Code Hunt: Context-Driven Interactive Gaming for Learning Programming and Software Engineering

Nikolai Tillmann[1], Jonathan de Halleux[1], Judith Bishop[1]
Tao Xie[2], R. Nigel Horspool[3], Daniel Perelman[4]
[1]Microsoft Research, Redmond, WA 98052, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[3]University of Victoria, Victoria, BC V8W 2Y2, Canada
[4]University of Washington, Seattle WA 98195, USA
Email: nikolait,jhalleux,jbishop@microsoft.com
taoxie@illinois.edu, nigelh@cs.uvic.ca, perelman@cs.washington.edu

ABSTRACT

Code Hunt is a web-based serious gaming platform for players to solve coding duels, a type of puzzle based on programming and software engineering. In Code Hunt, a player iteratively modifies code to match the functional behavior of a secret code segment. The functional behavior is defined based on unit test cases shown as input-output pairs. To guide players to modify the code segment, Code Hunt provides feedback based on test generation through the Pex engine. In Code Hunt, the way of writing code is very different from the way in traditional software development since there are no known requirements (either informally/formally documented or existing in developers' mind); the game aspect in Code Hunt is essentially re-engineering from sample expected behaviors observed from generated test cases. Various types of context exist in Code Hunt including the duel and the test cases, as well as the player's history and any hints that are given. In this position paper, we discuss how such context assists the players to solve coding duels while offering the players learning and fun experiences.

1. CODE HUNT

Code Hunt (https://www.codehunt.com/) [5, 3] is a web-based serious gaming platform where players write code to advance through levels. Code Hunt runs in the cloud on Windows Azure, and can be played by players via any modern browser [1]. It is a significant extension of a serious gaming website Pex4Fun (http://www.pex4fun.com/) [6, 7] by instilling more fun and entertaining effects, adding hint generation, adding language support to Java, etc. A player can play the Code Hunt game by walking through a series of sectors, each of which further contains a series of levels. In each level, the player modifies the given code to solve a coding duel. Along the way of code modifications by the player, Code Hunt gives customized feedback (adaptive to the code modifications) to guide the player to make progress towards successfully solving the coding duel. Figure 1 shows the user interface of game playing in Code Hunt.

Figure 1: User interface of game playing in Code Hunt

In particular, in Code Hunt, the player is shown a player code segment (in the form of a method with at least one method argument and non-void return), as displayed on the left hand side of the screen (Figure 1), and is asked to iteratively modify such code segment to match the functional behavior of the secret code segment not visible to the player. The functional behavior is defined based on input-output pairs (with the method arguments as input and method return as output). To guide the player to modify the code segment, along the way, Code Hunt provides feedback as failing test cases (under what sample inputs the player code segment and secret code segment produce different outputs) along with passing test cases (under what sample inputs the two code segments produce the same outputs). Such feedback is displayed on the right hand side of the screen as a table. To supply such feedback, Code Hunt is based on test generation provided by Pex [4], a white-box test generation tool that uses dynamic symbolic execution (which explores feasible execution paths for achieving high code coverage).

2. TYPES OF CONTEXT IN CODE HUNT

We next describe various types of context in Code Hunt for guiding a player in solving a coding duel while offering the player learning and fun experiences. Figure 1 shows the following context types annotated with numbers.

Earlier solved coding duels (annotated as 1 in the figure). A coding duel can be designed to be dependent on a previous coding duel solved by the player already. For example, a "Factorial digit sum" problem posted on the Project Euler website (https://projecteuler.net/problem=20) can be transformed to

a coding duel C_1 with the hidden functionality of "computing the sum of the digits in the factorial number x! given number x". In addition, before this coding duel C_1 is attempted by the player, another related coding duel C_0 can be designed for the hidden functionality of "computing the factorial x! given number x". The solution for solving coding duel C_0 forms a stepping or constitute stone for solving coding duel C_1. Such type of context may be explicitly visible to the player (e.g., the coding duel C_1 initially has the player code segment displayed as the earlier player code segment written by the player for solving coding duel C_0), or may be in the player's mind in an implicit way.

The secret code segment of the coding duel (annotated as 2 in the figure). The secret code segment of the coding duel is intentionally hidden from the player, who aims to discover the functionality of the secret code segment based on feedback iteratively. Such type of context is not visible to the player before the player successfully solves the coding duel. In other words, such type of context is intentionally hidden from the player when the player is attempting to solve the coding duel.

The playing history of the coding duel (annotated as 3 in the figure). Typically the player makes a number of attempts (forming the playing history of the coding duel) before the player successfully solves the coding duel. The playing history details of the coding duel are not displayed on the player's user interface. Currently the player has to take notes of expected input-output pairs (observed over iterations) either as comments or if-statements in the player code segment or on a convenient place outside Code Hunt. Note that the expected sample input-output pairs reported by Code Hunt are customized to the code modifications made by the player, and thus are different across different iterations. Such type of context is not explicitly displayed to the player but lies in the player's mind.

The coding duel being modified by the player (annotated as 4 in the figure). When playing the Code Hunt game, iteratively the player makes an attempt by modifying the coding duel and clicking the "Capture Code" button to gather feedback. The coding duel being modified by the player is displayed on the left hand side of the screen. Note that only the latest version of the coding duel is displayed. Such type of context is explicitly visible and displayed to the player.

The input-output pairs reported to the player (annotated as 5 in the figure). The feedback that the player gathers is in the form of input-output pairs, displayed as a table on the right hand side of the screen. The input-output pairs are classified into two groups: failing test cases where the outputs of the secret segment and the player segment are different for the same inputs; passing test cases where the outputs of the two segments are the same for the same inputs. Note that only the input-output pairs produced after the latest click of "Capture Code" are displayed. As stated earlier, the expected sample input-output pairs reported by Code Hunt are different across different iterations of clicking "Capture Code"; thus, the player may want to take notes of observed expected input-output pairs. Such type of context is explicitly visible and displayed to the player.

The hint reported to the player (annotated as 6 in the figure). Besides the input-output pairs displayed to the player, Code Hunt sometimes also displays a hint to indicate which line of code the player may want to modify in order to make progress towards solving the coding duel. Such hint can be generated in various ways, e.g., analyzing the solutions successfully completed by other players for the same coding duel. Note that it is desirable to generate hints that can guide players who struggle in solving coding duels while not compromising the learning effect (or the competition purpose when Code Hunt is used for contests). Such type of context is explicitly visible and displayed to the player.

3. DISCUSSION

Example maintenance tasks in traditional software development include feature addition or bug fixing, often accompanied with known requirements for the feature to be added or the failing behavior to be fixed. However, in Code Hunt, although the code modifications on the given player code segment can be viewed as a maintenance task, the corresponding requirements are not known. In fact, in Code Hunt, the key for successfully completing a maintenance task (i.e., solving the given coding duel) is to discover such requirements based on feedback given by Code Hunt. In Code Hunt, solving the given coding duel is not an ultimate goal; the ultimate goal is to provide learning and fun experiences for the players when they solve the given coding duel. Such main differences in Code Hunt and traditional software development can help us make an analogy between these two settings and compare their respective contexts.

Various types of context in software development have been proposed by the research community. For example, a task context [2] refers to "the information (a graph of elements and relationships of program artifacts) that a programmer needs to know to complete that task." In Code Hunt, as a slightly different interpretation, relationships of program artifacts are on functional behaviors, referring to the expected relationships of the inputs and outputs of the code segment. However, in the traditional interpretation, relationships of program artifacts are often on structural characteristics, referring to program dependencies or calling relationships, etc. More broadly, for software artifacts in traditional software development, some example contexts include their change history (corresponding to the playing history of the coding duel), requirements (corresponding to the secret code segment of the coding duel, and implicitly the input-output pairs reported to the player), dependent tasks (corresponding to earlier solved coding duels), discussions and knowledge exchanges about those tasks and artifacts (corresponding to the hint reported to the player).

Acknowledgments. Tao Xie's work is supported in part by a Microsoft Research Award, NSF grants CNS-1434582, CCF-1434590, CCF-1434596, CNS-1439481, CCF-1349666, CCF-1409423, and NSF of China No. 61228203.

4. REFERENCES

[1] J. Bishop, J. de Halleux, N. Tillmann, N. Horspool, D. Syme, and T. Xie. Browser-based software for technology transfer. In *Proc. SAICSIT, Industry Oriented Paper*, pages 338–340, 2011.

[2] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proc. FSE*, pages 1–11, 2006.

[3] N. Tillmann, J. Bishop, N. Horspool, D. Perelman, and T. Xie. Code Hunt: Searching for secret code for fun. In *Proc. SBST*, pages 23–26, 2014.

[4] N. Tillmann and J. de Halleux. Pex – white box test generation for .NET. In *Proc. TAP*, pages 134–153, 2008.

[5] N. Tillmann, J. de Halleux, T. Xie, and J. Bishop. Code Hunt: Gamifying teaching and learning of computer science at scale. In *Proc. Learning at Scale*, pages 221–222, 2014.

[6] N. Tillmann, J. de Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Proc. ICSE SEE*, pages 1117–1126, 2013.

[7] T. Xie, N. Tillmann, and J. de Halleux. Educational software engineering: Where software engineering, education, and gaming meet. In *Proc. GAS*, pages 36–39, 2013.