Gamification of Software Engineering Education

Eman Sherif Emily Bledsoe Andy Liu Brian Nguyen

November 2019

1 Research Context and Problem Statement

An important practice that programmers often find tedious and time-consuming is testing their code. However, unless code is properly analyzed and executed, no one can guarantee its functionality and correctness. Today, society is so reliant on software that a minor bug in the code can cause major inconveniences in our daily lives. Examples include: identify theft, device malfunctions, or even glitches in video games. Still, many software engineers struggle to create effective test cases and fail to see the importance of testing. This can be attributed to the lack of proper reinforcement during their education [4]. Students in software engineering courses find testing to be trivial and uninteresting. To ensure that students learn proper software testing during their programming education, we want to create a method to facilitate the process.

Since software testing is such a broad subject, we've decided to narrow our focus to statement coverage. Statement coverage is a principle within software testing where the user tries to have every statement executed to verify its functionality. Rather than testing every input the code could execute, statement coverage reduces the number of test cases and maximizes the amount of code covered [1, 4]. This form of testing allows for the developer to avoid redundant tests and code.

```
int function(int x) {
  if(x > 3) {
    x++;
  }
  if(x > 5) {
    x++;
  }
  return x;
}
```

Figure 1: Sample code illustrating how statement coverage can be applied

Therefore, complete statement coverage is one of the most simple and fundamental criterion to follow for building test cases. Yet, it is often overlooked by even professional developers. Achieving higher statement coverage is directly correlated with the likelihood of identifying more defects and increasing the dependability of software [5, 6, 7]. Consider the code in Figure 1, by applying the principles of statement coverage, inputting integer values greater than five as the parameter of the function would cover more lines than integer values less than five. Using this approach, it would be more efficient for the tester because there are fewer test cases to write and review. Simultaneously, it is also more efficient for the program because there are fewer test cases to run.

Even though the importance of statement coverage is emphasized, the lack of enthusiasm that students have when learning the subject greatly diminishes what they actually learn [10]. Therefore, researchers have turned towards gamification as a potential countermeasure [4]. Gamification can be defined as the application of gaming elements into other areas of activity. From previous studies on specific elements in gamification, it had been concluded that the most frequent and effective components in the context of education were a leader board system, level progressions, a point/scoring system, and animations/avatars [2, 3]. According to Khaleel and Sahari, these elements of gamification were proven to have positive effects on a student's motivation, learning skills, and amusement [9].

Taking these studies into account, a reliable, gamified version of statement coverage has yet to be created as a learning supplement. However, with the compelling attributes of gamification, we can directly address the problems surrounding software testing education. Through a new learning environment, we hope that students will feel more engaged and motivated. Then if the gamification process of statement coverage education is successful, we will expand it to more principles in software testing.

2 Proposed Solution

In this section, we will illustrate how gamification features can keep players engaged and entertained while they learn about statement coverage.

In order to do so, we will create a game that teaches statement coverage. The game will display an executable function with parameters, where the goal is to execute all lines of code within the method. To

achieve this goal, the player will have as many attempts as they need to decide what argument(s) will execute new lines of code and the functionality to pass in those arguments will be provided. Referring to the code in Figure 2, if the player were to pass in the integer four, the program would print stars next to the lines executed as seen in Figure 3.

```
int function(int x) {
  if(x > 3) {
    x++;
  }
  if(x < 1) {
    x--;
  }
  return x;
}
```

Figure 2: Sample level of the game before any attempts

```
int function(int x) {
* if(x > 3) {
*   x++;
* }
* if(x < 1) {
    x--;
  }
* return x;
}
```

Figure 3: Sample level displaying how the game could look after one attempt

After each attempt, if all the lines of code have not been executed, the player will have another attempt at choosing an argument(s) for the method to run. This process will continue until all the lines of code in the given method are executed, at which point every line will have stars next to them, concluding the level. This program will serve as our basic template for learning statement coverage without gamification elements.

After building the basic structure of our game, we will be developing gamification elements that will make the game more enjoyable and engaging. Some examples we hope to implement is a scoring system, a leader board, and awards/achievements. We hypothesize that these gamification elements will further increase the level of engagement and motivation the game can provide, which will help students learn about statement coverage.

In the gamified version, after a level finishes, a score will be given based on the number of attempts taken. Players who solve the level in fewer attempts will receive higher scores than those with more attempts. Using this scoring system, players will be incentivized to execute the lines of code in the least amount of attempts, which correlates back to the principles of statement coverage.

Our version of the leader board will be determined by the final score of each player with the highest scoring players being presented on the leader board for each specific level. We anticipate that the player will be motivated to perform better after seeing their place on the leader board.

Furthermore, awards/achievements will be granted after the players have completed certain tasks within the game. Much like the studies aforementioned, we speculate that our awards/achievements system will also make players more engaged and inspired to learn more about the respective topic.

A potential improvement is to create a graphical interface through the Unity game engine that will allow users to immerse themselves in a better gaming environment. The user experience is an important factor in creating a game and is crucial in producing an engaging game. Previous works have shown that merely adding levels and leader boards were insufficient in giving the students an enjoyable experience. In fact, exaggerated validation in the form of visuals and sounds has immense effects in making the player feel involved [8].

3 Evaluation and Implementation Plan

3.1 Evaluation Plan

To evaluate whether gamification is effective within statement coverage education, we will conduct an experiment that focuses on the player's enjoyment, understanding, and engagement. Within this experiment, we will have two separate groups: one that plays our game without any of the gamification elements (control group) and one with the gamification elements (experimental group). The control group will play a version

of our program that does not include the option to: see their score, compete against a leader board, or other features that will be added in. The experimental group will have access to all those components of the game. Each group will be asked to complete the same nine levels of the game, but the last three levels will be optional. Consequently, we will be able to determine their level of enjoyment by observing which players decided to play the additional optional levels. However, our prime method of measuring the enjoyment, understanding, and engagement of the player will be by giving them a survey before and after they play the game.

Due to that fact that our results depend on how much the players will learn, we will be specifically targeting novice programmers who do not understand statement coverage because they will have the most room for growth and improvement. Additionally, to minimize the possibility of confounding variables such as race, age, and gender, the participants chosen will have a diverse set of backgrounds.

After the participants are selected, they will be brought into a private computer lab for the experiment. Then, the player will fill out a pre-game survey through a Google form. The survey will ask for the player's demographics such as gender, age, ethnicity, currently enrolled CSE courses, etc. In terms of logistics, the survey will also ask for their experiences and prior knowledge of statement coverage. These responses will serve as our base data, enabling us to measure the growth of each participant after they play the game. Once they finish the game, we will provide another Google form, asking the user for general feedback and their levels of engagement, enjoyment, and understanding of statement coverage. These surveys provide us with crucial data to assess the validity of our experiment.

Based on the optional levels and responses of the surveys, if the experimental group retains more players for the optional levels and has positive feedback in the surveys, then we can determine that the gamification features were successful. However, if the results between the two groups do not significantly vary, then it can be deduced that gamification process did not work in the context of statement coverage education.

While our research is limited to the gamification of statement coverage within software testing, it will be able to open the doors to the gamification of other software testing elements. The gamification of software testing education is relatively new, but future research will be able to build on our results to find more efficient gamification techniques.

3.2 Timeline

Even though an implementation of a Unity-based GUI would help us compare a gamified version of the game to a non-gamified design more clearly, the time commitment required to do so precludes us from pursuing a GUI. Therefore, our timeline revolves around objectives that are absolutely necessary to perform the experiment. The most efficient way to test our hypothesis is to focus on a text-based game on the terminal and to implement the most basic gamification features. Thus far, we have developed three out of our projected nine levels with gamified elements of a scoring system and a leader board. Also, pre-game and post-game surveys have been created to gather crucial feedback. We do not need more than nine levels because we will be focusing on applying, testing, and improving our gamification features throughout the remainder of the year, outlined in our timeline below.

Fall Quarter 2019

Weeks 7-10
1. Read papers on gamification
2. Read papers on statement coverage
3. Consider gamification features to integrate
4. Implement the scoring features to game
5. Create 3 levels of the game
6. Create potential pre-game/post-game surveys.

Winter Quarter 2019

Weeks 1-4
1. Record surveys from beta testers and adjust game accordingly
2. Implement the multiplayer feature to game
3. Create 3 additional levels of the game of medium difficulty

Weeks 4-7
1. Brainstorm level designs that could additionally test branch coverage
2. Implement 3 levels of harder difficulty
3. Settle on a name for the game
4. Start implementing graphical features

Weeks 7-10
1. Continue implementing graphical features
2. Continue to enhance aspects of game with responses from testers
3. Improve on version of the game without gamification features

Spring Quarter 2019

Weeks 1-3
1. Finalize game without gamification features
2. Start working on research poster

Weeks 3-5
1. Continue to adapt game to user feedback
2. Start experimenting and surveying user experiences of the game with/without gamification features
3. Develop poster related to research

Weeks 5-7
1. Continue to survey differences between game with/without gamification features
2. Make final revisions based off user feedback
3. Develop poster related to research

Weeks 7-10
1. Review data gathered to finalize results, conclude experiment, and brainstorm potential future research.
2. Finish summarizing research poster.

References

[1] J. Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proceedings of the 12th International Conference on Testing Computer Software*, pp. 111–123, Citeseer, 1995.

[2] M. V. Mäntylä and K. Smolander, "Gamification of software testing-an mlr," in *International Conference on Product-Focused Software Process Improvement*, pp. 611–614, Springer, 2016.

[3] M. R. de Almeida Souza, K. F. Constantino, L. F. Veado, and E. M. L. Figueiredo, "Gamification in software engineering education: An empirical study," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, pp. 276–284, IEEE, 2017.

[4] B. S. Clegg, J. M. Rojas, and G. Fraser, "Teaching software testing concepts using a mutation testing game," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, pp. 33–36, IEEE, 2017.

[5] X. Cai and M. R. Lyu, "The effect of code coverage on fault detection under different testing profiles," in *ACM SIGSOFT software engineering notes*, vol. 30, pp. 1–7, ACM, 2005.

[6] M.-H. Chen, M. R. Lyu, and W. E. Wong, "An empirical study of the correlation between code coverage and reliability estimation," in *Proceedings of the 3rd International Software Metrics Symposium*, pp. 133–141, IEEE, 1996.

[7] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, pp. 57–68, ACM, 2009.

[8] E. D. Mekler, F. Brühlmann, A. N. Tuch, and K. Opwis, "Towards understanding the effects of individual gamification elements on intrinsic motivation and performance," *Computers in Human Behavior*, vol. 71, pp. 525–534, 2017.

[9] F. L. Khaleel, N. Sahari, T. S. M. T. Wook, A. Ismail, *et al.*, "Gamification elements for learning applications," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 6, no. 6, pp. 868–874, 2016.

[10] A. Deak, T. Stålhane, and D. Cruzes, "Factors influencing the choice of a career in software testing among norwegian students," *Software Engineering*, p. 796, 2013.

Revision Notes

1. Switched out references utilized for Research Context and Problem Statement. We were unable to explain how we build upon old references. With new references, we can connect to our research.

2. Changed the organization of Research Context and Problem Statement so that it's easier to follow and find the main problem.

3. Added citations for new references.

4. In Research Context and Problem Statement, defined statement coverage and gamification more clearly. Also, we connect the two topics to our research better.

5. Revised proposed solution entirely.

6. Added and defined Agile Software Development and Single Responsibility Principle to Proposed Solution.

7. Added the programming language we plan on using and the types of functions we are going to use to Proposed Solution.

8. Added who we were going to test and how we are going to test them to our Evaluation.

9. Removed GUI from timeline.

10. Changed total levels from 30 to 12

11. Removed all iterations of a stand-alone "this"

12. Explained how our surveys help us make changes and provided an example

13. Changed all references to BibTex

14. Added more references (6-10)