An Agile Software Engineering Course with Product Hand-Off

Jason B. Shepherd
Buena Vista University
Storm Lake, Iowa, U.S.A.
shepherd@bvu.edu

ABSTRACT

This paper describes a novel design for an agile software engineering course that emphasizes keeping product artifacts updated throughout development. The signature transformative event in the course is the mid-semester project "hand-off," at which point teams trade projects with other student teams and must make immediate progress despite no prior knowledge of the new project's design, coding conventions, or documentation. Course features are described along with their implementation and assessment.

KEYWORDS

Software engineering education, agile software engineering, peer learning, hand-off

ACM Reference Format:
Jason B. Shepherd. 2018. An Agile Software Engineering Course with Product Hand-Off. In *SEEM'18: SEEM'18:IEEE/ACM International Workshop on Software Engineering Education for Millennials , May 27-June 3 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3194779.3194792

1 MOTIVATION

In industry, one of the most notable trends towards managing change in a project is the shift towards agile software development [6]. In response to this trend, educators have sought to utilize agile methods in software engineering courses, often using agile frameworks such as eXtreme Programming (XP) [2] or Scrum [10] to guide development. Activities intrinsic to agile frameworks (such as pair programming, test-driven development, informative workspaces, etc.) encourage both active learning and peer learning. This type of learning is ideal for engaging today's millennial students.

One goal for software engineering courses is to prepare students who will ultimately enter real-world software development environments. By extension, a software engineering course that utilizes agile methods should help students value readable code and informative documentation in a software project. However, using agile methods does not automatically create this appreciation in students. In fact, the opposite is equally likely. Students tend to be motivated by grades. If students view completing weekly tasks for a grade as the sole goal, there is no obvious incentive for them to properly maintain their project. After all, they already feel they "know" their code. Students are skeptical of the importance of the design model, clean interfaces, and readable or well-commented code in preparation for the inevitability of change. Given that students often seek immediate short-term results without considering long-term consequences, it is necessary to encourage students to focus on project refactoring beyond the simple application of agile methods.

This paper describes the design of an undergraduate software engineering course that uses agile methods, modern toolsets, and a series of graded activities to incentivize the proper updating of project artifacts. The most unique of the graded activities is the project *hand-off*. The hand-off is a one-week period where student teams switch projects. During this week, a new team gains temporary ownership of another team's project. This new team must continue the development tasks previously planned by the original project team during the original team's weekly planning meeting. The success of the new team is dependent to a large degree on how well the project has been maintained to that point. At the end of the week, the student teams provide feedback to each other.

The rest of this paper is organized as follows. Section 2 describes prior work in creating agile software engineering courses. Section 3 presents specifics of how the hand-off is conducted and how the exercise is assessed, along with other activities that incentivize product upkeep. Section 4 shares observations made from three distinct offerings of this course. Section 5 offers conclusions and lessons learned.

2 RELATED WORK

The infusion of agile software development methods in undergraduate software engineering courses is not a new endeavor in any sense. Descriptions of courses and their prominent features have been presented in [4]. Some of these courses ascribe closely to either XP or Scrum, or they use a prescribed process that is a combination of both (similar to the course design described in this paper).

The iterative nature of agile methods is one that is espoused by the ACM/IEEE computer science curricular guidelines [7]. Iterative development gives students a chance to practice and apply lessons learned from one development cycle to the next. Rapid feedback is paramount, but it can be difficult for instructors to keep up. Goldman et al. [6] suggest the use of graduate students to serve as process coaches to alleviate this problem. A majority of students' time should be spent developing and practicing agile methods, yet there must be a way to formally disseminate knowledge about agile methods and other software engineering concepts. Santos et al. [9] suggest the use of mini-lectures to share relevant information prior to when students need it.

SEEM'18, May 27-June 3 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5750-0/18/05…$15.00
https://doi.org/10.1145/3194779.3194792

Project selection is important when teaching agile methods. Stapel et al. [12] suggest that instructors should have students develop new software as opposed to enhancing already written programs. This allows them to focus on practicing agile processes rather than spending time acquainting themselves with an existing project's idiosyncrasies.

Course materials have been developed to support such agile software engineering classes. A frequently adopted textbook for agile software engineering classes is Engineering Software as a Service (ESaaS) by Armando Fox and David Patterson [5]. ESaaS is paired with an extensive set of online resources and recommended frameworks and tools that instructors can use to deliver their courses. The resources make ESaaS very attractive since much of the prep in an agile software engineering course is spent selecting frameworks and tools, and then preparing tutorials for students on how to use these frameworks and tools.

3 COURSE DESIGN

3.1 Overview

The course described in this paper is intended for second and third year undergraduate computer science students who have had a prerequisite course in data structures. This course has been offered in three distinct semesters with a course limit of 24 students per semester.

Students are partitioned into teams of at least four students during the first week of the class. Projects are semester-long and chosen by the instructor. Customers are on-campus, and the instructor acts as the process coach. Teams are always arranged to make an even number of teams. This allows each single project to be given to two teams. Teams that have the same project will participate in the product hand-off with one another shortly after mid-term. The hand-off is described in Section 3.3.

Development cycles in this course are called *sprints*. Sprints do not begin until the third week in the semester. The first two weeks are spent learning about general software process models, our specific development process, how to use GitHub [8], and how to complete tasks within our development process using GitHub and Slack [3]. The development process we use is a blend of agile methods that borrow concepts such as pair programming, test-driven development, and informative workspaces from eXtreme Programming (XP) as well as a feature/issue backlog concept from Scrum. Sprints are short, one-week endeavors that draw their tasks from the backlog.

The sprints, product hand-off, and individual self-evaluations are the major evaluative activities in the course. Figure 1 shows when these activities occur. Each activity is described in detail in the subsections that follow.

3.2 Sprints

Sprints often begin with a mini-lecture (approximately 10 minutes), either to introduce various software engineering topics or to provide well-timed hints as students begin to struggle with an aspect of their projects. Then, students depart the lecture classroom and head to their designated "offices" – rooms within the building reserved for each team that serve as XP's notion of informative workspaces. Teams select one team member to serve as the project manager

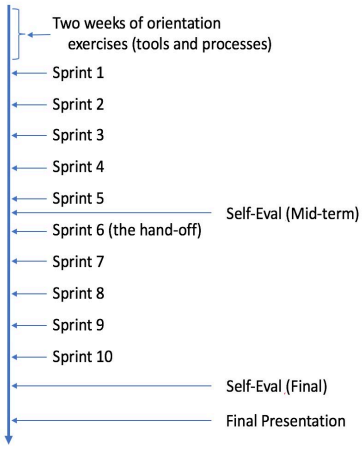

Figure 1: Course schedule

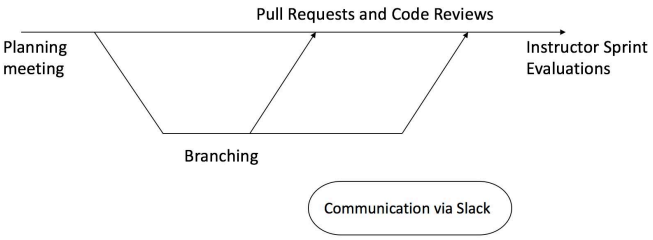

Figure 2: Sprint activities

(PM) for the sprint. The team conducts a planning meeting, during which the team identifies features/issues to be worked on in the current sprint. Since teams' projects are hosted on GitHub, the PM enters the issues, assigns pairs of developers to each issue, and then groups the issues together using a GitHub milestone for the sprint.

Pairs commit code within their own GitHub branch and then utilize GitHub's *Pull Request* mechanism to bring their branch changes back into the master branch after a code review is conducted by other team members, who serve as reviewers (see Figure 2). Commits and pull requests must be done properly to ensure that all work is traceable back to a now-closed GitHub issue. At the end of each sprint, the instructor evaluates the team's progress in terms of how well issues were handled from a procedural standpoint, and how thoroughly the team conducted code reviews within the context of a pull request. Each of these activities contribute to the overall maintainability of the software project, though in a way that is mandated by the course and the instructor. The evaluation form for each sprint milestone can be found at [11] and is free to reuse under the MIT License [1].

Whenever teams are not in class, they communicate at least initially using Slack. Each team has their own private Slack channel that they can use to notify one another of their progress, raise questions about design issues, negotiate a time to meet in person, or converse with the instructor.

3.3 Product Hand-off

By mid-term, teams have completed five sprints. In Sprint 6, teams have their regular planning meeting to choose the issues for the sprint, but they do not assign developers to those issues. At this point, each team will switch projects with the other team who has been assigned the same project. Each *new* team will try to complete the issues the "old" team left for them in one week's time. This is known as the hand-off. Teams may not communicate with one another; the new team may not interact with or ask questions of the old team. Thus, the new team's chances of success depend solely on the quality of the old team's project artifacts. Hand-off between teams with the same project allows this phase to run more smoothly since teams already understand what the product does at a high level.

At the conclusion of Sprint 6, the new team fills out the hand-off evaluation form. The form guides the new team in addressing the project's status categorically. The categories are:

(1) Install, Build, and Run – Does the project include a `README` file that explains how to setup a development environment and how to build and run the project. This includes a description of how to run unit tests.
(2) Issue Management – Can we tell what happened in the previous sprint? Is it clear what is scheduled to happen in the current sprint?
(3) Code – Is the codebase well organized? Does it follow stated coding conventions and use informative comments? Is the codebase devoid of obsolete files and does it include unit tests with good test coverage?
(4) Documentation – Is the Wiki up to date and informative? Does it include information that could be used to create issues/feature requests for future sprints? Is there design documentation and is it current?

The purpose of the students' comments is purely formative. It is not used to produce a summative overall score. Further, the hand-off evaluation form is not completed until the end of the sprint in order to give the new team time to reflect on the state of the project and on their own work. It also gives them time to "cool down" from any early sprint frustrations that are inherent to adopting a new project. The old team is then able to see the comments at the beginning of Sprint 7.

The hand-off evaluation comments are not used to score the old team's efforts in the hopes of eliciting more honest commentary. Past experience indicates that students are reluctant to grade one another for fear of retribution if the roles are reversed.

Once the evaluations are completed, the instructor grades the thoroughness and quality of the new team's evaluation. The instructor's evaluation introduces accountability to the new team as reviewers of the old team's project. The new team knows they must do well in reflection and in providing feedback in order to get a good grade, which incentivizes good feedback. The end result is effective peer learning.

Figure 3 shows the process of evaluation during the hand-off and the actors involved. When Sprint 7 starts, teams return to their original projects for the remainder of the semester. The hand-off evaluation form and a rubric for instructor evaluation of the form can be found at [11]. It is important to note the different angles

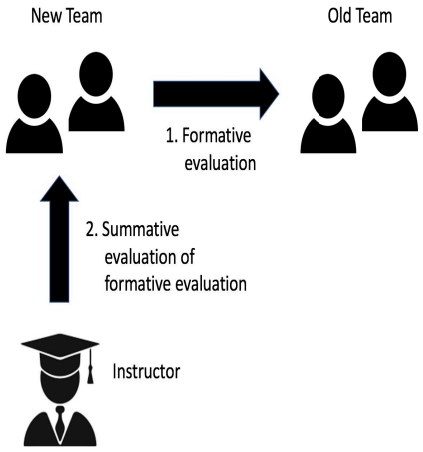

Figure 3: Order of evaluation during hand-off

through which sprints and hand-offs are used to motivate students and assess their learning. The sprints assess student work from the instructor's point-of-view. The hand-off assesses product quality from other students' points-of-view, although the assessment is purely formative.

3.4 Self-Evaluations

The assessed activities explained in Subsections 3.2 and 3.3 are team-level assessments. Individual performance must also be measured. The self-evaluation is intended to mimic the self-evaluations often done in industry as part of compensation and performance review. The self-evaluation considers students' reflections of their work along with supporting evidence (commits, pull requests, code reviews, unit tests contributed, contributions to design documentation, etc.). The instructor uses the self-evaluation and corroborating evidence to grade the self-evaluation using a rubric. The self-evaluation and accompanying rubric can be found at [11].

4 OBSERVATIONS

The motivation for this course design was to impress upon students the importance of good documentation and well-maintained code in preparation for the inevitability of change in a software product. We saw evidence that students bought into these ideas, both during the hand-off and again at the end of the semester when asked to reflect on the hand-off in the context of the whole semester.

One thing we looked for was thoughtfulness and variety of comments on the hand-off evaluation forms. Comments submitted on the hand-off evaluations were numerous and ranged from critical (e.g., missing functions/code) to cosmetic (e.g., typos in documentation). Table 1 shows the types of comments made by new teams to the old teams, with the comments grouped by the project area addressed by the comment.

The variety of comments proved helpful to students on future sprints. Nearly all comments made by the new team were addressed by the end of Sprint 8, and all those addressed were tied to a specific issue the team created in GitHub. In fact, teams improved overall

Comment Type	Mentions
Code - poorly organized	4
Code - obsolete	4
Code - missing	2
Code - poor error messages	2
Code - poor comments	5
Docs - bad project setup instructions	6
Docs - bad build instructions	2
Docs - typos	3
Docs - missing/incomplete design	3
Docs - missing content/out-of-date	4
Tests - missing	6
Tests - not working	4
Issues - not grouped into milestones	7
Issues - lacking detail	8

Table 1: Types of comments from hand-off evaluations

on how they managed issues within GitHub in Sprints 7 - 9. Prior to the hand-off, GitHub issues contained minimal information and little evidence of team collaboration. After the hand-off, issues incorporated not only detailed information needed to fix the issue, but also detailed commentary by those assigned as the need for design refactoring arose.

Student reactions to the hand-off and the course in general came from course evaluations. Course evaluations were conducted anonymously through the Web during the last two weeks of the semester. The evaluation itself consisted of both closed-end questions and open-ended responses. One open-ended question asked students to reflect on the hand-off. Two students expressed a distaste for allowing another team to modify their code and temporarily losing control of their project, which also suggests a strong sense of product ownership. Over 75% of those answering the question stated the hand-off was a worthwhile activity. Approximately 50% of those answering the question articulated an increase in their own self-efficacy with respect to future workplace preparedness because of the hand-off.

5 CONCLUSION AND LESSONS LEARNED

Each semester this course was taught, both project selection and the hand-off were treated differently in an attempt to see what worked best. It became clear that there are three factors to consider when implementing product hand-off in a course.

5.1 Hand-Off the Same Project

In the first semester product hand-off was attempted, teams did not have the same project. The rationale for this practice was that in professional practice, developers may need to ramp up quickly in projects with little a priori knowledge. To compensate for learning a new project, the hand-off sprint was extended from one week to two. Despite this compensation, the hand-off failed. In hindsight, it is reasonable to expect a seasoned industry professional to ramp up quickly on a project, but it is not reasonable to expect the same from a second year computer science student.

If it were desirable to perform the product hand-off with different projects, it may still be possible if the projects to be handed off were related in some way. For example, one project might be an app that uses a sophisticated API, and the other project might be an implementation of that API. Pairing projects in this manner might help ease the transition from one project to another during the hand-off; however, projects of this nature would be very difficult to invent every semester.

5.2 Hand-Off Earlier

Students have suggested that we conduct the hand-off earlier than Sprint 6. They indicate two incentives for doing so. First, students may receive the benefits of the hand-off sooner. An earlier appreciation for a well-kept software product would give students more time to use their development sophistication. Second, having the hand-off in Sprint 6 does not give teams much remaining time to polish their finished product. Students noted this was particularly frustrating, especially when the new team made changes that the old team found undesirable and in need of refactoring.

5.3 Choose Familiar Languages and Frameworks

For reasons similar to why we tried handing off different projects, we also experimented with handing off the same project but where the projects were written using different languages/frameworks. We found second and third year students are not yet adept at learning new languages/frameworks, and this only distracts from the main focus of practicing agile methods.

REFERENCES

[1] 1988. MIT License. https://opensource.org/licenses/MIT. (1988). Last retrieved 2018-02-09.

[2] Kent Beck and Cynthia Andres. 2004. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional.

[3] Stewart Butterfield, Eric Costello, Cal Henderson, and Serguei Mourachov. 2013–2018. Slack. https://slack.com/. (2013–2018).

[4] Jennifer Campbell, Stan Kurkovsky, Chun Wai Liew, and Anya Tafliovich. 2016. Scrum and Agile Methods in Software Engineering Courses. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 319–320. https://doi.org/10.1145/2839509.2844664

[5] A. Fox and D.A. Patterson. 2013. *Engineering Software as a Service: An Agile Approach Using Cloud Computing*. Strawberry Canyon LLC. https://books.google.com/books?id=3kqjmwEACAAJ

[6] Alfred Goldman, Fabio Kon, Paulo J. S. Silva, and Joseph W. Yoder. 2004. Being Extreme in the Classroom: experiences Teaching XP. *Journal of the Brazilian Computer Society* 10 (11 2004), 5 – 21. http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-65002004000300002&nrm=iso

[7] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA. 999133.

[8] Tom Preston-Werner, Chris Wanstrath, and PJ Hyett. 2008–2018. GitHub. https://github.com/. (2008–2018).

[9] Viviane A Santos, Alfredo Goldman, and Carlos D Santos. 2012. Uncovering Steady Advances for an Extreme Programming Course. *CLEI Electronic Journal* 15 (04 2012), 2 – 2. http://www.scielo.edu.uy/scielo.php?script=sci_arttext&pid=S0717-50002012000100002&nrm=iso

[10] Ken Schwaber. 2004. *Agile Project Management With Scrum*. Microsoft Press, Redmond, WA, USA.

[11] Jason B. Shepherd. 2018. SE: Agile Software Engineering course materials. http://github.com/jbshep/SE/. (2018).

[12] Kai Stapel, Daniel Lübke, and Eric Knauss. 2008. Best Practices in Extreme Programming Course Design. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, New York, NY, USA, 769–776. https://doi.org/10.1145/1368088.1368197