Evaluation of the SEPA in Teaching Undergraduate Software Engineering in the Traditional Computer Science Curriculum

Hubert A. Johnson
College of Mathematics and Computer Science
Montclair State University
Upper Montclair, New Jersey 07043
johnsonh@alpha.montclair.edu

Abstract

Experiences provide computer science majors need to mirror the typical situation a student will encounter after graduation. The frustration as well as the benefit of working in a group can only be appreciated if experienced first hand. This has made the introduction of software engineering concepts into the traditional computer science curriculum an invaluable component in all computer science courses.

The use of Software Engineering - A Practitioner's Approach (SEPA) in an undergraduate software engineering course taught at Montclair in the Spring of 1997 was an attempt to expose the students to situations as typical of real-world conditions as possible. In assessing the impact of this approach on undergraduate software education, I concluded that the SEPA can provide a rich and significantly worthwhile experience for students as the students in this course had very positive feedback regarding the course.

Introduction

An essential part of the software engineering course was the team projects. The projects were intended to give students a first hand experience in industrial or business problems. Unlike many other software engineering courses which use a single project and have all teams work on the same project, this course allowed different teams to work on different projects. This, of course, presented certain difficulties for the instructor in terms of grading each project based on its level of sophistication. The students, on the other hand, faced two different and major problems: Dealing with the software engineering concepts and learning a new language—Ada. The reason for using different projects was to cater to the different levels of interests and the sophistication of the students.

Background

Most of the students in this course had never before worked on a group project. For those who had, the projects were not as structured and time consuming as the ones in this course.

The course, which used Software Engineering: A Practitioner's Approach (SEPA), was designed to provide as practical an experience in software development as possible. In this course, although an attempt was made to present a balanced treatment of the topics discussed, more emphasis was placed on development issues (analysis, design, coding, testing, management.)

The classroom component of this project-oriented course was organized as a lecture/meeting. The class met for three fifty-minute periods each week. The first two periods each week were devoted to lecture and the last to team meetings.

Adapting the guidelines suggested by Pressman, at the first class meeting the handout given to (and discussed with) students addressed the issues of course organization, course requirements, team organization, and course evaluation.

Course Organization

Course organization described the course, explained the organization of the course, outlined the lecture topics and project assignment due dates, gave hints on working in groups and writing up the project document. Finally, some suggested projects were given and a detailed description of the assignments related to the project. The focus of this course was on (1) taking an in-depth look at the concepts and process which a software engineer need to master and employ in software development, and (2) applying these concepts in building a non-trivial team term project.

In this course, students were exposed to a systematic software development process that included milestones and quality assurance checks, by making assignment submission modular and incremental in nature. Students had the opportunity to design, manage, and implement a medium-size project. Both the lecture and the projects covered topics in software engineering management, problem specification and analysis, system design techniques, documentation, system testing and performance evaluation, software maintenance, reliability, and current programming and run-time environments. Every effort was made to avoid just offering an advanced programming course as software

engineering. The course was explicitly designed to present, and emphasize, topics relevant to each phase in the software engineering process

A list of possible projects from which students could choose was provided at the first class meeting. Examples of suggested projects used in the course were:

A string-processing program: This would include facilities for operating on a string (read a line of test and convert all upper case letters to lower case, copying, comparing, concatenating, and replacing).

A scheduling program: This would provide a scheduling committee in a department at a university the facility to assign courses to faculty, classroom, and other things.

There were a total of twelve suggested project possibilities. Students, however, had the option of choosing a project outside those proposed by the instructor. If a project was chosen outside those suggested (and some teams did) the instructor's approval had to be obtained before proceeding.

The document distributed to students at the first class meeting also stipulated the due date for each deliverable for the semester. This took the form:

Week 2: Teams are required to submit a description of the project the team has chosen

Week 3: Submit a draft of the paper (described in week 2)

Week 4: The final version of the paper due.

The instructor made a point to emphasize (in writing) that the main purpose of the functional specification (required in week 4) was to explain what the students were going to do for their project rather than how the goals would be accomplished. The functional specification would later be coupled with a management plan and design to comprise the proposal to the sponsor/client (a role played by the instructor). As a guide, the handout to students specified what the contents of the functional specification should entail. Each phase of the software development had a "guide" as to the nature of what was expected by the instructor as deliverables.

Course Requirement

There were reading assignments and other homework related to the project. An important requirement of the course was a weekly log kept by each student of the time spent on different course-related activities.

Team Organization

This was a class of 19 students. These students divided into teams with three or four in each team. The teams were "democratically" formed in that students were instructed to choose the person(s) they wish to work with. The reason for this approach was to reduce the likelihood of someone using his/her being placed in a particular group as an excuse for unsatisfactory work, which could then significantly affect the quality of the project. Each member of a team had the opportunity of taking on interchanging roles for each phase of the project development. Each served as team leader, recorder, and editor at some point.

The role of the leader was mainly to delegate responsibilities to each team member. The leader also served as the project coordinator/editor and was responsible for overseeing the interfacing of the components developed by individual team members into a working system.

Course Evaluation

To help students give a fair assessment of the course, they were instructed at the outset (and continually reminded throughout the semester) to keep a written record (a log) of the things they found difficult, laborious, interesting, useful, or disliked.

About a week before the end of the semester students were reminded to start working on their written evaluation of the course. They were assured that a candid evaluation was expected and that this would have no bearing on their final grade. The process of gaining students' trust so they could write as freely as possible in their evaluation, was an easy one as many of the students had previously done some course with me.

A Summary of Students Comments

The experience in working in teams was a valuable one as it gave them some experience in what it is to make compromises.

The team meetings were extremely useful as they served to clarify certain project-related concepts/activity about which someone might have had doubt or misunderstanding. Using the third class period for team meetings was a great idea as it resolved conflicts in students' schedule that prohibited some from meeting outside of class.

Deadlines set by the instructor for submission of project modules were useful as it served to keep the teams focused and prevented procrastination. Some team members, however, found the deadlines somewhat taxing.

Students learned a lot from serving in different roles on a team during the software development process.

Students were amazed at, and initially disliked, the amount of documentation that had to be produced. This dislike
was, however, dismissed or tempered considerably when they realized that referring to the documentation made writing the program and producing a user's manual a lot easier than expected.

The specifications and well-written designs made communication problems among team members almost nonexistent.

One team member wrote: "It's clear that the greatest amount of time was spent on Design, which definitely facilitated the coding phase. Design followed fairly smoothly from the functional specification, and since we

spent so much time on Design, both top-level and detailed, the coding was easy to organize."

Dealing with the software engineering concepts and learning a new language at the same time was challenging.

Self-doubt. Some students expressed some initial feeling of self doubt during the early stages of the system development. This doubt they claim, however, was short-lived because of the lectures and the clear guide provided by the documentation, as well as the discussions with fellow team members.

The logs proved very valuable as, according to the students, these logs raised their level of consciousness as it relates to the projects. It caused them to start monitoring their time and to develop a greater sense of being able to estimate the time required for a task.

Another student wrote: "The original project schedule plans were somewhat contrived to fit into the school semester, thus we ran out of time near the end of the semester and so were not able to implement the testing phase."

Problems

There were two major problems encountered in this course. The first was the difficulty with unstable environment on which the Ada compiler was installed. This led to students using the GNU compiler which they down-loaded from public domain on the web.

The second pertains to one team. Starting with the design phase one member of one of the teams refused to compromise on the proposed design suggested by other team members. This student developed his own design. Because the course was designed to provide "real-world experience" in working on team projects, the student was eventually convinced to go along with the majority. This then gave rise to social and ethical issues, and discussions in which the entire class participated. The student was so adamantly opposed to his team's design that although he eventually performed his share of tasks as a team member, he proceeded to implement his own design as well.

Students Recommendations

As indicated earlier, at the first class meeting students were not only informed of the requirements to keep track of their likes and dislikes but they were also required to give written recommendations for improving the features of the course they disliked. This request was made with the hope that such recommendations would serve to make the course a more effective one, and thus enable students to benefit from it the next time it was taught.

It is interesting to note that although a major dislike was for the amount of work involved in producing the documentation, heading the list of recommendations was the need for the next group of students doing this course (and anyone developing software) to experience the task of producing the required documentation in developing a system.

Many students felt that this experience should not just be limited to this particular course but should be utilized in other computer science courses as well.

Students strongly recommended that the course be taught using the programming language normally used (currently C++), instead of having them grapple with a new language and the software engineering concepts simultaneously. This they stated would perhaps be more manageable if the system was more stable.

Another recommendation was that this course should be made a required course for computer science majors instead of being optional, as it currently is.

Conclusion

I feel that the SEPA is a valuable technique which can be easily and effectively integrated into the computer science curriculum. As it was implemented in this specific software engineering course, students were provided a rich experience working on a team project and in learning what it takes to be a successful software developer. Students developed a sense of responsibility and accountability (to their teammates). The quality of their projects was outstanding, as was demonstrated by the presentation given by each team at the last class meeting for the semester.

Based on the results, more of this approach, although on a limited scale, will be incorporated into several of my other computer science courses. Modules have already been adopted into my CS1, CS2, and CS3 courses. The SEPA model is one which I would, without hesitation, encourage computer science educators to incorporate into their courses. It takes a tremendous amount of planning prior to the start of the semester but the results are well worth it when one observes what the students are able to accomplish.

In considering the students' recommendations, in the future I will use the programming language that is in general use in the department (currently C++). Another consideration would be to allow students to use any language with which they are comfortable.

References

1. ACM/IEEE-CS Joint Curricula Task Force (1990) Computing Curricula 1991. Tucker, Allen B.,et. al., ACM Press/IEEE Computer Press.

2. Booch, Grady. "Software Engineering With Ada 3rd Ed."; Addison-Wesley 1996.

3. Gersting, Judith L. "A software Engineering 'Frosting' on a Traditional CS-1 Course" p 233-235, Twenty-first SIGCSE Technical Symposium on Computer Science Education. Vol 26, No 1, March 1994.

4. McFarland, G. "The Benefit of Bottom-up Design" p 43-51; ACM Software Engineering Notes, Vol. 11, No. 5 Oct 1986.

5. Mynatt, B. T., "Software Engineering with Student Project Guidance." Prentice Hall 1991.

6. Pressman, R., "Software Engineering: A Practitioner's Approach 4th Ed." McGraw-Hill 1997.