TEACHING SOFTWARE ENGINEERING IN THE ADULT EDUCATION ENVIRONMENT

Steven M. Jacobs
TRW Defense and Space Systems Group
Redondo Beach, California 90278
and
Continuing Education in Engineering and Mathematics
UCLA Extension
Los Angeles, California 90024

ABSTRACT

Teaching the evolving subject of software engineering has only recently been explored in the literature within the last five years. In a university-level, evening school environment, problems in the area of software engineering education arise due to 1) the quantity and approach of introducing software engineering concepts and 2) the background and motivation of the students. Working adults can be introduced to the components of the software life-cycle by a careful selection of reading assignments, lectures, discussion, and a team programming project. This paper addresses the problems associated with software engineering in adult education and presents a working solution.
Keywords: software engineering, software engineering education, software life-cycle, structured analysis, structured design, top-down concepts, structured programming, verification and validation, software management, chief programmer teams.

INTRODUCTION

One of the key issues facing the field of software engineering in the 1980's is the need to develop and standardize software engineering education. Only within the last five years has the design of a software engineering curriculum even been addressed (1,2,3,4). Jensen and Tonies(5) state that there are two problems associated with teaching software engineering. The most severe problem is the infancy of software engineering. New developments are constantly being made in this field, as is supported by the fact that the term "software engineer" is only a decade old. The lack of proven software design strategies and sufficient, current textbooks is indicative of this problem.

The other problem is the difference between the academic approaches versus the industrial methodologies of software engineering. For example, composition and goals of programming teams from academia differ drastically from the industrial counterpart, such as a Chief Programmer Team, as defined by Baker(6).

New developments in software tools and design methodologies that are developed in either environments are often slowly integrated into the software engineering classroom. There is very little experience in teaching software engineering and an acute shortage of teaching materials. Software engineering education is the newest engineering discipline and has only begun to become a standard part of traditional computer science coursework.

In a university-level evening school or adult education environment, the problems become even more acute. The quantity and approach of introducing software engineering concepts must be carefully planned. This presentation must be geared toward students who are often working professionals of varying backgrounds, returning to an academic environment for further training and professional development.

This paper describes a working solution to the problem of teaching software engineering in an adult education environment. Working adults are introduced to elements of the entire software life-cycle, from preliminary analyses to final operation and maintenance phases. Software engineering concepts are introduced throughout the one-quarter course, by way of reading assignments, lectures, discussion and a team programming project.

The remainder of this paper describes one approach that has been successfully followed by the author. The types of classroom materials and techniques are discussed, Problem areas and helpful suggestions are also included.

CLASSROOM ENVIRONMENT

A number of constraints must be considered before beginning a software engineering class in an adult education environment. The quantity of software engineering concepts and the approach used in introducing them must be geared toward the background of the students. Available computers, materials, texts, and other logistics matters must also be considered.

The Students

Students in an evening school environment are mostly working adults with often a minimum amount of available free time. There are a number of types of students. One is the business computing professional, the typical COBOL programmer working daily on a large-scale computer. Another in the scientific computer programmer, mostly familiar with FORTRAN, BASIC, and PASCAL for such applications as defense, space, or energy research and development. The third is the engineer or scientist who has little programming experience, but is participating on systems development with programmers who follow or appear to follow modern software engineering practices. Independent consultants may also enroll, but they can come from any of the above backgrounds. A last student type is the university-trained, full-time student, who has received a traditional, academic computer science education.

The motivation for the students varies. Some are enrolled for professional and personal development. Others need a knowledge of software engineering concepts to participate in a modern systems development team. Still others attend because their management requested or required it. Full-time students expect to learn something "practical". All the students have a minimum amount of time to learn the maximum they can about software engineering.

Some students are looking for an alternative to an expensive, four-day software engineering seminar that includes primarily the philosophy of one software development methodology.

THE COURSE

The course taught by the author is called "Structured Software Development". The approach was to describe the components of the software life-cycle in general terms the first evening: the analysis and design phases, coding and implementation phases, and testing. Then, each of the components were described in detail the remainder of the quarter: approximately one three-hour class meeting for each of the analysis, design, programming, testing, etc., phases. Each of these subjects could justify a larger part of the course or even an entire course. However, to educate working adults in software engineering principles, a survey of the salient information of each subject is appropriate.

The order of this presentation is extremely important. Although the logical development of a software system follows the analysis, design, implementation, test, etc., phases, in that order, a word of caution is given in regards to the sequence of presenting this material. Historically, structured programming and system testing strategies were developed long before software design methodologies in an almost "bottom-up" fashion. Consequently, more concrete knowledge exists today concerning the former strategies rather than the latter. Hence, to start introducing more abstract concepts of software systems analysis and design to the evening student before discussing structured programming constructs can cause confusion or demotivation. This author recommends starting with structured programming, implementation, and testing concepts and then backtracking to the more abstract analysis and design methodologies.

More details about which concepts and methodologies are described in the next section. The course is discussed by emphasizing classroom techniques and providing numerous references for the actual course material. Industrial software development standards, structured walkthroughs, and software management considerations were also included in the course. Evening students can relate these concepts to their own experience and company policies.

Class Materials

The required text has been Jensen and Tonies. This comprehensive text satisfied the authors' goals in presenting the entire software life-cycle with software engineering concepts and examples. Recommended texts were that of DeMarco(7) and Yourdon and Constantine(8). The combination of these texts presents one of the more popular software design strategies in detail, using tools such as data flow diagrams, structure charts, and transform and transaction analyses.

One consistent approach to software design is presented, rather than a survey of design strategies. The Jackson(9) design methodology is introduced and compared as an alternative. Lecture material also covers software management from Yourdon(10). Testing is discussed from a number of sources including Myers(11). Supplementary material and team project (see next section) came from Yourdon(12). Table I illustrates a possible course syllabus.

Outside readings were encouraged and discussed. Even a film (13, 14) on top-down program design was presented to the students. Numerous real-world examples of software engineering concepts were interspersed throughout the class. Such practical concepts are covered by Irvine(15) and Mulhall(16).

Syllabus for

"Structured Software Development"

Week*	Topic**	Principal Reference
1	The Software Life-cycle and Top-down Programming	Yourdon(12)
2	Structured Programming	Jensen and Tonies(5)
3	Structured Analysis	DeMarco(7)
4	Structured Design	Yourdon and Constantine(8), and Jackson(9)
5	Midterm Examination	All the above
6	Testing	Myers(11) and Jensen and Tonies(5)
7	Software Management	Yourdon(10)
8	Actual Industry Standard Practices	Miscellaneous(15, 16)
9	Structured Walk-throughs	Yourdon(10)
10	Final Examination	All the above

* One week implies three class hours

** Emphasize topics by expanding to more than one session for longer school terms

TABLE I

Team Programming Project

The most appropriate method for assigning a software engineering class a programming assignment is by forming programming teams. The author followed the advice of an excellent "how to" article on programming team projects in the classroom that was developed by Khailany and Saxon(17). This work, mixed with chief programmer team concepts, is an excellent and rewarding vehicle for student experimentation with the philosophy and concepts introduced in the classroom.

The teams were made up of three students each: a chief programmer, a support programmer, and a program librarian, three key members of the software development team according to the author(18). Selection of team members and the team project took place as described in Kahilany and Saxon. Students chose from four computer program specifications selected by the author. Assignments were made on a week-to-week basis, paralleling the class discussion. For example, when design strategies were discussed, system design using design tools such as structure charts was assigned, due by the next meeting.

Student-teams submitted a completely developed software system, including a project library and documentation as a final product. Evening students combine their newly learned software engineering concepts with their experience and company software policies to create a high quality result. Copies are made for each team member to keep.

Structured Walk-Throughs

To introduce the students to programming team peer reviews and to encourage the egoless programming philosophy of Weinberg(19), structured walk-throughs, as defined by Yourdon, were conducted by each team on the last regular class meeting. Two weeks prior to the walk-throughs, each team was given the assignment to provide a short (one to two page structure chart, diagram, or structured English description of their program design) hand-out to distribute to the entire class one week before the walk-throughs are to be held. Then, the students were assigned to review the specifications and designs of each team's work.

The structured walk-throughs were then held. Each team appointed a spokesperson to walk through the design and the team responded to questions from the entire class. This proved to be a maturing exercise for those evening students not accustomed to peer reviews in a work environment. It was a gentle enough experience, however, to illustrate the benefits of walk-throughs.

SPECIAL SUGGESTIONS

A number of miscellaneous problems appear in managing a class such as this. Some solutions can be handled by mere common sense. Others, more notable, are described below.

One such problem is the student who desires software engineering knowledge with little or no programming experience. A solution is to assign that person the role of program librarian on the team project. This person then serves as chief documentor and learns by watching and reading his or her teammates' coding. Numerous, straightforward examples of software engineering concepts aid the student new to the field and reinforce concepts of the student with a better background.

Due to the types of adult education students combined with the constant proliferation of software engineering advances, discretion is advised for determining what material could actually be presented and what readings should be encouraged for outside of class. Handouts of key articles and a bibliography(20) are helpful. Choosing one text as the focal point for class discussion was necessary for continuity.

CONCLUSIONS

An evening school course in software engineering can be invaluable experience for students, programmers, engineers, or scientists. One quarter is enough time for students to grasp new software engineering concepts and to experiment with the techniques, as opposed to attending an expensive, four-day seminar of one software development philosophy.

Students are usually self motivated to learn and apply their new knowledge in their classwork and at work or school. They are eager to learn and to discuss alternative software engineering practices. They are interested in sharing their ideas with their instructor, their teammates, and their classmates.

A class in the field of software engineering in any environment, and especially in the adult education setting, should not be taught dogmatically. Software engineering is an evolving discipline and should be treated as such.

ACKNOWLEDGEMENTS

The author wishes to thank Ms. Estelle Klingler of the Continuing Education in Engineering and Mathematics Program of UCLA Extension, Los Angeles, California, and Dr. Tom Renfrow of the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California for their contributions to and review of this paper.

BIBLIOGRAPHY

1. Freeman, P., Wasserman, A.I., Fairley, R.E., "Essential Elements of Software Engineering Education", 2nd International Conference on Software Engineering, San Francisco, CA, 1976, IEEE Cat. No. 76CH1126-4C.

2. Wasserman, A. I., and Freeman, P., "Software Engineering Concepts and Computer Science Curricula", IEEE Computer, June 1977.

3. Wasserman, A. I. and Freeman, P., Editors, Software Engineering Education, Proceedings of an Interface Workshop, Springer-Verlag, 1976.

4. Horning, J. and Wortman, D., "Software Hut: A Computer Program Engineering Project In The Form Of A Game", IEEE Trans. on Software Engineering, Vol. SE-3, Number 4, July 1977.

5. Jensen, R., and Tonies, C., Software Engineering, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979.

6. Baker, F. T., "Chief Programmer Team Management of Production Programming", IBM Systems Journal, Volume II, Number 1, January 1972.

7. DeMarco, T., Structured Analysis and System Specification, New York: Yourdon, Inc., 1979.

8. Yourdon, E., and Constantine, L., Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979.

9. Jackson, M. A., "Principles of Program Design", New York: Academic Press, Inc., 1979.

10. Yourdon, E., Managing the Structured Techniques, New York: Yourdon, Inc., 1979.

11. Myers, G., The Art of Software Testing, New York: John Wiley and Sons, 1979.

12. Yourdon, E., Techniques of Program Structure and Design, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

13. "Top Down Design, Part I", Los Angeles, CA: University of California, Los Angeles, Edutronics Film Series in Computer Science.

14. "Top Down Design, Part II", Los Angeles, CA: University of California, Los Angeles, Edutronics Film Series in Computer Science.

15. Irvine, A., Standard Practices for the Implementation of Computer Software, Pasadena, CA: Jet Propulsion Laboratory Publication 78-53, September 1978.

16. Mulhall, B., and Jacobs, S., "A Technique For Comparative Assessment of Software Development Management Policies", Arlington, VA: AFIPS Press, Proceedings of the 1980 National Computer Conference, May 1980.

17. Khailany, A., and Saxon, C., "Conducting Project Team Classes in Data Processing", ACM SIGCSE Notices.

18. Jacobs, S., "The Many Faces Of A Program Librarian", INFOSYSTEMS, October 1978.

19. Weinberg, G., The Psychology of Computer Programming, New York: Van Nostrand Reinhold Co., 1971.

20. Kleine, K., "Selected Annotated Bibliography on Software Engineering", ACM SIGSOFT Software Engineering Notes.