Increasing the Effectiveness of Teaching Software Engineering: A University and Industry Partnership

Aldo Dagnino
ABB Corporate Research,
Raleigh, NC. USA
Research Area Software
aldo.dagnino@us.abb.com

Abstract

Software Engineering is a complex topic because it encompasses multiple disciplines such as team work, effectively managing change in organizations, understanding technology, understanding software development and its lifecycle, understanding the role of change agents, and also having a good business sense. A method derived from the collaboration between North Carolina State University and ABB, brings diverse techniques that simulate an industrial environment for teaching a senior level Software Engineering course. Eleven elements that have been incorporated to enrich the Software Engineering graduate course are described. The paper also shows how the progressive incorporation of the elements have resulted increased student satisfaction.

1. Introduction

Although universities put a lot of effort into preparing their Computer Science students for the software industry, a great number of employers complain about low level of readiness of university graduates that can solve problems in this industry. Employers are usually unsatisfied with the experience and knowledge that recent graduates possess. This is evident in many areas of Software Engineering such as project planning, software estimation, requirements development and management, software architecture and design, peer reviews, and verification and validation, among others. Moreover, essential skills such as effective team work, communication and facilitation skills, continuous software process improvement, and managing organizational change are often lacking in recent graduates. Also, recent graduates that start work in industry are often surprised on how many elements from the Software Engineering field they were not exposed during their academic studies. One way to bridge this gap is to develop a method that includes not only teaching software engineering theory but also allows students to practice the principles in a simulated industry environment and develop skills useful to the discipline.

Many universities have developed courses in Software Engineering. However, many authors coincide that this is not sufficient to provide the software industry with “ready-to-go” software professionals [4][5][6][7]. Students often regard the methods and theory taught in software engineering courses as abstract and purely academic concepts. Without experiencing their practical impact on realistic scenarios, students rarely develop a deep understanding or appreciation of important ideas and their actual implementations in software engineering [1]. Mirian-Hosseinabadi et al. [4] propose an experience based approach for teaching undergraduate software engineering courses at universities. The authors address a main weakness in teaching an undergraduate software engineering course which are the lack of consideration of the engineering concept in the software development lifecycle and neglecting practical application of engineering principles and disciplines in a software product. The authors illustrate how to use an experience based approach for teaching the software engineering course that can imply the application of engineering disciplines in the software products at different stages of software development process.

The course on Software Engineering is important for both the undergraduate and postgraduate programs of computer science and has as a primary objective to make the students industry ready by exposing them to the processes and practices of life cycle activities of software development. Desai and Joshi [2] explain that traditional approaches to teaching the course on Software Engineering have not had the desired impact on learning the discipline because of several factors, which include: (a) typical course instructors mostly lack exposure to industry practices; (b) the course material course fails to establish an appropriate context as the case studies referred are alien to the students; and (c) the course assessment focuses mainly on memory oriented questions. "Thus teaching the course on Software Engineering has got reduced to monotonous lecturing, in the absence of experience and case studies".

2. Software Engineering in Practice

A method to teach Software Engineering students important skills they will need in their future professional environment was developed and is currently being utilized in the Software Engineering Graduate course (CSC 510) in the Department of Computer Science at North Carolina State University. A summary of this method is shown in Fig. 1. The method in Figure 1 has four main elements : (i) teaching approach utilized to teach the course; (ii) artifacts that the students use to learn the subject; (iii) activities that the students develop during the course to learn the subject; (iv) additional non-traditional subjects relevant for the practice of Software Engineering that are taught to students. Section 3 discusses the non-grayed elements in the method that are considered "non-traditional" in a curriculum.

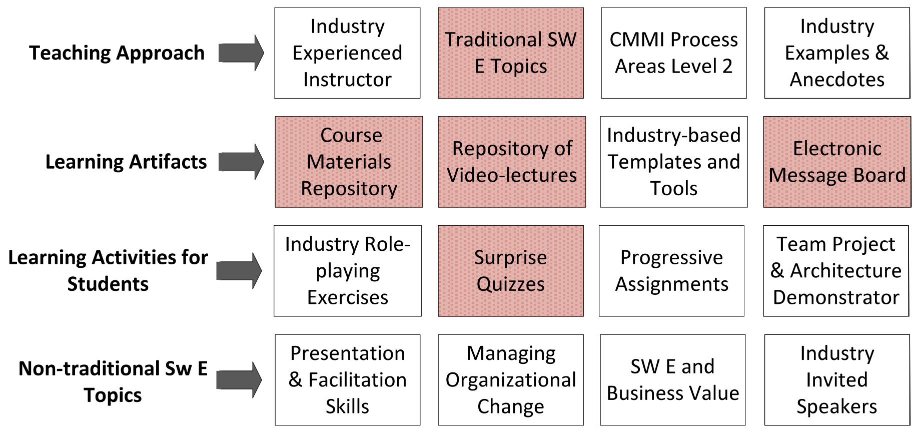
Teaching Approach
Industry Experienced Instructor
Traditional SW E Topics
CMMI Process Areas Level 2
Industry Examples & Anecdotes
Learning Artifacts
Course Materials Repository
Repository of Video-lectures
Industry-based Templates and Tools
Electronic Message Board
Learning Activities for Students
Industry Role-playing Exercises
Surprise Quizzes
Progressive Assignments
Team Project & Architecture Demonstrator
Non-traditional Sw E Topics
Presentation & Facilitation Skills
Managing Organizational Change
SW E and Business Value
Industry Invited Speakers

Figure 1. Software Engineering Teaching Method.

3. Non-traditional Elements of Software Engineering Curricula

Several elements shown in Figure 1 are currently present in most curricula of Software Engineering courses. Those elements are identified in Figure 1 as “greyed” boxes. Other elements have been added to the current Software Engineering course being taught at North Carolina State University. Based on feedback from students, adding these elements has resulted in a better course where students seem to better understand the Software Engineering discipline. The remainder of this section briefly describes these elements starting from the top layer and moving down to subsequent layers and going left to right. A comparative evaluation of the course before these elements were introduced in the curriculum and after is presented in Section 4 of this paper.

3.1. Industry Experienced Instructor

Having a software industry experienced instructor teaching the Software Engineering class has helped blending the academic concepts of the course with how they are put into practice in industry. An industry experienced instructor brings into the course important industry-related elements such as: (a) playing different industry roles during class exercises and progressive assignments, such as customer, project manager, product manager; (b) guiding students in their learning activities and also role playing; (c) developing and providing industry-oriented templates; (d) providing industry anecdotes; (e) and bringing an understanding on how the software industry works and values.

3.2. CMMI Process Areas Capability Level 2

The CMMI Level 2 process areas are fundamental in a software development organization to institutionalize a managed development process. These process areas include project planning (PP), project monitoring and control (PMC), requirements management (REQM), configuration management (CM), produce and process quality assurance (PPQA), supplier agreement management (SAM), and measurement and analysis (MA). CMMI Level 3 process areas that need to also be included in the curriculum include requirements development (RD), requirements management (RM), technical solution (TS), product integration (PI), verification (VER), and validation (VAL). Providing details on these process areas and explaining how they are implemented in industry is very important so that students get an appreciation on how mature software industries implements these process areas.

3.3. Industry Examples and Anecdotes

Students enjoy listening about industry examples and anecdotes originating from direct experiences that the instructor has lived through directly related to the topics being taught during the course. For example sharing industrial horror experiences such as last minute software deliveries with not enough verification, or not preparing risk mitigation strategies, or not managing requirements creep, etc. Sharing funny stories such as the time during an estimation session a squirrel came into the room and the project manager caught the squirrel with the waste basket. These anecdotes help students to better remember the topics discussed.

3.4. Simple Industry-Based Templates and Tools

Moving to the learning artifacts layer, students need to learn that templates and tools are essential in industry as they should embody the institutionalized development processes. Templates and tools must be simple to use and understand, bring value to its users, and help ensure the high quality of work products. It is also the experience of the author that these templates and tools must not be seen as a "burden" to the users and therefore must embrace the concepts of agile software development. It is important that Software Engineering students be exposed to these templates and tools and directly experience them, so that they will be comfortable using them in industry after they graduate.

3.5. Industry Role-Playing Exercises

Moving down to the learning activities layer, role-playing exercises help simulate an industrial environment and allow students to directly experience situations that will be confronted in their future careers. Role-playing exercises need to be illustrative and short enough to be part of the theoretical instruction during a class. Role-playing exercises include: (a) software estimation; (b) peer reviews; (c) risk identification and mitigation; (d) black-box testing; (e) software process appraisal; and (f) managing organizational change. Mixing the theory and making students break up into work groups to have hands on experiences in these exercises is very important. Comments that students make after these exercises emphasize that role playing exercises "is a great way to understand the intricacies and details of important elements in Software Engineering".

3.6. Progressive Assignments

Progressive assignments are assignments that build on each other and are done individually by each student. The objective of these assignments is for the students to individually start a project from the initial stages of the software development lifecycle and complete the assignment where a partial architecture of a software system is developed. In a semester, the optimal number of progressive assignments is four. At the beginning of the semester, students select one of three potential real-world software systems to build in the progressive assignments and four assignments are defined. First, from a high-level definition of the system (the instructor is the project sponsor/customer), the objective of the first assignment is to develop an initial business case associated with the software to be built, define a partial list of use cases, features, and market requirements. In the second assignment students develop product and technical requirements from the market requirements defined. In the third assignment students develop a project plan, with estimates, resources, timelines, risks, WBS, and skills required to carry out the project. In the final assignment students develop the architecture by defining the quality scenarios from the business drivers and requirements defined.

3.7. Team Project and Architecture Demonstrator

A team project is formed by three to four students. Each team utilizes the materials developed in the individual progressive assignments and decide what elements of their assignments are considered to implement certain functionality of the software utilizing the architecture and technology of their choice. The objective is that the team works together to develop and implement a demonstrator of their system architecture by prioritizing architecture scenarios and quality attributes defined. Also, the team is required to develop test cases that are utilized to verify the architecture and ensure that the quality attribute scenarios are properly implemented. For example, the system may need to perform some analytics with certain level of performance, or the system must be able to receive information from sensors and be able to scale to a certain number of sensors without degrading its performance, etc.

3.8. Presentation and Facilitation Skills

Moving down to the non-traditional Software Engineering topics, students are taught throughout the course presentation and facilitation skills through the class exercises, role-playing exercises, and ultimately the final project presentations. An industry experienced instructor, is expected to have played many roles in a software development organization and this experience can help her/him to guide the students in playing their roles properly throughout the course. When students begin their professional careers, they will find that presentation and facilitation skills are essential to be successful.

3.9. Managing Organizational Change

For an organization to have a successful continuous process improvement culture it requires managing organizational change very effectively [3]. It is important that students learn models to assess the readiness of an organization to implement change at a given point in time. Students then learn the different roles and their behaviors in a change situation that the sponsor, change agent, people affected by the change, advocates of change, and detractors of change play. Topics such as overt and covert resistance are discussed. Moreover, strategies on how to improve the readiness of the organization to embrace change effectively and maintain a healthy continuous process improvement environment are discussed.

3.10. Software Engineering and Business Value

All software development organizations are driven by increase in profits and reduction of costs. Hence, all Software Engineering processes, assessments, improvement programs, development, and activities are driven by financial analyses. It is essential that Software engineering students understand the financial implications of the discipline. For example, students are taught how business drivers influence the functionality and architecture of a software system, how a software process improvement is linked to a potential business benefit such as cost of poor quality reduction, how lifecycles are linked with potential economic benefits, what is the financial “sweet spot” when developing a verification plan, etc.

3.11. Industry Invited Speakers

The author has observed that students gain invaluable experience and appreciate that industry invited speakers make presentations throughout the course. This allows the students to have a variety of points of view and enrich their learning experiences. Having three to four invited speakers from industry is a good number in a semester course.

4. Historical Instructor Evaluation

At the end of the semester, the university conducts a survey among the students that took the class to evaluate the instructor and the learning value provided during the course. The author (who has several years of industry experience) has been teaching this course since 2009 to date and will teach the course in the winter of 2014. Table 1 shows the evaluation (1-5, being 5 the highest) of four key evaluation criteria that have been tracked until 2013.

Table 1. Historical Instructor Evaluation Teaching Sw E Course

Selected Evaluation categories	Class Mean 2009	Class Mean 2010	Class Mean 2011	Class Mean 2012	Class Mean 2013
Overall this course was excellent	2.8	3.9	3.8	4.2	4.5
This course improved my knowledge on the subject	3.1	4.5	4.6	4.2	4.6
The course assignments were valuable aids to learnin	3.0	4.1	4.1	4.3	4.4
Overall, the instructor was an effective teacher	2.9	3.9	3.9	4.2	4.5

As observed in in Table 1, the first time this author taught the course, the evaluation on those criteria were by far the lowest. As the author began in 2010 to progressively incorporate the elements described in Section 3 the student evaluations improved.

5. Conclusions

Teaching Software Engineering is difficult as traditional courses focus on teaching theoretical aspects of the discipline with not much practical application. This paper presents an approach to teach a one semester graduate course in Software Engineering with emphasis on industrial experiences and how the theory applies to practice. This paper presents a method that has been utilized at NC State University to improve how the graduate level Software Engineering course is taught. The method includes new elements that have been incorporated into the course. An improvement in the instructor evaluation from the students has been shown as these elements have been incorporated into the course curriculum.

References

[1] D. Dahiya, D., “Teaching Software Engineering: A Practical Approach “, ACM SIGSOFT Software Engineering Notes, March 2010, Volume 35 Number 2,

[2] P. Desai, and G. H. Joshi, “Activity Based Teaching Learning in software Engineering”, IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA), 2012, pp. 1-6.

[3] Kautz, K., Westergaard Hansen, H., and Thaysen, K., “Applying and adjusting a software process improvement model in practice: the use of the IDEAL model in a small software enterprise”, ICSE’00 Proceedings of the 22nd International Conference on Software Engineering, 2000, pp. 626-633.

[4] S. Mirian-Hosseinabadi, Z. Aghakasiri, S. Alireza, P. Delfani, and M. Ghandehari, “Emphasizing Experiences in Teaching Software Engineering Courses”, 2nd International Conference on Education Technology and Computer (ICETC), 2010, pp 149-153.

[5] D. L. Pamas, "Software Engineering Programs Are Not Computer Science Programs", IEEE Software, 1999, vol. 16, pp. 19-30.

[6] J. Sukhodolsky, "Teaching Software Engineering to Undergraduates", In Proceedings of the International Conference on Information Systems and Engineering, 2003, pp. 165-173.

[7] A. Van der Hoek, D. G. Kay, D. J. Richardson, "Informatics: A Novel, Contextualized Approach to Software Engineering Education", Lecture Notes in Computer Science, 2010, vol. 4309, pp. 147-165.

[8] C. Wohlin, B. Regnell, "Strategies for industrial relevance in software engineering education", The Journal of Systems and Software, 1999, vol. 49, pp. 125-134.