Evolving an Undergraduate Software Engineering Course

Mark J. Sebern
Milwaukee School of Engineering

Abstract

Many undergraduate software engineering courses combine team projects with discussion of development cycle concepts. It can be difficult to connect these elements in a coherent way, especially when the lecture is a broad survey and the project is sharply focused on meeting the needs of a client.

This paper describes the evolution of a senior software engineering project course that incorporates iterative development of a classroom example and an object-oriented process based on commercial software tools. Although the course time frame (an academic quarter) is too short for significant iteration on the team project, students can participate in an accelerated version of the process by making a small increment to the in-class example. This approach retains the benefits of a realistic, client-centered team development project, while providing experience in a contemporary software development process based on commercial CASE tools.

Introduction

For a number of years, the computer engineering curriculum at the Milwaukee School of Engineering (MSOE) has incorporated a senior course in software engineering (CS-489). Traditionally, this course has combined project work done in large teams for “real” clients (typically MSOE departments) with lecture material covering the software engineering process.

The Good News

The CS-489 software engineering course has been well received by students, and has produced a number of valuable software products. Some recent projects include:

- A program that identifies unoccupied workstations in computer labs across the MSOE campus.
- A three-dimensional visualization program for time- and frequency-domain data from an artificial heart.
- A Web-based system for exploring possible student class schedules.

In some cases, the quality of the software product has been remarkable, especially since we operate on the quarter system and have only ten weeks for each project.

In working on the projects, students also learn much about team building and cooperative effort, and often apply these lessons in the two-quarter “capstone” senior design sequence (CS-400 and CS-401). Even when a team fails, valuable lessons can be learned. One student, reflecting on such an experience during a job interview, was told by the recruiter that his company “normally expects a new hire to take two years to learn that.”

The Bad News

In spite of many successes, CS-489 still had some problems:

- Most students enter the course with little knowledge of the large-scale software development process.
- Earlier versions of CS-489 had taken a survey approach in lecture, covering a variety of analysis and design methods. The result was insufficient depth in any one approach to permit serious use by the students.
- The analysis and design documentation produced by the teams varied widely in format and content, making it difficult for one team to critique and profit from work done by the others.
- Student teams had little access to CASE tools designed to support any of the analysis and design methods discussed in lecture.
- On the project, time permitted only one iteration through the software development process; students did not have an opportunity to apply the lessons learned on the first try.

Course Redesign

In the process of redesigning CS-489, we sought to benefit from the experience of others. Pierre Robillard has described a project course incorporating training on team dynamics, project monitoring, quality assurance, and the use of a CASE tool[8]. David Hutchens and Elizabeth Katz[6] attempted to deal with problems similar to ours, and considered eliminating the team project or spreading the project across two courses. As an alternative, they adopted an iterative development approach that incorporated three passes through the development cycle, delivering increasing functionality each time. They recommended using only a single project for all teams, so that class time could be spent discussing the project and that students would be more comfortable commenting on each other's work.

Scott Brown, Norman Wilde, and John Carlin[3] have reported on the use of software maintenance as a vehicle for teaching the development process. By making relatively small changes to a large software system, the focus could be kept on the process rather than on coding details. In a paper on balancing product and process in introductory software engineering courses, Donald Bagert[1] also recommended the use of software maintenance in follow-up courses.

While many aspects of these approaches were attractive, not all fit well with the goals and constraints of CS-489. With only a ten-week quarter to work with, it seemed unlikely that three iterations on a significant project could in fact be accomplished. Even though the use of a single project for the entire class would simplify matters, a significant clientele has developed for the course projects, and we were reluctant to discourage people who had contributed project requests by serving only one at a time.

With these considerations in mind, the following course parameters were established:

- The lecture material on analysis and design methods would focus on a single technique. We chose the Booch method and a corresponding text[2].

- Projects would be chosen as in the past, though preference would be given to maintenance or enhancement of prior projects. Each professor would decide whether teams would work on a common project or on separate projects.
- Much of the lecture material would be structured around an actual application, separate from any team project. This lecture example would itself be an iterative enhancement effort, led by the instructor with the participation of the entire class.
- An object-oriented CASE tool (Rational Rose) would support the modeling and design of the lecture example.

Lecture Content

As background for the team projects and the in-class example, a number of traditional software engineering topics are covered. These subjects include:

- The software crisis, including brief case studies of software project failures, cost overruns, and the human cost of bad software development practices. (Peter Neumann's book[7] is a good source for these anecdotes.)
- The Software Engineering Institute (SEI) Capability Maturity Model (CMM), and the key practices identified as central to disciplined software development.
- The software life cycle, including the traditional stages of analysis, design, implementation, testing, and maintenance. (It is important for students to recognize these various activities, even if they are incorporated into a process other than a unidirectional "waterfall" model.)
- A brief overview of traditional structured analysis and design techniques and data modeling.
- A thorough introduction to object-oriented analysis and design, focusing on the Booch method[2].
- A review of cleanroom software engineering, formal methods, software metrics, and estimation.
- A brief discussion of the Personal Software Process, a set of software process practices that can be implemented by the individual software developer, as described by Watts Humphrey[4,5].

The schedule for these topics is coordinated with the other course activities.

In-Class Example

In the author's section of CS-489, the lecture example chosen was a faculty workload scheduling system. This system is used by a department head to assign classes and other duties to individual faculty, using a calculation system that takes into account class size, type of course, whether a particular course is being taught for the first time, and other factors. The specification for the scheduling application calls for graphical manipulation of relevant objects (e.g., professor, course, section, classroom) and the ability to print reports of individual and aggregate data.

Students are given a somewhat ambiguous preliminary specification, which is discussed in class. As part of a discussion on requirements elicitation, students attempt to clarify and complete the

specification. Later, they are shown the current version of the application and the analysis and design documentation on which it is based. For example, the initial iteration may have the ability to maintain the list of instructors, but not yet handle workload elements such as course sections. At this point, the application interface may resemble Figure 1.

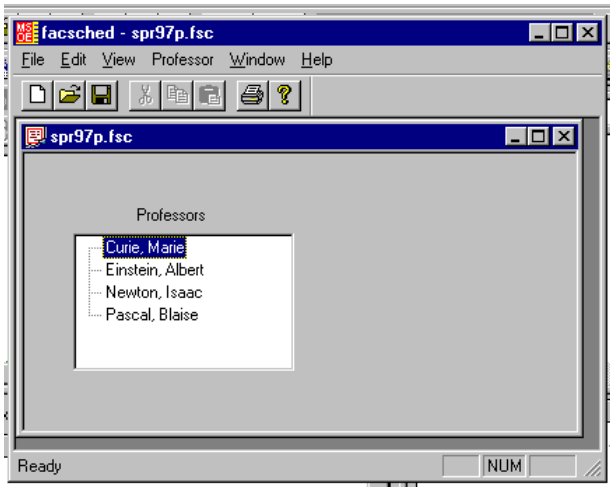

Figure 1. First phase of scheduling application

Along with the application, students are given a chance to explore the Rose model used to construct it. One class diagram for the example model is shown in Figure 2.

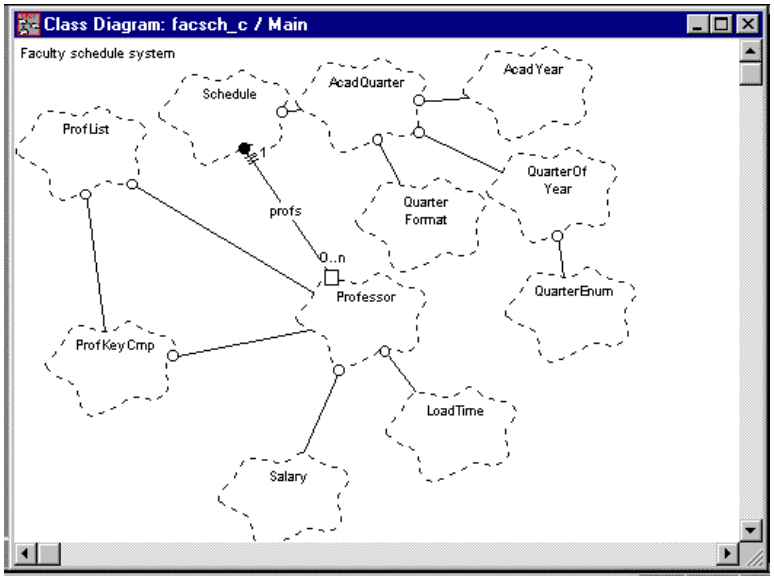

Figure 2. Partial class diagram of scheduling application

Because at this point not all students in the class have had prior experience with C++, some review material is included. The concepts of objects, attributes, behaviors, and relationships are more important in this context than some of the more esoteric details of C++ syntax and semantics. Since the computer engineering curriculum at MSOE is now in transition from C to

C++ as a primary language, this lack of object experience is expected to be less of a problem in the future.

In lecture and class exercises, the next phase of the example application is planned. The students study the relevant parts of the preliminary specification, and prepare a detailed description of the requested enhancement. The existing model is augmented to include any new objects, attributes, and behaviors.

The updated analysis model is then used as a basis for detailed design. For example, object relationships may be implemented with existing container classes, such as those in the Standard Template Library (STL)[9]. Information on the physical model (e.g., partitioning of class specifications and implementations into source modules) is updated in the Rose model. The CASE tool's code generation capability is then used to modify and create the program source files.

Course Project

As mentioned above, teams may work on separate projects or on different portions of a common project. The author requests resumes from all students, detailing their skills and experience, and assigns them to large teams (5-6 members) so that most students are working with colleagues other than their partners from prior courses.

Teams interview their clients to determine the requirements they are to meet. Lecture material gives the teams advance warning of some of the difficulties they may encounter, such as clients who disagree, experience difficulty in articulating their needs, or have unrealistic expectations. Each team prepares an analysis document detailing the system requirements, making use of the CASE tool for modeling.

Analysis documents are reviewed by another team and by the client(s). Once approved, the analysis becomes a basis for the system design, embodied in a second document. Again, the use of the CASE tool is encouraged, as appropriate. When different projects are involved, the design document is reviewed by the team itself, since at this point the specific domain knowledge and familiarity with the overall system structure become more important.

The teams then proceed to implementation. At least one formal code review is done during class, and teams are encouraged to follow the same procedure for the entire system. Test plans are prepared and executed, and user documentation is written. At the conclusion of the course, each team presents its work to the class and acceptance testing is done by the client(s).

Another MSOE faculty member, Professor Jeff Blessing, has applied a similar approach to the implementation of Web applications with Java. The preliminary version of our CASE tool's Java support left much to be desired, but otherwise the experience was quite successful.

Summary and Future Directions

Initial indications are that the integration of commercial tools and a single object-oriented design technique into CS-489 has been beneficial. We would like to augment this approach by incorporating more of the background material into earlier courses.

Some course materials for CS-489 are available on the World Wide Web, at `http://www.msoe.edu/~sebern/asee97/swe/`.

Acknowledgment

Lab facilities for CS-489 are supported in part by a software license grant from Microsoft Corporation's Instructional Lab Grant program.

References

1. Bagert, D. J. Balancing process and product. *Proceedings of the Ninth Conference on Software Engineering Education*, (Apr 1996), 78-84.
2. Booch, G. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.
3. Brown, S. M., Wilde, N., and Carlin, J. D. A software maintenance process architecture. *Proceedings of the Ninth Conference on Software Engineering Education*, (Apr 1996), 130-141.
4. Humphrey, W. S. *A Discipline for Software Engineering*, Addison-Wesley, 1995.
5. Humphrey, W. S. *Introduction to the Personal Software Process*, Addison-Wesley, 1997.
6. Hutchens, D. H., and Katz, E. E. Using iterative enhancement in undergraduate software engineering courses. *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, (Feb 1996), 266-270.
7. Neumann, P. *Computer-Related Risks*, Addison-Wesley, 1995.
8. Robillard, P. N. Teaching software engineering through a project-oriented course. *Proceedings of the Ninth Conference on Software Engineering Education*, (Apr 1996), 85-94.
9. Stepanov, A. and Lee, M. *The Standard Template Library*. Hewlett-Packard Co., Palo Alto, CA, 1995.

Biography

MARK SEBERN is a professor in the EECS department of the Milwaukee School of Engineering, and director of the Bachelor of Science in Computer Engineering program. He received Ph.D. and B.S. degrees in electrical engineering from Marquette University in 1974 and 1972, respectively. Prior to joining the full-time MSOE faculty in 1994, he worked in industry for twenty years as a computer engineer and consultant.