COMPUTER SCIENCE EDUCATION 7, 109–132, (1996)

Organizing and Managing Software Engineering Team Projects

Hossein Saiedian
University of Nebraska at Omaha

Industrial software development today requires a fundamental education in computer science as well as the ability to work productively and collaboratively in a team environment. Employers will therefore favor graduates who have mastered computer science and software engineering concepts and can apply them while developing a software system. To produce computer science graduates possessing the skills necessary to succeed in the workplace, team-oriented software engineering courses with real projects (and with real clients) are increasingly emphasized. It is, however, difficult to successfully present a software engineering course that covers software engineering concepts and offers opportunities to apply them during a project in a team environment. The difficulties lie in project selection, team formation, team and project organization, process management, and, finally, grading. The objective of this article is to discuss these difficulties and provide suggestions for alleviating or avoiding them.

1. INTRODUCTION

There are various approaches to teaching a software engineering course. Some instructors discuss theoretical concepts and leave it to their stu-

I thank Evans Adams, Donald Gotterbarn, Linda Northrop, and Stuart Zweben for the valuable insights they provided during a SIGCSE panel [10]. My special thanks go to Renée McCauley for organizing and moderating the panel, and for commenting on an earlier version of this article. I am also grateful to my software engineering students for their valuable feedback especially those who participated in the surveys. Comments from anonymous reviewers have been useful. This work was supported in part by a UCR grant from the University Committee on Research, and in part by a UCAT grant from the University Committee on the Advancement of Teaching, University of Nebraska at Omaha, Omaha, Nebraska.

Correspondence and requests for reprints should be sent to Hossein Saiedian, Department of Computer Science, University of Nebraska at Omaha, Omaha, NE 68182–0500. E-mail: hossein@cs.unomaha.edu

Section 5. Criteria for team composition are given in Section 6, while an explanation of the project deliveries appears in Section 7. Regardless of how much thought and care is put into selecting a project and forming the teams, there are still a number of issues and obstacles that surface during the project development. Section 8 enumerates the most common problems and offers some suggestions for alleviating or avoiding them. A difficult task in administering a project-intensive, team-oriented class is to determine how individual team members are evaluated. Evaluation and grading issues are presented in Section 9.

2. A PROJECT-INTENSIVE APPROACH IS ESSENTIAL

University teaching is often compared with the way in which industry works outside the university. While a university career is largely judged by individual achievement and assessment, a computing career is largely judged by group cooperative activity. Since real-world software development is a team-oriented activity, university students should be exposed to such activity in order to meet the needs and demands of industry. This can sometimes be achieved through internship, but internship opportunities are not available to all. However, experience with a realistic group project in an academic environment can be a valuable substitute [1–3]. That is why team projects are becoming an essential part of courses like software engineering, database management systems, and systems analysis and design [1,3–7]. Early participation in and exposure to team projects would help students to understand, for example, the communication aspects and interpersonal skills vital to being professionally competent in the real world. A project-oriented course also highlights some of the practical issues that face software development teams in industry.

Team projects have a high positive impact on students and are becoming the preferred method of teaching the computer science courses mentioned. The objective is not to merely meet real-world necessities, but also to enhance teaching in educational institutions. Learning by doing is essential for software engineering course [5], and a project-intensive approach gives an opportunity to students to apply the concepts they have been exposed to in the classroom and readings. Students gain experience of actually applying methods and techniques for software requirements definition, design, implementation, and testing presented in the classroom and gain some practical experience in project management and preparing user's manuals. Thus, students learn how to solicit requirements, how to design, how to map the design into implementation, how to test the implementation, how to write user's manuals, and

Interdependence among all is the key to any successful project. It is practically impossible for an individual to possess all the skills required for the project at hand. Therefore, higher productivity can be achieved only through cooperation and when the individual members are conscious of the fact that one person "can't do it all" [4]. Students, by participating in team-oriented projects, learn about their personal and professional strengths and weaknesses, and recognize the need to help and appreciate strengths and weaknesses of other individuals in the group. In addition, students' talents are balanced by those of others.

3.3 Creative Solutions Are Encouraged

Creativity flows out of individuals when they involve themselves in active discussions. Team-oriented projects, where students feel comfortable with each other and share an informal relationship, help team members come up with more creative solutions, because of the freedom to express their ideas and criticize others' ideas, and it gives them a wider perception of and a different outlook on a given task, which helps to analyze problems in a very objective manner. In addition, group projects tend to have "hidden" requirements which only emerge under diligent probing by team discussions [4].

Brainstorming sessions come up with a variety of possible solutions to some particular problem or set of closely related problems. The ideas suggested by one person often inspire ideas in other people; a group that interacts well together in this way can be much more creative than any one person in the group. At the end of the brainstorming session, team members will evaluate all the ideas that came out of it, measuring all the ideas according to a common set of criteria. Performing this evaluation may eliminate several ideas, but may also either suggest modifications to others or suggest new ones.

3.4 Students Develop Vital Communication Skills

Students must do their part of the project using other students' work. This generates a substantial amount of discussion and communication between students. The volume of communication and natural interaction among team members encourages each member to develop and sharpen his or her communication and coordination skills.

3.5 Students Share Knowledge

The team-oriented project will also provide the opportunity for a student to learn new skills and practice them. In addition to acquiring new

cation problems grow as the size of the project and teams grow, and when communication problems predominate the team and the project will both be in deep trouble. Occasionally, when the team is too large, it may be possible for one or more team members not to do their share of the work.

4.2 Single Project and Multiple Teams

Another alternative is to still choose a single large project, but to divide it into multiple "independent" components. The class is divided into multiple teams, and each component is assigned to a team. The author has had a certain degree of success in organizing and managing such an approach. This alternative is most successful when the problem can effectively be divided into independent components. A major advantage of such an approach is that it may encourage healthy discussion among teams (or those teams whose components depend on each other) and thus may teach students a lesson or two about cooperation between teams. Furthermore, the class as a whole may feel a sense of accomplishment because they have, as a group, developed a reasonably large software system.

4.3 Multiple Teams and Single Project

Yet another alternative is to identify a smaller project, divide the class into a number of teams, and have each team work independently on the same project. The merit of this approach is that it will cause healthy competition among teams. Furthermore, it is substantially less demanding on the instructor as he or she has to know only one set of requirements for the same small project, and it is certainly easier to manage and supervise teams that work on a smaller project. The author has had a great deal of success with such an approach also.

4.4 Multiple Projects and Multiple Teams

A fourth alternative is to obtain multiple projects, divide the class into multiple teams, and assign each project to a different team. This alternative may be quite demanding on the instructor, as he or she must familiarize himself or herself with the requirements of multiple projects in order to supervise the students' work. Furthermore, the instructor must be careful in selecting projects with the same degree of complexity in order to fairly balance the work among teams.

There are certainly other alternatives. During a SIGCSE panel on a similar topic [10], the panelists (including the author) discussed a number of

Although students working on a real project will gain invaluable communication and interpersonal skills, there are a number of issues related to real projects (and real clients) that an instructor (as well as the students) must be aware of. We discuss some of these issues here.

If the client is an enterprise from the business community, students working on the project may have to make off-campus trips during the development process in order to interact with the client and the potential users of the system. Real projects also tend to be large, and the client is only interested in the practical value of the project and most likely will not understand (or care) about the pedagogical objectives. Thus, it becomes a responsibility of the instructor not to allow the client to exert direct or indirect pressure on the students. As observed in [11], the instructor must ensure that the customer understands that the project may not be completed on time, and that if the project is completed, students are no longer obligated to maintain the software. Furthermore, to control the entire development process, it is essential to establish firm requirements early in the semester and not allow the customer (or the students) to change the requirements. In fact, "frozen requirements" and firm milestones are important preconditions for a successful project. It is equally important for students (and sometimes, the client) to know what the students' obligations are with respect to the course as a whole (e.g., the students also have to study and take exams).

5.2 Nonrealistic Projects

As an alternative to a real project with a real client, the instructor may choose to provide students with a definition of a software problem describing the desired functionality of a system (typical examples may include a registrar system or an airline reservation system). The project description would require students to consider developing a system that would address the needs of some "imaginary" client (although the instructor may serve as the system procurer). The requirements are usually rather artificial.

The primary advantage of such projects is that students will no longer have to make off-campus trips and no longer have to face issues discussed earlier. However, students very often may need to make assumptions about the problem, expected operations, and those requirements of the "imaginary" enterprise that are not specified in the project description. Depending on the assumptions made, the development process may be either oversimplified or unnecessarily complicated. In addition, if the instructor requires a special feature or mandates a specific approach to the construction of the software, students may feel that

presentation and ask all team members in a team to participate and contribute to their own team's presentation.

7. PROJECT DELIVERIES

The project is organized in a way similar to that described by Kant [14] and Tomayko [15]. A phased approached is used and each team is required to achieve four milestones at approximately three-week intervals. In general, each team must produce a document (e.g., requirements specification, test plan, or user's manual) for each major phase of the software life cycle, and at the end of the semester do a presentation highlighting the features of the system, and a demonstration showing its functionality. The development phases that we emphasize include:

1. Requirements definition and specification
2. Architectural design
3. Detailed design
4. Implementation and testing.

Typical projects whose final software product contains around 3,000 to 4,000 lines of code are selected. Our projects are usually developed for real clients.[1] A short description of the documents that are normally produced for each of the software development phases is given below.

7.1 Requirements Definition and Specification

The purpose of this phase of software development is for the students to explain what they will be doing do for the project, as opposed to how they will accomplish it. They will discuss what their system will look like and produce examples of its behavior. Each function of the proposed system is specified along with legal values or ranges that the function accepts for input and the corresponding output value. Students are asked to follow the guidelines given in the textbook, but normally, a requirements definition and specification includes a title page, an intro-

[1]Examples of real projects our students have developed include a course, classroom, and time scheduling system for our computer science department, a large, 14,000 + line-of-code system for a mathematics lab that provides a friendly facility for numerous kinds of record keeping and report generation for introductory mathematics students. The mathematics lab project was divided into five independent components, each team working on a different component.

Table 1
Grades and Approximate Schedules for Developing the Project

Project Phase	*Approximate Duration*	*Points*
Requirements definition/specification	3 weeks	100
Architectural design	2 weeks	100
Detailed design	3 weeks	100
Implementation and testing	3 weeks	100
Presentation and demonstration	during the last week	100

Note. Projects count toward 50% of students' final grade.

guidelines very carefully and are asked to gear their overall implementation toward modularity and expandability rather than speed. Individual modules are first tested. Each team must carefully plan and document the order in which modules are to be integrated. Functional testing, performance testing, and acceptance testing are scheduled. Each team prepares a test plan that includes a statement of objectives and success criteria, integration plans, testing methodologies, and schedules. Some team members use CASE tools at this stage to produce partial Ada code for Ada packages.

Each team is also responsible for developing a user's manual. The user's manual should have a structure that is evident both to someone reading it straight through and someone who will look for a particular topic. The user's manual is organized as follows: a table of contents, an introduction that concisely describes the system, an overall description of the style of user interaction, detailed systems operations, a list of known features and deficiencies, and an index.

CASE tools are used primarily during the first three phases of development. (The integration of CASE tools into an introductory software engineering course is discussed by Saiedian [17].) The instructor gives introductory lectures and materials on using the CASE tools prior to a particular phase of the project. Table 1 gives an approximate timing schedule for each of these phases during a 15-week semester as well as the points for each part of the project, plus the project weight in a student's overall grade. Each part of the project (including presentation and demonstration) is worth 100 points for a total of 500 points, and account for 50% of a student's grade. The other 50% of a student's grade is determined by classroom exams and quizes.

Students will be graded on the quality of the work they produce, not on how many hours a week they spend on it. However, they are requested to create professional-looking documents, not only for "clients," but also for communication among themselves. Portfolios, labeled theme binders, and the like, are recommended. Each part of the project is grad-

dents have difficulty in expressing their ideas such that the other team members can understand. The problem may be compounded by the fact that some team members may have a "foreign" language or accent.

8.1.2 Lack of Leadership Quality. Assertiveness is an important quality in a team leader, but it should not negatively intervene with his or her decisions regarding the team. When the team leader tends to be unreasonably authoritarian, the outcome can be disastrous. Teammates might have a rebellious attitude which hinders the productivity to a great extent. At times the leader may become prejudiced to certain individuals, which is also a disadvantage to team work.

8.1.3 Conflicts in Members' Schedules. Conflict in schedules is another major problem students have during team projects. Each student's schedule varies from other members in the team because of course and work schedules. It sometimes becomes an impossible task to coordinate a team meeting where the presence of each member is vital. That is why sometimes team meetings are postponed to the weekends, which slows down the process.

8.1.4 Lack of Confidence. In the initial stages of the project, some members undergo a confidence crisis. The fact that there may be two different types of students (i.e., traditional and nontraditional students) in a classroom makes for greater diversity. For example, it may be very overwhelming to a 22-year-old student who has never worked in the "real world" to be suddenly exposed to a large project and have to work with those students who may have been professionals for many years. The professionals are also facing a new kind of challenge in dealing with a problem in a new way. Both situations could lead to a lack of confidence about one's abilities, and students may become flabbergasted and frustrated with the magnitude of the project. When team members are faced with a new and overwhelming task, confidence tends to falter even in the most confident person.

8.1.5 Limited Time. Usually the project assigned to a team is of moderate size. In spite of that, trying to achieve the project goal within the semester time can be cumbersome. Because unexpected difficulties arise during the project and last-minute efforts, there can be lot of pressure on students, especially at the end of the semester. Lack of cooperation can lead to uneven distribution of work, and the common goal may not be achieved if cooperation is not exhibited by every member of the team. Sometimes a student cannot devote enough time to the project because of other obligations. Personal problems may make a student drop the course, and this can lead to unexpected extra project work for others.

Traditionally, a male member with a senior academic standing was considered to be the most appropriate leader. However, a leader must have earned the title. Communication skills, background knowledge, fairness, positive attitude, and coordination skills should be considered as primary factors. Many times a woman or a minority member would be the most effective team leader. If one has already some practical project management experience, he or she should be the most logical candidate. A democratic approach should be chosen by the team in which all individuals contribute to decision making. The team leader (or any individuals in the team for that matter) should avoid investing personal egos, but compromise and cooperate with the other team members. Additional materials on team leadership are in [19].

The team leader must be able to take proper disciplinary action when necessary. Instead of feeling responsible for doing a disproportionate amount of work, the leader must accept reprimanding as a real-work team leader would do in the same situation. The work should be divided and assigned among the members depending on individual abilities.[2] If someone is not actively participating in the meeting discussions or is not expressing an opinion, the team leader should casually ask him or her for input.

It may occasionally be productive to rotate the role of leadership among the teammembers to promote flexibility. One alternative is to have a different leader for each phase of the development to ensure that each member experiences the challenges of team leadership.

8.2.3. Scheduling Conflicts. To avoid or minimize scheduling conflicts during the semester, it is essential to establish a tentative meeting schedule and deadlines early in the semester. Established dates and deadlines are important to overall team success and alleviate many of the conflicts in the future. An initial meeting between each team and the instructor may be useful. The instructor may help the team in delegating responsibilities, in supervising the development of a timetable for the project, and in reminding the students about the significance of attending the meetings and doing their share of the work.

8.2.4 Confidence Crisis. To increase members' confidence, the team leader should continuously put the project into perspective and help quell the feelings of being buried by the project. Once the team members real-

[2]Note that some members are admittedly more productive than others (because of prior experience related to the project). However, that is not necessarily a detriment, as there are varying levels of experience and skills on any software development team, academic or otherwise. The experiences of others promote growth for the inexperienced.

promise," "procrastination," "lack of cooperation," and "personal problems," and offers his suggestions for alleviating or possibly avoiding these problems. In an excellent article in *Communications of the ACM*, Rettig and Simons [21] presented many development and management ideas for small teams in the real workplace from which a software engineering instructor can greatly benefit. In fact, the entire issue of *Communications* is devoted to orchestrating project organization, teams, and management. General issues in software engineering education are covered by a number of excellent articles that appeared in a special issue of the *IEEE Transactions on Software Engineering* (Volume 13, Number 11, November 1987). Allen Parrish and his colleagues [22] discussed a software engineering course in which equal weight and emphasis is given to management concepts. They stress general managerial principles applied to organizational problems arising specifically from technical circumstances such as software development and programming. an undergraduate software engineering program and its mechanisms for leadership skill development is reported in [23]; the program is unique in that it attempts to prepare undergraduate students to become effective leaders in industry.

9. EVALUATING STUDENTS' PROJECTS

Each part of the project is graded based on functionality, accuracy, and completeness of its content as well as its organization(e.g., appropriate title, section, and paragraph names) and appearance (e.g., consistent page numbers). Each graded part is given back to the students who then make the necessary corrections and modifications. At the end of the semester, all project parts are assembled and resubmitted as the final version of the project. The final project is once again graded for completeness and consistency.

A major issue in offering team-oriented project courses is in establishing a fair system for grading the final product of a successful (or failed) project and in determining the grades for individual members of a team. The grading process requires a lot of thought and care, particularly when judging whether a team member has done his or her share of the work. The instructor must determine whether students be evaluated individually or as a team. For example, if the final project earns n points, should each team member receive n points? Should some members who contributed less earn fewer points? What would be the criteria for determining who contributed less? Should contributions be viewed in terms of technical, time, or leadership contributions? The instructor has to find ways to evaluate the performance and contribution of each individual student. A useful method

ly outweigh the difficulties in offering the course, especially when we hear our employed students express their appreciation for the useful experience they gained. The course has been a real learning experience, and we hope that this article proves useful to those who are or will be teaching such a course in the future.

REFERENCES

[1] B. Bruegge, "Teaching an industry-oriented software engineering course," *Software Engineering Education* (LNCS 640) C. Sledge, ed. New York: Springer-Verlag, pp. 65–87.

[2] J. Burnes and E. Robertson, "Two complementary course sequences on the design and implementation of software produces," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 11, pp. 1170–1175, 1987.

[3] H. Saiedian, *Guidelines for a practical approach to the database management system course,"* Journal of Information Systems Education, vol. 4, no. 1, pp. 23–29, 1992.

[4] H. Etlinger, "A retrospective on an early software projects course," *ACM SIGCSE Bulletin*, vol. 22, no. 1, pp. 72–77, 1990.

[5] M. Moore and C. Potts, "Learning by doing: Goals and experiences of two software engineering project courses," in *Software Engineering Education* (LNCS 750), J. Díaz-Herrera, ed. New York: Springer-Verlag, 1994, pp. 151–164.

[6] L. Northrop, "Success with the project-intensive model for an undergraduate software engineering course," *SIGCSE Bulletin*, vol. 21, no. 1, pp. 151–155, 1989.

[7] M. Shaw and J. Tomayko, "Models for undergraduate project courses in software engineering," *Software Engineering Education* (LNCS 536), J. Tomayko, ed. New York: Springer-Verlag, 1991, pp. 33–71.

[8] K. Bosworth and S. Hamilton, *Collaborative Learning: Underlying Processes and Effective Techniques*. San Francisco: Jossey-Bass, 1994.

[9] N. Wirth, "A plea for lean software," *IEEE Computer*, vol. 28, no. 2, pp. 64–68, 1995.

[10] E. Adams, D. Gotternbarn, L. Northrop, H. Saiedian, and S. Zweben, "A 1994 SIGCSE panel on organizational issues in teaching project-oriented software engineering courses," held during the 25th ACM SIGCSE Technical Symposium on Computers, Phoenix, AZ, 1994.

[11] J. Comer, T. Nute, and D. Rodjak, "Some observations on teaching a software project course," *Issues in Software Engineering Education*, R. Fairly and P. Freeman, eds., New York: Springer-Verlag, pp. 115–130, 1989.

[12] T. Scott and J. C. II, "Team selection methods for student programming projects," *Software Engineering Education* (LNCS 895), Edited by R. Ibrahim, ed. New York: Springer-Verlag, 1995, pp. 295–303.

[13] T. Scott, L. Tichenor, J. R. Bisland, and J. C. II, "Team dynamics in student programming projects," *SIGCSE Bulletin*, vol. 26, no. 1, pp. 111–115, 1994.

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

[15] J. Tomayko, "Teaching a project-intensive introduction to software engineering," Tech. Rep. CMU/SEI-87-TR-20, Software Engineering Institute, Sept. 1987.

[16] A. Diller, *Z: An Introduction to Formal Methods*, 2nd ed. Chichester, UK: Wiley, 1994.

[17] H. Saiedian, "Integrating CASE technology into the software engineering education," *Computer Science Education*, vol. 5, no. 2, pp. 189–210, 1994.