Teaching Software Testing with a Mutation Testing Game

José Miguel Rojas and Gordon Fraser
Department of Computer Science, The University of Sheffield, UK
{j.rojas,gordon.fraser}@sheffield.ac.uk

Abstract
Software testing is crucially important in a world dominated by software. Software testing is also inherently difficult and requires theoretical expertise and practical experience. However, standard testing techniques are often perceived as boring and difficult, and thus do not feature as prominently in programming education as they maybe should. In order to address this problem, we aim to make testing education more interesting with gamification. The Code Defenders game uses gameplay elements to engage students in the testing process in a competitive and fun way. Our hope is that if students perceive writing tests as a fun activity, they will later become better software engineers, with better testing skills, and with more inclination to apply thorough testing. In this work-in-progress paper we describe our initial experiences with Code Defenders, as well as open challenges on the way to making testing education fun.

1. Introduction
It is common wisdom that software needs to be thoroughly tested: Insufficient testing is known to cause project failures, large economical damage, and can threaten people's lives. Many large and successful software companies have made testing an integral part of the day-to-day jobs of their developers. However, this does not generally hold, and often software developers are not well educated to actually do testing. Indeed there are reports that testing plays a much less prominent role in the daily activities of software developers than one would hope it to do (Beller et al., 2015).

One possible root cause for this lies in how software developers are educated: Programming courses in higher education and outside it generally focus more on the creative aspects of coding, rather than the analytical process of testing. While some basic testing is usually taught, education rarely manages to get people "test infected"; that is, it does not convey testing as an enjoyable task (Barriocanal et al., 2002; Patterson et al., 2003). At face value, this is not surprising, as systematic testing is a comparatively boring task in the face of more exciting alternatives, such as writing new code.

We aim to make testing education more interesting by applying gamification principles. We want to engage students with testing by using the human competitive nature as incentive, and by mapping testing tasks to gameplay components. We hope that if students perceive writing tests as a fun activity, they will become better software engineers, with better testing skills and more inclined to apply thorough testing.

As a first step towards this goal, we have developed Code Defenders (Rojas & Fraser, 2015), a game that makes use of a testing technique known as *mutation testing*. Players of the game can participate in one of two roles: As attackers, they aim to create *mutants*, which are subtle modifications of the program under test. The more difficult it is to reveal a behavioural difference between the original program and a mutant, the better the mutant. As a defender, players aim to write tests that reveal these behavioural differences, i.e., *kill*, these mutants. Both roles encourage and train an understanding of software bugs and how they are detected by tests, and how to create good tests.

Initial studies revealed that students engage willingly with testing using Code Defenders. However, there remain numerous challenges on the way to making Code Defenders a viable approach to teach testing.

2. The Code Defenders Game
Code Defenders uses elements of *mutation testing* to gamify software testing. This section therefore introduces mutation testing before going into details on the gameplay.

2.1. Mutation Testing
Mutation testing is a software engineering technique aimed at guiding the development of test code and at evaluating its quality (DeMillo et al., 1978). Given a program under test, mutation testing involves the automated creation of a set of variants of the program, called mutants, which differ from the original program by small syntactical changes. The underlying premises are that software developers tend to write almost-correct programs (known as the *competent programmer hypothesis*) and that small syntactical changes can be representative of complex real-world program faults (known as the *coupling effect*). The resulting set of mutants can be used to quantify the quality of existing tests by measuring how many of the mutants are detected: A test is said to *kill* a mutant when the outcome of executing it on the original program is different from the outcome of its execution on the mutant. Mutants that are not killed can guide further test generation efforts. It is possible for a mutant to be semantically equivalent to the original program, in which case there exists no test that could distinguish it from the original program. Detecting equivalent mutants is an undecidable problem and typically requires human intelligence.

2.2. Gameplay
Code Defenders implements a gamification approach to mutation testing. The main components of the technique are built into an intuitive gameplay that allows players to produce mutants, develop tests and reason about mutant equivalence.

In its current version, Code Defenders is a two-player game, and players use the Java language and the JUnit testing framework. Each player takes the role of either the attacker or the defender. The goal of the attacker is to create subtle mutants of a class under test that are hard to kill by editing the source code of the class under test. The goal of the defender is to create strong JUnit tests that kill the mutants created by the attacker. The game is played in rounds of attack and defence, where the attacker takes the first turn of play. Once the attacker has submitted a mutant, the turn is passed to the defender, who then tries to create a test with the right assertions to detect the mutants. A round is completed when the attacker has submitted a new mutant and the defender has countered with a new test. The game then proceeds until the number of rounds chosen when creating the game have been played. Upon completion of each round, a mutation analysis is performed to determine whether the newly created test has killed any live mutants. A special scenario arises when the defender suspects a mutant to be equivalent: Instead of providing a new test, the defender can trigger an equivalence duel, challenging the attacker either to accept that the mutant is equivalent, or to produce a killing test to prove that it is not equivalent.

The point scoring system of Code Defenders is intended to encourage attackers and defenders to produce good quality mutants and tests, and to decide the winner of a game. A mutant scores points for the attacker when it survives long in the game. The more rounds it survives, the more points the attacker scores. On the other hand, a test scores as many points for the defender as mutants it kills. In the case of equivalence duels, the attacker can score extra points by submitting a killing test for the mutant claimed equivalent, but can lose the points scored by that mutant otherwise. More details on the design, implementation and the point scoring system of Code Defenders can be found in an earlier publication on the gamified mutation testing system (Rojas & Fraser, 2015).

3. Initial Findings
We performed an initial evaluation using a controlled study with undergraduate and graduate students from the University of Sheffield. In this section, we present some initial findings from this experiment.

3.1. Evaluation Study
The study consisted of two 30-minute sessions in a computer room, where 38 participants were asked (using a random task assignment) to write tests manually for two small, yet challenging Java classes or to play Code Defenders on them—either attacking or defending. In each session, one third of participants wrote tests manually and two thirds played the game. All participants used the two different classes.

A training phase preceded the two main sessions of the experiment. As part of the training, a brief tutorial on unit testing and mutation testing was presented, followed by an introduction to Code Defenders.

Through short, intuitive tasks, the participants were guided to familiarise themselves with the main interfaces of the game. Furthermore, to conclude the training phase, all participants played an actual Code Defenders game on a simple example class.

In total, 32 games were played and 26 manual testing tasks were completed. On average, 3.8 rounds were played on each game, leading to 121 mutants and unit tests for two Java classes used as subjects. On the other hand, the manual testing tasks resulted in 145 unit tests. While a quality evaluation of these tests is necessary to fully assess the comparison, these preliminary results suggest that the gamification approach implemented by Code Defenders was able to engage students in the task of writing unit tests.

We are currently working on the analysis of the mutants and tests produced during the study. The research questions we are interested in answering through the study involve (a) the kind of mutants developers create on Code Defenders and how they compare with mutants created using state-of-the-art mutation operators, (b) whether using Code Defenders leads to stronger tests than manual testing, and (c) how effective developers are at killing mutants and at detecting equivalent ones.

3.2. Student Feedback

At the end of the study, the students were asked to fill in an exit survey reflecting on their experience with Code Defenders. The survey covered demographics and addressed qualitative aspects of the study from the participants' perspective.

52% of the participants were undergraduate students, 37% were Master's or PhD students and the rest were either professional developers or academics. All participants are in Computer Science or Software Engineering-related fields, have a diverse degree of experience programming in Java but generally little to no industrial work experience (66%). The majority (76%) had used JUnit or a similar testing framework before and understood well or very well the concept and usage of mutation testing, although most admitted to only rarely or occasionally writing unit tests when programming.

When asked about their experience with Code Defenders, although a small group had difficulties understanding the game (6), all participants claimed having enjoyed playing it and a vast majority (92%) agreed that writing unit tests on Code Defenders is more fun than writing unit tests while coding. Interestingly, we observe a strong tendency that creating mutants is more enjoyable than creating tests. Whereas these responses indicate that the testing task is more engaging for participants when it is performed in a gamified scenario, supported by most participants acknowledging that they would even play the game for fun in the future, there is still some disagreement about whether writing tests in a professional IDE would have led participants to producing stronger tests, possibly due to the sometimes long idle time when waiting for opponents to play their turns.

The ways in which Code Defenders could be improved were also addressed in the participants survey. A list of suggested features was presented to participants, who were asked to indicate whether they thought each suggested feature was important, just nice to have or rather irrelevant. Incorporating the notion of code coverage by highlighting the code covered by each test seems to be the most important aspect to improve in the Code Defenders. Almost as important seems to be the creation of a single-player mode that would avoid having to wait for an opponent to join the game and play turns promptly. Improving the capabilities of the code editor, a multi-player mode and support for other programming languages are, to a less degree, also important and nice to have features from the participants' perspective.

To capture more aspects that were not covered explicitly in the survey, participants also had the opportunity to suggest improvements for the game in two categories we consider crucial: user interface and point scoring system. Besides some editor capabilities, there were several comments on the point scoring system: The way in which Code Defenders rewards players was determined in an ad-hoc manner with two purposes in mind: fairness and engagement. Some interesting suggestions involve a) making the scoring system time-dependent, i.e., a test scores less points if it took longer to write it; b) different number of points depending on the complexity of the mutants; and c) allowing players to gamble on the number of points to be scored by making trials before actually submitting mutants or tests. Some participants also mentioned the need for a leaderboard to compare themselves to other players.

Participants were finally given the opportunity to make general suggestions to improve Code Defenders. One of them is particularly humbling and actionable: “Fix bugs”. Others involve features we are planning to explore in the near future, such as (a) limiting the time available per turn, (b) allowing defenders to run a new test on the original code before submission, and (c) improvement of the equivalence challenge protocol in the hard mode, given that it is a bold move at the moment to claim a mutant is equivalent when its code is not revealed to the defender.

4. Open Challenges

We envision that Code Defenders can be useful in at least two contexts: Testing education, and crowd-sourcing test development. In both scenarios, the human factor will play a crucial role in the potential success of the approach. Although our preliminary experience with Code Defenders suggests developers enjoy playing the game and also prefer the gamification approach over manual testing, several open questions and challenges remain:

- Breaking code (i.e., creating mutants) tends to be seen as a more fun task than defending it (i.e., writing tests). How to achieve similar motivation levels for both attackers and defenders?
- How to overcome the long waiting times in between turns? Would alternative mechanisms to the current round-based, such as an asynchronous gameplay for example, be more suitable?
- Our initial empirical evaluation involved two relatively simple classes with no dependencies beyond standard data structures. Will the game continue to be practical and enjoyable when used on more complex, real-world programs?
- Code Defenders is currently a two-player game only, but we are working on a multi-player mode that will allow several attackers and several defenders to play the same game. This poses several challenges to the current design of the game. Will competitiveness among sides (attackers vs. defenders) and teamwork within sides complement or conflict?
- Code Defenders can serve as an educational resource to support instructors in delivering theoretical software testing concepts such as fault-detection effectiveness or code coverage, and it could also provide meaningful practical experience to the students through one-on-one or multi-player games in the classroom or even as a self-tutoring tool in a single-player mode. However, the actual educational value of these alternatives need still to be assessed in the field, and leads to follow-up questions such as whether Code Defenders can be used for assessment and how to prevent players from cheating the game in the classroom.

5. References

Barriocanal, E. G., Urbán, M.-Á. S., Cuevas, I. A., & Pérez, P. D. (2002). An experience in integrating automated unit testing practices in an introductory programming course. *ACM SIGCSE Bulletin*, *34*(4), 125–128.

Beller, M., Gousios, G., Panichella, A., & Zaidman, A. (2015). When, how, and why developers (do not) test in their IDEs. In *Proceedings of the 10th joint meeting on Foundations of Software Engineering (FSE)* (pp. 179–190).

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978, April). Hints on test data selection: Help for the practicing programmer. *Computer*, *11*(4), 34–41. doi: 10.1109/C-M.1978.218136

Patterson, A., Kölling, M., & Rosenberg, J. (2003). Introducing unit testing with BlueJ. *ACM SIGCSE Bulletin*, *35*(3), 11–15.

Rojas, J. M., & Fraser, G. (2015). Code Defenders: A Mutation Testing Game. In *The 11th international workshop on mutation analysis.* IEEE. (To appear)