Design of a Serious Game on Exploratory Software Testing to Improve Student Engagement

Niels Doorn[1,2][a], Tanja E. J. Vos[1,3][b] and Beatriz Marín[3][c]

[1]*Open Universiteit, The Netherlands*
[2]*NHL Stenden University of Applied Sciences, The Netherlands*
[3]*Universitat Politècnica de València, Spain*

Keywords: Software Testing Education, Exploratory Testing, Game Based Learning.

Abstract: Teaching software testing in computer science education faces challenges due to its abstract nature and students' focus on approaches using paradigms based on rationalism. Exploratory testing, which uses a paradigm based on empiricism and employs reflective learning, is under-represented in computer science curricula. To address this gap, game-based learning presents promising approaches to enhance engagement and foster critical thinking in software testing education. This position paper presents the design of a serious game to support the teaching of exploratory software testing to improve the students' engagement. The game integrates software testing tours and uses Socratic questioning as scaffolding to promote deeper reflection-in-action, allowing students to experience hands-on learning in software testing. Using a mapping review, this study identifies the most effective gamification techniques for software testing education and principles of Socratic questioning. Based on these findings, we designed a game that focusses on exploratory testing scenarios, where players follow a tour-based test strategy on a system under test.

1   INTRODUCTION

Software testing is a crucial component of the software development life cycle and a highly valued skill in the industry. However, integrating software testing effectively into Computer Science curricula has been a challenge for educators (Garousi et al., 2020)(Scatalon et al., 2020).

Other studies investigated how students approach testing, revealing that many adopt the so-called 'developer approach' rooted in a design paradigm based on rationalism (Doorn et al., 2021)(Doorn et al., 2023). This approach focusses on algorithmic problem solving and structured planning, often leading to incomplete testing practices. This approach lacks exploration and context awareness, which is essential to gain insight into software quality. To address this, a paradigm shift based on empiricism is proposed that encourages experimentation, asking questions, and critical thinking (Doorn et al., 2021)(Doorn et al., 2023).

In this paper, we state our position that the sense-making of students in learning testing within an empiricism-based paradigm can be effectively enhanced and supported by employing *software testing tours*, scaffolded by *Socratic questioning*, through the *serious game* we propose.

Software Testing tours (Bolton, 2009)(Kaner et al., 1993) are testing heuristics that use metaphorical "tours" to guide testers through different areas of an application, helping them detect defects, improve usability, and identify edge cases. Each tour serves as a specific approach or perspective for examining the software, such as concentrating on its features, data, configuration, or user behaviour. For instance, the Feature Tour is designed to help testers become familiar with the application's primary features, whereas the Complexity Tour delves into the most complicated portions of the system where defects are prone to occur. These tours motivate testers to think analytically, systematically alter inputs and conditions, and investigate areas that might be neglected in other testing methods. Testing tours prove to be highly effective in exploratory testing due to their focus on creativity, flexibility, and thorough evaluation of the software.

Socratic questioning (Paul and Elder, 2019) is a pedagogical method that fosters critical thinking, reflective inquiry, and problem solving by challenging assumptions and encouraging deeper analysis. In software testing, it complements empirical testing

[a] https://orcid.org/0000-0002-0680-4443
[b] https://orcid.org/0000-0002-6003-9113
[c] https://orcid.org/0000-0001-8025-0023

by promoting exploration, iterative learning, and a deeper understanding of system behaviour. By guiding students to question their approaches and consider alternative test scenarios, it aligns naturally with the investigative nature of exploratory testing.

Serious games (Deterding et al., 2011) Facilitate learning by simulating real-world scenarios, allowing students to experiment and learn in a safe and controlled environment. Moreover, serious games offer personalised learning experiences, adapting to the player's skill level to maintain an optimal balance of challenge and ability to keep students engaged and let them receive instant feedback, helping them identify and correct errors in real time, and ultimately enhancing their learning experience. In general, gamification. has shown positive effects in improving the teaching of complex topics (Dicheva et al., 2015)(de Sousa Borges et al., 2014)(Caponetto et al., 2014)

The main contribution of this position paper is the design of a serious game we call BugOutbreak, where students apply testing strategies through interactive game play, with Socratic questioning integrated to stimulate critical thinking, and guided by software testing tours.

In earlier work, we designed a learning activity based on Risk Storming using TestSphere (Bib, 2020) cards together with Socratic questioning using a digital tool to randomly select a Socratic question. Although beneficial for learners, the design of this teaching activity lacked sufficient elements to create a fully immersive experience that is needed to be useful as an effective educational instrument. In this paper, we address this limitation by designing a serious game to support the learning of exploratory software testing.

The paper is organised as follows. Section 2 states our position on the use of a game we designed to teach exploratory software testing. Section 3 details the design of the game. Section 4 presents strategies for integrating the game into educational contexts. Section 5 provides strategies for integration of the game into education. Section 6 presents relevant related work in the field. Finally, Section 7 concludes the work.

2 HOW TO TEACH TESTING

In this section, we elaborate on the role of sensemaking and Socratic questioning and argue why our proposed game-based learning approach is well-suited to support teaching testing according to the paradigm based on empiricism.

Sensemaking is the cognitive process through which individuals construct meaning from complex and ambiguous situations (Odden and Russ, 2019). In software testing, this involves developing an understanding of the system under test (SUT), identifying potential issues, and iteratively refining the testing strategies. Good testing requires testers to continuously adapt their understanding based on empirical observations.

In the context of learning, effective sensemaking in testing involves cycles of hypothesis generation, empirical observation, and reflection as depicted in Figure 1.

Figure 1: The sensemaking cycle representing the dynamic nature of sensemaking during testing.

Students must develop the ability to recognise patterns, ask meaningful questions about software behaviour, and refine their mental models of the system. However, without proper guidance, students may struggle to engage in the necessary sensemaking, leading to ineffective testing approaches that rely too heavily on structured methodical techniques instead of embracing the exploratory nature of the task.

To support the sensemaking in testing, we propose Socratic questioning as a scaffolding technique. When applied to testing, Socratic questioning guides students to challenge their assumptions about the SUT, explore different strategies, and reflect on their findings.

High-level examples of Socratic questions relevant to testing include: What assumptions are we making about the expected behaviour of the system? What happens if we interact with this feature unintentionally? How does this component behave under extreme or unexpected inputs? Finally, how do we know that this issue is critical, and what additional evidence can we gather?

To guide students into exploratory testing, we propose the use of software testing tours.

Given these theoretical foundations, our hypothesis is that a game-based learning approach integrating Socratic questioning will foster an effective sensemaking process aligned with the paradigm based on empiricism for test education. The key advantages of this approach include:

Active Engagement in Sensemaking. The game requires players to continuously observe, hypothesise, and experiment, ensuring that sensemaking remains active and iterative.

Guided Reflection Through Socratic Questioning.
By embedding Socratic prompts, students receive ongoing guidance that challenges their assumptions and encourages deeper inquiry.

Reinforcement of Empirical Testing Principles.
Players experience first-hand how to adapt their strategies based on real-time observations and feedback.

Guidance into Exploratory Testing. The use of software testing tours helps students follow different testing strategies.

Collaborative Learning Opportunities. The cooperative mechanics of the game allow students to learn from the testing approaches of each other.

By combining these elements into a structured yet flexible learning experience, our approach addresses key challenges in software testing education and supports the development of critical skills needed for effective exploratory testing.

In the following sections, we detail the design and implementation of our game and discuss how it can be integrated into educational contexts to maximise its impact on student learning.

3 THE DESIGN OF OUR GAME

3.1 Conceptual Design

The goal of the game is for players to collaboratively test a SUT in order to evaluate the quality. Each player will be assigned a different testing tour. Each of these tours focusses on a different aspect of the system and comes with a set of relevant Socratic questions to support students in designing tests. Examples of tours and their Socratic Questions are in Table 1.

In each turn, a player is presented with a Socratic question related to the tour assigned to that player. As players progress through their tours, they use the SUT to identify possible issues. This is done by interacting with the SUT simulated in the game. The SUT will generate a response that the player can verify in order to determine whether the SUT behaves according to expectations. If it does, the player did not find an issue. If not, potential issues have been found. In both cases, the player has advanced in testing the SUT. Points are awarded for successfully identifying issues or defects. These scores are displayed individually on the game leaderboard, adding a competitive aspect to the collaborative experience. The players also learn from each other's actions as they follow different tours.

Students win if they successfully identify enough issues before time or resources run out. The student with the highest score is the winner. The game is lost if too many critical bugs remain unidentified, causing a cascade of system failures, or if time runs out before the software is fully tested. Individual scores can be used as personalised feedback for players. After the game is finished, there is the option to evaluate each other and their own performance. The flow of the game is depicted in Figure 2.

3.2 Typical Game Play Scenario

Based on the game design, a typical game play scenario with three players could be as follows.

The SUT in this example is a program that calculates the average age of customers who booked vacations with a travel company. It retrieves data from a remote server in a `.txt` file where each line contains the customer's name and date of birth. The objective of the players is to test and verify the programs ability to handle various data inputs, identify critical bugs, and ensure accurate results. The three players in this session are randomly assigned the following testing tours: **Player 1** takes the *Data Tour*, focussing on input validation; **Player 2** takes the *Feature Tour*, focussing on core functionality; and **Player 3** takes the *Back Alley Tour*, exploring rarely tested areas.

Player 1 begins by reviewing known details about the program and identifies data validation as a critical area for testing. Prompted by the Socratic question, *"What assumptions have you made on the form and the values of the data that the system needs to process?"*. This question stimulates the player to consider using invalid date formats. Player 1 simulates inputs containing valid names but invalid ages (e.g., `"John Doe, -5"` or `"Alice Smith, abc"`), as well as lines missing age information (e.g., `"Jane Doe,"`). The SUT attempts to calculate the average, but throws errors or returns an incorrect result due to the invalid data. This allows Player 1 to identify a critical bug, earning points for detecting it and expanding the test coverage.

Player 2 takes the next turn, focussing on the core functionality of the program. After reviewing the purpose of SUT, Player 2 reflects on the Socratic question, *"What happens if a feature is misused, either intentionally or accidentally?"*. This question makes the player think about how to misuse the system and helps the player to come up with a test in which different input files are uploaded to the system. Player 2 then tests the SUT by uploading files with varying numbers of valid entries, such as 10, 100 (maximum capacity), and an edge case of an empty file. The SUT

Table 1: Software testing tours from (Bolton, 2009).

| Tour Type	Description and Example Socratic Questions
Feature	Explore core features of the software, testing their functionality and integration with other features. *Example:* "What is the primary purpose of this feature? How does it interact with other parts of the system?"
Data	Input various types of data (valid, invalid, edge cases) to uncover issues with data handling and validation. *Example:* "What kinds of data is this system expected to handle? How might it respond to unexpected input?"
Back Alley	Focus on testing obscure or rarely used features to identify hidden bugs. *Example:* "What features or pathways are less likely to be explored? Why might they be overlooked?"
Money	Test the most critical revenue-generating features of the system to ensure they perform flawlessly. *Example:* "What features are most critical to the system's success? How could failures in these areas affect the system's value?"
Bad-Neighbourhood	Test areas of the software that have historically had many bugs or stability issues. *Example:* "What areas of the system have caused problems in the past? What underlying factors might contribute to instability here?"
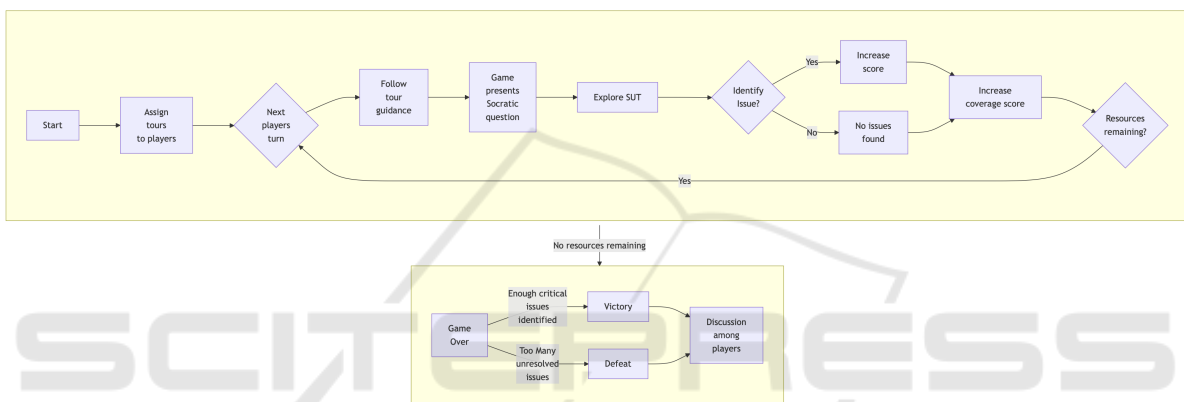
Figure 2: Simplified game flow of the cooperative software testing game for exploratory testing.

handles smaller inputs correctly but throws an exception for the empty file, revealing an issue with edge case handling. Player 2 documents this bug and earns points for increasing test coverage and identifying another critical issue.

Player 3 takes a different tour, focussing on rarely tested or obscure areas of the system. Guided by the Socratic question, *"What assumptions are we making about the reliability of these features?"*. This makes the player hypothesise that the feature might not have been tested with data containing special characters. The player tests the SUT by uploading a file with names containing special characters (e.g., `"José Álvarez, 34"`) and another corrupted file with random binary content. The SUT processes the file with special characters correctly but fails to handle the corrupted file. Player 3 documents the issue and suggests an enhancement for more informative error messages, earning points for identifying a rare issue, and providing actionable feedback.

After the initial round, the players collaborate to discuss their findings. Player 1 highlights the need for improved input validation, Player 2 emphasises bet-

ter error handling for edge cases like empty files, and Player 3 suggests focussing on server interaction and non-standard file formats.

As the game progresses, players identify several critical bugs and gain insight into the SUT's interactions with the remote server. They ultimately win the game by achieving sufficient test coverage and identifying all critical issues before running out of time or resources. A scoreboard shows individual scores based on contributions and a collaborative reflection reinforces the importance of teamwork and shared learning.

3.3 Development of the Game

For the development of the game, we are considering both a physical game and digital options, and a hybrid form. While a digital variant has the benefit of supporting multiple releases through one channel, the use of physical games in Computer Science programmes has the benefit of enabling direct collaboration and avoiding distractions. The game design supports both options and can be relatively easily con-

structed as a prototype game using paper cards and a printable game board. Once the game development is complete, a final offline version can be produced and distributed through existing channels to universities. To enable some parts that are easier to create digitally, a hybrid variant is also considered. This would mean that players would use both the game board and cards, combined with an app on their phone to enable a more comprehensive evaluation of tests and to support custom feedback.

4 INTEGRATION INTO EDUCATIONAL CONTEXT

One of the complications of integrating software testing into existing computer science courses, such as introductory programming or software engineering courses, is the lack of resources available to lecturers. Therefore, this game is designed with this in mind and integration does not require a complete overhaul of existing courses. It can be integrated into different forms of education. For example, it could be part of a tutorial on software testing, it could be recommended to students as part of a group work assignment, or it could be used as a classroom activity during a testing workshop.

The game is designed as a digital product, making the dissemination of new releases possible and the large-scale distribution feasible. This means that the game can be used in many different learning environments.

The game is designed as a supportive tool for teaching and learning exploratory testing, particularly for computer science students. Given the complexity of exploratory testing, the game focusses on practical learning experiences and is not suitable for **summative evaluation**. Summative assessments typically measure learning outcomes at the end of a course or unit and are often used in formal, high-stakes situations (Black and Wiliam, 1998). Since the game is not designed to be a safe assessment environment, meaning that it does not provide standardised controlled conditions to evaluate all students uniformly, it is unsuitable for summative purposes.

However, the game can serve multiple less formal assessment purposes in educational contexts.

It can be used as a *diagnostic assessment* tool at the start of a course to identify student prior knowledge, misconceptions, and skill gaps in exploratory testing, helping instructors tailor their teaching strategies (Popham, 2009).

As a *formative assessment*, the game offers ongoing feedback throughout the learning process, al-

lowing instructors to observe students' strategies in real-time and provide guidance to improve their understanding and application of testing methods.

The game also supports *self-assessment*, empowering students to reflect on their performance, identify areas for improvement, and develop critical thinking skills, essential in exploratory testing (Boud, 2013).

Furthermore, the game facilitates *performance-based assessment*, as it simulates real-world testing tasks, such as identifying bugs without predefined scripts, allowing students to demonstrate active problem-solving and decision-making skills.

Finally, it is suitable for *informal assessments*, where instructors can observe student progress in a low pressure environment, providing immediate insight into their learning (Bell and Cowie, 2001).

5 INTEGRATION INTO EDUCATION

This game has been designed to enable seamless integration without necessitating a comprehensive redesign of existing curricula. It can be incorporated into various educational settings. For example, it could serve as part of a tutorial on software testing, be suggested for group assignments, or be used as a classroom activity during hands-on testing sessions.

Due to the complexity of exploratory tests, the game emphasises practical learning tasks and is not suitable for **summative evaluation**. Summative assessments are generally used to evaluate student learning after completing a course or unit and are often used in formal, critical scenarios (Black and Wiliam, 1998). Since the game is not set up to provide standardised controlled conditions for consistently assessing students, it is not designed for summative use. However, within educational contexts, the game is suitable for various informal assessment roles. It can be employed as a *diagnostic assessment* tool at the course onset to detect students' prior understanding, misconceptions, and skill deficiencies in exploratory testing, helping instructors adapt their pedagogical approaches (Popham, 2009).

As a *formative assessment*, the game provides continuous feedback during the learning journey, allowing instructors to monitor students' approaches in real time and offer advice to improve their comprehension and use of testing techniques (Sadler, 1989). The game also supports *self-assessment*, empowering students to reflect on their performance, identify areas for improvement, and develop critical thinking skills, essential in exploratory testing.In addition, the game supports *performance-based assessment*, as it

simulates authentic testing tasks, such as identifying bugs without predetermined instructions, allowing students to exhibit active problem-solving and decision-making abilities.Lastly, it is appropriate for *informal assessments*, where educators can track student progress in a stress-free setting, providing immediate insights into their learning.

5.1 Evaluation Metrics

The study aims to measure students' autotelic experiences using a survey based on Sillaots' design (Sillaots and Jesmin, 2016). We will use a survey including open-ended questions for personal reflections, with demographic queries, and the participants' game scores to gather additional information.

To evaluate the impact of the game on the learning effectiveness of exploratory testing and the future intention to use the game, we will use well-known questionnaires such as the System Usability Scale (SUS) (Brooke, 1996) and the Game Experience Questionnaire (GAMEX) (IJsselsteijn et al., 2013) to collect perceptions of the lecturers using the game.

6 RELATED LITERATURE

In a previous study (Doorn et al., 2024) on the need to shift from a paradigm based on rationalism to a paradigm based on empiricism in the education of software testing with the help of gamification, we reviewed the literature. on serious games in computer science education, specifically focussing on software testing. The results indicated that gamification techniques, such as real-world scenarios, immediate feedback, and collaboration, have proven to be effective in computer science education. It emphasises the importance of competitive, collaborative, and inquiry-based learning methods in enhancing engagement and critical thinking in students.

The existing body of knowledge shows that while gamification and game-based learning are being applied in software testing education, none incorporate Socratic questioning. These two aspects, game-based learning and Socratic questioning, are key elements in the design of our game. To extend the literature review, we performed an additional mapping review (Kitchenham et al., 2011) of recent literature focused on two additional aspects: the use of gamification and game-based learning in exploratory testing and suitable frameworks to use in the evaluation of game design. The search process gathered 172 candidate articles, and we selected 17 articles applying

the inclusion criteria. The complete protocol can be found online[1].

6.1 Gamification and Game Based Learning on Exploratory Testing

The integration of gamification into educational settings, particularly in the field of exploratory software testing, has been shown to improve engagement and improve learning outcomes. For example, (Yan et al., 2019) demonstrated that gamification improves student participation in assessments, creating a more interactive learning environment. Similarly, the use of structured gamified strategies to teach exploratory testing, as described in (Costa and Oliveira, 2019), makes the learning process more hands-on and effective for students. In addition, the study by (Blanco et al., 2023) further supports the positive impact of gamification on software testing education, highlighting improvements in student comprehension and motivation when applying testing frameworks.

Frameworks combining gamification with exploratory testing, such as those in (Costa and Oliveira, 2020), improve the learning process by encouraging active participation and practical experimentation. Real-world experiences in teaching exploratory testing through gamified methods, reported by (Lorincz et al., 2021), highlight increased student engagement.

Despite these positive outcomes, there are challenges in implementing gamification, as outlined in (McCallister, 2019). These include technical difficulties in integrating gamification into existing learning platforms and balancing the fun aspect with educational objectives. Important for our objective are the study results that show that gamification continues to play an important role in student engagement, particularly in fostering active learning environments, as highlighted by (Adams, 2019). In addition, the use in assessments, as explored by (Zainuddin et al., 2023), improves motivation by making the learning process more interactive and aligned with learning objectives such as the SOLO taxonomy (Biggs and Collis, 1982).

6.2 Frameworks for Game Design

Educational games around testing often draw on established frameworks to create engaging and effective learning experiences. The principles of James Paul Gee (Gee, 2003) emphasise the importance of active learning, critical thinking, and situated meaning. These principles can guide students to actively ex-

---

[1]https://research.nielsdoorn.nl

plore software systems and develop testing hypotheses, much as testers do when solving real-world problems. Reflection and analysis of errors further reinforce learning through iterative testing processes, making these principles particularly useful for our exploratory testing games.

The Game-Based Learning (GBL) framework (Ma et al., 2011)(Squire, 2011) emphasises the need for clear goals, immediate feedback, and challenge-reward structures. Immediate feedback on test progress ensures that players remain engaged, while progressively difficult testing scenarios provide a rewarding sense of accomplishment as they improve their skills.

Flow theory (Nakamura et al., 2009), focusses on maintaining player immersion by matching game challenges with player abilities. In testing games, this can be achieved by adjusting the difficulty of the bug or the complexity of the system to ensure a continuous state of flow. Clear goals and immediate feedback loops are essential to keep players engaged, allowing them to feel control over their actions as they explore and test software systems.

The Cognitive Apprenticeship model (Dennen and Burner, 2008) highlights learning through guided practice, reflection, and scaffolding. This scaffolding process mirrors how real-world testing skills are developed through practice and feedback. Scaffolding is also part of the 4C/ID model (Van Merriënboer et al., 2002). Both models emphasise learning through participation in authentic real-world tasks, with a focus on complex skill development.

Finally, Self-Determination Theory (SDT) (Deci and Ryan, 2012) and the LM-GM model (Lim et al., 2015) emphasise autonomy, competence, and relatedness, with game mechanics designed to align with learning objectives. Our game allows players to explore systems and develop competence by solving increasingly complex testing challenges.

7 CONCLUSION

Testing education is flawed because it follows paradigms based on rationalism while neglecting empiricism, which emphasises learning through experience and observation. In this position paper, we argue that game-based learning using software testing tours and Socratic questioning can improve the teaching of empirical testing.

To support this position, we conducted a mapping review and found that there is work on game-based learning for software testing, and specifically for exploratory testing, but they face challenges in balancing the fun activities with educational goals. To overcome these challenges, we discuss the frameworks for game design and evaluation of student engagement that we use.Based on this, we presented the design of a game to support the teaching of exploratory software testing. Using different software testing tours and Socratic questions to scaffold their testing efforts, players simulate real-world challenges, in order to create an engaging experience to train their exploratory software testing skills.

Immediate future work is related to the development of the game. Future work also includes the evaluation of the game in software engineering and software testing courses.

ACKNOWLEDGEMENTS

REFERENCES

(2020). Risk-storming. [Online; accessed 13. Jan. 2024].

Adams, S. (2019). *The Role of Gamification in the Facilitation of Student Engagement: An Exploratory Application*. PhD thesis, University of Stellenbosch.

Bell, B. and Cowie, B. (2001). *Formative assessment and science education*. Springer Science & Business Media.

Biggs, J. B. and Collis, K. F. (1982). *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, New York.

Black, P. and Wiliam, D. (1998). *Inside the black box: Raising standards through classroom assessment*. Kings College London School of Education London.

Blanco, R., Trinidad, M., and Suarez-Cabal, M. (2023). Can gamification help in software testing education? findings from an empirical study. *Journal of Systems and Software*, 200:111647.

Bolton, M. (2009). Of testing tours and dashboards. Accessed: 2024-10-03.

Boud, D. (2013). *Developing student autonomy in learning*. Routledge.

Brooke, J. (1996). Sus: A quick and dirty usability scale. In *Usability evaluation in industry*, pages 189–194. Taylor & Francis.

Caponetto, I., Earp, J., and Ott, M. (2014). Gamification and education: A literature review. In *European Conference on Games Based Learning*, volume 1, page 50. Academic Conferences Int. Ltd.

Costa, I. and Oliveira, S. (2019). A systematic strategy to teaching exploratory testing using gamification. In

14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pages 307–314. scitepress.

Costa, I. and Oliveira, S. (2020). The use of gamification to support the teaching-learning of software exploratory testing: An experience report. In *IEEE Frontiers in Education Conference (FIE)*.

de Sousa Borges, S., Durelli, V., Reis, H., and Isotani, S. (2014). A systematic mapping on gamification applied to education. In *29th ACM SAC*, pages 216–222.

Deci, E. and Ryan, R. (2012). Self-determination theory. *Handbook of theories of social psychology*, 1(20):416–436.

Dennen, V. and Burner, K. (2008). The cognitive apprenticeship model in educational practice. In *Handbook of research on educational communications and technology*, pages 425–439. Routledge.

Deterding, S., Dixon, D., Khaled, R., and Nacke, L. (2011). From game design elements to gamefulness: defining" gamification". In *15th international academic MindTrek conference: Envisioning future media environments*, pages 9–15.

Dicheva, D., Dichev, C., Agre, G., and Angelova, G. (2015). Gamification in education: A systematic mapping study. *Journal of educational technology & society*, 18(3):75–88.

Doorn, N., Vos, T., and Marín, B. (2024). From rationalism to empiricism in education of software testing using gamification. In *18th INTED*, pages 3586–3595. IATED.

Doorn, N., Vos, T., Marín, B., Passier, H., Bijlsma, L., and Cacace, S. (2021). Exploring students' sensemaking of test case design. an initial study. In *21st Int. Conference on Software Quality, Reliability and Security Companion*, pages 1069–1078. IEEE.

Doorn, N., Vos, T. E., and Marín, B. (2023). Towards understanding students' sensemaking of test case design. *Data & Knowledge Engineering*, page 102199.

Garousi, V., Rainer, A., Lauvås Jr, P., and Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, 165:110570.

Gee, J. P. (2003). What video games have to teach us about learning and literacy. *Computers in entertainment (CIE)*, 1(1):20–20.

IJsselsteijn, W., de Kort, Y., and Poels, K. (2013). The game experience questionnaire. Technical report, Technische Universiteit Eindhoven.

Kaner, C., Falk, J., and Nguyen, H. Q. (1993). *Testing Computer Software*. Wiley, 2nd edition.

Kitchenham, B., Budgen, D., and Brereton, O. (2011). Using mapping studies as the basis for further research–a participant-observer case study. *Information and Software Technology*, 53(6):638–651.

Lim, T., Carvalho, M., Bellotti, F., Arnab, S., De Freitas, S., Louchart, S., Suttie, N., Berta, R., and De Gloria, A. (2015). The lm-gm framework for serious games analysis. *Pittsburgh: University of Pittsburgh*.

Lorincz, B., Iudean, B., and Vescan, A. (2021). Experience report on teaching testing through gamification. In *3rd*

Int. Workshop on Software Testing Education, pages 15–22. ACM.

Ma, M., Oikonomou, A., and Jain, L., editors (2011). *Serious Games and Edutainment Applications*. Springer, London.

McCallister, A. (2019). *The Technical Challenges Of Implementing Gamification: A Qualitative Exploratory Case Study*. PhD thesis, Capella University.

Nakamura, J., Csikszentmihalyi, M., et al. (2009). Flow theory and research. *Handbook of positive psychology*, 195:206.

Odden, T. and Russ, R. (2019). Defining sensemaking: Bringing clarity to a fragmented theoretical construct. *Science Education*, 103(1):187–205.

Paul, R. and Elder, L. (2019). *The thinker's guide to Socratic questioning*. Rowman & Littlefield.

Popham, W. J. (2009). *Assessment literacy for teachers: Faddish or fundamental?* Phi Delta Kappan International.

Sadler, R. D. (1989). Formative assessment and the design of instructional systems. *Instructional Science*, 18(2):119–144.

Scatalon, L. P., Garcia, R., and Barbosa, E. (2020). Teaching practices of software testing in programming education. In *Frontiers in Education Conference (FIE)*, pages 1–9. IEEE.

Sillaots, M. and Jesmin, T. (2016). Multiple regression analysis: Refinement of the model of flow. In *10th European Conference on Games Based Learning*, pages 606–616.

Squire, K. (2011). *Video Games and Learning: Teaching and Participatory Culture in the Digital Age*. Teachers College Press, New York.

Van Merriënboer, J., Clark, R., and De Croock, M. (2002). Blueprints for complex learning: The 4c/id-model. *Educational technology research and development*, 50(2):39–61.

Yan, Y., Hooper, S., and Pu, S. (2019). Gamification and student engagement with a curriculum-based measurement system. *International Journal of Learning Analytics and Artificial Intelligence*.

Zainuddin, Z., Alba, A., and Gunawan, T. (2023). Implementation of gamification and bloom's digital taxonomy-based assessment: A scale development study. *Interactive Technology and Smart Education*, 20(4):512–533.