Software Engineering Senior Design Course: Experiences with Agile Game Development in a Capstone Project

Tucker Smith
The Univ. of Texas at Dallas
800 West Campbell Road
Richardson, TX, USA
tss063000@utdallas.edu

Kendra M.L. Cooper
The Univ. of Texas at Dallas
800 West Campbell Road
Richardson, TX, USA
kcooper@utdallas.edu

C. Shaun Longstreet
The Univ. of Texas at Dallas
800 West Campbell Road
Richardson, TX, USA
shaun.longstreet@utdallas.edu

ABSTRACT

The importance of capstone senior design project courses is widely recognized for undergraduate software engineering curricula. They provide students with an opportunity to integrate and apply theoretical knowledge (both from previous courses and newly acquired for the project) on a team, improving both their technical and soft-skills. Here, we report our experiences using an agile development method for a game project; this is a radical shift from our previous course offerings that were based on waterfall, model driven development. This report is unique and valuable, especially for software engineering education, which goes beyond the discipline-specific limits of computer science curricula.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education, Computer Science Education; D.2.0 [**Software Engineering**]: General

General Terms

Design, Experimentation

Keywords

Senior Design, Capstone Project, Game Development, Agile Methods

1. INTRODUCTION

The Software Engineering Project (SE 4485) at the University of Texas at Dallas (UTD) is a required, semester-long capstone project that emphasizes teamwork and the practical implementation of Software Engineering (SE) principles. It is intended to both prepare students for real-world industry and to reinforce their ability to work together in a team-driven software development environment. Most projects are industry-oriented, with some actually being contracted by industry sources.

The topics of these projects come from a variety of sources including industry, professors, and the students themselves. At times, students work on a game as a project choice. Some results have been reported in the literature on using games in Computer Science capstone projects[1]; our prior experience using them in an SE capstone project at UTD has been mixed. Students are enthusiastic to work on such projects, yet often have difficulty completing the course with a functioning game. Whether this is due to over-ambition, lack of time management, or poor teamwork coordination, it has left these particular students disappointed in their own abilities, and deprived them of a complete view of the SE process.

Traditionally, SE 4485 follows a model-driven development methodology. Relying on heavy planning (project management, software quality assurance), with detailed requirements specifications, design documents, and significant modeling in CASE tools, before any implementation can take place. This primarily waterfall method had the advantage of being very straightforward and consistent with UTD's entry-level survey SE course (CS 3354).

We have found that a model-driven methodology may not be ideal for projects involving game design. Games typically require a knowledge of several, varied technical domains. This can include graphics, artificial intelligence, computer networks, sound, and many others. At face value, game design provides a rich educational opportunity; however, these benefits are not without drawbacks. Students must first become comfortable with the domains before they can even begin. Once students begin on the project, they will continue to build on this initial understanding incrementally, slowly making it deeper and more complete. Again, an excellent educational opportunity, but this changing understanding means that early designs will change rapidly, and continue to change through the project's evolution. This inability to create a valid up-front design, means that a waterfall process is not really an appropriate choice.

In this particular incarnation of the UTD SE Project, we implemented a variation of a well known agile method, Scrum. Differing from the waterfall method, Scrum paces work in discrete 2-week *sprints*. Each sprint encompasses all of the typical stages for a small set of software features (e.g. planning, design, implementation, testing). Instead of occasional, and time-consuming, progress reports, each sprint ends with a *demo* where the features completed in that sprint (known as the *sprint backlog*) are showcased. After each sprint, a functional (though incomplete) application is delivered, as opposed to a complete (and hopefully functional) application at the end of the project, as with waterfall. Every planned feature is kept on a list called the

[1]For examples see Section 4: Related Work

Final Forecast revolved around two players (on different machines) manipulating a real-time 3D terrain simulator. Beginning with a barren plot of rock, one player took on the role of “Life,” raising mountains and then eroding them to produce fertile valleys in which plants could thrive. The second player became “Death,” countering “Life’s” efforts with destructive floods, soil erosion, and drought. The game tracked the amount of vegetation covering the map, and the winning player was the one that controlled the most terrain by the end of the game. It was developed entirely in C++ with the OGRE graphics engine.

Sidebar 1: The Final Forecast Project

project backlog.

Students in the course broke into two teams of their own choosing. Each team then chose their own project to implement provided it met provided course constraints. Both decided to work on games. One team in particular chose to develop a multiplayer terrain simulation game called “Final Forecast” (see Sidebar 1).

By course conclusion, the Final Forecast group completed all but two of its original user stories. The remaining requirements were deemed inconsequential and too time-consuming for release. Final Forecast’s success is evident in its being a contender in the 2011 UTD ComputingFest Competition, a contest where UTD Engineering faculty nominate the most noteworthy of undergraduate projects. There, it will compete with projects created by undergraduate students across the College of Engineering. The other group’s project similarly completed on time.

In this paper, we describe how this course was implemented, and what effects the Agile process had in the use of gaming projects, with Final Forecast as an example. In Section 2 we lay out the course organization, tools used during the course, and the grading and assessment process. Our observations and a discussion of our experiences are presented in Section 3. A brief related work survey is in Section 4, followed by our conclusions in Section 5.

2. METHODOLOGY

2.1 Course Organization

The course was organized in three phases: project initiation, progression, and termination (Figure 1). Final Forecast is used as a running example in this section, additional details about the project are available at [9].

Project Initiation. Students first organized themselves into small teams based on personal relationships, compatible schedules and shared interest in a project idea. Next, they needed to establish member responsibilities, and to establish team communication channels.

Final Forecast was a team of four seniors. Group responsibilities included hosting and maintaining the source code repository, as well as physical maintenance of backlogs. Communication channels included email, phone, and a Redmine project management server.

Teams then had to develop a project proposal that met the course project constraints: that it be distributed, include either a GUI or database, and be implemented in an object-oriented programming language. These are consistent with typical industry projects, and reflect real-world skills that are often sought by employers. They also serve to establish a ‘baseline’ difficulty within which each project would, at the very least, have completed by semester end.

With this proposal, teams were required to develop a project backlog of at most 20 user stories, with each user story being a short description of a user’s goal in the application. Teams then selected a small subset of these stories to form the first sprint backlog. This backlog was encouraged to be small, as students would need extra time to become familiar with tools and the development process. Teams ended this phase by submitting their proposal and backlogs as assignment 1.

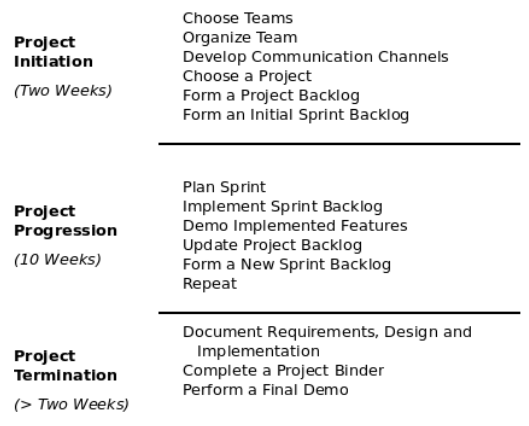
Project Progression. During this phase, teams would carry out development. Each sprint, teams would attempt to complete each feature on the sprint backlog. This would involve lightweight planning and design, coding, and testing. At sprint end, teams would complete another assignment, which would consist of demonstrating these features to the instructor, and updating the project backlog. They would then formulate a new sprint backlog, and prepare for the next sprint.

Project Termination. This phase began with teams documenting their projects. This would involve reverse-engineering architecture and detailed design in the form of UML class, component, and sequence diagrams from the completed application. These diagrams, along with requirements, test cases, code fragments, and screenshots would form a final project binder. This binder serves not only as an educational exercise in SE, but also as a portfolio piece for potential employers. The project was concluded with a final demonstration.

2.2 Development Tools

All teams were required to use a source-code management repository such as CVS, which is a common industry tool. Outside of this, any other tools or libraries were at the discretion of the teams themselves. To promote discovery skills and life-long learning, teams were expected to seek out and become proficient with tools that would advance their projects.

For instance, Final Forecast utilized the OGRE graphics engine, OIS (Open Input System), and the Boost C++ libraries (particularly Boost.asio). Having little or no experience with these libraries the team identified the need for them, installed, read documentation, attempted tuto-



Project Initiation *(Two Weeks)*	Choose Teams Organize Team Develop Communication Channels Choose a Project Form a Project Backlog Form an Initial Sprint Backlog
Project Progression *(10 Weeks)*	Plan Sprint Implement Sprint Backlog Demo Implemented Features Update Project Backlog Form a New Sprint Backlog Repeat
Project Termination *(> Two Weeks)*	Document Requirements, Design and Implementation Complete a Project Binder Perform a Final Demo

Figure 1: An overview of the course organization.

rial programs, selected features and then began to integrate them into their project. In doing so, the team exposed themselves to far more than the library features ultimately used in Final Forecast.

2.3 Course Assessment

Students were assessed through seven individual assignments worth 75% of the final grade. To provide a more encouraging environment for early failures, only the highest five grades of these assignments were calculated into final grades. The remaining 25% of the final grade derived from the final project demonstration and documentation.

Each individual assignment (excluding the first) involved a project demonstration. This involved checking out the application source from a CVS server, compiling it, and demonstrating each feature implemented during the sprint. Every step was observed by the instructor to eliminate the possibility of dishonesty. If not all features on the sprint backlog were completed, the development team needed to justify their progress. A student's failure to attend this demonstration without a university excused absence resulted in a grade of zero for that assignment.

Attendance of class was otherwise ungraded, though students were encouraged to use class time as an opportunity to meet, discuss progress, and plan sprints.

3. OBSERVATIONS AND DISCUSSION

3.1 Problems Encountered

A primary issue students encountered was creating and keeping a regular meeting schedule. Student responsibilities often extend beyond a single course project (e.g. other coursework, employment, family), leading to chaotic and incompatible schedules. Daily Scrums, and pair programming are both practices that are predicated on the assumption that the team will at least be working on the same day. From a teaching perspective, however, this challenge encouraged students to practice critical communication skills required in agile development.

The Final Forecast team attempted to mitigate this communication gap by having 'staggered' meetings, where the whole team participated in at least two meetings per week, but not necessarily with the whole team meeting simultaneously. They augmented this with an extended progress meeting of two hours before the weekly class meeting.

This communication handicap led to the exacerbation of another problem: inconsistent technical knowledge across the team. While one team member might be experienced in one domain or tool, others might not. This is a common problem for any project, and only a significant problem when communication suffers. For instance, when a technical problem popped up in Final Forecast, as long as four days could potentially pass before someone knowledgeable about the problem could meet with the team member in need.

Another issue that Final Forecast encountered at the end of the course was that reverse-engineering the models revealed design flaws. These flaws were found to be the cause of a significant performance loss, and had been introduced while adding new features onto a previous iteration of the networking subsystem. While discovering this did illustrate to the team that these models are important, it nonetheless highlights a drawback of this approach.

3.2 The Effects of Agile

In general, most undergraduate students have been exposed to an entire culture of gaming. The Final Forecast team gravitated to a game, not necessarily because of the challenge involved, but because they liked games. The creative control in a game typically far outweighs that of, say, a sensor network, or a ticketing system, where 'creative' features can quickly become detrimental. And it is through this creative process that students can participate in an active learning experience; once a student puts his or her own ideas into the system, they own a part of it and will continue to be a stakeholder, not just a passive bystander transcribing someone else's ideas.

We saw this with Final Forecast; students spent entire meetings discussing interesting or fun ideas to add into the game. Most were infeasible under the time constraints, but merely by expressing them, they would inspire new ideas in others. This feedback loop would be present in any environment with close teamwork, but is especially symbiotic when applied to games, with their creative extensibility.

When such close teamwork was possible, the turn-around time of problems that occurred during development was reduced. The issues that occurred with Final Forecast almost always involved build environment configurations, or simple misunderstandings of software libraries; none of which took very long to resolve once a team member knowledgeable on the subject was involved. Moreover, while working closely, team members tend to develop personal relationships that extend beyond the classroom. Half of the Final Forecast team was placed there, not out of preference, but simply randomly–they did not know anyone else in the class. Even after half of the team moved on to industry or graduate school, they still remain in regular contact. The ability to work cooperatively in a rapidly shifting team environment with people from different backgrounds and responsibilities on a successful project are highly desired skills for potential industry employers.

The greatest issue with this is, of course, that close teamwork is extremely difficult if team members do not have opportunities to physically meet. Normally, this course would meet once a week for over two hours, reflecting a model-based approach. One suggestion is to choose a Monday-Wednesday-Friday class schedule instead. This would help to ensure that students have individual class schedules that are amenable to meeting multiple times per week.

Though implementing an agile process in a classroom environment has its drawbacks, waterfall processes are not without their own issues. Students are not typically experienced software engineers; their knowledge is usually of a strictly theoretical nature. While they may know how to build an up-front model of a software system, they lack the experience to generate them with any quality. The resultant models are often flawed in deep but subtle ways that will only become apparent during implementation.

Fixing these flaws is a problem. The students, obviously, are oblivious to them. Professors cannot effectively review the models due to time constraints, and Teaching Assistants, if available, may not have the necessary skills to review multiple models. Inevitably, students would identify their mistakes through trial and error. In previous iterations of this course, planning and modeling took up approximately two months of the course, while the remaining two months were spent on implementation and testing. If students did not realize their mistakes during planning, there was little time to fix it, and any implementation they did towards this bad model would likely need to be thrown out.

To make matters worse, students would divide the labors of modeling and coding. Students who excelled in coding would do all implementation. Students who were better at documentation would do all planning. Beside from the implicit educational problem, this meant that, if communication was not perfect, the models and implementation would begin to diverge wildly. This invalidated the entire point of the model, and left the students with a flawed understanding of how SE works.

4. RELATED WORK

The integration of games and SE education has been considered from several perspectives: organizing the development of an application as a game for the class [2]; investigating e-learning games that simulate the software development process (e.g., [5, 6, 4]); or developing a game as course work (e.g., [1, 3, 7, 8, 10, 11]). When the games are developed by the students, four categories can be considered based on the kind of course they are used in. [10] and [1] use games to explore topics typically in introductory, breadth-oriented SE courses. [11] uses games in a specialized SE course on software architecture. [8] involves a non-capstone game development course that peripherally teaches SE topics. [3] and [7] use games in capstone game programming computer science courses. We discuss these latter two categories in more detail as they are the closest to our research. To the best of our knowledge, experience reports on a SE capstone project course that uses games are not available in the literature.

Of particular interest to this work is [8], which, while primarily focused on teaching game development, also includes a heavy emphasis on Scrum. Many of the student difficulties that they describe are consistent with our own experiences.

The courses in [8], [3] and [7] have much in common. They are organized as a mix of traditional lectures and a major project. For the project, the students form their own teams, choose their own game project, have a set of constraints they must follow (e.g., create working prototype, demonstrate and answer questions about their project), and are encouraged to share their results with the community in a display or a competition. The team formation in [7] is distinct, as groups are composed of two computer science students and one artist. This interdisciplinary team formation is possible as the Computer Science capstone course is run in co-operation with a game course in the School of Visual Arts.

[3] and [7] are described as computer science courses, not SE; there is no discussion of requirements, testing artifacts, or the development process used by the students for the project. A game design is included in the course description in [3], but is not characterized in the short paper.

5. CONCLUSIONS

This paper has described the application of a Scrum-like agile process to an undergraduate SE capstone course, and the effects that such a process had on game development as a project. This approach attempted to address issues that had been observed when using a waterfall, model-based process, and results suggest that agile, with its emphasis on teamwork and fast releases, is more effective in aiding students to produce functioning games. In this single experience, we found that an agile method encourages high-level critical thinking skills for SE students in an engaged and highly motivated environment. It also provided our students with opportunities to practice skills that are highly desirable to industry employers; skills that are not typically promoted in traditional approaches to teaching SE development.

6. REFERENCES

[1] K. Claypool and M. Claypool, *Teaching software engineering through game design*, 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education, Caparica, Portugal, June 27-29, 2005, pp. 123–127.

[2] J.J. Horning and D.B. Wortman, *Software hut: a computer program engineering project in the form of a game*, IEEE Transactions on Software Engineering, July 1977, vol.SE-3, no.4, pp. 325–30

[3] R. M. Jones, *Design and implementation of computer games: a capstone course for undergraduate computer science education*, SIGCSE Bulletin, March 2000, vol.32, no.1, pp. 260–4.

[4] Emily Oh Navarro, and André van der Hoek, *SimSE:an educational simulation game for teaching the Software engineering process*, Annual Joint Conference Integrating Technology into Computer Science Education: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education; 28-30 June 2004.

[5] C. Shaun Longstreet and Kendra M.L. Cooper, *Using Games in Software Engineering Education to Increase Student Success & Retention*, 2011 Conference on Software Engineering Education and Training, 2011 (to appear).

[6] J. Ludewig, T. Bassler, M. Deininger, K. Schneider, and J. Schwille, *SESAM-simulating software projects*, Fourth International Conference on Software Engineering and Knowledge Engineering, 1992, pp.608–615.

[7] I. Parberry, T. Roclen, M.B. Kazemzadeh, *Experience with an industry-driven capstone course on game programming*, SIGCSE Bulletin, March 2005, vol.37, no.1, pp. 91–5.

[8] Jonas Schild, Robert Walter, and Maic Masuch. *ABC-Sprints: adapting Scrum to academic game development courses*, FDG '10: Proceedings of the Fifth International Conference on the Foundations of Digital Games; 187-194 2010.

[9] T. Smith, D. Walters, J. Savant, and T. Langford. "Final Forecast Project Site." Internet: `http://utdallas.edu/~tss063000/finalforecast/`, 2011.

[10] E. Sweedyk and R.M. Keller, *Fun and games: a new software engineering course*, ACM SIGCSE Bulletin, September 2005, 37(3), 138–142.

[11] A. I. Wang and B. Wu, *An Application of a Game Development Framework in Higher Education*, International Journal of Computer Games Technology, Special Issue on Game Technology for Training and Education, Volume 2009.