Creation of a Serious Game For Teaching Code Review: An Experience Report

Barış Ardıç
Bilkent University
Department of Computer Engineering
Ankara, Turkey
baris.ardic@bilkent.edu.tr

İrem Yurdakul
Bilkent University
Department of Computer Engineering
Ankara, Turkey
yurdakulirem@hotmail.com

Eray Tüzün
Bilkent University
Department of Computer Engineering
Ankara, Turkey
eraytuzun@cs.bilkent.edu.tr

***Abstract*—Code review, a manual inspection of source code by developers other than the author, is a frequently used practice for improving code quality in the software development life-cycle. Employing a tool-based review of code changes has become the norm for a wide variety of open source and industrial systems. Despite its widespread usage and importance, software development practices such as code review are often not addressed in typical Software Engineering education. To address this knowledge gap, we propose to use a serious game approach for teaching code review practices. In this study, we define our learning objectives and design a code review serious game along with its companion quizzes. Then we conduct a small preliminary experiment in order to procure feedback. Using the results of the experiment and participant interviews, we improve our game prototype for integration into a software engineering course while optimizing the initial experiment for student's benefit. We document the process, lessons learned and the future directions of the game. The results we gather indicate that the game is ready to be used in a software engineering course setting.**

***Index Terms*—code review, code inspection, software engineering education, serious games, defect classification, experience report**

I. Introduction

Code review process is established as an essential part of application life-cycle management, since it plays an important role in reducing software defects and improving quality of software projects [1]. Despite its widespread usage and importance in the industry [2], code review practice is often not explicitly addressed adequately in typical Software Engineering or Computer Science curricula [3]. To address this knowledge gap, we would like to teach the best practices, workflow and potential code quality improvements in the code review process. From our earlier experiences in capstone projects, students were applying code reviews with real tools used in practice, however since they do not have any prior code review training, the maturity of the code review processes were low.

To address this, we propose to use a serious game based approach for teaching code review process. This would be complimented by conducting code reviews in capstone projects.

We believe serious game is the right format, since learning by practicing is a significant portion of software engineering education and game formats are proven to increase user engagement, serious games are a viable format for teaching software engineering related processes [4].

The main objective of this study is to design a code review serious game that can be integrated into software engineering related course. Whereas the objective of the planned future work which this study builds towards is teaching the code review concept including its benefits, competences, characteristics and theoretical knowledge. To achieve these objectives, this study makes the following contributions:

- Developed the first serious game on code review that is based on a defect classification that uses code review related defect types.
- Created companion quizzes and tutorials in order to ease the game's integration into a university course.
- Shared our experiences of creating this game and lessons learned throughout the process.

The next section provides a background, while Section III provides a detailed insight into the game including its learning objectives, flow and content creation. After discussing the preliminary evaluation of the project's current version in Section IV, the paper concludes with the future work in Section V.

II. Background

Code Review (CR) is a process that is widely utilized in industrial and open source software community [5]. Because of code review's widespread usage across the industry, being able to review code is one of the most important skills to prepare the university students for the software industry. To teach this crucial skill, several studies use serious game format for teaching CR practices.

Serious games are often utilized for increasing the entertainment and engagement factors in the learning process beyond traditional learning while having learning objectives, interactive elements and some game elements [6]. The following studies are examples of CR related serious games where the main focus is to teach CR in a university setting.

Xie et al. [7], [8] designed Pex4Fun which provides coding duels. The aim of the player is modifying the given code's behavior until it is correct. During this process, feedback for their code is provided to the player and skills like testing, debugging and inspecting code are improved. So, the game provides a platform to teach CR indirectly.

Atal and Sureka [9] designed Anukarna which focuses on teaching the benefits and best practices of CR in a software

978-1-7281-6807-4/20/$31.00 ©2020 IEEE

engineering process based serious game. The game is decision-making based and the order of decisions simulates a project's CR workflow. The game is set on a game tree where correct decisions give access to higher scored nodes. Anukarna provides fewer game elements than its counterparts, however, it establishes a clear mapping between learning objectives and decisions made during gameplay.

Lastly, Guimaraes [10] has the most similar approach to our design. Players review the code in order to find the defects and select the reasons for the defects from a predefined list either by themselves or in as a team where their defects are grouped by voting for the final submission. Lack of emphasis on game elements and challenging content are its major shortcomings. Moreover, being a desktop application limits the games accessibility for software engineering courses.

III. Game Design

This section describes the work done regarding the application in four subsections. Section III-A defines our learning objectives, while Section III-B presents a mapping between learning objectives and features where we treat learning objectives as application requirements. Section III-C describes the overall game mechanics and how defect classification fits into the game flow. Section III-D discusses the challenges that occurred during content creation for the levels of the game.

A. Learning Objectives

To teach the code review practice to undergraduate students, we established two main categories for learning objectives from several studies [1], [5], [11], [12]. The first one is teaching the overall CR workflow and its best practices while the second one is teaching students the skills to find defects in a given code piece. CR has been utilized not just for finding defects but also for other purposes like understanding defects and anti-patterns, transferring knowledge between developers and exploring alternatives to existing solutions [5].

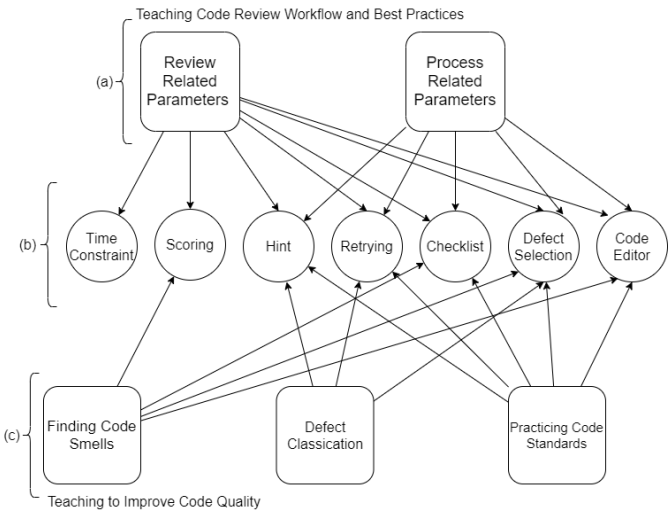

Fig. 1. Mapping learning objectives to game concepts where squares are learning objectives and circles are game elements.

B. Mapping Learning Objectives to Game Design

All of our learning objectives have at least one corresponding feature in our game design for them to be realized. We have five main learning objectives under two categories mentioned in Section III-A. Part-a in Figure 1 shows the first category consisting of review related parameters like review size and time spent and process related parameters like CR actors, their responsibilities, review checklists etc. Part-c in Figure 1 shows the second category with code quality related objectives which are defect classification, code standards and teaching the skills to find code smells. Part-b illustrates our game elements while connections imply that a game element plays a role in the realization of a given learning objective. For example, the defect classification objective is realized by hint, retrying and defect type selection game elements.

C. Game Flow

The ultimate goal regarding the implementation of the Code Review Serious Game (CRSG) is to accurately simulate a real-world CR scenario while incorporating our learning objectives. In order to achieve this, we designed an interface that is divided into three parts; the editor, help menu and the defect list. This interface can be seen in Figure 2 where the editor is on the right, defect list is on the upper left and help menu is on the lower left.

The code that is associated with a challenge is displayed on the editor in read-only form. This piece of code is manually injected with examples of defects from our defect classification. The players are expected to inspect this code and select the lines containing defects. After the corresponding line(s) is selected, players add it to their defect list and choose a reason for the defect. This process is repeated until no more defects can be detected. The reasons are presented in a nested menu in order to represent the taxonomy provided by the defect classification of Mäntylä et al. [13]. We adapted it by subtracting some subcategories like "timing" and "memory leak" for simplicity which our planned or existing game content would not cover. An example selection from the final taxonomy (that is represented as the nested menu) is illustrated in Figure 3 while all of the options can be seen in Figure 4. After the defect list is completed, the player submits it to get evaluated against the answer key. Then, we calculate player metrics like scores, time spent, number of false positives and number of hints used. Figure 2 shows a screenshot from the last level of the game, after a user has pressed "see answers".

The guide section is available at any moment during the game, presenting the players with; description of the current level, a guide explaining the items in the defect classification and a general CR checklist. There are five levels in the current version with an increasing order of difficulty. The initial levels focus on evolvability and programming style related defects. As the players proceed, new levels introduce different and more complex defects.

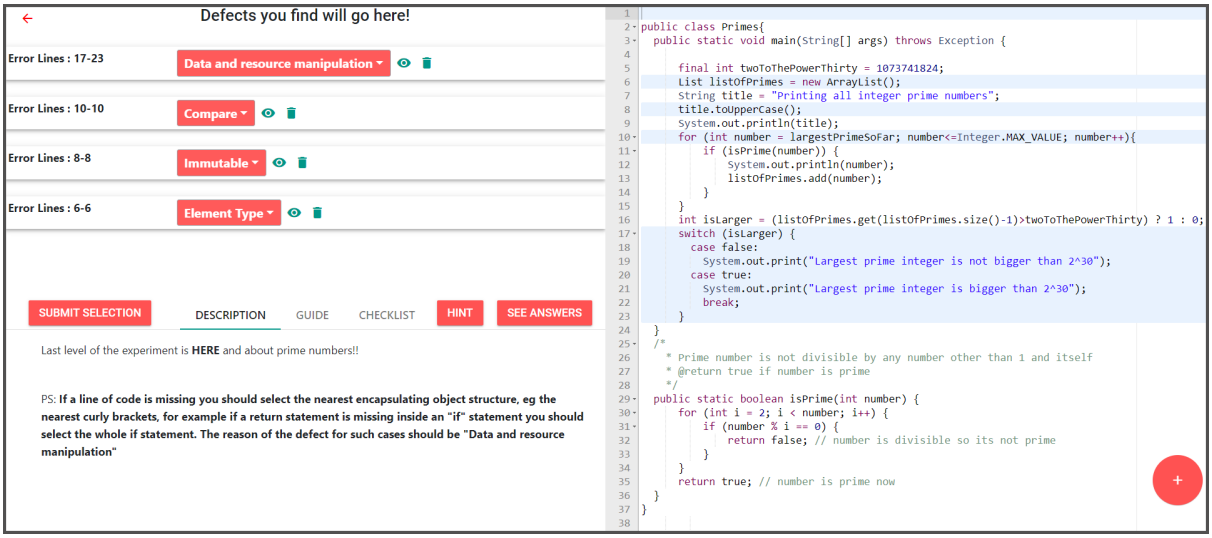

Fig. 2. Screenshot of the main game screen

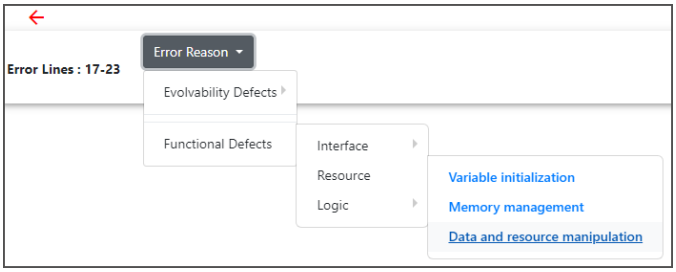

Fig. 3. Defect type selection menu.

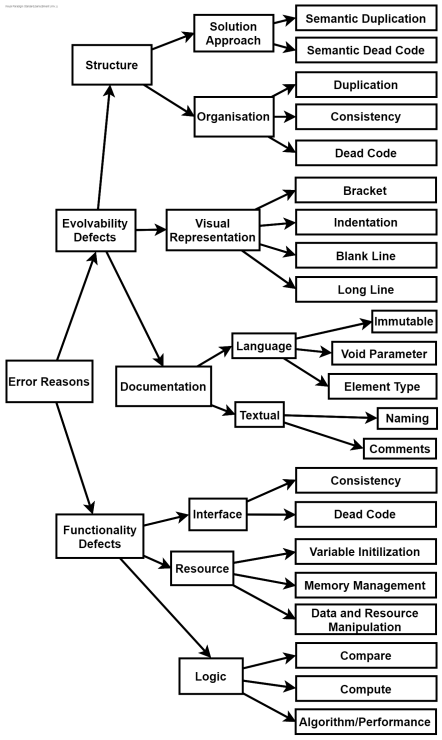

Fig. 4. Defect taxonomy adapted from [13]

D. Content Creation Challenges

Coming up with realistic CR scenarios in a limited setting, which we present a piece of code or a single file to the player, is a hard task because the players cannot see the rest of the hypothetical repository. In order to create a game with a realistic CR setting, players are not allowed to edit the code in the challenge. Moreover, because of this approach players cannot compile the code. This aspect deprives the players from "experimenting" with the given code piece. The only point players are aware of is that the code given does not produce any compile time errors. This is an important assumption to make, since the code to be reviewed in a real setting must be complete and tested. Therefore, we are not including any defects that would cause compile time errors in our challenges.

Generating appropriate errors that can occur in CR in this limited environment causes the content to move towards very simplistic errors like false comments or indentation related errors. Even though these types of errors are a legitimate part of errors detected during CR, their dominance over more "complex" defects causes the challenges to have a theme resembling "fix the style of the given code" which is not ideal for our use case.

The counterpart of this effect also occurs, making the challenges too difficult or detail oriented. Although the scenario of a harder challenge is more desirable, the level of difficulty needs careful tracking since one of the goals for this game is to be "playable" by students. Therefore, challenges that contain defects related to complex algorithms or index tracing are avoided. Otherwise, the challenges where we aim to create a realistic CR setting turns into a programming skill-oriented exercises. Some challenges can include these types

of "programming skill" oriented challenges but similar to the too simplistic defect scenario, we do not want this element to turn into the dominant theme for our game.

IV. Preliminary Evaluation

This section describes the preliminary evaluations in three subsections. Section IV-A explains our experiment setting while Section IV-B presents the feedback given by the participants. Section IV-C discusses how the game and the experiment should evolve regarding the feedback given. We shared the raw data of the experiment results online. [1]

A. Experiment Setting

Our preliminary experiment consisted of three steps. The process started with a pre-survey asking for the general educational background of the participants as well as their familiarity and confidence level with CR and its related concepts. After this initial step, the experiment continued with the pre-quiz. It consists of 27 questions which range from measuring simple CR related knowledge to interactive questions where a piece of code and some background information are given to the participants. They were then asked to conduct a review and find the defect(s) in the code piece. One of the questions from the quiz is given as an example below.

Q26) *Inspect the following code snippet, unfortunately, the author did not test it well. Now, as the reviewer, you must find the defect that the author missed using the provided test case. What is the output of the test case?*

```
public class Case{
    public static void switchCasePrimer(int
    caseIndex){
        switch(caseIndex){
            case 0:
                System.out.print("0");
            case 1:
                System.out.print("1");
                break;
            case 2:
                System.out.print("2");
                break;
            default:
                System.out.print("3");
        }
    }
    //Test Case
     public static void main(String []args){
        for(int i = 0; i < 4; i++){
            switchCasePrimer(i);
        }
     }
}
```

A) 01123[2] *B) 0123 C) 01223 D) 1123*

Before starting the game, participants were given a short tutorial on a practice level. Then, all five levels were played till completion. Completion times for the participants for each step are measured and recorded. Participants could make use of the presence of the authors or the internet for their Java related questions.

[1] https://figshare.com/s/d102309b6848a53a2d8f

[2] '1' is printed twice since case 0 is missing a break statement.

The last step of the experiment started with the post quiz consisting of the same 27 questions. The aim here was to observe game related effects on the participants' answers. With a similar approach taken with the quiz, a post survey focusing on participants' experiences was given. This post survey also included the same questions about CR related concepts so we would be able to observe any changes to the answers provided by the participants. After concluding the experiment, we also conducted follow up interviews with all participants regarding their experiences and feedback for the quiz questions.

All of the seven participants of the experiment are computer science students. Out of these seven participants, four of them were graduate students and three of the are senior year students. These participants are at a higher knowledge level than our target audience for CRSG regarding their software engineering skills therefore they are a good source for getting proper feedback.

B. Feedback From Participants

In this section, we incorporate the interview results with observations from surveys & quizzes, and then discuss everything we have gathered from the experiment under the following subheadings: mistakes, advice and clarifications. Also, Table 1 presents durations obtained from the experiment phases.

Mistakes: As with any application, CRSG had some bugs while some of the experiment questions were unclear. Fortunately, none of the complications or bugs blocked participants from answering the questions or playing the game. While conducting the interviews, the interviewees were presented with our answer key for all of the questions asked. Due to some wording problems such as asking for "side effects" of the CR process instead of "side benefits", participants were confused. With CRSG, the feedback leads us to some poor design elements. In some levels, the defects we used could be explained using different reasons from our defect classification. For example, an array indexing related defect could be explained by both "Algorithm/Performance" and "Data and Resource Manipulation".

Advice: Most of the advice of participants points out the shortcomings of our defect classification. Some participants stated that they would like code examples to be provided for error reasons. Some others stated that it is hard to navigate the user guide, thus it should be introduced in a separate page or exercise. Also, it is stated that the usage of guide inside the browser was inefficient because it required a lot of scrolling. Moreover, some participants thought that the game was too

TABLE I
EXPERIMENT SETUP DURATIONS

Phases of Experiment	Min. Duration	Max. Duration
Pre-Survey	5 min.	8 min.
Pre-Quiz	19 min.	34 min.
Play Session	35 min.	61 min.
Post Quiz	7 min.	13 min.
Post Survey	5 min.	20 min.

hard to be finished reliably. A quote from an interview is provided as an example below:

"It was an enjoyable game for learning reviewing and practicing java skills. It could have more levels with smaller increases in difficulty, current version ramps up in difficulty too quickly."

Confusions: The inconsistencies in the experiment results indicated some confusing points that originated from participants' unfamiliarity with CR concepts such as review checklists. Also, users were confused about defects covering multiple lines and whether using hints affected their scores during the experiment.

C. Discussion

Since the experiment duration presented in Table 1 is longer than we prefer for a larger scale experiment due to feasibility, we utilized the interview feedback to make some omissions. During the interviews, participants were given a 5-point Likert scale for evaluating each question in the quiz regarding their quality and ability to measure the benefits of CRSG. We assigned a score (1 to 5) to each point in the Likert scale and calculated the average point for each question. The average for all questions is $3.9/5.0$ and there are some questions significantly below the average. Additionally, we asked participants to evaluate whether playing the game did or would create a difference in their approach to what is asked for each question in a binary format. We summed their answers for each question and the questions which get 0 or 1 out of 7 are also assumed to be below average. We plan to omit below average questions in our next experiments in both cases. Our intention here is to increase the experiment quality while decreasing the time spent.

Comparing the results of pre & post quiz answers for all participants, it is possible to say that for all questions, at least 1 participant got a higher score from that question. For questions with multiple correct options, the selection of the right choices increased. For example, in the post quiz, we observed that all of the participants were aware that reviewers do not edit source code, although this was not the case before playing the game. Moreover, one of the quiz questions was asking about what to include while opening a defect during the code review. The game did not have an explicit way of conveying this information but, since line numbers carried a big role in defect marking and selection phase, participants understood that it was an essential piece of information. However, there were also a small number of negative changes in participants' answers between pre & post quizzes. During the interviews, we confirmed that these negative effects were due to the long experiment duration. We also aim to eliminate other shortcomings found during the experiment in upcoming versions of CRSG. For example, next version will allow for multiple reasons to be correct for error reason selection. Additionally, we aim to create more and higher quality content for the game to further increase its usefulness as a course assignment. Moreover, we intend to share CRSG and all its related materials with the education community as an open source project to get more feedback.

V. Future Work and Conclusions

In this study, we shared our experiences of creating a serious game which aims to teach the concept of code review. After a review of the related literature, we first determined the learning objectives of the game and then designed the game flow utilizing a defect classification [13]. The game and its companion artifacts were evaluated by seven computer science students. Their feedback will be used as a guide in the next iterations of the game. According to the survey results, majority of the participants evaluated the game as pretty enjoyable while the rest were feeling neutral about it. After we improve CRSG according to the feedback, we plan to integrate the game into the syllabus of an object-oriented software engineering course. The larger number of participants will allow us to further statistically validate our approach.

References

[1] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at google," in *ICSE (SEIP)*, F. Paulisch and J. Bosch, Eds. ACM, 2018, pp. 181–190.

[2] J. Klünder, R. Hebig, P. Tell, M. Kuhrmann, J. Nakatumba-Nabende, R. Heldal, S. Krusche, M. Fazal-Baqaie, M. Felderer, M. F. G. Bocco, S. Küpper, S. A. Licorish, G. Lopez, F. McCaffery, Top, C. R. Prause, R. Prikladnicki, E. Tüzün, D. Pfahl, K. Schneider, and S. G. MacDonell, "Catching up with method and process practice: An industry-informed baseline for researchers," in *ICSE (SEIP)*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 255–264.

[3] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, "Closing the gap between software engineering education and industrial needs," *IEEE Software*, vol. abs/1812.01954, 2019.

[4] T. M. Connolly, M. Stansfield, and T. Hainey, "An application of games-based learning within software engineering," *British Journal of Educational Technology*, vol. 38, no. 3, pp. 416–428, 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8535.2007.00706.x

[5] L. MacLeod, M. Greiler, M.-A. D. Storey, C. Bird, and J. Czerwonka, "Code reviewing in the trenches: Challenges and best practices," *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2018.

[6] M. Ulicsak and M. Wright, "Games in education: Serious games," *Futurelab*, 2010. [Online]. Available: https://www.nfer.ac.uk/games-in-education-serious-games

[7] T. Xie, N. Tillmann, and J. De Halleux, "Educational software engineering: Where software engineering, education, and gaming meet," in *2013 3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change (GAS)*. IEEE, 2013, pp. 36–39.

[8] N. Tillmann, J. De Halleux, T. Xie, S. Gulwani, and J. Bishop, "Teaching and learning programming and software engineering via interactive gaming," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1117–1126.

[9] R. Atal and A. Sureka, "Anukarna: A software engineering simulation game for teaching practical decision making in peer code review," in *QuASoQ/WAWSE/CMCE@ APSEC*, 2015, pp. 63–70.

[10] J. P. R. Guimarães, "Serious game for learning code inspection skills," Master's thesis, Universidade Do Porto, 2016.

[11] S. Sripada, Y. R. Reddy, and A. Sureka, "In support of peer code review and inspection in an undergraduate software engineering course," in *CSEET*. IEEE Computer Society, 2015, pp. 3–6.

[12] G. Rong, J. Li, M. Xie, and T. Zheng, "The effect of checklist in code review for inexperienced students: An empirical study," in *CSEET*, D. Chen, M. Barker, and L. Huang, Eds. IEEE Computer Society, 2012, pp. 120–124.

[13] M. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?" *IEEE Trans. Software Eng.*, vol. 35, no. 3, pp. 430–448, 2009.