A Balancing Act: Teaching and Researching Agile Software Development

David P. Harvie and Arvin Agah
United States Military Academy The University of Kansas

Abstract

Conducting research in an academic setting can be challenging. The requirements of responsible and effective data collection must be balanced with the obligation to provide the students with a fair and productive academic experience. This paper presents the development of a semester-long undergraduate Computer Science course on software engineering that simultaneously taught students agile software development using Scrum, while collecting data on a proposed methodology called Targeted Scrum. Small team sizes and shorter iterations facilitated students’ opportunities to learn and increased the number of data points for research. The innovative classroom and proper utilization of collaborative tools provided a more realistic environment to learn and practice agile software development, and simplified data collection. Safeguards were taken to ensure that student participation in the research was strictly voluntary. The course design and execution resulted in very positive student learning experiences, as well as gathered valuable data for research purposes.

Keywords

Scrum, Agile Software Development, Mission Command, Line of Effort, Software Engineering Education

Motivation

Currently, software engineering and military command and control (as expressed by mission command) are striving to be more agile and resilient to changing environments. Neither field has the panacea for complex problem solving. However, the military has developed some good techniques for responding to change due to over 11 years of continuous combat.

The authors used inspiration from mission command\(^1\) to improve a specific agile software development method, namely, Scrum\(^2,3\). Two specific weaknesses with Scrum are a lack of initial planning and a lack of an overall architecture. This approach, called Targeted Scrum, addressed the first weakness with the addition of a Product Design Meeting at the onset of the Scrum process. This meeting is subsequently repeated in Scrum to assess the validity, and to make necessary changes to the product's design. Targeted Scrum addressed the second weakness with the addition of two artifacts, namely, the product's end state and lines of effort (LOEs). Both of these artifacts are derived from mission command. They are used to establish and communicate the product's overall architecture. The additional meeting and artifacts would assist with the initial planning with Scrum and contribute to a more stable product framework.
Introduction

How does one design a course that simultaneously collects research and educates students on agile software development? This paper focuses on the design of a semester-long agile software engineering course developed for upper-level undergraduate students in Computer Science and Computer Engineering at the University of Kansas to conduct research on a proposed agile software development methodology\(^4\). Students showed great interest in this course because it offered the students a unique opportunity to learn and gain practical experience with agile software development. Thus, the challenge was to construct a course that facilitated the conduct of experiments with agile software development while fulfilling the obligation to the students for a productive and fruitful academic experience.

Research Hypothesis

The research hypothesis is that modifications to agile software development based on inspirations from mission command can improve the software engineering process in terms of the planning, prioritizing, and communication of software requirements and progress, as well as improve the overall software product.

The authors tested this hypothesis by developing two software projects in a classroom environment, one using a traditional agile software development methodology, Scrum, and the other using the proposed agile software development methodology, called Targeted Scrum. The objective was to test the hypothesis for agile software development specifically to Scrum with the intent that the positive results may potentially lead to applying these inspirations to agile software development methodologies as a whole\(^5,6\).

Related Work

Scrum is frequently used to teach agile software development in a software engineering classroom environment. Devedžić and Milenković\(^7\) ran multiple case studies using Scrum in undergraduate and graduate courses. Based on their experience, they recommended that Scrum teams should be small with four to five students and that the Sprint iteration length should be one to two weeks in order to generate more iterations. They also suggested that the Scrum teams self-organize and that the teacher fulfill the role of the software client. Mahnic\(^8\) implemented Scrum in a software engineering capstone course. In that capstone course, the first three weeks were dedicated to teaching Scrum, developing user stories, and project planning. There were three Sprints of duration of four weeks each. Recognizing the other academic demands placed on the students, the daily Scrum meetings were held twice a week. Wagh\(^9\) implemented Scrum in a fifth semester software engineering course with associated lab period. The Scrum teams consisted of five to six students, with two Sprints of approximately one month each excluding examination days. The teams were also asked to use the Eclipse Integrated Development Environment (IDE)\(^10\) to develop their code. Werner and Ross\(^11\) implemented Scrum in an undergraduate software engineering course. Their Scrum teams consisted of five to seven students, and there were three Sprints during the course. The role of Scrum Master rotated in the team for each Sprint. It is important to note that these studies involved teaching Traditional Scrum using one project for the associated courses.
Desired Learning Outcomes and Course Structure

This course, labeled EECS 690: Agile Software Development was a special topics course, and it was the only course offered specifically for agile software development during the Spring 2014 semester. The class met twice a week, Mondays and Wednesdays, for 75-minutes per class time. The course had two desired learning outcomes:

1. Understand the history and philosophy of agile software development methodologies
2. Learn and apply an agile software development methodology by creating and developing two software projects

In order to accomplish those two learning outcomes, the course was divided into three major components. The first component was the introductory phase lasting about three weeks. During this phase, the students were introduced to the history of agile software development and formally instructed in both Traditional Scrum and Targeted Scrum. While Scrum was the primary focus of this introductory phase, the students were also introduced to the Agile Manifesto\textsuperscript{12} and Extreme Programming (XP)\textsuperscript{13}. The second component was Project 1 which lasted about five and one-half weeks. The course was scheduled so that the Project 1 code development was completed prior to Spring Break. The students presented Project 1 briefings upon their return from Spring Break. The final component was Project 2 which similarly lasted about five and one-half weeks and culminated at the conclusion of the course.

Student Profile

The students were surveyed at the beginning of the course to capture some basic demographic information, their previous experience with developing software, and their knowledge of Java\textsuperscript{14} and Eclipse IDE. All 26 students answered this initial survey. All enrolled students were undergraduates with seven juniors and 19 seniors. Twenty-one students (85\%) were computer science majors, four students (15\%) were computer engineering majors, and one student (4\%) was an information technology major. All of the students had previous academic experience with software engineering.

Nearly three-fourths of the class (19 students) had previous experience developing software in industry or as part of an internship. The majority of those 19 students had industrial/internship experience of a year or less. It is significant that a majority of those with industrial/internship experience used some form of Agile Software Development methodology as part of that experience.

Surveying the entire class, seventy-seven percent of the class (20 students) had experience with Agile Software Development prior to the start of the course. Scrum was the most popular Agile Software Development with 18 students (69\% of the class) having used Scrum. The next most popular Agile Software Development methodology used was Test Driven Development with only seven students (27\% of the class) having experience with that methodology.

Class Environment
The class was held in a model classroom that was designed for hybrid teaching (flipped classroom). This state of the art classroom greatly facilitated group collaboration through the use of half-oval tables which easily accommodated 5-6 students each. The focal point of each desk was a computer monitor that the students could use to display an image from a computer or display a physical document using the document scanner. The instructor could also display images from the instructor computer station to each desk monitor. This desk layout allowed the student teams to write and edit code and artifacts collaboratively. Also, each desk had a microphone that could broadcast over the classroom speakers. This allowed a student to brief the entire class clearly without having to raise one's voice. Another important feature of this classroom is that all the walls are covered with whiteboards. These whiteboards enabled the students to further plan, design, and collaborate together.

It is important to note that this model classroom was shared with other classes. This meant that students collaborating on a whiteboard had to capture that collaboration prior to having to erase the board at the end of class. Also, this shared classroom environment meant that teams could not leave posted charts or graphs on the whiteboards. Overall, this classroom was a great venue to host a class on agile software development since the layout and the technology were specifically designed to facilitate collaboration.

**Balancing Education and Research**

The instructor in this course had to simultaneously balance the roles of educator and researcher. As an educator, the instructor had to teach the students the requisite knowledge and skills necessary to achieve the desired educational outcomes of the course. This responsibility included teaching a novel hybrid of Traditional Scrum, Targeted Scrum, in such a manner that the students could successfully implement it. The risk of the educator role is that the instructor could possibly bias the students toward the novel hybrid affecting the research. As a researcher, the instructor had to be somewhat detached to how the students used and assessed both methodologies. The risk of the researcher roles is that if the instructor was too detached that students may fail to achieve the desired educational outcomes.

The balancing of these two roles were assisted by process of going through an Institutional Review Board (IRB). Since the research involved human subjects via surveys, observations, and interactions, the protocols had to be submitted for approval to the IRB. The IRB at the University of Kansas is the Human Subjects Committee of Lawrence (HSCL) which is part of the University of Kansas Office of Research. There were three keys points to these protocols. The first key point is that participation in the research was strictly voluntary. A student who signed up for the course could choose not to participate in the research without any repercussion. Also, a student who elected to participate in the research could choose to withdraw from the research at any time without consequence. The second key point is that all personally identifiable information would be removed. The participating students were assigned a class identification number (1-26) which was not associated to any other identification number so that could not be personally identified later through this research. Also, the results of the surveys would be aggregated and individual surveys purged no sooner than May 2015 in order to provide an increased measure of anonymity. The third key point is that the surveys were redacted and evaluated only after the completion of the course and submission of grades. This was done so
that the participating students would not feel any pressure or influence to answer surveys in any particular manner.

The next balancing act of education versus research was determining the number and complexity of software development projects. The course was limited to one semester which meant that all the research had to be collected during that one semester. Since the goal of the research was to compare two software development methodologies, it was necessary to have two projects in order to make that comparison. Having two software development projects actually fit well within the desired learning objectives. The two projects enabled the students to learn from their first project and apply that knowledge to their second project. The tradeoff of increasing the number of projects to two is that the complexity of those projects had to be reduced. More complex projects would have allowed a more strenuous assessments of both software methodologies. However, this increased complexity would have come at the expense of the students being unable to reasonably complete the projects and potentially not achieve the education outcomes. Therefore, the complexity of both projects had to be set by the realistic expectations of what the students could achieve given approximately one month per project.

The final balancing act of education versus research was determining the size and composition of the individual teams. Scrum teams typically have 7 members. The size of the Scrum teams in this course was set to 3-4 members. From a research perspective, the smaller teams resulted in a greater number of teams allowing for more data points for the research. Also, the smaller team size would make the communication and collaboration tasks simpler. The downside is that both software methodologies' ability to handle the communicative complexity of larger teams could not be fully assessed. From an education perspective, the smaller team size helped facilitate team meetings outside of class. The only times that the teams could be guaranteed to meet together were during the twice-a-week class meetings. In addition, the small team size allowed all of the students to take on the roles of Product Owner or Scrum Master at some point throughout the semester. Rotating the Scrum roles had a marginal risk of affecting the research as it introduced a new variable between the two projects. However, the educational experience of the students applying agile software development in different roles far outweighed that minimal risk. The students in the course knew each other from previous courses. Therefore, they were allowed to self-organize into teams of 3-4 members during the first class period.

Protocol

Given the time constraints of a one-semester course, two software development projects were developed and assigned. The Sprint duration was set to two weeks, and a single project consisted of two Sprints. For the first project, half of the teams developed software using Traditional Scrum while the other half used Targeted Scrum. For the second project, the teams switched development methodologies. The purpose of assigning half of the teams to Traditional Scrum while the other half is assigned to Targeted Scrum is to eliminate the effect of students performing better on the second project from biasing the results in favor of whichever methodology is used during the second project. The instructor served as the software client for all teams for both project in order to retain and communicate a common set of objectives. The programming language for all projects was standardized to Java. Students were encouraged to develop their code utilizing the Eclipse IDE in order to take advantage of plugins developed for
that IDE. The students were informed that their code would be analyzed using the Eclipse Metrics 1.3.6 plugin.

For each project, there was an assigned Product Owner and a Scrum Master. At the onset of a project, the instructor gathered all the Product Owners together. Then, the instructor, acting as the client, gave the Product Owners a verbal description of the desired software product. Some product facts, such as specific prices for goods or services were given to the Product Owners in electronic form. However, the main requirements of the product were only provided verbally. The Product Owners were then responsible for taking this information back to their Scrum teams. Any question that the Product Owners asked the client was answered, and the answers were communicated to the teams via electronic document on the course Web site. This electronic document ensured that all teams had access to the same information at all times. Upon the conclusion of the first Sprint of the project, the client would once again gather the Product Owners and convey an additional requirement. This new requirement would necessitate the Sprint teams to re-evaluate their software design and architecture.

The projects were designed so that they could be reasonably finished within a four-week time frame. It was also attempted to make the difficulty level of the two projects equivalent in order to compare the results from the two projects.

**Empirical Data**

The empirical data consisted of four sources: (1) Software development artifacts; (2) Student surveys; (3) Instructor observations; and (4) the developed software code. Software development artifacts such as Product Backlogs, Sprint Goals, and LOEs were necessary for both the students' academic evaluations and the understanding of how the teams determined, prioritized, and translated the project requirements into delivered code. Initially, the intent for the students was to submit their artifacts via e-mail to the instructor at established deadlines. However, most of the teams used Google applications such as Google Docs, Google Sheets, and Google Slides to collaborate on their artifacts. Thus, the students to give the instructor permission to link to their collaborated artifacts, and then those artifacts could be archived to a separate drive for evaluation once the students e-mailed the instructor that the specific artifact was ready to be downloaded.

The second source of empirical data came from the two end of project surveys and the final comparison survey. Those two surveys, as well as an initial survey on the students' backgrounds and knowledge of agile software development, were created using Google Forms. Google Forms was chosen because surveys could be easily created using this tool, and it automatically sent the results of the surveys to a Google Sheets spreadsheet that instructor could analyze.

The third source of empirical data came from instructor observations of the teams' interactions and meetings. Given that the research took place in an academic setting, the only times that students could be mandated to meet was during the class time. The teams were directed to use specific class times for the meetings directed by the Scrum methodology used, such as Sprint Planning, Daily Scrum, and Sprint Reviews. During the course of these in-class meetings, the instructor would observe and record how well the teams collaborated and communicated.
The three sources of empirical data so far described pertain solely to the process. Process, however, is not the end state for either Traditional Scrum or Targeted Scrum. Ultimately, the students are developing a product - software. A “better” process that results in a “poorer” product may not actually be a better process. Thus, it was also necessary to compare the software development by both processes. Teams were required to submit their executable code and source code. Both the executable and source codes were necessary in both the academic as well as research evaluations of the teams. All teams used code repositories to collaboratively develop their code. The teams either used GitHub\textsuperscript{18} or Bitbucket\textsuperscript{19} for their repositories. The instructor was given permission to link to the teams' repositories. The instructor would clone the code to a separate drive when the teams indicated that the code was ready to be submitted. This submission technique was highly effective and realistically replicated how software engineers would develop code in an industrial setting.

**Educational Outcomes**

The 26 students in the course met the two desired educational outcomes. In their seven teams, the students successfully applied agile software development methodologies to create working software that met customer requirements. The course was requested to be taught again based upon the positive feedback regarding the course and the demand for the subject material. As a result, this course was taught, with modifications, in Spring 2015.

One part of assessing the educational outcomes was assessing the common artifacts of both Traditional and Targeted Scrum (i.e. the Product Backlogs and the Sprint Backlogs). These artifacts were graded and then rank ordered (1 represents the highest grade) to provide a comparison (see Table 1). For Project 1, the three teams using Targeted Scrum had a better average ranking (3.0) than the four teams using Traditional Scrum (4.5). Once the teams switched Scrum methodologies for Project 2, the four teams now using Targeted Scrum had a better average on their graded artifacts (3.5) than the three teams now using Traditional Scrum (4.3). It appears that the process of Targeted Scrum does assist the students to produce better Scrum artifacts. It is also important to note that the difference in rankings between the two sets of teams decreased from 1.5 to 0.8 suggesting that the students as a whole improved in producing Scrum artifacts after using both Traditional and Targeted Scrum.

<table>
<thead>
<tr>
<th>Team</th>
<th>Project 1</th>
<th></th>
<th>Project 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Artifacts Grade (Rank)</td>
<td>Final Product Grade (Rank)</td>
<td>Artifacts Grade (Rank)</td>
<td>Final Product Grade (Rank)</td>
</tr>
<tr>
<td>A</td>
<td>93.0% (7)</td>
<td>91.4% (3)</td>
<td>97.3% (4)</td>
<td>95.7% (1)</td>
</tr>
<tr>
<td>B</td>
<td>93.5% (5)</td>
<td>85.7% (5)</td>
<td>100.0% (1)</td>
<td>91.4% (3)</td>
</tr>
<tr>
<td>C</td>
<td>100.0% (1)</td>
<td>75.7% (7)</td>
<td>99.5% (2)</td>
<td>78.6% (7)</td>
</tr>
<tr>
<td>D</td>
<td>93.5% (5)</td>
<td>90.0% (4)</td>
<td>94.8% (7)</td>
<td>87.1% (4)</td>
</tr>
<tr>
<td>E</td>
<td>97.5% (2)</td>
<td>98.6% (1)</td>
<td>99.5% (2)</td>
<td>95.7% (1)</td>
</tr>
<tr>
<td>F</td>
<td>95.8% (3)</td>
<td>92.9% (2)</td>
<td>96.5% (5)</td>
<td>85.7% (5)</td>
</tr>
<tr>
<td>G</td>
<td>94.8% (4)</td>
<td>85.7% (5)</td>
<td>96.3% (6)</td>
<td>84.3% (6)</td>
</tr>
</tbody>
</table>

*Legend: Traditional Scrum; Targeted Scrum*
Another part of assessing the educational outcomes was assessing the final products of both
design. Of the seven teams, one team consistently produced the best software and another team
consistently produced the worst software. This suggests that the choice of Scrum methodology
did not impact students at the highest and lowest ends of academic proficiency. However, there
were changes within the other five teams. The remaining three teams that initially used
Traditional Scrum then transitioned to Targeted Scrum either improved their ranking or stayed
the same. The remaining two teams that initially used Targeted Scrum then transitioned to
Traditional Scrum both dropped in ranking.

At the end of every course at the University of Kansas, the students are given anonymous
surveys to evaluate both the course and the instructor. Of the 26 students in the course, 19 turned
in evaluations of EECS 690 to the Department of Electrical Engineering and Computer Science
at the University of Kansas. The students were presented with 11 statements that they were asked
to respond to with on a five-point scale from one (Strongly Disagree) to five (Strongly Agree).
The results, shown in Table 2, were very positive concerning the course and the instructor. These
student evaluations gave the authors confidence in the experiment as the students clearly
understood the material and assessed the course as beneficial to them.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Course Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classroom presentations are appropriately planned and organized.</td>
<td>4.84</td>
</tr>
<tr>
<td>Assignments (reading, papers, projects, exercises, etc.) are relevant and helpful in learning the subject.</td>
<td>4.79</td>
</tr>
<tr>
<td>I am acquiring the knowledge, skills, and abilities this class is intended to promote.</td>
<td>4.95</td>
</tr>
<tr>
<td>So far, this class has met my expectations for content coverage based on the course description in the catalog.</td>
<td>4.84</td>
</tr>
<tr>
<td>This course is a worthwhile academic experience.</td>
<td>4.84</td>
</tr>
<tr>
<td>The instructor effectively communicates the materials.</td>
<td>4.84</td>
</tr>
<tr>
<td>The instructor has established good rapport with the class.</td>
<td>4.95</td>
</tr>
<tr>
<td>The instructor provides meaningful and timely feedback to student.</td>
<td>4.95</td>
</tr>
<tr>
<td>Instructor has presented the course requirements and grading criteria.</td>
<td>5.00</td>
</tr>
<tr>
<td>Overall the instructor is an effective teacher.</td>
<td>4.95</td>
</tr>
</tbody>
</table>

*Scale: 1 = Strongly Disagree; 5 = Strongly Agree*
Research Outcomes

The course allowed the successful comparison of Traditional and Targeted Scrum and the collection of pertinent data. Based on the data from this course, the instructors found that Targeted Scrum did better in assisting the students in planning and prioritizing requirements for their software projects versus Traditional Scrum. However, the data showed that Targeted Scrum had only a negligible effect in helping the students to communicate requirements and progress versus Traditional Scrum.

In addition to comparing the two software development methodologies, the students assessed the artifacts of Targeted Scrum. The students assessed the end state and LOEs as helpful in visualizing progress and better defining the project's completeness. However, the students also assessed that the additional work to maintain these artifacts seemed redundant and increased the difficulty in maintaining coherency among all software development artifacts. The data and feedback captured during this semester long course will be used to adjust Targeted Scrum and plan for future research.

The instructors identified a couple of difficulties in conducting software development research in an academic venue. First, the students are only guaranteed to meet during class time. Ideally, agile software development should be practiced daily. However, this is not realistic in an academic setting. Second, the classroom was shared with other classes. This meant that students could not post, update, and leave charts and information on the boards. In agile software development, burn down charts and other artifacts would be left on whiteboards and walls for all to see and assess. When planning for research in an academic settings, these and other limitations must be taken into account.

Conclusion

The proposed, developed, and taught course was able to successfully balance the scholarly research interests with the academic interests of the students taking the course. This balancing effort required conscientious planning of desired educational outcomes, course schedule, project requirements, and research goals. Also, conducting research in an academic setting is risky because the students have to both enroll and to volunteer to participate in the research. Participation in research was not mandatory. However, balancing educational requirements and scholarly research interests was beneficial to both students and instructors. Not only did the students learn the desired educational outcomes, they also were exposed to the planning and conduct of scholarly research which they might use in future academic endeavors. Also, the instructors not only gathered data for research but they received valuable feedback and insight from students that will lead to future research opportunities.

Acknowledgements

This article reflects the views of the authors and does not reflect the official policies or views of the U.S. Army or United States Military Academy.
References


David P. Harvie

Dr. David P. Harvie is an Assistant Professor in the Department of Electrical Engineering and Computer Science at the United States Military Academy and a lieutenant colonel in the U.S. Army. His research interests include software engineering, network security, and data mining. He is a member of the IEEE and ACM. Dr. Harvie received his B.S. in Computer Science from the United States Military Academy, M.S. in Computer Science from North Carolina State University, and Ph.D. in Computer Science from the University of Kansas.

Arvin Agah

Dr. Arvin Agah is Associate Dean for Research and Graduate Programs in the School of Engineering and Professor of Electrical Engineering and Computer Science at the University of Kansas. He has edited two books and published more than 160 refereed articles in the areas of artificial intelligence and robotics. Dr. Agah has taught at the Department of Engineering Systems at the University of Tsukuba in Japan, the Center of Excellence in Remote Sensing Education and Research at Elizabeth City State University in North Carolina, and the Department of Mechanical Engineering at University of Canterbury in New Zealand.