Cross-Departmental Collaboration for the Community: Technical Communicators in a Service-Learning Software Engineering Course

Joseph Chao and Jennifer Brown
Bowling Green State University, Bowling Green, OH, USA

jchao@address.edu; jkbrown@bgsu.edu

Abstract

This paper discusses a collaborative service-learning approach to a software engineering course that involved partnering with local non-profit organizations and collaborating with a technical communication class. The main goals of the collaboration with the technical communication class were to provide the students with a real-world project that gave them experience with a cross-departmental team collaboration and to improve the documentation accompanying the software that was developed for the non-profit organizations. Another goal was to, in turn, reduce the burden on the computer science instructor to provide technical support for the software after the end of the semester.

We describe the courses involved, the goals for and method of collaboration, limitations, student survey responses, and lessons learned from this collaboration. As expected with a first attempt at a cross-departmental collaborative project, student survey results showed both positive and negative impressions of the collaboration. With further transforming of the curriculum, we believe this type collaboration holds value as an effective method of providing real-world experience, not only with developing software and working with a client, but also with collaborating with team members from other disciplines.

Keywords: Software Engineering, Agile Software Development, User documentation, Active Learning, Service-learning, Real-world project, Technical Communication.

Introduction

Traditionally in a project-based software engineering course, students learn software development skills by working on tightly controlled classroom projects provided by instructors. While such projects provide valuable software development experiences to students, service-learning projects expose students to real-world situations that cannot be easily replicated in classroom projects.

Service-learning is an active-learning pedagogy that integrates community needs with student learning. As defined by Bringle and Hatcher (1995), service-learning is a "course-based, credit bearing educational experience in which students (a) participate in an organized service activity that meets identified community needs, and (b) reflect on the service activity in such a way as to gain further understanding of curricular content, a broader appreciation of the discipline,

and an enhanced sense of personal values and civic responsibility."

Service-learning in software engineering has been embraced by many such as Liu (2005), Poger and Bailie (2006), Song (1996), and Tadayon (2004). They found that service-learning software-development projects not only provide students with real-world experience in their technical and social skills, but also instill civic responsibility and ownership in students. Some (Purewal, Bennett, & Maier, 2007; Rosmaita, 2007) also suggest that the service-learning pedagogical approach may attract more motivated and higher-achieving students to the computer science discipline, which could be a major benefit for the discipline, especially when computer science student enrollment has been decreasing for the last few years (Carter, 2006).

While service-learning in software engineering courses is not new, it has not been widely applied in the discipline partly because of its challenges, which include, most notably, additional time and organizational demands on instructors, and maintenance needs after the completion of the projects. The issue of additional demands on instructors needs to be addressed according to one's circumstance; the system maintenance issue could also be quite complicated. One solution to the maintenance issue is to have a support center, as suggested by Chase, Oakes, and Ramsey (2007). An "Agile Software Factory" has also been proposed by Chao and Randles (2009) in supporting the maintenance effort for student service-learning projects. As another solution, this paper suggests adding technical writers to the service-learning project teams with the intent of producing better documentation for easing the burden of maintenance.

It is our experience, in past service-learning software development, that documentation produced by the student software developers was typically low quality and/or scarce. The causes for the weak documentation can be attributed to two main reasons: 1. There is limited time for software development itself, which implies limited time for creating documents, and 2. Students in computer science or software engineering are not trained in technical communication (writing) and do not enjoy writing much.

Aiming to improve the quality of the software documentation for the service-learning projects, the instructor from the Computer Science Program and the instructor from the Scientific and Technical Communication Program collaborated in fall semester of 2008 with the goal of adding technical communication skills to the software development teams. That goal manifested itself as a curriculum that involved upper-level technical communication students serving as the technical communicators for software development students. Each team of students worked with a client from a local non-profit organization to develop software that would fulfill a need of the client. The technical communication students wrote the user-centered release notes and delivered those to the client with each iteration of the software. Also, the technical communication students wrote the final user documentation as an HTML help file.

At the beginning of the semester, the software engineering students were divided into six teams and assigned an already-solicited service-learning project for a community partner. Next, the technical communication students were assigned to one of the six teams. Four of the teams were assigned two technical writers each; two of the teams were assigned one technical writer each. Our collaboration, in total, involved fifty-six students—forty-six from the software development class and ten from the technical communication class.

The service-learning projects were provided through the Agile Software Factory (http://agile.bgsu.edu). The Agile Software Factory (ASF) was founded in 2008, with a grant from the Agile Alliance, by a faculty member under the Department of Computer Science at our university. The Agile Software Factory has three main goals:

1. Promote the practice of service-learning at the University, particularly within the Department of Computer Science.

2. Cultivate connections between students in the software development class and non-profit organizations that need software developed (which helps to achieve goal #1).
3. Provide ongoing support for the developed software.

To expand on these three goals, the ASF is one way to better equip our computer science students for their future careers by providing them with real-world experience using the agile approach to software development. The ASF also provides the safety net and support of a classroom combined with the excitement and pressure of completing a professional-quality project for a real client.

As we continue to develop the ASF and solicit sponsorships, we intend to provide more opportunities for students outside of the software development class to be involved in the ASF.

The Service-Learning Software Engineering Course

Software Development in the Department of Computer Science is a project-based software engineering course that teaches the crafts of software engineering to students via a large-scale hands-on software project. Because this is the first software engineering course for most of the students, it is expected to cover all topics throughout the complete software development life cycle, including planning, analysis, design, implementation, testing, and maintenance of large software systems. In addition, project management and other human aspects of software development are discussed. Although it is possible to teach software engineering using a tightly controlled classroom project, anecdotal evidence has shown that students learn better in a real-world environment. Despite knowing how challenging implementing real-world software projects in a classroom setting can be, the instructor decided to adopt the service-learning pedagogy as a way to provide students with real-world experience.

In the fall semester of 2008, six new software development projects from local community partners were selected for students to work on. A total of forty-six students, mostly undergraduate seniors and first-year graduate students, in two class sections were grouped into six teams, one for each project. After adding student technical writers from a technical communication course in the English Department to the teams, each team ended up with seven to ten members.

One of the major challenges of teaching this service-learning course is that most students who take the course do not have any prior knowledge in software engineering and are required to complete a quality software system in a short sixteen-week semester. The instructor must quickly provide enough information/knowledge for the student to start the project as early as possible so that there will be enough time for the teams to produce a quality system that can be useful to the client. Thus, the first three weeks of the semester were used to quickly introduce the concept of software engineering, software process models, project planning, and requirement analysis to facilitate the first customer meeting scheduled during the fourth week.

To mitigate the risk of not delivering a quality system to the client at the end, an iterative and incremental agile software process based on eXtreme Programming (Beck, 2000) and Scrum (Schwaber & Beedle, 2001) was used for all teams. With the iterative and incremental approach and fast customer feedbacks, it ensures the delivery of a useful system for the customer at the end of the semester. This agile process model was applied successfully by student teams in a similar course taught previously by the same instructor (Chao, 2005), and was recommended by many other educators, such as Alfonso and Botía (2005).

The project was broken into five iterations, separate time periods of two to three weeks each. Iteration 0 was intended for project preparation, including tasks such as meetings with customers, research on technologies, and preliminary project planning and estimation; Iterations 1 through 4 each contained a set of user stories (system requirements) to be completed, tested, and delivered

to the client for evaluation and feedback at the end of the iteration. All computer science students in a team were to be developers with a shared role in project planning and management. Because this course does not have an associated lab, students schedule their own meetings and time for project development.

Technical writers were introduced for solving the initial problem that instigated the collaboration: poor end-user documentation. We wanted to provide the software users with usable documentation that would enable them to efficiently complete the desired tasks on their new software, and to reduce the burden on the computer science instructor to provide software support after the product had been delivered. The collaboration also provided both groups of student valuable learning experience in providing quality user documentation for a real-world project. The documents the teams created collaboratively included a project plan (revised after each iteration), release notes for each delivery, a user manual, and online help.

Collaborating with the Technical Writers

At the beginning of the semester, the software engineering students were informed of the collaboration; the technical communication students were informed of the collaboration prior to the start of the semester via emails from their instructor.

To mimic a real-world situation and to effectively work under the sixteen-week time constraint of a college semester, the first thing the software engineering students learned in their class was their project deadline. The instructor determined the deadline and required that each software project consist of five iterations, the deadlines of which were also set by the instructor. The students, however, were responsible for interviewing their client, listening to the client's needs, assessing how a software program could meet those needs, and then determine the extensiveness of the product. In line with the agile approach to software development, the students also broke the total extensiveness of the product into the five required iterations, deciding what functionality would be completed and delivered to the client with each iteration.

After the software engineering students had detailed out their projects, they were assigned one or two technical communication students. The extensiveness of the project determined whether a team was assigned one technical writer or two—the teams with more extensive projects were assigned two technical writers; the teams with less extensive projects were assigned one technical writer.

The instructors of the courses had explained to both classes that the technical communicators would complete the release notes for the client with each iteration, create the final user documentation in the form of an online help file, and provide their expertise to make any team document more usable. However, the students were to, as a team, determine what specific role the technical communicators would fill, when they would receive the computer science students' notes about each iteration so they could document it, and when they would need to submit the release notes to the computer science students so the computer science students could provide the notes to the client with the iteration. The collaborative teams were also to determine how the technical communication students would access the software and how the computer science students would communicate with them and keep them abreast of the progress of the project.

Schedule

Because project completion was limited to the sixteen weeks within the semester, the students were allotted no more than two weeks to work on each iteration before providing the iteration's deliverable. Table 1 shows the schedule the students were provided, limited to only the milestones for the project:

Table 1: Semester Schedule

Week(s)	Service-learning project task
1 & 2	Introduction to software engineering and the projects.
3	Form software development project teams and role assignments, as well as the software development and technical communication combined teams.
4	First customer meeting and Iteration 0 (I_0), which consisted of planning and a requirements analysis.
5	Second customer meeting to review requirements and project plan.
6 & 7	Work on I_1 and accompanying documentation.
8	Deliver I_1 (a working system) with accompanying release notes and updated project plan to client.
9	Work on I_2 and accompanying documentation.
10	Deliver I_2 (a working system) with accompanying release notes and updated project plan to client.
11	Work on I_3 and accompanying documentation.
12	Deliver I_3 (a working system) with accompanying release notes and updated project plan to client.
13–15	Work on I_4. Perform qualitative usability test.
16	Deliver final software product and accompanying documentation (as a compiled help file) to client.

Method of Student Communication

The software engineering students did not work on their projects solely in class, but worked asynchronously outside of class as well. The technical communication class was an online class that semester, so unless the students made an effort to meet face-to-face in their own time, they collaborated electronically and usually asynchronously. Because of the asynchronous component of the collaboration, the students needed effective tools to communicate with one another. In addition to phone calls and emails, the students also used several online collaboration tools:

- Wikis
- A file exchange server
- Microsoft Visual Studio Team Systems, which includes a version control system

These tools also allowed each student to document the work he or she had done on the project and keep his or her team members abreast of the progress. Moreover, these tools allowed the instructors to evaluate each student's contribution to the project.

Assessment and Evaluation

Challenges/Limitations

Before the semester began, we, the computer science and technical communication instructors, brainstormed any foreseeable limitations with the collaboration in an effort to negate some of those limitations and avoid disaster. We determined, based on past experience, that the major limitation would most likely be time. Most of our students carry full course schedules, are involved in organizations on and off campus, and work a part-time job. It would therefore be difficult for our students to coordinate group meetings with one-hundred-percent attendance.

We decided the best thing we could do to negate the effects of time limits was to design our courses so that the collaborative service-learning projects served as the focus as well as the capstone of the courses. All class assignments, therefore, were building blocks to completing the project by cultivating the skills the students would need for the project. Additionally, the computer science instructor designated some in-class work days for project.

Assessment

The value of any new approach to learning or teaching can be lost if its effectiveness isn't evaluated. Therefore, before the semester began, we established a set of criteria to determine if the collaborative service-learning project was successful, if it should be continued in the future, and how it could be improved in the future:

1. Quality of documentation: Is the documentation usable for the clients? We will determine this by way of client surveys sent one week after final release and two months after final release. We will also have the students perform a brief qualitative usability test with the clients.
2. Student feedback: Do the students agree that they learned something valuable from the project itself and the collaboration? We will determine this through an anonymous survey at the end of the semester.
3. Student articulation of learning: Can the students point to the goals/benefits of the steps in the process and relate them to an outcome of the project? Can students identify what they would do differently next time if they were faced with the same or a similar project?

Survey Results

In order to receive student feedback on the collaboration, we conducted two different surveys—one for the computer science students in the Software Engineering course and one for the technical communication students.

Survey of the computer science students

An anonymous survey to the forty-six computer science students was conducted at the beginning of the second iteration (the ninth week of the semester), to which forty-four students responded. An overwhelming majority (93%) of the students enjoyed working on their real-world project, and more than 95% students believed that the skills learned in the class were applicable to the real world. Forty-three percent of the students felt that the workload in the class was either high or too high due to the project demand, and less than 5% thought that it was low.

On the questions related to their clients, 93% of the students believed that they understood the needs of their clients, but only 75% of them thought that their clients were satisfied with their work so far. While 77% of the students believed that they had acted professionally with their clients, only 61% of them thought that the communication with the clients was prompt and painless.

Most of the students (80%) felt that they were working with a good team, 79% were satisfied with the project progress at this point, and 89% of the students were confident that they would produce a useable system at the end to meet the client's needs.

Concerning the technical writers on their teams, 75% of the computer science students understood the contribution by their technical writer, but a mere 45% of them thought the collaboration was effective, purposeful, and useful. Table 2 below shows more details for the student responses on the survey questions.

Table 2: Survey results of the Software Engineering students after the first iteration

Questions	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I enjoy working on the real-world project in this course.	25 (56.8%)	16 (36.4%)	2 (4.6%)	1 (2.3%)	0
The skills I learned in this class are applicable to the real world.	24 (54.6%)	18 (40.9%)	2 (4.6%)	0	0
The workload of this class is ...	Too High 4 (9.1%)	High 15 (34.1%)	Just Right 23 (52.3%)	Low 2 (4.6%)	Too Low 0
My team has worked with/interacted with the customer as professional service providers would.	8 (18.2%)	26 (59.1%)	10 (22.7%)	0	0
The communication between my team and the customer has been prompt and painless.	7 (15.9%)	20 (45.5%)	9 (20.5%)	8 (18.2%)	0
I understand the customer needs for the system.	16 (36.4%)	25 (56.8%)	2 (4.6%)	1 (2.3%)	0
The customer was satisfied with the iteration plan presented in our project plan.	13 (29.6%)	20 (45.5%)	11 (25%)	0	0
I feel that I am working with a good team.	11 (25%)	24 (54.6%)	7 (15.9%)	2 (4.6%)	0
I am satisfied with the project progress so far.	7 (15.9%)	28 (63.6%)	5 (11.4%)	4 (9.1%)	0
I am confident that my team will produce a useable system at the end that meets the customer needs.	16 (36.4%)	23 (52.3%)	4 (9.1%)	1 (2.3%)	0
Our collaboration with the technical communicators on our team has been effective, purposeful, and useful.	4 (9.1%)	16 (36.4%)	18 (40.9%)	5 (11.4%)	1 (2.3%)
I understand what the technical communicators contribute to our team project.	9 (20.5%)	24 (54.6%)	8 (18.2%)	2 (4.6%)	1 (2.3%)

When asked in an open-ended question of "What do you like about this class?" working on a real-world project and/or interacting with real clients was clearly the student favorite with 25 references. Others indicated that they have learned a lot, enjoyed the teamwork, etc. When asked in another open-ended question of "What do you dislike about this class?" several students disliked the class examination and a couple of other non-project related issues. On project-related feedbacks, six students had concerns about the high workload, five students were frustrated with teamwork problems, and two students were unhappy with the grading mechanism for the project.

Survey of the technical communication students

The technical communication students were asked to complete an online, anonymous survey during week thirteen of the semester, just after their documentation for iteration 3 was due. The survey consisted of twelve statements with which the students were asked to rank their level of agreement. The instructor did not want to provide them with a "neutral" option as an answer to any of the questions, but to restrict them to choosing on the side of somewhat agreeing or some-

what disagreeing as an alternative to "neutral." Of the ten students in the course, nine of the students completed the survey.

Overall, the technical communication students agreed positively (in varying degrees) with the statements, suggesting they did indeed see value in the collaboration and their team was, for the most part, working effectively. Table 3 provides the survey statements and the students' responses.

Table 3: Survey results of the technical communication students after the third iteration

Questions	Strongly Agree	Agree	Somewhat Agree	Somewhat Disagree	Disagree	Strongly Disagree
I believe this collaboration has given me an experience similar to collaborating in a work setting.	4 (44.4%)	2 (22.2%)	0 (0.0%)	2 (22.2%)	0 (0.0%)	1 (11.1%)
I believe that what I have learned from this collaboration will be useful in my future career.	3 (33.3%)	3 (33.3%)	2 (22.2%)	0 (0.0%)	1 (11.1%)	0 (0.0%)
I have gained beneficial skills in working with a team through this collaboration.	3 (33.3%)	3 (33.3%)	1 (11.1%)	2 (22.2%)	0 (0.0%)	0 (0.0%)
I have consistently done the best work I could do throughout this collaboration.	4 (44.4%)	5 (55.6%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Our ENG/CS team has communicated effectively throughout this process.	3 (33.3%)	0 (0.0%)	3 (33.3%)	2 (22.2%)	1 (11.1%)	0 (0.0%)
I feel the CS team members effectively use my skills as a technical communicator.	2 (22.2%)	3 (33.3%)	1 (11.1%)	0 (0.0%)	1 (11.1%)	2 (22.2%)
Through this collaboration, I have come to better understand how to create documents for a client.	4 (44.4%)	3 (33.3%)	1 (11.1%)	0 (0.0%)	1 (11.1%)	0 (0.0%)
Through this collaboration, I have come to better understand how to manage a client relationship.	2 (22.2%)	1 (11.1%)	5 (55.6%)	0 (0.0%)	1 (11.1%)	0 (0.0%)
Our ENG/CS team worked collaboratively to determine my responsibilities.	1 (11.1%)	3 (33.3%)	2 (22.2%)	1 (11.1%)	1 (11.1%)	1 (11.1%)
I understood my responsibilities for the collaboration at the beginning of the semester.	1 (11.1%)	1 (11.1%)	2 (22.2%)	1 (11.1%)	1 (11.1%)	3 (33.3%)
I now understand my responsibilities for the collaboration.	3 (33.3%)	4 (44.4%)	1 (11.1%)	0 (0.0%)	0 (0.0%)	1 (11.1%)

The one statement the students most disagreed with was "I understood my responsibilities for the collaboration at the beginning of the semester." The instructors had intentionally left the specific responsibilities of the technical communicators vague in order to allow the students to decide within their teams what tasks exactly the technical communicators would perform and what contributions they would make to the team. They had, as discussed previously, been informed of their three basic duties: write the release notes and final documentation, and edit to make all team documents more usable.

However, the survey results for the statement "Our ENG/CS team worked collaboratively to determine my responsibilities" received the second lowest rating for agreement (tying with "I feel the CS team members effectively use my skills as a technical communicator"), indicating the students did not work as a team to determine the specific duties of the technical communicators, contrasting with the instructors' expectations for the collaboration.

This lack of team-determined responsibilities for the technical communicators seems as though it could be connected to the survey results for the statement "I feel the CS team members effectively use my skills as a technical communicator": perhaps the computer science students did not know enough about what the technical communicators could bring to the team in order to collaboratively decide upon the technical communicators' tasks and use their expertise efficiently. This could perhaps also be the reason behind the fact that only 45% of the computer science students thought the collaboration was effective. Additionally, the survey of the computer science students was taken after the first release. A second survey later in the semester might reveal that they've come to better understand the expertise the technical communication students have to offer and see value in the collaboration.

The technical communication students were also asked to provide qualitative feedback on their experience up to that point in the semester. Five of the ten students provided comment, and in general, their comments addressed three main issues:

1. Difficulty communicating with their team.
2. Computer science students' unfamiliarity with the role of a technical communicator.
3. Their own uncertainty as to their role in the project.

Lessons Learned

We begin our assessment of the success of the project with a partial assessment relative to the second criterion: student feedback. As a blanket statement, according to the surveys, the students had difficulty navigating their way through this collaboration; some found value in the collaboration, and some did not.

While the students did not unanimously affirm the value of the collaboration, enough of them did seem to reflect positively on the experience and see value in it to justify attempting the collaboration with another group of students during another semester when both courses are offered. We, as the instructors, however, have gained valuable insight from this collaboration, which we will apply to our next attempt. We've categorized these lessons learned into three areas discussed below. Overall, we have learned that for future iterations, we must detail processes to the teams or provide discussion points for the teams to determine processes among themselves.

Team Collaborations

Most of the problems with team collaborations were a result of poor communication. One problem was that, contrary to our expectations, our students often didn't negotiate communication and workflow on their own. In the future, we will either explicitly tell them to determine their own

workflow around their iteration due dates, or we will provide them a step-by-step process, which is what the technical communication instructor provided her students for the third iteration.

A second problem with communication was the fact that neither the computer science nor the technical communication students recognized what information they needed to communicate to their team members from the other course. For example, the technical communication instructor assumed that the technical communication and computer science students would communicate to one another about accessing the software program being developed. However, the computer science students did not think of telling the technical communicators how to access the software, and the technical communicators did not think of asking for access to the software.

The final problem resulting from communication was a lack of scheduling within the teams. The students did not work out a schedule among their team members that would provide deadlines to complete all work for an iteration. For example, even once the technical communication students understood that they were to completely revise the computer science students' release notes, there was not time for them to do that on some occasions because development took longer than expected. In future collaborations, the instructors may have to provide the teams with a list of discussion points for their first meeting to determine roles, responsibilities, access privileges, etc. We may consider a guaranteed two-day window between when the iteration is done and when the technical communicators must submit their release notes to the client.

Business Analysis

The technical communication students were not part of the first client meeting, in which the needs of the client, and, in turn, the requirements for the software, were determined. Therefore, many of the technical communication students never knew what the defining purpose of the software program was and what it was supposed to do for the client.

The technical communicators were involved in subsequent meetings with the client; however, many of those meetings focused on technicalities and programming aspects that were not applicable to the technical communicators, rather than focusing on the higher-level aspects that would be relevant to them and the client.

In the future, we would consider involving the technical communicators in the first client meeting. That way, they could understand what role the software fills and what the user's original needs were. The technical communicators could also, after hearing the user's needs, perhaps provide some suggestions for what the software could do for them. Indeed, they could perhaps help out with the "business analysis" portion of the project; technical communicators are taught to be advocates for users and to always think of how to reduce mental burden on their audience.

Documentation

The technical communication instructor assumed that the technical communication students would know, based upon their previous knowledge in technical communication, that when they were asked to write user documentation for each iteration, they were not to simply edit the engineer-centered release notes written by the computer science students. Instead, they were to completely re-work the notes to make them suitable for the user: remove information that was irrelevant to the user, write step-by-step instructions for the user to work with the features of the iteration, provide examples to enhance user understanding, etc. The purpose of the computer science students' release notes was simply to provide the technical communicators with an update on what the new features were and how they worked, not to serve as the actual user documentation.

For future collaborations, the technical communication instructor will make certain the technical communicators understand that the notes provided by the computer science students are not the user documentation—they are simply scratch notes to the technical communicator about what the

system should be able to do. The technical communicators are supposed to review the notes, work with the software, and write the user-centered release notes to accompany delivery of the iteration to the client.

A second lesson learned in the area of documentation resulted from discrepancies between the documentation provided to the clients and the documentation provided to the technical communication instructor. For the first few iterations, the software engineering and technical communication instructors did not receive the same version of the technical communicators' release notes. Apparently, the students created release notes to submit for the client deadline, but then further revised them to submit to the technical communication instructor, who would then grade the release notes. That situation was, however, corrected mid-semester; the students were told that they were to submit the same release notes to the client and their instructor. This will be explained clearly at the beginning of the semester for the next collaboration.

Conclusion

Though the students did not overwhelmingly agree that they found irreplaceable value in this collaboration, there was enough positive feedback from them that the instructors will pursue the collaboration in future semesters. They will do so with new insights that will ideally make future collaborations smoother and help the students see the value of the collaboration more clearly.

The instructors have yet to assess the collaboration based on three of the four pre-established criteria for determining the success of the project. After the end of the semester, the instructors will survey their students again, asking questions that specifically provide insight into the assessment criteria of whether students can articulate their learning. Additionally, the instructors will survey the clients for their feedback on the software and accompanying documentation and compare it to the feedback from previous semesters to see if there is any improvement in satisfaction with the deliverable. We believe this type of service-learning collaboration holds significant potential as an effective pedagogical approach and will continue to transform it as we receive more feedback at the end of this semester and through future semesters.

Acknowledgements

We would like to thank the following groups of people:

- Our students: for your flexibility and your cooperation as we work through this new type of collaboration, as well as for your open feedback about the project and what could be improved in the future.
- Agile Software Factory: for providing the projects for this collaboration and serving as the connection point with our community partners.
- Agile Alliance: for your interest and support in this service-learning endeavor. We are certain that your efforts will have a positive effect on the students' learning and their value to employers when they graduate.

References

Alfonso, M. I., & Botía, A. (2005). An iterative and agile process model for teaching software engineering. *Proceedings of the 18th Conference on Software Engineering Education and Training (CSEET'05), Ottawa, Canada,* April 18-20, 2005, 9-16.

Beck, K. (2000). *Extreme programming explained: Embrace change.* Addison-Wesley.

Bringle, R. G., & Hatcher, J. A. (1995). A service-learning curriculum for faculty. *Michigan Journal of Community Service Learning, 2,* 112-122.

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. *Proceedings of the 37th SIGCSE technical symposium on Computer science education. SIGCSE'06, Houston, Texas, USA.*

Chao, J. (2005). Balancing hands-on and research activities: A graduate level agile software development course. *Proceedings of the Agile Development Conference (ADC'05).*

Chao, J., & Randels, M. (2009). Agile software factory for student service learning. *Proceedings of the 22nd Conference on Software Engineering Education and Training (CSEE&T '09), Hyderabad, India,* February 17 - 19, 2009.

Chase, J. D., Oakes, E., & Ramsey, S. (2007). Using live projects without pain: The development of the small project support center at Radford University. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education Conference. SIGCSE'07, Covington, Kentucky, USA.*

Liu, C. (2005). Enriching software engineering courses with service-learning projects and the open-source approach. *The 27th International Conference on Software Engineering (ICSE'05), St. Louis, Missouri, USA,* May 15-21, 2005.

Poger, S., & Bailie, F. (2006). Student perspectives on a real world project. *J. Comput. Small Coll. 21*(6), 69-75.

Purewal, T. S., Bennett, C., & Maier, F. (2007). Embracing the social relevance: Computing, ethics and the community. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education Conference. SIGCSE'07, Covington, Kentucky, USA.*

Rosmaita, B. J. (2007). Making service learning accessible to computer scientists. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education Conference. SIGCSE'07, Covington, Kentucky, USA.*

Schwaber, K. & Beedle, M. (2001). *Agile project management with Scrum.* Prentice Hall.

Song, K. (1996). Teaching software engineering through real life projects to bridge school and industry. *SIGCSE Bulletin. 28*(4), 59-64.

Tadayon, N. (2004). Software engineering based on the team software process with a real world project. *J. Comput. Small Coll. 19*(4), 133-142.

Biography

Dr. **Joseph T. Chao** is an Associate Professor in the Department of Computer Science at Bowling Green State University. He has taught courses in all aspects of the software development lifecycle, including programming, systems analysis and design, database systems, usability engineering, and software engineering. Prior to entering academia, Dr. Chao has more than seven years of industry experience in software development, including three years as Director of Software Development. His research focus is on software engineering with special interests in agile methods, programming languages, and object-oriented analysis and design. Dr. Chao is the Director of the Agile Software Factory at Bowling Green State University, which he founded in 2008 with a grant from the Agile Alliance. The Factory provides students with service-learning opportunities in software engineering. Dr. Chao holds an M.S. in Operations Research from Case Western Reserve University and a Ph.D. in Industrial and Systems Engineering from The Ohio State University.

Jennifer Brown is an instructor in the Scientific & Technical Communication Program within the Department of English at Bowling Green State University (BGSU). Jennifer teaches courses in introductory technical communication, online documentation, and professional editing. She has been teaching at BGSU since 2005, and previously worked as a technical writer/trainer for a software and internet marketing company. Jennifer has also freelanced as a technical writer, editor, and consultant for individuals and corporations. She earned her MA in Technical Communication from Minnesota State University, Mankato.