Promoting Creativity, Innovation and Engineering Excellence

A Case Study from Sri Lanka

Shahani Markus Weerawarana, Amal Shehan Perera, and Vishaka Nanayakkara
Department of Computer Science and Engineering
University of Moratuwa
Moratuwa, Sri Lanka
shahani.w@gmail.com, {shehan, vishaka}@uom.lk

***Abstract*—We present the design of a software engineering project course targeted towards fostering innovation and creativity and in instilling a sense of software engineering rigor in students. The discussion includes our experiences in conducting this course over the past three years for the students following the Bachelor of Engineering (Honors) degree program in the Department of Computer Science and Engineering of the Faculty of Engineering at the University of Moratuwa. We describe the challenges we faced and our approaches towards addressing such issues as well as our encouraging successes.**

***Index Terms*—computer science education; engineering education; software engineering; innovation; creative thinking**

I. INTRODUCTION

The Sri Lankan IT industry, in a phenomenon common to many Asian countries, has had a strong demand for excellent software engineers with a trend of demand exceeding the supply. Hence, as the premier engineering university in Sri Lanka, we were motivated to explore course designs that would enable students to be highly productive soon after graduation, by reducing the learning curve that they would face in IT industry environments that are powered by strong software engineering best practices. In addition to possessing good software engineering skills, we also noted that the most successful young software engineers displayed confident independent thinking and a sense of innovativeness and creativity in their approach to solving problems. Hence, we were also motivated towards fostering such capabilities in our engineering graduates.

In this paper, we discuss how we designed the software engineering course sequence in the Department of Computer Science and Engineering at the University of Moratuwa. In particular, we describe our experiences in designing, conducting and evolving a key practical course within this sequence – the *Software Engineering Project* – a compulsory fifth- semester course. This course was designed and has continued to evolve with very specific goals of promoting innovation, creativity and software engineering rigor within the students.

II. BACKGROUND

The University of Moratuwa is considered to be the premier technical university in Sri Lanka. It consists of three faculties - the Faculty of Architecture, the Faculty of Engineering and the Faculty of Information Technology. Each year the Faculty of Engineering admits approximately 1,000 students who obtained the highest marks in the country in Physics, Chemistry and Mathematics subjects at the highly competitive Advanced-level examination.

The Department of Computer Science and Engineering (CSE) was added to the Faculty of Engineering in 1985 to offer a specialization in Computer Science and Engineering to cater to the then nascent IT industry of the country. The expectations of the government and the burgeoning IT industry were that graduates from this degree program would provide the expertise required for this new field.

Curriculum development of the CSE degree program has been supported by the financial assistance and advice of many experts from the Ruhuna-Keele link project [1], the Japan International Cooperation Agency (JICA), the Korea International Cooperation Agency (KOICA), the Voluntary Services Overseas (VSO) program and the World Bank project on Improving Relevance and Quality of Undergraduate Education (IRQUE).

The CSE department has maintained very strong links with the IT industry through a Department Industry Consultative Board (DICB) which is represented by all major IT employers and the Computer Science and Engineering Society which is the alumni network of the CSE department. This link has been bolstered by an annual recruitment fair and many other informal gatherings where industry participants continue to provide valuable feedback on potential curriculum improvements.

Despite being one of the youngest departments in the Faculty of Engineering, by actively catering to the requirements of the Sri Lankan software industry and the IEEE/ACM guidelines for CS curricula [2], the undergraduate program offered by the CSE department has become one of the most popular programs in the university. This trend has

978-1-4673-2418-2/12/$31.00 ©2012 IEEE

resulted in each new batch of 100 students entering the CSE department consistently being among the highest performers based on the competitive common engineering course examinations held at the end of first semester in the Faculty of Engineering.

III. Curriculum Overview

The CSE department undergraduate program leads to a Bachelor of Engineering (Honors) degree after an eight-semester program. The students begin the CSE program in their second semester at the university, following a competitive selection examination. The CSE program comprises of courses in a variety of disciplines such as computer science, mathematics, communication skills, and managerial and entrepreneurial skill development.

In addition to the courses taught at the university, in their third year, the undergraduates are exposed to industry experience through a compulsory six-month industrial training program. Thus, the CSE undergraduate program is structured such that the students acquire academic knowledge as well as real-world practical skills via industry internships.

IV. Sequence of Software Engineering Courses

The sequence of software engineering courses in the new curriculum was designed to systematically introduce students to increasingly sophisticated software engineering skills.

The sequence begins with programming skills where Java is taught as an object-oriented programming language. In the second course of the sequence, formal object oriented principles are emphasized and the focus shifts towards applying such principles in designing and implementing software solutions. Here, the students learn how to create a software solution by systematically walking through a solution engineering process by: 1) understanding the context of a problem by gathering requirements; 2) envisioning the overall solution through a high-level architecture; and 3) designing and implementing the solution components. The students develop their skills through small team projects throughout the semester.

The third course in the sequence introduces students to many aspects of production-level software engineering. The students learn the theoretical concepts, principles, processes and best practices of how professional software engineering is done in the software industry. This theoretical walkthrough of real-world software engineering ranges from the rigorous engineering practices necessary for mid-sized to large-scale software systems. The focus is on introducing the students to a variety of software engineering processes models [3] such as the Waterfall Model, the CBSE model and the Evolutionary Models, and how such models are modified and improved to leverage iterative and agile software development practices. The students are taught how these theoretical models are "realized" by software engineering project teams in the software industry via practical process structures such as the Rational Unified Process (RUP) framework [4].

The following semester, the students follow the *Software Engineering Project* course, which is the focus of this paper. In this course, the students get the opportunity to practice the concepts introduced in the previous semester's theoretical course. Since the objective is for each student to experience the practical aspects of real-world software engineering, the projects are completed individually. Yet as discussed below, the course emphasis is on using many of the rigorous industrial software engineering practices in the process of implementing a software solution. The requisite practices cover many important aspects of the software development lifecycle ranging from requirements gathering to project management to quality assurance.

Each of the courses briefly discussed above are compulsory for students in the CSE department. In the subsequent semesters of the software engineering sequence, the department offers many optional courses, such as "software architecture and design," "software process and management," "formal methods in software engineering" and "distributed systems."

V. Evolution of the Software Engineering Project Course

For many years, the final-year group project was considered to be the major component in the curriculum that contributed towards inculcating practical aspects of software engineering. With the curriculum revision that occurred in year 2000, continuous assessment (CA) based evaluation and other project oriented course components were introduced at lower levels other than only in the final year. It is important to note that the CSE department conducted some extra-curricular activities named "mini projects" for second and third year students prior to the curriculum revision, since the department understood the need for developing their practical skills.

Following the year 2000 curriculum revision, a *Programming Project* course (later renamed as the *Software Engineering Project* course) was introduced for the third-year students in the CSE department. This course was evaluated only through CA. The first time this new course was offered, a single project idea was given to all the students to be implemented as their individual project. This methodology was not successful, since most students implemented the project in an uninspired manner, as yet another mundane course requirement. In subsequent years, the methodology was to source ideas from the local software industry and to give these ideas to the students to individually implement solutions under the guidance of software industry professionals. Some of the problems that the local software companies provided to the students had insufficient details and the projects were highly varied in scope. Further, inconsistent supervision and support by the industry partners made it difficult to evaluate and achieve the desired learning outcomes. This experience highlighted the need to understand the challenge and significance of a proper evaluation framework in practically oriented courses that featured heavy CA emphasis.

In the following years, the industry-led projects methodology was abandoned and the students were asked to design and implement software applications that they perceived to be important and interesting. In this approach, many students were able to identify and implement good projects. Yet, once again, there was wide variation in the scope and quality of the

individual projects. The main question that concerned the department was whether the students really gained valuable practical experience about the software implementation process, since it appeared that many students were engaged in a process of "hacking code" in order to complete their projects on time.

By reflecting on these experiences, we realized the importance of aligning the compulsory individual programming project course with the compulsory theoretical software engineering course that was also offered in the third year of the CSE program. With this objective in mind, we re-named the programming project course as the *"Software Engineering Project"* to reflect its underlying intention, and we actively focused on re-designing the course structure and assessment methodology.

VI. Design of the Software Engineering Project Course

Based on industry feedback, we noted that when fresh graduates join the IT industry, one of the initial challenges they face is developing software components in projects with pre-existing and typically large codebases. In colloquial Software Engineering parlance, this is known as a "brown-field scenario," as opposed to a "green-field scenario" where a software engineering team starts developing a project from scratch [4].

We observed that students tend to find green-field scenario projects easy to tackle. In such scenarios, they only need to understand the requirements and then they have complete control over the architecture, design and structure of the software system, which the students are able to handle with confidence. In contrast, the brown-field scenarios tend to be intimidating to the students, particularly when they are initially exposed to such projects.

To gain confidence in a brown-field scenario, software engineers need to be skilled in reading documentation, following discussions in wikis, blogs, forums, and mailing lists, and in reviewing code segments in order to understand the vision and architecture of the overall software system. Further, they need to be creative in designing their software components within the constraints and architectural principles of the existing software system. In order to uphold the existing engineering rigor in a project as they design and implement their own software components, they need to appreciate the importance of sharing and discussing their ideas and design decision in associated wikis, forums and mailing lists. They need to feel comfortable in updating the appropriate project documents with relevant descriptions of the important changes they have made. They also need to ensure that the quality of their code matches the coding guidelines and practices and the quality of the pre-existing codebase. Finally, they need to ensure that their software components work within the larger software system as intended, by implementing specific test cases and rigorous regression testing.

Based on these industry expectations for a fresh graduate, and based on our strong motivation to promote innovation, we identified the following goals for the *Software Engineering Project* course:

1) To encourage students to think innovatively;
2) To provide students with an experience of doing an individual creative design within certain constraints;
3) To improve a student's self confidence in working in a brown-field scenario;
4) To provide students with an end-to-end experience of following rigorous software engineering best practices, including adherence to coding best practices and a variety of documentation;
5) To provide students with a sense of individual achievement and an opportunity to demonstrate their accomplishments.

The overall structure of the course was then engineered to achieve these goals.

At a high level, in order to primarily achieve goal #3, we require the students to develop a component, extension, or app into or upon an existing open source software project [6]. To achieve goal #1, we encourage the students to generate their own project ideas by guiding them through a creative thinking process. These activities naturally lead to achieving goal #2. The course structure and assessment process have been tailored to achieve goals #4 and #5. The mapping of the learning outcomes from a student perspective, the course structural components as discussed above, and the course goal targets are given in Table I.

This course is initiated with a workshop where the students are first briefed on the course details, its expected learning outcomes and the assessment process. We then stress upon the opportunity for innovation provided by this course. Since many students are not aware of how they should "start" the creative thinking process, we encourage the students to think of situations where thoughts of the nature, "if only there was a way to…," "if only I could do…," "if only I had…" have flashed in their minds during their daily, mundane tasks. We then highlight how such simple flashes of thought can lead to a potential software innovation. We also encourage them to brainstorm ideas with their friends. Finally, to inspire the students, we invite former students who have completed the course with great success to discuss how they progressed through the creative thinking process as well as how they then went on to complete their innovation projects within the course framework. We then give the students one week to generate their innovative ideas and gather background material in order to defend their ideas to convince themselves and others that the idea is feasible.

In the next step of the course, the students individually meet with the course lecturers to "market" their ideas and "defend the innovation project feasibility." Depending on their passion, strength of argument, and technical feasibility, they are allowed to progress on to the next stage of the course. Occasionally, some students find it difficult to generate ideas and defend them. After a couple of attempts, the lecturers come to the rescue of such students by providing them with a couple

of ideas from an idea pool and they progress on to the next stage of the course.

TABLE I. OUTCOMES RELATED COURSE STRUCTURE AND PLAN

Overall Learning Outcomes for the Student *After completing this course, students should be able to…*	**Structural Course Components and Plan**	**Targeted Goals**
Develop a component in a software system.	• Think of a potential project idea • "Market your idea" • Complete the development and demonstrate the outcomes at the end of each phase	Goals (1), (2), (3) and (5)
Develop software individually.		
Apply software engineering processes and principles in developing software.	• Practice the iterative software engineering process by planning two phases for the project • Design an agile development project plan incorporating the two phases, with appropriate milestones	Goal (4)
Generate relevant software engineering documents.	• Feasibility Report • Requirements Specification • Software Architecture and Design Document • Quality Assurance Plan	Goals (3) and (4)
Abstract out important concepts in a project and write a technical paper.	• Submit technical paper to the department student research symposium.	Goal (5)

After completing the creative idea generation process, the course moves on to the rigorous development stage. The students are briefed on the expectations, the documents they need to deliver and the associated deadlines. They are expected to consider their own time constraints, their familiarity with the necessary technologies, as well as the course deadlines in developing their individual project milestones. Following software engineering best practices, the course guideline specifies that the individual project should be divided into two phases, with a demonstration scheduled at the end of each phase. The students are urged to follow an agile development process by including several verifiable iterations within each phase. They are also encouraged to include the generation of associated test cases and short time periods for bug fixing in their plans. In alignment with rigorous project practices, the students are expected to submit a fortnightly project progress report. The students then incorporate all these considerations and deliverables to create an individualized project plan, using a project management software tool. Throughout the semester the student is expected to validate their progress against their project plan, and when necessary, make changes to their project plans to accommodate unforeseen circumstances, thereby maintaining the project plan as a living artifact.

The course then progresses with the students working on the projects during the course lab periods as well as their own scheduled times. The students submit the document deliverables through the Moodle course management site [7]. The students are expected to maintain their codebase in a Subversion code repository [8] and they are encouraged to use many software engineering workbench tools such as build environments (Maven), test frameworks and continuous integration tools.

VII. ASSESSMENT PRACTICES

A proper evaluation framework is very important for the success of the software engineering project. As indicated in the previous section a large number of deliverables are expected from the students throughout the semester. Each deliverable contributes towards the final grade. Some of the deliverables are evaluated offline while some of the deliverables, such as the project demonstrations, need to be evaluated in-class. With a class size of 100 students, this requires a significant quantum of evaluation hours. A single lecturer along with a grader cannot be expected to complete all the required evaluation tasks. Hence we receive assistance from other academic staff members in the CSE department and we leverage our industry partners to conduct the in-class evaluations.

This methodology of using IT industry experts for in-class demonstration evaluations has two distinct benefits. In addition to helping with the evaluation workload, the industry experts provide the students with much-valued practical feedback based on current industry best practices. Facilitating such industry participation requires careful planning. Before each evaluation we send a request to most of the local software engineering companies to submit possible candidate evaluators who possess relevant academic credentials and practical experience. Based on their availability and qualifications, we invite the most suitable candidates for the course evaluation session. It is important to note that most of the evaluators are former graduates of the CSE department.

In order to maintain a consistent evaluation, assessment guidelines are prepared in advance and sent to the evaluators. The assessment sheets are structured as 5-point Likert scale [9] categories to reduce the variability amongst the evaluators.

A sample set of evaluations are done at the beginning of the evaluation session with the participation of all the evaluators. At the end of the sample evaluations a quick discussion is conducted to compare the assessments and to "calibrate" each evaluator thereby facilitating greater consistency across evaluators. The evaluators are then grouped based on their technology experience and academic seniority. Each demonstration assessment is conducted by a panel comprising at least three evaluators including at least one academic staff member. Following the evaluation session, the assessment sheets are collated, statistically analyzed and adjusted to maintain consistency in evaluation across multiple evaluations panels.

The course includes a mid-semester evaluation. The sole purpose of this evaluation session is to give constructive feedback to the students to enable them to get back on track if necessary. This session also gives an opportunity for course

instructors to re-scope the project deliverables of some projects. In some cases the scope is reduced and in other cases additional requirements are added to the scope to make it more challenging for the student. In rare situations, a student has also been allowed to completely change their project during the mid-semester evaluation, by taking unavoidable circumstances into consideration.

As one of the final deliverables, a technical paper needs to be written by the students for a research symposium organized by the CSE department. These papers are reviewed with the assistance provided by academic staff members who are on study leave and CSE alumni who are at other universities. Conference management tools are used for the administration of the paper reviews for the symposium.

VIII. Student Feedback

In general student feedback has been positive regarding the evolution of this *Software Engineering Project* course. The most significant aspect that the students have highlighted is the confidence that they have gained towards developing industry-level applications using proper software engineering practices. The second most significant aspect of this course that they have highlighted is the realization of the importance of software engineering processes towards the success of a software engineering project.

The students have also noted that becoming involved in open source projects as required by this course, had helped them secure Google Summer of Code awards [10]. Other benefits that the students have noted include the experience of using a software engineering tool stack, and the improvement in their technical communication skills.

However, the students have also indicated a dislike towards the large number of documents required as deliverables in the course, despite the lecturers' and the industry evaluators' assurances that such tedious technical documentation would be required when they join the industry.

IX. Successes and Challenges

The practices that the students learn in this project and the confidence that they have gained may have significantly contributed towards the steadily increasing number of Google Summer of Code (GSoC) awards consistently garnered by CSE department students since 2005. The CSE department has been producing the highest number of GSoC award winners in the world, every year, since 2007 [10]–[13].

Some software innovations developed in this course were entered in competitions, and students have won national and international awards. For instance, one student won a Mananthan award [14] in India for developing a browser extension to convert content of non-Unicode Sinhala web sites to Unicode. Another student won an mBillionth award [15] in India for developing an Android browser that can render Sinhala and Tamil web content thereby bridging the rendering capability gaps of the Android OS.

It is also important to note that some of the completed projects have been uploaded to official sites and have had significantly high downloads. Some of the projects have been accepted and up-streamed into their respective open source codebases. Some of the products developed as part of this course have also attracted interest by commercial companies.

As discussed in Section VII, the research symposium organized by the CSE department includes the technical papers written by the students in this course. This course requirement has been helpful in instilling a mindset towards writing research papers for peer-review and in providing the students an opportunity to present their accepted papers in a formal symposium. This activity has proven to be highly beneficial for most students, since they graduate with peer-reviewed publications in their resumes.

The final year group projects have displayed a significant improvement after the introduction of this third-year course. The students' final year projects show a notable emphasis on the software engineering process.

One of the toughest challenges of the course is the evaluation process. As discussed in Section VII, the success of the course depends on the assessment process, which includes regular evaluations and feedback, enabling students to realign their effort. Finding a team of evaluators who are capable of performing the required duties and maintaining consistency in grading with multiple evaluators continues to be a challenging effort.

The CSE department has faced practical difficulties in this course due to the scarcity of hardware resources. Currently we do not have a dedicated server for the software engineering workbench tools such as the code repository needed for the course. There is a particularly high interest towards developing Android applications among the students. Due to the lack of access to Android smartphones in the department, the students have to depend solely on android simulators which are not very reliable.

X. Future Expansions

The Sri Lankan higher education sector is at present at a juncture where there is a tremendous emphasis given to enhancing employability of the graduates through the World Bank assisted Higher Education for the Twenty-first Century (HETC) program of the Ministry of Higher Education [16]. Many new initiatives have been introduced to include student based learning activities in the curricula and the sharing of best practices. We see a great opportunity for the CSE department to share our experiences in conducting this course and to eventually expand this course to other degree programs in the country which are targeted towards producing software engineers.

XI. Conclusion

The *Software Engineering Project* course offered by the CSE department of the University of Moratuwa has been highly successful in its vision of fostering a sense of creativity and innovation with engineering rigor in students, as evident in the observable positive changes in many students' approach towards software engineering and their increased level of confidence in software development. This conclusion is further

supported by direct feedback from the students, feedback from the IT industry, and the many awards and student achievements resultant from this course.

As discussed in this paper, we had to address many challenges in the process of continuously reviewing and improving this course over the past three years. We hope that the learning experiences from this interesting case study will be helpful in addressing similar challenges faced by other undergraduate engineering programs seeking to promote similar characteristics in their students.

REFERENCES

[1] K. Tillekeratne, "Keele-Ruhuna link programme in computer science: A postscript," *J. Inform. Technol. for Develop.*, vol. 5, no. 4, pp. 445–452, 1990.

[2] Association for Computing Machinery. (2005). *Curricula Recommendations* [Online]. Available: http://www.acm.org/education/curricula-recommendations.

[3] I. Sommerville, *Software Engineering*, 8th ed. Harlow, MA: Addison-Wesley, 2007.

[4] P. Kroll and B. MacIsaac, *Agility and Discipline Made Easy: Practices from OpenUP and RUP*, 1st ed. Harlow, MA: Addison-Wesley, 2006.

[5] S. Laningham. (2008). developerWorks Interviews: Booch, Nackman, and Royce on IBM Rational at five years [Online]. Available: http://www.ibm.com/developerworks/podcast/dwi/cm-int021908txt.html.

[6] Open Source Initiative. (1998). *The Open Source Definition* [Online]. Available: http://www.opensource.org/docs/osd.

[7] *Moodle*. (2008). [Online]. Available: http://www.moodle.org/.

[8] Apache Software Foundation. (2012). *Apache Subversion* [Online]. Available: http://subversion.apache.org/.

[9] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, vol. 22, no. 140, pp. 1–55, 1932.

[10] Google. (2012). *Google Summer of Code* [Online]. Available: http://code.google.com/soc/.

[11] Google. (2010). *Google Summer of Code Statistics 2005–9* [Online]. Available: https://spreadsheets.google.com/spreadsheet/pub?key=0AgL4N-OdGxhQcDZEdW9BMmxKVG9LbVV6b1NxNnJhWlE&gid=5.

[12] Google. (2011). *Who's Being Schooled? Google Summer of Code 2011* [Online]. Available: http://google-opensource.blogspot.com/2011/10/whos-being-schooled.html.

[13] Google. (2012). *Google Summer of Code 2012 Stats – Part 2* [Online]. Available: http://google-opensource.blogspot.com/2012/05/google-summer-of-code-2012-stats-part-2.html.

[14] Digital Empowerment Foundation. (2010). *The Manthan Award: Award Winners 2010 – SiyaBasScript* [Online]. Available: http://manthanaward.org/section_full_story.asp?id=968

[15] mBillionth Award South Asia. (2011). *Winners 2011 – SETT Browser for Android* [Online]. Available: http://mbillionth.in/milestones/mbillionth-2011/winners-2011/sett-browser-for-android/.

[16] Ministry of Higher Education, Sri Lanka. (2011). *Higher Education for the Twenty-First Century (HETC) Project* [Online].Available: http://www.hetc.lk/.