STUDENT DEVELOPED COMPUTER SCIENCE EDUCATIONAL TOOLS AS SOFTWARE ENGINEERING COURSE PROJECTS*

Vincent A. Cicirello
Computer Science and Information Systems
Stockton University
Galloway, NJ 08205
cicirelv@stockton.edu

ABSTRACT

In a one semester course on software engineering for upper level computer science students, students typically learn the fundamental software processes spanning the software development lifecycle – from requirements specification through architectural design, implementation, testing, and evolution, along with the software tools that support the development activities. Courses in software engineering often incorporate semester long team projects, where students collaborate on a software development project. Thus, in addition to developing the technical skills associated with software development, the software engineering course is a common place where computer science students develop their skills in teamwork and collaboration, as well as in communications. The nature and type of projects in such courses varies. Some integrate projects where students develop software for "real clients" such as on campus departments, or local non-profits, while others have been increasing exposure to open source development with students contributing to existing open source projects. In our software engineering course, we have recently introduced course projects where teams of students specify, design, and implement software that assists computer science students in learning fundamental computer science topics. In this paper, we present our experience with such CS learning tool projects, including a discussion of the impact on student outcomes in a senior-level course on software engineering. We

compare this experience to our prior use of "real projects for real clients" in this same course. We find that students who participate in the CS learning tool projects perceive an increase in learning progress on fundamental principles, theories, and factual knowledge, as compared to their peers in course sections with "real projects for real clients," with an equivalent effect on teamwork and collaboration skills. Most surprisingly, the students who develop CS learning tools report a significantly higher level of progress on industrial relevant skill development as compared to the students who develop so called "real projects for real clients."

1. INTRODUCTION

Software Engineering courses at the undergraduate level often incorporate team projects where groups of students undertake a software development project. In addition to learning the technical aspects of the topic, such team projects enable the students to develop the communications, teamwork, and other social skills that are critically important to their future careers (e.g., [1] [2] [11]).

Projects in Software Engineering courses come in a variety of forms. Recently, there has been an increase in what some have termed Real Projects for Real Clients Courses (RPRCCs) [6], in which teams of students interact directly with a real client, such as a local non-profit or a department on campus, to develop software to meet that client's needs (e.g., [3] [4] [5] [6]).

In other cases, student teams contribute to open source projects (e.g., [8] [9]). Both RPRCCs and courses with open source projects attempt to bring real world experiences to the classroom. In an RPRCC, this real world experience is in the form of interaction with a real client, and specifying, designing, and implementing a system to meet that client's requirements. While in the case of open source projects, the real world experience is one in which students must cope with the challenges associated with working with an existing code-base that may lack important documentation related to design, algorithmic details, or other aspects.

Our prior experience in teaching Software Engineering as an RPRCC resulted in rather mixed results [4]. In that prior work, students self-evaluated their progress toward a set of learning objectives. The student self-rated progress in both developing problem solving skills, as well as communications skills, was significantly higher than the nationwide averages for all CS courses from institutions that use the same evaluation instrument. This is consistent with those who argue that such RPRCCs enable development of communication, teamwork, and other such soft skills. However, we were surprised to find only marginal difference in how students in our RPRCC perceived their development of industrial relevant skills as compared to the nationwide average across all CS courses.

In the two most recent offerings of the course used in our prior study, we have replaced the RPRC with a different project component. Specifically, each team of students specifies, designs, and implements educational software to assist students in learning a CS topic. Teams are free to choose the CS topic. All other course aspects remained the same. Our findings, which we discuss later in more detail, include a

surprising significant increase in student perceptions of their progress in industrial relevant skill development as compared to the RPRCC, along with a moderate increase in their self-rated progress in developing problem solving skills. No significant difference was seen in terms of communication skills between the two project types. An additional advantage that we observe is that the CS educational tool projects enable students to further strengthen their knowledge in other CS areas (i.e., within the chosen CS topic of their project).

2. COURSE STRUCTURE AND CONTENT

Our Software Engineering course is typical in academic content. We use Sommerville's textbook [10] and cover software process models, agile software development, requirements engineering, system modeling, architectural design, implementation, software testing, software dependability, and security engineering.

The course is 4-credit hours and meets for two hours a week in a computer lab, and two hours in a classroom. Lab assignments cover development tools, such as version control, design tools, etc; and the lab also provides a dedicated common time for reviewing project progress with the instructor. We have a dedicated CS lab with features enabling effective collaborative software development such as dual-monitor workstations organized into 4 collaborative groups of 5. Each group of 5 workstations has its own ceiling mounted projector that projects onto a whiteboard.

Students are assessed using a combination of exams, the team project, and a project reflection essay. The exams test software engineering knowledge. Each individual student writes an essay reflecting upon how their team project enhanced their academic learning experience, and vice versa, using specific examples from throughout the semester. The team projects integrate all aspects of the software development process, and have a common set of deliverables coinciding with course content. The deliverables include the software architecture, test cases in a unit test framework, UML models, as well as the software itself (a prototype midway through and the final version at the end). Students use git for version control. At the end of the semester, the teams demonstrate their software to the class. Teams are also required to maintain and submit logs of their activities (e.g., meeting minutes, etc.).

Each deliverable is assigned a team grade. However, 10% of an individual's project grade is an assessment of that member's teamwork. Studies suggest using a variety of measures to assess teamwork (e.g., [7]). Thus, we use a combination of student self-assessment of their own individual teamwork, and of how well their team worked as a whole, along with peer assessment of teammates, and instructor's observations from group lab activities. The teams' activity logs provide additional teamwork related data. To protect teams from non-participative members, each team develops a contract at the start of the semester, setting team ground rules and expectations. In that contract, teams specify a procedure that they use at the end of the semester to apportion their project grade among the team members. A 5 person team is given 500 points to allocate such that each member is allocated at least 70 points and no more than 110. A member with X points receives X% of the team project grade, rewarding those who do extra work to make up for slackers.

3. STUDENT PROJECT TOPICS

The student teams were allowed to choose the CS topic for which they developed their learning tools. They were given some constraints. No two teams (in the same semester) could choose essentially the same topic (a brief project proposal in the first week of the semester was used to control this). The system they develop is required to be interactive (i.e., not conveying static information), adaptive to the user (e.g., adjusting to the user's skill level in the topic), and maintain state across executions (e.g., memory of what the user, or users if application is multi-user, have done in the past). It's required to have a graphical interface, but otherwise choice of platform is at the students' discretion (teams have produced web applications, mobile applications, as well as desktop applications). A rather popular approach taken by several student teams is to use a game format.

In the two sections of the course with the CS learning tool projects, there were a total of 40 students organized into 8 teams of 5 students each. Two of the teams (in different semesters) developed applications related to artificial intelligence (A.I.). One was a web application for learning about the behavior of different genetic algorithm crossover and mutation operators. It provided animations of how the genetic operators work along with an interactive mode where the user applies the genetic operators while the system provides feedback on correctness. The other A.I. related project was a desktop application for learning about different A.I. search algorithms. The user is quizzed on which frontier node should be expanded next by a particular search algorithm, such as A*.

Two teams (also in different semesters) developed games for practicing conversions among number bases. One was a mobile application with a game format similar to Tetris, where numbers from the source base drop down forming a stack and disappear only if the player correctly indicates its equivalent in the target base. The player must keep the stack from reaching the top of the screen. The game gave the player the choice of source and target bases from among decimal, binary, octal, and hexadecimal. This was one of the more popular projects among other students in the course during the end of semester project demonstrations.

Two other projects were related to combinational logic circuits, one related to Karnaugh maps and the other to Boolean expression equivalencies. Another team developed an interactive application for learning about sorting algorithms, and another for learning introductory object-oriented programming concepts.

4. EFFECT ON STUDENT OUTCOMES

Our institution uses the IDEA system [12] for end of semester course evaluation. In addition to the usual summary data, it provides comparison data for other courses in the IDEA database across all institutions that use the system as well as discipline comparison data. Instructors select among a list of general learning objectives those most relevant to the course. Students then self-evaluate, on a scale of 1 to 5, their own individual progress toward each of the learning objectives.

We discuss the results for 5 of these learning objectives: (Objective 1) gaining factual knowledge (terminology, classifications, methods, trends); (Objective 2) learning fundamental principles, generalizations, or theories; (Objective 3) learning to apply course material (to improve thinking, problem solving, and decisions); (Objective 4) developing specific skills, competencies, and points of view needed by professionals in the field most closely related to this course; and (Objective 5) acquiring skills in working with others as a member of a team.

Figure 1 summarizes the results. "CS Learning Tool" refers to a combination of two sections of the course using the project approach of this paper. "RPRC" refers to two sections of the same course where the project was a "Real Project for a Real Client" [4]. All course sections were taught by the same instructor. "Nationwide CS Courses" is the comparison data from the IDEA organization, averaged across all CS courses from all institutions nationwide that use the IDEA evaluation instrument. We show 95% confidence intervals in the graph.

Previously with the RPRCC, there was very little difference, as compared to nationwide CS courses, in students' perceptions of progress toward gaining factual knowledge (Objective 1) as well as in their progress toward learning fundamental principles and theories (Objective 2). However, with students in the course now developing CS educational tools, there is a significant increase in student reported progress on these objectives. This is likely due to students learning not only the software engineering concepts, but also in further strengthening their prior knowledge within the chosen CS topic of their project.

We also see additional improvement in developing problem solving skills (Objective 3) beyond what was already seen with the RPRCC. The CS learning tool projects and the RPRCC both led to equivalently high ratings of progress on teamwork skills (Objective 5), significantly higher than the reported averages for nationwide CS courses. The performance differential relative to all CS courses is explained by the team projects. The lack of difference on this objective between the CS learning tool projects and the RPRCC indicates that the topic of the projects is not so important when it comes to developing teamwork skills. Rather, it is the act of working together as a team on a significant sized project that accomplishes this.

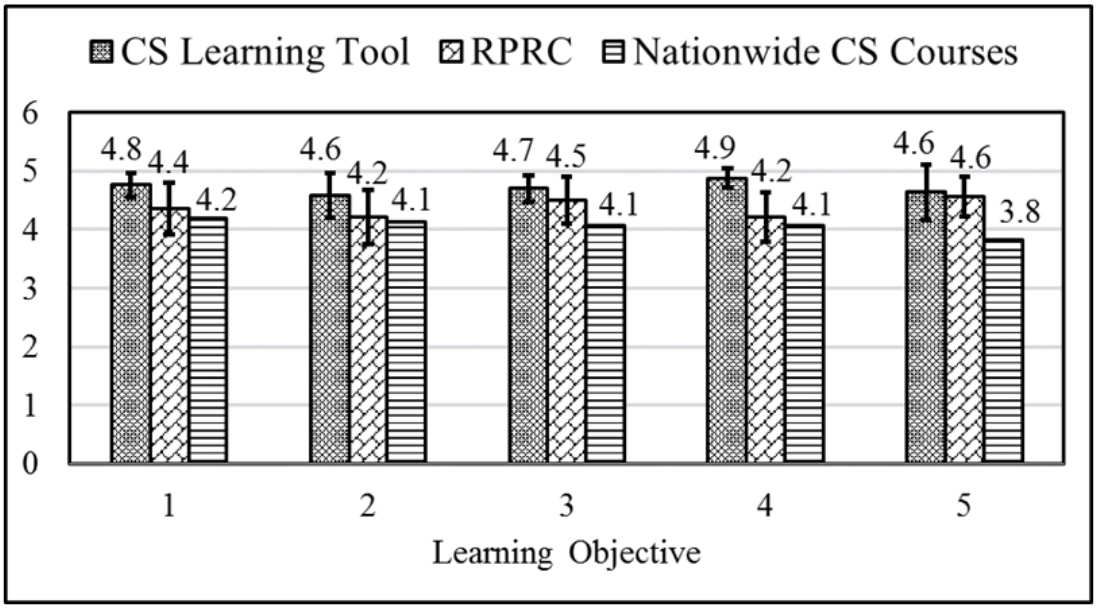

Figure 1: Student self-assessed progress on learning objectives.

Most surprising are the results on the objective related to professional relevance (Objective 4). Previously, we saw that the RPRCC had little effect on students' perceptions of developing "skills, competencies, and points of view needed by professionals in the field…." With the CS learning tool projects, students self-evaluated their progress at a very significantly higher level as compared to the RPRCC, as well as compared to nationwide CS courses. In fact, this was the highest rated objective by the students in the course sections with the CS learning tool projects. Is this due to students' choice of project topic (e.g., because they choose the topic, the project becomes more interesting to them, enabling better focus on the course content)? Or is it related to the earlier discussion of Objectives 1 and 2 (i.e., strengthening skills in the CS topic of their project might be the underlying factor in this objective as well)? Or in the RPRCC, do they simply not see the relevance of interacting directly with a "client," or possibly not view the "client" as "real" since they were at least partially hand-picked by the instructor?

5. CONCLUSIONS

In this paper, we explored the effects of type of team project on student perceptions of progress on a variety of learning outcomes in a software engineering course. We specifically focused on the potential benefits of student developed CS educational tools, where the aim of the student project is to develop software that can assist other CS students in learning fundamental CS concepts. We examined how students perceived their strengthening various skills as compared to students in prior sections of the course where an RPRCC format had been used.

Regardless of project theme (CS learning tools or RPRCC), students saw the benefit to enhancing their ability to work in teams, as well as in improving their problem solving skills (slightly more so with CS learning tool projects). A significant additional positive

outcome of the CS learning tool projects is that students recognized a deepening of their knowledge of fundamental CS principles and theories, as well as factual knowledge. This derives from developing software to help others learn fundamentally important CS concepts. To do so, the students themselves must strengthen their own knowledge in these areas. Additionally, though we previously saw no significant effect on students' perceptions of industrial relevant skill development in an RPRCC, with CS learning tool projects the students in the course do acknowledge a very significant effect in this area.

REFERENCES

[1] Abernethy, K., Treu, K., Teaching computing soft skills: an experiential approach, *Journal of Computing Sciences in Colleges*, 25, (2), 178-186, 2009.

[2] Carter, L., Ideas for adding soft skills education to service learning and capstone courses for computer science students, *Proceedings of SIGCSE '11*, 517-522, 2011.

[3] Chase, J. D., Oakes, E., Ramsey, S., Using live projects without pain: the development of the small project support center at Radford University, *SIGCSE Bulletin*, 39, (1), 469-473, 2007.

[4] Cicirello, V.A., Experiences with a real projects for real clients course on software engineering at a liberal arts institution, *Journal of Computing Sciences in Colleges*, 28, (6), 50-56, 2013.

[5] Klappholz, D., Teaching a female friendly RPRCC (real projects for real clients course) introduction to software development at the middle school, high school or freshman college level, *Journal of Computing Sciences in Colleges*, 25, (3), 2010.

[6] Klappholz, D., Almstrum, V.L., Modesit, K., Owen, C., Johnson, A., A framework for success in real projects for real clients courses, *Software Engineering: Effective Teaching and Learning Approaches and Practices*, 157-190, IGI Global, 2009.

[7] Sabin, M., Assessing collaborative and experiential learning, *Journal of Computing Sciences in Colleges*, 25 (6), 26-33, 2010.

[8] Smith, T., Gokhale, S., McCartney, R., Understanding students' preferences of software engineering projects, *Proceedings of ITiCSE '14*, 135-140, 2014.

[9] Smith, T.M., McCartney, R., Gokhale, S.S., Kaczmarczyk, L.C., Selecting open source software projects to teach software engineering, *Proc. of SIGCSE '14*, 397-402, 2014.

[10] Sommerville, I., *Software Engineering*, 10th Edition, 2016.

[11] Tan, J., Phillips J., Incorporating service learning into computer science courses, *Journal of Computing Sciences in Colleges*, 20, (4), 57-62, 2005.

[12] The IDEA Center, 2016, http://ideaedu.org/, retrieved May 11, 2016.