AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE

20 Years of Software Engineering - and What Lies Ahead

May, 11-13, 1988

CANBERRA

PROCEEDINGS

Sponsored By:

The Australian Computer Society

Co-Sponsored By:

The Institution of Radio and Electronics Engineers Australia
The Institution of Engineers, Australia

EXPERIENCE WITH A PROJECT ORIENTED COURSE IN SOFTWARE ENGINEERING

Victor B. Ciesielski
Karl Reed
Jacob L. Cybulski

Department of Computer Science
Royal Melbourne Institute of Technology
GPO Box 2476V
Melbourne Vic. 3001

ABSTRACT

The advanced software engineering subject in the Department of Computer Science at RMIT provides students with an opportunity to experience the "real world" of software development. We simulate, as closely as possible, the situation where a software house is commissioned to develop software for a client. Students work in teams of 4-7 and the project involves most aspects of software product development, including contract negotiations, estimating and project planning and monitoring. Two major projects have been offered - a prototype electronic office system kernel and an expert system shell. We have found that the the subject provides a significant learning experience for the students, giving them the opportunity to integrate methods and tools presented in previous subjects. Students who complete the course show reasonable maturity with regard to software engineering. They have developed reasonable skills in working in project teams, in estimating the time required to develop high quality software, and in negotiating and dealing with clients. Post graduation feedback from employers and students has been excellent.

1. Introduction

Undergraduate courses in Computer Science are well known for the practical loads that they generate. In general, however, assignments can be divided into three categories:-

a) small individual assignments designed to test ingenuity, or familiarise the student with some new technique or language feature,

b) large individual projects,

c) team projects, such as the implementation of of a complier, or a modest commercial system.

Assignments are frequently artificially constructed to meet narrow pedagogic goals, and the results are not generally used by those other than the developers. However, in DP departments and software houses programmers are required to work in teams, to generate programs and documentation that will be used by others and to define and satisfy the needs of users, often working from poor specifications. Thus there can be a large gap between the educational experience of students and the situation in the workplace, as has been noted by other software engineering educators [6], [5] and [4].

The Advanced Software Engineering subject in the Department of Computer Science at RMIT tries to close this gap. In addition to presenting formal course work on the Software Engineering aspects of project management and system design, it also provides students with an opportunity to experience the "real world" of software development using Horning's "software hut " model [5]. The basic idea is to simulate, as closely as possible, the situation where a software house is commissioned to develop software for a client. Students work in teams of 4-7 and the project involves most aspects of software development including contract negotiations, project management, specification, estimating, scheduling, software work breakdown, progress reporting, coding, acceptance testing, renegotiation of work to be done as deadlines approach, dealing with difficult clients and clients who don't know what they want and sometimes change their minds (the lecturers).

2. The RMIT View

The Computer Science Degree at RMIT is, in effect, a double major in which students cover more than 56 semester-hours of computer science in the first two years. Students entering the final Software Engineering subject have taken courses in Data Structures, Database systems, File Design, Systems Analysis, Commercial Programming, Graphics, Compiler Writing, Operating Systems and Numerical Programming. They are competent in Fortran, Cobol and Pascal, and have experience on two different operating systems (UNIX and NOS/VE). They have worked on several large team projects. They will also have taken six semester hours in Software Engineering covering topics such as modularity, testing, metrics, algorithm re-use, macroprocessors and decision table translation. They have therefore comparable formal training to students in a graduate course in the U.S. [1]. Students taking the Advanced Software Engineering subject are thus well equipped and ready for some specially designed project that will consolidate their existing knowledge while providing them with an experience as close to "real life" as possible.

The RMIT view of Software Engineering training is that Software Engineering subjects should integrate existing skills and knowledge as well as emphasising and introducing selected topics.

The development of a series of courses in any discipline generally presupposes that course developers have some understanding of the philosophical basis of that discipline. The designers of the Software Engineering syllabi at RMIT had, as a primary goal, the training of graduates capable of producing high quality software. They were also influenced in part by Parnas' papers, [8] and [7], by work on the use of finite state machines as a design and implementation technique, and above all, the power of modular programming.

3. Description of the Projects

The projects prescribed for the Advanced Software Engineering subject run for about 3.5 months elapsed time, and typically consume 4 man-months in effort. The subject would account for about one quarter of a student's total load which would include other major project work.

The projects offered to the students reflect the different interests of the lecturers. Two different projects have been offered so far - an electronic office system and an expert system shell. The Prototype Electronic Offices System Kernel, Peosk [10], is a relatively large project that requires use of rapid prototyping techniques for successful completion. The Expert System Shell, on the other hand, allows emphasis to be placed upon formal, phased development, with emphasis on intermediate deliverables in the form of detailed specifications, tight schedules etc.

The projects have a number of characteristics which are very important from the point of view of the students' current experience and the sorts of things we want them to learn from doing the projects:

(1) The projects are much too big to be done by one person, so that students must develop co-operation and co-ordination skills.

(2) The projects are specified at a high level, functionally oriented fashion, so that the students are required to use the specification techniques learnt in previous courses to develop a detailed specification for coding.

(3) The projects emphasise the notion of product development for a client. The students are are required to exercise judgement in deciding which parts of the design need to be discussed with the client and the parts over which they have total control.

(4) The projects stress the importance of estimating programming effort. More credit is given to groups who are able to propose a project plan, monitor progress and renegotiate the project deliverables as necessary, than to groups who propose massive amounts of work and spend many hours of work in an attempt to complete the proposed work.

(5) The projects require students to put into practice software engineering methods and tools covered in pre-requisite subjects.

(6) The projects can be generally only be completed through the use of separable design[1] , and the recognition of "fundamental algorithms"[2]

3.1. Peosk - Electronic Office System Kernel

PEOSK consists of a multi-user electronic mail system, a file system, a commandless text formatter, an editor and a refer again or "tickler" system. A more complete description can be found in [10].

In addition, students must implement a fixed, "tiled" windowing system, and conform to unspecified "user-friendly" design objectives. The project draws on on operating system experience as well as general programming, and database skills.

3.2. Bullet - Expert System Shell

The initial design specification for the expert system shell project is shown in appendix II. The design calls for a relatively straight forward rule based shell where facts are represented as OBJECT-ATTRIBUTE-VALUE triples and forward chaining, backward chaining, inexact reasoning and explanation are supported; thus meeting the current market expectations on what an expert system shell is or

[1]The separate design postulate states that elements of a project can be designed in parallel under clearly defined circumstances.

[2] Fundamental algorithms are those which must be "discovered" before a system can be completed.

should be.

4. Experience with the Projects

Students were assigned to groups in a semi-random fashion by the lecturers who ensured that different groups had similar levels of programming skills (based on students' marks in prerequisite subjects), specialist expertise (e.g. not all students were familiar with Expert Systems) and time available for the project (determined by hours committed to the corequisite project subjects). This way students were forced to form professional working relationships rather than work with their friends. There has only been one bad failure with this approach. In one case the group dynamics did not work very well, the original project leader was unable to effectively manage the group and the group elected a de facto leader. The students attempted to hide this situation from the clients, only admitting that there were massive problems when it was obvious that they would not be able to deliver a working system. The students were so concerned about their marks that they invoked the students' rights office.

4.1. Contract Negotiations

The initial contract proposed by the clients is shown in Appendix I, and is naturally biased in favour of the clients. The students were required to negotiate a mutually acceptable contract with the clients. Initially they tended to be rather hesitant in suggesting changes, but after some experience they became quite competent, and in some cases quite aggressive, in debating various points in the contract.

Many of the clauses in the initial contract were drafted in a way that would make them unacceptable to most software houses, for example, clauses 5, 6, 8, 9, 10, 12, and 13. Most students immediately picked up the problems with clause 13. Many students thought that clauses 5 and 6 were reasonable, we think that they were unable to step outside of the student-lecturer roles and evaluate these clauses from the perspective of a software house. Many students accepted clauses 8 and 9, reflecting little experience with normal business practice. Almost all of the students thought that clause 12 was reasonable, although most software houses would prefer the client department rather than EDP to be the judge of satisfactory completion. Students were so confident of their ability to deliver on time that the lateness penalties did not concern them at all at the time of signing the contract.

Some students showed a reasonable understanding of the issues involved in requesting clauses asking for reasonable access to the consultants, changes to design specification requiring re-negotiation of contracts, and substantial bonuses for early completion.

Although they missed things in the initial contract that got them into trouble later most students felt that dealing with the contract was one of the most valuable experiences from the subject.

4.2. Role Playing During Negotiations

Staff attempt to play various roles during negotiations, simulating a real life situation. The success of these "war games" depends upon the maturity of the student negotiators. Problems have occurred in the past when the students have failed to pursue their own interest to the point where the client can capitulate with honour (See also McKeeman's comments[6]). Student teams will, on these occasions, grind to a halt because a specification cannot be reached or unrealistic functionality is attempted.

These difficulties can be overcome by providing "management consultants", independent members of staff who can advise the contractors on their negotiating strategy[3].

[3] We have tried to solve this problem by having the lecturer wear different hats. We had some success with this approach but some students became confused by the role playing under these circumstances.

4.3. Project Management

The students were advised to appoint one of the group members as the project leader. The project leader was to be responsible for all contact with the clients and with organising the activities of the group. We advised students to ensure that the project leader should do no programming. In quite a few cases project leaders could not resist the temptation to assign themselves programming tasks, to the detriment of the overall project. Invariably those groups that took our advice turned out better quality products.

The project leaders were required to prepare and present weekly reports stating progress in the last week and projected work for the next week. Any changes to the project plan were to be discussed with the client. It is interesting to note that very few of the project leaders were able to do this on a weekly basis, claiming to be "too busy doing the real work" to be able to prepare a report.

The students' estimating skills developed remarkably during the course of the project. Initially they promised everything, but towards the end of the project they were much more realistic about what they could accomplish.

4.4. Designing the System

In both projects the initial design specifications contained requirements which, if interpreted literally, would have resulted in cumbersome systems. The Expert System shell, for example, had a requirement to be "totally menu driven". The objective of these requirements was for the students to realise the problems of the requirement and suggest changes. Students tended, however, to assume that "the client knows what he wants" and implement these features anyway.

The screen layouts as specificed in the Request For Proposal for PEOSK are extremely poor. Students are supposed to recognize this, and to negotiate a new, improved design. There are, in addition, no precise proposals for the functions required. The students are supposed to develop and "sell" a functionality spec. to the clients. As already noted, PEOSK cannot be completed if the students fail to anticipate the client's requirements, and fail to undertake detailed design in parallel with negotiation. A separate, small negotiating team is essential.

4.5. Coding Testing and Documenting

Documentation of the programs and data structures tended to be very well done, However user manuals tended to be somewhat sketchy.

Students take some requirements literally, and fail to provide sufficient documentation to allow systems to be used. Password protection is a requirement on PEOSK accounts for example, and students quite correctly apply this to the system's administrator account. They invariably fail, however, to document the system administrator's password, for security reasons, one presumes!

Students tended to take on more work than they could really do, they thought that by working all night the night before the due date they could complete anything.

PEOSK project experience has been that it is essential to freeze the system specifications as early as possible to allow coding to begin. In addition, teams which have opted for rigid top-down design and implementation invariably complete only the user interface. In contrast, those who adopt a rapid prototyping path usually completed the whole system (See also Boehm's comments [2]).

PEOSK systems generally exhibit poor command error diagnosis and recovery.

5. Conclusions

Students begin to appreciate that there is much more to software development than programming and that significant time and effort must go into planning, monitoring and co-ordination activities.

They are able to apply the various techniques taught in class, and to appreciate their benefits and limitations.

We make a substantial effort to simulate a controlled, real-life situation that would be encountered by a software company, or an internal edp department. We believe that this approach is superior educationally, to externally originated and controlled projects.

Students learnt, some the hard way, that a system which doesn't work, but which would be terrific if it did work, is of no use to a client who has already made plans on the basis that the software would be available on a certain date.

The subject produces graduates who have some theoretical experience with estimating techniques and some practical experience with them. We feel this is very important in an industry where estimating is very much a seat-of-the-pants activity.

Graduates of the course are able to be effective earlier in their jobs as a result of the project. They have already learned many of the lessons and made most of the mistakes that beginning programmers make in their first jobs.

The students have an appreciation of the importance of contracts in software development and the kinds of negotiations that are needed to generate a contract that is satisfactory to two parties.

Post graduation feed back from employers and from students is excellent, and has encouraged us to retain the original format of our course.

ACKNOWLEDGEMENTS

We wish to acknowledge the enthusiastic participation of eight generations of CS387 students who have made this course possible. Special acknowledgement is due to the class of 1981, which produced working versions of PEOSK in Cobol on a Cyber under NOS, complete with multiuser electronic mail, and a tiled interface.

Mr. C. A. Adams, Mr. R. S. V. Pascoe and Mr. M. L. Ross have acted as consultants at various times, as have Messr's Koch, Delios and Allison of the Department's support staff. However, it is the Degree course structure as a whole which makes this subject possible, and the calibre of the teaching activity of those around us. We thank them all.

REFERENCES

1. M. A. Ardis, "The Evolution of Wang Institute's Master of Software Engineering Program", *IEEE Transactions on Software Engineering SE-13*, 11 (November 1987), 1149-1155.
2. B. W. Boehm, "An Experiment in Small-scale Application Software Engineering", *IEEE Transactions on Software Engineering SE-7*, 5 (September 1981), 482-493.
3. K. R. Cox and D. W. Walker, "Software Engineering Projects in a Computing Degree", *Proceedings of the First Pan Pacific Computer Conference*, Melbourne, Australia, September 1985, 399-409.
4. P. Freeman, "Essential elements of software engineering education revisited", *IEEE Transactions on Software Engineering SE-13*, 11 (November 1987), 1143-1148.
5. J. J. Horning and D. B. Wortman, "Software Hut: a Computer Program Engineering Project in the Form of a Game", *IEEE Transactions on Software Engineering SE-3*, 4 (July 1977), 325-330.
6. W. M. McKeeman, "Experience with a Software Engineering Project Course", *IEEE Transactions on Software Engineering SE-13*, 11 (November 1987), 1182-1192.

7. D. L. Parnas, ''On the Criteria to be used in Decomposing Systems into Modules'', *Communications of the ACM 15*, 12 (December 1972), 1053-1058.

8. D. L. Parnas, ''A Technique for Software Module Specification with Examples'', *Communications of the ACM 15*, 5 (May 1972), 330-336.

9. D. L. Parnas and D. P. Siewiorek, ''Use of the Concept of Transparency in the Design of Hierarchically Structured Systems'', *Communications of the ACM 18*, 7 (July 1975), 401-408.

10. D. E. Sivies, V. H. Obelt and K. Reed, ''The Implementation of a Prototype Office Automation System under Unix'', *Proceedings of the First Pan Pacific Computer Conference*, Melbourne, Australia, September 1985, 604-616.

11. D. M. Weiss, ''Teaching a Software Design Methodology'', *IEEE Transactions on Software Engineering SE-13*, 11 (November 1987), 1156-1163.