Teaching software testing with role-play games

Charlotte Pierce
Emma Yench
charlotte.pierce@monash.edu
emma.yench@monash.edu
Monash University
Clayton, Australia

Course Other, Software testing
Programming Language None
Resource Type Lab
Knowledge Unit Software Development Methods
CS Topics Testing and Test-driven Development, Tracing and Debugging

Synopsis

Role-play activities have been used for many years to help students engage with abstract content and build understanding in a fun and collaborative way [1, 2]. They are particularly popular in teaching object-oriented programming and concurrency [3, 5], as it is relatively straightforward for students to take on the role of objects within a software system.

Software testing is typically seen as a dry and uninteresting discipline [7]. There are several reasons for this: it involves many abstract concepts, the process of designing and writing software tests can be quite repetitive, and students struggle to see the value of thorough testing without real-world experience. These challenges create an environment where teaching software testing is particularly challenging compared to other areas of computing. To address this, we designed role-play games with the goal of improving student learning and engagement in a software testing class.

In the game, students are given a short case study relating to an ice-cream machine, and take on the role of either a tester or the machine. Testers are given a description of the machine's purpose and its inputs. Machines are given the same description, and instructions describing how to respond to inputs. Testers then work to find the bugs in the machine by verbally interacting with them.

In evaluating the role-plays over a single semester of an undergraduate introductory software quality and testing unit with approximately 260 students at an Australian university, we found that game players had a statistically significant improvement in confidence in their software testing skills. Players also expressed significantly positive sentiment towards the games, and a strong perception that playing them helped them learn [4].

In this work we present four role-playing scenarios, including instructions for the machine and tester roles, and for the teacher running the exercise. Each scenario has a different focus, but all are designed to help students learn black box testing techniques and practice critical thinking related to test design, choice of test parameters, and bug identification. The first scenario is designed to help students learn the structure of the exercise and build rapport with their peers. The second introduces a requirement to document tests and any bugs that were found. The third tasks students with choosing an appropriate testing technique for the given scenario, and the fourth requires students to use state transition testing specifically. These can be easily adapted, extended, or re-themed for different contexts or focuses.

Keywords

Software testing, group activity, role-play game

ACM Reference Format:
Charlotte Pierce and Emma Yench. December 2025. Teaching software testing with role-play games. In *ACM EngageCSEdu.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3786355

1 Engagement Highlights

The role-play games encourage collaborative learning in a low-stakes environment. In our delivery, classrooms where role-play games were being played were lively and relaxed, and set a positive tone for the remainder of the session even as we moved on to other activities. Students developed friendly rivalries, became more open to helping each other, and engaged with the following activities in the session more readily. This all occurred in an environment where we were initially warned that the games would most likely fail as students would refuse to talk with one another.

The role-play games use the following practices from the NCWIT engagement framework [6]:

ACM EngageCSEdu, December 2025.

ACM ISBN 979-8-4007-2421-3/2025/12
https://doi.org/10.1145/3786355

Scenario	Focus
1	Learning how the game works
	Building rapport with peers
2	Documenting tests
	Writing bug reports
3	Choosing an appropriate testing technique
4	State transition testing
	Drawing state machine diagrams

Table 1: The focus of each role-play scenario

- **Use Well Structured Collaborative Learning** Students are learning together through discovering software bugs as a team, collaborating on accurately reporting bugs, and helping each other design tests. The game is intentionally designed as an ungraded activity so students can feel safe making mistakes without the threat of penalty.
- **Encourage Student Interaction** The game is inherently interactive. There are no digital components, and it requires players to talk with one another to 'run' tests, decide which tests to try, and discuss the outputs.
- **Give Effective Encouragement** Students encourage each other throughout the game, and staff can further facilitate a positive learning environment. In our evaluation, students reported a significantly positive increase in confidence in their software testing skills as a result of playing the game [4].
- **Provide Opportunities for Interaction with Faculty** Staff facilitate the game, giving hints and verifying bugs.
- **Address Misconceptions About the Field of CS** This game emphasises the collaborative and fun nature of CS work, which is often seen as dry and isolating.

2 Recommendations

2.1 Expected knowledge

The intended focus of each role-play is described in Table 1. These suggest the knowledge students are expected to have to play each scenario. For example, scenario 1 has no prerequisite knowledge. Scenario 2 requires students to know how to document tests and write a bug report. In scenario 3 students must select a specific black-box testing technique. Finally, in scenario 4 students must use state transition testing.

The intention is that the four scenarios are of increasing difficulty (i.e., role-play 1 is the easiest, and role-play 4 is the hardest). However, they can be played in any order if desired. If they are played in an alternative order, we suggest always doing role-play first as it is designed to introduce the structure of the game.

2.2 Running the game

We recommend playing the role-play game towards the start of a class, and playing one per week, with an optional week's break between role-plays 2 and 3 (assuming weekly classes). The collaborative nature sets a positive tone and gets students talking with one another, which then benefits any following activities. It should take approximately 30 - 45 minutes for students to play.

For the first role-play, it can help for teaching staff to briefly demonstrate how the game works. For example, one staff member can run a few tests while another staff member acts as the machine. If there is only one staff member, they can ask for inputs from the students. This is a quick and clear way of showing how the game works.

The instructions can be effectively shared through the use of a QR code. For students who would prefer to read instructions off their laptop, we recommend the QR code is also annotated with a shortened URL. To play the game, students should split into groups of at least 4. At least 2 people in each group should take on each role, so that everyone has someone to work with and no single person is highlighted. We recommend that instructors emphasise that the ice-cream machine only takes verbal instructions to avoid any unwanted physical contact, and that it only takes one instruction at a time.

Students should be encouraged to shout 'bug report!' or alert a staff member when they believe they have found a bug. However, we recommend not verifying whether the bugs are correct or not until the end of the game. This encourages students to think critically about whether they have verified the boundaries of a bug, and whether the behaviour they observe is the result of a single bug, multiple bugs, or interacting bugs. Instead of verification, staff can provide hints (e.g., "are you sure, or do you need to run some more tests?") and encouragement, and direct the students to keep testing.

2.3 Game wrap up

When most groups seem to have either found all the bugs or stopped progressing, we recommend ending the game. As a 'wrap up' and reflective activity, you can then have a discussion. This should include verifying what the bugs actually were and exploring how students found them, or how they could have if they didn't.

In many cases we observed students 'finding' 8 or 9 bugs when there was only 5. This led to rich discussions on determining the boundaries of a bug's behaviour, and the risks of being overly specific (or overly vague) in describing bugs.

2.4 Adding friendly competition

We found it effective to get each group of players to nominate a team name, and publicly tally the number of bugs found

by each team as they played. This provides incentive to the students to progress, encourages some friendly rivalry, and provides the basis for rich discussion at the conclusion of the game in how each group approached finding bugs, how many bugs there actually were, and how to find the boundaries of a bug.

3 Materials

The attached zip contains the materials for all four role-play scenarios. Each scenario is in its own folder, and is split into instructions for students taking on each role (i.e., 'machine' and 'tester'), and instructions for staff.

References

[1] S. K. Andrianoff and D. B. Levine. 2002. Role playing in an object-oriented world. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*. 121–125.

[2] A. Bertoni. 2019. A reverse engineering role-play to teach systems engineering methods. *Education Sciences* 9, 1 (2019), 30.

[3] E. Erturk. 2015. Role play as a teaching strategy. In *National Tertiary Learning and Teaching Conference*.

[4] Anonymised for review. Anonymised for review. Anonymised for review. In *Anonymised for review*.

[5] A. G. Harrison and D. F. Treagust. 2006. Teaching and learning with analogies: Friend or foe? *Metaphor and analogy in science education* (2006), 11–24.

[6] NCWIT. 2025. Engagement Practices Framework. (2025). https://ncwit.org/engagement-practices-framework/

[7] Philipp Straubinger and Gordon Fraser. 2024. Gamifying a Software Testing Course with Continuous Integration. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. 34–45.