Engineering Realistic Software in a University Environment

M.J. Oudshoorn, A.L. Brown and K.J. Maciunas

Department of Computer Science
University of Adelaide
Adelaide SA 5005
Australia

Abstract. *This paper discusses the approach and philosophy of teaching software engineering within a university environment at the University of Adelaide. The discussion revolves around the project work set in 1995. The project work attempts to expose students to realistic problems which they are likely to encounter in the workforce after graduation. The focus of the course is on the software engineering process and the quality of the system produced.*

1. Software Engineering Education

The Department of Computer Science at the University of Adelaide believes that software engineering is an essential part of any graduate's education as it properly prepares them for the workforce and links together their experiences from previous subjects in the Computer Science curriculum. The subject, *Software Engineering and Project* was introduced into the undergraduate curriculum in 1992. The benefits of the project oriented course has been discussed in [Oudshoorn94].

The aim of the course has been to teach software engineering through a group-based project which exposes the students to the issues of dealing with programming-in-the-large and provides them with experience in the complete software lifecycle. The project also exposes students to human resource issues by virtue of its group nature. Tutorials and some 24 lectures covering topics such as group dynamics, project management, verification and validation, design, quality assurance, modern software engineering environments, safety-critical and real-time software engineering accompany the project. The lectures and tutorials essentially serve to support the project and to consider software engineering topics that may not be directly relevant to the project work. Readings are also set from the textbook and associated references [Booch94, Pressman92, Sommerville92] to encourage students to learn for themselves. Students are also encouraged to share information amongst themselves. The subject represents a 25% workload for a single semester (15 weeks) with students expected to devote 7 hours each week to the project. This results in projects of some 15,000 lines of code or 450 person-hours work.

We believe that the only way to learn software engineering is to do it. As a result, the projects selected in the past have been realistic and challenging. Our expectation is that students learn about, and apply, software engineering principles, rather than produce fully functional and marketable software, while working as a member of a group in an environment that, as closely as possible, models the workplace situation. We have the students focus on the software process and the quality of their software and associated documentation. We have chosen to have each of the students in the class participate in the same project rather than attempting to place the students in industry where they could work on a current software project. This allows us to fairly assess the students relative to each other and to monitor their progress during the semester with some ease. Finally, class sizes of 120 do not really make industry placement a feasible option.

Past projects have included a referencing system similar to the Endnote product [Niles91], software to simulate a personal organiser [Rupp92], and a simulation of a building management and evacuation system[1]. For each of these projects, students have been required to understand and analyse the problem before submitting a requirements analysis, a functional specification, demonstrate a prototype, produce a user manual and finally demonstrate a working final product. The format of the project has been extremely successful and we did not wish to depart from that format in 1995.

However, we have identified two deficiencies in the project work which we wanted to address in 1995: firstly, previous projects have not required students to think about, or work in, areas with significant safety-critical or real-time aspects; and secondly, students have not interfaced with external hardware. Given that these are growing areas for software developers, we felt it was desirable the students should graduate with at least some idea of what work in these areas entails. The only past project which vaguely addressed these concerns was the building management system but students were able to ignore any real-time issues through the use of a simulator and the safety-critical

[1] We wish to thank the Adelaide Metropolitan Fire Service and the South Australian Country Fire Service for helpful advice and insight.

aspects were viewed by the students as somewhat secondary. There was also no real building available to run their software on.

2. Changes to Course in 1995

To address these issues, we searched for a software system that was interesting, achievable in the course of a single semester, real-time, safety-critical and involved some kind of interface to outside hardware that was under computer control. We eventually decided on a railway network which could be simulated through a computer controllable model railway. The actual design and implementation of the model railway is discussed in [Oudshoorn96].

There were several reasons for choosing such a system for the students to work with. Access to TransAdelaide staff, both train drivers and controllers, provided us with a detailed knowledge of the workings of the actual rail line which we had chosen to model. Several students caught trains on this line to travel to and from University and could hence "see" the real system at work. Finally, the real-time and safety-critical aspects of the project are immediately obvious and cannot be overlooked.

The use of a railway model is not new and has been tried with mixed success elsewhere. Reports indicate that the major difficulty with such projects has been the expense of the trains as students continually crash them. We address this by insisting that students simulate the rail network and demonstrate a working system before gaining access to the model railway. This further reflects reality where a contractor would not be permitted access to TransAdelaide rolling stock in order to debug their software. The availability of computer controllable model railways has also made our job easier than it has been for others before us.

Another difficulty that we have experienced in this subject is the disruption caused by students withdrawing from the subject part way through the semester. This has the potential to force us to reallocate some groups and this is disruptive to the students involved and to the timetable the students must adhere to with respect to product deliverables. We hope that the interesting project will help reduce the number of students wishing to withdraw. However, we have also conducted a rudimentary personality test on each student and then balanced the randomly generated student groups so that there was at least one member from each of the four categories (leader, follower, creative thinker and worker). Past experience has shown that groups without leader personalities have difficulty as do groups with more than one strong leader.

3. Project Outline for 1995

In 1995 the University of Adelaide gave the Department of Computer Science a Quality Teaching Grant to construct a real-time system for projects in software engineering. With the grant, we have constructed a railway network that simulates the Adelaide to Belair commuter line. This line used to be a two track network, one track for each direction. However, one track was recently sold to Australian National Railways to be converted to standard gauge for the Brisbane-Perth freight line. The result was a single track for passenger traffic with four passing loops.

The topology of the constructed network corresponds closely to the actual topology of the rail line. The differences are deliberate and serve to make the model simpler and the problem easier for the students. Interference from other passenger services crossing the actual Adelaide-Belair line is not considered in the model, although a freight train which crosses the model rail line has been introduced so that students must consider external events to the passenger service. The freight train should have priority when crossing the commuter line as restarting a freight train is very expensive in terms of fuel. In consultation with the TransAdelaide timetabling staff we have been able to simulate the real rush-hour passenger loads presenting the students with a real-life problem some of them are actually part of each day. With a time scaling of 3 real seconds per minute of simulated time, the running times for train services correspond closely to the real timetabled running times.

The actual project that the students undertook in 1995 is the timetabling and running of three passenger trains to provide a service between Adelaide and Belair in the Adelaide Hills. Students were provided with arrival rates for passengers at various times during the day and had to ensure that no passenger waits more than 20 minutes for a train. The students' projects had to be flexible enough to handle delays at stations (wheelchair access to a train takes a considerable amount of time) and delays in tracks (speed restrictions due to works on the tracks). In addition, the software had to be able to handle difficulties such as passing loops, and so forth, becoming unavailable. Other constraints on their systems are relatively obvious: namely no collisions, and no derailments.

Students were told that their software was a prototype for an automated (driverless) train system. There are a number of mechanisms in the model rail system for students to locate trains are which could be realised with sensors in a real system. Students also had to provide an X windows [Nye92] interface for a controller to interact with their software. This interface was to show the location of trains and allow the controller to set track speed limits and restrictions. Finally, students had to simulate the rail network and show that their system was working before being permitted to execute it on the model railway.

The problem domain also provides some additional constraints on the software to be written by the students. For example, points can only be set when no train is directly over them otherwise a derailment will occur. Students were expected to analyse the problem domain with sufficient care and derive these, somewhat obvious, constraints for themselves.

It should be noted that the project is deliberately defined in such a way that the student groups are able to take the project and define specific project goals for themselves. This allows those groups whose members have a high degree of competence to evolve the project further than those groups whose members struggle or lack confidence. In addition, the vagueness of the original project outline provided to students forces them to seek clarification from the lecturers who are the clients in this project. The lecturing staff (there were three involved in the teaching of this subject in 1995) all have different ideas about the project and try to behave in a manner that emulates real clients as much as possible; that is, they present students with differing and sometimes conflicting views. Students gain valuable experience in having to decide for themselves what is important and essential versus what is peripheral to the project. They are also expected to budget their own time and resources and so another important consideration for the students is the amount of time it will take to develop a solution.

4. Project Environment

The student development environment is intended to be a combination of tools such as RCS [Tichy85] and the GNAT Ada95 compiler [Dewer95]. At the start of the project a complete set of Ada95 [Ada95] to X bindings was not available so a pre-existing set of bindings for Ada83 [Ada83] to the Athena widget set [Peterson92] was used. Unfortunately this required the students to write their graphical user interfaces using Meridian Ada. To allow the rest of their system to be written using GNAT Ada, a simple process control package was provided. This allowed child processes to be spawned and communicated with by exchanging strings. The package provides for non-blocking I/O so that individual processes need not be blocked if some part of the total system fails to respond. It also allows a simulation program to be written that mimics, in real time, the behaviour of the railway network. This permits students to do realistic testing of their software control system. A project group is not permitted access to the physical rail network until they can demonstrate convincingly, using their simulation, that their control system will work.

4.1. Software Deliverables

For both the prototype and final demonstration the students are required to submit an easy to install release of their software. This allows their software to be automatically relinked or recompiled by the customer, possibly on a different computer architecture. The real interface to the railway network is only supplied at this stage. One reason for this approach is to encourage the groups to maintain working versions of their system so that last-minute trivial changes do not destroy their only working system. The tendency to make last minute alterations to code before demonstrating it to staff has appeared to be irresistible to many students in the past. This practice has resulted in some groups not being able to demonstrate a working product as the last minute change has introduced serious bugs.

4.2 Documentation

In addition to the software that the students must submit, a requirements analysis, functional specification and a user manual must also be submitted. Hard deadlines are set for the delivery of the documentation and software, although students are permitted to review their documents and are invited to resubmit all documentation at the end of the course. This provides staff with the opportunity to comment on the documents and provides the students with the opportunity to revise their submissions.

A high level of quality is expected in the documentation and students are encouraged to take part in writing classes that are provided by the University. Staff have noted that students often tend to ignore the documentation and testing of their software in the subjects that they have studied earlier. As a result of this, a high percentage of the marks for the project are allocated to the documentation associated with it. This helps to correct the misunderstanding held by some students that employment as a programmer or software engineer means that the whole day is spent coding.

5. Students

This course is undertaken by a wide cross section of the student community, but the two dominant groups are: students studying Mathematics and Computer Science and those studying engineering, particularly electrical engineering and computer systems engineering. The “maths” students take this course in their final (third) year, whilst the engineers take it in their third or fourth year of study (of a four year degree). In addition to these two main classes of students, there are also a small number of students who are completing a one year postgraduate diploma for whom this course is mandatory. This latter category represents some students for whom the programming language Ada is a new language and this is the first course in which they have to write substantial amounts of code.

The students find this course considerably more challenging than other courses offered by the department although the intellectual content of the course is quite low. The unstructured nature of the project work forces students to adopt a far more methodical approach to time management than they otherwise would.

5.1 Perception prior to starting

The students are aware that this course has a large project associated with it before commencing the course. The actual project is not announced until after the course starts so the particular project chosen each year plays no part in selecting particular interest groups to the course.

Students are aware of what the previous years students did by way of project work and have “folklore” about how much work was involved. It is fair to state that they are prepared for a substantial amount of effort, but the vast majority of students falsely believe that this effort will be coding effort, rather than design, documentation, testing and coding.

Informal interviews with the students have revealed that they perceive the project to be fair and unbiased. No student felt that the choice of a “model trainset” favoured any particular category of student. In particular, the female students did not believe that the male students held some advantage which was a fear expressed by some colleagues prior to the start of the course.

5.2 Perceptions after finishing

Overall, most students believed the project to be “too big” and felt that they were poorly prepared for the task they were set. Virtually all the students believed, however, that if they were to tackle the project again, they would be able to deliver the system on time and with the desired functionality. Many of the project groups had deep concerns about some of the information they were given pertaining to the model train set - we made no attempt to filter all the information and provide them with only the “relevant” material. As a consequence of this, many projects foundered because the students failed to recognise the need for simple feedback in the system and instead tried to predict the exact position of the trains by using simple models of the train’s acceleration and velocity profiles. Interestingly enough, project groups with final year electrical engineering students also fell into this trap and produced systems which ignored the feedback the system provided, despite having undertaken a course on engineering feedback control systems.

Of the 18 project groups which undertook the course, only one group managed to implement a complete control system which was able to move multiple trains on the track. Most groups were able to control a single train in a limited way and some groups were unable to move any trains at all. This “failure to achieve” engendered a reasonable level of dissatisfaction amongst the students, with many commenting that they had worked very hard all semester and had nothing to show for it at the end. This feeling of failure was not justified as all groups had produced more than adequate user interfaces, and other support software. The major failing of these groups was simply an inability to build the system up from a small working system, rather than initially attempting to build a large system; this is despite the advice given to them.

6. Conclusions

The use of this style of large scale, group oriented project has been outlined in an earlier paper [Oudshoorn94] and the extension of this methodology to encompass safety critical and control systems has been described. The choice of a model railway as a target is a good one as the issues involved are easily appreciated by the students and intuitive. In addition to this, interviews with students has revealed that there is no perception of this project favouring a particular group of students. We believe the model rail system is a valuable tool and intend to make use of it again.

The benefits of this style of project are manifold, but the primary benefit to the student comes from the requirement to address the real-time nature of the project. The students become aware of issues they have never before had to consider varying from issues such as time order complexity in algorithms, communications latency and communications reliability. Whilst these problems might be hidden from the implementors of such a system in real life by the use of a real-time operating system, in the context of a student project the exposure of these issues has a high educative value.

Whilst it is too early to draw statistically meaningful comparisons with past projects in this course, there are some interesting observations which can be made. Firstly, the scope of the project was a little too great. The students were required to not only implement the control system as described but also a simulator with which to test and debug their control system. The simulator itself was a reasonably challenging project in itself and while it should have given the students great insight into the task at hand, it was the source of a great deal of frustration and loss of productive time for the students. In addition to this most of the simulators produced by the students did not exhibit appropriate real-time behaviour. This resulted in many groups being able to demonstrate their system operating correctly with their simulator, but when connected to the real system they failed to process events quickly enough (the usual failure mode was to process at about one third the required speed). Because of these problems, we propose to provide the students with an accurate software simulation of the system in future. This should effectively gain the students about a 20% improvement in their productivity (most groups dedicated at least one group member to this task). Secondly, the personality tests that were used to construct groups seems to have had little effect on the number of group arguments and disputes in comparison with previous years, but has perhaps resulted in fewer students withdrawing from the course and thus has minimised the disruption to the subject resulting from the need to reorganise the student groups.

One source of stress to the students undertaking the project is the provision of irrelevant information (some of which the students requested themselves!). We believe that this is necessary, as it models real life and intend to make no changes to the information provided to the students although some lecture material is planned which addresses this issue.

The major cause of failure to achieve by the students is a difficult one to address: many groups simply failed to listen to advice given. All the groups which were having problems sought help and in most cases were told explicitly what to do and why. In almost all cases, this advice was ignored by the students concerned and it was only at the final demonstration of their project work that they admitted that they had ignored the advice. In most cases, the students suggested that had they listened, things might have worked for them with some suggesting we give lectures on this subject! It may also be the case that the fairly “obvious” nature of the project work lulls them into thinking that they will be able to handle the work in their usual *ad hoc* way.

It is interesting to note that the most successful group was the last one to start coding having spent the majority of their time planning and evaluating their design. The groups which experienced difficulties in demonstrating their final software admitted that they could retrospectively see the value of planning and designing their solution carefully prior to coding, rather than as they coded. The admission that they at least learned from their mistakes is good as it is preferable that they make these discoveries within the University environment rather than in the workplace.

Whilst this project has been more difficult for the students to complete than previous projects, it has nevertheless been a very valuable learning experience to the students. Exit interviews with the students continue to show they regard this course as the most valuable one the department teaches. Feedback from employers has shown that they regard this project very highly and were impressed by the descriptions students gave of their work. The department has been contacted by employers directly, seeking to find out what we teach and how we teach in the subject *Software Engineering and Project.*

7. Acknowledgments

The authors wish to acknowledge the support of the University of Adelaide in providing a grant to enable the purchase of the train set and development of support programs. We wish to thank Tammy Braybrook and Karlie Hutchens for their programming efforts.

The authors are indebted to TransAdelaide staff for taking the time to discuss the real-life issues and problems involved in a rail system, and for helping us simulate those difficulties using our model.

References

[**Ada83**] *The Programming Language Ada*, Reference Manual, American Standards Institute Inc. ANSI/MIL-STD-1815A-1983, Springer Verlag, Lecture Notes in Computer Science, Volume 155, 1983.

[**Ada95**] *Ada95 Reference Manual*, International Standard ANSI/ISO/IEC-8652:1995, January 1995.

[**Booch94**] G. Booch, Object-Oriented Analysis and Design, second edition, Benjamin/Cummings, 1994

[**Dewer95**] R. Dewer and E. Schonberg, *"The GNAT Project: A GNU-Ada9X Compiler*", Free Software Foundation, 1995.

[**Niles91**] Niles and Associates, *Endnote Plus*, 1991.

[**Nye92**] A. Nye and T.O'Reilly. *The Definitive Guides to the X Window System*, Volume 4, X Toolkit Intrinsics Programming Model, OSF/Motif 1.1 Edition for X11 Release 5, O'Reilly and Associates, Addison Wesley, 1992.

[**Oudshoorn94**] M.J. Oudshoorn and K.J. Maciunas, *Experience with a Project-Based Approach to Teaching Software Engineering*, Proceedings of the Southeast Asian Regional Computer Confederation 5th Annual Working Conference on Software Engineering Education, Dunedin, New Zealand, IEEE Computer Society Press, November 1994, pp. 220-115.

[**Oudshoorn96**] M.J. Oudshoorn, A.L. Brown and K.J. Maciunas, *Simulating Real-Life Software Engineering Situations in the Classroom*, Software Engineering: Education and Practice, Dunedin, New Zealand, January 1996.

[**Peterson92**] C.D. Peterson*, Athena Widget Set - C Language Reference Manual*, X Window System, X11 Release 5, MIT X-Consortium, Cambridge, Massachusetts, 1992.

[**Pressman92**] R.S. Pressman, *Software Engineering. A Practitioners Approach*, third edition, McGraw Hill, 1992.

[**Rupp92**] Rupp Corporation, *Organizer Link II Software*, 1992.

[**Sommerville92**] I. Sommerville, *Software Engineering,* fourth edition, Addison Wesley, 1992.

[**Tichy85**] W. F. Tichy, RCS – A System for Version Control, Software Practice and Experience Volume 15, Number 7, pp 637–654, July 1985.