Teaching Mutation Testing using Gamification

José Miguel Rojas and Gordon Fraser

Department of Computer Science, The University of Sheffield, United Kingdom
{j.rojas,gordon.fraser}@sheffield.ac.uk

Abstract. Software quality and testing are at the heart of software engineering, but they are not always at the heart of software engineering education. As a consequence, advanced techniques such as *mutation testing* are often neglected and do not become part of the standard repertoire of a graduate software engineer. We propose the use of gamification to teach mutation testing and to strengthen testing skills. We introduce Code Defenders, a mutation testing game, which can assist educators in delivering complex mutation testing concepts and is intended to make the learning experience more enjoyable and fruitful for students.

1 Introduction

It is essential that software is thoroughly tested, but testing is a challenging and inherently error-prone activity, and it is generally perceived as less fun than creative activities such as programming [4]. This holds for practitioners as well as in higher education, where testing maybe does not receive the attention it should. This lack of engagement with testing is a contributing factor to the lack of adoption of advanced testing techniques such as *mutation testing*, despite its potential to improve fault-detection effectiveness of testing activities. Mutation testing intrinsically relies on hard to solve computational problems and its application in practice requires skills that currently are not provided by software engineering education. Besides cursory treatment in dedicated testing books, we are aware of only one educational module proposed by Barbosa and Maldonado [1].

There is evidence to support the idea that *gamification*, i.e., the use of game playing elements (competition with other players, fantasy scenarios, game rules, point scoring, etc.) to make difficult tasks less tedious and easier to grasp, can have a positive impact in education [6]. Following this insight, we introduce Code Defenders [8], a novel web-based game that implements a gamification approach to mutation testing, in which the core concepts of mutation testing are mapped to simple game elements.

In this paper, we explore the use of gamification to teach mutation testing concepts to software engineering and computer science students. We discuss the educational aspects involved in the Code Defenders game and propose its incorporation into programming and testing courses. Our research hypothesis is that through the use of a mutation testing game, students will be able to grasp all relevant mutation testing concepts while having fun, and in the end become better software developers and testers, who produce higher quality software.

2 Background

2.1 Mutation Testing

Mutation Testing [2,3] simulates software faults in a program in order to measure the fault detection ability of an existing test suite. A number of modified versions of a program are produced by systematically performing small changes in the original code. Each modified version is called a *mutant* and corresponds to the application of one particular pre-defined *mutation operator*. A mutation operator characterises a type of change that can be applied to a program to mimic typical syntactic errors programmers make. Examples of basic mutation operators are *statement deletion* (e.g., `while(x<n){}` instead of `while(x<n){x++;}`) or *arithmetic operator replacement* (`x--;` instead of `x++;`).

The mutation operators simulate only a subset of all the faults a programmer could possibly make; the hope that this is sufficient is based on two hypotheses or empirical principles [3]: Programmers tend to write correct or almost correct programs, and most software faults experienced programmers introduce are due to small syntactic errors – this is known as the *competent programmer* hypothesis. This validates the idea of focusing testing on detecting these kinds of faults, because a test suite that can distinguish all programs differing from a correct one by only simple errors is so sensitive that it also implicitly distinguishes more complex errors — known as the *coupling effect*.

The process of mutation testing continues with the execution of all mutants against the existing test suite. If the execution of a test on the original program and its execution on a mutant differ, the mutant is said to have been *killed* by the test. Alternatively, if no test is able to detect a mutant, the mutant is said to have *survived*. When a mutant survives, it can be because the test suite is insufficient and hence more tests are needed, or because the mutant is equivalent. An *equivalent mutant*, although syntactically different, is semantically identical to the original program. Determining equivalence is an undecidable problem in general and therefore human effort is needed to distinguish equivalent mutants from mutants that are just hard to kill (*stubborn* mutant, in mutation testing jargon). As a result of this process, a *mutation score* is computed as the ratio of mutants killed to non-equivalent mutants, and is used to estimate the fault-detection ability of the existing test suite. Testers can then improve their test suites by adding new tests that kill the surviving mutants.

A visual overview of all the concepts involved in mutation testing in the form of a conceptual model is presented by Barbosa and Maldonado [1].

2.2 The Code Defenders Mutation Testing Game

Code Defenders (`http://code-defenders.dcs.shef.ac.uk`) is a web-based game that implements an approach to gamify mutation testing. The main notions of the technique are built into the gameplay of the game: A unit under test (a Java class) takes a central role, and players take the roles of attackers and defenders. The attackers' goal is to create subtle mutants of the unit under test which are hard to detect. The defenders' goal, in turn, is to produce strong tests which kill the attacker's mutants and can serve as an effective test suite for the unit under test. Following standard mutation analysis, Code Defenders considers a mutant killed when execution of a test on the original unit under test differs from its execution on the mutant –more specifically, when a test passes on the original unit under test and fails or ends with an error on the mutant– hence detecting the defect represented by the mutant.

Code Defenders is played in rounds of attack and defence. Points are awarded to attackers according to how many rounds their mutants survive, and to defenders according to how good their tests are at killing mutants. Equivalent mutants constitute a special component of the gameplay. If the defender suspects a particular mutant is equivalent, an equivalence duel can be triggered. The attacker is then challenged either to accept that the mutant is equivalent – giving away points – or to prove it is not by submitting a test that kills it – scoring extra points. For more details, we refer the reader to a separate publication about the gameplay [8].

3 Code Defenders as a Learning Environment

Using Code Defenders in an educational scenario was part of the motivation to develop the game in the first place [8] (while another motivation lies in the use of gamification to overcome some of the hard computational problems mutation testing poses). In this section, we discuss the use of the game as an educational tool. In designing this integration, we seek to achieve an increase in engagement and motivation among students, as well as in their understanding of the contents and the development of relevant practical skills.

Due to their nature of information items, the formal definition, history, underlying principles (competent programmer hypothesis and coupling effects) and potential application domains of mutation testing need to be presented in a traditional fashion. After that, when the actual process of mutation testing is described, the use of Code Defenders can be incorporated.

3.1 Conceptual Education

Mutation testing core concepts introduced in Section 2.1 are naturally embodied into the storyline of Code Defenders. The task of an attacker is expressly to create good mutants, whereas the task of a defender is to create good tests. Both players are constantly informed about the state of all mutants in the game, whether they are alive, have been killed, or have been identified as equivalent. In that sense, educators can interleave theory material introducing these concepts with the use of the game to exemplify them.

It is a challenging task to contextualise the use of any tool to arbitrary groups of students. The problem is even harder if one intends to accommodate to different skill sets and learning strategies students may have or different interaction levels in which students will feel confortable. The design of Code Defenders allows to create educational material to provide engaging and meaningful practical experience suitable for diverse groups of students.

3.2 Practical Experience

Puzzles: Creating testing puzzles in Code Defenders is straightforward. The educator needs to create a new game and seed it with mutants or tests, depending on whether the intended task for the students is to write tests or to create mutants, respectively. Puzzles can complement the delivery of new concepts to students well. For example, before introducing the notion of mutant equivalence, a puzzle can be presented to students in which an equivalent mutant is seeded and students are asked to create a test to kill it, but without revealing its equivalence. The expected outcome is that students will be able to realise the impossibility of the task without even being familiar with the formal concept.

Duels: This is the most natural kind of practice available in Code Defenders. It amounts to creating a game and asking pairs of students to play against one another, each taking the role of the attacker or the defender. Different programming skill levels can be accommodated by using target classes of different complexities and either the *easy* or *hard* level of the game. In the easy level, mutants are fully disclosed to the defenders, whereas in the hard level, defenders must infer the nature of each mutation based on its location in the code.

Self-tutoring: In order to complement classroom education, Code Defenders can be used as a self-tutoring system in which students, at their own pace, can further enhance their testing skills. To achieve this, we plan to design a set of examples, integrate existing automated mutation testing (e.g., Major [7]) and test generation tools (e.g., EvoSuite [5]) to act as automated players in the game, develop a script of all possible game actions for each example, and add extra help components to guide students in the process.

Rewards: Code Defenders implements a simple scoring system that rewards players in terms of the quality of their mutants and tests. As most gamification approaches, this scoring system is meant to leverage the competitive nature of humans to maintain good levels of interest and commitment among students. Furthermore, the Code Defenders leaderboard can help as a rewarding mechanism for all students in the class. Teamwork can also be accommodated for in the game by having students play as teams of attackers or defenders.

3.3 Assessment

Code Defenders can easily be extended to serve as evaluation framework. A class under test can be uploaded alongside a list of pre-defined set of mutants and tests. These are then used as a game script, similar to puzzles or the self-tutoring mode, but for which students performance is tracked. Educators can use the collected data as feedback on the strengths and weaknesses of their students.

4 Conclusions and Future Work

In this paper, we have laid out a plan for using gamification of mutation testing as an educational resource in programming and testing courses. We have discussed how Code Defenders can be used as an instructional tool to engage students in testing tasks and foster a deeper, practical understanding of fault-detection effectiveness. Furthermore, the design of the game also lends itself well to support delivering theory lectures on mutation testing and to inform students evaluation.

The ideas discussed in this paper will be evaluated empirically in future work. As a pilot study, we plan to integrate the system into software engineering modules taught in the Department of Computer Science at The University of Sheffield. The effects of using Code Defenders to teach mutation testing can be measured by comparing course grades, lecture attendance and session evaluations. We expect experimental results will support our vision that through gamification students can develop the required skills to apply mutation testing in practice and use it as a tool for building higher quality software.

Besides the aforementioned intended impact on traditional education, the long-term goal of this work is more global: to change the way mutation testing is taught in general. Not only can Code Defenders serve as an educational tool in traditional education, but it can also be useful in online education, where a number of programming and testing courses exists (e.g., on Coursera, MIT Open Courseware, Udacity) but practical educational tools are rarely available.

Ultimately, the purpose of teaching mutation testing is to equip learners with a powerful tool to support the development of high quality software artefacts. Being able to identify good mutants and write good tests can directly influence developers' performance during software development. This is measurable by empirical studies in which students are assigned implementation, maintenance or testing tasks.

Code Defenders is available for playing online at:

`http://code-defenders.dcs.shef.ac.uk`

References

1. Eduardo F Barbosa and Jose C Maldonado. Establishing a mutation testing educational module based on IMA-CID. In *Second Workshop on Mutation Analysis*, pages 14–14. IEEE, 2006.
2. Timothy Alan Budd. *Mutation Analysis of Program Test Data.* PhD thesis, Yale University, New Haven, CT, USA, 1980.
3. R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, April 1978.
4. Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. Bug hunt: Making early software testing lessons engaging and affordable. In *ACM/IEEE Int. Conference on Software Engineering (ICSE)*, pages 688–697, 2007.
5. Gordon Fraser and Andrea Arcuri. EvoSuite: automatic test suite generation for object-oriented software. In *ACM Symposium on the Foundations of Software Engineering (FSE)*, pages 416–419, 2011.
6. J. Hamari, J. Koivisto, and H. Sarsa. Does gamification work? – a literature review of empirical studies on gamification. In *47th Hawaii International Conference on System Sciences (HICSS)*, pages 3025–3034, Jan 2014.
7. René Just. The Major mutation framework: Efficient and scalable mutation analysis for Java. In *ACM Int. Symposium on Software Testing and Analysis (ISSTA)*, pages 433–436, 2014.
8. José Miguel Rojas and Gordon Fraser. Code Defenders: A Mutation Testing Game. In *The 11th International Workshop on Mutation Analysis*. IEEE, 2015. To appear.