Constructing Coding Duels in Pex4Fun and Code Hunt

Nikolai Tillmann, Jonathan de Halleux
Microsoft Research
Redmond, WA, USA
nikolait@microsoft.com
jhalleux@microsoft.com

Tao Xie
University of Illinois at Urbana-Champaign
Urbana, IL, USA
taoxie@illinois.edu

Judith Bishop
Microsoft Research
Redmond, WA, USA
jbishop@microsoft.com

ABSTRACT

Pex is an automatic white-box test-generation tool for .NET. We have established that games can be built on top of Pex to open the tool to students and to the general public. In particular, we have released Pex4Fun (`www.pexforfun.com`) and its successor Code Hunt (`www.codehunt.com`) as web-based educational gaming environments for teaching and learning programming and software engineering. In Pex4Fun and Code Hunt, the main game type is a coding duel, where a player writes code in a method to achieve the same functionality as the secret method implementation, based on feedback provided by the underlying Pex tool. Players iteratively modify their code to match the functional behavior of the secret method. The scope of duels extends from the simplest one-line method to those including advanced concepts such as writing parameterized unit tests and code contracts. We have also used the game type for competitions with thousands of players, and have found that it differentiates well between beginners and top coders. This tool demonstration shows how coding duels in Pex4Fun and Code Hunt can be constructed and used in teaching and training programming and software engineering.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Symbolic execution*

General Terms

Languages, Experimentation

Keywords

Symbolic execution, educational platforms, gaming for learning

1. INTRODUCTION

To offer an online platform for learning programming and software engineering, in 2010, Microsoft Research released Pex4Fun [12, 14] (`www.pexforfun.com`), built on top of Pex [10], an automatic white-box test-generation tool for .NET. Pex4Fun is an interactive-gaming-based platform for .NET programming languages such as C#, Visual Basic, and F#. It targets teachers, students, enthusiasts, and even software practitioners. The core type of Pex4Fun games is a *coding duel*, where the player has to solve a particular coding problem. In a coding duel, a player writes code in a method to achieve the same functionality as a secret method implementation, based on feedback provided by the underlying Pex tool. The provided feedback is in the form of input-output test cases, some of which indicate functionality matching, while the rest of which indicate mismatching.

Pex4Fun quickly gained popularity in the programming community: since it was released to the public in June 2010, the number of clicks of the "Ask Pex!" button (indicating the attempts made by players to solve games at Pex4Fun) reached over 1.5 million (1,517,859) as of June 9, 2014. Pex4Fun has provided a number of open virtual courses (similar to MOOCs in spirit) including learning materials along with games used to reinforce students' learning. The level of programming ability required to solve a duel varies enormously from almost no knowledge of coding (e.g., a duel where the secret implementation could be `return -x;`) to advanced training on unit testing, as described in Section 4. Over several years, we have shown that Pex4Fun, with its gamification and tutorials, was an effective learning platform [11].

Although Pex4Fun was and is very popular, we wanted to explore more deeply the gamification aspect of the platform. It was clear from users' comments that they were prepared to stay with the gaming there because it presented a challenge. We felt that we could make Pex4Fun more fun and appealing to a wider audience. Thus, Code Hunt was born.

Code Hunt [9,11] (`www.codehunt.com`) was released in early 2014. It evolved from Pex4Fun by incorporating more gaming aspects so as to be more engaging. With coding duels as the game type, Code Hunt organizes games in a series of worlds, sectors, and levels, which become increasingly challenging. In each level, the player has to discover a secret code fragment (i.e., coding duel) and write code for it. As the game develops, the underlying Pex tool gives customized progress feedback to the player via the generated unit tests (called clues). When the player's code achieves the same result as the secret implementation, Code Hunt flashes "Captured Code" and provides a score to the player based on how good the code was. The game also has sounds and a leaderboard to keep the player engaged. Other improvements for Code Hunt are that it offers Java as a supported language (via a source code translator) and it runs on Microsoft Azure, making it scalable to a large number of simultaneous users. Currently we have over a million users, and the cloud is kept active with up to 20 cores at peak time.

In the rest of this paper, Section 2 presents background information (including DSE, parameterized unit testing, and code con-

ISSTA '14, July 21-25, 2014, San Jose, CA, USA

tracts). Section 3 presents the game type of coding duel. Section 4 describes how to construct teaching and learning materials based on coding duels. Section 5 reports on our experience with using coding duels for competitions and Section 6 concludes the paper.

2. BACKGROUND

Dynamic Symbolic Execution (DSE). Pex is a test-generation tool based on Dynamic Symbolic Execution (DSE) [4], which realizes symbolic execution [5] by leveraging runtime information from concrete executions. Through iterations, DSE systematically increases code coverage such as block or branch coverage. In each iteration, DSE executes the program under test with a test input, which could be a default or randomly generated input in the first iteration or an input generated in one of the previous iterations. During the execution of the program under test, DSE performs symbolic execution in parallel to collect symbolic constraints on program inputs obtained from predicates in branch statements along the execution. The conjunction of all symbolic constraints along an executed path is called the path condition. Then DSE flips a branching node in the executed path to construct a new path that shares the prefix to the node with the executed path, but then deviates and takes a different branch.

Parameterized Unit Testing. A key methodology that Pex supports is parameterized unit testing [13]. Parameterized unit testing extends previous industry practice by allowing unit test methods to have parameters. This extension serves two main purposes. First, parameterized unit tests are specifications of the behavior of the methods under test: not only exemplary arguments to the methods under test, but also ranges of such arguments. Second, parameterized unit tests describe a set of traditional unit tests that can be obtained by instantiating the methods of the parameterized unit tests with given argument-value sets. An automatic test-generation tool such as Pex can be used to generate argument-value sets for parameterized unit tests.

Code Contracts. Based on the concept of Design by Contract [7], Code Contracts [6] from Microsoft Research allows developers to write design-by-contract specifications. These Code Contracts take the form of preconditions, postconditions, and object invariants. Preconditions specify conditions that must hold before a method can be executed. Postconditions specify conditions that must hold after a method is completed. Object invariants specify conditions that the objects of the class should always satisfy. Object invariants are checked for every non-static, non-private method's entry and exit, and for every non-private constructor's exit. Class invariants can be treated as preconditions and postconditions for these methods.

3. OVERVIEW OF CODING DUELS

The theory behind creating coding duels in Pex4Fun and Code Hunt is the same, and is described in this section and the next section. Currently, only Pex4Fun provides the capability for players to add duels, and thus the description here is written for Pex4Fun. In Code Hunt, duels (known as levels) are created by experts and uploaded behind the scenes. A new editor to enable the same capability as in Pex4Fun is under construction.

A game creator (who could be any user around the world) can create a coding duel. A duel consists of two methods with the same method signature and return type[1]. One of these two methods is the secret (golden) implementation, which is not visible to the player. The other is the player implementation, which is visible to the player and can be empty implementation or faulty. The player implementation can include optional comments to give the player some hints in order to reduce the difficulty level of the duel.

After a player selects a coding-duel game to play, the player's winning goal is to modify the player implementation (visible to the player) to make its behavior (in terms of the method inputs and results) to be the same as the secret implementation (not visible to the player). Clearly, without any feedback or help, the player has no way to guess how the secret implementation would behave. The player can start getting feedback by clicking the button "Ask Pex" to request under what sample method input(s) the player implementation and the secret implementation have (1) the same method result and (2) different method results.

Pex4Fun leverages Pex to generate such feedback and determine whether the player wins the game: the player wins the game if Pex cannot generate any method input to cause the player implementation and the secret implementation to have different method results.

4. CONSTRUCTION OF CODING DUELS

There are five steps for a user to create a coding duel. First, the user logs into the Pex4Fun platform. Second, the user writes a secret implementation as a `Puzzle` method that takes input arguments and produces a result. Third, the user creates a coding duel by clicking a button "Turn This Puzzle Into A Coding Duel" (appearing after the user clicks "Ask Pex!"). Fourth, the user edits the player implementation (i.e., program text visible to players) by clicking the coding duel Permalink URL, which opens the coding duel, and by filling in a slightly useful outline of the implementation (with optional comments) that players will eventually complete. Fifth, the user publishes the coding duel after the user finishes editing the visible `Puzzle` method text by clicking the "Publish" button.

A `Puzzle` method can be turned into a coding duel only if it fulfills two main requirements: (1) it must have a non-void return type, so that the behavior of the secret implementation and the player implementation can be compared using their return values; (2) the `Puzzle` method must have at least one parameter, so that Pex can generate argument values for it.

The game creator has great flexibility in controlling the difficulty of a coding duel. He or she can vary (1) the complexity of the secret implementation; (2) the similarity level of the player implementation (visible to players) to the secret implementation; (3) the strength of the hints given in code comments in the player implementation.

We next use examples to illustrate how one can construct coding duels as teaching and training materials for two topics: parameterized unit testing and writing code contracts; these examples were used in teaching a graduate software engineering course [12].

4.1 Teaching and Training Parameterized Unit Testing

In a coding duel, the player is asked "For each coding duel, you need to complete the given incomplete parameterized unit test to match the secret parameterized unit test for testing the `UBIntStack` class that implements a bounded stack that holds unique integer elements." The initial player implementation of the coding duel is shown in Figure 1 and its solution (the secret implementation) is shown in Figure 2. In particular, in the player implementation given to the player, we already include test-scenario setup (including some assumptions) and test oracles (i.e., assertions), and the player is asked to fill in the middle part of the parameterized unit test, which is the `Puzzle` method itself. As shown in Figure 2, the middle part includes additional test-scenario setup (including some

[1]The method signature of a coding duel must have at least one input parameter. The return type of a coding duel must not be void.

```
public static string Puzzle(int[] elems, int capacity, int elem)
{
    if (capacity <= 0) return "Assumption Violation!";
    if (elems == null) return "Assumption Violation!";
    if (elems.Length > (capacity + 1)) return "Assumption Violation!";
    UBIntStack s= new UBIntStack(capacity);
    for (int i = 0; i < elems.Length; i++)
       s.Push(elems[i]);
    int origSize = s.GetNumberOfElements();
 //Please fill in below test scenario on the s stack
 //including necessary assumptions (no additional assertions needed)


 //The lines below include assertions to assert the program behavior
    PexAssert.IsTrue(s.GetNumberOfElements() == origSize + 1);
    PexAssert.IsTrue(s.Top() == elem);
    PexAssert.IsTrue(s.IsMember(elem));
    PexAssert.IsTrue(!s.IsEmpty());
    return "s.GetNumberOfElements():" + s.GetNumberOfElements().ToString() + "; "
      + "s.Top():" + s.Top().ToString() + "; "
    + "s.IsMember(elem):" + s.IsMember(elem).ToString() + "; "
     + "s.IsEmpty():" + s.IsEmpty()  + "; ";
}
```

Figure 1: An example coding duel for teaching parameterized unit testing

```
public static int Puzzle(int pref_id, string value)
{
    ....
 //Please fill in below test scenario on the s stack
 //including necessary assumptions (no additional assertions needed)
    if (s.IsMember(elem)) return "Assumption Violation!";
    if (s.GetNumberOfElements() >= s.MaxSize()) return "Assumption Violation!";
    s.Push(elem);
    ....
}
```

Figure 2: The solution to the example coding duel shown in Figure 1

assumptions) and the method under test (i.e., the `Push` method).

The comments in the player implementation inform the player *partially* what exactly they need to accomplish in terms of the secret-implementation functionality. Thus the player needs to do some "guessing" based on the feedback given by Pex4Fun. For example, the comments in the player implementation do not inform the player which method of `UBIntStack` is the method under test (it is `Push`) or what additional test-scenario setup is needed. In the example coding duel in Figure 1, additional test-scenario setup includes that the `elem` being pushed is not in the stack already and the stack is not full. The player needs to "guess" such information based on the feedback given by Pex4Fun.

4.2 Teaching and Training Code Contracts

In an example coding duel, the player is asked to "translate the natural-language requirement above the method below to be in code contracts." The initial player implementation of the coding duel is shown in Figure 3 and its solution (the secret implementation) is shown in Figure 4. To construct this coding duel, we extracted the natural-language requirements (used in the coding duel) from real-world evaluation subjects (API documents) used in related previous work [8]. In particular, two code contracts are needed for the requirements, as shown in Figure 4.

The comments in the player implementation inform the player what the player needs to accomplish in terms of the secret implementation's functionality. The player does not have to "guess" based on the feedback given by Pex4Fun. Such design style is in contrast to many coding duels in Pex4Fun, where "guessing" is heavily involved and no or few hints are given to players.

5. CODING COMPETITIONS

An application of Pex4Fun and Code Hunt that is gaining in popularity is their use in competitions. There are three main reasons: (1) the gaming aspect is attractive, (2) there are clear winning criteria, and (3) cheating is not possible. Both platforms have been used for competitions with success.

In May 2011, Microsoft Research hosted a competition on solving coding duels[2] at the 2011 International Conference on Software Engineering (ICSE 2011). The ICSE 2011 coding-duel competition received 7,000 Pex4Fun attempts, 450 duels completed, and 28 participants (though likely more, since not everyone logged in to enter the competition). Recently, Pex4Fun inspired a new competition form [1] in the 2013 International Conference on Functional Programming (ICFP 2013) Programming Contest[3]. Competing entirely over the Internet, more than 300 participating teams of programmers from around the world were asked to complete a series of programming tasks, using any programming languages and tools they desired, to address an extremely challenging scenario in program synthesis. Results were assessed using Microsoft Research's Z3 [3] running on Windows Azure to compare submitted solutions to actual solutions to determine correctness, in a similar way as coding duels in Pex4Fun. Over the competition's 72 hours, Z3 received about a million requests and successfully decided all, except about 300 problem instances, within an imposed time limit

[2] `http://research.microsoft.com/ICSE2011Contest`

[3] `http://research.microsoft.com/en-us/events/icfpcontest2013/`

```
//param:name:This name also needs to be a valid identifier,
//which is no longer than 32 characters, starting with a letter (a-z)
//and consisting of only small letters (a-z), numbers (0-9) and/or underscores.
//Can you write preconditions in code contracts for the above natural-language requirements?
public static int Puzzle(string name)
{
    return name;
}
```

Figure 3: An example coding duel for teaching and training code contracts

```
public static int Puzzle(string name)
{
    Contract.Requires(Regex.IsMatch(name.Substring(0,1),@"[a-z]"));
    Contract.Requires(Regex.IsMatch(name,@"^[a-z0-9]*$"));
    return name;
}
```

Figure 4: The solution to the example coding duel in Figure 3

of 20 seconds, the overwhelming majority within a matter of a few milliseconds.

In April 2014, we were able to test out Code Hunt at a very large competition in the Greater China Region called Beauty of Programming. In three rounds, 2,353 students scored in the game, with an average 55.7% puzzles solved across this large number. Since Code Hunt runs on Microsoft Azure, we have all the statistics. We could see that, on average, it took players 41 tries to capture the code for puzzles. However, what really interested us were the 350 top students, who solved all the puzzles, even the most difficult ones. These students needed only 7.6 tries on average to solve a puzzle, showing that Code Hunt can reliably surface the better coders. Code Hunt is now being offered for more competitions, with the only challenge being the creation of new worlds of puzzles.

6. CONCLUSION

Traditional coding exercises work from a specification. Our experiences with coding duels have shown how successful teaching and learning can be achieved over a range of educational levels using platforms that guide users to a solution for an unknown problem. Pex4Fun, and its successor, Code Hunt, are both browser-based games [2] that are attractive, scalable, and interesting as research projects because of the data that they generate. In future work, we plan to extend Code Hunt to provide hints, as well as to include a social experience where players can add duels.

Acknowledgments

Tao Xie's work is supported in part by a Microsoft Research Award, NSF grants CCF-1349666, CNS-1434582, CCF-1434596, CCF-1434590, CNS-1439481, and NSF of China No. 61228203.

7. REFERENCES

[1] T. Akiba, K. Imajo, H. Iwami, Y. Iwata, T. Kataoka, N. Takahashi, M. Moskal, and N. Swamy. Calibrating research in program synthesis using 72,000 hours of programmer time. Technical report, Microsoft Research, December 2013.

[2] J. Bishop, J. de Halleux, N. Tillmann, N. Horspool, D. Syme, and T. Xie. Browser-based software for technology transfer. In *Proc. Annual Research Conference of the South African Institute for Computer Scientists and Information Technologists (SAICSIT 2011), Industry Oriented Paper*, pages 338–340, 2011.

[3] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, pages 337–340, 2008.

[4] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005)*, pages 213–223, 2005.

[5] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[6] F. Logozzo. Practical verification for the working programmer with codecontracts and abstract interpretation. In *Proc. 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2011)*, pages 19–22, 2011.

[7] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1988.

[8] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language api descriptions. In *Proc. 34th International Conference on Software Engineering (ICSE 2012)*, pages 815–825, 2012.

[9] N. Tillmann, J. Bishop, R. N. Horspool, D. Perelman, and T. Xie. Code hunt - searching for secret code for fun. In *Proc. 7th International Workshop on Search-Based Software Testing (SBST 2014)*, pages 23–26, 2014.

[10] N. Tillmann and J. de Halleux. Pex - white box test generation for .NET. In *Proc. International Conference on Tests and Proofs (TAP 2008)*, pages 134–153, 2008.

[11] N. Tillmann, J. de Halleux, T. Xie, and J. Bishop. Code Hunt: Gamifying teaching and learning of computer science at scale. In *Proc. 1st ACM Conference on Learning at Scale (Learning at Scale 2014)*, pages 221–222, 2014.

[12] N. Tillmann, J. D. Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Proc. International Conference on Software Engineering (ICSE 2013), Software Engineering Education (SEE)*, pages 1117–1126, 2013.

[13] N. Tillmann and W. Schulte. Parameterized unit tests. In *Proc. 5th joint meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2005)*, pages 253–262, 2005.

[14] T. Xie, N. Tillmann, and J. de Halleux. Educational software engineering: Where software engineering, education, and gaming meet. In *Proc. 3rd International Workshop on Games and Software Engineering (GAS 2013)*, pages 36–39, May 2013.