Becoming Agile using Service Learning in the Software Engineering Course

Brian Hanks
Fort Lewis College
Durango, CO 81301 USA
hanks_b@fortlewis.edu

Abstract

This experience report describes a three year journey toward agility in a software engineering course. Students in the course work in small project teams to develop an application for a real client using the service-learning model. In the first year, a formal plan-driven approach was used, and only two of four projects were completed successfully. A more agile approach was taken the second year, but there were still failing projects. In the third year a highly agile approach based on short iterations and user stories was used. This was more successful, as every project team delivered working software to their clients. This paper also discusses some of the challenges of managing and evaluating student work when using agile methods.

1. Introduction

This paper describes a three year journey toward agility in a software engineering course in which students develop applications for real-world clients. During this time the course has evolved from using a plan-driven approach with deadlines and milestones to using an agile approach based on iterations and user stories. In the process of this evolution I have also become more agile as an instructor.

At Fort Lewis College there is a strong emphasis on the liberal arts and the role of service-learning in education. As defined by Jacoby, “service-learning is a form of experiential education in which students engage in activities that address human and community needs together with structured opportunities intentionally designed to promote student learning and development” [8]. It is important to note the two components of service-learning: service to the community in a learning context. A successful service-learning project benefits both the students and the community.

In the computer science curriculum, the software engineering course is a natural place to include service-learning because of its project-based focus and emphasis on the complete software development lifecycle. The learning opportunities for the students are significant:

- They get to work on real projects rather than throwaway toy projects devised by the professor. This leads to greater student motivation and involvement with the projects.
- They gain experience working with clients who may be non-technical and may change their minds frequently. This gives them valuable real-world experience that is impossible to replicate in the classroom.

The experience is also valuable for the students professionally. The projects truly enhance the students resumés, and their clients are an excellent source of references. Together, these factors help our graduating students as they enter the job market.

Service-learning has been used elsewhere in the computer science curriculum with positive results. At Butler University, students who participate in service-learning projects show improved communication, teamwork, and organizational skills [10]. Other institutions using service-learning in the computer science curriculum include Grand Valley State University [5], Purdue and Notre Dame [9], and Southwest Missouri State [15].

2. Course Background

At Fort Lewis College, we teach software engineering in a two-semester sequence. The syllabus focuses on the entire software lifecycle from project initiation to installation. This is a project-focused course, in which students work in small teams using the service-learning approach to develop applications for community partners.

The course meets twice a week in a traditional classroom for two hours. Class time is spent using a mixture of traditional lectures, seminar style discussions of assigned readings, active learning exercises, and team meetings. The amount of time allocated for team meetings increases during the first semester as the projects get started. In the second semester, I usually allocate at least 45 minutes per class meeting for team meetings.
period to the group meetings. During these meetings I circulate among the groups to answer questions and offer guidance. I also allow the teams to meet with their clients during these periods, because it is a time when all of the students can attend without conflict.

This course sequence is required for students majoring in Information Systems (IS) and is an elective for our Computer Science (CS) majors. Prior student experience varies widely. The only pre-requisite is two semesters of programming, either in Java (CS majors) or Visual Basic (IS majors). The Java students have some exposure to JUnit and Eclipse, but the Visual Basic students have worked exclusively in the Microsoft Visual Studio.NET environment. Some students have much more experience, both in terms of coursework and in work-related experience via summer internships or part-time employment. Few of the students have had any formal experience in most aspects of software engineering, and have only vague concepts of software processes, requirements, design, testing, or project management.

This rudimentary student background leads to some significant challenges when trying to teach agile methods. As noted by Kent Beck [1], with agile methods you “turn the dials to 10”. How can you do this if you don’t even know what the dials are?

For this reason, I feel that it is essential to expose students to software engineering concepts from both the agile and plan-driven viewpoints. This gives them the perspective to evaluate and compare the approaches, to determine which approach might be better suited for a particular project, and to prepare them for a variety of working environments. As Gary Pollice noted in a SIGCSE 2005 panel, “we need to find the right ways to introduce students to agile practices while preparing them for all kinds of projects they might encounter in their professional careers. Just teaching agile methods is not sufficient for software engineering courses.” [2]

One of the challenges of the service-learning approach is finding suitable projects. They need to be challenging enough to engage a group of students for an academic year, but not too complex. Potential clients also have to be willing to spend time working with the student groups. It is also important for clients to realize that the students are inexperienced developers who may not successfully complete the projects.

Before the beginning of the academic year, I meet with campus and community partners to discuss potential projects. I work with them to understand the general scope of the projects, and make the final decision about which projects to accept.

During the third or fourth week of the fall semester, each client visits the classroom to present the basic requirements of their project. After attending the presentations, the students have to ‘apply for a job’ on one of the projects. This application consists of a cover letter and résumé identifying their skills and interests in the projects. I create the project teams based on these applications. Most students get to work on their first choice of project, but in some cases they are assigned to a different team to ensure that the teams are similarly sized and that the students on the teams have the necessary skill sets for the projects.

The approach that I have used to guide and measure the student teams has become more agile over the three years that I have taught the course. One measurement tool that I have used consistently has been a customer satisfaction survey. On this survey, I ask each client to indicate how satisfied they are with the project and the level of communication they have had with the student team members. I also ask them to identify any students who they feel did an exceptional job. The results of the survey directly affect the students grades on the project.

3. Course History

As noted above, the project approach has become more agile during the last three years, beginning with the 2004 academic year.

3.1. Year 1

In the first year, 18 students enrolled in the fall semester; 17 of them continued in the winter term.

As a new faculty member teaching the course for the first time, I decided to largely follow the approach of the previous course instructor, including using the same textbook [7]. I supplemented the textbook by having students read and critique articles on topics not included in the text, including agile development [6], risk management [3], open source software [13], and testing [11]

Ultimately, the text had a couple of serious weaknesses: (1) its organization and focus closely followed the waterfall model, and (2) it implicitly assumed that by following the steps it described all projects would be successful.

One major difference between my approach and that of the previous instructor was the use of service-learning projects. There were three service-learning projects: an intranet application to support the business school in its re-accreditation process, a multimedia application for the campus career center to help students improve their interviewing skills, and a web-based application for the career center that helped students develop professional résumés. I assigned students to the project teams (of five students each) using the process described earlier.

Three students were allowed to work on a project that they proposed: a web-based system that would allow stu-
students to sell used textbooks to other students. I acted as the 'client' on this project.

From an instructor’s perspective, the waterfall model has a lot of appeal. It is easy to describe and easy for students to understand. It fits well with the traditional academic model of assignments with due dates. This makes it simple to give all groups the same task (such as completion of the requirements specification) even though they are working on different projects.

From the students’ perspective, however, the waterfall model makes less sense. They don’t see the value of many of the tasks, and they are anxious to start the “real work” of coding. The waterfall model also has the same drawbacks here as it does in industry: requirements change, it is hard to estimate task duration, schedules slip, and testing suffers as a result if the project is completed at all.

During the course, students worked with their clients to identify and document the system requirements. The students found this very challenging, and their specifications were, for the most part, vague, ambiguous, and untestable. One reason for this was that the text book spent a lot of time discussing requirements, but did not give good examples or describe techniques for documenting them.

Only two of the projects were successfully completed. One of the failed projects could be attributed to a number of factors, mostly related to a lack of team cohesion. Students in this group failed to do the work that they committed to, frequently missed group meetings, and did not communicate with each other. When they finally realized that they were far behind schedule, they attempted to complete the project in the last two weeks of the course. Their weekly status reports indicated that they were working on the project, but in the final analysis this turned out to be largely untrue.

The students who worked on the used textbook application also failed to complete their project. A major reason for this was due to poor planning. They assumed that some tasks would be much easier than they actually turned out to be, and they overlooked some significant tasks. Because of this, they did not put much in much effort during the early part of the project because they thought that they had plenty of time.

Both of these failures could have been mitigated by not using the waterfall model. Problems would have become clear to both the students and the instructor earlier, which could have led to corrective actions.

3.2. Year 2

Due to the unsatisfactory outcome in year 1, I completely revised the course the second year. Because so many of the problems encountered by students the first year were related to requirements and project management problems, I replaced the textbook with two practitioner-oriented books: one on software project management [16] and one on requirements analysis [14]. I also used the book Rapid Development [12], which predates the agile movement but espouses many of the same philosophies: use iterative development, don’t waste time on unnecessary tasks, and work closely with your customer. This text was very successful – students liked reading it, and found the case studies very relevant and applicable. I supplemented these texts with many of the same readings used in the previous year.

This year there were only 11 students who enrolled in the course. Nine of them continued to the second semester. There were two projects: a web-based application to allow the college to better track alumni, and a system to allow the campus dining services to improve management of their catering operations.

I started the course with a comparison of software process models, including the waterfall, spiral, iterative development, and extreme programming models. I hoped that this would allow students to evaluate the strengths and weaknesses of the models, and apply an appropriate model to their projects. I allowed students to choose their own processes based on the characteristics of their projects, and to set their own schedules for project deliverables. For the most part, this was not successful – students didn’t have enough experience to make this choice effectively, and most tended to choose waterfall-like approaches, or fall back onto them. The students also tended to miss their self-determined deadlines; work was frequently turned in late and was clearly done at the last minute.

One team successfully completed their project this year. The other team did not complete their work, and this was largely due to interpersonal issues within the team: failure to meet commitments, lack of communication, and implicit expectations that someone else would do the work.

3.3. Year 3

After two years with only a 50% project success rate, I decided this year to require students to use an agile approach on their projects. I continued using the Rapid Development text based on student feedback from the previous year, again supplemented with readings from various sources. Instead of using the project management and requirements textbooks, I decided to address these topics on an as-needed basis in course lectures and during my informal meetings with the student groups.

There were 19 students in the fall semester; 15 of them continued in the winter term. Fortunately, the non-returning students were from three separate groups, so that none of the teams faced a serious personnel shortfall. In the first semester, the teams had 4 or 5 members, but during the second semester the team size was either 3 or 4 students.
I had difficulty finding enough appropriately sized service-learning projects this year. There were four projects this year; two of them were for external clients, and I was the client on the other two. However, these were not just busywork projects, as they fulfilled real needs of the computer science department. Although these projects may not completely match a true service-learning model, they satisfy many of the criteria: students work on a real project to meet a community need while learning about software development.

The external projects include a web-based membership management application for a non-profit professional organization and a desktop application to track resource usage by students in one of our campus student service organizations. This is the first time that I have used an off-campus project, and there were some additional logistical issues associated with it. Because the data being managed was associated with external individuals, it was necessary for the students and the instructor to sign a non-disclosure agreement with the client.

The projects for which I was the client include a programming assignment submission system and a web-based application for students to sign up for their required advising appointments. Although I was the client, the homework submission system will be used by all computer science instructors, and the advising system will be made available to all faculty on campus.

After assigning the project teams, I discussed some of the steps that the students needed to take on their projects. Because the projects started in the fifth week of the fourteen week semester, I required the project teams to plan for at least two iterations over the remainder of the semester. Naturally, because of their lack of experience, these plans were somewhat vague and in some cases misconceived.

For example, one group spent their first iteration defining stories and writing use cases. Their second iteration looked at database schema design. In some sense, this group mapped a waterfall process onto the iterations of the agile model. At the end of the first semester, this was the only group that was not able to demonstrate working software.

A second group had technical difficulties that they struggled to overcome. Some of this was due to their choice of development environment, and some was due to environmental constraints on the campus network. This group got so involved trying to work out the technical issues that their iteration plans became completely meaningless. This group also violated the YAGNI (“You ain’t gonna need it”) principle of simplicity and included a relational database in their system unnecessarily.

This led me to realize that the groups needed more training in iteration planning, user story management, and story point estimation. I spent the first lecture of the second semester with an in-depth review of this material, and gave much more direction to the groups about what they needed to do in their iteration planning meetings. I plan to revise the course syllabus to include this material earlier and more often next year.

Based on this, students were required to turn in an iteration plan at the beginning of every iteration. The plan is a one-page document that includes the iteration end date and a list of stories and associated story points that will be completed during the iteration. The groups were expected to work with their client to identify the stories included in each iteration. At the end of every iteration, the groups performed a retrospective, and turned in a one or two page report that identified the project aspects that went well, those that did not go well, and their plans for improvement. Together, these two documents allowed me to track group progress and intercede when necessary.

I let each group determine the durations of their iterations (although they had to have at least three in the second semester), and surprisingly to me they chose iterations ranging from one to three weeks. Since each student worked on the project for approximately four hours per week, this meant that the teams had only 12 to 16 student hours of effort available per week. As a result, a story point tended to be 2 to 4 hours of work, and the groups planned from 5 to 15 story points per iteration.

The student groups were required to estimate (using story points) every user story. Each group was allowed to define their own measure for a story point, but had to record the amount of time they spent working on each story to calibrate their scale and improve their estimation abilities. Students also used the concept of velocity to determine the number of story points to attempt in each iteration. Over the course of the semester, the students showed real progress in this area. In previous years the students either estimated very poorly or not at all, and failed to track progress against their estimates. The agile approach is clearly an improvement over previous experiences in this area.

At the end of the first semester, the clients indicated that they were unsatisfied with the level of communication that they were having with their project teams. The teams were keeping me relatively well-informed of their progress, but failed to maintain contact with their clients. This occurred even though the students were aware that client satisfaction was one of the items used to evaluate their projects. I had to reiterate the importance of frequent client communication with the students, and emphasized that one critical measure of a successful project is a satisfied client.

In addition to helping the teams move forward on their projects, iteration planning has helped me identify problems earlier. For example, one group had significant teamwork challenges, similar to those experienced by failing groups in the previous years. However, these problems became apparent much earlier due to the agile approach, when actions
could be taken to address the problems. As a result, the students became more aware that their behaviors were the source of these problems, and were able to modify them and still complete their project successfully.

The outcome was much more positive this year. All four teams delivered working software to their client. Both external clients were highly satisfied with the results, and have expressed interest in further projects next year. The two projects for the computer science department were also successful, and were piloted during the spring semester. Students in other classes who used the homework submission system were very impressed, and are looking forward to taking the software engineering course as a result.

There were still challenges however. The team that used a waterfall-like approach in the first semester tended to fall back on this approach in the second semester. Their was particularity apparent when faced with the end of semester deadline: the students worked all night to complete the final critical stories, and later admitted that they had been slacking some in their previous iterations. However, it was a valuable learning experience for them, as they understood that they should have taken a more agile approach.

Some teams did not complete every user story. For example, one project was delivered with only a subset of the desired reporting features. The advising sign-up application is also missing several features. However, the agile approach allowed every team to be successful. All four teams responded well to changing user requirements, and were able to deliver at least some functionality to their clients.

4. Challenges

Using agile methods in this course has required me to become much more agile in my teaching approach. Because each group is moving at its own pace on a unique project, they need to be introduced and guided on agile practices at different times. It is necessary to meet with each group individually at least once a week. This allows me to better measure progress, and to identify project and team issues that need to be addressed. The groups can get required instruction in these areas when they need it. While this may be less efficient for the instructor (because of the need to repeat instructions at different times to different groups), the students find it more relevant because it is immediately applicable to their projects.

A second challenge is teaching agile methods to students who have no understanding of what the practices mean in general. For example, how do you discuss test-first development with students who don’t know basic software testing principles? This makes it necessary to discuss both plan-driven and agile approaches to these practices. I don’t consider this to be a negative, however. As I discussed earlier, I believe that it is necessary to provide students with instruction in various software engineering processes so that they will be able to evaluate their pros and cons. This will also allow them to be productive using whatever process is followed by their future employers.

4.1. Communication

Regardless of the amount of stress put on this, it takes a lot of effort to convince students to communicate regularly with their clients, particularly in the first semester. They tend to meet once or twice at the beginning of the project, and then they just want to jump into development. The results of the first customer satisfaction survey are often a wake-up call to the students, and lead to much improvement in this area.

No matter how hard you try, some student groups just don’t (or won’t) communicate among themselves. The agile process itself does not seem to have much impact on this, although the iteration planning and retrospective practices make the impact of this lack of communication apparent much earlier, and allow the instructor to assist the groups in overcoming their problems.

4.2. Grading and Evaluation

In a traditionally run course, every student group is expected to turn in work at the same time. This is not a realistic option in an agile setting where the student groups determine their own iteration schedules, and decide what stories belong in each iteration with their client. In this situation, the instructor has to be more willing to accept work at different times.

Instead of only evaluating the project itself, I have put a lot more emphasis on process evaluation. The iteration planning and retrospective documents are a key element of this, as are the customer satisfaction surveys. These allow me to evaluate the students’ work to answer the following questions:

- Are the students planning for iterations?
- Are they doing post-iteration reviews to tune their process?
- Is their customer satisfied with their progress and level of communication?
- Are they following agile principles, or are they just doing a waterfall model wrapped in iterations?

I also ask the students to evaluate the performance of their teammates three times: once at the end of each semester, and once in the middle of the second semester. The students evaluate themselves and their teammates on the quality of their work and the quality of their effort. I
find that the students are honest in their evaluations—they know when they aren’t doing their share, and this approach is effective in getting them to admit it. I use the results of these surveys to adjust individual student grades (either up or down) on the projects.

4.3. Other Agile Practices

Due to the wide variety of different technologies and environments used in the projects, it is difficult to implement some agile practices such as automatic builds or test first development. These practices are also challenging due to the widely varying backgrounds of the students. For example, four different automated build environments would have been required in year 3. The students are not capable of setting these up themselves, and I don’t have the resources available to manage this. Similarly, it is difficult to implement test first development.

Pair programming is also difficult for some teams due to schedule challenges. However, while this would be helpful for many of the groups, it is not as important educationally because the students have been exposed to this practice in their introductory programming courses.

5. The Future

Although it has not been a completely smooth transition, I feel that the overall journey toward agility in this course has been fruitful. Project failures were an unfortunate outcome in the first two years I taught this course, but in the third year every group delivered functional software that met at least some of their client’s needs.

Next year, I plan to expand my emphasis on user stories and iteration planning. I will be using the book User Stories Applied [4], and will supplement this text with early emphasis on iteration planning and customer communication. I am also considering how to include some other agile practices, such as test-first development, into the course.

References