A Two-Course Sequence of Real Projects for Real Customers

Christian Murphy, Swapneel Sheth, Sydney Morton
Department of Computer and Information Science
University of Pennsylvania
Philadelphia PA 19104
{cdmurphy, swapneel, sydneym}@seas.upenn.edu

ABSTRACT

Since 2012, over 1,100 students at our institution have participated in software engineering courses in which they had the opportunity to partake in "real projects for real customers." Unlike typical one-semester courses or yearlong capstones, our approach is unique in that we offer a two-course sequence in which one group of students develops the initial implementation in the first course and different students maintain and improve the code in the second.

This paper presents our experiences in teaching these courses and serves as a blueprint for other educators who wish to create similar interventions for their students over a two-course sequence. In addition to describing our motivation and the structure of the courses, we discuss how we address issues of scale by using students as Project Managers and the benefits of doing so. We also present empirical evidence that the projects help students feel more confident working in groups, using the agile development process, and working with a real-world customer.

Keywords

software engineering; real-world projects; real-world clients

1. INTRODUCTION

Computer Science educators seek to engage their students in "real projects for real customers" for a variety of reasons, including exposing them to the experience of working in groups and with non-technical stakeholders, giving the students a sense of ownership and personal investment, and allowing them to realize that someone else cares about the quality of their code.

This paper introduces a two-course sequence using "real projects for real customers." In the first course, targeted at upper-level undergraduates, students begin new projects with customers from outside the course, applying the core software engineering principles that they are learning in the class. In the second course, targeted at graduate students, a different group of students continues working on the projects, including bug fixing, adding new features, and refactoring. This second course is focused on software maintenance and gives students the unique opportunity of working with code that they did not write, which is a situation faced by virtually every professional software engineer, but is an experience not otherwise found in most curricula.

We have used this sequence at our institution since 2012: to date, 605 undergraduate students have completed the five offerings of the first course, and 508 graduate students have participated in the five offerings of the second course. During this time, the students have implemented 76 different Android, iOS, and web applications for 54 customer organizations.

SIGCSE '17, March 08-11, 2017, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/3017680.3017742

2. COURSE OBJECTIVES

Almost every Software Engineering course is bound to have some sort of group project, so that students have first-hand experience working in groups, working on large code bases, using source control, adhering to code conventions, etc. In our courses, we sought to address three particular shortcomings of the traditional group project.

First, even when a project is positioned as "solving a real-world problem," students may feel that the output of their project (whether it's a paper, a design, a model, a device, a piece of software, etc.) is simply going to be discarded at the end of the semester, and that no one really cares about its quality except for possibly the TA who will grade it. It is important that the students get the feeling that someone else *does* care about the quality of what they produce, and that it will potentially live on even after the term is over. By having the students work directly with a customer, the project is seen as a professional engagement in which the students are delivering a product that needs to be of high quality. Accordingly, the students put in more effort because they know that what they produce is ultimately worth more than just a grade.

Second, we noticed that when students define the requirements and specifications on their own—even with guidance from the instruction staff—they may be inclined to limit themselves to things they are comfortable with, and omit or change requirements at the end of the project if they seem to be too challenging. Likewise, when the instruction staff dictates the requirements, students may feel constrained and consider the project simply "a big homework assignment." We wanted the students to receive the requirements from someone *outside* the course, preferably someone "non-technical," and elicit the requirements themselves. That way, the students get the experience of working with someone who

perhaps can't quite explain the requirements clearly, who perhaps hasn't thought everything through, and is prone to changing his or her mind.

Last, many software engineering and capstone courses include projects in which students create a new product or application from scratch. Although they may need to learn about and use libraries and frameworks that are new to them, the students start with a blank slate and are free to design and implement the code as they choose. However, once in industry, students will rarely have the chance to start a brand new project with no pre-existing code, but rather will have to read and understand code that was written by someone else. Our goal was to give our students—particularly the graduate students—experience working with existing code in which they were expected to modify it, improve its quality, and add new features, as they would in industry.

3. RELATED WORK

There has been much research into using group projects in a variety of CS classes. In this section, we summarize the important related work focusing on using real-world projects.

The work by Bloomfield et al. [2] is most closely related to ours. They describe a two-semester capstone project done by seniors at the University of Virginia. There are some similarities to our courses: students work in teams and develop software for local non-profits; and they discuss challenges with project selection, mentor selection, and legal issues pertaining to the course. There have been several other papers on longer projects and capstone courses as well. Neyem et al. [6] describe a capstone course framework for teaching software engineering using real-world projects. Bruegge et al. [3] describe a methodology for real-world projects and report on their experience working with 300 students over a four year period. There are many differences, though, between our work and all the others: in our case, since these are two separate courses (one undergraduate and one graduate) there is little overlap between the sets of students in the two courses; we have a much larger number of students and projects leading to additional challenges and approaches for dealing with them (Bloomfield et al. had 14 projects over two years while we've had over 40 projects in a single semester.); and, we have a more detailed empirical evaluation that highlights the benefits of our courses (see Section 5). All of these aspects are highlighted in more detail in the following sections and, in particular, in Section 4.

There has been a lot of work on using projects in software engineering courses. We highlight only the recent related work for brevity. Szabo [9] describes using software projects to teach practical software maintenance. Anslow and Maurer [1] describe how they taught a software development course using real-world projects. Pauca and Guy [7] teach software engineering using socially relevant real-world projects. Tafliovich et al. [10] conducted an empirical study on how projects in such classes should be evaluated. Vasilevskaya et al. [11] also provide an assessment model that is suitable for large project courses. There has also been a recent panel at SIGCSE [5] on integrating live projects in a variety of computer science classes. While all of these use projects in their courses, they are all done in a single semester whereas our two semester course sequence is unique. Further, several of these also do not have real-world customers, which is an integral part of our courses.

4. COURSE ORGANIZATION

Unlike other multi-semester courses in which students develop projects for customers, e.g. senior capstone courses [2], our sequence is unique because the students who start the project in the first course are not the same as the ones who continue to develop it in the second. Likewise, whereas other institutions have all students working on the same project [4] or have no more than 40 or so students [2], some of our course offerings have had over 160 students, working on as many as 40 different projects in the same semester.

This section describes the manner in which the projects are organized and run, as well as the schedule for each course.

4.1 Project Management

The projects in the two courses are run using an agile approach with short iterations of 2-3 weeks each, in which the focus is on keeping the customer engaged and implementing the most important features each cycle. Crucial to the success of any agile project, particularly in a classroom setting, are weekly standup/progress report meetings that ensure that the students are making progress and that the project can change direction as issues arise. Whereas others who have run similar projects in their courses use part of their lecture times for group meetings, this approach would not work for us given the scale of the course (typically over 100 students) and the customers' schedules: it is not realistic to assume that all 20 or so customers would always be available, say, at 10:30 on Tuesday mornings, and it is important that the customers attend the meetings, of course. Likewise, there is only one instructor for the course, and he or she cannot attend all of these meetings.

Our solution—which has been extremely successful—was to create a Project Manager (PM) role for TAs whose responsibility it is to keep the projects moving forward, make sure the students understand what is expected of them, and schedule and attend the weekly meetings.

Project Managers are students who have previously taken the course, as opposed to being students currently in the course who are part of the team. PMs typically manage 2-3 projects per semester, and are compensated at the same hourly pay rate as traditional TAs. When hiring PMs, we look for students who are organized, who communicate well, and who want to be involved in the course but are looking to work only a few hours a week. To date, 57 students have acted as PMs, and all but six have been undergraduates, even for the second course, which is targeted toward graduates. Section 6.2 below describes how students have benefitted from acting as PMs in the course.

4.2 First Course: Development

The first course in the sequence[1] is generally taken by undergraduate students in their second or third years who have completed CS2/Data Structures and may concurrently be taking Algorithms and/or Computer Organization. They typically have had a few semesters of Java but may not have worked on a program of more than 1000 lines or with more than one other programmer. Thus, an objective of the course is to give them experience working on a project that is too big for them to do on their own, and to consider how to design the program so others may change it later.

[1] http://www.seas.upenn.edu/~cis350

The projects in this course are typically "clean slate": the customer has an idea of what he or she would like built, and may have put together a list of requirements or some wireframe diagrams, but no code has been written yet. Customers are almost always part of the university community, particularly since they are familiar with working with students and are aware of the fact that sometimes student projects do not go as planned. Although we do not ask our customers to provide a requirements document up front, we do seek customers who have a clear understanding of who the end-user of the product would be, so that they can participate in requirements gathering exercises conducted by the students. Customers are also expected to participate in weekly meetings throughout the semester: the more they are involved, the more likely they are to get the product that they want.

In this course, the project runs for approximately 10 weeks, using a slightly modified agile development methodology. After an introductory ("release planning") meeting, the students are asked to elicit and document requirements from the customer and then write user stories, which are stored in a project management system such as JIRA. The project then consists of four two-week iterations, each of which starts with an iteration planning meeting in which the students meet with the customer to prioritize the user stories to develop in this iteration. Iterations are two weeks each in order to keep a steady pace; we observed that with three-week iterations, students waited until the last week to do all the work. At the end of each iteration, any new user stories are identified, effort estimates are updated in JIRA, and then the unfinished user stories (and known bugs) are re-prioritized for the second iteration.

Note that there is no "final push" at the end of the semester to complete the project. The students are expected to complete the same amount of work in each iteration, and if the project is not completely finished, that is totally fine, as long as what is delivered is working and documented. At the end of the project, students submit a final report as well as hand-off documentation to be used by the students who continue the project in the follow-on course.

4.3 Second Course: Maintenance

The second course in the sequence[2] is intended for students in our terminal Masters program. These students often have some professional experience and are expected to have some understanding of the basic software development lifecycle.

The emphasis of the course is on the quantifiable aspects of software quality and on software maintenance, including refactoring, testing, debugging, adding new features, etc. As such, the goal of the course project is for the students to continue working on an existing piece of code that they did not implement themselves, rather than starting a new project from scratch. This is the only place in our curriculum where students have the opportunity to work with and improve an existing code base, which of course is an important skill in the professional workplace.

The second course is organized in a similar manner to the first course (in terms of process, using PMs, etc.) but has a slightly different schedule:

- Weeks 1-2: Code inspection. Students become familiar with the existing code by looking for "code smells," violations of code conventions, design anti-patterns, etc. These topics are covered in the lectures prior to this part of the project. The result of this phase is a Defect Log and a prioritization of the issues to be addressed.
- Weeks 3-4: Refactoring and bug fixing. Students then clean up the code by applying refactoring patterns, improving analyzability, etc. The goal is for the students to become familiar with the code through this refactoring, so that they can more easily implement new features later. Note that the goal is not (necessarily) to change the functionality of the app, just its design, though known bugs may be fixed.
- Weeks 5-6, 7-8, 9-10: Development iterations. Once familiar with the code, students have a release planning meeting with the customer to identify new features to be implemented. As in the first course, user stories are written by the students and prioritized by the customer, and the scope of work is dictated by the number of story points. Iterations are still only two weeks long, even though the students often need a bit more time to understand and modify the existing code; the amount of expected test coverage is also higher in the second course than it is in the first course.

At the end of the project, students write a hand-off document, in case the project is continued in a subsequent offering of the course or is turned over to another group of developers.

5. EVALUATION

In order to evaluate the success and effectiveness of these types of group projects, we sought to answer the following research questions:

- **RQ 1:** Do the projects have a positive effect on student learning?
- **RQ 2:** Does working with a customer have a greater impact than not working with a customer?

We primarily sought to discover the benefits that the courses have on the students' ability to work with other people and work with a customer, and also to see if their overall confidence as software engineers increased as well.

5.1 Methodology and Respondents

In order to answer the two research questions, we designed an online survey that students could fill out at the beginning of the course and at the end of the course for extra credit. The students could answer these questions on a 5-point Likert scale with 1 being "not confident at all" and 5 being "very confident." The questions were the same for the two courses. Three questions were used in the evaluation of the courses as follows:

1. How confident do you feel developing software as part of a group? (RQ1)
2. How confident do you feel with the agile process model? (RQ1)
3. How confident do you feel working with a real customer on a software development project? (RQ1, RQ2)

In this paper, we focus on analyzing the responses for students who answered both the Pre- and Post-Project surveys in the two most recent offerings of the courses. From the first course in the sequence, 61 out of 162 students completed

[2]http://www.seas.upenn.edu/~cis573

both surveys and for the second course in the sequence, 59 out of 133 students completed both. We only consider data from the students who answered both surveys in order to analyze the change in scores of particular students.

5.2 Results

As shown in Table 1, the averages for all three survey questions increased from the Pre-Project survey to the Post-Project survey. In all cases, the students felt more confident working in groups, using the agile development process, and working with a customer at the end of the semester than they did at the beginning.

We used paired t-tests to determine the statistical significance of our results. We tested all six combinations (three questions × two courses), i.e., for each row of the table. The null hypotheses were of the form: there is no difference between the Pre-Project and the Post-Project Average for students in the [First/Second] course for Question [1/2/3]. For five of the six combinations, we were able to reject the null hypotheses and our results are statistically significant for $p < 0.05$. Thus, there is evidence to support the belief that the course projects had a positive effect on the students.

The only question that did not have a p-value below .05 was Question #1 ("How confident do you feel developing software as part of a group?") for students in the second course. As shown in Table 1, the average for the Pre-Project survey (3.76) was already quite high, since presumably the graduate students had already taken courses involving groupwork, thus the students might not have increased their scores very much. However, the average still increased for this question to 3.98, demonstrating that the projects had a positive (though not statistically significant) effect.

Although there are other factors affecting student learning, such as lectures and homework assignments, since the averages of the survey questions all increased, and five of the six with statistical significance, we can state that the answer to RQ1 is "yes, the projects do have a positive effect on student learning."

In answering the second research question, we split the students up into two subgroups: those who worked with a customer and those who did not. In the most recent offerings of the course, not all students worked on projects with customers, partly because of scale and partly because of student preferences. We analyzed the results of the first two survey questions between the two subgroups and did not see any statistically significant differences. This is not surprising because we would not expect working with a customer to impact students' perceptions of working in a group or learning about agile processes.

We would, on the other hand, expect a greater impact for the third survey question, "How confident do you feel working with a real customer on a software development project?" Table 2 summarizes the results of our survey and statistical tests. For the first course, there were only 12 students who worked with a customer and answered both the Pre-Project and Post-Project surveys. They reported feeling much more confident working with a customer in the Post-Project survey (average: 4.33) compared to the Pre-Project survey (average: 2.92). These results, however, are not statistically significant. Similar to earlier, we used paired t-tests to compare the differences and we could not reject the null hypothesis in this case (we believe, due to the small sample size). On the other hand, there were 49 students who did not work with a customer and filled out both surveys. The averages in the Pre-Project and Post-Project survey are close to each other (3.02 and 3.14 respectively) and there is no statistically significant difference between the two.

For students in the second course, the averages for the Post-Project survey are higher than the Pre-Project survey for both sets of students: those that worked with a customer and those that did not. Both these differences are statistically significant using paired t-tests for $p < 0.05$. A statistically significant difference for students who worked with a customer is not surprising; what is surprising though is that even students who did not work with a customer felt much more confident at the end of the semester. We attribute this to two factors: first, even though they did not work with a customer, all the other aspects of the course (such as maintenance, testing, and debugging as described earlier) would contribute towards students being more confident if they were to work with customers on future projects; second, since the second course is taken mainly by graduate students, they would have had similar experiences in other courses as evidenced by the higher averages for the surveys overall.

Finally, we wanted to compare students who worked with a customer versus those who did not. We used Fisher's exact test here to see if the differences were statistically significant. For the first course, the Post-Project average for students who worked with a customer is much higher than those who did not work with a customer (4.33 vs 3.14). The differences are statistically significant in this case for $p < 0.05$. On the other hand, there is no statistically significant difference for students in the second course. The reasons for these are similar to the above.

Thus, based on our statistical analysis, it is clear that the answer to RQ2 is "yes, working with a customer does have a greater impact than not working with a customer." This impact is more pronounced for students in the first course than in the second course.

6. OUTCOMES

In addition to the positive educational outcomes described above, the two-course sequence has been successful in other ways as well.

6.1 Student Motivation

Beyond the initial objectives (Section 2) for undertaking these sorts of projects, in retrospect we realized that there are other benefits to our approach in how they motivate the students.

When students work with a real-world customer, they feel the passion that the customer has for the subject matter, and that motivates them to help their customer be successful. Since the customer is engaged throughout the semester and is actively involved in the project, not only are they more likely to get what they wanted, but this also has the side effect of further motivating the students: through this engagement, students feel the customer's passion and are energized to help them reach their goals. Almost all of the customers are university faculty, staff and graduate students who are passionate about what they do, and the students are surely affected by hearing them talk about their work. Even undergraduates who have only recently learned how to program realize that they can help these domain experts see their apps come to life, and that is a very thrilling and rewarding experience that motivates them further.

Table 1: Summary of Survey Responses
*** indicates results that are statistically significant when comparing Pre- and Post- Survey Averages for $p < 0.05$.

How confident do you feel . . . ?		Likert Scale (1—not confident to 5—very confident) Pre-Project Average	Post-Project Average
Developing software as part of a group? (#1)	First Course	3.38	**3.98*** **
	Second Course	3.76	**3.98**
With the agile process model? (#2)	First Course	2.93	**3.82*** **
	Second Course	3.73	**4.14*** **
Working with a real customer on a software development project? (#3)	First Course	3.00	**3.38*** **
	Second Course	3.79	**4.12*** **

Table 2: Working with a Real Customer on a Software Development Project
*** indicates results that are statistically significant when comparing Pre- and Post- Survey Averages for $p < 0.05$.
+++ indicates results that are statistically significant when comparing students who worked with a customer vs did not work with a customer for a given course and a given survey for $p < 0.05$.

How confident do you feel working with a customer on a software development project?		Likert Scale (1—not confident to 5—very confident) Pre-Project Average	Post-Project Average
Worked with a customer	First Course	2.92	**4.33+++**
	Second Course	3.93	**4.17*** **
Did not work with a customer	First Course	3.02	**3.14**
	Second Course	3.67	**4.07*** **

We have also noted that students are often more afraid of disappointing their customer than they are of disappointing their own instructor. This is perhaps speculative, but we might summarize the students' mindset as such: "If I tell my instructor that I didn't do my work or do a bad job of it, then I just get a bad grade; I can live with that. But if I tell my customer—someone who's essentially a stranger—that I didn't do something, or if I let them down, then I might be embarrassed and this person may think less of me; I wouldn't like that." Because students work in direct contact with their customer, there is social cost to showing up for weekly meetings with nothing to say or admitting that the work they did is not very good. This is not to say that avoiding embarrassment is the students' primary motivation, of course, and the above-mentioned reasons certainly provide enough "carrot" so that "stick" may not be necessary, but it is something that combined with the other reasons has led the students to achieve truly remarkable work.

6.2 Project Manager Benefits

Although these types of projects have led to positive outcomes for both customers and students, perhaps not surprisingly it is the Project Managers who tend to benefit the most from their experience in the course. In the same manner that traditional TAs benefit, PMs gain confidence in themselves, become more engaged in the CS department, are looked up to by peers, and feel a sense of "giving back" to the academic community. However, beyond the typical TA role of leading recitation sessions and grading exams, PMs also gain professional and life experience by taking on responsibility for the overall direction of the project, resolving conflicts between group members, and needing to communicate with someone from outside Computer Science. All of these aspects are often cited by industry as common "knowledge deficiencies" that are lacking in graduating students [8].

The PM role also allows for increasing inclusiveness and involving more students in the course instruction staff. For instance, in the most recent offering of the first course (163 students), we were only able to hire eight TAs, but had 20 PMs, each of whom managed an average of two projects and put in about two hours per week. Increasing the instruction staff by an additional 20 students allowed us to hire students who perhaps did not want to commit more time, or who did not feel confident enough in their understanding of the course material, but still wanted to be involved. Over half (30 out of 57) of the PMs who have participated in the course since 2012 are women, and this type of role can certainly be used to increase diversity in the instruction staff. Last, at least seven of the students who acted as PMs are currently working in industry at companies such as Microsoft and Facebook as program or product managers, and all of them reported that their experiences in these courses expanded their perspectives of potential career paths in computing.

7. CHALLENGES AND LESSONS LEARNED

Others have described some of the challenges of engaging in "real projects for real customers," particularly as they relate to intellectual property, maintenance, and problems working with customers [2]. We conclude by discussing additional challenges and lessons learned from the things that are unique about our courses, particularly the large number of students and the fact that students in the second course work on code developed by different students in the first.

7.1 Scale

One of the challenges of offering such courses to large number of students (over, say, 100) is finding enough PMs who can participate in the project over the duration of the semester. As noted above, though, since the time commitment is usually only around one hour per project per week

and students only need to be organized and responsible and do not have to worry about mastery of the course material, the number of applicants to the PM position often exceeds the demand. When the number of PMs starts to get large (in the most recent offering of the first course in the sequence, we had 20 PMs for 43 projects), we have hired Head Project Managers, whose role it was to ensure that the PMs themselves were staying on track, and to help the instructor with administration and infrastructure.

Likewise, it may be difficult to find enough customers, especially since we prefer to have each team of students work on a different project, so as to minimize demands on the customer's time. In some cases, students can work on projects for which there is no customer. At the start of the semester, the PMs have the opportunity to propose ideas for projects, and in such cases, the PM acts in the customer role, in the same manner that a product manager would at a software company. Students are also permitted to propose their own ideas as well, as long as they adhere to the process and meet with their PM each week. Although students who work on these projects miss out on some of the motivations and benefits, as shown above, even students who did not work with an actual customer reported feeling more confident working with one at the end of the project than they did at the beginning, indicating that the nature and structure of the project are still beneficial in this regard on their own.

7.2 Working with Existing Code

Since the code used in the second course of the sequence is the output of the first course, there is an implicit assumption that the code is available and that it works. The students in the first course are required to keep all code in a GitHub repo of which the instructor and PM are owners, and the students also submit a snapshot of the code via our course management system at the end of the term. The code is bound to have bugs, of course, but to date no group has submitted code that does not compile or simply does not work, meaning that there is always something for the students in the second course to start working with.

Often the students in the second course ask about contacting the original authors of the code for help; however, this is expressly forbidden, and we try as much as possible to anonymize the code and documentation that we provide to students in the second course. To help with this, students in the first course write a handoff document that maps features of the software to parts of the code, and vice-versa, so that students know where to start when they want to make changes. On the rare occasion when it has been necessary to contact the original authors, e.g. to get information about a license key or login credentials, the instructor or PM does so, so that the students do not then make further inquiries.

8. CONCLUSION

Since 2012, over 1,100 students at our institution have benefitted from our two-course sequence of "real projects for real customers," which increases students' motivation as well as their confidence, and provides them with a unique opportunity to maintain and improve code that they did not write themselves.

Scaling this approach to large classes is challenging, but can be done by employing students as Project Managers, who benefit greatly from the experience. We would like to thank all 57 PMs from our two courses, without whose dedication and hard work this literally would not have been possible.

9. REFERENCES

[1] C. Anslow and F. Maurer. An experience report at teaching a group based agile software development project course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 500–505. ACM, 2015.

[2] A. Bloomfield, M. Sherriff, and K. Williams. A service learning practicum capstone. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 265–270. ACM, 2014.

[3] B. Bruegge, S. Krusche, and L. Alperowitz. Software engineering project courses with industrial clients. *Trans. Comput. Educ.*, 15(4):17:1–17:31, Dec. 2015.

[4] J. Campbell, S. Kurkovsky, C. W. Liew, and A. Tafliovich. Scrum and agile methods in software engineering courses. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16. ACM, 2016.

[5] J. D. Chase, P. Uppuluri, T. Lewis, I. Barland, and J. Pittges. Integrating live projects into computing curriculum. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 82–83. ACM, 2015.

[6] A. Neyem, J. I. Benedetto, and A. F. Chacon. Improving software engineering education through an empirical approach: Lessons learned from capstone teaching experiences. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 391–396. ACM, 2014.

[7] V. P. Pauca and R. T. Guy. Mobile apps for the greater good: A socially relevant approach to software engineering. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 535–540. ACM, 2012.

[8] A. Radermacher and G. Walia. Gaps between industry expectations and the abilities of graduates. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 525–530, 2013.

[9] C. Szabo. Student projects are not throwaways: Teaching practical software maintenance in a software engineering course. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 55–60. ACM, 2014.

[10] A. Tafliovich, A. Petersen, and J. Campbell. Evaluating student teams: Do educators know what students think? In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 181–186. ACM, 2016.

[11] M. Vasilevskaya, D. Broman, and K. Sandahl. An assessment model for large project courses. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 253–258. ACM, 2014.