Pex for Fun: Engineering an Automated Testing Tool for Serious Games in Computer Science

Nikolai Tillmann, Jonathan de Halleux (Microsoft Research)

Tao Xie (North Carolina State University)

{nikolait, jhalleux}@microsoft.com, xie@csc.ncsu.edu

Although there are various emerging serious games developed for education and training purposes, there exist few serious games for practitioners or students to improve their programming or problem-solving skills in the computer science domain. To provide an open platform for creating serious games in learning computer science, we have developed a web-based serious gaming environment called Pex for Fun, in short as Pex4Fun (http://www.pexforfun.com/), for learning critical computer science skills such as problem solving skills and abstraction skills. It works on any web-enabled device, even a smart phone. It comes with an auto-completing code editor, providing a user with instant feedback similar to the code editor in Microsoft Visual Studio. It is a cloud application with the data in the cloud, enabling a user to use it anywhere where Internet connection is available. New learners of programming could play games there created by us to master basic programming concepts such as arrays, loops, and exception handling. In fact, any user could create new games for others to play.

A large user community quickly developed after we published the Pex4Fun web during 2010 summer. Just half a year later, web-site visitors had already submitted more than 300,000 programs for Pex4Fun to judge and give feedback. Behind the scene of Pex4Fun, its underlying technology is called dynamic symbolic execution [GKS05, TD08], which has been realized by a white-box testing tool called Pex [TD08], the backbone of Pex4Fun.

Dynamic Symbolic Execution

Dynamic symbolic execution (DSE) [GKS05, TD08] is a variation of symbolic execution [KING76] and leverages runtime information from concrete executions. DSE is often conducted in iterations to systematically increase 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. DSE relies on a constraint solver to (1) check whether such a flipped path is feasible; if so, (2) compute a satisfying assignment -- such assignment forms a new test input whose execution will follow along the flipped path.

Pex: White box Unit Testing Tool for .NET

Pex (http://research.microsoft.com/projects/pex/) is an automatic white-box test generation tool for .NET, based on dynamic symbolic execution. We have integrated Pex into Microsoft Visual Studio as an add-in. Pex can generate test inputs that can be integrated with various unit testing frameworks such as NUnit and MSTest. We have applied Pex to a core component of the .NET architecture, which had already been extensively tested over five years by approximately 40 testers within Microsoft. The component is the basis for other libraries, which are used by thousands of developers and millions of end users. Pex found various issues in this core component, including a serious issue. In addition, Pex has been used in classroom teaching at different universities (such as North Carolina State University, University of Illinois at Urbana-Champaign, and University of Texas at Arlington), as well as various tutorials both within Microsoft (such as internal training of Microsoft developers) and outside Microsoft (such as invited tutorials at .NET user groups) [XDTS10]. We have also developed a large number of open source research extensions upon Pex (http://pexase.codeplex.com/).

A key methodology that Pex supports is *parameterized unit testing* [TS05], which extends the current industry practice based on closed, traditional unit tests (i.e., unit test methods without input parameters). In parameterized unit testing, unit test methods are generalized by allowing parameters to form parameterized unit tests. This generalization 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. Figure 1 shows a parameterized unit test for testing the `Add` method of a `MyHashSet` class.

```
// which values will trigger collisions in MyHashSet?
[PexMethod]
public void TestAddContains(int x, int y) {
  var s = new MyHashSet();
  s.Add(x);
  s.Add(y);
  PexAssert.IsTrue(s.Contains(x));
  PexAssert.IsTrue(s.Contains(y));
}
```

Figure 1. A parameterized unit test for testing the Add method of a `MyHashSet` class

Migrating Pex to the Web – Pex4Fun

Our original motivation for migrating Pex to the web and giving the birth to Pex4Fun was in fact to improve accessibility of Pex to a large group of target users: without being required to install Windows or Visual Studio first, anyone with an Internet browser and Internet connection around the world could experience the power and benefits of Pex with just a mouse click for instructing Pex to test some existing sample code under test or some typing in a text box within the browser and a mouse click for instructing Pex to test user-written code under test. Having such web-based version of Pex also falicitate active learning of attendees during our tutorials around Pex so that they could have convenient

hands-on experiences with Pex during the tutorial presentation, no matter what computing environments their laptops have.

Figure 2 shows the user interface of the Pex4Fun web. The middle of the main Pex4Fun web page includes a working area: an editable text box filled with some given source code. Below the working area, a user could click the "Ask Pex!" button to instruct the underlying Pex to test the source code in the working area. After Pex finishes testing the source code, the testing results provided by Pex are shown below the "Ask Pex!" button. The displayed testing results include a table of interesting input and output values, often uncovering surprising corner cases. The source code in the working area can be written in C#, Visual Basic, or F#. The working area has Intellisense support; for example, when a user hits the '.' after a name in the editable text box, the user gets code completion in the browser, just as in Visual Studio. With Intellisense support, Pex4Fun provides hints about what a player can type at the moment. Such hints make it easy to learn the syntax and libraries of the chosen programming language.

Comparing to the Visual Studio plugin of Pex, Pex4Fun incorporates a simplified version of Pex, allowing a user to use certain system libraries and types that the user defines explicitly in the source code from the working area. In addition, we impose an upper bound on Pex's testing time spent on testing the source code. Such limitations are necessary to prevent security attacks by malicious users. Otherwise, malicious users could retrieve or modify information from the system environments such as file systems of the Pex4Fun server (which is running Pex), or cause the server's CPU to hang with a non-terminating program or very complex program.

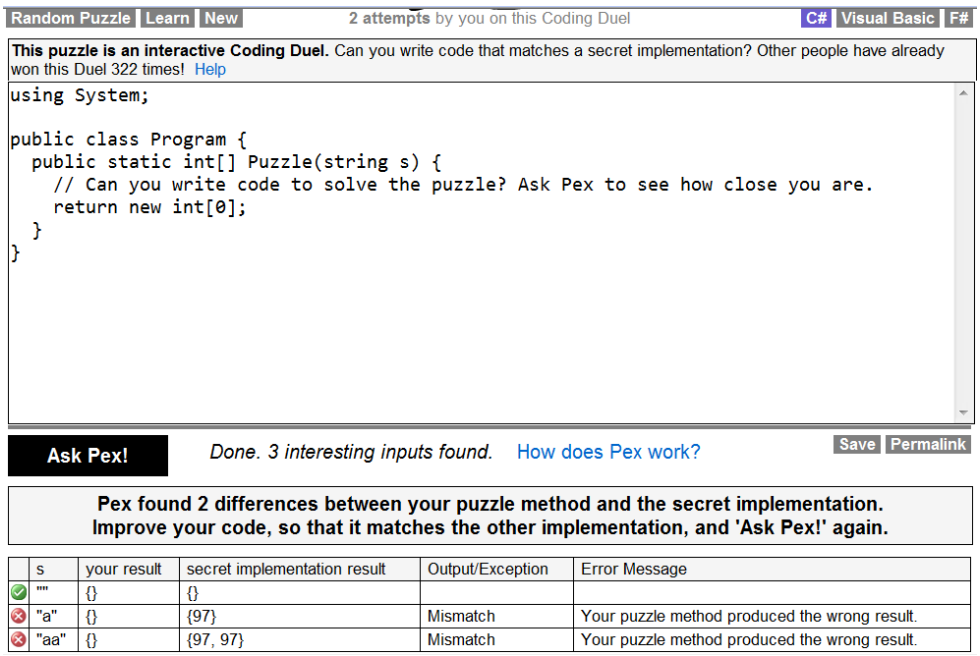

Figure 2. The user interface of the Pex4Fun web

Add Some Fun to Pex4Fun – Puzzles

Since the first release of Pex4Fun, we have prepared an initial set of given code examples called puzzles being displayed in the working area for users (players) to play with. Each puzzle has the main method

with the name `Puzzle`. After a puzzle is loaded in the working area, a player could click the "Ask Pex!" button to compile and run it. The compilation and execution happen on the Pex4Fun server; the Internet browser displays only the testing results. The main `Puzzle` method can take parameters and return values. In order to run such a `Puzzle` method, someone must provide argument values. Pex automatically finds interesting argument values by analyzing the code. These generated input argument values and produced return values are displayed as a table of input and output values below the working area. The player could click each row in the table to view more details, such as console output or stack traces.

An example puzzle is `HashSetTestAddContains` (http://www.pexforfun.com/Default.aspx?language=CSharp&sample=HashSetTestAddContains), which shows how Pex and Pex4Fun support parameterized unit testing. The parameterized unit test in this puzzle is shown in Figure 1. One can think of a parameterized unit test as a special kind of puzzle, which should never cause any bad behavior, in all code that gets called, for all possible input values. In this particular puzzle, a player has to figure out what it means to add two values to a `HashSet`. The puzzle comes with its own implementation of a `HashSet`, but one would want to write this kind of "specification" as shown in Figure 1 before even writing any implementation code (such as in Test-Driven Development [Bec03]). In fact, when preparing this puzzle, we intentionally left some bugs in the implementation. In the code comment above the `Puzzle` main method, we include "// Which values will trigger collisions in MyHashSet?" as a puzzle question for the player to answer by studying the implementation code and deriving an answer to the puzzle question. After the player clicks the "Ask Pex!" button, a table of interesting input and output values is displayed, and the table includes an argument value for triggering collisions in `MyHashSet`. Then the player could verify whether the answer that the player comes up with is consistent with the one that Pex comes up with.

Another example tricky puzzle is CreditCardNumber (http://www.pexforfun.com/Default.aspx?language=CSharp&sample=CreditCardNumber). It has been known to many that every credit card number has some kind of check digit in it. It is well documented how this check digit works. In this puzzle, given some code that implements a validity checker for credit card numbers, a player is asked to figure out an example of what a (potentially) valid credit card number is. The player may probably want to use pen and paper for some side-calculations before clicking "Ask Pex!".

To some extent, these puzzles are like a competition between a player and Pex: see whether the player could perform as well as Pex. In other words, the player is trained to have the skills of manually generating some special test inputs based on studying a code implementation. Such skills could involve program understanding skills as well as skills for simulating dynamic code behavior in the player's mind. However, we realized that these puzzles have at least three main limitations in terms of providing serious games to players. First, there is no built-in support for automatically checking whether an answer in a player's mind is consistent with the answer that Pex comes up with. In fact, there was no built-in mechanism to allow the player to write down her answer in mind. Second, a player could "cheat" the puzzle game by either (1) simply clicking "Ask Pex!" without coming up with her answer first, or (2) copying and pasting the Puzzle method implementation to Visual Studio and apply the full Visual Studio version of Pex. Third, these puzzles enable only one round of the player's interaction with the puzzle game: one click of "Ask Pex!" and then the game is over. In other words, the interactive nature in many games is lacking here. Overall, the fun level of these puzzles is not high enough, and the target skills for players to gain may be limited.

Adding More Fun to Pex4Fun – Coding Duels

To address various limitations of puzzles that came with the initial release of Pex4Fun, we have introduced *coding duels*, each of which is an interactive puzzle. In a coding duel, a player's task is to implement the `Puzzle` method to have exactly the same behavior as another secret `Puzzle` method (which is never shown to the player), based on feedback on what selected values the player's current version of the `Puzzle` method behaves differently and the same way, respectively. Figure 2 shows the screen snapshot of Pex4Fun when a player is solving a coding duel (http://pexforfun.com/default.aspx?language=CSharp&sample=ChallengeStringint). Coding duels are different from earlier-described simple puzzles, where a player's task is to simply guess what a given `Puzzle` method does or answer puzzle questions (if any embedded in code comments), by studying the code of the `Puzzle` method.

To start with a simple coding duel, a player could do the following steps. Click an example coding duel from the Pex4Fun web (http://www.pexforfun.com/Page.aspx#learn/puzzles); then the player could see a dummy implementation that does not do much. Click "Ask Pex!" to see how the dummy implementation differs from the secret implementation. Compare the dummy implementation's result to the secret implementation's result. Analyze the differences and change the code to match the secret implementation result for all input values or as many input values as you could. Click "Ask Pex!" again. Repeat this process until the player wins the coding duel or could not make any progress and give up. When a player has won the duel, the player could try other coding duels. Pex4Fun can track a player's progress such as how many attempts the player has made on a specific coding duel (as shown near the middle top of Figure 2) if the player signs in to Pex4Fun with a Windows Live™ ID. After the player signs in, the Pex4Fun web also tracks how many coding duels the player tried to win, eventually won, and which ones the player created herself. Pex4Fun also remembers the last program text that the player wrote for any particular coding duel, and allows the player to resume the playing of the coding duel any time later.

In the current implementation, Pex4Fun does not separately store the actual contents of coding duels in the cloud. Instead, the generated URL itself encodes the language and program text in a compressed form. As a result, the URLs might be quite long, depending on the size of the program text in the player's implementation. In addition, during each step in solving a coding duel, the bottom of the Pex4Fun page also includes a line to include URLs of the player's puzzle solving history and statistics in each past step when solving the coding duel such as "Puzzle history (show Permalinks): initial; +155s: 1. Ask Pex: 3 errors (9s); total time: 155s". Such puzzle solving history could allow the players to easily jump back to any of previous versions of their working implementation, like backtracking in the search space for the secret implementation.

Behind the scene, coding duels leverage the technique of dynamic symbolic execution: given a method $f(x)$ from a player (initially being a dummy method implementation specified by the coding-duel creator) and a method g (x) from the coding-duel creator (being the secret implementation), Pex explores the following meta program *h* with dynamic symbolic execution:

$$h(x) := \text{Assert}(f(x) == g(x))$$

With dynamic symbolic execution, Pex generates a test suite that is customized to both programs. Every time the player submits a new implementation version of method $f(x)$ by clicking "Ask Pex!", Pex

generates a new test suite, showing any behavior mismatches to the secret implementation, or, if there are no mismatches, indicating that the player wins the coding duel.

With respect to learning and teaching, coding duels could help train different skills of players, including but not limited to the following ones:

- Abstraction skills. The displayed list of generated input argument values to exhibit different behaviors and same behaviors, respectively, are just exemplary argument values, i.e., they are not exhaustive set of argument values for exhibiting different or same behaviors. Before figuring out how to change the player's implementation to get closer to the secret implementation, the player needs to *generalize* from the observed exemplary values and the same or different behaviors exposed by them for the player's implementation and the secret implementation.
- Problem solving or debugging skills. Solving a coding duel requires the player to conduct iterations of trials and errors. Based on the observed exemplary argument values and behaviors, the player needs to decompose the problem: grouping exemplary arguments that may exhibit the same category of different behaviors, e.g., due to lacking a branch with the conditional of `if (i>0)`. Next the player needs to come up with a hypothesized missing or corrected piece of code to make failing tests (different-behavior-exposing tests) pass and passing tests (same-behavior-exposing tests) still pass. Then the player needs to conduct an experiment to validate the hypothesis by clicking "Ask Pex!". Therefore, solving a non-trivial coding duel could involve the exercise of different problem solving skills.
- Program understanding and programming skills. If the initial dummy implementation is not that "dumb", including non-trivial code, the player needs to understand first what the dummy implementation is doing. It is obvious that the players need to have good programming skills to solve a non-trivial coding duel.

Creating and submitting new coding duels, which other players can try to win, could be easily accomplished by a user in Pex4Fun. There are five steps for a user to go through in order to create and publish coding duels.

- Step 1: sign in, so that Pex4Fun can maintain coding duels for the user.
- Step 2: write a secret implementation starting from a puzzle template where the user can write the secret implementation as a `Puzzle` method that takes inputs (i.e., method arguments) and produces an output (i.e., method return).
- Step 3: create the coding duel by clicking a button "Turn This Puzzle Into A Coding Duel" (appearing after the user clicks "Ask Pex!").
- Step 4: edit dummy 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 more useful outline of the implementation (with optional comments) that players will eventually complete.
- Step 5: publish the coding duel after the user finishes editing the visible Puzzle method text by clicking the "Publish" button.

Note that a Puzzle method could be turned into a coding duel only if the method fulfills certain requirements. For example, the Puzzle method must have a non-void return type, so that the behavior of the secret implementation and the player's implementation could be compared using their return values. The Puzzle method must have at least one parameter, so that Pex could generate argument values for the Puzzle method. The parameter and return types must not refer to any user-defined types, but just basic types (from mscorlib), or arrays of basic types, so that Pex could avoid complications

[TXTDS09] of generating method sequences for producing user-defined argument values or generating observer-method sequences for comparing the states of user-defined return objects. The Puzzle method must not mutate any of the arguments, or their fields or elements (but use the `Clone` method for arrays, or other means to avoid mutation of arguments), so that Pex could avoid complications when passing the same argument reference `x` to both the player's implementation `f` and the secret implementation `g` when testing the meta program: `h(x) := Assert(f(x) == g(x))`.

After a user creates a coding duel, besides publishing it in Pex4Fun, the user could also post the coding duel via a Permalink on our MSDN Forums for Pex (http://social.msdn.microsoft.com/Forums/en-US/pex/threads/) or via emails to share and discuss it with other people.

High flexibility on controlling the difficulty of solving a created coding duel is at hands of the coding-duel creator: (1) the complexity of the secrete implementation could vary; (2) the similarity level of the dummy implementation (visible to players) to the secrete implementation could vary; (3) the strength of the hints given in code comments in the dummy implementation could vary. These advantages in creating coding duels make Pex4Fun an attractive open platform for the community to contribute coding duel games, besides the list of built-in coding duels created by us.

Adding Social Dynamics to Pex4Fun – Ranking, Live Feed, Coding Duel of the Week

To add more fun to Pex4Fun, we have developed a number of features related to social dynamics, making games in Pex4Fun a type of social games. For example, Pex4Fun allows a player to learn what coding duels other people were already able to win (or not). For a given coding duel opened by a player, the description text box above the working area shows some statistic such as "Can you write code that matches a secret implementation? Other people have already won this Duel 477 times!" as shown in Figure 1.

Ranking of users and coding duels. A user could click the "Community**"** link on the Pex4Fun main page to see how the user's coding duel skills compare to other users. The community area lists coding duels that other users have published as well as high score lists. In the high score lists, users are ranked based on the number of coding duels solved by them or based on their points earned in the following ways, which we have designed to encourage users to participate in various Pex4Fun activities:

- One point earned for every coding duel that the user wins, except for coding duels that the user created.
- One point earned for every won coding duel that the user rates, except for coding duels that the user created. A user can rate any coding duel that the user wins as "Fun", "Boring", or "Fishy". All ratings are shared with the community.
- One point earned for every course the user registers for (a course could be created by a teacher in Pex4Fun for users to register as students).
- One tenth of a point earned for every coding duel that the user created, and somebody else attempts to win.
- One point earned for every coding duel that the user created, and somebody else wins.

Live feed. A user could click the "Live Feed" link on the Pex4Fun main page to see what coding duels other players are winning (or not) right now. Maybe someone else is trying to win a coding duel that the user has created. Figure 3 shows a screen snapshot of Live Feed of Pex4Fun.

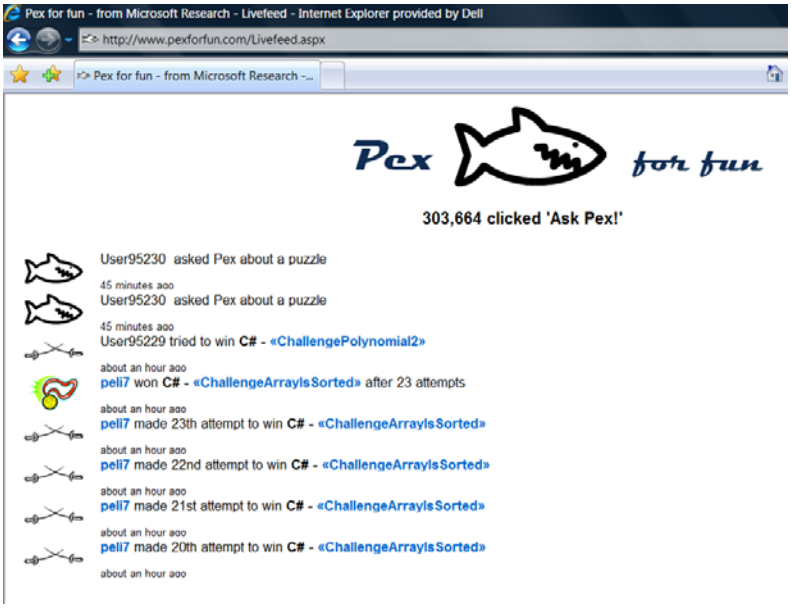

Figure 3. A Screen Snapshoot of Live Feed of Pex4Fun

Coding Duel of the Week. To promote Pex and Pex4Fun in popular social media, we have created a page for Pex and Pex4Fun at Facebook (http://www.facebook.com/PexMoles) in posting news there such as new features of Pex or Pex4Fun. As of Feb 4, 2011, 1,842 people "like" this Facebook page. More recently, to promote coding duels created by the community, every week, we randomly pick a coding duel written by a user and feature it on the Facebook page.

More Learning and More Fun to the Future

Besides the fun game aspects provided by Pex4Fun, Pex4Fun also provides an attractive open platform for a teacher to incorporate games of puzzles and coding duels when teaching a course. A teacher could create a virtual course with organized topic pages that include tutorial-style texts and games of puzzles and coding duels. We are creating more well-organized topic pages and courses for helping teachers in effectively incorporating Pex4Fun in teaching undergraduate or K-12 computing courses. In addition, we are exploring effective virtual tutoring supports to student players for handholding them to solve a coding puzzle when they face challenges in solving them independently. Finally, engineering more fun into Pex4Fun is always our continuous effort in ongoing and future work.

References

[Bec03] K. Beck. Test Driven Development: By Example. Addison-Wesley, 2003.
[GKS05] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In Proc. PLDI, pages 213–223, 2005.
[KING76] J. C. King. Symbolic Execution and Program Testing. Communications of the ACM, 19(7):385–394, 1976.
[TXTDS09] S. Thummalapenta, T. Xie, N. Tillmann, P. de Halleux, and W. Schulte. MSeqGen: Object-Oriented Unit-Test Generation via Mining Source Code. In Proc. ESEC/FSE, pp. 193-202, August 2009.
[TS05] N. Tillmann and W. Schulte. Parameterized unit tests. In Proc. ESEC/FSE, pages 253–262, 2005.

[TD08] N. Tillmann and J. de Halleux. Pex-white box test generation for .NET. In Proc. TAP, pages 134–153, 2008.
[XDTS10] T. Xie, J. de Halleux, N. Tillmann, and W. Schulte. Teaching and Training Developer-Testing Techniques and Tool Support. In Proc. SPLASH 2010, Educators' and Trainers' Symposium, pages 175-182, 2010.