A CASE STUDY: INTEGRATING A GAME APPLICATION-DRIVEN APPROACH AND SOCIAL COLLABORATIONS INTO SOFTWARE ENGINEERING EDUCATION

Weifeng Xu and Stephen Frezza
Department of Computer and Information Science, Gannon University, 109 University Square, Erie, PA, U.S.A.

Keywords: e-Learning, Software Engineering, Undergraduate Education, Game Development, Social Collaboration, Curriculum Design, Case Study.

Abstract: Teaching software engineering to undergraduate students is a challenge task. Students are expected to understand both technical and social aspects of software engineering. This paper presents a complete case study of a hybrid approach that systematically combines a game application-driven approach and social collaborations into the software engineering curriculum at the undergraduate level. The case study consists of 1) proposing a new curriculum design process, 2) identifying a set of software engineering principles, practices, and online collaborative learning tools by following the design process, 3) proposing a semester-long game project, 4) integrating the principles, practices, and the collaborative learning tools into the game development process and 5) delivering the principles, practices, and tools to students during the game devolvement. The results of the case study, including analysis of the related project documentation and students' feedback indicate that adopting the games app-driven approach motivate students to learn in teams, help transferring knowledge effectively between instructors and students and facilitate achieving the student learning objectives.

1 INTRODUCTION

Teaching software engineering through game application (app) development has become more popular in recent years. The key benefit of introducing game development into software engineering curricula is to motivate students.

Claypool (2005, p. 123) presents initial work towards the goal of more effective software engineering education, describing the implementation of a game-centric software engineering course. The focus was on modules that allow for a hands-on practice of software engineering theory, where the sum of the modules culminates in a working computer game that clearly illustrates successful software engineering practices and provides students with particular satisfaction of a complete and useful development effort. An alternative to Claypool's approach is one that divides the software engineering into learning modules, Cagiltay (2007, p. 405) emphasizes the outcomes of the courses, *e.g.*, focuses on improving students' abilities in four areas: problem solving, the application of previously learned knowledge, the use of independent learning and learning by doing.

Some educational researchers focus on teaching specific topics within software engineering. For example, Paul and Fu-Shing (2008, p. 1) (2007, p. 237) present an approach for teaching design patterns that emphasizes object-orientation and pattern integration. They present a case study centered on *EEClone*, an arcade-style computer game implemented in Java. Students analyzed various design patterns within *EEClone* and learned how to apply design patterns in their own game software. Other researchers use games to teach languages and project management through game development (Joe and Amber, 2008, p. 250) (Yan, 2009, p. 969) (Wolz and Carey, 2007, p. 322).

Although a game-based approach for teaching software engineering is appealing, several main problems pertain to designing and delivering a course inclusive of app-driven development (Ragan, Frezza, & Cannell, 2009, pp. T1A-3):

- Social Aspects: Developing software systems is a complex socio-technical activity. The

Xu W. and Frezza S..
A CASE STUDY: INTEGRATING A GAME APPLICATION-DRIVEN APPROACH AND SOCIAL COLLABORATIONS INTO SOFTWARE ENGINEERING EDUCATION.
DOI: 10.5220/0003445300230032
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (ICEIS-2011), pages 23-32
ISBN: 978-989-8425-56-0

social interactions and collaborations among each team member can enrich a learning process. More specifically, we need to understand whether we are able to integrate online collaboration tools in software engineering education, and how these tools help students understand software engineering process.

- Pedagogical Aspects: Need for a systematical way to help faculty members to design a game-based software engineering curriculum. *i.e.*, How to assure that the appropriate computing and engineering learning objectives are covered and achieved in the game development? How to derive and implement the course content based on software engineering disciplines? How to evaluate students' performance in terms of these disciplines?
- Case Study: The educational research to date lacks a complete case study, including an executable game and online documentations, for illustrating a game-based approach, how it addressed desired software engineering principles, good practices, as well as how collaboration tools were integrated into the course(s) and curriculum.

The objectives of the research are to: 1) systematically propose a hybrid approach which combines a game app-driven process and social collaborations for designing a software engineering course curriculum for undergraduate students; 2) explain each element of design process in details via a case study, particularly how the course content is derived from student learning objectives; 3) demonstrate how the key concepts, principles, the best practices of software engineering, and collaborative tools are covered and applied during the game development, and 4) publish the key artifacts of the case study online as references for other faculty members.

The rest of the paper is organized as follows: Section 2 describes the game app-driven curriculum design process; Section 3 demonstrates a complete case study using game app-driven process and these collaborative tools and Section 4 concludes the paper and discusses some possible future work.

2 HYBIRD CURRICULUM DESIGN PROCESS

The curriculum design is an important quality aspect of the course content delivery. For comparison purpose, this section describes the difference between the traditional and the hybrid curriculum design processes. The hybrid approach is the integration of a game-app driven and the social collaborative approach.

2.1 Traditional Curriculum Design

Traditionally curriculum design has focused on the transferring of discrete pieces of information, *i.e.*, facts, disciplines, and formulas, from instructors to students (Dick and Carey, 2000). Because the information is considered important in its own right, the traditional curriculum designers often build the outcomes-based curriculum based on these knowledge segments. More specifically, in a traditional design process, instructors outline the objectives of the course, derive the expected program outcomes through the student learning objectives of the course and check if these objectives are achieved by comparing the actual outcomes to the expected outcomes. Figure 1 shows the overall traditional course design process.

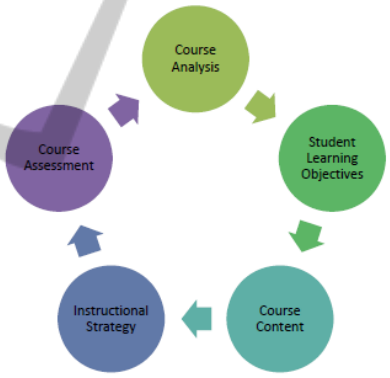

Figure 1: Traditional curriculum design process.

For example, the process for designing software engineering curriculum is comprised of the following five elements:

- Course analysis: Studying course materials, analyzing and understanding the characteristics of software engineering, and identifying potential audiences.
- Student learning objectives: Identifying key concept of software engineering and outlining the learning objectives of the course, *i.e.*, defining the accomplishment students are able to achieve by the end of the course.
- Course content: Developing the course materials that will be delivered to students and assemble the course. It requires determining the course content suitable for

undergraduate students, implementing and assembling the content.

- Instructional strategy: Determining the best approach that an instructor may take to achieve learning objectives. Instructors can create learning environments and specify the nature of the activity in which some key concepts can be implemented.
- Course assessment: Evaluating if these learning objectives, i.e., the outcomes of the teaching and learning, have been met and providing feedback to students. The course assessment results also help the course analysis.

2.2 Game App-Driven Course Design

The game app-driven approach for designing software engineering curriculum is a different way of organizing the elements in the traditional course design approach.

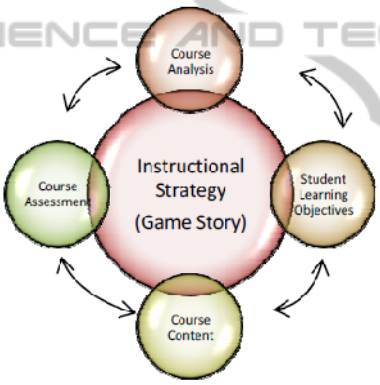

Figure 2: Game app-driven approach curriculum design process.

As shown in Figure 2, the instructional strategy is located in the center of the design cycle, *i.e.*, the game is a platform and driving force for knowledge transmitting. The rest of four elements are driven by the game development and also have impacts on how game should be designed.

The game app-driven approach is not a passive knowledge transmission. The outcomes of the course are reflected in the results of the interactions between instructional strategy and these elements, whereas the adapting of games is the key realization of the instructional design. For example, one of the objectives of the course is to let students demonstrate the understanding of the iterative development process. The objective requires: 1) the game should be feasible to be designed, implemented, and delivered in multiple iterations; 2) instructors should design several project milestones and deliveries so that students can experience incremental development.

Designing a game story that fits the educational context of software engineering is the main concern for game app-driven approach. Instructors need to answer questions similar to those listed in Table 1 in terms of course analysis, student learning objectives, course content, and course assessment. For instance, before adopting the game-based approach, both instructors and students should be technically ready for the approach. The questions arise as: 1) Are instructors confident in teaching game development in software engineering? 2) What are the backgrounds of the students who take the course? 3) Are those students ready for developing games? Failure to answer these questions may fail adopting the approach.

2.3 Social Collaborations

Developing software systems is a complex socio-technical activity. The game app-driven approach covers the technical aspect of software engineering. However, successfully developing a software system needs the collaborative work, *e.g.*, idea discussion and knowledge sharing, in a development team. Building social collaborations into software engineering complements the game app-driven approach as the social collaboration plays a vital role in predicting the success of software system (Meneely, Williams, Snipes, and Osbourne, 2008, p. 13).

In current undergraduate education, faculty members propose team-based projects mainly for helping students to comprehend software engineering principles and experience good software engineering practice. The social aspect of software engineering is often overlooked.

Social collaboration crosscuts all elements of curriculum design process. For example, in course analysis, the social aspect of the software engineering needs to be identified, and thus understanding the social factors has impacts on software projects is considered as one of the student learning objectives. Naturally, project managements and collaboration tools will be part of the course content due to the team-based project. Finally, the team-based collaboration and performance should be evaluated as a part of the course assessment.

In the new proposed hybrid approach, we focus on the team-based learning process rather than the projects itself. We assume students will achieve the desirable learning objectives and build quality software system if they follow the well-designed learning process. The team-based learning approach has become an increasing valuable methodology as

Table 1: Questions for designing games.

Elements	Questions
Course Analysis	What is the nature of software engineering?
	Does the game development approach reflect the concept of the computer science and engineering?
	Are instructors confident in teaching game development in software engineering?
	What are the backgrounds of the students who take the course? Are those students ready for developing games?
	What are the constraints to use game app-driven approach?
Student learning objectives	Any principles and concepts can be covered in the game development, such as software process, software development lifecycles, the phases and activities of a lifecycle, and the artifacts (documents and code) created in each phase of a lifecycle?
	Any management issues can be covered in the game development, including issues of project planning tools and diagram, team organization, human factor, and risk analysis?
Course Content	How to map student learning objectives to the course content? *i.e.*, How to implement the student learning objectives in the game?
	Are course content consistent in the game development process?
	Any constraints we have to meet in the game development process? For instance, time constraints, resource constraints, and technology constraints.
Course Assessment	How to assess the objectives of the course?
	What are the outcomes of the course?
	How to evaluate students' performance? If students work in the team, how to evaluate each individual?

social interactions of the team can enrich a learning process. In team-based learning, students work in groups on problem-based projects. More specifically, a group of students take a problem from instructors, apply it to a real life situation with these projects, and present the solutions. These students will be collaboratively engaged in analysis, design, problem solving, decision making, and investigative activities to accomplish the project

We are interested in identifying and adopting some free online collaboration methods and tools for the software engineering course. One of the approaches is to use existing online social networks, such as email, texting, discussion boards, Wiki, Facebook, etc. These existing social services can be integrated into the game development activities to facilitate the development processes. In addition, we try to find some online management tools to manage software development processes, intermediate products, and documentations.

3 *GANNONOPOLY*: A CASE STUDY

In this section, a game story, named Gannonopoly, is proposed to serve as the core of the hybrid approach. The social aspect of software engineering, mainly includes project management and team collaboration, crosscuts the whole game development process. Each question listed in Table 1 and issue mentioned in section 2.3 will be addressed in the case study. Those answers are organized in the corresponding subsections.

3.1 Game Story

For the game app-driven approach, the game story serves as the platform and the driving force to achieve the desirable learning objectives. Thus it is essential that a suitable game story is developed so that instructors can create learning environments and specify the nature of the development activities in which the key concepts and learning objectives of the course can be implemented. (Ragan, Frezza, & Cannell, 2009, pp. T1A-2)

A suitable story can be found by combining the landmarks of the University with the very successful Monopoly™ game story. This has the advantage of familiarizing players with the landmarks located on and surrounding our campus. Anyone that is interested in or associated with Gannon University, including prospective, current, and former students, faculty, and staff will enjoy playing our game. Thus *Gannonopoly* became the Gannon University electronic version of the classic board game of Monopoly™. The objective of *Gannonopoly* is to become the wealthiest player through the buying, renting, and selling of various properties. The game can be played on any computer with two to eight human players.

The *Gannonopoly* has a game 'board' in which Gannon properties and other spaces are depicted. These spaces include jail, go to jail, student parking, tuition raise, etc. For example, players begin on the space marked "Go" and are awarded a $200 salary

each time that they pass over this space. Chance and community chest cards present a random movement or monetary action for the current player. The "Go to Jail" space sends a player directly to a jail. A player can get out of the jail by rolling doubles on any of their next three turns or by paying a fine of $50.

The game also contains two animated dice that determine the distance to advance the current player's token. When a given token lands on a property that is able to be purchased, a screen appears asking the player if he or she wants to purchase the property. If the property cannot be purchased, it must be rented from the player that owns it.

The game ends when one player becomes bankrupt, or runs completely out of money. You are declared bankrupt if you owe more than you can pay either to another player or to the bank on your current turn. At the end of the game, each player's accumulated money and property (valued at 50% of their purchase price). These two amounts are then summed for each player to determine the winner. The richest player wins!

3.2 Course Analysis

The nature of software engineering is twofold: software engineering as a computing discipline and software engineering as an engineering discipline. The nature means and aims of software engineering are summarized in Table 2 (Wang, and Patel, 2000, p. 10). Table 2 indicates 1) As both an engineering and as a computing discipline, students need to understand life cycle methods, such as specification, design, implementation, and evolution; and 2) As an engineering discipline, students additionally need to understand standards (*e.g.*, designing and coding standards), engineering tools, etc. The nature of software engineering requires students master necessary documenting and programming skill for developing large-scale software. In the proposed game-app driven approach, these principles, life cycle methods, and engineering approaches need to incorporate into the development of *Gannonopoly*.

There are other issues need to be addressed to apply the game-app driven approach to the course curriculum design. For example, what is the experience of both instructors and students with game development technology? Some issues need to be addressed before implementing the course:

- *Are instructors ready to learn designing games?* Some preliminary studies show the compelling results of teaching game in software engineering (Baker, Navarro, and Hoek, 2005, p. 3), we eager to learn game developing skills and to know how to systematically incorporate the game

Table 2: Nature of software engineering.

Nature	Means	Aims
A computing discipline	Life cycle methods: - specification - design - implementation - evolution	Document and Programming
An engineering discipline	Engineering approaches: - standards - methodologies - tools - processes - organizational methods - management methods - quality assurance systems	Large-scale software

systematically incorporate the game developing into software engineering course.

- *Do instructors plan to use a game engine to develop the Gannonopoly?* Students do not have to use a game engine to develop the game. They may choose to develop the game using other objective-oriented languages, such as Java, and C++. However, developing the game from scratch is not an ideal approach to introduce the computing and engineering principles to students as software engineering is not only about programming. Some game engines are available, including PyGame™ for Python, XNA™ for .NET, JGame™ for Java, Gamemaker™ for C++, etc. These game engines can provide instant results for demonstration purpose.
- *Which game engine fits to the course and the Gannonopoly game?* As junior undergraduate students don't have much real-world programming experience and almost never systematically done any semester-long project before, we choose Gamemaker™ as the game engine for developing the *Gannonopoly*. The key features of using Gamemaker™ include easy to learn drag-and-drop actions, built-in C++ style programming language for advanced developers, good tutorial, large community supports, etc.
- *Is the scope deliverable in one semester?* We expect that the game, or partial game, will be developed by the end of the semester. It is mainly determined by the problem domain and scope of the *Gannonopoly*. As the game is a Gannon university version of monopoly game, the problem domain of the project is relatively easy to understand. Once the problem domain is understood, the scope of the game can be easily determined during the requirements.
- *How are existing online social networks able to enrich students learning process*? For choosing appropriate collaborative tools for students, the anticipated results need to be defined or estimated.

Table 3: Software Engineering Student learning objectives.

Levels	Student learning objectives
Knowledge	List software engineering life cycle methods and engineering approaches for software engineering.
	Identify basic steps and good practice for specifying requirements, designing systems, and implementing systems.
	Outline software engineering methodologies, processes, management methods, and methods for system quality assurance.
Comprehension	Illustrate the activities involved in software requirement engineering, software development, testing and evolution.
	Differentiate different software process models, software engineering methodologies, management methods, and propose of quality assurance plan.
Application	Demonstrate choosing appropriate develop methodologies, tools, process models, and other engineering disciplines for developing different type of projects.
	Apply UML to specify software requirements and design for small problems.
Analysis and Synthesis	Propose a game project plan with other team members, including a game story, a team organization, and a project schedule.
	Research the feasibility of developing the game.
	Develop software requirement, design documents, and other related artifacts for the proposed game using software engineering approaches mentioned above.
	Create and demonstrate the game in the class.
Evaluation	Evaluate the quality of the game.
	Recommend future improvements.
Social interactions and collaborations	Understand the social factors have impacts on software engineering.
	Understand the social interactions and collaborations tools can be used for facilitating the software engineering process and improving the productivities.
	Be able to choose appropriate online collaboration tools for each activity in the development process.
	Be able to demonstrate how to use the online collaboration tools.

3.3 Student Learning Objectives

Student learning objectives can be derived from the software engineering means and aims listed in Table 2 and the social aspect of software engineering. We use Bloom's Taxonomy (Bloom, Engehart, Furst, and Krathwohl, 1956) to write student learning objectives.

Bloom's Taxonomy is a classification of learning objectives within education. There are six levels in the taxonomy, moving through the lowest order processes to the highest: knowledge, comprehension, application, analysis, synthesis, and evaluation. By applying verb wheel based on Bloom's taxonomy, we derive the student learning objectives and list in Table 3.

The software engineering student learning objectives indicate that intellectual tasks at the knowledge, comprehension, and application level are generally considered less cognitively demanding than tasks at the levels of analysis, synthesis, and evaluation. The latter three are considered as the higher levels of Bloom's taxonomy, and the first three are the foundation of higher level learning. To reach these levels, students need to be equipped with high level critical thinking, creative thinking, and problem solving skills. In the case study, only the higher levels of Bloom's taxonomy are involved the game development.

3.4 Course Content

The course content is organized to support the achievement of the student learning objectives. We use *Gannonopoly* as the running case to demonstrate the course implementation. Due to time and resource constraints, only main disciplines will be covered during the development of the game. These disciplines include requirements specification, design, implementation, software quality assurance, collaboration tools, software processes, and management methods.

3.4.1 Requirements

Requirements elicitation and requirements specification are two critical tasks in the requirements engineering. While developing *Gannonopoly*, students should be able to practice the techniques to capture and document requirements. More specifically, during the requirement phase, students are instructed to write a game story, define problems, organize a development team, estimate the resource, make up a reasonable schedule, adopt a

use case approach to derive functional features, and document the results in IEEE format. As the results, the functions of the system are described in Table 4. The complete artifacts of *Gannonopoly* can be accessed at http://code.google.com/p/gannonopoly/.

Table 4: Functional features of *Gannonopoly*.

	Functional Features
1	An interactive game board triggering different movements shall be the main basis for the entire game.
2	The system shall simulate the rolling of two dice through animation.
3	Multiple players shall be able to play the game.
4	A "scoring" mechanism shall be used for keeping track of the money held by each player.
5	A deed management mechanism should be used for keeping track of properties owned by each player.
6	The system shall save, exit, and reload the game from its current position.

It is worth mentioning that although the use case approach is an effective requirement elicitation technique for many types of projects, students may experience difficulties to specify the behaviors of games that heavily reply on detecting and response to the external events of the system. Event-response tables are a convenient way to collect these event and response information (Wang and Patel, 2000, p. 1). The event-response tables can be organized based on use cases. For example, the following event-response table describes all of the possible events that are associated with the player setup screen (*i.e.*, functional feature 3). The screen allows players to choose the number of players and their desired player tokens. The definition of tokens is defined in the glossary as different markers that players can choose from to represent them during the game. Note that due to the space limit, only partial of the events is shown in the table.

Table 5: Event-response table for player setup.

Event	System State	Response
User clicks "New Game" button	Welcome screen is displayed	1.Player setup screen is displayed
User clicks "+" button	Player setup screen is displayed and number of players is currently less than 8	1. Number of players is increased by 1 2. Number of tokens is increased by 1
User clicks "+" button	Player setup screen is displayed and number of players is currently at 8	1. Nothing happens
…	…	…
User clicks on token	Player setup screen is displayed and the icon is unlocked	1. Nothing happens

3.4.2 Design and Implementatioin

During the game design, students are expected to answer following questions:

- Is there a generic application architecture that can be used as a template for *Gannonopoly*?
- How to decompose the system into to sub-systems or components?
- How to organize these sub-systems or components?
- What architectural style or model is appropriate for *Gannonopoly*?
- How should the architecture design be documented?

After searching for the solutions, students figure out an event-process system architecture design is a good fit for designing the game. The system architecture diagram of the *Gannonopoly* is shown in Figure 3. It describes the main components of the game system and how they interact with one another. The figure shows that *Gannonopoly* consists of three major system components. The property management system that allows the user to choose to buy, rent, and view their properties, the banking money system keeps tracking of the amount of money possessed by each player, and the player positioning system allows the player token to move, land, pass over, and view the information for the current space.

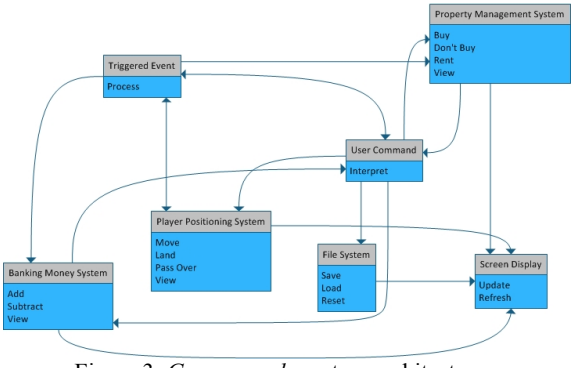

Figure 3: *Gannonopoly* system architecture.

Note that other components are also shown in the system architecture diagram, such as the triggered event, user command, file system, and screen display. However, these are not exactly implemented distinctly in the code since they are supported by the Gamemaker™ engine. As part of the design process, students should be able to identify and use those reusable modules.

During the implementation, some software engineering principles should be covered as well, including:

- Object-orientation: Gamemaker™ requires creating instances before defining any events and action. The events and action are corresponding to the events and response in event-response tables. Students are expected to understand the traceability between the tables and implementation in Gamemaker™.
- Standards: Coding standard needs to be applied for consistent coding style.
- Reusability: Code reusability is an important issue during implementation. Students learn how to find reusable library online and realize reusable code save their time and improve the system quality.

3.4.3 Iterative Development, Management and Colloration Tools

Iterative development is so fundamental that we intend to adopt the interactive approach to develop Gannonopoly. The game specification, design, and implementation are broken down into a series of increments that are each developed in turn. The main benefit of using iterative development is to avoid project failure by monitoring the project progress constantly. At least three iterations need to be delivered, e.g., before the midterm, after the spring break, and the final version. We label these iterations as iteration 1, 2, and 3, respectively. For the demonstration purpose, only three iterations of the main menu and the game board are listed from Figure 4 to Figure 9.

Figure 4: Iteration 1 main menu.

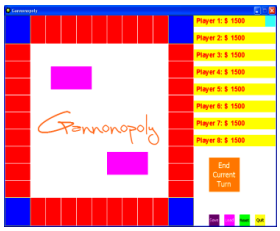

Figure 5: Iteration 1game board.

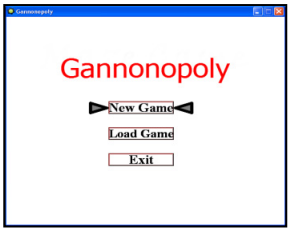

Figure 6: Iteration 2 main menu.

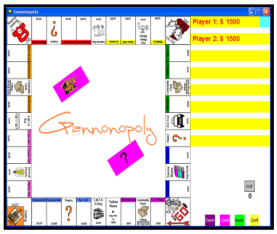

Figure 7: Iteration 2 game board.

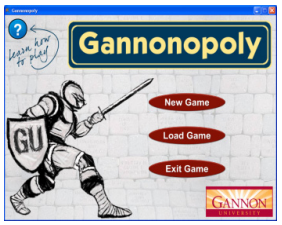

Figure 8: Iteration 3 main menu.

Figure 9: Iteration 3 game board.

The iterative development relies on good project managements and online collaboration tools. One of the teaching goals of software engineering is to allow students to be aware of and experience various project management issues, including team management, process management, product management, configuration management, etc. Furthermore, we expect students to know how to utilize management and collaboration tools to smooth the development process, increase the productivity, and improve the software quality. These tools used in the development of *Gannonopoly* are described in Table 6.

Table 6: Management tools used in *Gannonopoly*.

Name	Management	Purpose
Google Calendar	Team	Scheduling meetings, iteration deadlines, other project-related events
Google Wave and Google Talk	Team	Real-time communication and collaboration between participating team members
Google Code	Product	Hosting project, including file downloads
Issue tracking	Process, product	Tracking issues, such as requirements gathering, designing, implementation, bugs, etc.
Wiki	Product, process	Wiki documentation, including requirements, design, testing documentation, and source
TortoiseSVN	Configuration	Version and iteration management

During the development, students are advised to use Google Project Hosting (http://code.google.com/hosting/) to manage their projects. Project Hosting on Google Code provides a free collaborative development environment for open source projects, which is an integrated tool of member management,

version control, wiki, code repository, and issue management. Note that although other professional management tools are available in the department, such as JIRA (http://www.atlassian.com/software/jira/), we use Google Project Hosting mainly because it is a free and light-weighted tool.

3.4.4 Course Assessment

Course assessment for the game app-driven approach is threefold: 1) evaluating the team performance by examining if all the course content has been experienced by the team; 2) evaluating individual's performance by peer-evaluation; and 3) studying students' feedback.

Software engineering is a social engineering, thus we expect students to work as a team interactively and collaboratively. In the game app-driven approach, *Gannonopoly* is a platform for students experimenting all software engineering principles and good practices. Therefore, only one grade will be assigned to the each artifact of *Gannonopoly* developed by the team. These artifacts include all documentations and iterations developed based on computational and engineering disciplines.

The individual's performance is determined by each student's personal report. The items need to be included in the report are listed in Table 7.

The essential of the project report is a reinforcement of the computing and engineering principles in the course content. The case study shows that the quality of the personal reports well reflects each individual's performance in the team. The list of sample personal reports can be found http://perceval.gannon.edu/xu001/teaching/2010spring/cis315/project/stu_report/. The peer evaluation mainly reflects each team member's contribution and collaboration in the semester-long project.

Table 7: Student personal report items.

Items	Descriptions
Introduction	The background of the game, including motivation and the game story
Requirement	The features of the games
Design	The architecture of the game
Project management	The issues related to project management, including tools students have used
Results	Screenshots
Discussion	What you have learned? What would you like to change if you restart the project
Peer evaluation	Team member/self evaluation

The study of students' feedback is another way to assess if the objectives of the software engineering have been archived. Our online evaluation system (Mak & Frezza, 2006, pp. M5G-14) supports anonymous collection of faculty and student feedback based on the student learning objectives. The collected information from this iteration of the course will be used for analysis and improvement of the next course offering.

4 CONCLUSIONS AND FUTURE WORK

This paper presents a case study to systematically demonstrate a hybrid approach combining game app-driven approach and social collaborations into undergraduate software engineering education. In the approach, a new curriculum design process is proposed, in which a game is the center element of the process. The social interactions and collaborations crosscut all activities of the software development processes.

In the case study, the *Gannonopoly* game project is identified as a platform and driven force to help students understand the key software engineering principles and good practices. The key principles are derived from both computing and engineering disciplines, and will be implemented in course content based on the new curriculum design process. The goal of the game app-driven approach is to deliver the course content through the game development in a systematic way.

Essentially, the game app-driven approach for software engineering curriculum design is comprised of three components: 1) a component for systematically identifying key software engineering concept by instructors, 2) a project-based learning method for students, and 3) a "fun factor" component to motivate students. The social concern complements the game app-driven approach as social interactions and collaborations are the important factors to predicate the success of software projects. Several online collaborations tools are chosen to enforce the concept of the social interactions and to facilitate the development process.

We compare the new approach (implemented in Spring 2010) to the traditional way of teaching software engineering (Spring 2009). Two teams were formed in each semester. Each team consisted of 3-4 students. The results of the case study are encouraging. We list some observations here:

- Student involvements. Due to the "fun factor" of the game, students have spent average 50% more time on the project. On the other hand, peer pressures motivate other

students to work hard to achieve the final goal.

- Software process. All the development processes are documented online. It is good indication that student follows software development process.
- Product qualities. The final product, including an executable game, all online documentations, and user manual, indicates students understand the key principles of computing and engineering and are capable of adopting these principles and practices to develop large-scale software products. These are the ultimate goals of the software engineering course.
- Faculty interactions. Although we focus on the interactions and collaborations between students, the online collaboration tools also facilitate faculty members to provide consistent feedbacks during the development. For example, faculty members can make comments on each wiki page.

The future work is related to our observations in the class. Some interesting questions have arisen during the teaching: 1) how students pick up their team numbers? *i.e.*, are grades or performance the factors when students pick up their team number? As we allow students to form their own team, it seems students more comfortable work with someone they already known, regardless performance in other classes. Does it have any impacts on leaning objectives? 2) How to measure the results of collaborative leaning? Students heavily rely on Wiki for documenting requirements, design, and testing plan. It can be seen as a collaborative leaning process for undergraduate students. Is there any way to measure the effectiveness of collaborative leaning and improve the collaborative leaning process? 3) How Google wave can be used in software engineering education? Students have tried to use Google wave to communicate and collaboratively develop documents. More study can be done to evaluate the impact of use the new technology.

REFERENCES

Baker, A., Navarro, E, and Hoek, A., 2005, 'An Experimental Card Game for Teaching Software Engineering Processes', *Journal of Systems and Software special issue*, 75:1-2, pp. 3–16, 2005.

Beane, J., 1991, 'The Middle School: The Natural Home of the Integrated Curriculum', *Educational Leadership 49*(2), pp. 9-13.

Bloom, B, Englehart, M., Furst, E, Hill, W, and Krathwohl, D., 1956, 'The Taxonomy of Educational Objectives, the Classification of Educational Goals', Handbook I: Cognitive Domain.

Cagilta, E, Nergiz, 2007, 'Teaching Software Engineering by Means of Computer-Game Development: Challenges and Opportunities', *British Journal of Educational Technology,* Vol.38, Issue 3, pp, 405-415.

Claypool, L., and Claypool, M., 2005, 'Teaching software, engineering through game design', In Proceedings of the *10th Annual SIGCSE Conference on Innovation and Technology, in Computer Science Education*, Monte de Caparica, Portugal, pp. 123-127.

Dick, W., and Carey, J., 2000, 'The Systematic Design of Instruction', New York: Addison-Wesley.

Halla, K, and Moirao, D., 2000, 'A Guide to Writing Better Objectives for the Cognitive Domain', http://cstep.csumb.edu/Obj_tutorial/ Accessed May 20, 2010.

Joe, L., Amber, S., 2008, 'Teaching game programming using XNA', In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, Madrid, Spain, pp. 250-254.

Mak, F. and Frezza, S., 2006, 'Collection, Maintenance, and Validation of a Set of Effective Objective Evidence,' *in Proceedings of the International Conference on Engineering Education 2006* (ICEE), San Juan, PR, pp. M5G-12 – M5G-17.

Meneely, A., Williams, L., Snipes, W., and Osbourne, J., 2008, 'Predicting Failures with Developer Networks and Social Network Analysis', In proceeding of the *16th ACM SIGSOFT Foundations of Software Engineering (FSE*), Atlanta, GA, pp. 13-23.

Paul, G., 2007, 'Computer games as motivation for design patterns', In Proceedings of the *38th SIGCSE technical symposium on Computer science education,* Covington, Kentucky, USA, pp. 223 - 237 .

Paul, G., and Fu-Shing, S., 2008, 'Teaching Design Patterns through Computer Game Development', *Journal on Educational Resources in Computing (JERIC)*, vol. 8 n.1, p.1-22, 2008.

Ragan, E., Frezza, S., and, Cannell, J., 2009, 'Product-Based Learning in Software Engineering Education', 40th ASEE/IEEE Frontiers in Education Conference, San Antonio, Texas, USA, pp. T1A-1-T1A-6.

Wang, Y. and Patel, D, 2000, 'Comparative Software Engineering: Review and Perspectives', Annals of Software Engineering, Springer, vol.10, pp. 1-10.

Wiegers, K., 2003, 'See You in Court', Software Development 11(1), pp. 36-40.

Wolz, U., Pulimood, S., 2007, 'An integrated approach to project management through classic CS III and video game development', In Proceedings of the *38th SIGCSE technical symposium on Computer science education*, Kentucky, USA, pp. 322-326.

Yan, L., 2009, 'Teaching Object-Oriented Programming with Games', In Processings of the *sixth International Conference on Information Technology: New Generations*, Las Vegas, NV, pp. 969-974.