A Taxonomy of Organizational Alternatives for Project-Oriented Software Engineering Courses

Hossein Saiedian
Department of Computer Science
University of Nebraska at Omaha
Omaha, Nebraska 68182–0500 USA

Abstract

Industrial software development today requires a fundamental software engineering education as well as the ability to work productively and collaboratively in a team environment. In order to produce 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. There are a large number of organizational issues and alternatives that one must be aware of in order to successfully offer a project course that both covers the software engineering concepts as well as opportunities to apply them. The objective of this article is to discuss these alternatives and issues and provide suggestions for alleviating or avoiding potential problems.

1 Introduction

University teaching is often assessed in comparison with the way in which industry works outside the university. While a university career is largely judged in 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. That is why team projects are becoming an essential part of courses like software engineering, database management systems, and systems analysis and design courses. An early participation and exposure to team projects would help a student to understand the communication aspects, interpersonal skills, etc., that would be vital for 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 above. In fact, the objective is not to merely meet the real world necessities, but also to enhance teaching in the educational institutions. Learning by doing is essential for a software engineering course [3], and a project-intensive approach gives an opportunity to students to apply the concepts they have been exposed to in the classroom and readings. The 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, the students learn how to solicit requirements, how to design, how to map design into implementation, how to test the implementation, how to write user's manuals, and so forth.

2 Merits of a Team Approach

Team work can be defined as a group of people working together, sharing their knowledge and expertise, and coming up with creative solutions towards achieving a common goal. In other words, the most distinguishing characteristic of a team is that its members have, as their highest priority, the accomplishment of team goals. They may have strong personalities, possess highly developed specialized skills, and commit themselves to a variety of personal objectives they hope to achieve through their activities, but to them, the most important business at hand is the success of the group in reaching its goals. The team members are encouraged to support one another, collaborate freely, and communicate openly and clearly with one another. With these in mind, and in order to give our students a broad background in software engineering, and to provide them real work experience, we feel that a team approach is necessary. Other advantages of a team approach are summarized in the rest of this section.

Team work prepares students for real-work episodes. The computing industry as a whole is heading towards "group work" and group-work computing is a requirement of the workforce. Every company is stressing "team-oriented persons" rather than individual brilliance. When students get into the real world, the skills that they have acquired when working on a team will help them go a long way in their career.

A software engineering project usually will be of a moderate size and has to be completed in a time frame of one semester. Such a project cannot be completed successfully by a single student. Teamwork is best to accomplish this successfully.

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 could be achieved only through cooperation and when the individual members are conscious of the fact that one person "can't do it all" [2]. 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 will be balanced and shared.

The students must do their part of the project using other students' work. This generates a substantive amount of discussion between students and helps sharpening their communication and coordination skills. Finally, the productivity of a software development process can be improved with a team oriented approach. In an environment where there are strict deadlines for a project, the team approach helps dividing the work load and allow the members to work in parallel and accomplish the task within the time limit. This approach will take less time for the overall development. Furthermore, one can do a better job concentrating on a specific task rather than on the entire work. Since in team work it is possible to share the work among the members, a student can concentrate on his or her assigned job. The quality of the end product will be improved because a student can do a thorough and detailed job by putting all his or her knowledge in that specific area.

3 Project/Team Organization

There are a number of organizational alternatives for project-oriented software engineering courses. Each of these alternatives have their own merits and drawbacks. Several of these alternatives are discussed in this section. As stated earlier, once the instructor (and students) become aware of the potential drawbacks, they will cope more effectively with the problems when they arise.

3.1 Single Project, Single Team

One alternative is to choose a single large project and form a single team consisting of the entire class. Such an approach is plausible, effective, and successful if the class size is not too large. The instructor must become quite familiar with the requirements of the project in order to most effectively supervise the class progress and work. If the class size is too large, a number of problems emerge. A large project and a large team implies a lot of communication. As noted by Wirth [5], communication 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 size is too large, it may be possible for one or more team members not do their share of the work.

3.2 Single Project, Multiple Teams

Another alternative is to still obtain 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.

3.3 Multiple Teams, Single Project

Yet another alternative is to identify a smaller project, divide the class into a number of teams, and have each team to 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 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.

3.4 Multiple Projects, 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 the teams.

There are certainly other alternatives. During a 1994 ACM SIGCSE panel on a similar topic, the panelists (which included the author) discussed a number of other alternatives including a *single project/multiple teams* alternative, where each team works on a different phase of the software life cycle, and a *multiple projects/multiple teams* alternative, where each team works independently on *all* projects. One panel member suggested a project that would definitely result in failure. The rationale behind such an alternative is that many times the students gain a great deal of experience and insight from a failed project. Yet other alternatives reported in the literature include projects that span over the course of two or more semesters.

4 Team Formation

Team sizes and team formation may have a major impact on the overall performance of the teams. There are a number of alternatives available for organizing the software engineering teams. Issues discussed in this sections only apply to "multiple team" arrangements. We will discuss three methods for selecting team members and will give references to other articles where additional alternatives are presented.

4.1 Randomly Selected Teams

One alternative is to select the members for each team randomly. While the approach is quite easy and straightforward, there are a number of drawbacks to randomly-selected team members that an instructor should be aware of. The most important drawback is the potential for unfair distribution of talent, experiences, and academic knowledge. Thus, one team may end up with excellent students while another team may include students with lesser programming experience and lower academic achievements. An inferior team cannot naturally compete with other teams and may be evaluated unfairly and with bias while a stronger team may become the instructor's "favorite."

The only merit of randomly selecting the team members, as observed by Scott and Cross [4], is that the team members may be getting acquainted with each other for the first time and therefore demonstrate a kind of "newness" or "freshness" when they work together. Such an acquaintance may in fact result in new friendships, especially among students who come from different countries and cultures.

4.2 Teams by Academic Performance

A second alternative is to balance teams by the academic performance and achievements of the team members. The objective is to make the teams relatively equivalent in terms of experience, knowledge, maturity, and performance. Selection of the team members may be based on such factors as grade point average, present classification, work experience, and courses taken. The team member selection process thus may be quite complicated and requires access to students' profiles that may not readily be available. One way to prepare a usable profile is to distribute a special "student profile" form at the beginning of the semester and ask students to provide the relevant information.

The most important merit of the above approach is that the teams will be almost "equivalent" in terms of their talents and strengths. Of course, it is not always possible to ensure absolutely "equivalent" teams.

4.3 Students Forming Teams

A third alternative (used frequently by the author) is to delegate the team formation responsibility to the students. Usually, students who have known each other previously and who have worked together in the past group themselves in the same teams. When students are teamed with individuals who they have known, their performance is usually maximized. Scott and Cross [4] enumerate additional advantages in this approach:

1. Since members of the team already know each other, little effort is needed by the instructor to get them started.
2. Since team members already know each other and are familiar with each other's strengths and weaknesses, there tend to be less friction among them. Actually frictions may occur but they are usually solved by the team members instead of reporting them to the instructor.
3. The team members are usually more familiar with each other's schedule, are more understanding of each other's circumstances, and thus may have less difficulty finding meeting times.
4. The team members may feel more pressure "not to let down one's friend" and thus may attempt

to resolve any issue that arises with the help of each other.

Scott and Cross [4] also recognize two potential drawbacks to student chosen teams: (1) the collective capabilities of teams of "people you already know" may be limited, and (2) the approach is not quite characteristic of the industrial teams whose members you do not always know. The authors discuss other selection methods, including the uses of "psychological profiles."

In the software engineering courses that the author teaches, a combination of the latter two alternatives is used. Students are asked to form teams to do a group project. The size of each team is usually kept uniform (e.g., 4–5 members per team, and usually 4–5 teams in each class). Because smaller groups teach fewer lessons about teamwork, sometimes larger groups of 5–6 are formed to make students face the challenging issues of project management. The total number of students in the class plays a role on the team sizes.

The students are encouraged to form their own teams of compatible individuals. However, the instructor will attempt to ensure that the talents and experiences between the teams are balanced. To avoid unfair or unbalanced distribution of knowledge and experience among teams, it is sometimes required to move a student from one team to another just to maintain parity among the teams. Each team selects a "team leader." The team leader is responsible for coordinating the activities of the other team members as well as communicating with the instructor. The team leader is also responsible for final technical decisions as well as making sure that everyone does his or her share of the work. The projects will take most of the semester, with major write-ups at three week intervals. There will also be a formal oral presentation and a final demonstration of the completed project at the end of the semester. The instructor should stress on the formality and professionalism in each presentation and ask all team members in a team to participate and contribute to their own team's presentation.

5 Project Selection

There are two choices in selecting a software engineering project: (1) a realistic project from the local business community or (2) a typical "academic" project (e.g., a simplified airline reservation system, a student registration system, a library system, etc.). There are advantages in and problems with either of these two approaches which are explained below. Although there may not be absolute solutions for all the problems, it is important for the software engineering instructor (as well as the students) to be aware of these problems in advance so that they can cope with them more effectively should they arise.

5.1 Realistic Projects

A realistic project from a real customer may be chosen. Care must be put into selecting a project of an appropriate size to ensure that the project can be completed within a semester and that the development process does not get out of control.

A real project will provide the students the experience of tackling a real problem and of formulating their understanding of the requirements (instead of being given artificial and/or trivial statements of the requirements). Students will have to deal with a real, external client, and will realize how important it is to have good communication skills to interact with the client (and users) who are not sophisticated in the use of computers and who are often ambiguous and unclear as to what they really want. Thus the students will gain important real-world experience while interacting with their client and will realize that no amount of lecturing is as effective as real life episodes. Furthermore, the presence of an external client may provide additional incentive for the students to do a professional job and to develop their own self-confidence in building a system for real use. At the same time, the students will improve their understanding of the software development process and will develop a sense of accomplishment.

Although the 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, the 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 [1], the instructor must ensure that the customer understands that the project may not be completed on time, and that if the project is completed, the 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 allowing the customer (or the students) to change the requirements. In fact "frozen requirements" and firm milestones are important pre-conditions for a successful project. It is also equally important for the students (and sometimes, the client) to know what the students' obligations are with respect to the course as a whole (i.e., the students also have to study, take exams, etc.)

5.2 Non-Realistic Projects

As an alternative to a real project with a real client, the instructor may choose to provide the students with a software problem definition, describing the desired functionality of a system (typical examples may include: a registrar system, an airline reservation system, and so forth). 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 rather artificial.

The primary advantage of such projects is that the students will no longer have to make off-campus trips and would no longer have to face issues discussed earlier. However, the 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 oversimplified or unnecessarily too complicated. In addition, if the instructor requires special feature or mandates a specific approach to the construction of the software, the students may feel that those feature have been contrived to illustrate some "academic" point and are not necessarily important when developing a real project.

A major compromise between the two approaches discussed above is to select a software project from within the campus (preferably within one's own department or college) where the client will understand the "voluntary" nature of the work, its pedagogical implications, and can understand that the project may not be completed (on time).

6 Concluding Remarks

Team projects in software engineering courses are essential as they provide practical exposure to many of the concepts discussed in the textbook or presented by the instructor. Our objective in offering a project-intensive, team-oriented software engineering course has been to teach our students some of the practical issues that face software engineers in industry, to provide them an opportunity to gain team experience during various phases of the software development process, and to expose them to project management issues.

We have identified a number of important organizational issues associated with offering such a course and have presented our "guidelines" for avoiding or minimizing potential problems. It is true that we spend a great deal of time and effort in administrating a team project and have to face many demanding challenges. But the benefits gained by our students definitely outweigh the difficulties in offering the course, especially when we hear our employed students express their appreciation for the useful experience they have gained.

Acknowledgments. The author wishes to thank Evans Adams, Donald Gotterbarn, Linda Northrop, and Stuart Zweben for the valuable insights they provided during a 1994 ACM SIGCSE panel. Special thanks go to Renée McCauley for organizing and moderating the panel, and for commenting on an earlier version of this article. This work was supported in part by a UCAT grant from the University Committee on Advancement of Teaching, and a UCR grant from the University Committee on Research, University of Nebraska at Omaha. An extended version of this article has been submitted to *Computer Science Education.*

References

[1] J. Comer, T. Nute, and D. Rodjak. Some observations on teaching a software project course. In R. Fairly and P. Freeman, editors, *Issues in Software Engineering Education*. Springer-Verlag, 1989. (Proceedings of the 1987 SEI Conference on Software Engineering Education).

[2] H. Etlinger. A retrospective on an early software projects course. *ACM SIGCSE Bulletin*, 22(1):72–77, 1990. (Proceedings of the 21st ACM SIGCSE Symposium on Computer Science Education).

[3] M. Moore and C. Potts. Learning by doing: Goals and experiences of two software engineering project courses. In J. Díaz-Herrera, editor, *Software Engineering Education*, LNCS 750, pages 151–164. Springer-Verlag, 1994. (Proceedings of the 7th SEI Conference on Software Engineering Education).

[4] T. Scott and J. Cross II. Team selection methods for student programming projects. In R. Ibrahim, editor, *Software Engineering Education*, LNCS 895, pages 295–303. Springer-Verlag, 1995. (Proceedings of the 8th SEI Conference on Software Engineering Education).

[5] N. Wirth. A plea for lean software. *IEEE Computer*, 28(2):64–68, February 1995.