REFLECTIONS ON A NOVEL APPROACH TO
TEACHING SOFTWARE DEVELOPMENT

Dr. D. B. Lowe and Mr. C. A. Scott

Faculty of Engineering
University of Technology, Sydney
PO Box 123 Broadway NSW 2007
Ph: +61 2 9514 2397 Fax: +61 2 9514 2435
email: {cascott,dbl}@ee.uts.edu.au

ABSTRACT: This paper describes the trial of a novel approach to the teaching of software development. The normal regime of lectures, tutorials, and laboratories has been replaced by a combination of intensive workshops, team based project work and self-paced learning. The conventional assessment by programming assignments and a final exam has been replaced by a combination of mastery and advanced level assessment: satisfactory completion of all mastery components guarantees a pass mark, completion of negotiated advanced tasks leads to higher grades. In this paper we describe the approach which we have taken, and the educational implications of the reduced face-to-face contact. We look at the mechanisms (including a significant level of electronic communication) which we used to ensure adequate progress and support. We also consider the advantages (such as flexibility and an increased sense of ownership by the students of their own learning) which our approach provided over the more traditional approaches to teaching software development. Overall, the new approach appears to meet our aims and is likely to replace the existing syllabus. From the first semester trial a number of improvements were identified and are currently being implemented.

INTRODUCTION

Software development and programming skills are taught as an integral component of the various undergraduate degree courses (Electrical Engineering, Computer Systems Engineering and Telecommunications Engineering) taught within the School of Electrical Engineering at the University of Technology, Sydney. This teaching occurs primarily within two subjects (Software Development 1 and 2). Over the last 3 years a series of problems with both the learning and teaching within these subjects has become increasingly apparent. During Autumn semester 1996 we trialed (using a single class of volunteer students) a radically different approach to the learning within Software Development 2. This trial incorporated a number of significant innovations aimed at addressing the major problems which were identified. In this paper we will look at the background which lead to the problems, the particular mechanisms we have used to address the problems, and the outcomes of the initial trial.

BACKGROUND

History

The authors belong to the Computer Systems Engineering (CSE) group within the School of Electrical Engineering at the University of Technology, Sydney. Over the last 15 years the software component of the degree courses taught by the School have evolved considerably. In 1982 the school switched its initial teaching language from Fortran to Pascal and then in 1985 introduced the first significant component of software engineering. At this stage the sole programming subject was a relatively traditional (in terms of content, approach and assessment) 3 hours per week, 16 teaching weeks, undergraduate subject. Over the ensuing decade progressively more material on software engineering was added: data flow diagrams and data dictionaries, pre- and post-conditions, coupling and cohesion, and abstract data types. Additionally the use of the programming language was expanded to cover data structures: queues, linked list, stacks etc. By the end of 1992, the material had expanded to become two subjects: “Fundamentals of Computing” and “Software Engineering 1” (each being 3 hours per week, 14 teaching weeks).

During 1993, the Software Thread Group undertook a review of the software subjects which resulted in the complete redesign of the subjects, which were renamed Software Development 1 (SD1) and Software Development 2 (SD2). This review had two primary objectives. The first was to investigate the relevance of the material which was being taught. The software development field has advanced significantly over the last decade, and although the software subjects had been incrementally updated, there were a number of changes which had not been incorporated. The most significant of these was the rapid rise of Object-Oriented (OO) programming languages and development methodologies. The subjects’ curricula was radically changed to bring it up to date in light of these changes.

The second objective was to investigate the educational effectiveness of the subject structure and teaching methods. During the review a number of significant problems were identified with the software development abilities of the undergraduates. These problems related predominantly to their depth of understanding, particularly of the concept of quality and the need for appropriate development methodologies. The result of this review was a major overhaul of the way in which the subjects were structured, taught and assessed. The major
innovation in this respect was the introduction of a series of small group-based workshops into the subjects. These workshops include peer reviews, critical analysis of concepts, software walkthroughs, and case studies. These were aimed at getting the students to focus on the need for, and the context of, appropriate software development methods (Biggs 1990) by encouraging the students to critically analyse the subject material, deconstruct case studies, and to analyse what both they and their peers are doing (Figure 1).

At the time of writing, the redeveloped forms of SD1 and SD2 have been running for approximately two and half years. Many of the problems which were identified in the 1993 review have shown significant improvement.

Problems with the Existing Approach

Despite the evident improvement provided by the changes made in 1993, a number of additional problems and observations have arisen. Firstly, and possibly most significantly, as our experience with different patterns for teaching software has increased, it has become increasingly apparent that lecture and tutorials may not be the most appropriate mechanisms to use in teaching programming skills. To a very large extent it would appear that software skills (and in particular, programming) is best learnt using a problem-based approach. The complexity of software systems means that a progressive approach is also appropriate.

A second problem with the current teaching approaches is that it does not lead to the students having a sense of ownership of their own learning. To a large extent this is because the specific programming skills are covered in a very mechanistic fashion. Finally, a third problem is that the current approach does not adequately lead to sense of the groupwork nature of software development.

A long term problem which we had yet to address effectively was not related to the teaching approaches, but rather with the assessment in the subject. We have always actively encouraged peer assistance, but have found that in many cases (especially in programming-based assessment tasks) this extends to outright copying and plagiarism and is often extremely difficult to detect or cope with. Indeed we have had a number of recent cases where students have submitted a working program with source code which could not be compiled. Often the assessment has been structured in a ways specifically designed to circumvent these problems (such as including an oral examination which the students were required to pass in order to gain a pass for the subject).

One additional issue which should be mentioned is that flexible learning patterns and approaches are becoming increasingly important, both within our degree program and on a wider scale. The existing approach offers some flexibility in assessment, but very little in study or learning approaches.

DESCRIPTION OF THE NEW APPROACH

As a mechanism for investigating possible solutions to some of the above problems a trial class was run during the Autumn semester 1996 which incorporated a large number of innovations. During the initial enrolments at the beginning of the semester we described our proposed approach and asked for volunteers. Nineteen students volunteered for the trial class. A check on these students past academic record revealed that the range of achievement and average marks were relatively similar to those of the student population overall.

Subject Structure

The subject had three major learning elements: intensive workshops, projects, and self-learning modules. This structure is shown in Figure 2. The workshops provide the core introductory material, and covers two 3 hour sessions early in the semester (to introduce the fundamentals and kick-start the learning process) and then three intensive whole-day (9:30am to 4:30pm) Saturday sessions to analyse in-depth aspects of the subject content.

The self-learning modules progressively introduce the students to specific programming skills. Three modules were constructed which contained a number of steps. Each of the steps contained a number of stages which were required to be completed in the sequence presented. Each stage was one of the following:

- **Discussion** - This provided background material on general concepts and placed the section into context.
- **Reading** - This gave pointers to additional sources of information which the students were asked to read in order to further develop their understanding.
- **Work** - The students were required to follow the steps given in order to develop their understanding.
- **Action** - This provided the students with the opportunity to develop and test their level of understanding. In particular, while completing this section they were required to
The students were required to submit various tasks, after which they were given information relating to how the task should have been completed so that they could evaluate their own progress.

- **Submission** - The students were required to submit the results of some of the activities. This was not for assessment, but rather so that their progress could be assessed in order to identify any possible problems with their learning and progress.

The project provided the mechanism for integrating the various concepts covered in the learning modules and workshops. The subject was heavily project based. At the beginning of the semester the students were broken into teams of three students (with one team of four). Each team was requested to identify a software project which was of direct interest to them. Assistance was provided to ensure that the suggested projects were of an appropriate scope. In having the students develop their own proposals, it was intended that the students would have a vested interest in and control over their project, and therefore have a higher degree of motivation.

Each group was then provided with a list of all the projects suggested and was required, after various negotiations, to contract with one group to develop their system. Thus each group was the client for their own system and the developer for some other groups system. This was intended to ensure that the students developed a solid understanding of the perspectives from both sides of the development process. Over the duration of the semester each client-developer pair of groups met numerous times in order to discuss requirements, development progress and (at the end of the semester) submission and acceptance of the completed project. Although the students were required to submit specifications and designs as intermediate steps during the course of the semester, like the self-learning modules these were not assessed, but rather only used to ensure progress and provide feedback to the students.

Figure 3 contains an extract from the material provided to the students.

In addition to the direct results of the development process, each group was required to keep a comprehensive log book which detailed all interaction with both their client (for whom they were developing) and their contractor (for whom they were the client), including times and content of meetings and other contacts (including any electronic contact) and those involved. The log book also was required to include reflections on the client/contractor relationship, especially with respect to any difficulties encountered.

### Subject Assessment

The assessment in the trial was intended to be both as flexible as possible, and to resolve some of the problems which existed with the conventional subjects (in particular, plagiarism and motivation). We have found that there is a very strong tendency (especially amongst the students who are struggling to develop an effective understanding) to develop a “mark-accumulation” attitude towards assessment tasks. It was not uncommon to overhear comments such as “Well, I got 46% for the first assignment, and 62% for the second assignment, that means I only need 44% in the final exam”. One other issue which we were careful to address was the balance which needed to be struck in assessment tasks between the process which students used and the resultant product which was developed. We would often find with software subjects that students could develop an excellent piece of software but have used a thoroughly inadequate process. Similarly we would occasionally have students who adopted an excellent process and carried it out very thoroughly but ended with a poor piece of software (often due either to poor coding skills or to time constraints). We needed to consider assessment tasks which would require that an adequate level of competence was reached in both these areas.

<table>
<thead>
<tr>
<th>Wk</th>
<th>Date</th>
<th>Teaching Component</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tue 5th Mar</td>
<td>Administration, form groups, subject overview, group process, intro to SW Eng., SASD and C</td>
<td>Project issued (5/3) Learning Contract issued (5/3)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>Project Topic due (12/3)</td>
</tr>
<tr>
<td>3</td>
<td>Tue 19th Mar</td>
<td>Negotiate project contracts, intro to C</td>
<td>Project Selection due (19/3)</td>
</tr>
<tr>
<td>4</td>
<td>Sat 30th Mar</td>
<td>SASD Workshop</td>
<td>LDC Module 1 due (4/4) Learning Cont. negotiated (4/4)</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>Project Spec. due (12/4)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Sat 27th Apr</td>
<td>Coding Workshop</td>
<td>LDC Module 2 due (26/4)</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>Project Design due (3/5)</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Sat 25th May</td>
<td>Testing &amp; Maintenance Workshop</td>
<td>LDC Module 3 due (24/4)</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>Project Complete due (7/6)</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>Learning Cont. due (14/6)</td>
</tr>
<tr>
<td>Exam</td>
<td></td>
<td></td>
<td>Oral and Coding Examinations</td>
</tr>
</tbody>
</table>

**Figure 2**

Figure 2 contains an extract from the material provided to the students.
The major project focuses on the analysis, design, implementation and testing of a major software system. You will be required to work in a team of three students (unless otherwise arranged with the facilitator). During the project you will fulfil two roles: client and contractor.

In the role of client you will come up with a problem which you wish to have solved and then contract out the development of a software system which satisfies your requirements to another SD2 group. You will need to liaise with the contractor and provide them with a problem statement, meet with them to discuss your requirements and answer any necessary questions, and then evaluate the system which they have developed.

In the role of contractor, you will need to contract with a client (i.e. another group of students) to develop their system. You will be given a problem statement from a client. You will then be expected to analyse the problem, identifying any issues which need to be resolved with the client, and develop a formal specification of the system to be developed. This specification will then be used as the basis for a detailed design and implementation of the system.

To address these problems we adopted a radically different assessment approach. The assessment in the trial was composed of two major elements: the mastery assessment and the advanced assessment. The mastery assessment components were not awarded any marks (so that the students could not adopt a ‘mark-accumulation’ attitude) but the students were required to achieve a satisfactory level of performance in every component. If they did so, then they were guaranteed a pass for the subject. The mastery assessment components included attendance at the workshop sessions, completion of the learning modules, and adequate performance in the major project and an oral and coding examination at the end of semester. In line with the policy of not assigning marks for the mastery components, the oral/coding examination was not marked. Rather the students were required to demonstrate an adequate level of competence both when asked to explain their understanding of certain concepts (such as ‘How do you think structured analysis contributes to the quality of software?’) and when asked to develop, code, compile, and execute a short (typically less than 10 lines) program (such as ‘Write a short program to print all prime numbers less than 100’).

The optional advanced assessment was the mechanism we provided to enable students to gain a mark higher than a pass. This element of the assessment nominally contained two components, though the students were given full scope to completely renegotiate any or all components of the advanced assessment. The two components which formed the standard advanced assessment was an evaluation of the students performance in the group project, and a learning contract. The project was assessed partially on the basis of the process which they adopted and partially on the resultant product, and was carried out using both peer assessment and assessment by the subject facilitator. The learning contract was left completely open and the students were required to negotiate both the content and the evaluation of the assessment task (though example possibilities were provided to give the students an idea of the scope and range of possible tasks).

We would have liked to investigate the option of allowing the students to select whether they wished to be awarded a mark of Satisfactory/Unsatisfactory (if they were not attempting the advanced assessment components) or the more conventional Fail/Pass/Credit/Distinction/HighDistinction + Mark (if they were attempting the advanced components). Unfortunately we were unable to do so within the constraints of the current subject result mechanisms - though we aim to continue to pursue this possibility.

Communication Mechanisms

The final major innovation which was incorporated in the subject was the mechanisms which we used to support communication between the students and the facilitator. The reduced face-to-face contact between staff and students and the loss of the weekly lectures (which is often the time teams use to communicate) was compensated for through the use of internet technology.

At the beginning of the semester the students were required to obtain an email account (and provide their team members and the subject facilitator with their email address) on the Universities central computing facilities. In addition they were encouraged to develop skills in accessing this account from either home or work. This process was facilitated through the loan of modems to those students requiring them and a reference guide covering the internet productivity tools available and a tutorial on their use. Assistance was given to those students who had difficulty with this process. During the semester the students were required to check their email on a regular basis (at least every two or three days).

Email was then used as a major mechanism for communication amongst group members, between groups, and between the students and the facilitator. For example, the learning modules required submission of certain work (not for assessment, rather so that progress could be assessed). These submissions typically did not coincide with formal class sessions and so occurred via email. Similarly, the students were actively encouraged to raise any questions or difficulties with the subject facilitator via email. The success of this is indicated by the volume of email traffic which was generated. Over the course of the semester the subject facilitator received or sent 453 email messages to the 19 students involved in the trial (an average of approximately 1.7 messages per student per week). Similar rates have been observed during the second semester of this trial.

REFLECTIONS

The new approach has a number of advantages over the more traditional approach to teaching software development. The students were provided with a great deal of flexibility with respect to their time, a feature particularly appreciated by part-
time students. Although submission dates for all components (such as the independent learning modules and project components) were provided these were treated as suggestions and were not enforced - though if they were not met the students were contacted in order to discuss their progress. The only fixed deadline was that all tasks had to be completed by the end of the semester. This flexibility applied equally to the less demanding lecture timetable and to the ability to work from home via the internet. The results from the student log books which the students were asked to keep indicates that the teams made extensive use of email; one student was even able to be a fully participating team member even though work commitments had taken him overseas. Observations indicated that the students’ motivation was greater, as they had more control over the learning activities and as a result seem to progress faster than students following the regular syllabus.

Prior learning is recognised via the self-learning modules. While these modules may take a learner some time to work through and understand, only the last exercise in the module is submitted as evidence of understanding. Those students with prior programming experience in the relevant programming language, or a good understanding of programming in general are able to skip directly to the last exercise and complete it without doing the whole module.

From an academic viewpoint, the new approach offered a number of advantages. The same flexibility to manage time that applied to the students also applied to the staff involved. Interacting with students via email is more efficient, allows a response at any time, and provides a written response. The assessment load is lowered as there is no grading required except at the end of semester. The grading that does have to be done is, however, more enjoyable than that for a traditional software assignment as every group project and advanced task is different. The self-learning modules are assessed throughout the semester but there is no grade attached; the student only has to provide a satisfactory attempt and feedback is purely formative. Plagiarism, a problem in software/programming subjects, is less of an issue as every project and learning contract is different and although the self-learning modules are the same, individual student understanding is ultimately assessed via an oral and practical exam which will identify those students who do not understand the work.

An important skill for software engineers is teamwork. Group assignments are therefore important as they provide an environment which requires teamwork. At the same time, the group assignments allow for more complex systems to be specified which are less likely to be successfully developed unless the students adopt a software engineering process. Problems often arise with group work due a lack of regular contact, different expectations amongst group members, and different participation levels of group members. Extensive use of electronic forms of communication (mainly email) contact can be easily maintained at the convenience of each group member. By having the group task assessed at the mastery level, the students aiming for high grades do not have to work harder to compensate for those students with lower expectations. Students aiming for higher grades and willing to devote the extra effort necessary do so through an optional advanced task which is individual, unless otherwise negotiated. If a student does not actively participate as a member of their team then they are less likely to have learnt the material required to pass the oral exam.

One of the major problems which was encountered was the difficulty which students appeared to have with the pacing of their own learning. Their was a surprising dichotomy in the feedback from the students at the end of the semester. Approximately half the students (especially those who were mature age part-time students) commented that they were very much in favour of the increased flexibility and considered that it has considerably enhanced their learning experience. The other half of the students took almost the opposite view and commented that although they liked the flexibility they felt that the lax control over such things as submission dates caused them to procrastinate to their detriment; a view reinforced by feedback from subject questionaries “I could only fault (the lecturer’s) leniency”. This group believed that tighter control over submission deadlines would have been an improvement. A possible compromise would to leave the flexibility but provide much closer monitoring of those students who fail to meet the suggested deadlines. Another possibility would be to get the students to select, at the beginning of the semester, the degree of enforcement of deadlines which they felt would benefit them.

Another modification which we feel would be beneficial would be to refine the approach taken to contracting between the ‘clients’ and ‘developers’ for the major project. We found that since little was at stake with respect to the relationship between groups, many of the students did not take this relationship as seriously as we would have liked. A possibility which might improve motivation would be to provide each group with an initial amount of ‘SD2 Money’ and have each group charge for their development services. At the end of semester each group would then have certain assets - the money they had ‘earned’ from their development and the value of the product (which the facilitator would ‘value’) which they had payed to have developed after negotiating suitable fees (and possibly contract clauses, etc.). These assets could be used to contribute to the subject assessment, or some form of auction could be held on which the students could use their gains (one suggestion was to auction personal tutoring sessions with the staff members). These ideas are still being refined.

It is also worthwhile pointing out that the Faculty of Engineering at UTS is currently undertaking a major restructuring of the undergraduate programs. A likely outcome of this restructuring is that the software subjects are likely to become common across the entire Faculty. This means that flexibility in content is likely to become as important as flexibility in structure. The approach we have set up would easily facilitate this. For example, one possibility may be to give the students much more responsibility for selecting not only their approaches to learning (as is the case with the trial we have discussed) but also for selecting the specific focus of their own learning - even to the extent of possibly allowing them to modify the programming language or development paradigm they choose to learn.

CONCLUSIONS

From the perspectives of the academics involved and the students that participated, the new approach to teaching
software development appears to be a success. Comments from the end of semester survey confirm the student view. On the positive side students liked ‘... (assessment) measuring the level of understanding of each student’; the ‘flexibility’ - a frequent comment, and ‘Learning at our own pace and using email for communication and (assignment) submission’. The students also provided feedback concerning ways to improve the subject, most notably to have more but shorter sessions and to better facilitate the advanced task (learning contract) negotiation. The first of these is easily resolved, the second most likely derives from the lack of familiarity of learning contracts and will, at least in part, be resolved by providing the students with more examples of suitable learning contracts.

The new approach has successfully met the goals intended for it. Academic workload is reduced and is less monotonous as much of the assessment is formative and it is only necessary to grade work for those students who have aimed higher and undertaken an advanced task. Plagiarism, while not stopped, is less of a problem as the oral exam readily identifies those students who do not have a satisfactory understanding of the subject content. Prior learning is recognised as the self-learning modules can be completed very quickly. Problems associated with group work are reduced as there is no pressure on grades and the promotion and subsequent use of electronic communication lowers the need for face to face meetings. Flexibility of learning and ownership of the group project increased student motivation leading to better learning outcomes.

ACKNOWLEDGMENTS

The authors wish to thank the UTS Vice Chancellors Development Fund. This fund provided financial support which significantly assisted with the electronic communication aspects of this project (including the purchase of a number of modems which were loaned to students for the duration of the trial).

BIBLIOGRAPHY

Biggs, J.B. (1989) 'Approaches to the enhancement of tertiary teaching' Higher Education Research and Teaching 8:7-25


Lowe, D.B. (1994) Software Development 1: Course Information and Notes, School of Electrical Engineering, University of Technology, Sydney


University of Technology, Sydney, School of Electrical Engineering, "School Handbook", 1993