SE and CS Collaboration: Training Students for Engineering Large, Complex Systems

Mohammad Nauman[1] and Muhammad Uzair[2]
[1]City University of Science and Information Technology, Peshawar, Pakistan.
[2]NWFP University of Engineering and Technology, Peshawar, Pakistan.

Abstract: *Today's software industry is characterized by fast growth and diversity. To engineer software in such an environment, software engineers are required to work with large teams and handle large complex systems, involving common off-the-shelf components, open source software and outsourced resources. This poses a serious challenge for software engineering institutions.*
In this paper, we outline a framework for collaboration among computer science and software engineering programmes within a university with the goal of training students for engineering large, complex systems. We chart a three phase layout for the framework in which students of both programmes work together to simulate the industry's practices by designing, building, integrating and testing a large, complex system.
We consider the issue of evaluating students in such a framework and give alternatives for certain variables so as to fit the framework in different environments.

Keywords: Software engineering, computer science, education, complex systems

1. INTRODUCTION

Software engineering education has come a long way and is no longer in its infancy. There are several forums working on the curricula and pedagogical issues related to software engineering education. Among these are the *SIGCSE Technical Symposium on Computer Science Education* and the *Conference on Software Engineering Education and Training*. Detailed layout of program and course contents have been developed by many universities.

Despite all of this, there is still a large gap between the techniques taught to the university students and those practiced by the industry to fulfil the requirements of their clients [1]. The problem is multifaceted.

The software industry requires software engineers to be able to build very large, complex systems in a short span of time. Not only that, it also requires them to build software using outsourced human resource, common off-the-shelf components or even through modifying open-source parts developed elsewhere [2]. Software engineering, in other words, is moving towards its *engineering* aspect faster than ever.

Students studying software engineering are taught the latest techniques and practices but upon exposure to the industry, they find that other techniques are employed there. Software practitioners on the other hand, employ techniques they learn through experience as those they learned during their education are of little use in the industry. This leads not only to a waste of teaching time but also means that sub-standard techniques are employed by the industry.

In this paper, we identify the sources of this problem and discuss some techniques employed by other researchers to address it. We then develop a framework as a solution to this problem and compare it with the related techniques. Finally, we give an insight into the future prospects of this research.

2. PROBLEM OVERVIEW

2.1. The problem of size and time

The difference between software engineering taught in educational institutions and that practiced in the industry is mainly that of size [3]. Most educational institutes carry out a final project in which students are expected to engineer a software project. This project is usually kept small enough to be completed within the semester's time. Following such a technique means that students are rarely exposed to large and complex systems and thus fail to appreciate the difference in techniques required for small, fairly simple and large, complex systems. So, through the problem of size, we identify that there is also an issue of available time.

2.2. Gap in industry and academia

Several studies have been carried out to identify the solution to this problem. The gist of these is that there is a gap between practices of the industry and instruction in the academia [5]. This gap needs to be filled either by bringing practitioners to the students or by exposing the students to the industry through internships. Both of these solutions are difficult to implement – again because of the issues of complexity and available time.

Another solution is to simulate the environment of the industry within the educational institutions through carefully designed programmes.

In related works, we discuss some of the techniques previously employed to bridge this gap.

2.3. Diverse nature of skills required

Third facet of the problem is the diverse nature of skills required to become a software engineer. [4] identifies skills in which a good software engineer requires expertise. These range from computer science and project management to communication and interpersonal skills. Acquiring all these skills is never easy for a student and makes it very difficult for instructors to design a course encompassing all these skills.

3. RELATED WORK

To train students for handling large, complex systems, several universities have developed strategies of diverse natures.

[3] have employed a *cross-term*, *cross-team* educational software process to educate students in engineering very large systems. The process involves students of subsequent batches on the same project starting from students of one batch who are required to produce certain deliverables in the form of software engineering artifacts. The students of next batch continue from the point in the process where the previous batch left off. Students work with all aspects of software engineering and can appreciate the complexities involved in engineering a very large, complex software system.

[5] have used a technique of involving students in a rigorous 8 semester *Software Factory* in which each student gains experience in every participatory role of the software lifecycle. These include requirements gathering, software engineering, project management, software testing and all other associated roles.

A unique solution has been proposed and implemented by [6]. The students create three games in one semester. They are allowed to work more and more independently as they progress through the semester until finally, they create the last game completely on their own. Games were chosen by the institute because of their complexity and high software engineering demands. One downside of this approach is that the students on which this technique was applied were domain experts in game development and this technique can therefore not be used for all software engineering students.

All three of these techniques try to teach students all aspects of software engineering through their projects. We propose that a distinction be made between what is being done correctly in the institutes and what is not. A framework can then be developed which focuses on the deficiencies of current pedagogical techniques to bridge the gap between academia and industry. In the rest of the paper, we develop our framework by identifying these deficiencies.

4. CS AND SE COLLABORATION TECHNIQUE

4.1. Background work

The problem of size and complexity of the nature of software engineering education engineering requires that such a distinction be made regarding what aspects of SE education are the most important. These should then be given a preference in software engineering education curricula and in the projects that students are expected to complete.

[7] carried out an extensive survey on practitioners in the industry and identified some aspects of software engineering upon which colleges and universities need to focus more. These were the aspects that the surveyed practitioners felt were most important for them but were under-emphasized in the academia. Among the top ten of these aspects were:

- Testing and quality assurance
- Maintenance
- Project management
- Object oriented analysis and design
- Requirements gathering

There were also some aspects which, according to the surveyed practitioners, were over-emphasized in the curricula. Among these were:

- Numerical methods
- Programming language theory
- Complexity and algorithm analysis

A study of the results of the survey leads to the conclusion that all the subjects normally associated with computer science, are over emphasized in the educational institutions while those normally associated with software engineering are under-emphasized.

The authors conclude that more attention needs to be given to the under-emphasized courses. Problem remains that these are vast subject in their own right and are difficult to be covered in full detail in the given amount of time.

From among the aspects identified as being under-emphasized in software engineering education, we focus on object oriented analysis and design, project management and testing in our framework.

Now we give the layout of our framework for collaboration among CS and SE programmes.

4.2. Separating concerns of CS and SE

Although software engineering and computer science are practically inseparable, their scope is different. A limit has to be defined as to how much a software engineer needs to know of computer science technicalities such as programming language theory and algorithm analysis. As is clear from the survey described in *background work*, software engineering education institutions need to focus more on software engineering aspects of their curricula in order to produce graduates better suited for today’s software industry.

Concerns of software engineering courses and computer science courses are separate [10] and the final projects need to reflect this separation of concerns. In the software industry, software engineers are never coders. Low level implementation is strictly the job of programmers. Software engineers need to work with programmers and programmers need to understand how to take the artifacts of software engineering processes and turn them into quality code.

4.3. A collaboration framework

In our framework, we propose the simulation of this aspect of the industry through collaboration of two separate programmes in a university offering undergraduate and graduate courses in both software engineering and computer science.

We divide the framework in three phases. Education institutions can implement each phase in one semester.

4.4. 1st phase

For the first phase of the framework implementation, a large complex system is decided upon by the instructors of both computer science and software engineering courses. The software system should be complex enough so that *only* the system analysis and design can be completed by the students in a semester's time.

The main aim of this phase is to help students become proficient in object oriented analysis and design – an important part of software engineering practices in the industry today [11].

The students of software engineering programme are given the specifications and are expected to complete the system analysis and design by the end of this phase. The goals of this phase would be for the students to:

- produce, a complete system design by the end of the phase,
- identify any common off-the-shelf components or open source components that can be reused in their system and
- give complete specifications (including functional and non-functional requirements as well as any integration specifications) for any components that they need developed by the programmers.

The artifacts of this phase become the inputs for the second.

Computer science students are not involved in this semester.

4.5. 2nd phase

In the second phase of the framework, the specifications developed in the first phase are given to computer science students. Only the components that need to be developed or modified are given to the students who work throughout the phase to complete the components. The goals of this phase for the students will be to:

- create the components exactly according to the specifications,
- modify any open source components that need to be modified for reusability and
- create interfaces for future integration of the components

The components developed in this phase will be integrated to create the whole system in the next phase.

The students of software engineering are not involved in this phase.

4.6. 3rd phase

In the third phase, students of software engineering and computer science programmes work together for the integration of the components into the system. In this phase, the emphasis will be on:

- Team work
- System integration
- System testing
- Quality assurance

Each team of computer science programme is assigned to a team of software engineering programme. They work together to produce the final project by the end of the phase.

5. EVALUATION AND ASSESSMENT

Assessment of students in the first and second phase is based on the artifacts produced and on achievements of goals specified.

Evaluation of software engineering students in the third phase is based on compliance with requirements and quality of the software. Both the functional and non-functional requirements are considered. The evaluation of computer science students in this phase is based on technical aspects of their work such as correctness, performance and modifiability of produced components etc.

6. ISSUES ADDRESSED BY THE FRAMEWORK

The framework we have proposed addresses the following issues:

- *Project management* is addressed through having students create a complex system through almost all the stages of design and development.
- *Environment of the industry is simulated* by having students collaborate not only with their team members but also with other teams.
- *Faculty peer collaboration* is facilitated by having instructors of computer science and software engineering work together.

An important fact to be noted is that our framework spans only three semesters. The rest of the coursework can (and should) focus on aspects and standards of software engineering principles. This would ensure that the students learn to cope with the industry while still learning the principles that aid the engineering of quality software [8].

7. ALTERNATIVES AND OPTIONS

The layout described above is only a general view of the framework. The framework can be modified to suit the requirements and environment of a specific university. We give some of the variables which can be changed in the framework.

- The number of semesters can be increased to accommodate requirements engineering in the activity.
- Instead of providing students with requirements for a “fake” system, the industry can be involved and real problems can be solved through this framework. Requirements can be articulated by involving community partners in the project [9].
- Students of different universities can be brought together in a joint venture by the universities to further diversify the nature of the activity.

8. ISSUES

There are certain issues associated with the use of this framework:

1. *Requirements gathering and maintenance, two important aspects of software engineering are not covered at all.* We believe that requirements engineering is a vast subject and would require far more time than our framework in its basic form can accommodate. This issue can, however, be addressed if the span of the framework is increased as discussed in *alternatives and options*. Maintenance on the other hand is one of those aspects of software engineering which are very difficult to teach in software engineering courses because of its nature in terms of extended periods of time. We cannot accommodate maintenance in our framework but believe that it is a separate concern which should be addressed elsewhere.
2. *Inputs of one team based on outputs of another team*. This is an important point and should be kept in mind during evaluation. It is however, our approach that to simulate the actual environment of the industry, such variables should not be eliminated. In an industry, a software engineer has to deal with a lot of variables among which the skill of programmers is an important one. Similarly, since programmers have to deal with specifications provided by software engineers in the industry, this simulation is an accurate one for them too.

9. Vision for future

This framework is an outline of the plan we have for collaboration among students of computer science and software engineering. We plan on working on the specifications of the framework and identifying more variables in the following areas:

1. Identifying a large scale system which can be built in three semesters' time span while still being complex enough to give a taste of today’s software industry.
2. Clarifying variables for evaluation of the students throughout the framework.
3. Implementing the framework on graduate/undergraduate students of City University to evaluate the success of the framework.

10. CONCLUSION

The highly dynamic environment of today’s software industry requires software engineers to work with large teams and handle large complex systems involving common off-the-shelf components, open source software and outsourced resources. This poses a great challenge for the educational institutions as there exists a gap between practices of the software industry and techniques taught by the academia.

In this paper, we have identified some of the sources of this problem and have outlined a framework which can be adopted to address these issues by creating collaboration among students of software engineering and computer science.

Our three-phase framework complements the fundamental traditional course work by providing the students an opportunity to enhance their skills through the process of engineering a large, complex system. It will enable the students to realize the nature of work, environment, skills and techniques employed in practice by the software industry. We have also outlined goals for future work in this framework.

REFERENCES

[1] M. Shaw, J. Herbsleb, and I. Ozkaya, “Deciding what to design: closing a gap in software engineering education,” in *Proceedings of the 27th international Conference on Software Engineering*, 2005, pp. 607-608.

[2] M. J. Hawthorne, D. E. Perry, “Software Engineering Education in the Era of Outsourcing, Distributed Development, and Open Source Software: Challenges and Opportunities”, in *Proceedings of ICSE 2005, ACM.*

[3] Chang Liu, "Enriching Software Engineering Courses with Service-Learning Projects and the Open-Source Approach," in *the 27th International Conference on Software Engineering (ICSE'05)*, 2005.

[4] Freeman, P., “Essential Elements of Software Engineering Education,” *IEEE Trans. on Software Engineering*, SE-13, 1987, pp. 1143-1148.

[5] J. Tvedt, R. Tesoriero, K. Gary, “The Software Factory: Combining Undergraduate Computer Science and Software Engineering Education”, in *Proceedings of the International Conference on Software Engineering 2001 (ICSE 2001).*

[6] E. Sweedyk, and R. M. Keller, “Fun and games: a new software engineering course” in *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education,* ITiCSE '05. pp. 138-142.

[7] Timothy C. Lethbridge, “The relevance of software education: A survey and some recommendations”, *Annals of Software Engineering, Volume 6, Issue 1 - 4,* Mar 1998. pp. 91

[8] C. Ghezzi, and D. Mandrioli, “The challenges of software engineering education”, in *Proceedings of the 27th international Conference on Software Engineering. ICSE '05*. ACM Press, New York, NY, 2005.

[9] C. Liu, C., "Partnering with and Assisting Community Partners in Service Learning Projects to Tailor and Articulate Project Requirements," *abstract accepted by the 2005 Frontiers in Education Conference 2005,* Indianapolis, Indiana, USA, October 19 - 22, 2005.

[10] W. Mitchell, “Is software engineering for everyone?” in *Proceedings of the 2nd Annual Conference on Mid-South College Computing.* ACM International Conference Proceeding Series, vol. 61. 2004, pp. 53-64.

[11] P. Ciancarini, “On the education of future software engineers”, in *Proceedings of the 27th international Conference on Software Engineering,* ICSE '05. ACM Press, New York, NY, 2005, pp. 649-650.