

Evaluating A Software Engineering Project Course Model Based On Studio Presentations

John C. Knight and Thomas B. Horton

Department of Computer Science, University of Virginia, Charlottesville, VA 22904

horton@cs.virginia.edu knight@cs.virginia.edu

Abstract --We present a model for a software engineering project course that has proven successful as the capstone of our required sequence of software courses in our computer science major. It differs from many similar courses because all teams work on the same project, defined by the instructor (not projects for "real" customers). The course's project centers on weekly "studio presentations" during which student teams present preliminary results and receive immediate feedback. This model allows us to efficiently offer a project course for over a hundred students. The nature of the project itself supports other course goals. Assessment of our program and the course indicates that the course successfully fulfills its role within our degree program.

Index Terms - laboratories, project courses, software engineering education, studio presentations

INTRODUCTION

Teaching software engineering is a complex and difficult task. There is a great deal of material that has to be conveyed to the student in a wide variety of subfields. The practicing software engineer needs to understand the many approaches to the phases of the software lifecycle, to project management, and to the software process in order to be successful in building modern software systems.

Yet knowledge acquired in a typical classroom setting is not sufficient. It is also necessary for the student to gain skills and insights that are acquired typically through experience. Indeed, many aspects of the discipline require expert judgment that can be learned best only by interaction between a student and teacher that is of the *master/apprentice* form. In design, for example, the difference between a good software design and a bad one is often subtle and very application specific. Only after having his or her mistakes with design pointed out by an expert does the student really gain the right type of judgement. The problem with the master/apprentice approach, however, is seen immediately when there is only one master but dozens of apprentices as is typically the case in a modern university.

The need for the student to acquire experience in software development is well-known by educators, and the approach that is usually taken is to form a semester course on software engineering around a project of some sort, frequently a project that is undertaken by students working in groups. This approach is fairly successful but all too often in software

engineering, essential experience is finally acquired in the workplace. The end result is systems that either never become operational or lack important qualities of dependability or maintainability.

We have developed a new approach to teaching the "experience" element of software engineering. The core of the approach is a teaching paradigm called *studio laboratories* that attempts to provide as much as possible of the master/apprentice relationship in a context where the apprentices vastly outnumber the masters. The studio laboratories are supplemented by a number of other techniques that, when taken together, create a novel teaching environment.

The model for this course differs from many other software engineering project courses. No attempt is made to find a "real" customer or simulate such a stakeholder. Rather, students are organized into teams and all teams work on the same project. In addition, the project only varies slightly from year to year. These characteristics have advantages and disadvantages, but they provide a valuable educational experience for large numbers of students in our program.

This course was developed and delivered by this paper's first author at the University of Virginia. In 1999, the second author co-taught this course while visiting on sabbatical. In 1999 and 2000, he adapted the course for delivery under different conditions at Florida Atlantic University.

The course concepts that are presently used are described in the next section. We present the project scenario for the course and an outline of its form and content in section 3. In section 4 we review the way in which the course goals have been met and discuss our experience. Finally we present our conclusions in section 5.

COURSE CONCEPTS

Organization

The course is organized around a semester-long project in which a significant software system has to be developed. The overall structure of this course loosely matches a typical development process. The goal is to provide as much experience of typical development issues as possible without imposing excessive time demands on the students. A sequence of steps is undertaken by the students where each step encompasses as much process realism as possible. This

approach allows a wide range of experiences in the limited time available.

The major components of the course are weekly lectures, laboratories that take place somewhat less frequently than once per week in which students work in groups, and regular individual assignments. Most of the laboratories use the technique that we have developed called *studio laboratories* and the remainder are more typical closed laboratories.

Student Groups

Being able to work in a group is a critical skill that software engineers need to have and we organize the students in this class into groups of four. The students are required to view their groups as small software companies. They name the companies, set up Web sites for the companies, and are expected to act as professional representatives of professional organizations. Our experience is that students typically take great pride in their "companies".

Being able to work in a larger organization is a different yet equally important skill, and to convey the important elements of this skill two student groups are paired to form one team. The semester project is undertaken by a team not by a single group, and so in order to get a complete working project the two student groups must work together successfully as a team. The primary pedagogical value of the team structure is to force precise inter-group cooperation. Communication between the groups in a team has to be undertaken in a professional manner including the exchange of documents, structured and restricted interaction between identified contact points in each group, and detailed process activities.

Closed Laboratories

Closed laboratories are the type of laboratory that is common in Physics and Chemistry. A closed laboratory requires a student to undertake some prescribed activities at some prescribed time and at a prescribed location. The intent is to permit students to undertake carefully designed exercises in an environment where faculty and teaching assistants can be available for assistance. This approach is very effective in teaching topics such as elementary programming where simple syntax or interface difficulties would slow a student considerably if he or she were working by themselves.

In the course reported here, there is no need for a highly structured and detailed laboratory schedule. Rather, we use closed laboratories for learning higher-level conceptual elements of software engineering. For example, one of the closed laboratories focuses on learning the concepts of risk identification and reduction by prototyping. Part of the laboratory requires students to develop small, throw-away prototypes to evaluate prescribed risks. A second part engages all the students in the class in a discussion of the risks that they face in the project. Another closed laboratory requires students to integrate the two project parts from the groups in each team. They have to do this for the first time in the laboratory itself so there is no chance to debug the whole system before the laboratory. The laboratory requires the

students to prepare a detailed integration and test plan. The intent, of course, is to expose students to the very complex issues that arise in planning and undertaking system integration.

Studio Laboratories

The notion of a studio laboratory is adapted from a practice that is sometimes used in schools of architecture where students prepare and display the results of a design exercise. The artifact is examined by an instructor and discussed with the student and perhaps other members of the class.

Our version of this approach is to have student groups make presentations to the entire class about artifacts that they have created during software development, about management issues, or about experiences during development. Class members and faculty then ask questions and discuss the details of the presentation. A single studio laboratory lasts for about seventy five minutes and addresses just a single topic such as a specification, a design, or a management plan. During any given laboratory there is sufficient time for about four group presentations. All groups in the class have to prepare presentations and the choice of which groups actually present is random.

Using the studio approach, all the students in the class see several different approaches to the same software engineering problem in a single laboratory period and hear discussion of the approaches. The effect is to reveal the majority of the mistakes made by the class without having to review the work of the entire class. But since the entire class hears the presentations and discussion, most of the time students see their own misunderstandings and mistakes as part of the presentation of at least one of the groups. They can then benefit from the corrections discussed.

Deliverables to be graded are due a week after the laboratory in which they are discussed so that good ideas discussed in the laboratory can be incorporated into deliverables by all groups. In principle, therefore, students can always get a high grade on their laboratory work.

We conclude this section by noting how our approach to a software project course compares to other approaches that have been used elsewhere.

In our course, there is a single well-defined problem for all the teams. While students do carry out requirements-related activities by interacting with the instructional staff, there is less emphasis and energy spent on requirements than would be for student-projects based on "real" customer projects. There are of course advantages and disadvantages to this approach. Students do participate as much in user-centered development activities. On the other hand, by working on a common project and presenting their results in studio laboratories, students see and evaluate a large number of possible solutions for the problems they are working on each week. Having a common project allows the instructor to manage a large number of students in a project course; CS340 typically has an enrollment of 120 students, staffed by one faculty member and about six teaching assistants. The division of the project into

components that two halves of the team build together allows the students to develop process, integration and teamwork skills more effectively than a model where individual teams work on different projects. We believe the advantages outlined here outweigh the need to try to give students a more “realistic” experience with real customers in a one-semester course.

PROJECT SCENARIO

The semester project is the development of a control system for a mobile robot (see Figure 1). Various scenarios have been used during the development of the course, and the current one is that the robot is one of the NASA rovers on Mars. The project requires the development of software for the robot itself to control all aspects of the device, software to provide an interface to be used by a human controller on a desktop machine, and software at both ends to support communication. Remote operation of this form means that real-time control is not possible and nor is real-time image display. Thus, the command interface has to provide facilities for preparing and transmitting sequences of commands for later execution and for displaying various combinations of still images. The onboard system has to provide facilities to protect the robot in the event that it loses communications or approaches an obstacle in a dangerous way. Finally, the onboard system has to be equipped with a system to support remote debugging from a user interface displayed on the machine that runs the command interface.

The robot used in the course is a heavily modified ER1 from Evolution Robotics. The ER1 is a small mobile platform with two independently driven wheels and a castor. The body of the ER1 is a framework of aluminum struts, and three infrared proximity sensors and two cameras are mounted on the body. One of the cameras is fixed in orientation and the other is mounted on a pan/tilt head that is accessible from a software interface. Both cameras feed digital images, and USB interfaces provide access to the cameras and the ER1’s microprocessor system that controls the wheels and the sensors.

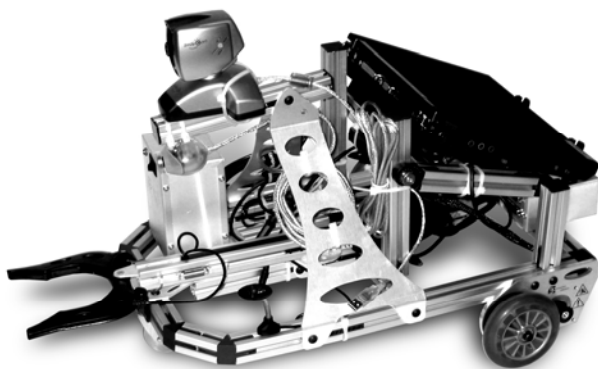


FIGURE 1
ER1 MOBILE ROBOT

For purposes of this project, the software for the control system is broken into three components:

- The operator command interface. This software has to provide a graphic interface for the operator that permits the operator to manipulate the robot and its cameras conveniently and easily.
- On-board robot control. This software has to provide local robot control, accept commands from the operator and effect them, deal with the acquisition of telemetry, and implement the autonomous and debugging functions.
- Communication link. This software has to provide reliable two-way communication between the other two software components.

PROJECT OUTLINE

In this section we outline the major elements of the project and the order in which they occur. Each element is associated with a laboratory. There are a number of dependencies between project elements that are intentional and have specific pedagogical purposes. The major project elements are:

- *Risk-reduction prototyping.* Students are introduced to the notion of development risk and risk-reduction techniques. Risk reduction is emphasized throughout the course, and student groups are required to maintain lists of risks as the project proceeds.
- *Requirements elicitation.* The staff associated with the course act as the “customer” and provide requirements information. As part of the associated laboratory, a mock up of the command and debugging interfaces have to be developed in Visual Basic for evaluation by the “customer”.
- *Specification.* A specification for the system is developed but in three parts; one of the groups in each team develops the specification for the command interface software and the other group develops the specification for the onboard software. The groups work together to develop the specification of the communications system. An important aspect of the project takes place at this point—the groups exchange specifications and the rest of the development proceeds with each group implementing the specification they received. In this way, each group is able to experience the issues that arise when implementing a specification written by a different organization as well as those that arise when another organization implements a specification that they have written.
- *Implementation planning.* Once a specification has been completed, the next activity is to plan the implementation. In this activity, groups have to prepare Gantt and PERT charts, cost estimates, and work-breakdown structures. As part of this project element, students are introduced to the problem of having to work with limited amounts of critical hardware, such as robots. To allow the preparation of the project software each group has to build a simulation tool for the other half of the project. This allows testing without requiring either the other team’s software or a robot to be available.

- *System design.* In this project element, the design of the system is prepared. This is a project element in which the studio laboratory format has proven especially useful. Object-oriented design is difficult to teach and difficult to learn. By having students attempt the system design and then present them, many mistakes and misconceptions about object-oriented design are revealed.
- *End-to-end prototype.* This project element addresses the myriad of risks associated with bringing all the parts together and it gives students experience with system integration. Students are required to prepare an integration plan and only work together in the way documented in the plan.
- *Enhanced prototype.* The next project element is a prototype that provides much more functionality than the end-to-end prototype developed earlier. This is an implementation of and exposure to the Spiral development process.
- *Inspection.* There is not sufficient time in this course to cover everything, and one of the topics that is not covered thoroughly is verification. The general topic of verification is important and a very cost effective verification technique is inspection. In this project element, the source code developed is inspected using a rigorous inspection technique. This is another example of the power of studio laboratories. In the laboratory, the group presentations have to document the inspection artifacts, such as checklists, and also their inspection experience, such as defects found. It is almost always the case that students actually do find defects that were previously unknown to them. Discussing this in front of the staff and students in the class is a powerful way to present the effectiveness of inspections.
- *Delivery.* The final project element is delivery in which the complete system is demonstrated. Throughout the project, students are required to pay special attention to group management. Each group creates a Web site for their company, and, as the project proceeds, more and more management material is required to be maintained on the Web sites. In addition, each laboratory report has to contain a management report that documents group meetings, activities, and so on.

ASSESSMENT AND EVALUATION

This course is the subject of on-going assessment and evaluation. The course is central for four of our BSCS degree’s program outcomes, defined as part of our ABET accreditation activities. These outcomes state that our graduates will:

- *Outcome 4:* Have applied knowledge of areas of computing to create solutions to challenging problems, including specifying, designing, implementing and validating solutions for new problems.
- *Objective 9:* Have demonstrated a self-directed ability to acquire new knowledge in computing, including the ability to learn about new ideas and advances, techniques,

tools, and languages, and to use them effectively; and to be motivated to engage in life-long learning.

- *Objective 10:* Have demonstrated the ability to work effectively in a development team.
- *Outcome 12:* Comprehend important issues related to the development of computer-based systems in a professional context using a well-defined process to guide development.

Numerous anecdotal comments on the course’s effectiveness have been received from both alumni and employers, but our systematic assessment has begun with the use of alumni surveys and the evaluation of groups’ work-products.

A survey given to a large sample of alumni included several questions that directly addressed the course outcomes defined for the course. Responses from 1997, 1999, and 2001 graduates were analyzed to see how students believe our program, including this course, prepared them in comparison to their peers. Specifically, they were asked to rank on a scale of 1-4 how well prepared they were after graduation in a variety of areas of skill or expertise. For several of these areas, this course is the primary place in the curriculum where a specific area or skill is addressed. The 45 responses represented 29.0% of the total number of graduates of the BSCS program these three years. The results are summarized in Table I.

TABLE I
ALUMNI SURVEY RESULTS ON STUDENT PREPAREDNESS

How well do you believe your degree prepared you in comparison to your peers? (1= very well 4 = not well at all)	
Problem solving as part of the creation of software or hardware systems	1.43
Working in teams, including following a well-defined development process	1.55
Creating and evaluating designs of software or hardware system	1.74
Engineering of large systems composed of multiple subsystems or components	2.20
Verification, validation, and other quality issues	2.37
Management issues and skills	2.39

None of our required courses directly focus on management issues, so the score of 2.39 for the last question in the table can be used as a “touchstone”, i.e. a baseline measure for something that is not a major emphasis in our program. The two questions with scores close to this touchstone are *verification, validation and other quality issues* and *engineering of large systems*. This suggests that our program, including CS340, is less successful in these areas in comparison to other scores. The second highest score is *working in teams following a well-defined process* is very encouraging, given that this is major component of CS340. We also believe that CS340 is an important factor in the other two scores higher than 2.0, especially for the area *creating and evaluating designs*. Thus we conclude this data indicates

success in achieving Outcomes 4, 10, and 12 listed earlier, and as noted CS340 is one of the key courses in our program that address these outcomes.

As noted earlier in this paper, there is insufficient time in CS340 to pay enough attention to verification. The low score shown in Table I suggests that this really is an area within our program that requires attention.

Outcome 9 focuses on life-long learning, a goal that is somewhat difficult to measure. In CS340, student groups are required to learn and use a variety of new languages and environments without being given formal training or lectures. We feel that this course is a major factor in any success we can identify related to this outcome. For a somewhat larger group of respondents, the alumni survey from Fall 2002 showed that our students do exhibit behaviors of life-long learning post-graduation:

- 51 of 59 respondents had read a computing or technical book after graduation.
- 28 of 60 respondents subscribed to a computing or technical journal.
- 59 of 62 had taken steps to learn a new language, library, system, etc.
- 37 of 62 had taken a professional development or training course.
- 16 of 61 had completed a graduate degree, and 3 more had taken some graduate courses.

We have not yet carried out surveys that tie these results directly to what students learn in CS340, although we believe that this course (and our senior thesis experience) are the major experiences in our program that develop student skills for this outcome.

In spring term 2003, we examined the grades assigned to a subset of the individual work-products created by student teams to assess student performance in specific areas of software engineering as covered by the CS340 class project. Our goal was to identify specific areas that students might be struggling. Table II gives the average score (out of 100) for a set of work-products. These results indicate that students struggle more with the process and management issues in comparison to more software-oriented activities, i.e. requirements analysis and specification. These results (as well as exam scores and students comments) have led to changes made in 2004. The instructor created clearer specifications for the management materials that groups must write and upload to the group Web sites. In addition, in 2005 the instructor added required meetings between groups and teaching assistants, who are instructed to help groups better meet expectations in this area.

TABLE II
SCORES FOR TEAM WORK-PRODUCTS

<i>Team work-product:</i>	<i>Score:</i>
Management Report	50
Project Risk Analysis	71
Implementation Planning	67
Requirements Analysis	83
Requirements Specification	87

In student focus groups, some veterans of CS340 have noted that within student groups certain students take on all responsibilities for programming while others devote themselves completely to documentation and management. The approach followed in the course provides a good pedagogical experience for the major aspects of the software process and management. However, it appears that not all students get sufficient practice of some technical topics because they are undertaken by one or two members of each group only. This leads to an uneven learning experience, and in the last year this has been corrected by adding weekly assignments to the set of activities that students undertake.

Each year we receive a number of unsolicited comments about the course from students who have graduated. Some of these from the last two years are listed in Table III. We also find that industry representatives on our advisory board know and remember the course and how it is organized. We believe this speaks to the course's value for our graduates who are hired by these companies. In Fall 2005, an external review team carried out a site-visit to evaluate our degree program, and CS340 was singled out for special recognition in their report.

TABLE III
COMMENTS FROM GRADUATES

“When [a company] flew me out to Seattle for the last round of interviews, I brought with me a copy of the Requirements Specification that I helped write in CS340. Half of my interviewers centered their discussions on my experiences in CS340. They seemed happy to see that we at UVa are able to pick up experiences dealing with cross-team dependencies, project management, design of large software systems, and the writing of specifications.”
“The way the robot project is setup is unique. Having two separate partner teams that have restricted cross-team interactions may seem artificial in an academic environment, but it really helps simulate real world scenarios. Having this partner team experience and having written a specification were the two most useful things I learned in CS340 for becoming a program manager.”
“...the weekly class presentations were really good experiences. Nothing can get you prepared for the corporate environment like getting grilled by your professor!”
“My learning curve was incredible during the first few months here, partly because so much of it was material I saw in CS340.”
“The skills I picked up in CS340 helped me get the position and helped me succeed during the internship.”

CONCLUSIONS AND FUTURE WORK

The nature of the student project, the organization of groups and teams, and the use of studio presentations all contribute to a software engineering course for our third-year students. Moreover, our approach allows us to support a large course enrollment in a way that results in a course that alumni and employers praise. This course has been refined each year that it has been taught for the past decade. During that time, close to 1,000 students have taken the course.

Our alumni survey results and comments from current students, alumni, and employers give us confidence that our course is meeting its goals within our program. In the future

we plan to attempt a better assessment of individual students' skills in major areas like system design and understanding of software process.

ACKNOWLEDGMENT

The original development of CS340 was partially supported by National Science Foundation grant 9156112. Work to export and adapt the CS340 model at Florida Atlantic University was supported by National Science Foundation grant 0196219. Lockheed-Martin and Microsoft Research have provided donations to support the development and delivery of CS340 for many years.

REFERENCES

- [1] Bach, J., "SE Education: We're on Our Own," IEEE Software, vol. 14, no. 6, pp. 26-28, 1997.
- [2] Dart, P., Johnston, L., Schmidt, C., and Sonenberg, L., "Developing an Accredited Software Engineering Program," IEEE Software, vol. 14, no. 6, pp. 66-70, 1997.
- [3] Dawson, R. and Newsham, R., "Introducing Software Engineers to the Real World," IEEE Software, vol. 14, no. 6, pp. 37-43, 1997.
- [4] Dawson, R.J., Newsham, R.W., Fernley, B.W., "Bringing the 'real world' of software engineering to university undergraduate courses," IEEE Proc. Software Engineering, vol. 144, no. 5-6, pp. 287-290, 1997.
- [5] Grove, R.F. "A Proposal for Integrated Software Engineering Education," ACM SIGCSE Bulletin, vol. 29, no. 2, 1997.
- [6] Habra, N., and Dubois, E., "Putting into Practice Advanced Software Engineering Techniques through Student Projects," Proc. 7th SEI Conference on Software Engineering Education, San Antonio, Tex., Jan, 1994.
- [7] Jackson, U., Manaris, B., and McCauley, R., "Strategies for Effective Integration of Software Engineering Concepts and Techniques into the Undergraduate Computer Science Curriculum," ACM SIGCSE Bulletin, vol. 29, no. 2, 1997.
- [8] Johnson, H.A., "Integrating Software Engineering into the Traditional Computer Science Curriculum," ACM SIGCSE Bulletin, vol. 29, no. 2, 1997.
- [9] Lethbridge, T.C., "A survey of the Relevance of Computer Science and Software Engineering Education," Proc. 11th SEI Conference on Software Engineering Education, Atlanta, GA, Feb. 1998.
- [10] Lutz, M.J. and Naveda, J.F., "The Road Less Traveled: A Baccalaureate Degree in Software Engineering," ACM SIGCSE Bulletin, vol. 29, no. 1, 1997.
- [11] Maibaum, T.S.E., "What We Teach Software Engineers in the University: Do We Take Engineering Seriously?" Software Engineering Notes, vol. 22, no. 6, pp.40-50, 1997.
- [12] Tockey, S., "A Missing Link in Software Engineering," IEEE Software, vol. 14, no. 6, pp. 31-39, 1997.