International Journal of Informatics and Information System ISSN 2579-7069
Vol. 1, No. 1, September 2018, pp. 28-36

An Alliance-Based Term Project in Software Quality Courses: a Lesson Learned

Nien-Lin Hsueh [1,*]

[1] Feng-Chia University, Taiwan
[1] nlhsueh@fcu.edu.tw *;
* corresponding author

Abstract

Software testing education has become important in the field of software engineering education. In the previous software quality assurance course, students were asked to form teams to complete a term project. By working on term projects, students can learn programming skills and test skills in a practical way. However, from the experience of the last 3 years, we found that students only did unit testing and system performance testing well but did poorly in integration testing. In addition, students do not yet have the concept of system decomposition and integration, even though it is important during software development. In this paper we report our improvements to software testing course design - an alliance-based approach. In the term project, students are organized into teams and many teams are grouped into alliances. Each alliance has a team of masters building game platforms for others. The master team must define the application interface to interact with other gaming teams, and they must perform integration tests based on the defined interface. In this paper we report our experiences and student feedback on the educational approach.

Keywords: Software Testing; Software Engineering Education; Software Engineering;

1. Introduction

The Software Engineering Program has been conducted for many years in the department of Information Science and Computer Science, Feng Chia University. Within this program, we offer many specialized courses including object-oriented software engineering, human-computer interface design, and software quality assurance and testing (SQA). SQA is one of the most important subjects in this program. As many project-based [1, 3,4] and game-based [9,13] approaches have been proposed to assist software engineering education, we have been inspired to implement projects and games into our SQA courses in the last three years.

In this course, we introduce software testing techniques, software quality models and the concepts of the CMMI software process improvement framework. One of the features of the SE program is that students have to implement software projects as term projects.

In our first year course, we offer an 18 week course covering the following topics:

- **Part I Basic concepts.** Introducing the basic concepts of software, quality, software quality, software quality assurance, and software quality models.
- **Part II Testing Approach.** Introduce black box testing, white box testing, integration testing, system testing and acceptance testing. In black box testing, we introduce partition-equivalent testing, decision table testing, all pair testing. While box testing includes data flow testing, mutation testing. In system testing we teach stress testing, usability state testing, scenario testing. In addition, static code checking is discussed in our course.
- **Part III Quality Management.** Introduce the IEEE 829 testing document standard and the concept of software quality process improvement. The CMMI framework is also introduced.

To improve hands-on practice, we ask each student to complete the Online Exam System with the function in the final project project:

- Teachers can give exams; students can attend online exams; the system can evaluate the inspection automatically.
- Supports at least two types of questions: true / false and multiple choice questions.
- The teacher can allocate a score for each test, for example T / F 5 points each and multiple choice 8 points each, and the total score is 100 points.
- Account management, including enforcing accounts and setting up access authorizations.

Most students fail to complete projects and tests because the project is too big for them in the first year. When we reviewed the course design and feedback from students, we also found that students did not do much testing work but put all their effort into system programming. Programming is not the main objective of this course. Therefore, we turn the project into a game project which is carried out in another course in our SE (Object-Oriented Software Engineering) program. Students can expand the function of the game and test the system. Next, we reduced the lecture section from 3 hours to 2 hours to offer 1 hour of software testing practice, including the following topics:

- Debugging: Using the Eclipse IDE for debugging, including how to use break-points, watch variables, perspective debugging.
- Preventive programming: Implement Java Statements to implement defensive programming; UseJava Exception to handle programming exceptions.
- Unit tests: Implement the JUnita automated testing framework for unit tests and integration testing.
- White box test: Implemented EclEmmaplug to help calculate program branch coverage.
- Performance testing: Applying JMeter and Selenium to perform stress testing.

Even though we have done team-based projects, integration testing is difficult to practice because all members are on the same team thus defining the interface and doing integration testing is useless. To solve the problem, in our third year we introduced alliance based games into our course where teams were forced to discuss the interface and integration test strategy. The remainder of this paper is organized as follows. Section 2 provides an overview of software testing education. Part 3 describes our project design in the course. Section 4 describes the results of applying our approach to the course and Section 5 is our conclusion.

2. Related Works

In this section, we introduce some related work on software testing education.

Bowyer and Hughes performance of continuous integration education in software engineering courses [8]. Assessments are made of students' agile experiences rather than project outcomes, using a new set of process measures that examine student participation and performance in agile testing. The results indicate good participation by pairs of students, and clear understanding of agile processes and configuration management. However, the design and implementation of the interface are carried out by the same team, which reduces the efficiency of the test cases.

Janzen and Saiedian evaluated student effects using test-first and test-last approaches in the initial programming course [2]. Software testing, programmer productivity, programmer performance, and programmer opinion were compared between the test-first and test-last programming groups. The results showed that the test-first approach could improve student testing and programmer performance, but early programmers were very reluctant to adopt the test-first approach, even after having had positive experiences using TDD.

In [7], the authors discuss experiments conducted with undergraduate students in a year-long software engineering capstone course. The aim of their experiment was to compare the Test-First methodology with the Test-Last methodology in an undergraduate software engineering capstone course. This experiment evaluates the productivity of the programmer, the internal and external quality of the product, and the programmer's perceptions of the methodology. Surprisingly, the results from this experiment differed from many previous studies: the final test team was actually more productive and wrote more tests than their first test counterparts. Although more students showed a preference for the test-first approach, concerns about learning and applying TDD with unfamiliar technology were noted.

In [12], large and small group projects are carried out in capstone and software engineering courses. Teamwork is seen as an increasingly important aspect of education in Computer Science and related fields. One common way to include teamwork in an undergraduate curriculum is to include team projects in a wide variety of course offerings. They have taught large-group and small-group projects in upper division undergraduate programs in the past.

3. Alliance-Based Capstone Project

Prof. Shaw has identified the importance of distinguishing between different software development roles [11]. In our alliance based capstone, different students can join different teams and play different roles. In the master team, students need to define the interface between the master platform and the game system. In a game team, students need to develop and perform unit tests for their own play.

3.1. System Requirement

The aim of the term project is to help students learn practical testing skills in software development, and also improve programming skills. Teamwork is also a key element we want students to learn while working on projects.

What to develop. In contrast to the term project earlier in this course, the aim of the new course is to develop a network-based game in which the player can play the game over a network connection. A game center was developed to control all games. Since this project is a bit complicated for students, integration tests are also a big challenge for students. The following are the project requirements:

We plan to develop the gaming community; each team must develop a network-based game. The game status must be published in the game center so that everyone can see the game status through the game center panel. Game center also displays the status of all users when they log in, for example, waiting (other players) to play (with other players), or the winner. The game center also features the Hero Board of each game.

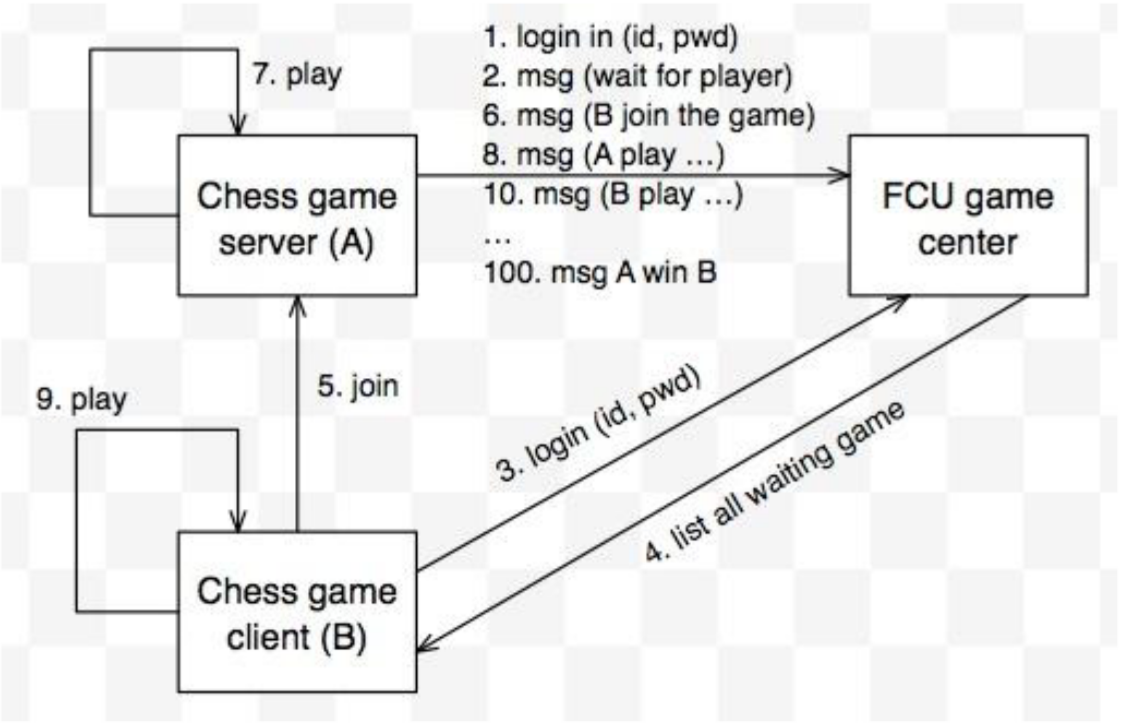

Figure. 1. Interaction between game center and game

The following is a possible scenario:

1) PlayerA enters the Game Center (GC), applies for an account, sets a password and alias;
2) PlayerA sees all game descriptions, and downloads the games he is interested in;
3) PalyerA installs the game and initializes it as a game server, then it enters a waiting state waiting for other players to join. The GC will display information for inviting other players.
4) PlayerB waits for information from the game, he downloads the game (if he has never installed it before), too. After he gets into the system, GC will send a list of games to him. When he selects a game to play, PlayerA and PlayerB hook up and start planning the game.
5) Other players can see the status of the game, for example Red knight moves to location (4,3), Black soldier kills Red Soldier, etc. Information is displayed on the game center panel and is provided by G1.

The second scenario describes how users can see the game board:

1) Anyone goes to the GC to view the Gradebook of all the games. If the user is not logged in, he can only see masked names like Chang xx or Huang xx.
2) PlayerA logs in on GC, he can see all his game history, for example, W3L5 (Win 3 and Lose 5). When he clicks on the game details, he can see his opponent in the game and replay the game.

In short, a Game Center function should have:

- A web based server, every alliance must have a game center and a web server.
- The functions of the web server are: (1) account registration and authorization; (2) receive messages from the game client and display them; (3) calculate the grade and rank (4) describes each game and the rules of the game.
- The Game Center team must define the protocol between the GC and the game, so that they can communicate.

Every game of chess must be on a client-server architecture, so that two players can play together. A chess game can be a game of Taiwan chess, a game of Chinese chess, a game of Go chess or a game of chess five or chess of Apple.

3.2. Testing Requirement

Testing requirements. The following are the testing requirements for students.

- **Defensive programming.** Use statements and exceptions to improve the testability of your code.
- **Testing specifications.** All alliances should list all functions in the specification, and develop your test plan based on those specifications. A good and complete test must include all functional tests in the specification. Moreover, there are many rules in every chess game. Each rule must be tested in detail. How can I use JUnit to assist with testing? how to use equivalence testing to design tests? Demonstrate testing in your test plan and design.
- **Testing coverage.** How do I use EclEmma to check test coverage?
- **Exploratory testing.** Test your system like a detective.
- **Integration test.** If your alliance has six games, can all of them interact with the game center properly? The center can assume multiple virtual clients when developing their central system; on the other hand, game teams can become virtual centers while developing their own games. Then, both of them test with stub components. When the unit tests are done, they then integrate all the systems and test if the interface is ok. The integration test is the main part of what we want to do in this course.
- **Stress test.** How many players can play the system at the same time? What about the response times for different workloads?
- **Automatic test.** Is it possible to test your system automatically? Manual tests on the game of chess will take a lot of time. For static reviews, is there a good way to check your code?
- **Test the system.** Is your test easy to use? You should test usability based on usability heuristics. Apart from that, stress test, test your system security or compatibility.

To complete a project, students must follow the process:

- **Cooperate.** 2-5 students form a team; and decide they will play a master team to develop a game center, or a game team to develop a chess game;
- **Negotiation.** In order to avoid all teams deciding to develop the game and no center being established, at this step the instructor encourages several teams to establish centers. All centers and games are established before the 5th week.
- **Development and testing.** Weeks 5-14 are a major development period. Students in the same alliance should discuss the interface between the center and the game on a weekly basis.

- **First review.** At week 15, all centers and teams have to present their prototypes. For the center, they presented the application interface (API) and the protocol between the center and the game. For games, they have at least demonstrated the game and explained how to integrate with the center. In the review, the instructor confirmed the interface was compliant and the test plan was fine.

- **Final review.** At week 17, students make a revision of the comments on the first review. Students make presentations in class to demonstrate system requirements, system architecture, system demos, test results, and lessons learned from the project. Lessons learned included the surveys in Table 1 and what skills were applied to their projects.

4. Results

Courses are conducted for two classes with 74 students in the same semester. In total there are 18 teams, 4 of which act as game centers. Center 1 and Center 3 both have 6 teams joining the alliance. Center 2 has only two teams, and Center 4 doesn't have any teams. To validate the architecture of Center 4, Center 4 is also required to develop games.

Most teams develop their own games. In 15 game projects, two game teams used open source from our object-oriented software engineering courses, and three teams modified programs from the web. Modifying existing projects is allowed because they have to make a lot of modifications to integrate with the central project. Apart from that, what we care about are the testing techniques applied in the project. Figure 2 is a snapshot of a game center. As you can see, there are four matches in the alliance.

Figure. 2. Snapshot of Game Center

Figure 3 is a snapshot of the use of EclEmmatool in a project, the green part means the statement is fully tested by their test cases; yellow means only a part of the condition was tested and red means all conditions (or statements) were tested. All teams should report the application of the test technique (or tool) in their final report.

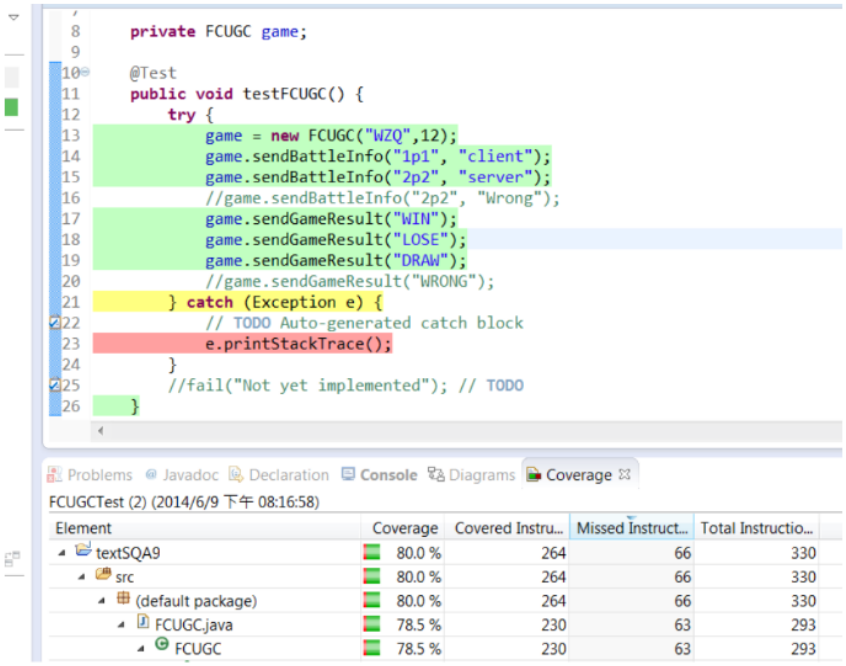

Figure. 3. Using EclEmma for Coverage testing

Table 1 shows our first survey of the testing practices implemented in the project. Each item is assigned a score from 1 to 5 where 5 indicates significantly applied practice. The exploratory testing concept is easier for students; they can play the game in a free way to test the system. Automatic testing, although the concept is easy but they do not have enough time to write a test program. Integration testing, a major experiment in the term project design, has far better applications than last year.

Table. 1. The survey about the testing requirements

Question	Degree
Defensive programming	3.4
Specification testing	3.8
Exploration testing	4.2
Coverage testing	3.2
Integration testing	3.8
Stress testing	3.0
Automatic testing	1.2
System testing	2.1

We have a further survey on alliance based project design. The question is "Through the term project, I cultivate ..." (I skipped the first sentence only for paper layout). In the survey, we differentiated game centers and games to explore possible differences. Because not all students on the game team joined the discussion about interface determination and integration testing, their scores were lower than those of the game center students. In the first and second questions, we can see that students really understand the importance of interface definition and system integration, but they still feel weak in integration testing. However, students made great progress on this issue and we are pleased that students can learn other teamwork practices.

Table. 2. The survey about the integration test

Question	Center	Game
I further understand the importance of defining interfaces between modules	4.8	4.5
I further understand the issues of software integration	4.2	4.0
I further understand how to perform integration testing	3.8	3.6
I further understand the importance of team communication	4.2	3.8

5. Conclusion

Software testing is an important practice in the field of software engineering. More research is introducing a project-based approach to software engineering education. However, they focus on unit testing or development first testing. From our previous experience, integration testing is the most difficult topic to learn in a software testing course. To solve the problem, we designed an alliance-based project approach in which students create interfaces between teams and performance integration testing.

Since developing the entire system is difficult for most students, we allow students to expand open source to meet project requirements. However, building such a game alliance system in one semester was still very difficult for students. Assuming that the developed system is incomplete, students can learn many software engineering practices and integration testing concepts and skills. In the future, we plan to improve our course design from the following perspectives:

- **Background skills.** Some basic skills like network programming web programming need to be improved before taking this course.
- **Sample project.** Give students an example of a project, so they can find out how to design a medium-large system in an alliance.
- **More practice.** We spent too much time testing the theory at the start of the course, but the students didn't understand it very well. If we teach theory after the project is over, maybe students can learn more.
- **Documentation.** Essential documentation for a project. All minutes of meetings at the meeting, functional specifications, design specifications and test plans should be recorded and reviewed by all members. Several projects failed due to bad documentation.

References

[1] Alan J Dutson, Robert H Todd, Spencer P Magleby, and Carl D Sorensen. A review of literature on teaching engineering design through project-oriented capstone courses. Journal of Engineering Education, 86 (1): 17–28, 1997.

[2] David Janzen and Hossein Saiedian. Test-driven learning in early programming courses. In ACM SIGCSE Bulletin, volume 40, pages 532–536. ACM, 2008.

[3] David Umphress, T Dean Hendrix, and James H Cross. Software process in the classroom: The capstone project experience. IEEE software, 19 (5): 78–85, 2002.

[4] Jane Huffman Hayes. Energizing software engineering education through real-world projects as experimental studies. In Software Engineering Education and Training, 2002. (CSEE & T 2002). Proceedings. 15th Conference on, pages 192–206. IEEE, 2002.

[5] Jeffrey Carver, Letizia Jaccheri, Sandro Morasca, and Forrest Shull. Issues in using students in empirical studies in software engineering education. In Software Metrics Symposium, 2003. Proceedings. Ninth International, pages 239–249. IEEE, 2003.

[6] Jesús Favela and Feniosky Peña-Mora. An experience in collaborative software engineering education. Software, IEEE, 18 (2): 47–53, 2001.

[7] John Huan Vu, Niklas Frojd, Clay Shenkel-Therolf, and David S Janzen. Evaluating test-driven development in an industry-sponsored capstone project. In Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on, pages 229–234. IEEE, 2009.

[8] Jon Bowyer and Janet Hughes. Assessing undergraduate experience of continuous integration and test-driven development. In Proceedings of the 28th international conference on Software engineering, pages 691–694. ACM, 2006.

[9] Kajal Claypool and Mark Claypool. Teaching software engineering through game design. In ACM SIGCSE Bulletin, volume 37, pages 123–127. ACM, 2005.

[10] Mark Ardis and Gary Ford. Sei report on graduate software engineering education. In Software Engineering Education, pages 208–249. Springer, 1989.

[11] Mary Shaw. Software engineering education: a roadmap. In Proceedings of the conference on The future of Software Engineering, pages 371–380. ACM, 2000.

[12] Michael V Stein. Using large vs. small group projects in capstone and software engineering courses. Journal of Computing Sciences in Colleges, 17 (4): 1–6, 2002.

[13] Nergiz Ercil Cagiltay. Teaching software engineering by means of computer-game development: Challenges and opportunities. British Journal of Educational Technology, 38 (3): 405–415, 2007.