Gamification of Software Testing and Common Patterns of Mistakes in Unit Test

Lu Keyuan

***Abstract*— This is the interim report for the final year project *Gamification of Software Testing and Common Patterns of Mistakes in Unit Test*. This project includes a research on common mistakes testers make in unit test and an implementation on gamification of learning and debugging a program.**

I. BACKGROUND

Software Testing always plays an important role in the industry of software engineering. For most enterprises, testers hold a status similar to core developers. Despite the importance of testing, software testing is usually not taken seriously in university study, neither in Computer Science nor in Computer Engineering. Thus, the software testing ability that most coder hold does not meet the need of Software development projects. Although there are courses like Software Engineering and Software Testing that make introduction to testing, there is still a desire to put more importance to Software Testing in general CS study. The lack of training in software testing leads to some bad habits or mistakes in writing tests.

What's more, there is a global trend of involving programming in several majors. For example, artificial intelligence in biology, automatic analysis implemented by Python in business analysis and using of Matlab in Mathematics. Thus, there is an increasing need for new testers. This trend explains why a more fun and efficient way of learning Software Testing has a high potential value in the market.

The process of learning is however commonly known as difficult and boring, which puts up challenges to educators. This leads to an unsolved problem, "How to improve students interests in learning?" One of the solution is gamification, turning the process of learning into fun tasks, the student become players and compete with each other for a better score. During the competition, students could acquire new knowledge more effectively

There are some online game focus on software testing like CodinGame[1], Code Hunt[2] and Code Defenders[3]. These online games provide platforms for players to write tests for given methods. As long as the tests are correct, players would get accumulate some points or beat a boss in the game.

II. LITERATURE REVIEW

A. Gamification

It is well known that game is interesting and entertaining, while for most people, study is boring and hard to focus on. So if the procedure of study can be combined with elements of games, it would be much easier to absorb new knowledge. In order to have a better understanding on gamification, I made a research on game elements.

The use of computer games as an educational tool[2] shows that elements of a game can be divide into two parts, abstract interface and concrete interface. Most of the concrete components are visualized in a game, containing Graphics, sound, technology, manipulation, memory, logic, mathematics and reflexes. On the contrary, the abstract components are important while hard to visualize, like the mode of playing, exploration, challenges, engagements, critical thinking, discovery, goal information, goal completion and practice. This is similar to a much simpler conclusion made in *How to Gamify Software Engineering*[4]. Reward and satisfaction are what triggers players motivation for a game. Besides, challenge, fantasy and curiosity are the three elements which influence the engagement in a game. With the results above, the challenge for this part is clear: How to turn the learning process from a boring one into a fun one with positive feedbacks

Fraser etc al made some investigation on software testing games like Code Defender and CodeinGame in *Gamification of Software Testing*[1]. In CodeinGame, the users controls a certain character in the game. To make movements, the player should write correct test for given code. Each individual test controls a selected action and after the whole test is right, the boss(enemy) in this scenario can be defeated by the player. This is a classic RPG design(Role-playing game) where players get satisfaction by empowering themselves in the process of game and finally beat the ultimate enemy of the game. Because of the process of the empowering in RPG, it is a good way to combine the growing level with a learning process, which is efficient to keep users attracted to writing tests. Studies confirm this result. For writing the same test, players of the CodeinGame focus more on their tests than the testers writing tests in IDE.

Code Defender shows another design of the software testing games. There is neither fictional characters nor fictional enemies, however, the enemy in this game is another actual player. To play this game, one player should create a battleground and choose a game class, the game will automatically generate a series of code. After another player joined the same battleground, the game starts. In each turn of the game, the attacker creates a mutant in the code and the defender write tests to kill the mutant. In this kind of design, the players get satisfaction by beating actual players and getting their rank up in the leaderboard.

[1] https://www.codingame.com
[2] https://www.microsoft.com/en-us/research/project/code-hunt/
[3] http://code-defenders.org/

B. Common mistakes in software testing

Another research on common mistakes in software testing is done because one of the aims of this project is to teach users of our system on writing correct tests to repair some common bugs in writing tests.

It is well known that every coder has unique habits when coding, some are good while some are bad for programming. Bad habits leads to incorrect codes. With materials searched from the Internet[4][5], combined with data I gathered by interviewing my classmates, here are some common patterns of mistakes when writing tests, especially unit tests.

1) Multiple assertions in one test: Normally there should be only one assertion in one test. But some testers write more than one assertion in a unit test. Although the test can still pass when all the assertions are passed, if the test fails, the tester cannot recognize the failed assertion at once. This will influence the efficiency of software testing.

2) Test only the successful path: It is happy to see the test passes and the program is running steadily according to the original design. However, passing the tests based on the successful paths does not mean that there is no bugs in this program. Statistics shows that most of the bugs found by testers are not related to these successful paths[7]. Testers should make sure that the program can handle the situation correctly under the circumstance that the user may type an illegal input.

3) Tests should have descriptive name: Among the programmers whom I interviewed, some of them have a habit of naming the test method as test1, test2, test3 and so on. This does not do much harm when they are testing by themselves because they know exactly what each test are for. However, it gives much pressure to other testers who wants to learn or check their tests for the reason that this kind of name gives limited information. A concrete name shows the target of a certain test method correctly

4) Not specific enough: There exists the situation that the test passes but bugs still exist while using the program. One of the cause of this situation is that the test is not specific enough. For example, when testing a specific input of a string *"input=Gamification"*, the right assertion should be *"assertEqual("Gamification", input)"*;. However, a test of *"assertNotNull(input);"* can also pass but not specific enough.

5) Only invoke the method under test: According to the testers whom I interviewed, I found that when invoking some simple method under test, few of them write individual test for this method. This leads to a bad habit of missing necessary tests. To make sure that the program runs without bugs, every method which may leads to error should be tested, even if the return value of this method is only a Boolean.

6) Test private method: Sometimes there existing some test methods which test private method in a program. However, the target of unit test is normally to check the correctness of the codebases public API. If there is a need to test the private method, instead of testing this method directly, a better way is that by testing the public method that calls it, the result can show if there are any bugs inside the private method.

7) Dependencies between tests: Under some circumstances, there are chaining tests, which means the result of a method has a directly influence on the next method. For example, in method1, an object is initialized. Then it is passed to the next method, method2, together with its parameters. To make a test on method2, it is wrong to include the initialization step in the testing method. Method1 should be tested first to make sure that the process of initialization does not go wrong.

III. Implementation

This project is based on a Java Web project created by Maven. It is connected with a MySQL database which stores information of users. Users can register an account in the registration page. Thus, a record will be added in to the database. In the login page, the system will compare the username and password the user input with the correct username and password stored in the database. Once the username and password are matched, the user login successfully and move into next part of this project, the learning part.

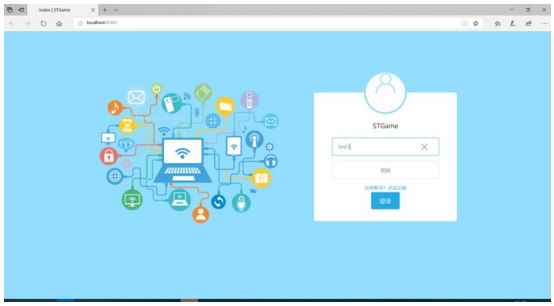

Fig. 1. Login page

The learning part shows some common patterns of mistakes that a tester may make when writing tests. Each page contains several tests, only one of them is correct and others contain one of the common mistakes. Each time a user select the correct answer from all the tests, certain points would be added into this account. After the learning part finishes, the system will added up all the points that the user get. If the total points reaches an already settled number, the user would be allowed to reach the next level, writing test by himself. Otherwise, he would be asked to study more on these testing mistakes and start the learning and answering part over again until he gets enough points

In the next part, users are given some methods and test requirements. Users can write tests in the blanks and the system will automatically judge the correctness of the tests. Each correct test will also earn certain number of points for the users. There will be a leaderboard in the system. Once a user gets some points, the profile of this user will be updated

[4]http://softwaregarden.io/common-mistakes-junior-developers-writing-unit-tests/

[5]https://kentcdodds.com/blog/common-testing-mistakes

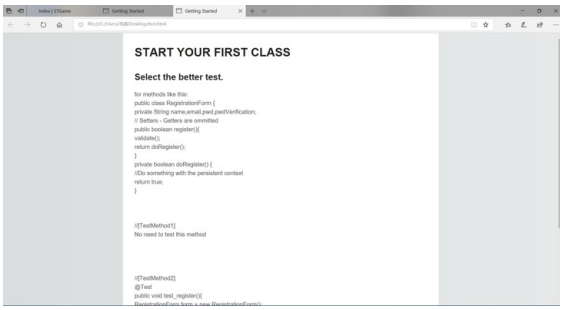

Fig. 2. Learning page

together with the leaderboard. With this leaderboard, the enthusiasm of users can be improved by the will of earning more points than other users

IV. Currently Done

Used Maven to build the Java Web project. Set up the database of the project. Implement of the login/register function. For the learning part, the questions are designed as basic questions. The logic of the answering part is successfully set up, as the system can automatically judge the correctness of the selection made by the users. Some artwork jobs are also done on the pages of login, register, learning and answering questions to make the UI more beautiful.

V. Future Work

The correctness of the questions that users answer is not connected to the database. Each players point will be recorded in the database and there will be a judgement on whether the user is allowed to enter the practice part.

In the actual practice part, there will be an online judgement system which automatically judge the correctness of the test written by the players.

References

[1] Fraser, Gordon. "Gamification of software testing." 2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST). IEEE, 2017.

[2] Amory, Alan, et al. "The use of computer games as an educational tool: identification of appropriate game types and game elements." British Journal of Educational Technology 30.4 (1999): 311-321.

[3] Deterding, Sebastian, et al. "Gamification. using game-design elements in non-gaming contexts." CHI'11 extended abstracts on human factors in computing systems. ACM, 2011.

[4] Dal Sasso, Tommaso, et al. "How to gamify software engineering." 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2017.

[5] Stewart, David B. "Twenty-five most common mistakes with real-time software development." Proceedings of the 1999 Embedded Systems Conference (ESC99). Vol. 141. 1999.

[6] Aniche, Mauricio Finavaro, and Marco Aurlio Gerosa. "Most common mistakes in test-driven development practice: Results from an online survey with developers." 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. IEEE, 2010.

[7] A. Vahabzadeh, A. M. Fard and A. Mesbah, "An empirical study of bugs in test code," 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, 2015, pp. 101-110.