Two Complementary Course Sequences on the Design and Implementation of Software Products

JAMES E. BURNS AND EDWARD L. ROBERTSON

Abstract—For many students, the first chance to produce software as part of a team comes with the first work experience outside a university. The difficulties of working with others are compounded by the problems of working in a new environment and for a client with ambiguous and changing goals. Although it is difficult to approximate the "real-world" accurately in an academic course, we have implemented two full-year course sequences which apparently give our students some insight into the problems they will face when they leave the university. One course requires the development and implementation of a software product by a team of undergraduates and the other requires experienced graduate students to act as supervisors for the undergraduate projects. We describe the content and structure of these two sequences, emphasizing how they support and enhance each other. We believe other curricula would benefit from similar courses.

Index Terms—Computer Science education, software engineering, team projects.

I. Introduction

As pointed out cogently by Fairley [4], the needs and desires of industry are not being consistently satisfied by bachelor level computer science graduates. In particular, industry wants software engineers, while most undergraduate programs produce entry-level computer scientists. One reason

---

James E. Burns is now with the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

Edward L. Robertson is with the Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

that software engineering is not part of the core of more undergraduate programs is that it is difficult to teach. Students without professional programming experience seem unable to appreciate the importance of accurate and complete specifications, planning and scheduling, test plans, good documentation, *etc.* We encourage our students to obtain professional experience through internships, but these opportunities are not available to all. Experience with a realistic group software development project in an academic environment can be a valuable part of an undergraduate education in computer science. Unfortunately, it is difficult to find suitable projects and to administer a course with many such projects. We have developed a pair of courses at Indiana University which largely overcomes these difficulties. Although we make no claim of producing full-fledged software engineers with these courses, we do feel that we convey many of the skills deemed important by Mills [9].

Our key innovation is to use students in a graduate level software engineering course (the *supervisors course*) as supervisors for teams of students producing a software product as part of an undergraduate course (the *project course*). Our supervisors are analogous to teaching assistants in a course at Carnegie-Mellon University described by Kant [6]. However, our supervisors are also students who gain valuable experience pertinent to the graduate course from their supervisory duties.

Project courses have been discussed as tools for teaching software engineering at least since Horning and Wortman [5]. However, our projects are almost always drawn from the business community, local government agencies, and University departments. This presents some difficulties in administration of the course, but we feel that the problems are more than outweighed by the benefits of dealing with real clients, who often have ambiguous and changing needs. This inherently avoids some of the problems in teaching software engineering to undergraduates which were pointed out by Shaw [13]. For other experiences with courses using projects from real users, see Bolz and Jones[3], Lee and Frankel [7], Oman [10], Perkins and Beck [11], and Sanders [12].

The final distinction between our project course and most others with which we are familiar is that our course is taught as a two semester sequence. This allows the time to go through the full development cycle, including installation and evaluation, but omitting maintenance. It would be difficult to compress the course into a single semester while continuing to use real projects, given the lack of experience that the students in the project course have with software development by teams. A single semester course is feasible if the students have extensive professional experience (see McKeeman [8]).

The next sections describe our experiences with the graduate course (Section II) and project course (Section III) over the last four years. The final section gives our conclusions.

II. The Graduate Course Sequence

The Software Engineering Management course sequence has recently been added to the graduate curriculum after being taught as a special topic for several years. Our course, similar in objective to the Software Project Management course taught at the Wang Institute [1], is a graduate level seminar with a limited enrollment (maximum fifteen, typically twelve). Students are required to have taken the Information Systems sequence at Indiana or to have comparable (usually professional) project experience. Although we have had occasional disappointments, most of the students admitted to the course are highly motivated and do a good job.

A. Typical Seminar Content

The two semesters of the Software Engineering Management sequence are different in structure and format. The first semester (three credit hours) includes, in addition to supervisory duties, a traditional seminar with students taking turns leading the discussion. The second semester (one credit hour) usually consists solely of supervising the completion of projects.

Course materials for the first semester seminar are drawn from classical works and current papers pertinent to the course. The specific materials covered vary from semester to semester, but a typical reading list is given in Figure 1.

A syllabus similar to Figure 2 is distributed and students are assigned to lead specific class discussions. A substantial portion of a student's grade depends on the quality of the seminars they lead and their degree of participation in the seminar. The ordering of the material in the seminar is designed to complement the companion project course. *E.g.*, management issues are discussed before the software life cycle because the supervisors earliest responsibilities are concerned with selecting teams and projects. Note that some classes are devoted explicitly to discussion of project status, but every class meeting devotes at least a few minutes to the projects.

B. Outside Speakers

Several notations of *outside speaker* appear in the syllabus. We have been fortunate to have had four or more visitors from industry in each offering of the seminar. The visitors are usually from firms that hire our graduates

Bab	*Software Configuration Management.* Wayne A. Babich. Addison-Wesley, 1986.
Bau	*Software Engineering: An Advanced Course.* F.L. Bauer, Ed. Springer-Verlag, 1975.
Bro	*The Mythical Man-Month.* F.P. Brooks. Addison-Wesley, 1975.
Boe	*Software Engineering Economics.* B. Boehm. Prentice-Hall, 1981.
Dav	*Tools and Techniques for Structured Systems Analysis and Design.* W.S. Davis. Addison-Wesley, Reading, MA, 1983.
DeM	*Concise Notes on Software Engineering.* Tom DeMarco. Yourdon Press, 1979.
Fai	*Software Engineering Concepts.* Richard Fairley. McGraw-Hill, 1985.
Kin	*Current Practices in Software Development.* D. King. Yourdon Press, New York, 1984.
Kop	*Software Reliability.* H. Kopetz. Macmillan Press, 1979.
Met	*Managing a Programming Project.* P.W. Metzger. Prentice-Hall, Englewood Cliffs, NJ, 1973.
PW	*In Search of Excellence.* Peters and Waterman. Warner, 1982.
Shn	*Software Psychology.* B. Shneiderman. Winthrop, 1981.
Sho	*Software Engineering: Design/Reliability/Management.* M.L. Shooman. McGraw-Hill, 1983.
Som	*Software Engineering. Second Edition.* I. Sommerville. Addison-Wesley, Reading, MA, 1985.
Wei	*The Psychology of Computer Programming.* G.M. Weinberg. Van Nostrand Reinhold, 1971.

Fig. 1. Typical Reading List for Software Engineering Management.

and feel that such contact is mutually beneficial. Often, the visitors are former students, some of whom have taken one or both of the courses described here. Talking with working practitioners enlivens the seminar and emphasizes the importance of the material. The students have been very positive about this feature of the course, and we strongly encourage its use.

Week	Reading	Topics
1	Som 1. Fai 1.	Introduction to Software Engineering.
2	Som 10,11. Fai 2. Wei 4,5. Bro 3,4. Shn.	Managing people and projects.
3	Dav B,C,Q. Kin 5,6.	System Life Cycle. Interviewing techniques. Feasibility studies.
4	Som 2. Fai 4.	Requirements definition. *Outside speaker.*
5	Dav G. Fai 3. Sho 6.4. Boe.	Cost/benefit analysis.
6	Dav D,E,L. Som 4.1.1. Sho 2.5,6.5.	Data flow diagrams and data dictionaries. Scheduling
7	Som 3. Fai 5. Dav H,I,K.	Software design. Specification techniques: HIPO, pseudocode, Warnier-Orr.
8	Dav N. Som 9.	Interface design. *Outside speaker.*
9	Som 8–8.5. Sho 6.6.2. Bau 4.B.	Documentation. *Outside speaker.*
10		Discussion of Requirement Definition Documents. *Outside speaker.*
11	Som 2.6.1. Current papers.	Prototyping. Project review.
12	Som 3,4. Dav F,M.	Physical design, systems flowcharts, file design, modularizing.
13	Som 5,7. Sho 2,4.	Programming design; implementation issues. Testing, debugging and validation.
14	PW	Final discussion of projects.

Fig. 2. Typical Syllabus for Software Engineering Management

C. Supervision Responsibilities

In addition to the seminar, the students have substantial responsibilities as supervisors for the student teams in the project course. The supervisors act as advisors to the teams; since they have been through a similar experience, their advice is usually valuable and often (but not always) heeded.

Another responsibility of the supervisors is to monitor and report team activities. They provide early warning before milestones are missed so that remedial action can be taken.

To invest the supervisors with the appropriate level of authority, it is imperative that their position be made clear to the students in the project course. In a professional situation, a supervisor achieves authority with the power to hire or fire, promote or demote, *etc.* Since the only ultimate authority in an academic class rests with the final grade, the project course students might ignore the advice and demands of supervisors if there is no grade relevance. We emphasize the importance of the supervisors by making it clear that the supervisors have a significant input into final project grades and by requiring that all deliverables be turned in to the supervisors rather than the instructors. We have found that it is valuable to stress the authority of the supervisors and have devised mechanism (described below) to enhance this authority.

D. Evaluation of Supervisors

Because the graduate course enrollment is small, our grading policy has been informal. We evaluate the students both on their participation in the seminar portion of the course and on their performance as supervisors. (There are no formal examinations.) They are judged somewhat on the basis of the success of the project, but the variability in projects makes their behavior relative to their particular circumstances more important. We seek comments from students and clients in judging a supervisor's ability, and the supervisors are aware of this.

III. The Project Course Sequence

The Information Systems course sequence has two major components, one based on traditional class work (lectures, exams, homework, *etc.*) and the other on a major team project. The lecture component covers file media (corresponding to CS 5 of Curriculum 78 [2]) during the first semester and database systems (corresponding to CS 11) during the second. The team project component, which is the focus of this paper, serves the goals of CS 14 of Curriculum 78, although it is not taught as a separate course. Both semesters in the sequence carry four hours of credit (recently raised from three).

Information Systems is nominally a senior level course sequence, but a number of juniors and graduate students are usual. (A mixture of backgrounds and maturities is educationally beneficial.) Normally, two sections are offered with a maximum enrollment of 50 students each. We expect the students to have had exposure to several high level languages with substantial experience and skill in at least one. Although programming assignments based on the lecture material are often given, our concerned here is with the course project, which spans both semesters.

Projects from real users are not commonly assigned as part of an academic course because of difficulties in finding suitable projects, problems with organizing teams, difficulties in monitoring, and inadequacies of the academic reward mechanisms. Although we cannot claim to have a foolproof method for handling team projects, our techniques do provide decent feedback and monitoring with a tolerable administrative load.

A. Finding Projects

Although it is possible to use internally generated projects, there are many difficulties. If all teams are given the same project the adminstrative burden is reduced, but it is nearly impossible to assure that all teams work independently. If distinct projects are assigned to the teams, then much time will be spent in clarifying the design of each project. We recommend instead that projects be solicited from interested users in the community. We have had no difficulty in soliciting projects from large and small businesses, government agencies, charitable organizations, as well as university departments.

On the first day of class, the students are assigned to find and report on a possible project. (We usually have a number of leads which can be made available to students who request help.) We encourage students to look for projects which will be appropriate in scope and content. (Because the project is part of the Information Systems course, we require that projects include development of a system for maintaining and accessing a data base.) Students are given two weeks to find a project and write a two page description. Those who are unable to find a project are asked to report on the current status of projects from previous years or to write a short report on an article selected from a software engineering journal.

We have generally had good luck in finding enough suitable projects. In the 1985–1986 school year, over fifty projects were suggested by the ninety students in the two sections. Of these, ten were rejected immediately because they did not satisfy our requirement relative to Information Systems or because the client was not located in the vicinity of the University. (From past experience, we have found that the chances for a successful project are severely reduced if the entire team does not have ready access to the client.) For 90 students, 13 to 18 projects are desired, so having 40 allowed selection of the most appropriate projects.

Maintaining good relations with the clients is important. The client must be aware of the time and other resources that will be required for the project and of the possibility that the project will fail. It is important to emphasize that maintenance will not be provided. (If a client does have a problem after the end of the course, we try to find students who are willing and able, perhaps for a fee, to make changes or fix errors. However, we cannot guarantee this service.) We depend on the supervisors to make sure that clients are aware of the responsibilities and liabilities entailed.

B. Assigning Teams

A professional manager usually knows something about the strengths and weaknesses of his staff and how well individuals work together. Because of the time constraints of an academic course, team assignments must be made before the instructor has much information about the students as individuals. (Teams are usually assigned in the fourth week.) We evaluate programming skill from a pre-test given on the first day of class (the pre-test has no affect on grading) and writing skill from the project proposals and alternative writing assignments. This helps us to balance skills across the different teams, but it doesn't help with personality incompatibility problems.

Classes are divided into groups of approximately fifteen students, with approximately the same number of skilled programmers and writers in each, as determined by the pre-test and writing samples. Supervisors are to become familiar with all the individuals in one of the groups. They then assist in choosing teams of four to seven students on the basis of personalities as well as skills. A team with three students is minimal if the students are to learn the appropriate lessons about working in groups, but larger teams are preferred because of attrition. Seven is the maximum size since a larger group would probably need some internal management structure. If a very large project is undertaken, it would be best to partition it into smaller subprojects and use the supervisors to coordinate the efforts of the teams.

C. Monitoring Progress

Because it is easy for a project to get into major trouble before the instructor is aware that anything is wrong, we depend on the supervisors to keep in close contact with their teams. The supervisors are expected to help the teams overcome their problems and to keep the instructors informed of project status.

D. Milestones

As a standard against which to measure progress, we use a set of milestones with fixed due dates. Of course, individual projects can vary somewhat from the ideal schedule, so the supervisors are allowed some variance.

The milestones shown in Figure 3 correspond to our own version of the software life cycle, tailored to the needs of the course. An extensive handout describes what is required for each milestone. Here, we give only a brief description of the milestones and explain how they interact with the administration of the course.

The Project Proposal is a very brief document used to make a preliminary judgement on the appropriateness of a suggested project. The Feasibility Study gives a more in-depth look at the need for the project and the resources required. One purpose of the Feasibility Study is to assess the com-

Milestone	Semester	Week Due	Description
0	First	2	Project Proposal.
1	First	4	Feasibility Study.
2	First	8	Requirements Definition and Preliminary Project Plan.
3	First	14	Logical Design and Test Plan.
4	Second	3	Physical Design.
5	Second	7	Implementation.
6	Second	10	Testing Completed.
7	Second	13	Installation.
8	Second	14	Post Mortem.

Fig. 3. Project Milestones.

mitment of the client toward the project, since experience has shown that this is a strong indicator of project success. After considering the Feasibility Studies carefully, the supervisors in consultation with the instructors decide which projects should continue. Then teams are selected and assigned to projects.

The Requirements Definition is the first substantial piece of documentation produced by the teams. It must be approved by the client before the project can continue. We also require a Preliminary Project Plan, which include a schedule for meeting the future milestones. Although the students are unlikely to produce an accurate schedule, we feel they benefit from their errors in planning.

The final milestone of the first semester is the Logical Design, which is supposed to specify precisely what the product will do to satisfy the Requirements Definition, but not say how the product will be built. We encourage supervisors to require early drafts of designs and to give substantial criticism. Having a predefined point during the project when the Logical Design must be completed has presented minimal difficulties. The Logical Design document and an in-class presentation of the design are the primary basis for the project grade in the first semester.

The deadline for the Physical Design milestone comes early in the second semester. (Preliminary work is done on the Physical Design in the first semester.) This detailed design should give all the design decisions necessary for project implementation.

The Implementation is deemed complete when 100% of the coding has been compiled without syntax errors. The Testing milestone is met when the error-free software has been demonstrated to the supervisor. Of all the milestones, these are the most fluid. Supervisors have great discretion in adjusting these due dates for the individual projects. The objective is to deliver the final product on time; the individual milestones merely assist in this process.

Installation is complete when the final software and all its documentation (user's guide, training manual, programmer's guide, *etc.*) has been delivered to the client and the client has received appropriate training. Verification of installation is the responsibility of the supervisors, but the instructors try to attend an on-site demonstration of the final product. The instructors also try to contact the client during the final two weeks between Installation and the end of the course to determine client satisfaction directly.

The final milestone, Post Mortem, consists of an in-class presentation and a brief document describing the team's experiences during the project. Its purpose is to contemplate what happened during the project, what went wrong, and how problems were overcome. We encourage the students to suggest ways that problems could be avoided in the future. The in-class presentations are important so that the class as a whole can benefit from the experiences of all the teams.

There is an inevitable and unfortunate conflict between the success of the team project and the value of the learning experience. A team is likely to do better if members can specialize in what they do best. However, this would deprive some students of participating in coding and debugging, while others would never do documentation. The milestone approach helps mitigate this problem, since it forces the entire team to focus on one aspect or another at different times.

E. Grading

It is always difficult to assign individual credit for a group project. Our solution is to grade the projects first, then to decide on grades for team members. Projects grades are based primarily on the written documentation provided with the delivered product, class presentations, product demonstrations, client feedback, and supervisor recommendations. Since the project grade is typically 40% of the course grade, the students are highly motivated to do well.

Students are advised early that they will be evaluated by their fellow team members. Each team member provides a written report explaining what was done (or not done) by each member and a numeric evaluation in which a fixed number of points are distributed among the team. This information and supervisor evaluations are carefully considered in assigning grades for individuals. The most common occurrence is that all students in a team receive the same grade, but it is not unusual for one student to receive an 'A' or a 'C' while the other members receive 'B's.

Occasionally, one or two students in a class seem to be unable or unwilling to work on a team project. In the past, we have used the threat of a poor grade as a means to encourage participation. Unfortunately, it has still been necessary to assign a grade of 'F' occasionally. Although this might punish the offending student sufficiently, it does not compensate for the additional difficulties encountered by the rest of the team. A more successful strategy has been used in recent years. We give the supervisor the option of *firing* the student from the team. The fired student could appeal to another team for admittance; if this plea fails, the student would be assigned an individual project with a maximum grade of 'B.' We have not yet needed to use the firing mechanism and have had no grades of 'F' on projects since it was in place. We are not sure whether the threat of firing or luck was responsible for the improved situation.

F. Results from the 1985–1986 Offering

After evaluation of the feasibility studies by the instructors and supervisors, we selected twenty projects (out of forty) to go forward. In retrospect, we should have selected fewer projects, for the average team size soon dropped below five when several students dropped late in the course.

Two of the twenty projects were cancelled at the end of the first semester. One client was unable to provide the necessary access to a time-shared computer because of security concerns at the corporate (non-local) level. In the other case, access to the client's microcomputer system was inadequate for development, and no alternate system was available.

Of the remaining eighteen projects, all but four appear to be completely successful. Two of the unsuccessful projects were not completed on time; grades were withheld until completion. The other two unsuccessful projects were inadequate for the intended purposes. In both cases, the projects suffered because of unavailability of the client.

Most of the projects were well received by the clients. The typical project was developed on a microcomputer using commercially available database tools. While these types of projects are not good models for large scale product development, they do provide valid experience in working with a team to develop software and in interacting with a real client.

IV. Conclusions

Our confidence in the validity of our approach is high because of the positive feedback we have had from the employers who have hired veterans of both the project course and the graduate course. There is keen competition for admittance to the project course, at least partly because student perceive that it is an asset when looking for a job.

While the instructors certainly benefit from the supervision performed by the students in the graduate course, the students can benefit even more. We have had many unsolicited comments indicating that the Software Engineering Management sequence has been very valuable to our graduates. The supervisory part of the course is essential in conveying the relevance of the material discussed in the seminar.

The use of students from the graduate course as supervisors in the project course augments and enrichs both courses and makes it feasible to offer real project experience in a large, undergraduate course. We would be interested in hearing from others who have had experiences with similar courses or who are contemplating offering such courses.

Acknowledgment

We wish to thank all of the students who have participated in the Information Systems and Software Engineering Management course sequences at Indiana University over the last four years. Their criticism and participation helped to develop a much better pair of courses. We also want to thank the individuals from outside Indiana University for participating in our graduate seminar and their employers for allowing them to come. A special thanks goes to Sid Kitchel, who greatly assisted with the administration of both course sequence in the 1985–1986 school year. Finally, we wish to thank the editor and the referees for their comments.

References

[1] M. Ardis, J. Bouhana, R. Fairley, S. Gerhart, N. Martin, and W. McKeeman, "Core course documentation: Master's degree program in software engineering," Technical Report TR-85-17, School of Information Technology, Wang Institute of Graduate Studies, Sep. 1985, 39 pp.

[2] R.H. Austing, B.H. Barnes, D.T. Bonnette, G.L. Engel, and G. Stokes, (eds.), "Curriculum 78: Recommendations for the undergraduate program in computer science—A report of the ACM Curriculum Committee on Computer Science," *Comm. ACM*, vol. 22, no. 3, pp. 147–165, Mar. 1979.

[3] R.E. Bolz and L.G. Jones, "A realistic, two-course sequence in large scale software engineering," *SIGCSE Bulletin*, vol. 15, no. 1, pp. 21–24, Feb. 1983.

[4] R. Fairley, "The role of academe in software engineering education," in *Proc. 1986 ACM Fourteenth Annual Computer Science Conference*, Cincinnati, Ohio, pp. 39–52, Feb. 1986. (Also available as Technical Report TR-85-19, School of Information Technology, Wang Institute of Graduate Studies, Oct. 1985.)

[5] J.J. Horning and D.B. Wortman, "Software hut: A computer program engineering project in the form of a game," *IEEE Trans. Software Eng.*, vol. SE-3, no. 4, pp. 325–330, July 1977.

[6] E. Kant, "A semester course in software engineering," *ACM Software Engineering Notes*, vol. 6, no. 4, pp. 52–76, Aug. 1981.

[7] K.Y. Lee and E.C. Frankel, "Real-life software projects as software engineering laboratory exercises," *ACM Software Engineering Notes*, vol. 8, no. 3, pp. 39–43, July 1983.

[8] W.M. McKeeman, "Experience with a software engineering project course," Technical Report TR-86-01, School of Information Technology, Wang Institute of Graduate Studies, Jan. 1986, 17 pp.

[9] H.D. Mills, "Software engineering education," in H.D. Mills (ed.), *Software Productivity*, Boston, MA: Little, Brown and Company, 1983, pp. 251–264.

[10] P.W. Oman, Jr., "Software engineering practicums: A case study of a senior capstone sequence," *SIGCSE Bulletin*, vol. 18, no. 2, pp. 53–57, June 1986.

[11] T.E. Perkins and L.L. Beck, "A project-oriented undergraduate course sequence in software engineering," *SIGCSE Bulletin*, vol. 12, no. 1, pp. 32–39, Feb. 1980.

[12] D. Sanders, "Managing and evaluating students in a directed project course," *SIGCSE Bulletin*, vol. 16, no. 1, pp. 15–25, Feb. 1984.

[13] M. Shaw, "Making software engineering issues real to undergraduates," in Wasserman, A.I., and Freeman, P. (eds.), *Software Engineering Education: Needs and Objectives*. New York: Springer-Verlag, 1976, pp. 104–107.