2017 IEEE/ACM 12th International Workshop on Automation of Software Testing

Gamification of Software Testing

Gordon Fraser
Department of Computer Science, The University of Sheffield, Sheffield, United Kingdom
Email: gordon.fraser@sheffield.ac.uk

***Abstract*—Writing good software tests is difficult, not every software developer's favorite occupation, and not a prominent aspect in programming education. However, human involvement in testing is unavoidable: What makes a test good is often down to intuition; what makes a test useful depends on an understanding of the program context; what makes a test find bugs depends on understanding the intended program behaviour. Because the consequences of insufficient testing can be dire, this paper explores a new angle to address the testing problem: Gamification is the approach of converting potentially tedious or boring tasks to components of entertaining gameplay, where the competitive nature of humans motivates them to compete and excel. By applying gamification concepts to software testing, there is potential to fundamentally change software testing in several ways: First, gamification can help to overcome deficiencies in education, where testing is a highly neglected topic. Second, gamification engages practitioners in testing tasks they would otherwise neglect, and gets them to use advanced testing tools and techniques they would otherwise not consider. Finally, gamification makes it possible to crowdsource complex testing tasks through games with a purpose. Collectively, these applications of gamification have the potential to substantially improve software testing practice, and thus software quality.**

I. Introduction

In software development practice, testing is often neglected [10], and lagging far behind the state of research [22]. Because of insufficient testing, programs crash, apps need to be constantly upgraded, huge economic damage is caused when software bugs are discovered in banking or flight management systems, and people have died due to software failures in cars and other safety related systems.

If an activity is so difficult, boring, or otherwise unattractive that people do not want to engage with it, then automation is often the proposed solution. Tests can be generated automatically, but even though automated test generation techniques have matured over recent years, it is still fundamentally limited: What makes a test good is often down to intuition, and automation applies unreliable and misleading proxy metrics [25]. What makes a test useful depends on an understanding of the program context, and automation leads to unrealistic tests without clear purpose. What makes a test detect bugs depends on understanding the intended program behaviour, and automation often requires extra human effort, for example to provide formal specifications. Thus, while automation can support testing, the importance of testing will never disappear for human software engineers.

A potential solution to this problem is offered by the recent trend of *gamification*: By turning difficult or boring activities into fun and competitive tasks, participants engage, compete, and excel. Educational games are now a common way to teach and practice complex concepts ranging from mathematics to languages. Gamification elements such as leaderboards are used in daily work as well as hobbies (e.g., fitness training). "Games with a purpose" even allow players to solve underlying problems, often without them being aware of this (e.g., participants of the Duolingo language training platform [47] implicitly translate the world wide web). Thus, gamification of software testing promises to address all problems, ranging from education, to practitioner engagement, to limitations of automation.

This paper explores existing and potential future applications of gamification in software testing in these three domains of education, practice, and crowdsourcing.

II. Gamification of Software Testing Education

There is a global trend to include programming throughout K-12 education, starting at the earliest key stages. For example, in the UK the new national curriculum for computing requires programming education to start for children already at key stage 1. Typically, the teaching relies on platforms that implement aspects of the curriculum using puzzle-based online tools suitable also for the earliest learners at KS1/2, such as Education City[1], or Code.org[2]. Later on, programming education often relies on graphical programming environments such as Scratch [38] or Alice [17], and there is evidence (e.g., [32, 53, 54]) that these approaches lead to students understanding and applying high level concepts such as student-created abstractions, concurrent execution, and event handlers. More advanced learners can make use of educational integrated development environments that simplify certain concepts (e.g., BlueJ [30] or Greenfoot [23]), before moving on to regular programming.

Software testing, however, is usually not taught until university, and there it is often a relatively neglected component of the computer science curriculum [13]. Even though testing is said to take 50% or more of the resources for software development projects, the share of a typical undergraduate curriculum is far less [41] and often comes too late, when bad habits have already been established [20]. This is in part because academic curricula emphasise more constructive activities such as software design and implementation. Even curricula based on the ACM/IEEE Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering [43]

[1]http://www.educationcity.com
[2]http://www.code.org

978-1-5386-1548-5/17 $31.00 © 2017 IEEE
DOI 10.1109/AST.2017.20

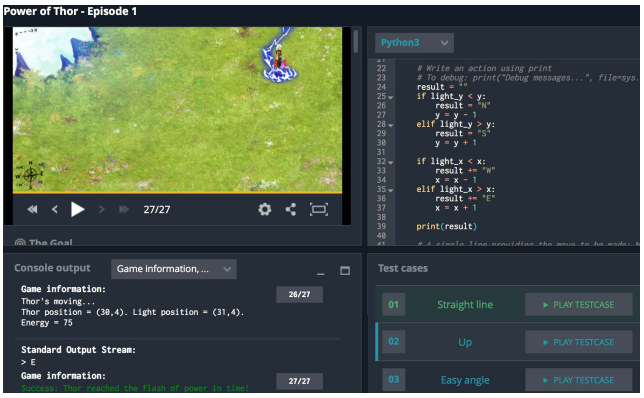

Fig. 1. CodinGame [2] in action: Players write code to control the actions in a game scenario; in this example, Thor has to retrieve the flash of power. Players have an explicit test suite (in the bottom right of the screenshot) and they can run individual tests or the whole test suite on their code.

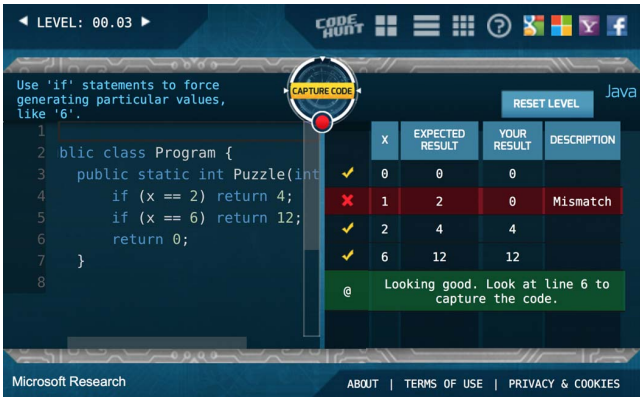

Fig. 2. Code Hunt [12] in action: Players write code for an unknown functionality, and receive feedback on how their code compares to the target function in terms of passing and failing tests.

tend to focus on development rather than testing. Although there is a trend towards dedicated courses on testing, this does not establish testing as an integrated part of any software development process, and does not achieve that testing is perceived rewarding and fun [15, 28]. There are some notable exceptions, such as attempts to teach programming in a test-driven way [26, 27], and to apply gamifying to the testing process [9, 20]. However, there are few attempts at integrating testing into school education [16]. Outside of higher education, testing is taught as part of professional accreditation (e.g., ISTQB[3]), where the taught material is similar to that taught in university testing courses [43], and thus is also disconnected from programming education.

Gamification is now common in programming education, and by considering some of these coding games we can see the potential for including testing in the gameplay: For example, CodeHunt [12], based on the earlier Pex4Fun [44], is an online game that integrates coding and test generation. While the aim is programming education rather than testing

[3]http://www.istqb.org

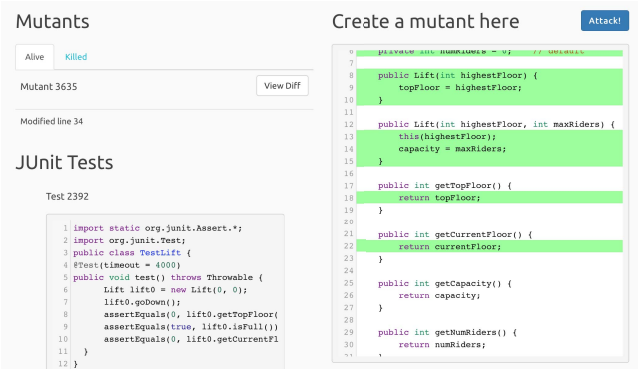

Fig. 3. Code Defenders [39] attacker view: Attackers create mutants (i.e., artificial faults that are as difficult to detect as possible) by editing the source code, while defenders write unit tests to avert these attacks.

education, there is a clear relation to testing, with tests being a part of the gameplay (see Figure 2). Other examples include CodinGame [2], where the player action is again writing code, but tests play a direct part in the gameplay, as players can check their code against individual tests (see Figure 1).

An example for a game that goes a step further and focuses on testing is Code Defenders [39], which is based on the idea of mutation testing. At the centre of the game is a program under test, and players take on one of two roles: *Attackers* aim to create mutant versions of the program; i.e., versions that have subtle differences. *Defenders* write test cases for the program under test, with the aim to detect as many as possible of the mutants written by the attackers. The game is driven by a scoring system, where attackers receive points for mutants that are not detected by the tests, and defenders receive points for detecting mutants. A special scenario is presented by equivalent mutants: Since a mutant can be semantically equivalent (i.e., there exists no test that can detect the mutant), defenders can claim equivalence, in which case the onus is on the attacker to prove non-equivalence by providing a test, or by accepting the equivalence (and losing the points for that mutant). Code Defenders can be played as a round-based two player game, where attacks and defence alternate, or as multi-player game, where teams of attackers and defenders compete. Initial experiments of the two-player duel mode in classroom settings has shown promising results [40], and we have outlined how the game can be adapted to cover aspects of the standard testing curriculum (e.g., coverage criteria) [11].

More generally, gamification is well suited in an educational setting to better engage students with software testing. This has been exemplified by Elbaum et al. [20], who developed Bug Hunt to let students apply different testing techniques to solve challenges (see Figure 4), and Bell et al. [9], who used storylines and quests to gradually introduce students to testing without explicitly telling them. However, there is much potential to improve software testing education further by developing and integrating more specific testing games and gamification approaches that cover all aspects of the testing curriculum.

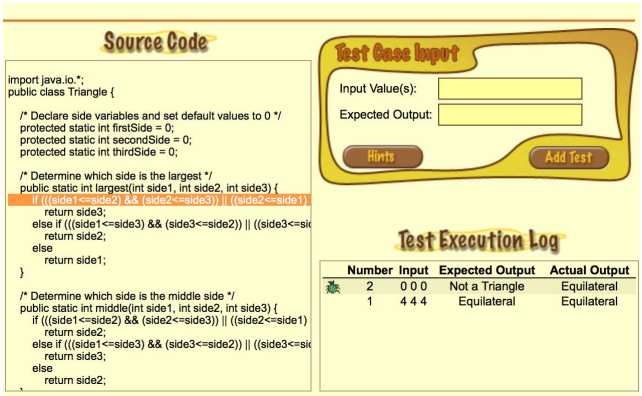

Fig. 4. Bug Hunt [20]: Players are given a textual specification and the source code of a program under test (the triangle classification program in this example), and have to provide test inputs and expected outputs to identify target bugs.

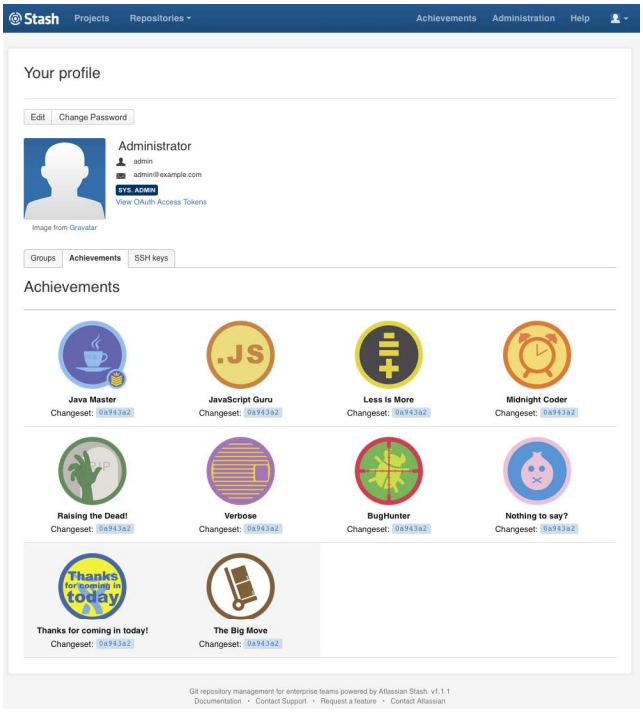

Fig. 5. Stash Badgr [3] badges for committing code to repositories.

III. Gamification of Software Testing Practice

In the context of general software engineering, there have been a number of attempts at using gamification to increase the motivation and performance of people participating in software engineering activities [37], such as removing static analysis warnings [7] or committing changes often to version control [42]. For example, Badgr for Stash (Figure 5) uses the standard gamification concept of badges to reward developers based on their interactions with the version control system. Microsoft Visual Studio has integrated an achievement system (Figure 6) that rewards professional usage of the IDE. There

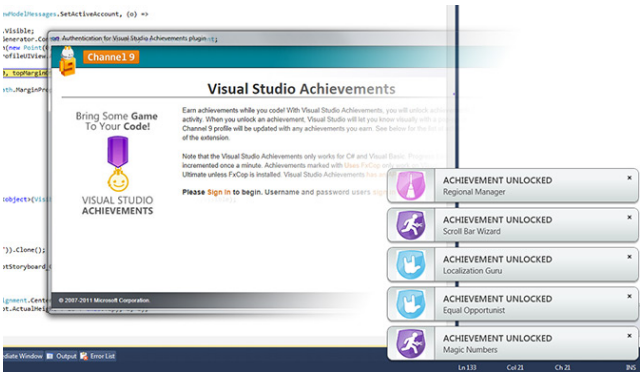

Fig. 6. Visual Studio Achievements [5] reward professional usage of the IDE.

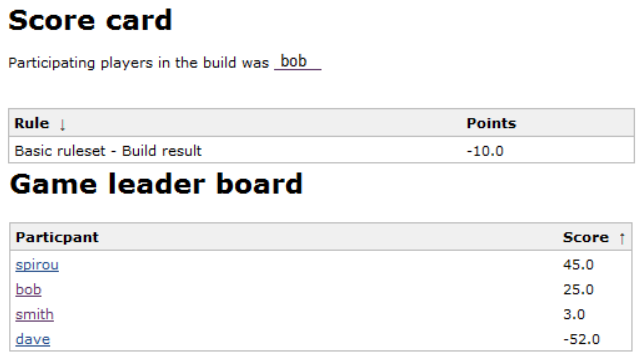

Fig. 7. The Jenkins CI Game [4] awards points for commits that fix tests or warnings, and deducts points for failing tests, warnings, and build breakage..

have also been testing-specific reward mechanisms such as the Python unit test achievements system [6]. Continuous Integration generally offers potential for games, as exemplified by the Jenkins CI Game (Figure 7), where players get awarded points for making tests pass and removing static analysis warnings, and lose points for introducing static analysis warnings, making tests fail, or breaking the build. Players can compare their ranking with the other developers on the Jenkins CI web interface.

These gamification approaches reward testing to some degree, but there is substantial potential for a deeper integration. However, one particular concern about the current state of gamification in software engineering [8] is that it is limited to basic reward aspects and neglects other game elements such as framing. A more holistic game experience could, for example, integrate the Code Defenders game [39] into the development process: Development teams would be the defenders for all the code they produce, and the attacker team for code written by other teams.

The CodeFights [1] platform exemplifies an alternative scenario of where gamification could be helpful: In CodeFights, players compete over different types of coding challenges, and the specific objective is recruiting: Companies can pay to get in contact with top players. While the game is coding-centric, tests are involved (see Figure 8, where code can be checked

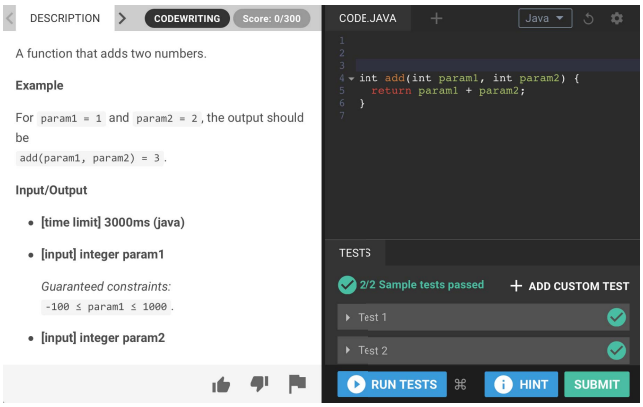

Fig. 8. CodeFights [1] lets players compete in different types of coding puzzles with the aim to provide recruitment opportunities; tests are not the focus, but still part of the game.

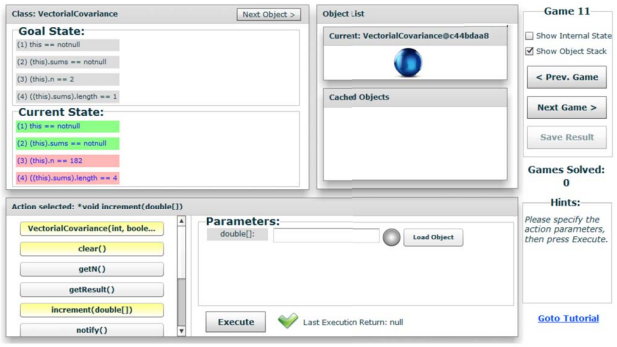

Fig. 9. Puzzle-based Automated Testing [14] lets players mutate objects to help automated test generation.

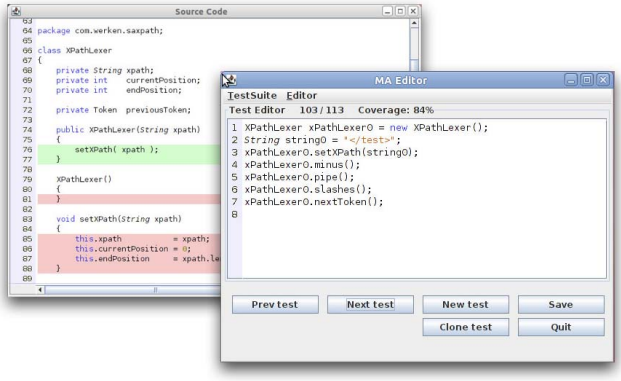

Fig. 10. Semi-automatic Search-Based Test Generation [36] queries the user for input when the search algorithm is stuck in a local optimum.

against individual tests similar to the earlier CodinGame example), and there would be potential to extend the idea to more testing-specific games.

IV. Testing Games for Crowdsourcing

The third main area of application for games in software testing considered in this paper are serious games [34], or "games with a purpose" [48]. The aim of these games is to serve some external purpose that is not necessarily connected with the aims of the individual players; for example, each player may only contribute a small part of an overall solution. Games such as Recaptcha [52], Duolingo [47], Peekaboom [51], ESP [46], Petch [49], Verbosity [50], or Phrase-slap [24] have been highly successful, attracting large numbers of players who spend vast amounts of time playing (e.g., a total of 500,000 human hours for the Peekaboom [51] game) and thus solve large numbers of computational problems (e.g., 50 million image labels for the ESP game [46]).

Although crowdsourced testing (crowd testing) [19] is now a common practice in industry, the focus is mainly testing of mobile and web applications [55], and does not use gamification. Testing has also been considered [45] as part of a general collaborative and crowdsourced approach to software engineering [33], and for some more specialised aspects of testing, such as the elicitation of test oracles for automatically generated tests with respect to API documentation using the microtasking platform Amazon Mechanical Turk [35]. Crowdsourcing with gamification has been used in the Crowd Sourced Formal Verification program of the US Defense Advanced Research Projects Agency (DARPA), which produced a number of games related to formal verification tasks: Dietl et al. [18] gamified the verification of basic dataflow and type properties in a game called Pipe Jam, Fava et al. [21] gamified the process of invariant generation in the Binary Fission game, and Logas et al.[31] gamified the process of loop summarisation in the Xylem game.

Test generation, more specifically, has been targeted by Chen and Kim [14], who designed a game where humans solve puzzles that represent object mutation or constraint solving problems (Figure 9). Pavlov and Fraser [36] demonstrated that some of the challenges faced by search-based test generators can be overcome by including human intelligence by using an interactive genetic algorithm in the EvoSuite tool (Figure 10), although they did not use any type of gamification. These approaches are essentially based on puzzle-solving, and do not make use of competitive or cooperative elements. Thus, there is substantial potential to develop these approaches further for applications in crowdsourced test generation, interactive search-based testing, or crowdsourcing of other testing aspects such as the test oracle problem. We have used Code Defenders, described in Section II, in a crowdsourcing scenario [29]: Teams of attackers faced teams of defenders, and competed over a set of open source classes. The tests resulting from these games are stronger (i.e., higher code coverage and mutation scores) than those produced by automated test generation tools, and the mutants resulting from the games are stronger (i.e., fewer trivial or identical mutants) than those produced by automated mutation analysis tools.

V. Conclusions

Software testing is essential to ensure that we produce software of sufficiently high quality, but testing is challenging and often neglected. Gamification offers an opportunity to change this: By changing how testing is taught, we can fundamentally influence the attitude of future software engineering towards testing. With gamification, developers can be engaged into testing activities Finally, by turning software testing into games with a purpose, we can solve complex testing problems using crowdsourcing and human intelligence. Initial work in this area has shown promising results, and the challenge going forward now lies in developing new game concepts around software testing and integrating them into teaching, practice, and crowdsourcing.

Acknowledgements

This work is supported by EPSRC project EP/N023978/1.

References

[1] CodeFights – Test Your Coding Skills. https://codefights.com. Accessed: 3.4.2017.

[2] CodinGame – Play with Programming. https://www.codingame.com. Accessed: 3.4.2017.

[3] Gamifying stash with badgr commit, code, achieve. https://www.atlassian.com/blog/archives/gamifying-stash-building-the-badgr-add-on. Accessed: 3.4.2017.

[4] Jenkins CI Game Plugin. https://github.com/jenkinsci/ci-game-plugin. Accessed: 3.4.2017.

[5] Microsoft Channel 9 – Achievements. https://channel9.msdn.com/achievements. Accessed: 3.4.2017.

[6] Unit Testing Achievements. http://exogen.github.io/nose-achievements/. Accessed: 3.4.2017.

[7] S. Arai, K. Sakamoto, H. Washizaki, and Y. Fukazawa. A gamified tool for motivating developers to remove warnings of bug pattern tools. In *Int. Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 37–42. IEEE, 2014.

[8] T. Barik, E. Murphy-Hill, and T. Zimmermann. A Perspective on Blending Programming Environments and Games: Beyond Points, Badges, and Leaderboards. In *Visual Languages and Human-Centric Computing*, 2016.

[9] J. Bell, S. Sheth, and G. Kaiser. Secret ninja testing with halo software engineering. In *Int. Workshop on Social software engineering*, pages 43–47. ACM, 2011.

[10] M. Beller, G. Gousios, A. Panichella, and A. Zaidman. When, how, and why developers (do not) test in their ides. In *Joint Meeting on Foundations of Software Engineering*, pages 179–190. ACM, 2015.

[11] J. M. R. Benjamin Clegg and G. Fraser. Teaching software testing concepts using a mutation testing game. In *Proc. of the International Conference on Software Engineering Companion (ICSE) 2017*, ICSE'17, 2017. to appear.

[12] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. de Halleux. Code hunt: Experience with coding contests at scale. *ACM/IEEE Int. Conference on Software Engineering (ICSE)(JSEET track)*, pages 398–407, 2015.

[13] D. Carrington. Teaching software testing. In *Proceedings of the 2nd Australasian Conference on Computer Science Education*, pages 59–64. ACM, 1997.

[14] N. Chen and S. Kim. Puzzle-based automatic testing: bringing humans into the loop by solving puzzles. In *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*, pages 140–149. ACM, 2012.

[15] H. B. Christensen. Systematic testing should not be a topic in the computer science curriculum! In *ACM SIGCSE Bulletin*, volume 35, pages 7–10. ACM, 2003.

[16] J. Collofello and K. Vehathiri. An environment for training computer science students on software testing. In *Frontiers in Education, 2005. FIE'05. Proceedings 35th Annual Conference*, pages T3E–6. IEEE, 2005.

[17] W. P. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice Hall, 2011.

[18] W. Dietl, S. Dietzel, M. D. Ernst, N. Mote, B. Walker, S. Cooper, T. Pavlik, and Z. Popović. Verification games: Making verification fun. In *Workshop on Formal Techniques for Java-like Programs (FTfJP)*, pages 42–49. ACM, 2012.

[19] E. Dolstra, R. Vliegendhart, and J. Pouwelse. Crowdsourcing gui tests. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 332–341. IEEE, 2013.

[20] S. Elbaum, S. Person, J. Dokulil, and M. Jorde. Bug hunt: Making early software testing lessons engaging and affordable. In *ACM/IEEE Int. Conference on Software Engineering (ICSE)*, pages 688–697, 2007.

[21] D. Fava, D. Shapiro, J. Osborn, M. Schäef, and E. J. Whitehead Jr. Crowdsourcing program preconditions via a classification game. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1086–1096. ACM, 2016.

[22] R. L. Glass, R. Collard, A. Bertolino, J. Bach, and C. Kaner. Software testing and industry needs. *IEEE Software*, 23(4):55, 2006.

[23] P. Henriksen and M. Kölling. Greenfoot: combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 73–82. ACM, 2004.

[24] C.-J. Ho, T.-H. Chang, and J. Y.-j. Hsu. Photoslap: A multi-player online game for semantic annotation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1359. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[25] L. Inozemtseva and R. Holmes. Coverage is not strongly correlated with test suite effectiveness. In *ACM/IEEE Int. Conference on Software Engineering (ICSE)*, pages 435–445. ACM, 2014.

[26] D. S. Janzen and H. Saiedian. Test-driven learning: intrinsic integration of testing into the cs/se curriculum.

In *ACM SIGCSE Bulletin*, volume 38, pages 254–258. ACM, 2006.

[27] C. G. Jones. Test-driven development goes to school. *J. Comput. Sci. Coll.*, 20(1):220–231, 2004.

[28] E. L. Jones. An experiential approach to incorporating software testing into the computer science curriculum. In *31st Frontiers in Education Conference*, volume 2, pages F3D–7. IEEE, 2001.

[29] B. C. José Miguel Rojas, Thomas White and G. Fraser. Code Defenders: Crowdsourcing effective tests and subtle mutants with a mutation testing game. In *Proc. of the International Conference on Software Engineering (ICSE) 2017*, 2017. to appear.

[30] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg. The BlueJ system and its pedagogy. *Computer Science Education*, 13(4):249–268, 2003.

[31] H. Logas, J. Whitehead, M. Mateas, R. Vallejos, L. Scott, D. Shapiro, J. Murray, K. Compton, J. Osborn, O. Salvatore, et al. Software verification games: designing xylem, the code of plants. *Foundations of Digital Games*, pages 1–8, 2014.

[32] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, 40(1):367–371, 2008.

[33] K. Mao, L. Capra, M. Harman, and Y. Jia. A survey of the use of crowdsourcing in software engineering. *RN*, 15(01), 2015.

[34] D. R. Michael and S. L. Chen. *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade, 2005.

[35] F. Pastore, L. Mariani, and G. Fraser. Crowdoracles: Can the crowd solve the oracle problem? In *IEEE Int. Conference on Software Testing, Verification and Validation*, pages 342–351. IEEE, 2013.

[36] Y. Pavlov and G. Fraser. Semi-automatic search-based test generation. In *IEEE Int. Conference on Software Testing, Verification and Validation*, pages 777–784. IEEE, 2012.

[37] O. Pedreira, F. Garca, N. Brisaboa, and M. Piattini. Gamification in software engineering – a systematic mapping. *Information and Software Technology (IST)*, 57:157 – 168, 2015.

[38] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[39] J. M. Rojas and G. Fraser. Code Defenders: A Mutation Testing Game. In *Int. Workshop on Mutation Analysis (ICSTW)*, pages 162–167. IEEE, 2016.

[40] J. M. Rojas and G. Fraser. Teaching Mutation Testing using Gamification. In *European Conference on Software Engineering Education (ECSEE)*, 2016.

[41] T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Communications of the ACM*, 44(6):103–108, 2001.

[42] L. Singer and K. Schneider. It was a bit of a race: Gamification of version control. In *Int. Workshop on Games and Software Engineering (GAS)*, pages 5–8. IEEE, 2012.

[43] The Joint Task Force on Computing Curricula. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Technical report, New York, NY, USA, 2004.

[44] N. Tillmann, J. De Halleux, T. Xie, and J. Bishop. Pex4fun: Teaching and learning computer science via social gaming. In *Conference on Software Engineering Education and Training*, pages 90–91. IEEE, 2012.

[45] Y.-H. Tung and S.-S. Tseng. A novel approach to collaborative testing in a crowdsourcing environment. *Journal of Systems and Software*, 86(8):2143–2153, 2013.

[46] L. Von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

[47] L. von Ahn. Duolingo: learn a language for free while helping to translate the web. In *Int. Conference on Intelligent User Interfaces (IUI)*, pages 1–2. ACM, 2013.

[48] L. Von Ahn and L. Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[49] L. Von Ahn, S. Ginosar, M. Kedia, R. Liu, and M. Blum. Improving accessibility of the web with a computer game. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 79–82. ACM, 2006.

[50] L. Von Ahn, M. Kedia, and M. Blum. Verbosity: a game for collecting common-sense facts. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78. ACM, 2006.

[51] L. Von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64. ACM, 2006.

[52] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. Recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[53] L. Werner, S. Campe, and J. Denner. Children learning computer science concepts via Alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 427–432. ACM, 2012.

[54] A. Wilson and D. C. Moffat. Evaluating scratch to introduce younger schoolchildren to programming. *Proceedings of the 22nd Annual Psychology of Programming Interest Group (Universidad Carlos III de Madrid, Leganés, Spain*, 2010.

[55] S. Zogaj, U. Bretschneider, and J. M. Leimeister. Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. *Journal of Business Economics*, 84(3):375–405, 2014.