2006-1209: TEACHING SOFTWARE ENGINEERING THROUGH COMPETITION AND COLLABORATION

Mark C Johnson, Purdue University

Yung-hsiang Lu, Purdue University

Teaching Software Engineering Through Competition and Collaboration

Abstract

This paper reports a case study in teaching senior-level software engineering using both competition and collaboration. The students were divided into teams to write computer games and competed in the second to last week of the semester. Meanwhile, each team had representatives to write libraries used by all teams. This course adopted several principles of "problem-based learning": students discovered the needed skills as they progressed throughout the semester. Because competition was conducted through a computer network, the students had to learn network programming and synchronization among concurrent programs. A competition required three programs: two players and one referee called "game server". In addition to network communication and concurrent programming, students also learned graphical user interface and game strategies later in the semester.

This course also emphasized team building and team management. Each team included five or six students. Five unique roles were suggested by the instructor and the students decided their roles by exchanging their resumes among their team members. A leader was elected by each team; the leader received bonus points when the team met its schedule and penalty if the schedule was missed. Even though this successfully kept all teams on schedule, team leaders suggested more participation in deciding the grades of team members.

We discovered three important aspects of using competition and collaboration in education. First, students consider collaboration within a team much more important than collaboration across teams. Second, it is important to select a game that is sufficiently challenging for the competition and allows sophisticated strategies. Third, competition itself cannot promote higher quality. If the teams only need to compete within the class without a higher standard, it is possible that all teams resort to simple strategies. Instead, the instructor should provide a reference player that implements an advanced strategy and then encourage students to defeat this reference player.

Introduction

A typical course on software engineering discusses software process, project management, requirement and design, and maintenance.[4,14] While these topics provide a theoretical foundation for the students to construct large-scale software, these concepts can be better conveyed through a semester-long team project. Students can learn how to collaborate with their teammates in the project. A recent study[13] suggested that students would be better motivated through competition. In the spring semester of 2005, a senior-level course on software engineering was taught using both collaboration and competition. In this course, students collaborated in two ways. First, they worked with their teammates in the projects.

Second, each team had representatives to form three cross-team committees. (a) The standard committee defined the common interface and wrote the library so that the program built by each team could compete. (b) The quality committee wrote testing code that used the standard interface. Any team that failed the tests would be disqualified from the final competition. (c) The contest committee decided the competition rules and wrote the code to decide the winner in each game. The committee also wrote a reference player; a team had to beat the player before entering the competition. The teams competed to win the Yali game, a two-player board game. Each player had twelve marbles. The player that could move eight marbles to the other side won the game. All teams had complete freedom in choosing the programming languages.

The course was offered using some principles of problem-based learning (PBL) to teach students how to solve real-world problems.[7,8] The problem was to construct a game that could compete with and win the games against the other teams. PBL was chosen because its procedures met the requirements of the course:[15] (a) The problem was introduced before any lecture. (b) The problem was realistic and could be encountered by the students outside the classrooms. (c) The students were encouraged to apply and integrate their knowledge from other subjects. PBL has been adopted in teaching software engineering,[1,3] first-year CS courses,[9] and programming.[12]

This paper presents our findings in using both competition and collaboration. Each student filed a weekly report about the amount of time spent on the project, overall satisfaction with the project, the evaluations of the other team members, and suggestions about the course. To encourage the students to report accurate amounts of effort, the reported numbers were not used for grading. We learned the following issues in this course. (a) Competition is a strong motivator; hence, collaboration within each team is highly successful. In contrast, cross-team collaboration is not appreciated among many students. (b) To encourage students to exploit their full potential, a strong opponent should be available as a reference. This reference should be provided by the instructor and all teams must beat the reference player to enter the competition. (c) As expected in many team projects, some members gave a greater effort than other members. This problem may be alleviated if the team leaders have more influence on each member's final grade.

This paper is organized as follows. We present previous work on the education of software engineering, problem-based learning, and the basic rules of the Yali game in the background section. The organization section describes the course organization and the students' backgrounds. The discovery section presents our discoveries and discusses how to use our discoveries to improve the course in the future. The paper ends with conclusions and acknowledgements.

Background

Education of Software Engineering

Software engineering is the study of how to design, construct, and maintain high-quality software. Software engineering usually involves four aspects: people, process, project, and product.[4] Many papers and books have been published on the contents of software engineering. Here we list a few recent studies on the education of software engineering. Yeh[16] described several requirements for future software engineers; these requirements included the understanding of business needs and better skills in communication. García et al.[10] discussed a common challenge in many curricula for teaching software: students learned programming first without sufficient understanding of software modeling. Workshops were held to teach software modeling in order to remedy this common problem. The students were required to develop their models and a volunteer group presented their solution. Their success was supported by the students' high passing rate of the examination in software engineering. Evaluating a teaching methodology is often difficult because every student is unique. Karoulis et al.[11] used two groups, one experiment group and one control group, to study the effectiveness of an instructional tool called "lesson sheet". The sheet is a table that associates the course outline with relevant information, such as charts or discussion in class. Chen et al.[5] used a classification tree method to teach black-box software testing. Billard[2] used UML (unified modeling language) and operating systems to teach software engineering. Operating systems were chosen for the foundation of the course because operating systems had a well-understood theoretical background. Thus, the students could learn both theory and implementation of software. Lawrence[13] used game competition to motivate students in learning data structures. When a student finished the programming project, the student could upload the program into a server and compete against the programs written by the other students. Since 2000, IEEE Computer Society has been holding programming contests that allow participants to define their own projects under some loosely specified guidelines.[6] The competition is designed for teams to develop long-term projects with special emphasis on teamwork. IEEE's competition differs from other competitions because the others usually allows the participants to solve well-defined problems within several days. In contrast, IEEE's competition allows participants to develop their software for several months.

Education of Software Engineering using Problem-Based Learning

Problem-based learning (PBL) has been applied to some courses teaching software development. Armarego[1] presented a case study of using PBL for software design. The paper provided detailed description of the course and many quotes from students about the advantages as well as the problems of using PBL. For example, some students appreciated the opportunity to solve problems that were more open and to develop better design skills. Meanwhile, some students were concerned about the vagueness of expectations. Some students also suggested better training in leadership for a course that required teamwork. Boehm[3] taught students to recognize and manage the risks in software development. The risk factors

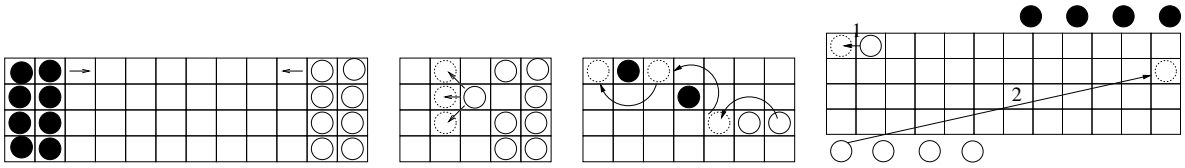

Figure 1: (a) Initial board configuration of a Yali game. (b) Three legal moves. The white marble can move to one of the three new locations shown as dotted circles. (c) A marble can jump over one or multiple marbles. (d) If a marble reaches the destination column (step 1), a side marble can be moved to the starting column (step 2).

were classified into 10 categories, such as personnel, legacy software, and performance. The students learned the skills to handle these potential problems in software. Even though PBL can help students apply and learn new skills for solving problems, it is unclear whether PBL can be still effective when most students lack "basic skills". Fekete et al.[9] discussed the issues in converting the entire first-year computer science curriculum into PBL. The main concern was that students might not fully understand some basic concepts and technical contents, such as complexity analysis, recursion, or inheritance. Their study indicated that students could learn these concepts well if the problems were constructed appropriately.

Yali Game

We chose the Yali game for the course presented in this paper. This game was chosen for three reasons. First, the search space of the game strategy is considerably smaller than some other more complex games such as chess. Second, there were few reference implementations on the Internet so the students had to design their own strategy. Third, this game was used in the past and there was a Yali board and 24 marbles in the Software Engineering Laboratory. Two students could play an actual game using the board. Several students reported that being able to play a game and move the marbles physically helped them design the strategies in computers.

Yali is a two-player board game as shown in Figure 1 (a). Several rules were modified by the contest committee. Each player has 12 marbles, including 4 side marbles. Each player attempts to move the marbles to the other side. There are three legal moves as indicated in Figure 1 (b): forward along the same row or diagonally. A marble can also jump over one or multiple marbles in a single step as shown in Figure 1(c). In the figure, the marble can jump over one white marble, one white and one black marbles, or one white and two black marbles. The three locations noted with dotted circles are all legal moves in a single step. The winner is the first player who moves eight marbles to the other side. Unlike many other board games in which the two players take turns, in Yali the next player is determined by the balance (i.e., center of gravity) of the board. As the marbles move forward, the balance of the board changes. If one player moves more marbles, the board will tilt to the

other side and the other play can move. For example, if one white marble moves forward in Figure 1 (a), the center of gravity moves toward the left and the player of the black marbles can move. Because of this rule, it may not be advantageous to take one move of multiple jumps because these jumps may allow the opponents to make several moves. If one player's move makes the board perfectly balanced, the opponent can make the next move. Once the board's balance changes, a player has five minutes to decide the next move. The player loses if no move is decided within five minutes. In addition to the 8 marbles, there are also 4 "side marbles" that can be used to improve the balance of the board. When a marble reaches the end column, the player can take one side marble to the starting column. This may allow the player to move more marbles by changing the board's balance.

Course Organization

Team Construction

The problem to be solved by the students was to develop intelligent games that could communicate and compete. The problem was given to the students in the first lecture. Very little details were provided by the instructor or the two teaching assistants about how to solve this problem. In particular, we did not specify the programming language (or languages) to use. We did suggest that using the network for communication could allow each team to select their preferred languages. Totally 26 students took the course; half of them were juniors and the other half were seniors. Each student had to commit to a one-hour lab session per week. The students were divided into five teams based on their lab sessions. One team had six members and each of the other four teams had five members. The team structure was suggested by the instructor as follows:

- team leader
- secretary: help leader, manage documents, web master, write testing programs
- architect and representative in the contest committee
- integrator and representative in the standard committee
- tester and representative in the quality committee

In the first lecture, each student was required to submit a resume of relevant background, including a list of familiar tools as shown in Table 1. Some students thought their skills were between two categories (for example, used and familiar) and did not choose either in the table. C and Python were required by prerequisites so all students were familiar with C and Python before taking this course. The lab sessions in the first week were used for team building. The team members had to introduce themselves and read each other's resume for determining the roles in the team. The teaching staff (the instructor and the two teaching assistants) did not assign the roles in each team.

tool / language	D	H	U	F
CVS	9	4	6	5
C++	2	1	7	15
Makefile	2	5	12	5
shell programming	2	3	11	8
Java	9	3	5	7
Python	0	0	12	12
Perl	12	9	2	2
Tcl/Tk	4	2	12	7
gdb	1	3	13	7
ddd	5	7	7	5
UML	16	2	3	3

Table 1: Student background reported in the first week for team building. D: do not know; H: heard; U: used; F: familiar.

Project Schedule

The project schedule was divided into three stages. Totally, 13 weeks of lab sessions are graded and each session is 5% of the final grade. The first stage encompassed three weeks. During the three weeks, each team had to determine the team organization and set up their development environment. The second stage spanned from the fourth to the eleventh weeks. In this stage, each team could decide their schedule. Before this stage started, a team had to submit a schedule. Every week the teaching assistants verified that the team met their own schedule. A reference schedule was provided by the instructor in the first week and most teams followed the reference schedule in this stage. For each week of the schedule, if a team missed its schedule then each member received a 20% penalty for that week and the team leader received a 40% penalty. If the team met its schedule for the week, the team leader received a 10% bonus. The third stage started in the twelfth week and ended in the fifteenth week with the final competition. The third stage consisted of three qualification tests. (a) In the twelfth week, each team had to demonstrate a working game that could compete against a human player and another computer. (b) In the thirteenth week, each team had to use a common game server for playing a game against another computer. We will explain the game sever later in this paper. (c) In the fourteenth week, each team had to defeat a "dumb" player at least once in five games. The dumb player randomly selected a legal move and was created by the contest committee. A team would be excluded from the competition if the team failed any of the three qualification steps. All five teams passed the qualification procedure and entered the final competition. Every team was encouraged to post the latest program (executable without the source code) on their web site in the fifteenth week so that the other teams could test and improve. The final competition was held on the last day of the fifteenth week. The sixteenth week was used for the students to analyze their competition results and to finish the final reports.

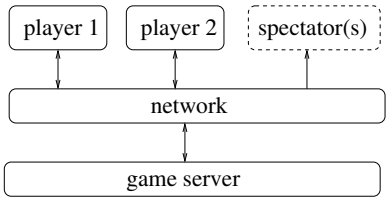

Figure 2: Two players communicate through the network. The game server is responsible for determining the turns and the winner. The server also supports spectators from other machines.

Game Architecture

Each team was allowed to choose the programming language (or languages) for constructing the game. In order to communicate, the standard committee was responsible for providing the communication mechanism and decided to use XMLRPC. XMLRPC allows programs on different machines to communicate using the Hypertext Transfer Protocol (HTTP). For the programs, such communication is similar to calling a function even though the implementation of the function is in another program on a different machine. This is called a remote procedure call (RPC). The format of the call is flexible by using the Extensible Markup Language (XML). Figure 2 shows the architecture of the games. Two players communicate through the network. The game server maintained the current location of every marble, calculated the board's center of gravity (and which player to move), and decided the winner. The server also allowed spectators from other machines. Because a spectator could not move the marbles, the figure used a one-direction arrow to indicate that the spectator was only a listener. This architecture allowed the teams to select their preferred languages. Since Python was taught in a prerequisite course, four teams used Python and C/C++. They used different tools to build the user interface, including Qt, Glade, and OpenGL. One team used Java for both the game strategy and the user interface. No team used a very complicated strategy so we did not feel that the choice of languages had any obvious impact on the competition results.

Discovery and Discussion

Cross-Team Collaboration

The three committees were formed to facilitate cross-team collaboration. The teaching staff encouraged every team to take advantage of the additional developers from the other teams in the committees. Among the three committees, the standard committee was considered the most successful because it worked closely with all teams. The contest committee decided the rules and wrote the reference player but did not contribute additionally to the five teams. The quality committee wrote several test programs to ensure that each team's player could communicate with the game server. No team requested the contest committee to provide

a more intelligent player and make the qualification process harder. No team asked the quality committee to write more test programs. We believe this was due to the following factors. First, most students worked closely with their team members and they did not treat the committees as resources. Second, many students were unfamiliar with the concept of RPC. The students used different programming languages and believed that common test programs could not be beneficial. Third, there was no grading incentive to improve the committees. In the future, we will consider separating the teams from the committees and possibly combining all committees into one. By doing so, the committee members can be more focused in providing the library and finding other teams' problems.

Team Management

Each student was required to report weekly efforts through a web site. The report was divided into the number of hours spent on the project, group meetings, and committee meetings. The reported numbers were not used for grading so students were encouraged to report the actual numbers. Each student spent between 55 and 200 hours throughout the whole semester of 16 weeks. On average, each student spent 100 hours on the project over the whole semester, including the meetings. The amounts of time varied significantly: the standard deviation was 45 hours. The median was 83 hours. The student that spent the most time was the chair of the standard committee because the committee constructed the game server used by all teams. Group leaders consistently spent more time than their members across all teams. One group leader suggested that this could have been caused by the grading policy. One group leader spent 90% more time than the group average. While there was incentive (bonus and penalty, explained in Section) for group leaders to meet the schedule, there was no authority by the group leaders to award or punish under-performing members. As a result, the leaders had to help those members and, in some cases, took over those members' responsibilities in order to meet the schedule. The course used weekly peer evaluation as part (5%) of the grades but the leaders did not have additional influence on the grades. One solution is to allow the leaders to participate in the grading process of the group members. It is unclear, however, whether the other students will respond positively if their grades are heavily affected by their group leaders.

Game Strategy

Another observation is that the competition did not produce highly sophisticated game strategies. Two teams decided to use greedy approaches for the game strategy. They assigned different scores to the columns and moved a marble to achieve the highest score without considering the opponent's move. Another team used the min-max algorithm with alpha-beta pruning without considering any jump over multiple marbles. Because no marble could be "captured" by the opponent's marble (like a chess game), many students felt that analyzing the opponent's potential moves would provide little information. In particular, one group discovered a strategy to win by keeping one marble in its original location. As the other marbles were moved to the other side, the board balance would give the opponent the turn

to move. The opponent could encounter a situation when there was no legal move and thus exceed the five-minute decision limit. This strategy was discovered several days before the competition and none of the other teams could beat this strategy. The contest committee decided that it was too late to change the rules and forbid this strategy. This team won the final competition. In the future, we intend to improve the game strategies in two ways. First, we plan to change the rules so that, when there is no legal move by one player, the other player has to move within five minutes. Second, in spring 2005, the reference player for qualification selected a legal move randomly. All teams successfully defeated the reference player and entered the final competition. We plan to improve the intelligence in the qualification process. If this player is more intelligent, the students will have to improve their strategy in order to qualify for the competition. In Lawrence's study,[13] a four-level search algorithm was used for competition and beating this algorithm could receive more points than beating a simpler algorithm. We believe that a better reference player can help the students set a higher goal and design better strategies.

Concurrent Programming

The game architecture as shown in Figure 2 has three programs running concurrently: the two players and the game server. Many students did not have experience with concurrency. Consequently, they did not anticipate some legal sequences of events. For example, the board's center of gravity sometimes allows an opponent to move several marbles before control passes to another player. As a result, one team's program crashed after receiving the update from the game server. This was because the team's program did not expect multiple marbles to have changed locations at once. Another concurrency related race condition occurred in an earlier version of the game server. When a player moved a marble and informed the game server, the game server did not send an acknowledgment. As a result, the player could move multiple marbles before the game server updated the board configuration and informed the other player. The problem was solved later by the standard committee using synchronous communication between the players and the game server. A player could not move another marble before the first move had been acknowledged by the server. Because of the nature of this project, we believe it will be helpful to all students if some basic concepts about concurrent programming, synchronization, and race conditions are introduced early in the semester.

Intelligence and User Interface

The competition results were 5% of the final grade. We used this small percentage in order to encourage the students to pay more attention to the development process as well as documentation and user interface. The user interface was another 5% of the final grade. Four teams paid special attention to improve their user interfaces. One team added cartoon characters to the surrounding of the board. Another team used movie posters as the board background. The third team allowed users to change marble colors. The team that received the highest score in user interface used OpenGL with three-dimensional view of the board.

Two animated figures stood on the opposite sides as the players. A player could change the camera angle to see the board from the top, the side, or as a player.

One team decided to provide a simple user interface and focused on the game strategy, even though the teaching staff repetitively reminded the team members that the user interface had the same weight in grading as the competition result. The team developed two strategies: a simpler one for testing the user interface and communication and a more complex strategy for competition. Unfortunately, they did not submit the advanced strategy on time (claiming to be only three minutes late) and the course policy did not permit any late submission. As a result, they were forced to use the simple strategy and lost all but one game. The team received lower scores, compared with the other teams, in both competition and user interface. This experience may help the team members understand that software development should be balanced. Focusing on the game strategy without improving the user interface could lead to lower scores in both. The team also learned the importance of meeting the deadline.

Newsgroups

Six newsgroups were established: one for the whole class and five for the the individual teams. Most postings in individual groups were related to team management, such as the time for group meetings and status updates. In the class newsgroup, two types of postings were the most common. The first type was announcements and clarifications by the teaching staff. The second type stated problems related to the game server. The game server was exercised by all groups in many different ways. Moreover, the standard committee was changing the interface of the server to solve some problems that were not expected earlier. Thus, many problems about the game server were reported to the newsgroup and the game server was regularly improved. Sometimes a new version of the game server caused multiple problems and many students reported problems through the newsgroup. This experience suggested that a newsgroup was ineffective for the library used by all students. Instead, we would need a bug tracker for the game server.

Future Offering

Some students suggested that Yali was not challenging enough to develop intelligent strategies for the game. A more sophisticated game should be chosen in the future. Several students were concerned that their teams could not continue if one member left the team due to, for example, dropping the course. One suggestion was to announce, in the first lecture, that during the semester the instructor would randomly change the team members. This would give the members the opportunities to learn how to handle personnel issues that could happen in real-world projects. Many students suggested that the source code and the documents be passed to future students so that they would not have to build everything from scratch. Finally, a better tracking system could help balance the workload among the students. We used individual reports to estimate the amounts of time spent on the project.

Conclusion

This paper presents a case study of using both competition and collaboration. The students collaborated in teams and competed against the other teams. The students also collaborated across teams to develop the game server, a reference player, and test programs. We adopted several principles in problem-based learning; the problem was to develop computer games that could compete. We report several observations and provide suggestions to improve future offerings of the course.

Acknowledgments

Prof. Lu is supported in part by National Science Foundation CAREER CNS-0347466. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors."

References

[1] J. Armarego. Advanced Software Design: A Case in Problem-based Learning. In *Conference on Software Engineering Education and Training*, pages 44–54, 2002.

[2] E. A. Billard. Introducing Software Engineering Developments to a Classical Operating Systems Course. *IEEE Transactions on Education*, 48(1):118–126, February 2005.

[3] B. Boehm and D. Port. Educating Software Engineering Students to Manage Risk. In *International Conference on Software Engineering*, pages 591–600, 2001.

[4] E. J. Braude. *Software Engineering: An Object-Oriented Perspective*. Wiley, 2001.

[5] T. Y. Chen and P.-L. Poon. Experience with Teaching Black-Box Testing in a Computer Science/Software Engineering Curriculum. *IEEE Transactions on Education*, 47(1):42–50, February 2004.

[6] A. Clements. Constructing a Computing Competition to Teach Teamwork. In *Frontiers in Education*, pages F1F–6, 2003.

[7] R. Delisle. *How To Use Problem Based Learning in the Classroom*. Association for Supervision and Curriculum Development, 1997.

[8] J. Dewey. *Democracy in Education*. MacMillian, 1963.

[9] A. Fekete, T. Greening, and J. Kingston. Conveying Technical Content in a Curriculum Using Problem Based Learning. In *Australasian Conference on Computer Science Education*, pages 198–202, 1998.

[10] F. J. Garcia and M. N. Moreno. Software Modeling Techniques for a First Course in Software Engineering: A Workshop-Based Approach. *IEEE Transactions on Education*, 47(2):180–187, May 2004.

[11] A. Karoulis, I. G. Stamelos, L. Angelis, and A. S. Pombortsis. Formally Assessing an Instructional Tool: A Controlled Experiment in Software Engineering. *IEEE Transactions on Education*, 48(1):133–139, February 2005.

[12] J. Kay and B. Kummerfeld. A Problem-based Interface Design and Programming Course. In *SIGCSE Technical Symposium on Computer Science Education*, pages 194–197, 1998.

[13] R. Lawrence. Teaching Data Structures Using Competitive Games. *IEEE Transactions on Education*, 47(4):459–466, November 2004.

[14] R. S. Pressman. *Software Engineering A Practitioner's Approach*. McGraw Hill, 5th edition, 2001.

[15] D. Roins. *Problem Based Learning for Math and Science*. SkyLight, 2001.

[16] R. T. Yeh. Educating Future Software Engineers. *IEEE Transactions on Education*, 45(1):2–3, February 2002.