Testing Robot Challenge: A Serious Game for Testing Learning

Anna Rita Fasolino
University of Naples Federico II
Naples, Italy
fasolino@unina.it

Caterina Maria Accetto
University of Naples Federico II
Naples, Italy
c.accetto@studenti.unina.it

Porfirio Tramontana
University of Naples Federico II
Naples, Italy
ptramont@unina.it

Abstract

Software testing education is becoming increasingly important both in academia and industry. Despite efforts to improve teaching approaches at the university level, many challenges persist for better preparing students for their future careers. In this position paper we present the Testing Robot Challenge tool implementing a serious game designed for motivating the students to practice testing and learn how to write effective unit tests in coverage testing. The game exploits the mechanism of the challenge that students can play against state-of-the-art tools for automated test case generation. It is configurable by teachers, in order to tune the complexity and type of challenges to the specific needs of the students and to the objectives of the course taught. To validate the tool, we performed a preliminary experiment involving 15 students of a Software Engineering course who provided generally positive feedback about it and useful comments for its future improvement.

CCS Concepts

• **Applied computing** → **Interactive learning environments**.

Keywords

Software Testing Education, Automated Testing, Gamification of Learning Activities, Serious Games

ACM Reference Format:
Anna Rita Fasolino, Caterina Maria Accetto, and Porfirio Tramontana. 2024. Testing Robot Challenge: A Serious Game for Testing Learning. In *Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation (Gamify '24), September 17, 2024, Vienna, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3678869.3685686

1 Introduction

Software testing education is indispensable in software development, yet often overlooked, contributing to a shortage of expertise in the software industry. Becoming an experienced software tester requires both understanding many strategies for writing high-quality test cases and a significant amount of practice [14].

Despite the well-known importance of software testing and the recommendation to integrate software testing into Computer Science and Software Engineering curricula as part of the educational experience [10], doing practice of software testing and test case design is not so frequent in software testing courses. In a recent academic course mapping study involving 22 software testing courses and 97 courses also including testing topics from 4 European countries, the authors found that structure-based testing is taught in all the analysed software testing courses (100%) but the laboratory activities devoted to the practice of software testing are very limited, usually due to scarce resources of time that are available to the teachers. [15]. Even when the academic degrees devote more time resources to software testing, there is the problem of addressing the students' low motivation to do testing practice [6]. Students generally do not derive a great deal of satisfaction from exposing flaws in their own programs, they perceive testing as not important, boring and repetitive, and do not acquire the practice and experience that software testing deserves. In addition, the traditional "passive learning" pedagogical approaches used to teach testing are not adequate to make students more motivated to write unit tests as they code.

To increase the motivation of students in doing practice of test case design, several strategies and approaches have been described [8]. One of the proposed solutions consists of using a tool during the testing learning experience, including tools providing a learning environment that guides students through a learning path [3] and interactively [13], tools aiming to support students in practicing testing in laboratory settings [16] , and tools that exploit a gamification approach to better motivate students [9], [5], [12].

Gamification is widely recognized as "the use of game design elements in non-game contexts" [2]. Gamification uses the philosophy, elements, and mechanics of game design in non-game environments to induce certain behavior in people, as well as to improve their motivation and engagement in a particular task [7]. Although several games are available to support testing education [17], many of them present the issue of not being easily integrable by teachers in the courses they teach. They may not perfectly fit the specific learning goals of the courses and the learning needs of the students [11]. Hence there is the need for more tools configurable by the teachers, in order to provide an attractive and accessible game experience to the students, irrespective of their testing skills and learning needs.

In this position paper we present the Testing Robot Challenge, a tool that we designed to fill in this gap. The tool implements a serious game for motivating the students to learn how to write effective JUnit test cases in coverage testing. The game exploits the mechanism of the challenge that students can play against an automated test generator. The tool is configurable by teachers, in order to tune the complexity and type of challenges to the specific needs of the students and the objectives of the course taught. A first prototype implementation of the tool has been developed in the context of the ENACTEST research project[1]. A case study has

[1] https://enactest-project.eu/

Gamify '24, September 17, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1113-8/24/09
https://doi.org/10.1145/3678869.3685686

been carried out to explore and preliminary validate the learning approach supported by the tool. The study has been performed in the context of a Software Engineering course offered in a Bachelor Degree in Computer Engineering at the University of Naples Federico II. Its results were encouraging and showed us several directions for future improvements. The remainder of the paper is organized as follows. Section 2 presents the tool features and Section 3 illustrates how we implemented it. Section 4 describes the case study we executed, while Section 5 reports final conclusions and future works.

2 The Testing Robot Challenge Tool

The Testing Robot Challenge tool has the aim to convoy several software testing topics (e.g., coverage testing, unit testing, JUnit framework, automated test generators, ..) by a serious game where the learners can experiment the listed topics in practice, by a gameful engaging experience. The serious game has been defined according to the framework proposed in [7] and presents goals, mechanics (the basic actions and control mechanisms that are necessary to gamify the interaction of practitioners with tools), evaluable actions (an action or condition related to players which can be evaluated in the game to assess if an achievement can be obtained as a result of it), a Rule Engine (to evaluate the actions), etc.

The tool has been designed as a Web application that provides the students the possibility to play challenges (a kind of missions) regarding the test case development activity using the JUnit framework. The challenge is played against an automated generator of test cases, named Robot. The aim of the game is to overcome the Robot in terms of the code coverage reached by the developed test cases. When the player wins the challenge, he/she is assigned a score. The game maintains a leaderboard of players showing how many challenges they won and the score they earned. The best performing players will be assigned rewards.

There are several types of challenge that can be chosen by the player once he has registered in the app. In the first type of challenge, named *Robot Fight*, the player can fight against a single Robot. To run this challenge the tool presents a code editing environment where the player can write JUnit test methods trying to cover as much as possible the code of a Java class under test. Once the student is confident with the adequacy of the written tests, he can challenge the Robot to propose its test cases and compare their respective coverage levels. The tool automatically evaluates the code coverage of both the student's and the Robot test cases, and decrees the winner of the game. When the player beats the Robot, he earns a score that will be used to build a leaderboard of different players. Figure 1 shows the UML activity diagram representing the game mechanics of the Robot Fight challenge. Figure 2 instead shows the page of the game offered to the players during the challenge execution. The upper left frame shows the editor for writing test cases and a toolbar to input commands, including the Compile/Run/Play commands. The upper right frame shows the code of the class under test that will be highlighted in red/green color after the test run, depending on the achieved code coverage. The lower left frame presents the test compilation results, while the lower right ones will show the results of the game and the winner of the challenge, reporting the different levels of code coverage achieved by the player and the Robot. In the current implementation of the tool it is possible for a player to choose both the Class and the Robot to challenge. The app offers several Robots having different test case generation capabilities. The Robots are based either on the Randoop[2] or Evosuite[3] test case generators, which are run for time slots of different lengths in order to generate test suites with different coverage levels. In this way we are able to simulate more Robots with different levels of "power". For having the Robot test cases immediately available and reducing the waiting time, the Robot test cases are not generated during the game, but are produced beforehand and stored in a repository. A second type of challenge offered by the tool is called *Boss-Rush* and allows a player to fight more than one Robot at each game attempt. The dynamic of the game is the same as the former type, even if the challenge has a greater complexity. The tool also offers the *Training* challenge that allows the player to play as much rounds as he wants against the same Robot, with the aim of exploring how to write better test cases, having the possibility to compare the test case coverage at each round. Another type of challenge named *Climbing* presents multiple levels to appeal the players with different skill levels. At each level the player has to test a different class. Each time the Robot is defeated, the player earns a score and reaches the successive level of the game. The scores are assigned based on the code coverage that was reached and the number of attempts that were necessary to beat the Robot, according to similar metric approaches used in the literature [1]. Different players can choose to run the same Climbing, so that the game builds a Climbing leaderboard and assigns a Reward to the player with the highest score.

The tool has been designed to be easily configurable by software testing teachers who want to tune the challenges offered by the application with the characteristics and learning goals of the courses they teach. To this aim, classes having different complexities can be uploaded and assigned to the Climbing challenges. The tool also offers the teacher a dashboard where he can monitor the students who actually registered to the tool, the number of challenges they played and won, and the time they spent for the challenge executions.

3 Tool Implementation

The Testing Robot Challenge tool has been developed as a Web application usable from a Web browser, to be accessible by teachers and students from different sites. The tool architecture is modular, based on the micro-service style, designed to be easily evolved and deployed. An architecture overview is shown in Figure 3 which illustrates the composing microservices (from T1 to T9) exposed by REST APIs and deployed in the back-end of the application.

Three of these services implement a data persistence, such as the T1 (offering the administrator front-end for logging and configuring the challenges offered by the tool), the T23 service (responsible for Players' registration, authentication, and access to the game), and T4, which is the Game Repository that offers the APIs for storing and retrieving all the data related to the played challenges. The T5 and T6 components implement the Game Front-End and Game Engine, respectively, using the Java Spring MVC framework and the open

[2]Randoop, https://randoop.github.io/randoop/
[3]EvoSuite, https://www.evosuite.org/

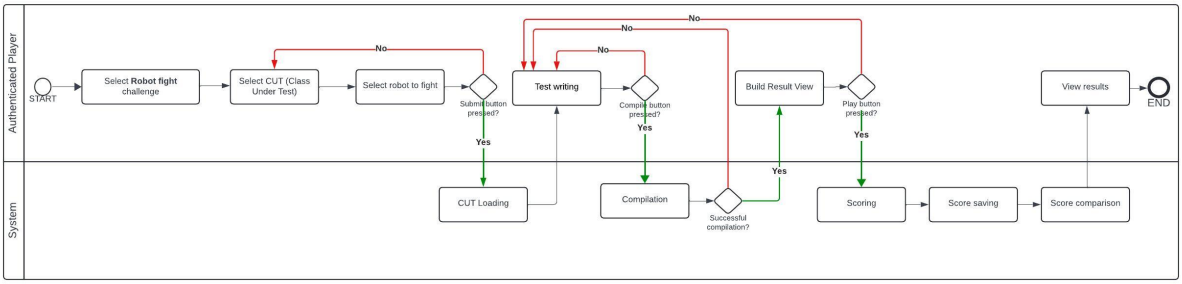

Figure 1: The Game Mechanics of the Testing Robot challenge

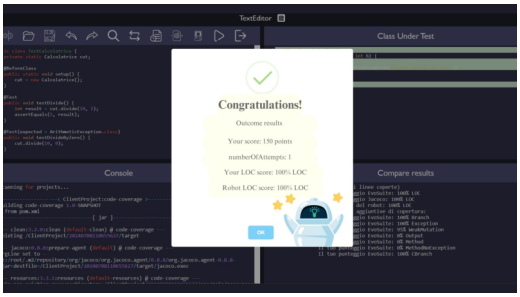

Figure 2: End of the Game

source Code Mirror component[4], usable for implementing code editing features in a Web browser. T7 offers the Build&Run services that are responsible for building and running the tests written by the player, and evaluating the achieved code coverage by the JaCoCo library. T8 and T9 provide the APIs for automatically generating the test cases for a given input class using Evosuite and Randoop, respectively. T8 also offers a dedicated API for evaluating additional coverage metrics and the mutation score achieved by test cases, using the Evosuite coverage evaluation and mutant generation features. The architecture integrates an API Gateway component.

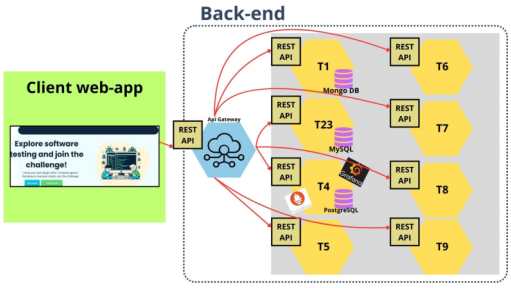

Figure 3: Architecture of the Web App

4 Validation

The goal of the proposed Testing Robot Challenge tool is to motivate the students to do more practice of unit testing by means of the gameful experience of the testing challenges. To validate our tool, we decided to carry out a case study with students of a Software Engineering course offered by a Bachelor Degree in Computer Engineering, that aimed to answer the following research questions:

- RQ1 What is the students' perception about the usability of the tool in the learning experience of coverage testing?
- RQ2 What is the students' perception about the usefulness of the tool to learn how to develop effective unit test cases?
- RQ3 What is the student perceived satisfaction in using the tool?

The subjects involved in the study were 15 students of a Software Engineering course offered in a Bachelor Degree in Computer Engineering at the University of Naples, Federico II. During the course the students had the opportunity to learn basics of white-box and black-box test case design techniques and how to implement test cases with JUnit. The students involved in the study were recruited voluntarily and did not receive any reward for their participation in the study. As to the classes involved in the challenges, we chose Java classes belonging to the SF110 repository originally proposed in [4]. The test classes have a size between 345 and 750 LOCs, with an average of 520 LOCs. The experimental procedure included the following steps:

- the students were presented the Game and its features in a practical 2-hours lecture, where they learned how to play the challenges against the Robots;
- each student was assigned a homework consisting in playing a Robot Fight, a Boss-Rush, and a Climbing challenge. There was no specific time constraint except that the homework had to be completed in two weeks;
- after completing the assignment, the students answered an anonymous Post- questionnaire designed to collect their opinions about the unit testing learning experience they made with the support of the tool.

4.1 Results

To answer our research questions, we analysed the answers to the questionnaire. It was composed of 29 questions, structured in three sections, each one addressing a different attribute among usability, usefulness, and user satisfaction. For each question, the students had to provide a level of agreement according to a five-value Likert scale (Strongly agree = 5. Agree =4, Neutral =3, Disagree=2, Strongly Disagree = 1). In a conclusive section of the questionnaire the students could also provide comments, suggestions or issues

[4]Code Mirror,https://codemirror.net/

encountered with the tool. The questionnaire and the frequency of the student answers are available online[5].

As to RQ1, the answers to the first part of the questionnaire showed that generally the students were neutral about the usability of the tool. In particular, on the basis of the latter 5 question answers, we deduced that some aspects of the tool might be improved, such as the messages it provides to correct errors or to recover from errors made. We further investigated with the students these answers, and understood that the denounced problems regarded the lack of support to solve compilation errors, since the Web app at the moment does not offer debugging features. We may address this limitation in the future evolution of the tool. As to RQ2, the results of the second section showed that most of students agreed about the usefulness of the tool for the learning experience, they were confident and comfortable while learning with the tool, the tool motivated them to learn and met their learning requirements. As regards the User Satisfaction investigated in RQ3, most of the students found the learning experience challenging and stimulating, they were able to achieve the stated goals, remained focused on the tasks, and were satisfied with the feedback provided by the tool. Finally, some of the student comments regarded specific implementation aspects of the tool that will have to be improved in future work: they complained about the small size of the frame offered for test case editing, about the low support offered by the editor to solve compilation errors, and difficulties found to write effective test cases in some challenges. We will address the former two issues in future implementations of the tool. As to the latter comment, since the experimented difficulties could be related to the previous background owned by the students, it confirmed the necessity of accurately tuning the challenges, according to the learning needs of the students. In the future, we intend to carry out further studies involving students having different testing skills and classes having different complexity, in order to evaluate the usefulness of the configurability features of the tool.

5 Conclusions and Future Works

In this paper we presented a tool providing a serious game that aims to motivate the students to do more practice of unit testing by means of the gameful experience of the testing challenges against a Robot. The tool was validated by a preliminary case study involving 15 students of a Software Engineering course offered by a Bachelor Degree in Computer Engineering. The obtained results about usability, usefulness, and perceived satisfaction from the students' point of view are encouraging and suggest possible directions about how to improve the tool in the future. These include enriching the game with new mechanics and components to support the existing challenges, such as the introduction of a hint/feedback system, a reward system for completing specific missions and the possibility of unlocking extra content. In future work, we also intend to design and carry out further experiments involving more students from different courses, to evaluate the benefits of using the tool in different learning contexts. We intend to make available our tool and the supporting teaching materials to foster its adoption by teachers in other Software Testing courses and to evaluate the tool features from the teachers' point of view.

[5] https://zenodo.org/records/13134335

Acknowledgments

This work has been partially funded by ENACTEST (European innovation alliance for testing education), ERASMUS+ Project number 101055874, 2022-2025. The first author thanks the Master students of the "Software Architecture Design" course at the University of Naples Federico II who contributed to the development of the Testing Robot Challenge Web application.

References

[1] Filippo Cacciotto, Tommaso Fulcini, Riccardo Coppola, and Luca Ardito. 2021. A Metric Framework for the Gamification of Web and Mobile GUI Testing. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 126–129. https://doi.org/10.1109/ICSTW52544.2021.00032

[2] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. 2011. Gamification. using game-design elements in non-gaming contexts. In *CHI '11*. ACM, 2425–2428. https://doi.org/10.1145/1979742.1979575

[3] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. 2007. Bug Hunt: Making Early Software Testing Lessons Engaging and Affordable. In *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*. IEEE CS, 688–697. https://doi.org/10.1109/ICSE.2007.23

[4] Gordon Fraser and Andrea Arcuri. 2014. A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite. *ACM Trans. Softw. Eng. Methodol.* 24, 2, Article 8 (dec 2014), 42 pages. https://doi.org/10.1145/2685612

[5] Gordon Fraser, Alessio Gambi, Marvin Kreis, and José Miguel Rojas. 2019. Gamifying a Software Testing Course with Code Defenders. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, 571–577. https://doi.org/10.1145/3287324.3287471

[6] Gordon Fraser, Alessio Gambi, and José Miguel Rojas. 2020. Teaching Software Testing with the Code Defenders Testing Game: Experiences and Improvements. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 461–464. https://doi.org/10.1109/ICSTW50294.2020.00082

[7] Félix García, Oscar Pedreira, Mario Piattini, Ana Cerdeira-Pena, and Miguel Penabad. 2017. A framework for gamification in software engineering. *Journal of Systems and Software* 132 (2017), 21–40. https://doi.org/10.1016/j.jss.2017.06.021

[8] Vahid Garousi, Austen Rainer, Per Lauvås, and Andrea Arcuri. 2020. Software-testing education: A systematic literature mapping. *Journal of Systems and Software* 165 (2020), 110570. https://doi.org/10.1016/j.jss.2020.110570

[9] Pedro Henrique Dias Valle, Armando Maciel Toda, Ellen Francine Barbosa, and José Carlos Maldonado. 2017. Educational games: A contribution to software testing education. In *2017 IEEE Frontiers in Education Conference (FIE)*. 1–8. https://doi.org/10.1109/FIE.2017.8190470

[10] E.L. Jones. 2001. An experiential approach to incorporating software testing into the computer science curriculum. In *31st Annual Frontiers in Education Conference*, Vol. 2. F3D–7. https://doi.org/10.1109/FIE.2001.963741

[11] Beatriz Marín, Tanja E. J. Vos, Monique Snoeck, Ana C. R. Paiva, and Anna Rita Fasolino. 2023. ENACTEST project - European Innovation Alliance for Testing Education. In *Proceedings of CAiSE 2023*, Vol. 3413. CEUR-WS.org, 91–96. https://ceur-ws.org/Vol-3413/paper13.pdf

[12] Antonio Materazzo, Tommaso Fulcini, Riccardo Coppola, and Marco Torchiano. 2023. Survival of the Tested: Gamified Unit Testing Inspired by Battle Royale. In *2023 IEEE/ACM 7th International Workshop on Games and Software Engineering (GAS)*. 1–7. https://doi.org/10.1109/GAS59301.2023.00008

[13] Rebecca Smith, Terry Tang, Joe Warren, and Scott Rixner. 2017. An Automated System for Interactively Learning Software Testing. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, 98–103. https://doi.org/10.1145/3059009.3059022

[14] Philipp Straubinger and Gordon Fraser. 2023. A Survey on What Developers Think About Testing. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. 80–90. https://doi.org/10.1109/ISSRE59848.2023.00075

[15] Porfirio Tramontana, Beatriz Marín, Ana C. R. Paiva, Alexandra Mendes, Tanja E. J. Vos, Domenico Amalfitano, Felix Cammaerts, Monique Snoeck, and Anna Rita Fasolino. 2024. State of the Practice in Software Testing Teaching in Four European Countries. In *17th IEEE International Conference on Software Testing, Verification and Validation (ICST) 2024*. https://doi.org/10.1109/ICST60714.2024.00015

[16] Wu Wen, Jiahui Sun, Ya Li, Peng Gu, and Jianfeng Xu. 2019. Design and Implementation of Software Test Laboratory Based on Cloud Platform. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 138–144. https://doi.org/10.1109/QRS-C.2019.00039

[17] Tamara Zivkovic and Miodrag Zivkovic. 2021. Survey of Learning Environments for Software Testing Education. In *7th Conference on the Engineering of Computer Based Systems* (Novi Sad, Serbia) *(ECBS 2021)*. ACM, Article 7, 9 pages. https://doi.org/10.1145/3459960.3459971

Received 2024-07-03; accepted 2024-07-24