer screen. A simple marking system (0 to 3) was used. The system was effective in that marking was done quickly and the effort was widely distributed between tutors. There were however three drawbacks. First, students were worried about inconsistency of marking between tutors, and there was no way of guaranteeing such consistency, given the large number of tutors—despite written instructions given to tutors, face-to-face meetings between tutors and instructors prior to the quizzes, and the simplicity of the marking system. Second, since ten different tutorial sessions per week ran to accommodate the 500 or so students, ten different versions of each quiz had to be written. Third, invigilation was rather difficult, despite the tutors' best efforts, given the proximity of students seats in the computer lab.

A third issue, perhaps the most significant one, became clear to us during the semester in 2004, and was confirmed in the final exam and during 2005. The most difficult aspect of the course was programming. The failure rate in the course stood at around 18% and the majority of students who failed did so as a result of programming. A number of measures were taken in response to this, including changes that allow a more gradual introduction of programming concepts, as well as more exercises solved in the class and the lecture notes.

The above three issues—tutoring, assessment and learning of programming concepts—are obviously related. However, for all their complexity, it is obvious that adequately-designed e-learning resources can play a major role in addressing them. This is particularly the case given the large number of students and the inevitable budgetary constraints in any curricular activity. The question asked in small, more conventional classroom environments where the teaching and learning community consists primarily of a teacher and a few dozen students is: "how best to achieve the learning outcomes of the course?" This question is best developed in a slightly different form for larger classes and more complex teaching and learning communities which include coordinators, instructors, tutors, administration staff, as well as a few hundred students. A more pertinent question in this case is: "what is most the cost-effective way of achieving learning outcomes among the highest possible number of the students, hence reducing the degree of failures in the course?" A self-practice online tool (SPOT), which addresses all three issues raised above, has been designed and developed, and is offered here as part of a possible response to this question.

SELF-PRACTICE ONLINE TOOL (SPOT): RATIONALE AND ARCHITECTURE

We developed an online tool with the following objectives:
a. to put in place better flexible learning resources for students.
b. to help students assess their own progress and provide with a clear path for seeking additional help.
c. to better integrate lectures and lab sessions.
d. to improve the quality of assessment through quizzes.

A database of online questions (DOQ1) with around 300 multiple-choice question was developed. The questions were grouped under nine categories: MS Excel basics, matrix algebra, matrix MATLAB operations, and the following sets of commands in MATLAB: text, conditional ("if" and "switch"), "for" loops, "while" loops, "function" and graphics. Each category was further divided in two groups corresponding to two levels of difficulty. Each question

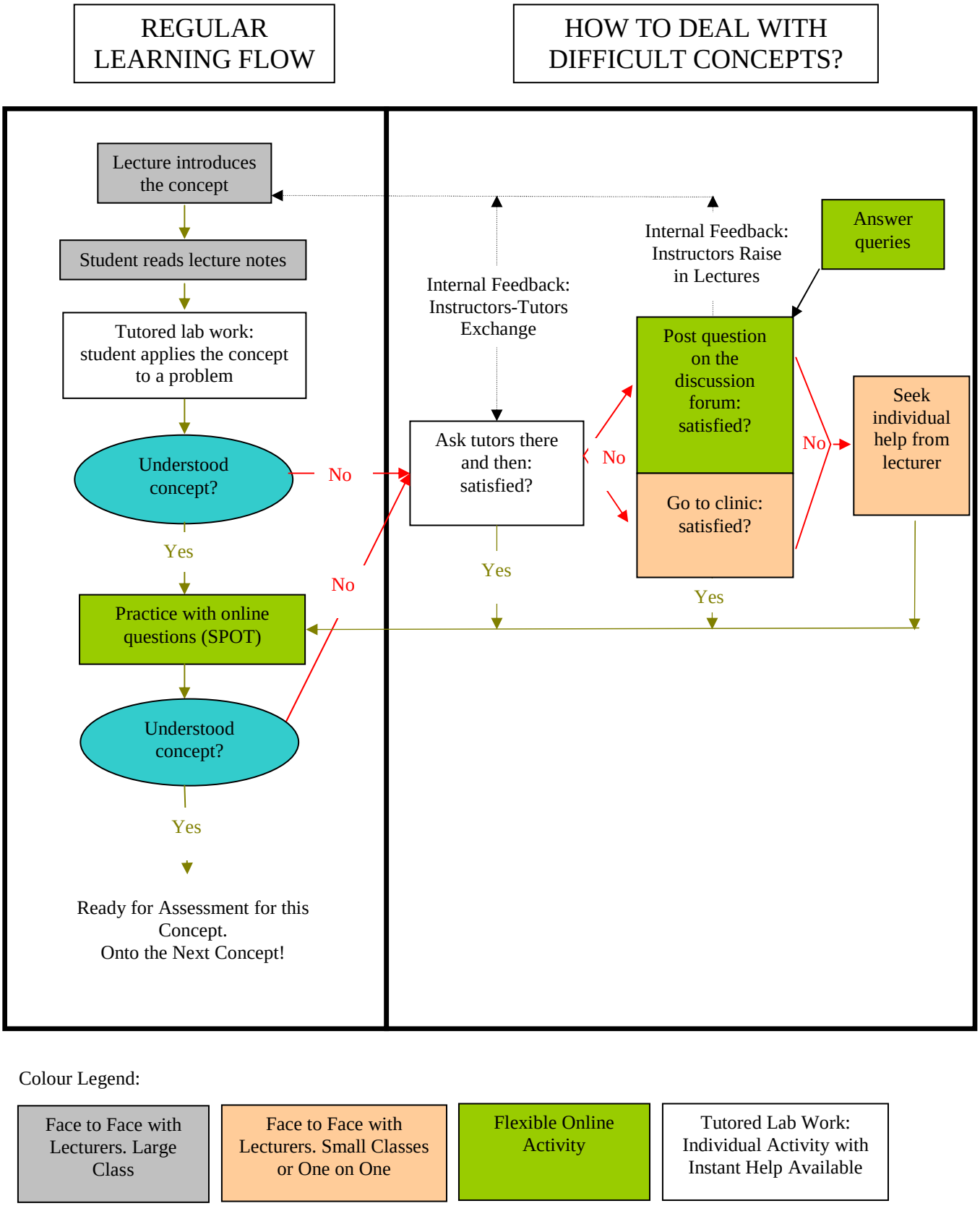

FIGURE 1
LEARNING ACTIVITY MAP FOR THE COURSE

carried five possible answers, as well as a few lines of justification for the correct answers and usually a note on each of the incorrect answer. DOQ1 questions assessed the student's understanding of the syntax and role of each set of commands. DOQ1 was later augmented with DOQ2 and DOQ3. DOQ2 is made of multiple-choice "skeletal" questions which presents students with small programs and ask them to fill in missing commands or spot errors in the programs. DOQ3 carries programming questions which asks students to write computer programs to solve given problem. Hence, DOQ1, DOQ2 and DOQ3 take the students through the process of learning programming commands, understanding how computer programs work and writing computer programs. (We will refer generically to DOQ1, DOQ2 and DOQ3, by DOQ, in the remainder of the paper). DOQ was then used to generate two WebCT tools:

a. A Self-Practice Online Tool (SPOT1, SPOT2 and SPOT3, corresponding to DOQ1, DOQ2 and DOQ3, respectively, and collectively called SPOT) that could be accessed online by students enrolled in the unit of study at any time. The student could choose a particular category and test his or her ability, by attempting to answer the question, checking whether he or she had answered correctly and get specific feedback on each answer, as well as general feedback on the question.

b. A quiz tool (QT) that would be used to run 3 quizzes over the semester. Quiz 1 would be drawn from DOQ1, quiz 2 from DOQ 1 and DOQ 2, while quiz 3 is entirely made of DOQ 3 questions.

Once DOQ was developed, SPOT and QT were easily set up within the WebCT environment, at no extra cost. SPOT and QT were assigned a specific role within a new course learning map, developed to address the problems discussed earlier (see Figure 1). The figure shows the regular learning which students probably went through most of the times. After attending a lecture introducing a new programming concept, the students read the corresponding lecture notes and lecture slides, went to the lab session to solve the corresponding problem and attempted the corresponding SPOT questions. Whenever they experienced difficulties, they could speak, one-on-one, to tutors during lab sessions, post a question on the discussion board for the course and go to the clinic session. Students could also choose to email or visit the course lecturers in their offices. Questions on the discussion board, as well as communication between tutors and instructors, helped the teaching staff keep track of the kind of difficulties arising in the class, which may then be specifically addressed by instructors during lectures. DOQ1 and SPOT1 were developed in time for semester 1 2006. DOQ2 and DOQ3, with SPOT2 and SPOT3, were developed in time for semester 1, 2007. The Respondus program was used for developing the questions, which were then exported into WebCT.

DISCUSSION

Half-way through the semester in 2006, students were asked to fill in an anonymous questionnaire about the course, including the following 3 questions about SPOT1 (since SPOT2 and SPOT3 had not been developed by then):

1. HOW OFTEN have you accessed SPOT1 since the beginning of the semester:

a. At least twice a week
b. Less than once a week
c. Less than once every two weeks
d. Not at all

2. HOW USEFUL did you find SPOT1 in helping you to learn programming concepts:

a. Very useful
b. Fairly useful
c. Not so useful
d. Not useful at all

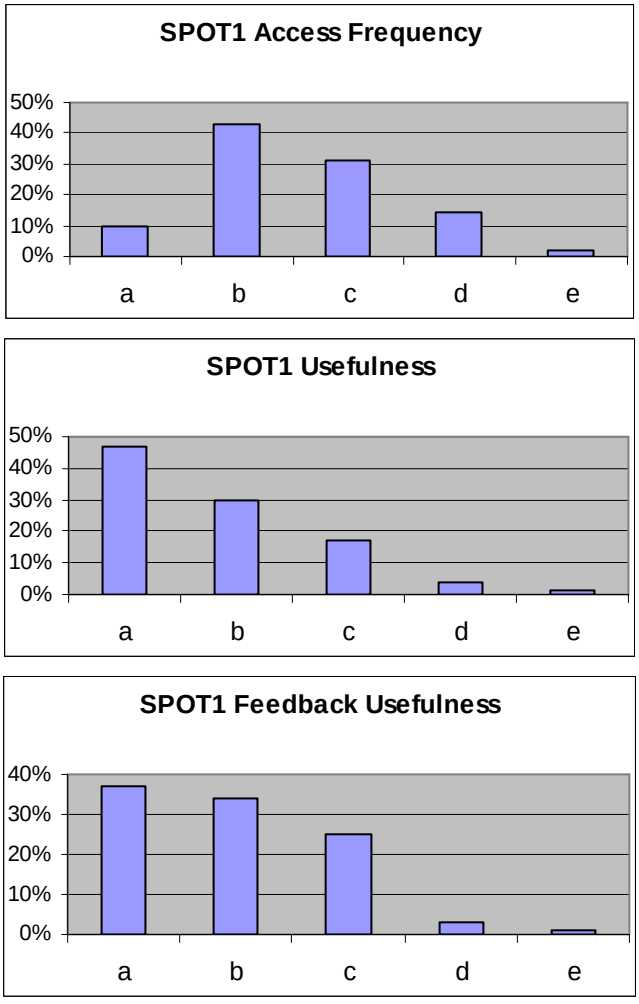

FIGURE 2
STUDENT RESPONSE TO SPOT1

3. How useful did you find the FEEDBACK on answers in SPOT1?

a. Very useful
b. Fairly useful
c. Not so useful
d. Not useful at all

Around half the class, 236 students, filled the questionnaire. Survey statistics for the above questions are shown in figure 2. While 90% of respondents used SPOT1 less than once a week, 45% of respondents found SPOT to be very useful, while around 75% found SPOT and its feedback system to be at least fairly useful. Selection bias, with more involved students more likely to answer the questionnaire, may have increased the percentage of students using SPOT frequently and finding it useful. However, this must be qualified by the fact that the survey was conducted half-way through the semester of the first delivery and concerned only SPOT1. A survey will be conducted towards the end of the semester in 2007, covering SPOT1, SPOT2 and SPOT3 and its results are expected to be more conclusive.

Students feedback scores for the ENGG1801 improved significantly from 2005 to 2006. While SPOT1 was not the only change introduced into the course after 2005 and could not therefore be given total credit for the improvement, it was certainly the most significant innovation.

The new tools brought a major reduction in complaints about the fairness of marking of quizzes. Even when long answers rather than multiple-choice questions were used in quiz 3, the online submission could now be transferred to a select group of tutors who performed the marking, hence ensuring more consistency. The tool provided students with more learning resources and enhanced the assessment quality of the course. The multiple functionality of such e-tools is a key factor in their cost-effectiveness and justifies more powerfully the required development cost. A more challenging question that we will pursue this year and the following one is the extent to which the tool has helped in better achieving the programming learning outcomes of the course.

ACKNOWLEDGMENTS

The Dean of the Faculty of Engineering at the University of Sydney, Professor Greg Hancock, kindly provided the first funds for SPOT development. The Teaching-Improvement Fund program of the University of Sydney awarded us a grant for the extension of SPOT1 into SPOT2 and SPOT3. Sam Smith and Chi Yan Tang played a vital role in the project by writing the questions for the databases of SPOT1, SPOT2 and SPOT3. James Underwood conducted valuable quality checks on SPOT2 and SPOT3. Stephen Sheely and his superbly competent WebCT staff at the University of Sydney provided us with indispensable technical support in dealing with many rather complex issues related to the WebCT and Respondus software.

REFERENCES

[1] Bourne J, Harris D, Mayadas F. 2005. Online engineering education: learning anywhere, anytime. *Journal of Engineering Education* **94**(1):131-146.

[2] Avouris NM, Tselios N and Tatakis EC. 2001. Development and evaluation of a computer-based laboratory teaching tool. *Computer Applications in Engineering Education* **9**(1):8-19.

[3] Stern F, Xing T, Yarbrough DB et al. 2006. Hands-On CFD educational interface for engineering for engineering courses and laboratories. *Journal of Engineering Education* **95**(1):63-183.

[4] Nguyen J and Paschal CB. 2002. Development of online instructional module and comparison to traditional teaching methods. *Journal of Engineering Education* **91**(3):275-283.

[5] Rosenberg J. 2000. Assessing online student learning via Dantes test.. *Virtual University Journal* **3**(1):1-7.

[6] Dutton J, Dutton M and Perry J. 2001. Do online students perform as well as lecture students? *Journal of Engineering Education* **90**(1):131-142.

[7] Hodge BK and Steele WG. 2002. A survey of computational paradigms in undergraduate mechanical engineering education. *Journal of Engineering Education* **91**(4):415-417.

[8] Christensen K, Rundus D, Fujinoki H, Davis D. 2002. A crash course for preparing students for a first course in computing: did it work? *Journal of Engineering Education* **91**(4):409-413.