Instructor's Roles in Software Engineering Course Projects

Stan Kurkovsky

Central Connecticut State University
1615 Stanley Street, New Britain, CT 060505, USA
kurkovskysta@ccsu.edu

Abstract. As a practice-oriented discipline, Software Engineering is most effectively taught by using a variety of hands-on experiences. Team-based projects where students can practice their technical and soft skills are a key feature of many software engineering courses aimed to prepare students for the realities of industrial software development. This paper surveys the objectives of software development projects in undergraduate courses and summarizes four roles in which the instructor can participate in student projects: a customer, an architect, a team manager, and a mentor. The paper is concluded with case studies illustrating the nature of these roles.

Keywords: Software engineering, instructor roles, case studies, industry.

1 Introduction

Most current textbooks covering the subject of Software Engineering (SE) do an excellent job covering many relevant topics and practices. However, students very often cannot relate to many practical aspects of SE, such as software process, professional ethics and requirements. Many practitioners and SE educators agree that students are often encouraged to learn more and become more passionate about these topics if they are presented through the prism of professional experience of the instructor and others, through a range of case studies, and, most notably, by involving students in a significant hands-on project, in which they can experience many practical aspects of SE.

As noted by Lethbridge [8], it is often unclear what percentage of instructors teaching SE courses have a "deep background in the field." Unlike many other CS courses, which can be successfully and effectively taught by any qualified instructor, SE is a discipline that often requires a special background, education and practical experience in large-scale software projects. Students in SE courses that include projects approaching industrial scale are exposed to and receive hands-on experience in many topics that range from software process, performance metrics and requirements elicitation to project management, planning and scheduling. Merely discussing these topics in class without applying and experiencing them firsthand in a project setting would have little or no impact on students because they often have problems relating to these practice-oriented concepts [1]. In project-based courses, students have a

chance to experience many issues that may arise in a typical industrial project. These issues can be rarely found in a purely academic setting; these may include complex team dynamics, unpredictability of customers, and issues in communication due to technical language barriers.
This paper reviews the role of software development projects in undergraduate SE courses and identifies four roles that the instructor can play in such projects to ensure that students receive the experience comparable to that of an industrial setting. These roles are: a customer, an architect, a team manager, and a mentor.

2 Industry and Software Engineering

According to Shaw [12], "universities have long felt the tension between an internal value system that emphasizes education in enduring principles and the demands of employers who want focused training in current technology." SE is an industry-oriented discipline; therefore, SE courses need to prepare students to be ready to enter the world of industrial software development. It is equally important to involve the industry in the process of SE education [6]. For example, graduates currently working in the industry and their employers often provide valuable feedback on the knowledge and skills acquired during their education.

A series of interviews and surveys conducted with a wide range of industry executives shows that the industry values a wide variety of skills and experiences, which include teamwork skills, ability to work under stress, technical proficiency, and ability to follow a software process [1, 18]. In particular, the industry values the right combination of technical skills (programming languages, algorithms, databases, design patterns, web-centric technologies, etc.); methodological knowledge (software process models, resilient architectures, quality assurance, etc.); soft skills (communication and presentation skills, being open to and being able to provide constructive criticism, being able to handle conflicts, assuming responsibility, constant learning and acquiring new skills, being an effective leader and a follower, etc.) [9].

Educators and practitioners agree that the lecture/examination approach does not work well for teaching SE [17]. Without hands-on experience, practical concepts become difficult to apply in industrial-scale software development, which may create an artificial divide between academy and industry. Three approaches are commonly used to address this problem. *Cooperative education programs* (co-ops) [8, 10, 11] provide many opportunities for professional development of students and help strengthen the relationships with the industry. Students with a co-op experience are usually more motivated and can easily place conceptual aspects into a practical context making such topics as project management, software process and quality assurance less abstract. *Case studies* [2, 4, 14, 17] allow students to immerse themselves in many details of one or more industrial-scale software projects without many of the associated drawbacks. Case studies can be presented and analyzed by students as problem/design/solution cases, or, for case studies detailing failed projects, as an analysis of factors that led to failures and a discussion of how to avoid them. Large-scale *course projects* [6, 15, 16, 18] provide an immersive environment

allowing students to experience many challenges that they will face in the industry. Most importantly, such projects provide students with an opportunity to apply many skills and knowledge acquired in other CS courses in a single practical and complex context. Students are exposed to and are provided ample opportunities to practice soft skills (such as teamwork, presentation and communication skills) that are demanded by the industry. Industrial organizations are the best source of such projects, especially when students already have a substantial amount of experience; otherwise students may encounter a number of problems related to uncertainty and changing requirements, excessive scale and complexity of the project that would be extremely difficult to tackle within the timeframe of a single course [6, 9, 19].

3 Instructor and Practical Experiences

Many SE course projects tend to oversimplify the problem. This may lead to oversimplification of the student roles who may end up focusing mainly on the artifact rather than on the process used to achieve it. Literature indicates [13, 19] that projects offered in SE courses should be real world, large scale and complex. On the one hand, students should be able to relate to the requirements and understand them without spending much time acquiring domain knowledge. On the other hand, students should be exposed to changing and evolving requirements, which is one of the main challenges of real-world software projects. Software projects should be large enough and have enough features to engage all members of the student team while enabling them to practice some degree of specialization and experience the advantages and challenges of teamwork. Software projects should be rich enough to allow students practice all previously acquired SE concepts, bring together knowledge from other specialized CS disciplines, and apply a number of modern tools and technologies. One of the most effective ways to learn SE is to expose students to an environment that resembles an industrial setting as much as possible [3]. In order to achieve this, SE courses should expose students to a balanced combination of three components: *teamwork* involving up-to-date *technology* grounded in a solid *theoretical foundation*.

3.1 Teamwork and Project Management

Modern industrial software development projects are rarely undertaken as an individual effort. Instead, a number of people in well defined roles work in teams to produce a set of project deliverables. SE courses must put an emphasis on teamwork and socio-ethical aspects, such as efficient communication, working under strict deadlines and resource constraints, project management and delegation of responsibilities. Large software projects often involve several collaborating teams distributed geographically as far as on different continents. Despite advances and low costs of long-distance collaboration technologies, lack of interpersonal contact remains a challenge. Clear design and specification documents are a significant part of the solution addressing this challenge. There is no need to recreate or simulate a long-distance collaboration environment in a SE course project; however,

demonstrating the importance of good team dynamics and interpersonal communication is an objective that can be achieved much easier.

The Guide to the Software Engineering Body of Knowledge (SWEBOK) [4] identifies ten areas of technical knowledge that are a part of the SE field and tends to downplay the importance of social factors. This may be exacerbated by the fact that most instructors have technical background and tend to focus more on technical aspects. Failure to emphasize the importance of social aspects of SE could lead to problems with intra- and inter-team communication that could otherwise be easily solved [19]. For example, requirement revisions may not be reflected in design documents because team members did not establish a close rapport.

Instructor playing the role of a team manager can keep track of how well the team is progressing towards achieving project milestones. Our experience indicates that frequent meetings with the team and individual team members combined with a sufficient number of written progress reports and other tangible artifacts helps keep the team on track to meet the deadlines and offers ample opportunities to discover any potential tensions within the team. The instructor should identify a number of milestones and corresponding artifacts; in such a structured environment, students will be able to focus better on achieving each milestone. Close and frequent contact with the team and individual members allows the instructor to emphasize the importance of clear communication and its impact on the success of the project.

3.2 Theory vs. Technology

Typically, students acquire a broad background from a number of core CS courses giving them a solid foundation in programming, data structures, networks, databases, etc. Many CS educators argue that curriculum should emphasize the theory and basic principles instead of focusing on the latest buzzwords or whatever happens to be the hottest technology du jour. They argue that despite rapid technological advances, basic principles remain the same; for example, if you are proficient in one programming language, another language can be picked up easily. On the other hand, industry often criticizes CS programs for not preparing graduates with cutting edge technological skills. In an industrial setting, the specifics of technological solutions are often determined by the requirements of the given project or demands of the respective customer. The resulting choice of technology and tools may not coincide with the preference of the developers.

In project-based SE courses, students can be trained to anticipate the possibility of having to acquire new technological skills. For example, if a project team has a solid background in a particular area (e.g. databases), the team can be asked to work on a project with an emphasis in that area, but using a technology with which the team has little or no previous experience.

3.3 Software Process

Students entering a SE course may have good experience with the waterfall software process model. But often students simply follow an ad hoc process when they

immediately begin coding and skip analysis and design altogether. This approach might work for simplistic projects often encountered in CS I and II. As a result, many students associate software process with simply coding. While there is no doubt that SE courses need to emphasize the importance of following a well-defined software process, it is equally important not to overstress it. As an opposite extreme, students may get the impression that the primary objective of a software process is to generate a lot of documentation. For example, using Rational Unified Process in a SE project may have such a result. Students need to understand that a high quality outcome is achieved neither by focusing on coding, nor by emphasizing the paperwork; on the contrary, one is worth very little without the other. Achieving software quality is a dedicated effort and it requires a well-defined software process with a clear quality assurance plan. Typically, undergraduate students understand the importance of quality, but often have problems grasping the concept of using a process to achieve it.

Real-life software projects may often be too demanding and complex to be successfully completed within an undergraduate course [19]. Students may have difficulty grasping the scope of the project and underestimating their abilities to complete the work given the time constraints. Students are often confused by the scale and the combination of skills and tools coming from a diverse range of CS topics that need to be combined and applied together in order to bring a given project to a successful completion. Also, students sometimes have difficulty relating theoretical principles learned in earlier SE and other CS courses to a practical large-scale SE project.

Instructor has a crucial role in assisting students to make a correct assessment of the project scope and to estimate the time needed to complete different parts of the project. The instructor also has an ability to point out any potential traps in requirements or development stages. Furthermore, if complexity is exposed at the early stages of the project, the requirements may be easily scaled down without compromising the overall functionality of the project or its timely completion. The instructor playing the role of a customer may often directly specify a portion of requirements and, therefore, help students avoid many traps and obstacles by hiding the most difficult aspects of real-world software projects where requirements are never known upfront and change frequently afterwards [7].

3.4 Roles that The Instructor Can Play

As outlined above, the instructor in a project-based SE course may play several roles.

As a *customer* who contracts a student team to implement a particular project, the instructor may provide a verbal or written description of the project to be completed. It is the students' responsibility to conduct the requirements elicitation interviews and to prepare formal requirements specification for subsequent validation with the customer and possible adjustment. The instructor may act as a customer during the acceptance testing throughout the project to make sure that the team is on the right track at every iteration of the project. The instructor acting as a customer can also act as a proxy if the project has been offered by an industrial partner, or some other department or organization on campus.

As an *architect* initially specifying the overall structure of the solution, the instructor can help students choose the right software architecture. This is a very important choice that has to be made early because it impacts the overall flow of the project. Later architectural changes can be extremely costly and can significantly jeopardize meeting the deadlines. Most likely, students do not have experience in making such architectural decisions and, therefore, it is crucial for the instructor to provide some guidance in this important process.

As a *team manager* concerned with risk management, the instructor needs to pay attention that the team efforts are not jeopardized due to a potential lack of students' soft skills and that these skills are built up as the project progresses. Although it is the students' responsibility to create all artifacts emerging from the project, the instructor can help the team adhere to the schedule by providing feedback based on the inspection of all produced artifacts. In this respect, the roles of the team manager and the customer may somewhat overlap because the same person will be providing feedback to the team from the technical point of view (artifact inspection), as well as from the customer point of view (verification of product implementation at each iteration).

In addition to the traditional role of giving lectures, the instructor needs to be a *mentor* offering guidance to each team and individual students addressing their specific needs and answering questions unique to their projects and the responsibilities in the context of the project. Furthermore, the instructor is uniquely positioned to advise individual students on how to resolve potential conflicts within a team. Such an advice given at the right time could make a difference between completing the project on time and allowing interpersonal tensions completely ruin any chance of collaboration between the team members.

4 Case Studies

Two case studies illustrate how instructors acted in different roles worked with students on software development projects offered in an undergraduate SE course. Acting as customers for both projects described below, the instructors provided students with verbal explanations of the problems. In each project, several follow-up interviews followed when students provided the customer with a version of user requirements written in natural language reflecting their current understanding of the problem.

4.1 SurveyCentral

Rationale: The current method used for end-of-semester course evaluations is cumbersome and dated. A question sheet is handed out to students in class who record their answers on a separate form. The forms are then placed in an envelope and returned to an administrative office where they are scanned into a database used for reporting the results. Students often choose not to answer all of the questions or neglect to even return the forms. In addition, comments where students elaborate on specific points can only be written on the question sheet and, therefore, are not

included in the database. Incomplete or uncollected data can skew the results and thwart the efforts to perform comprehensive analysis designed to improve instruction and the curriculum at the university. The objective of this project is to design and implement an application that would allow faculty to create and distribute course evaluations in a secure environment. The creation of the survey will be dynamic in its ability to produce an unlimited number of content areas with questions using a variety of answer techniques (true/false, multiple choice questions, and freeform text). The evaluations will be completed by students using a web browser while remaining anonymous. Statistical charts and reporting, in addition to the ability to download responses in a variety of formats, will be available to faculty.

Students were obviously familiar with some aspects of the described problem since they fill out such evaluation forms every semester. However, the instructor acting as a customer could also present them with an additional insight into the problem and explain why the current system needed to be updated. The most important objectives were to improve student participation in the surveys, especially encouraging them to provide written feedback, which could be easier for them to do if the surveys were offered online and available over a period of several days as opposed to the paper-based surveys which are handed out in class and must be completed within a short period of time. Statistical analysis based on the current system was very slow and not always reliable, which would not be an issue in an online system.

While playing the role of the team manager, the instructor emphasized the importance of a careful requirements analysis. The project was actually more complex than it appeared because of the capability to construct new surveys with several different types of questions. As a team manager, the instructor helped students lay out a schedule that emphasized the requirements elicitation and refinement, and design of a flexible system architecture, which helped streamline the development and complete the project ahead of time. As a team manager and a mentor, the instructor also helped resolve a conflict between two students who had a significant disagreement regarding the style and the level of detail of the requirements document; the same pair of students also clashed over the choice of technological solutions for the project implementations. Without taking sides, by picking a solution proposed by different student in each of the disputes, the instructor was able to diffuse a conflict threatening to stall the project.

As an architect, the instructor was able to guide students in the right direction when choosing the overall system architecture and thus avoiding the risk of costly revisions of the architecture at a later moment. In particular, students needed help with the relational database schema that would reflect complex relationships between questions and their types, instances of surveys and their templates, and organizing data for easy collection of statistics. Another important aspect that needed to be considered as a part of the system requirements, is how to ensure the student anonymity, allow each student submit evaluation only once and, at the same time, allow only the students enrolled in the course to submit an evaluation for that course. Unique keys could be generated and distributed to individual students who would later use them to log into the system and submit their surveys. However, to simplify the solution and to alleviate some privacy concerns, the instructor recommended using an approach when the keys generated by the system are printed and distributed to the students on paper.

4.2 TextTrader

Rationale: When the semester is over, many students would like to sell their textbooks. Typically, they have two choices: sell them to the university bookstore at a very low price or try to find another student on campus who would need the same textbook. A web-based system that can help students wishing to sell their books to connect with students wishing to buy those books could become a single point of contact resolving the problem of establishing a connection between buyers and sellers. Such a system would not support any financial transactions; its functionality will be limited to listing books by sellers, searching for books by buyers, and allowing buyers to contact sellers via email.

In this project, students had as much end-user perspective as the instructor acting as a customer. Therefore, the instructor's primary objective was to play the "devil's advocate" offering unusual but plausible scenarios of interaction with the system. For example, is it possible to cross-list a book among several disciplines, such as Computer Science and Management Information Systems? What happens if a visitor finds some listings objectionable? What to do if a listed textbook doesn't sell for a long period of time? If the book is sold, how is the listing removed from the system? From playing out such scenarios, students were able to derive a set of clearly specified functional and non-functional requirements that outlined the properties and behavior of the system in both standard and unorthodox scenarios.

The team had no problem quickly developing a simple and efficient overall architecture of this multi-tiered system. As an architect working on this project, the instructor suggested a few technological innovations to make the project more interesting and challenging. To facilitate the book search, the system uses Amazon.com API to help sellers verify the correctness of their listings by retrieving complete book information, as well as the image of the book cover.

The team chose to use the Struts framework as the platform for implementing this project because they felt that it offers the right selection of features needed by this project and because they were eager to learn this new technology. Concerned with the chance of encountering a steep learning curve, the instructor acting as the team manager advised the students about the risks and helped the team create a project schedule that would accommodate the possibility of abandoning Struts and using an alternative, with which the team was much more experienced. Thankfully, the team had no serious problems acquiring Struts development skills and delivered the project on schedule.

The student team working on this project had a very good team dynamics and seemed to have no visible conflicts that could jeopardize their successful work. Therefore, the role of the instructor as a mentor was focused primarily in consulting the team and individual students on general and technical matters pertaining to this project.

5 Conclusions

Software Engineering is one of Computer Science disciplines that have a strong grounding in practical applications. For students who envision their future in the industrial software development, it is extremely important to gain classroom experience in real-world large scale complex software projects, in which they are exposed to many aspects of teamwork and use current technology supported by the students' background in theory. In this paper, we surveyed a number of important goals of project-based SE courses aiming to achieve such an objective. These goals are best reached when the instructor participates in student projects in several capacities, which include being a customer, an architect, a team manager, and a mentor. By playing each of these four roles, the instructor exposes students to a reasonable amount of experiences and features typical to software projects in the industrial setting while, at the same time, minimizing the associated risks. To illustrate this approach, this paper presented two sample projects completed by student teams and discussed the specifics of the instructor's involvement in each of the aforementioned capacities. Very positive comments from students participating in these projects and from the employers of our graduates are strong indicators of the success of such an approach.

References

1. Carver, J., Vaughn, R.B., The importance of experience with industry in software engineering education, 19th Conference on Software Engineering Education and Training, pp. 19-23, IEEEE CS, 2006.
2. Burge, J., Troy, D., Rising to the Challenge: Using business-oriented case studies in software engineering education, Proceedings of the 19th Conference on Software Engineering Education & Training, pp. 43-50, IEEE CS, 2006.
3. Duim, L. van der, Andersson, J., Sinnema, M., Good practices for educational software engineering projects, Proceedings of the 29th International Conference on Software Engineering, pp. 698-707, IEEE CS, Washington DC, 2007.
4. Garg, K., Varma, V., A study of the effectiveness of case study approach in software engineering education, Proceedings of the 20th Conference on Software Engineering Education & Training, pp. 309-316, IEEE CS, 2007.
5. Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE CS, http://www.swebok.org.
6. Jaccheri, L., Morasca, S., On the importance of dialogue with industry about software engineering education, Proceedings of the 2006 International Conference on Software Engineering, pp. 5-8, ACM, 2006.
7. Jazayeri, M., The education of a software engineer, Proceedings of the 19th International Conference on Automated Software Engineering, pp. xviii-xxvii, IEEE CS, 2004.
8. Lethbridge T.C, Diaz-Herrera, H., LeBlanc, R.J, Thompson, J.B, Improving software practice through education: Challenges and future trends, International Conference on Software Engineering, 2007 Future of Software Engineering, pp. 12-28, IEEE CS, Washington, DC, 2007.
9. Lilienthal, C., Bleek, W.-G., Schmolitzky, A., Transferring experience from software engineering training in industry to mass university education - the big picture, Proceedings

of the 18th Conference on Software Engineering Education and Training, pp. 195-203, IEEE CS, 2005.
10. Pour G., Griss, M.L., Lutz, M., The push to make software engineering respectable, IEEE Computer, Vol. 33, No. 5, pp. 35-43, IEEE CS, 2000.
11. Reichlmayr, T.J., Collaborating with industry: strategies for an undergraduate software engineering program, Proceedings of the 2006 International Conference on Software Engineering, pp. 13-16, ACM Press, 2006.
12. Shaw, M., Software engineering education: a roadmap, Proceedings of the Conference on The Future of Software Engineering, pp. 371-380, ACM Press, 2000.
13. Stankovic, N., Software engineering for undergraduates, Proceeding of the 28th international conference on Software engineering, pp. 661-666, ACM Press, 2006.
14. Taran, G., Using games in software engineering education to teach risk management, Proceedings of the 20th Conference on Software Engineering Education & Training, pp. 211-220, IEEE CS, 2007.
15. Tvedt, J.D., Tesoriero, R., Gary, K.A., The software factory: combining undergraduate computer science and software engineering education, Proceedings of the 23rd International Conference on Software Engineering, pp. 633–642, IEEE CS, 2001.
16. Umphress, D., Hendrix, T.D., Cross, J.H., Software process in the classroom: the capstone project experience, IEEE Software, Vol. 19, No. 5, pp. 78-85, IEEE CS, 2002.
17. Varma, V., Garg, K., Case studies: the potential teaching instruments for software engineering education, Proceedings of the 5th International Conference on Quality Software, pp. 279-284, IEEE CS, 2005.
18. Vaughn, R.B., Teaching industrial practices in an undergraduate software engineering course, Journal of Computer Science Education, Vol. 11, No. 1, pp. 21-32, Routledge, 2001.
19. Vliet H. van, Reflections on software engineering education, IEEE Software, Vol. 23, No. 3, pp. 55-61, IEEE, 2006.