A SERIOUS GAME APPROACH TO INTRODUCE THE CODE REVIEW PRACTICE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Barış Ardıç
September 2021

A Serious Game Approach to Introduce the Code Review Practice
By Barış Ardıç
September 2021

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Eray Tüzün(Advisor)

Fazlı Can

Burkay Genç

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

A SERIOUS GAME APPROACH TO INTRODUCE THE CODE REVIEW PRACTICE

Barış Ardıç
M.S. in Computer Engineering
Advisor: Eray Tüzün
September 2021

Code Review is an accepted and widely utilized software engineering practice that focuses on improving code via manual inspections. However, this practice is not addressed adequately in a typical software engineering curriculum. We aim to help address the code review practice knowledge gap between the software engineering curricula and the industry with a serious game approach.

We determine our learning objectives around introducing the code review process. In order to realize these objectives, we design, build and test a serious game. We then proceed with a three-step case study with 280 students.

We evaluate the results by comparing the students' knowledge and confidence regarding code review before and after the case study, as well as by statistically evaluating how well they did both in the code review quizzes and the game levels themselves. Our analysis indicates that, students have a positive approach regarding playing the serious game while the statistical results show that students improve their knowledge by playing the game.

We conclude that our code review serious game had a positive impact on the students and is helpful for introducing the code review process. The game and materials for the case studies are made available online for educators.

Keywords: code review, code inspection, software engineering education, serious games.

ÖZET

KOD GÖZDEN GEÇİRME EGİTİMİ İÇİN CİDDİ OYUN BAZLI BİR YAKLAŞIM

Barış Ardıç
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Eray Tüzün
Eylül 2021

Kod gözden geçirme kaynak kodun başka bir geliştirici tarafından incelenmesini kapsayan kabul görmüş ve yaygın bir yazılım mühendisliği sürecidir. Yazılım endüstrisindeki yaygınlığına rağmen tipik yazılım mühendisliği programlarında kendine gerekli yeri bulamamaktadır. Bu tez yazılım mühendisliği programları ile endüstri arasındaki bu farkın giderilebilmesi için ciddi oyun tabanlı bir yaklaşım sunmaktadır.

Ciddi oyun yaklaşımının öğretim hedefleri kod gözden geçirme prensiplerine giriş yapmayı kapsar. Bu tez bahsedilen hedeflerin gerçekleştirilebilmesi için, ciddi oyun platformunun dizaynını, realize edilmesini ve işlevsellik testlerinden geçirilmesini kapsar. Oyunun test süreci toplam 280 öğrenci içeren üç vaka analizinden oluşur. Sonuçların değerlendirilmesi katılımcıların vaka analizinden önceki ve sonraki kod gözden geçirmeye dair bilgi ve özgüvenlerinin analiziyle yapılmıştır.

Bu analize olanak veren ciddi oyunun kullanımdan önce ve sonra uygulanan sınavlar ve anketlerdir. Analizler ögrencilerin ciddi oyuna karşı olumlu bir tutum sergilediğini göstermektedir. İstatistiki değerler de ögrenci bilgisinin ciddi oyunla etkileşimden sonra arttığına işaret etmektedir. Sonuç olarak ortaya konan kod gözden geçirme ciddi oyun platformu, kod gözden geçirme eğitiminde kullanılabilirliğini ispatlamıştır. Platform ve aidiyetindeki ölçüm materyalleri eğitimcilerin erişimi için internet ortamında mevcuttur.

Anahtar sözcükler: kod gözden geçirme, kod denetleme, yazılım mühendisliği eğitimi, ciddi oyunlar.

Acknowledgement

First and foremost, I would like to thank my advisor Eray Tüzün for his constants contributions and support regarding during this thesis and my education. I also would like to thank my colleagues, family members and friends for their insights and inspirations.

September 2021, Ankara

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Code Review is a manual inspection of source code by developers other than the author. This process is established as an essential part of application life-cycle management and applied frequently in modern software development. Conducting code reviews properly has been shown to have an important role in reducing software defects and improving the quality of software projects [1]. Despite the widespread usage and the emphasis given in the industry [2], code review practice is often not adequately addressed in typical Software Engineering or Computer Science curricula [3]. To address this knowledge gap between the software industry and higher education, we would like to introduce a serious game approach to teach the best practices, workflow, and potential code quality improvements in the code review process.

1.1 Research Problem

In our earlier experiences with capstone projects of software engineering courses where students were asked to apply tool-supported code reviews, we observed that the maturity of the code review process was low because of their lack of code review related training.

To address these poorly conducted code reviews, caused by the knowledge gap, educators would require a platform that conveys the fundamentals of the code review process while allowing the participants to practice their defect detection skills. Moreover, any additional exercises focusing on demonstrating the author's or the reviewer's role in the code review process will allow for a more realistic representation of the process. A combination of these aspects will allow students to start participating in a more mature code review process.

We believe serious games are a viable way to address this knowledge gap because learning by practicing is a significant part of software engineering education and game based formats are proven to increase user engagement [4] [5]. Another strength of the serious game approach is the added usability compared to more industry oriented options. For students with varying levels of experience and enthusiasm, introducing concepts with game based approaches tend to be easier compared to the more complicated professional tools because of differences in their learning curves.

We, therefore, propose a serious game based approach as the platform for introducing the code review process. The game based learning approach provides additional beneficial aspects to the aforementioned learning platform because game based learning increases student engagement. In our case having a gamified platform also enables additional use cases for application such as online homework or lab assignments.

To validate our approach, we aim to determine how successfully can the code review serious game (CRSG) introduce the code review concepts to students. To measure this, we present the following research questions and disclose the ways which this thesis will use to address them:

- **RQ1: How effective is the code review serious game for introducing students to code review and its related concepts?** The effectiveness of the serious game will be measured on two mediums. The first medium consists of the pre and post case study surveys where participants will provide information on their baseline an post gameplay knowledge,

confidence, and familiarity with the details of the code review process. The second medium is the code review knowledge quiz which the participants will take before and after the case studies. This will allow us to collect data regarding participants' knowledge of specific aspects of the code review process.

- **RQ2: How feasible is to use the code review serious game to introduce code review concepts within a course curriculum?**

  The medium that is used for measuring the feasibility of the code review serious game in the thesis will mainly include the survey mentioned above. This will be supported by the player related data that we collect from our participants. By timing and recording their actions on our platform, we can deduce their overall engagement with the platform or give feedback to the participants while they are engaged with the serious game to enable autonomous player progression which then increases the feasibility of the platform.

1.2 Contributions of the Thesis

The main objective of this thesis is to design, implement and evaluate a code review serious game that can be integrated into software engineering related courses without occupying much lecture time. To achieve these objectives, this study makes the following contributions:

- Developed the first publicly available code review serious game [1]
- Created companion guides and tutorials in order to allow players to progress in the game on their own which would then ease the game's integration into a university course.

[1] https://codereviewseriousgame.web.app/

- Designed quizzes and surveys in order to be able to evaluate the effectiveness of the game[2].
- Conducted multiple case studies with a total of 280 students for evaluating the game on scale, then we shared our results and findings. We also provided improvements to the platform between the individual case studies.

The rest of the thesis is organized as follows. The next chapter provides a background, while Chapter 3 provides a detailed insight into the game including its learning objectives, components, and flow. Chapter 4 describes the case studies in detail, while Chapter 5 presents the results of the case studies alongside their analysis. Chapter 6 provides our discussion regarding the study, then Chapter 7 provides the threats to validity. Chapter 8 concludes the thesis.

[2]https://bit.ly/3kvXiAb

Chapter 2

Background

Code review is a manual inspection of source code by developers other than the author of the source code [1]. A simplified overview of the code review process is provided in Figure 2.1 which is adapted from [6]. The process starts with the initial code segment which is altered until all concerns of the reviewer(s) are eliminated. The final version of the segment is accepted into the codebase.

The code review process is an established part of modern software development and is considered to be vital by leading software companies in the industry [2, 7]. The research regarding this process is often done with collaborations between researchers and companies. A case study done in Google [1] shows that a mature and mandatory code review process was accepted and its benefits were acknowledged by the engineers. Another study, from Microsoft, [8] congregates the challenges to expect and best practices to apply while establishing a modern code review process. Besides such collaborative studies, the benefits of code review have been a popular topic for empirical software engineering research. McIntosh et al. [9] have provided empirical evidence regarding the relationship between code review coverage and post-release defects. Using the code review data from popular and well documented open source projects like Qt[1], they show that both low code review coverage and participation, produce artifacts with up

[1]https://www.qt.io/

to five additional post release defects.

The rest of this chapter focuses on two main points. The first is to document the benefits of code review with empirical evidence in order to support the motivation of the study. The second point of focus is to demonstrate the viability of our approach by presenting the usage of serious games regarding teaching software engineering practices. We consider studies that aim to interactively teach code review or other software engineering processes as the studies that relate to this one.

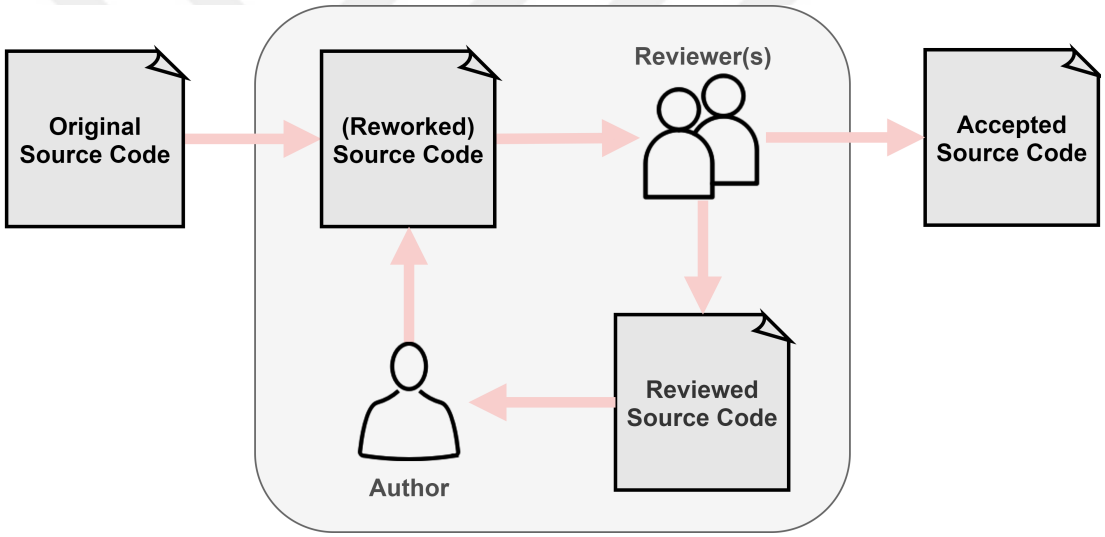

Figure 2.1: An overview of the code review process. [6]

2.1 Interactive Platforms for SE Concepts

Game based platforms are often utilized for teaching software engineering related concepts. A systematic mapping study by Souza et al. [10] investigates 156 studies between 1974 and 2016 about utilizing game concepts to teach software engineering practices. A common theme of motivation for these studies are the shortcomings of university lectures related to software development specific concepts. There are two main approaches regarding the scope of these studies.

First are the studies which combine multiple software engineering concepts

or oversee software development as a whole in their application flow; therefore focus on multiple aspects of software engineering at once. A leading study in this regard is SimSE by Navarro and Hoek [11] where the player controls a software engineering project by making decisions from a managerial perspective. Another example is the gamified course platform by Berkling and Thomas, where they develop a course resource navigation platform and improve upon it by utilizing the feedback from their students. This study has larger scope than ours since it aims to completely transform the way a course is taken. However, the methods which they advance their study is similar to ours as they are utilizing student feedback both as an evaluation measure and as a guide for improvement [12] [13].

Second, are the studies that focus on a specific software engineering process or skill. For example, Sonchan and Ramingwong [14] tackle software engineering risk management skills with the card based game ARMI where players identify and analyze the risks and try to strategize correctly for risk management during software development. Another example is the "Code Defenders" game by Rojas and Fraser where they present an analogy to the mutation testing technique by providing two roles to player. The player with attacker role mutates a code piece by making a change while the player with the defender role creates tests to catch the mutants. The back and forth moves between the players allows them to practice the basics of mutation testing [15].

Despite the examples so far, an interactive platform is not required to be delivered via software. More traditional mediums like board-game or pen&paper based formats are also utilized in the literature. One example is SCRUMIA by Wangenheim et al. [16] where students are divided into teams in the classroom environment. Each student is assigned a role from the Scrum framework. Students then advance in the provided scenario by following Scrum activities, achieving goals and in turn, gathering points. Another example with a larger scope is "Problems and Programmers" where the authors emulate the software development process with a card game which demonstrates the decision making process, trade-offs and best practices of development [17].

2.2 Serious Games for Code Review

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 [18]. The following studies are the closest examples in the literature for code review (CR) related serious games where the main focus is to teach CR in a university setting.

Xie et al. [19, 20] designed Pex4Fun, which provides coding duels. The aim of the player is to modify the given code's behavior until it is correct. During this process, feedback for their code is provided to the player, and skills such as testing, debugging, inspecting code are improved. So, the game provides a platform to teach CR indirectly.

Atal and Sureka [21] designed Anukarna which focuses on teaching the benefits and best practices of CR in a software engineering process based serious game. The game is decision-making based and the order of decisions simulates a project's CR workflow. It is set on a game tree where correct decisions give access to higher scored nodes. Anukarna provides fewer game elements than its counterparts like Pex4Fun; however, it establishes a clear mapping between learning objectives and decisions that are made during gameplay.

Lastly, Guimaraes [22] has the most similar approach to ours. 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 a team . The defects in their code snippets are refined by voting for the final set of defects before submission. Lack of emphasis on game elements and challenging content are its major shortcomings.

Our approach provides a more comprehensive breakdown of code review related concepts while also creating multiple mediums for data collection which are being used for a detailed evaluation of the platform. The previous studies on the topic lack detailed evaluations of their methods. Furthermore, the process of applying review comments is not addressed. We believe this is an important element since the actual benefits of code review are obtained from successfully applied reviewer

comments. In order to reflect this key aspect, we implemented the author mode to CRSG on later stages of the thesis.

Chapter 3

Game Design

This chapter consists of 3 sections. Section 3.1 describes our learning objectives while Section 3.2 reports on the preliminary experiment and the feedback we gathered from it. Section 3.3 demonstrates the flow of the game modes while providing a detailed breakdown of its components.

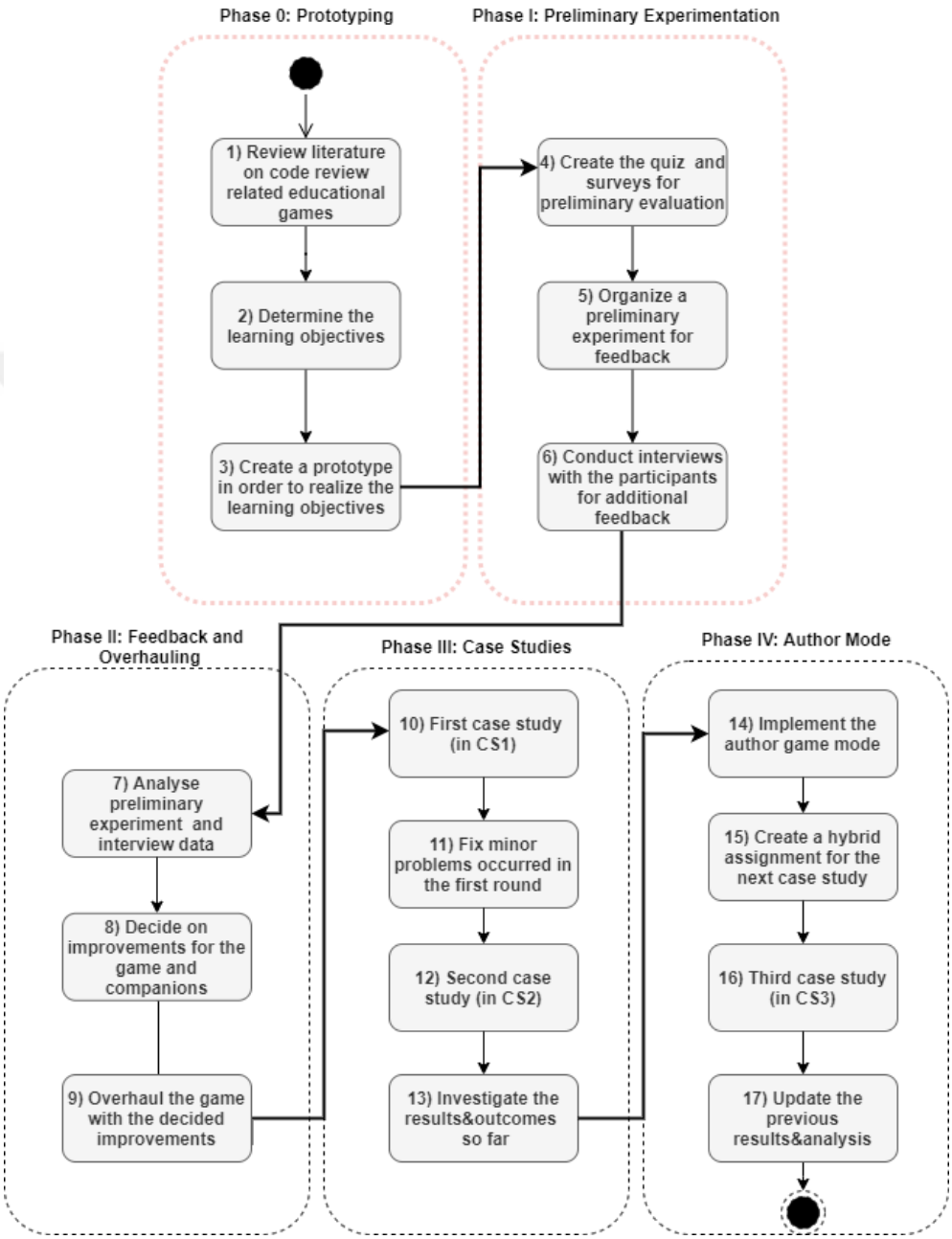

Figure 3.1: Overall workflow of the thesis

3.1 Learning Objectives

The code review serious game is designed around realizing two main objectives. The first is to convey general information regarding the code review concepts whereas the second is to allow players to train the skills which allow for effective defect detection during code review. While coming up with the general structure of the game, we further broke down these objectives and created game elements & components around these smaller objectives.

These main learning objectives are:

- Understanding code review related concepts
- Identifying code errors and smells

The first objective consists of code review roles, duties, and workflow. The second objective consists of finding code errors, learning to classify these defects to better communicate them, and to practice reviewing code. After this initial design, we tested the gameplay and made additions to the base game structure until we were satisfied with the emphasis that was given to each miniature objective. This process is represented by steps 2 and 3 in the overall workflow diagram in Figure 3.1. The mapping between the base objectives and game components can be seen in Figure 3.2, where rows *a* and *c* refer to the first and second objectives respectively, while row *b* refers to the game components and arrows indicate which objective is realized by which components. Author mode feature was added after the initial design was completed in order to simulate both roles that are involved in code review. Since it is a standalone game mode, it also inherits most of the features of the original reviewer mode.

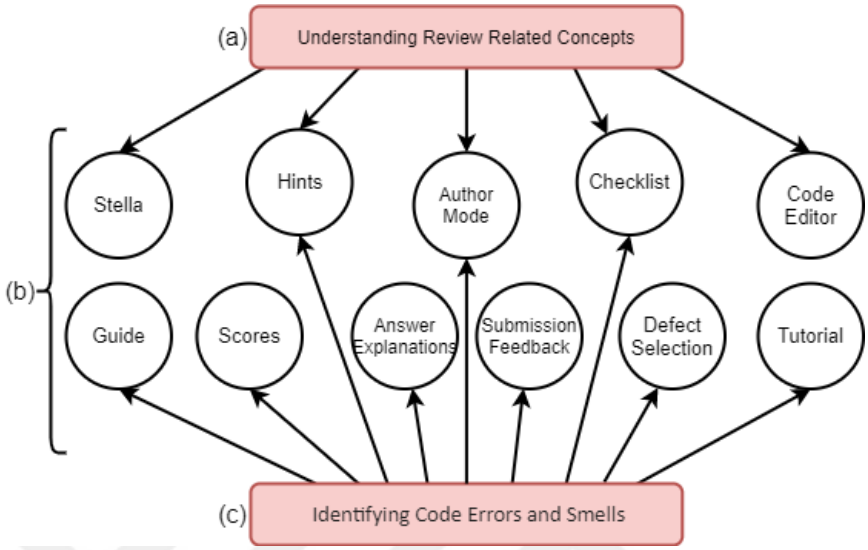

Figure 3.2: The mapping between game features and learning objectives

3.2 Prototype, Preliminary Experiment and Feedback

After establishing our learning objectives, we built a prototype, prepared quizzes, and surveys for evaluation. Moreover, we conducted a small preliminary experiment with 7 students in order to gather their feedback and eliminate the shortcomings of the prototype and the initial evaluation experiment. The details of this process are shared in [23]. We initiated this small experiment with a code review survey and a quiz, and then introduced the participants to the game by playing a tutorial level followed by a gameplay session where all game levels were played to completion by all participants. We concluded this experiment with a post-quiz and a post-survey.

The main goal of this preliminary experiment was to gather feedback. We also conducted follow-up interviews where we asked the participants to evaluate the quiz on a per-question basis. The experiment took about 3 hours from start to finish since we wanted participants to raise their opinions as they arose while we took notes. The durations for the components of the preliminary experiment

are given in Table 3.1. The follow-up interviews took about 30 minutes per participant. We evaluated the feedback from the participants throughout the experiment and the interviews, then used our findings to improve the evaluation materials. This was done by adding new features and reducing the size of the quiz and the survey by dropping the questions that were deemed to be the least clear or relevant to our learning objectives.

The main motivation for this process was the experiment duration presented in Table 3.1 being longer than we prefer for a larger scale experiment due to feasibility. We, therefore, 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. We assigned a score to each point in the Likert scale and calculated the average point for each question. The average of all is 3.9/5.0 and there are some questions that are significantly below the average. We omitted these questions in our case studies. 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 ones which get 0 or 1 out of 7 are also omitted from the case studies. Our intention here was to increase the material quality while decreasing the time spent on evaluation. By doing so, we improved the quality of the evaluation materials and reduced the experiment duration to a more feasible interval which was necessary for larger-scale case studies. These steps are represented with steps 3 to 8 in Figure 3.1. Before starting the large scale case studies we also overhauled the game by improving its theme and added

Table 3.1: Preliminary Experiment Durations [23]

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.

several features that were desired by the participants of the preliminary experiment. These additions are denoted by step 9 in the overall workflow figure 3.1.

As with any application in the early stages of their life-cycle, CRSG had some bugs. 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 lead us to some poor design elements. In some levels, the defects we used could be explained by 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".

The CRSG related advice from the participants was straightforward. Participants told that using the guide inside the browser was inefficient because it required a lot of scrolling. Some participants also stated that they would like code examples to be provided for error reasons. Most of their advice points out the shortcomings of our defect classification.

We eliminated the shortcomings described above in the more recent versions of CRSG. For example, the current version now allows for multiple reasons to be correct for error reason selection.

3.3 Game Flow

The final configuration of the game after the third case study consists of a tutorial, a practice level, two reviewer mode levels, and two author mode levels. The configuration before Phase IV in Figure 3.1 had four reviewer mode levels. The tutorial has several code segment examples where each has a single defect in it together with its classification, description, reason, and how the author could fix it. The practice level is an interactive manual that teaches the mechanics of

the game to the player. The game levels themselves are larger code segments with numerous code defects planted in them. For the reviewer mode levels, the player has to find all of the defects, along with their classifications, to be able to complete the levels fully. For the author mode levels, the players need to successfully address the comments that were already provided by the reviewer. Players are also able to see the answers to a level but once they do, they can not submit that level again. If the player is stuck at any point in a level, they can utilize hints or other helpful materials that we have prepared. The typical flow of a level for each game mode has been demonstrated in Figure 3.3. A more in depth analysis of the design choices and components of the game has been previously shown in [24].

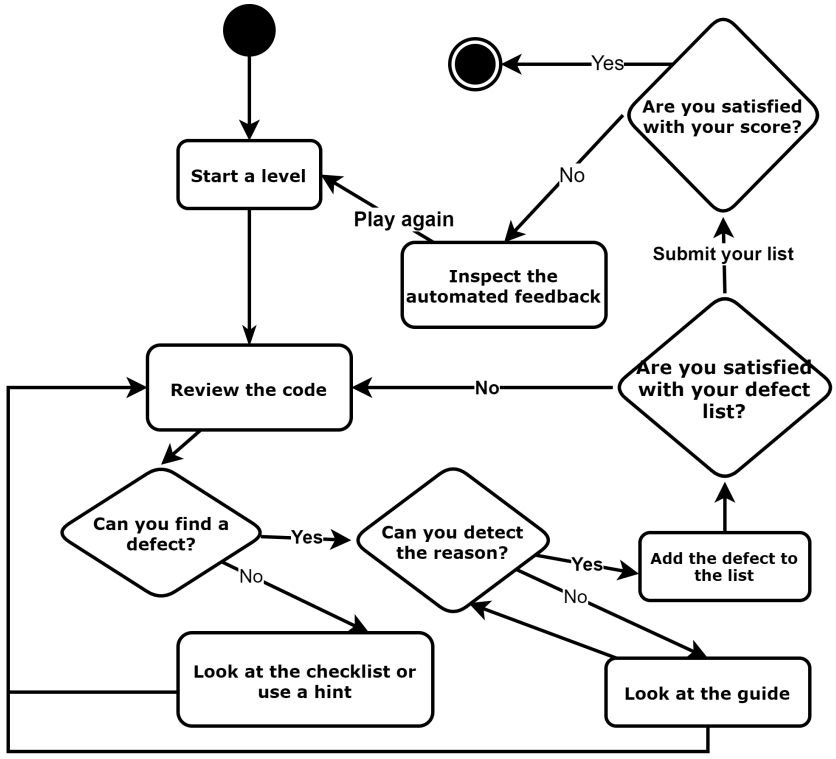

Figure 3.3: Typical level flow for both game modes

3.3.1 Reviewer Mode

A reviewer mode level starts with players trying to identify defects in the code segment. Upon finding a defect, they select the related lines and determine the reason for the defect by utilizing our preexisting taxonomy adapted from [25]. Repeating this process, the players create a list of defects that they found during the code review. Once they are satisfied with their defect list, they can submit the list for evaluation. Our scoring algorithm checks each defect in submitted by the player against our answer key. A defect consists of its starting line, reason, and ending line. As one can see in Figure 3.4, our scoring method awards the players the first point when they successfully detect the starting and ending lines of a defect. Only then, the player can get an additional two points if they also assign the correct reason for the defect. In this structure, each defect ends up being worth 3 points. In order to guide the player to the right direction regarding finding the defects, we provide feedback to each submission sent by the player. This feedback is tied in with our scoring. Each defect is matched with a color and the border of the defective snippet is changed to that color. Red is used for a defect that got 0 points from the scoring algorithm, meaning that the player was wrong about the location of the defect. In a similar fashion, yellow is used for defects submitted on the right lines but with an incorrect reason. Green refers to a correctly submitted defect.

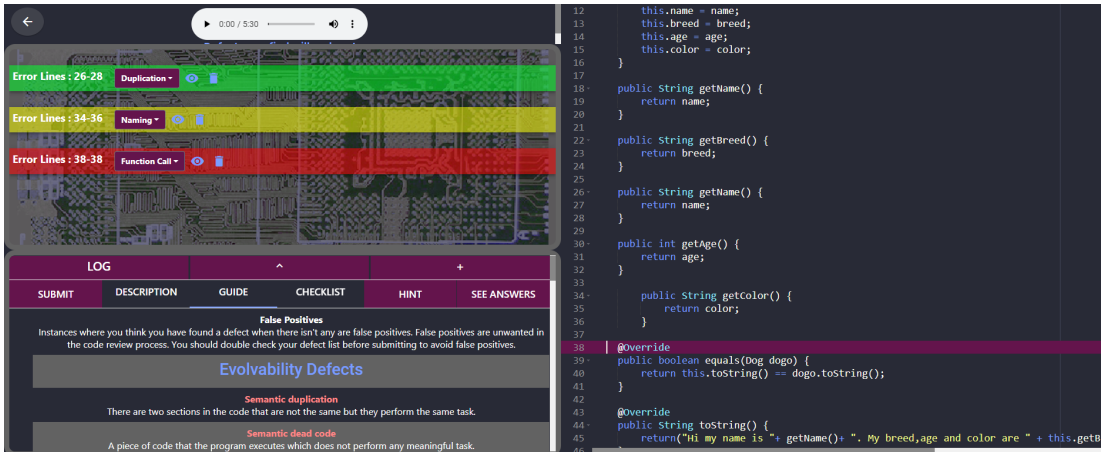

Figure 3.4: A reviewer mode level

Algorithm 1 Reviewer Mode Scoring

```
1: score ⇐ 0
2: for each sub in submittedList do
3:     for each defect in groundTruth do
4:         if sub.start == defect.start AND sub.end == defect.end then
5:             score ⇐ score + 1
6:             if sub.reason == defect.reason then
7:                 score ⇐ score + 2
8:             end if
9:         end if
10:    end for
11: end for
```

The state of the reviewer mode level with the progress feedback after a defect list submission can be seen in Figure 3.4 where the right-hand side shows the defective code, and the left side is divided between the defect list and the support menu. Left hand side of Figure 3.3 demonstrates how a reviewer level is typically played.

3.3.2 Defect Taxonomy

We wanted to use a realistic list of defect reasons for players to select from while playing, therefore we started with the defect taxonomy mined from review data by Mäntylä and Lassenius [25]. We further trim and adapt it to be used in the code review serious game. Most of the changes made to the taxonomy are made to make it more compliant with the Java language and to simplify it for players' convenience. For example, we subtracted some subcategories like "timing" and "memory leak". We present the taxonomy to the player as a UI element that they can hover, inspect, and select a relevant reason for their defects. Figure 3.5 represents the overall taxonomy that we use in the current version of CRSG.

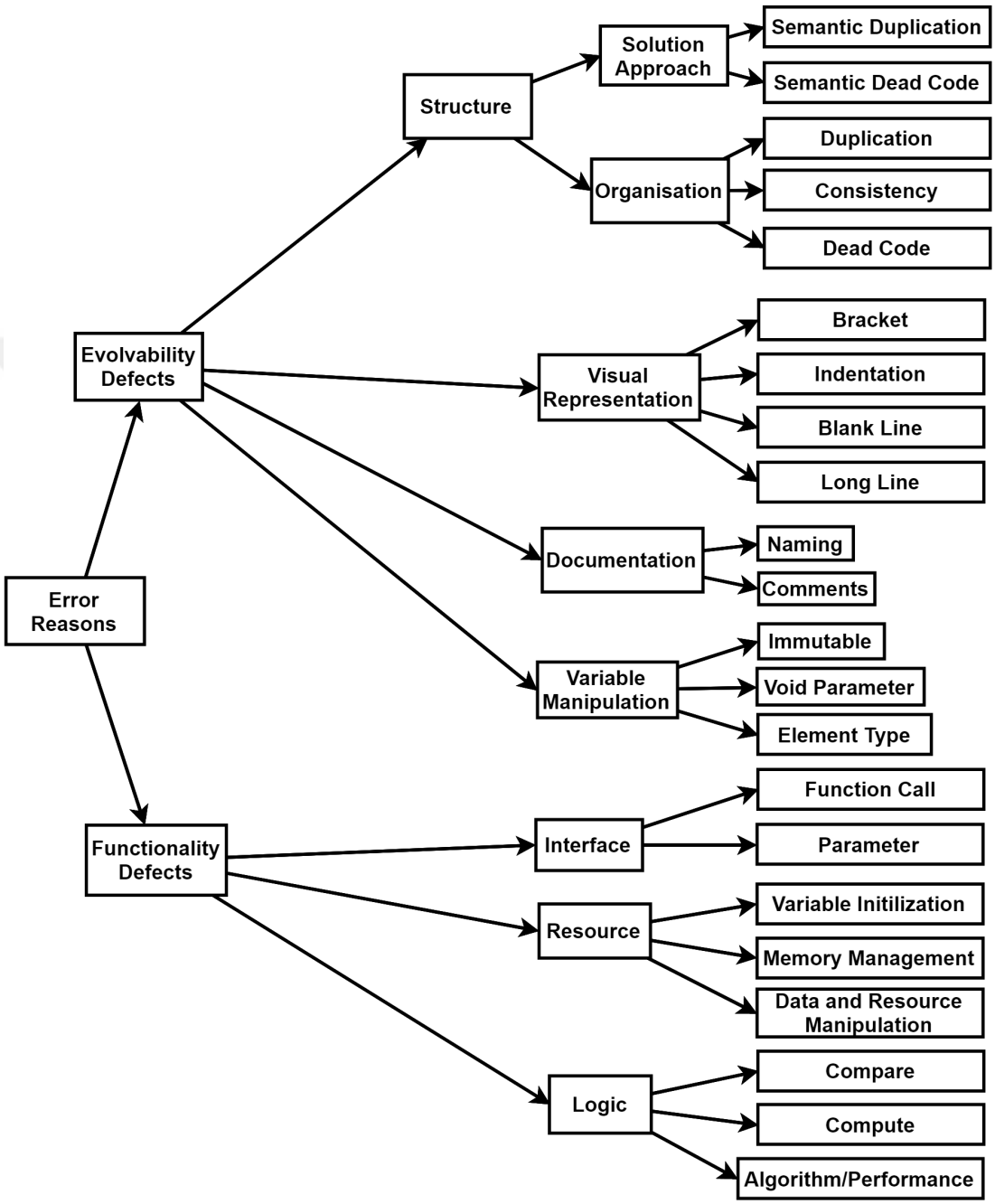

Figure 3.5: The defect taxonomy used in the code review serious game

3.3.3 In-Game Helpers

There are three in-game helpers. These are game components aiming to help smooth out the learning experience of the player like the tutorial or the practice level, but without having them leave the level they are playing, unlike the tutorial. While they are not as descriptive as the tutorial by themselves, they work without needing to disrupt the gameplay and help explain some aspects of CR that players might not have understood. These three helpers are available at the tabs on the bottom left of the editor screen and are explained below. The interface for the helpers can be seen in Figure 3.4 where the selected tab is "guide".

Guide houses the taxonomy tree of defect types, followed by concise definitions for all defect types listed below for reference. These defect types are colored in the same way as they were in the tutorial to help those that learn with color better. Also, to increase readability, each defect type is separated from one another by cells of alternating shades.

Checklist has a numbered list of items to check while reviewing code. These types of checklists have already been studied and they were shown to be effective for teaching code review [26]. Ours is structured in a way that the reviewer is encouraged to review the code starting from easier to see defects and move on to more contextual, harder to see ones (sometimes more specific to the programming language in use). It is also structured in alternating cells similar to the guide to make it more readable.

Show Answers, when clicked, shows the correct defects on the top left part of the editor screen and switches the bottom left tab to the description tab, changing its contents to the explanations of the defects listed above. This way, the player not only sees the correct defects but also has a chance to read the reason behind them. The game also automatically highlights all of the defective lines of code for player convenience.

3.3.4 Author Mode

The idea behind CRSG aimed for an accurate representation of the code review process. The initial version of the serious game only provided gameplay for the reviewer role. With the current version, we intend to cover both sides of the code review process, since we introduced a new game mode that approaches the process from the code change author's perspective. A typical level of this new game mode starts with an already reviewed piece of code where the comments and additional context regarding the initial iteration of the review are presented to the player. Using this information provided to them, players read through the code and the reviewer comments, deciding on whether the comments are valid. We have also implemented a direct way to reject false positive comments. All reviewer comments also have source code lines attached to them for the player to be able to go through them with ease. These components are demonstrated in Figure 3.6, where reviewer comments reside on the lower left-hand side while the editable code segment resides on the right-hand side. The top left side keeps a reference for the starting point of a level in case players need to revert to the original.

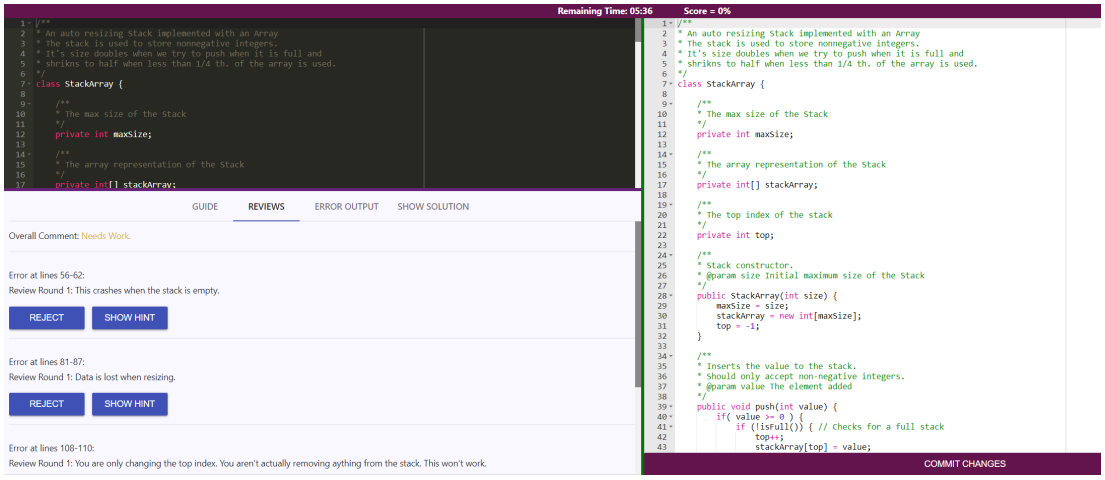

Figure 3.6: An author mode level

Using the reviewer comments, players try to apply their fixes to the source code for the reviewer comments which they intend to fix. After finishing their first

iteration of fixes (the player's first iteration corresponds to the second iteration of the code review process that is being simulated) the updated version of the code can be submitted to the code review serious game. We evaluate these submissions in the back-end against our test cases. The player is awarded points for the reviewer comments that they were able to apply correctly. For the comments that were missed, the corresponding reviewer comments are updated to show the problem persists. At any point during the level, players can also utilize hints which are different for each reviewer comment. This is done for players to not get stuck easily while applying the review comments. The differences regarding the flow of the two game modes can be observed by comparing the respective parts of Figure 3.3.

3.3.5 UI Components

The UI of the game is made to look like the player is controlling Stella's journey through space. Each game level is represented by a space station. Level numbers and descriptions are given under or over space stations in a white font without any background. This way, individual levels and what they mean are instantly revealed, but the text does not detract from the appearance of the map, nor does the UI feel too cluttered.

Each space station has a gap in its middle where the button for the corresponding level resides. This area is made as a part of the theme by being travel relays: Stella must help repair the relays by reviewing in order to be able to travel to the next station/level. To indicate the difficulty of the level the player wishes to go into, each successive relay is shaped like a polygon of an increasing number of vertices. The tutorial has 3 vertices, the practice level has 4, 1st level has 5 etc. Also, the mouse cursor is changed into Stella's spaceship, the Crane, to add further to the theme. The visual assets that were necessary for the game UI were drawn using Adobe Photoshop software.

There is also a music controller on the bottom of the screen that auto-plays music to increase immersion but it can be stopped, or its sound level could be

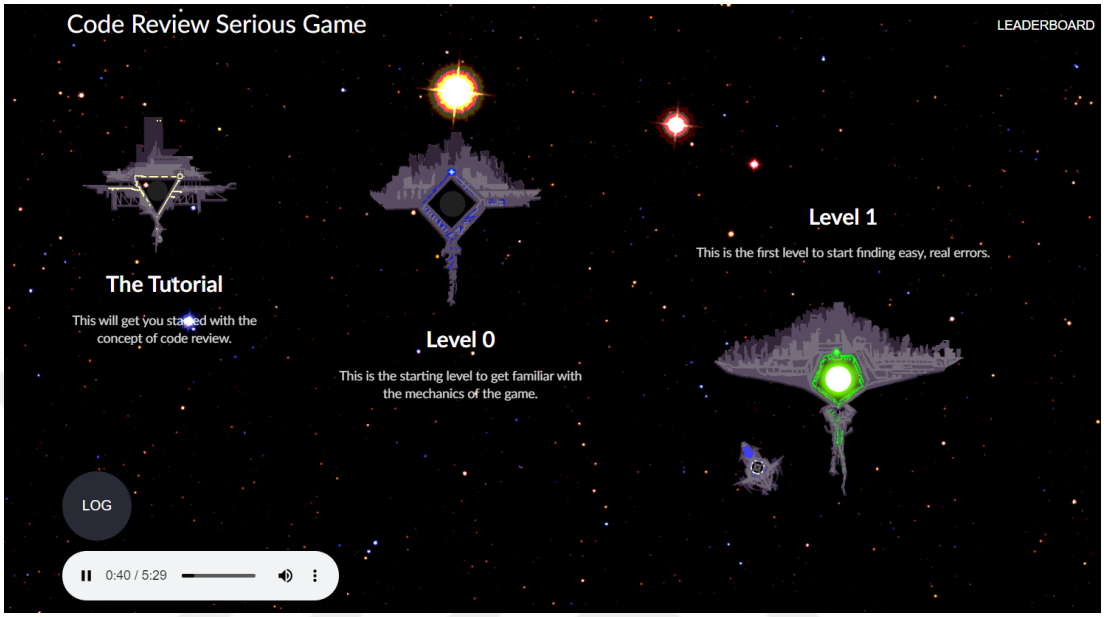

Figure 3.7: Main menu screen

adjusted.

Stella, the main character of the game, is a character that we introduce at the start of the game who interacts with the player at the start of and in between each level as their mentor. We utilized this character to convey code review related advice. The advice is a compilation of our experiences and code review guidelines of Thoughtbot [27] which is a web design on demand service. An interaction with Stella from the game is shown in Figure 3.8.

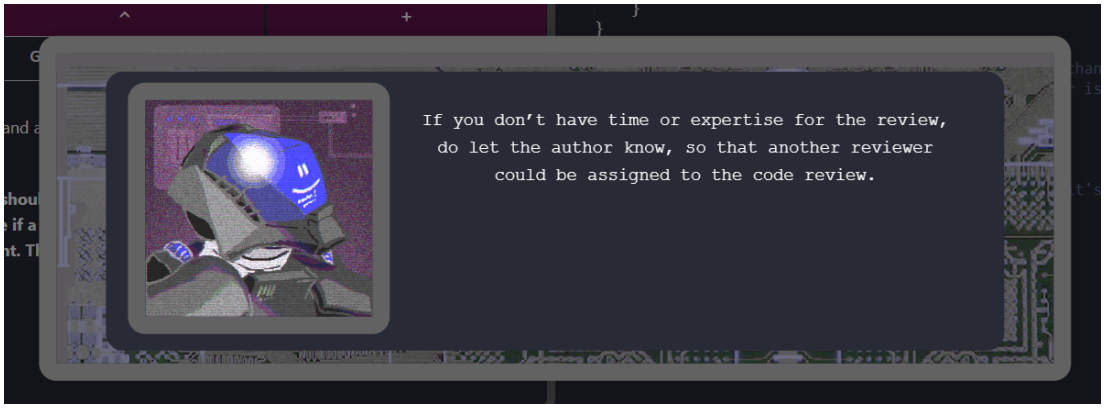

Figure 3.8: An interaction with Stella

3.3.6 Tutorial

The tutorial is made for familiarizing the players with defect definitions using examples with the defective and non-defective versions of small code segments. In the current version, it houses ten different code segments, each with a different type of code defect in it. The player can read the definition of an example to see what the type of the defect is, alongside an explanation of that defect type. To make the learning experience smoother for those who learn better with visual support, the two main defect classes, evolvability, and functional defects are denoted on the tutorial screen with the shape of a brain and a wrench respectively in addition to the text while different defect sets (sub-trees) in the taxonomy have unique colors assigned to them.

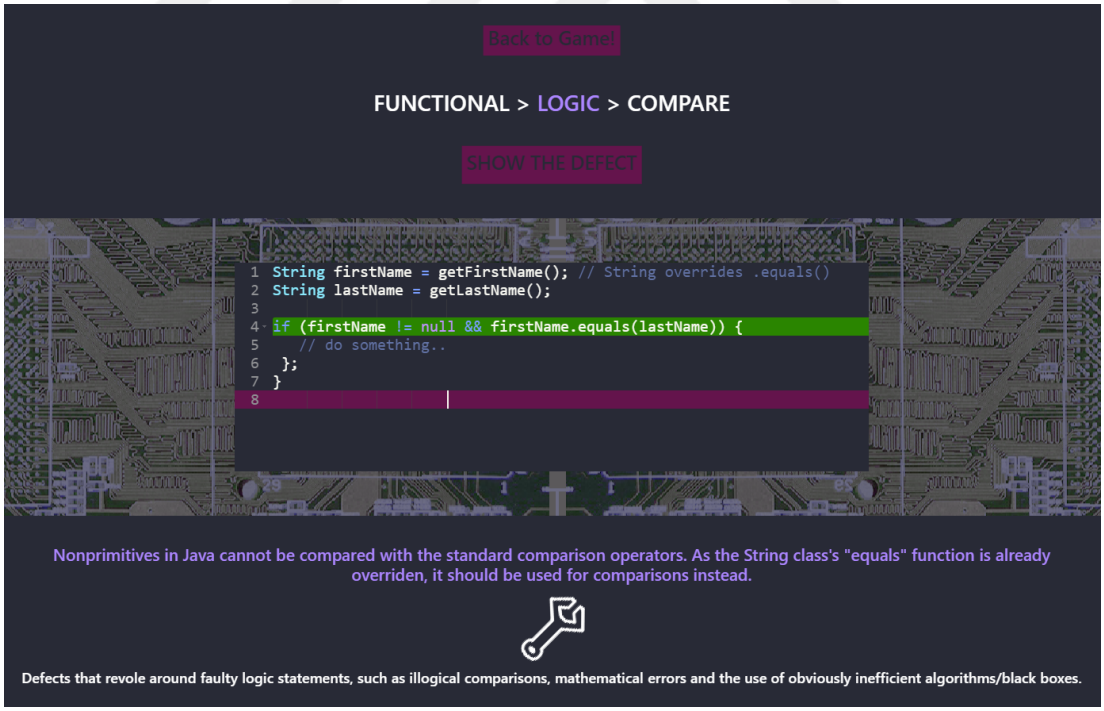

Figure 3.9: Tutorial

The player can click the “See Answers” button to see the solution of the shown example defect alongside how it can be avoided/fixed is explained. The definition of the defect type is given before seeing the solution. Pressing the button again returns the example to its original form so that the player can compare the defect

and the solution at their own pace without having to reload the page and scroll to the defect they were looking at each time.

Chapter 4

Case Study Design

In this chapter, we describe our case study format and then introduce the materials that were used for data collection.

4.1 Case Study Setting

For evaluating the contributions of this thesis, we adopt an explanatory case study method where we utilize multiple ways of data collection while repeating the case study in different semesters to improve the soundness of our findings. We utilize the checklist from Kitchenham et al. [28] to initiate our case study template. The case study is designed around being able to observe the changes in the knowledge of players before and after playing the game. In order to achieve such a format, we have compiled various auxiliary material aiming for both numerical evaluations and players' personal evaluation regarding their experience with the CRSG. This material, like all parts of this thesis, evolved over time with feedback from the preliminary experiment as described in Section 3.2 as well as with constant comments from play-testers throughout development.

The original intention of the case study was to hold in-person lab sessions. Due to the university's transition to online education because of Covid-19 pandemic

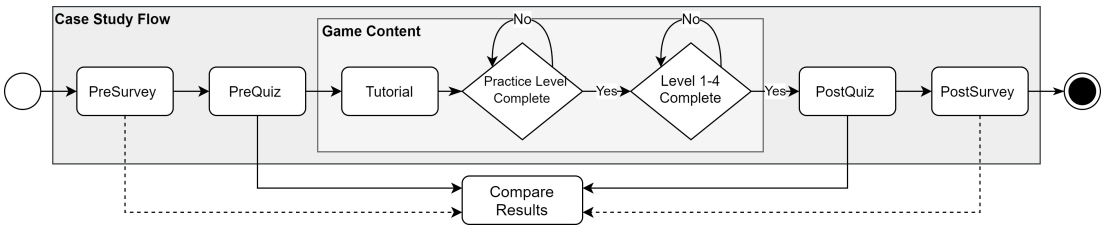

Figure 4.1: The flow of the case studies

Case Study No	Class	# of participants	Reviewer Mode	Author Mode
CS0 (Preliminary)	-	7	✓	-
CS1	CS319	52	✓	-
CS2	CS453	80	✓	-
CS3	CS319	144	✓	✓

Table 4.1: Case Study Details

regulations, we performed the case studies in a video conferencing format. The studies were done in three rounds, which are represented by steps 10, 12, and 16 in Figure 3.1, where each corresponds to a software engineering course in the computer science department. From here on, we mention each case study by numbering them chronologically as CS1, CS2, and CS3. CS1 was done in CS319 [29] which is a must-course taken by 3rd year students. CS2 was done in CS453 [30] which is an elective course generally taken by 4th year students. Similarly, CS3 was done in CS319 of the next year. The total number of participants was 280; however, some participants did not complete all steps of the case study. Among the complete 276 submissions, CS1 had 52 participants while CS2 had 80 and CS3 had 144.

The flow of a case study can be seen in Figure 4.1, which starts with a pre-survey that was sent out a day before. In all case studies, the scheduled meeting started with a short verbal tutorial and was followed by the pre-quiz component. After that, players proceeded with playing the game tutorial and the practice level. At any point, they could ask questions to us using the private chat function of the video conferencing application Zoom[1]. The study then continued with the

[1]www.zoom.us

actual gameplay of levels 1 to 4. The players who have completed the main gameplay levels filled out the post-quiz before signing off. We sent out the post-survey a day after the main session in all rounds. We closed any access to the game or the material between case studies to make sure participants did not prematurely interacted with the platform.

4.2 Case Study Characteristics

There are multiple ways to design and conduct a case study since it is not a formally defined phenomenon. In order to convey our approach in this thesis, we define the characteristics of our case study and their justification in this section. The term explanatory case study comes from Robson's classification for research purposes [31]. These types of case studies mostly deal with testing existing theories in confirmatory studies [32]. We build our case study around time series design, consisting of pre measurements and post measurements. The structure of the case study is available in more detail in Figure 4.1.

Case studies are intended for real world settings therefore they are expected to be realistic. Our setting for the case studies, therefore, utilizes samples of our target audience (students of a software engineering course). They are also conducted as laboratory assignments in order to increase realism. The level of realism in a case study often decreases the level of control the researchers have [32]. In order to negate the lack of experimental control over the process, we utilize multiple ways of collecting both quantitative and qualitative data since their combination provides a better understanding of the research subject [33].

Triangulation is the process of combining qualitative and quantitative data in order to prevent the shortcomings of relying solely on one type. Among the four types of triangulation defined by Stake in "The Art of Case Study Research" [34], the thesis utilizes the following two:

- Data triangulation : This process refers to having multiple sources for data

collection which are divided by time. The thesis achieves this by performing the case study on three different groups on three occasions.

- Methodological triangulation: This process refers to having multiple types of data collection methods. The details regarding how we achieved this are detailed in the following section.

4.3 Data Collection

In this thesis, we utilized three different sources for collecting data. The first source consists of our surveys done before and after the virtual lab session. The pre-survey aims to determine the CR knowledge and overall experience of the participants before coming into the case study. Post-survey aims to measure the overall participant impressions, and individuals' subjective familiarity and confidence after the case study.

The second source consists of the pre and post-quizzes. They are in fact identical; therefore, we can directly observe how participants' answers change after playing the game. Although the main purpose of the quiz is to enable observation regarding learning, it also acts as a comprehensive quiz with questions on CR knowledge, defect taxonomy, and programming. The majority of the quiz consists of high-rated questions by the participants of the preliminary experiment interviews (step 6 in Figure 3.1). These are accompanied by a few questions that we added after the preliminary experiment, totaling up to 24 questions [23]. The quizzes were the same for all case studies.

Our third source for data collection is the player logs of the game. Our cloud database logs the majority of the player activity during gameplay. From this data, we can deduce the time spend on each level by each user, how many times a defect list was submitted to each level, the contents of the submission, and scores related to the submissions. This data is complemented by player IDs, emails, and names in order to recognize players and differentiate between participants of different rounds in order to conduct a more in-depth analysis.

Chapter 5

Empirical Analysis

The evaluations of the case study are done by utilizing the mentioned sources of data collection. The data collected from these materials are available online [1].

5.1 Survey Results

In order to provide a better understanding of the demographics of participants, we asked questions regarding their industry and Java language experience in the pre-survey. Most of the industry experience consists of summer internships and part-time jobs. On average for all case studies, 46% of the participants have 0 to 3 months while 18% have 3 to 6 months and 12% have more than 6 months of experience. 94% of the participants claim to have some practice with Java while 57% considered themselves intermediates. The rest of the questions were related to CR knowledge, importance, and related concepts alongside a question on participants' confidence regarding performing CR. These questions were asked on both surveys in order for us to evaluate the changes to the answers. A detailed view of the answers and differences can be seen in Figure A.1. Each bar in the figure represents the percentage of participants that chose the respective option

[1] https://figshare.com/s/567912dbc8e39e41350c

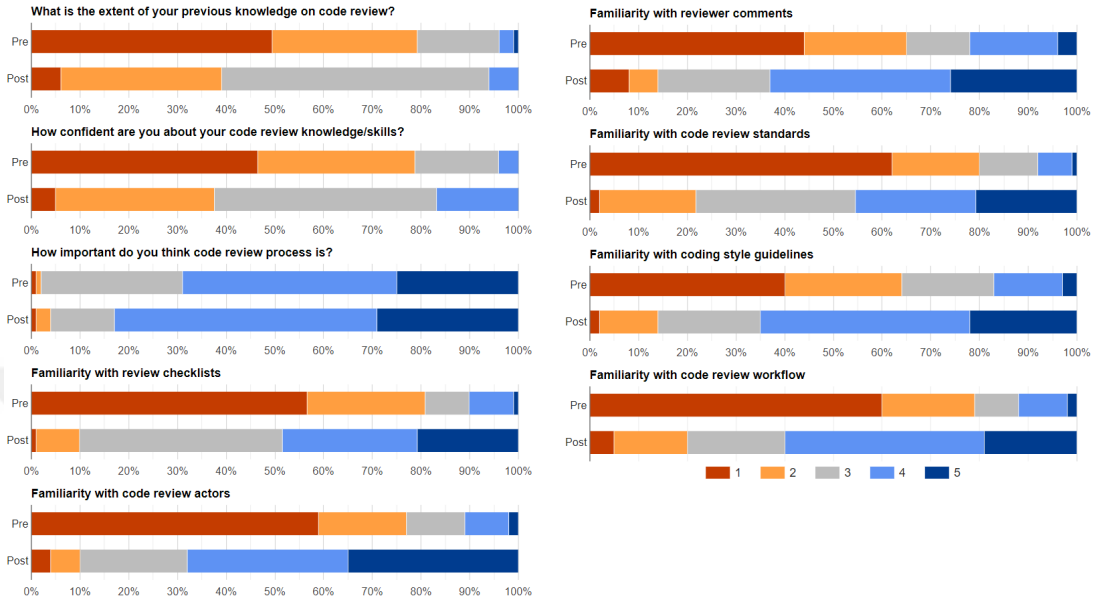

Figure 5.1: Results of the overlapping questions in the pre and post surveys.

from a 1 to 5 Likert scale. For the majority of the questions, we can see the shift to more positive options on the Likert scale from pre to post-survey. A summary of this figure is also provided in Table 5.1. From our analysis of both surveys, the most significant results we observe is that the participants' confidence regarding performing CR has improved by 54% approximately.

Question (Table A.1)	Pre	Post	Improvement
CR Knowledge (Q1)	1.76	2.61	48%
CR Confidence (Q2)	1.78	2.74	54%
CR Importance (Q3)	3.91	4.03	3%
CR Concepts (Q4)	1.88	3.64	94%

Table 5.1: Survey Summary

5.2 Quiz Results

The pre and post-quizzes consisted of the same 24 questions. The questions focus on 3 areas; CR, defect taxonomy, and programming. A detailed document

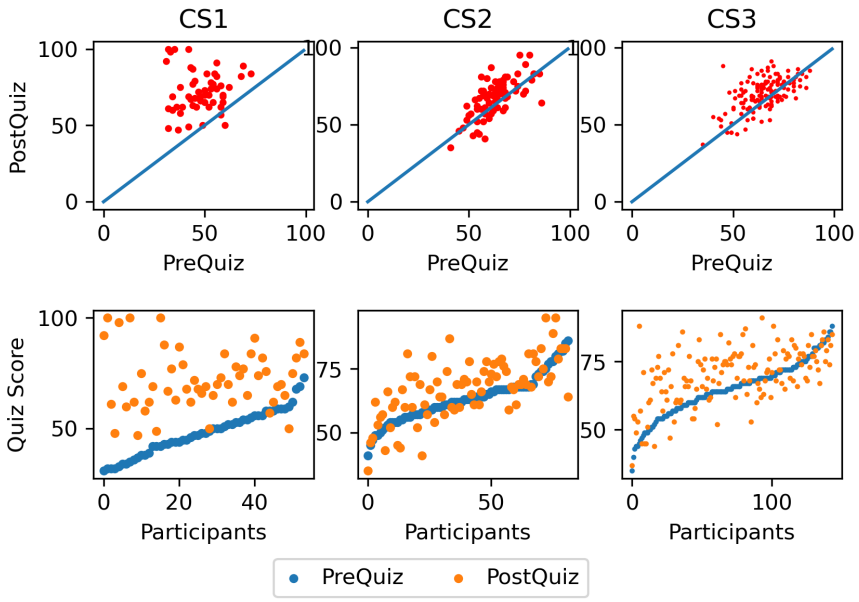

Figure 5.2: Changes in quiz scores

on the quiz alongside a summary of the quantitative results can be found in the appendix. The average score on the pre-quiz and post-quiz are 61 and 70 respectively. Overall score improvement between the installations of the quiz is around 10%. We summarized the average scores on the quizzes in Table 5.2 summarized according to the focus of the questions. Values inside the parenthesis are solely in regard to the results of round 1. As can be seen in Table 5.2, participants from CS319 have a higher improvement rate in parallel with them having lower scores in the pre-quiz. Because of this, we continue inspecting each case study round separately. Figure 5.2 demonstrates the distribution of quiz scores in each round. Each column in the figure corresponds to a case study round. The top row compares the pre and post-quizzes. Each point belongs to a participant and points over the $x = y$ line refer to a participant who improved their score after playing the game. The plots in the second row of Figure 5.2 are obtained by sorting the participants regarding their pre-quiz scores in ascending order where each vertical line on the plot that passes through one orange and one blue point represents a participant. Orange and blue points on the same vertical line refer to the post and pre-quiz scores of a participant respectively.

A paired-samples t-test was conducted to evaluate the significance of the quiz

results by comparing the success on the quiz before playing the game and after playing the game. The test indicates that there was a significant difference in the quiz results before gameplay ($M = 61.20, SD = 11.80$) and after gameplay, ($M = 69.68, SD = 11.55$) with $t = 10.66$ and p=.000. This particular test is applicable in our case since the students can be considered independent from each other and the overall distribution of the data draws closer to a normal distribution. This means that most of the data points are drawn in the neighborhood of the $x = y$ line if a scatter pilot is created from all quiz scores where pre-quiz and post-quiz are represented with x and y axes respectively.

5.3 Player Score Analysis

We used a cloud database for recording each submission of each player. The average score for the game levels besides the practice level is around 71%, while each level took about 10 minutes on average. More detailed information regarding the scores and time spent per level is made available in Table 5.3. We see that the time spent on singular defect is similar between all levels indicating that there were not any major drops of attention during the case study period.

In order to demonstrate the players' progress throughout the base game, we separated player scores for each level and calculated the average scores of all players on their specific number of submissions (e.g averaged scores from every fifth submission on level 3 for all the players). The results indicate a plot similar to linear lines as demonstrated in Figure 5.3. The linearity of the plots in the figure summarizes how consistently a player's effort is converted into points. The

Focus (Averaged)	Pre	Post	Improvement
CR Knowledge	208	219	11 (5%)
Defect Taxonomy	182	224	42 (23%)
Programming	184	197	13 (7%)

Table 5.2: Quiz Summary

data points of this section came from CS1 and CS2 which were later used to help determine the maximum number of submissions in CS3.

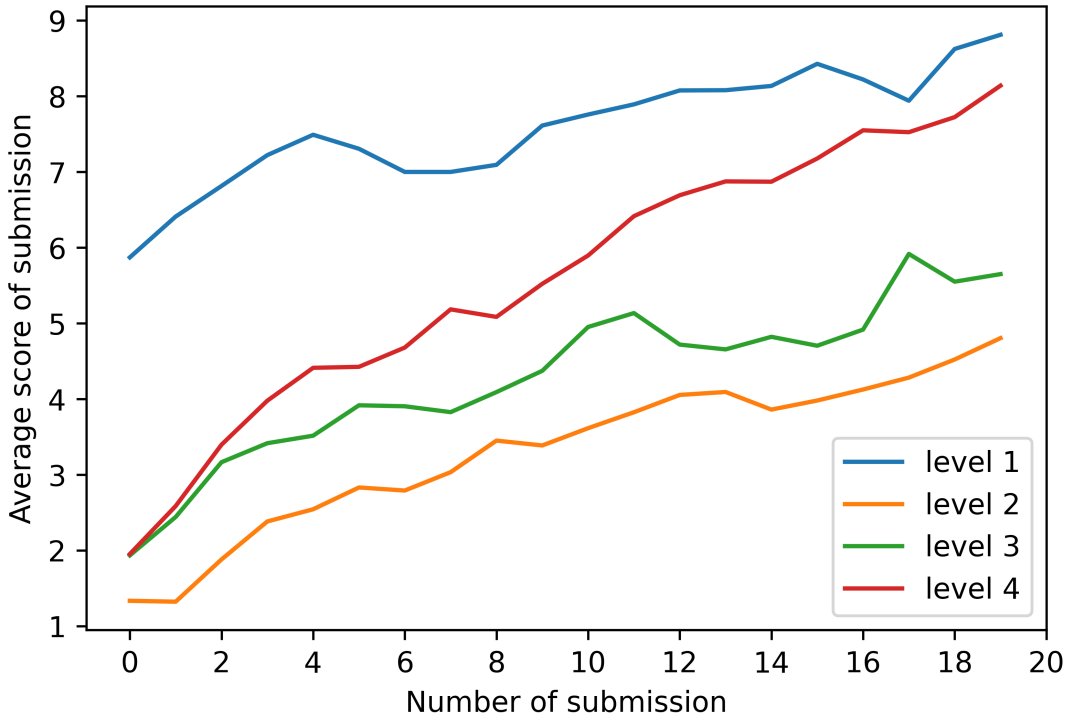

Figure 5.3: Average player scores per submission in reviewer mode

Level	Score(%)	Time Spent(s)	# of defects	Ts/Defect
1	73	557	4	139
2	64	733	4	183
3	76	376	3	125
4	71	780	6	130
Mean	71	612	-	144

Table 5.3: Player Averages per Level in CS1&CS2

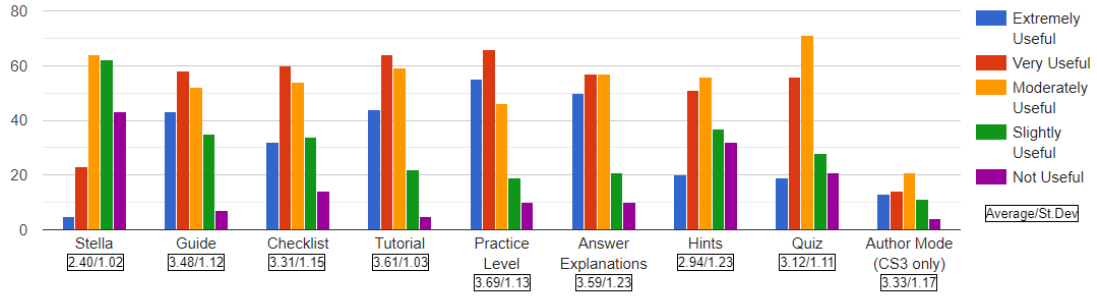

Figure 5.4: Evaluation of Game Components By Participants in CS1&CS2&CS3

5.4 Player Satisfaction and Feedback

This section presents our analysis regarding participants interactions with CRSG during the case studies.

Component Analysis data is gathered from each player in the post-survey in form of a question that lists the major game components and asks the participants to evaluate the usefulness of the components on a 5 point Likert scale. The distributions of participant responses and averages for each component can be seen in the Figure 5.4. According to the participant responses, the most useful component is "Answer Explanations" while the least useful component is "Stella". Standard deviations for all components are larger than 1, meaning that the majority of the participants agree with the average usefulness of a component. A relatively poor score for Stella is understandable with her being the only component that does not alter the game flow. For participants' convenience, we included the code review quiz and the author mode as a component in our component evaluation question in the post-survey as opposed to asking a separate question. The results indicate that nearly all components were received positively regarding their contribution to the game itself. The data in Figure 5.4 includes all case studies with the exception of author mode since it was developed between CS2 and CS3 as one can trace on Phase *IV* of Figure 3.1.

Free-text Answers

The post-survey included two free-text questions. In the first question, we asked participants to write down the things that they have learned during the case study. We have compiled their answers and performed open coding [35], where we organized each item from each participant into categories according to the concept the item was about. The final concepts we come up with at the end of open coding were "Defect detection skills", "Benefits of performing CR", "Benefits from practicing code inspection", "CR best practices", "Programming" and "Merit of serious games". We present the frequency of each concept in Figure 5.5. Note that on average each participant accounted for more than one item in the figure.

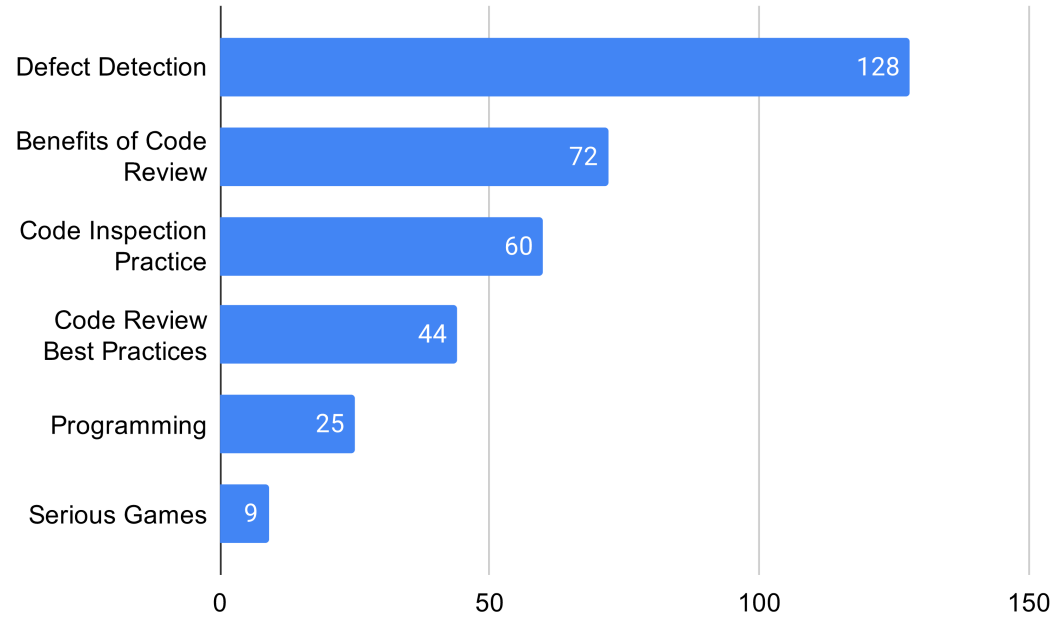

Figure 5.5: Open coding concepts extracted from the first free-text question

The second free-text question asked for feedback from the participants regarding their experience with the whole process. The negative part of the feedback was generally about user interface-related minor bugs in the system or shortcomings of a particular game component. The positive feedback consisted of feature suggestions and component specific&general praise. We provided some examples to these free-text answers below:

"-I find the game tutorial really useful since it was explaining each type of code error by giving examples. The examples were simple and explained the topic well."

"-I really liked the game. It was enjoyable and informative at the same time, but there's room for improvement. "

"-Stella is a nice touch, the information that she gives is full of important details of the code review process that is mostly overlooked. Adding another quiz for the information that Stella gives throughout the game, or updating the post-quiz for that matter, might be a good idea. "

Chapter 6

Discussion

6.1 Addressing Research Questions

RQ1: How effective is the code review serious game for introducing students to code review and its related concepts?

By investigating the data from the code review knowledge segment of the surveys, we see that participants believe they increased their code review knowledge and are more confident regarding conducting reviews. The shift to the more positive options on the Likert scale can be examined in Table A.1 while the averages of these options for all case study rounds can be observed in Table 5.1. We found survey data to be satisfactory regarding self evaluations of players. Especially the increase in students' confidence in Table 5.1 is evident of the code review serious games effectiveness regarding introducing code review. The quiz related data again shows that the players improved after playing, however, the interpretation of this factor is not as straightforward as survey data. While analyzing this portion, one first needs to realize the differences between rounds. The participants from CS319 are much more likely to have no prior knowledge regarding code review since the syllabus of CS453 involves a brief chapter in code review.

We observed the effects of these fundamental differences in rounds in the pre-quiz scores. The average of CS1 regarding the pre-quiz is around 50 while CS2 averaged around 65. The post-quiz averages are, however, much more similar.

There were a couple of variables that collectively resulted in CS1 showing more improvement. Firstly, a significant part of the points gathered from the quiz are not directly related to game contents, because the quiz was designed to be a standalone evaluation method. In parallel to these points, most of the improvement shown in the quiz comes from the defect taxonomy part, which is the part of the quiz that directly relates to gameplay. Another point of variation is the difference between the case study round dates. CS1 was done in lecture time more than a week before the university's "finals week" started. CS2 was done in the evening, two days before the start of the finals period. We believe that this was a contributing factor in the varying improvement rates between these rounds. As our expectations and data we collected indicate, the timing of our introductory activity seems to affect the outcomes. Because of scheduling and course load related factors, it is not always possible to perform this activity in its ideal setting, which is as an in-person, in-class activity. However, our students favored the interactive session to a regular class. In the CS3 post-survey, we added the following question to target this claim.

"*Do you think that this interactive session was more enjoyable and beneficial to you than a regular class?*"

Among the CS3 participants, 77 % of the respondents either agreed or strongly agreed to the question above on a 1 to 5 Likert scale. Students favoring these types of activities is enough of a reason to pursue them.

Considering the results of the case study and the factors above, we deduce that CRSG has merit regarding introducing code review practices. The extent of learning is not as easy to interpret from our limited data, however, it is possible to say that the activity better resonated with players who were completely unfamiliar with code review. The best use case for CRSG seems to be an "introduction to code review" activity for people who are unfamiliar with code review which is in

line with our purpose.

RQ2: How feasible is it to use the code review serious game to introduce code review concepts within a course curriculum?

We have already addressed the benefits of the code review serious game regarding the previous research question. Ease of application and time wise feasibility are remaining to be discussed. So far, we have presented CRSG as the main component of the case study in a setting which we use to evaluate the outcomes of the gameplay session. However, for the intended use case of the game as a code review focused class activity, feasibility plays a large role. In our case, the reserved time for the activity was 2.5 hours. In all rounds, the majority of the students were done with the session (from pre-quiz to post-quiz in Figure 4.1) around the two hour mark. We deduced that the gameplay (the game content portion of Figure 4.1) takes from 45 minutes to an hour for the majority while the rest of the time was spent on the other components. To perform the activity in a class setting, the quiz and survey components of the case study are not required to be included. Excluding these auxiliary components would allow for the gameplay to be completed in an hour indicating that the activity can be feasibly performed in class time or as an online session. With our online sessions, we observed the students to be largely autonomous, meaning that they had little to no confusion or technical problems during the gameplay. This was also considered to be a factor that increases the applicability of the code review serious game regardless of the setting. For short term evaluations of CRSG as a one-time activity, we can say that the evaluation results are satisfactory considering the time spent per participant interacting with the game content (demonstrated in Figure 4.1) during the case study is mostly consistent among levels. This indicates that the activity we presented to students was engaging.

Defect Detection Skills : On its own, theoretical code review knowledge is not enough for the development life-cycle to benefit from code review because reviewer experience is a contributing factor for a successful review process [36]. The practical benefits, besides education purposes, are tied to the reviewer's ability to detect defects or other unwanted attributes of the code that is in review.

We have broken down the process of detecting defects into two stages consisting of detecting the defect followed by determining the reason. The reason behind the defect was important to us because we have observed that some students were able to review the code but were inefficient or unable to communicate the defect, resulting in subpar review comments. To address this part of the communication process between the author and the reviewer, we integrated the defect taxonomy into our gameplay. The quiz had direct questions regarding the taxonomy and some programming question addressing both detection and reasoning, the results can be examined in Table 5.2. Participants seem to do well in taxonomy related questions with 23% improvement on average. However, we acknowledge that some of the taxonomy would be forgotten over time. Determining the long term benefits regarding overall defect detection skills would require follow-up review sessions or experiments.

6.2 Content Creation Challenges

Coming up with realistic CR scenarios in a limited setting, in 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 of "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 [1]. Therefore, we try to not include 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 simplistic errors like bad 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. Additionally, selecting Java as the language for our challenges amplified this effect, because most of the simple functional defects were detected in compile time and they were not desirable for game levels.

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 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.

there are known limitations of CRSG and for potential continuations of this thesis, we would like to state what can be done in the future to overcome them.

6.3 Limitations

First of all, since our need to evaluate the effectiveness of game components limited students gameplay time, we did not fully utilize the potential of the story elements that the game could have. This was a practical choice to save time for the case study evaluation materials. For CRSG to stand out as a complete package the story aspect needs to be better integrated into the gameplay by accompanying the player throughout the challenges. Another known limitation is the difficult process of creating content for CRSG as discussed above. But we would like to acknowledge that this process of content creation does not have to be done manually. There are two other ways of creating content.

First one assumes that a new version of CRSG with an increased scope that supports multiple files in a level is created. For such a version of CRSG, it should be possible to pull level content from actual code reviews performed on open

source projects. If such a suitable code review is detected, it can be used as a template for a new level. Another way of adding more content to CRSG is to utilize a data driven approach to automatically inject source code with defects. Being able to create such defected pieces of code is difficult and labor intensive but creating a successful application for this problem should also have a lot of potential applications in other fields.

Chapter 7

Threats to Validity

In order to decrease the uncontrollable variables and to extend our control over the case studies, we intended to conduct them as in person lab sessions. By doing so we would make sure that there would not be any communication between the participants, or they would not make use of online resources. Since we were not able to perform the case study in our preferred setting due to the Covid-19 pandemic, it was performed in a video conferencing format using Zoom [37] which was the preferred application by the university administration. Because of this change, we took some measures to protect the evaluation results. We prepared in game resources so that the participants would not seek help outside of the game. We also emphasized on several occasions that there was no incentive to utilize online resources. Additionally, we measured the time players spent where the level screen was out of focus (meaning that they were looking at some other program or page). These logs were also prompted to the players after the level screen was back in focus.

In order to prevent cheating during the activity, we emphasized that the grading would be done on a participation and completion basis instead of individual performance. Even with these precautions, we identified some participants that exploited the gameplay process, fortunately from our follow-up interactions with the participants, we found out their numbers are small enough (2 people) to not

affect the evaluation results. We did not consider their scores for the related part of the analysis. As we gained experience in performing this activity in an online format, additional features were added to the platform to keep the activity fair such as email verification or a maximum limit on the number of submissions made to the platform. Additionally, since we created the game content and its auxiliary components by ourselves, human errors could always affect the process negatively. We tried to negate this effect with the feedback from the preliminary experiment and interviews with its participants. This process is denoted in steps 5 to 9 in Figure 3.1.

Another required point of discussion is our quiz component of the case study structure. Ideally, one should avoid having students fill out the same quiz two times. However, since the case study aims to procure data regarding the changes that might occur before and after the gameplay, the most direct route of repeating the quiz was chosen intentionally. This is because any other measurement would either require us to balance two different quizzes regarding their content & difficulty or would have taken more than a single session the complete a case study. A more long term measurement could be provided by giving multiple code review tasks to students throughout the semester to observe the immediate or long terms improvements of students.

For our introductory activity, this type of measurement would be over the top because one also needs to consider conserving the workload balance of the classes and student's time. Another strategy that could have been employed for our measurements is to create a quiz that directly focuses entirely on the game itself. This was considered but not realized because we preferred to see the participants' results with a general code review quiz. The comprehensive aspect of these components provided us valuable insights to improve our own teaching material as well because we were able to observe which pieces of information were missing from the students after all code review related class activities were completed (these activities end with the post-quiz).

Moreover, we also acknowledge that applying the same test twice could also affect the results gathered from the post-quiz because participants have interacted

with the same questions in the pre-quiz. This phenomenon is known as a testing threat which is prevalent in single group pre-post research methodologies. In our case, the limited time frame of the case studies is a mitigating factor for the testing threat because the participants did not have any free time between the quizzes. This means that they also did not have the time to passively think and improve their answers to the quiz questions on their own. Furthermore, one could also say that a certain decrease in the interest regarding the questions would have manifested since the participants are taking the same test a second time. This decrease in the interest would have lead to a decrease in the overall score. We believe that, when combined, these two factors mitigated most of the testing threat. We also suspect that the latter factor played a role in the post-quiz results by hiding some portion of the participant improvement from the data.

The rest of this chapter investigates the external validity of our results which is defined as the generalizability of sample results to the population of interest, across different measures, persons, settings, or times [38]. The target audience for the code review serious game is university-level students in departments whose graduates could fill software development related positions (e.g software engineering, computer science). By selecting our participants directly from Bilkent university's software engineering focused courses CS319 [29] and CS453 [30], we intend to match the case study audience with the population of interest of the code review serious game. Using these courses as the audience guarantees a uniform minimum background for participants since in order to take CS319 one needs to complete two introductory programming and two data structure related courses. CS453 has an additional prerequisite which is CS319. To further increase our claim of generalizability regarding the results of our case study, in the future, we intend to continue to perform the activity annually in these courses while implementing it in other related courses both inside and outside of Bilkent university by collaborating with other instructors.

Chapter 8

Conclusion

Throughout this thesis, we have proposed, designed, implemented, and evaluated the code review serious game. The motivation for the code review serious game comes from the lack of representation regarding code review in curricula of computer science, software engineering, etc. We have designed the game to directly address our learning objectives and performed a preliminary evaluation in order to gather feedback and in turn improve the game to better meet the requirements of the learning objectives.

These learning objectives were:

- Understanding Code Review Related Concepts
- Identifying Code Errors and Smells

Utilizing the feedback from the preliminary experiment we have adjusted our auxiliary material(quizzes, surveys, guides, pre-gameplay tutorials, etc.) for better clarity. Also, the interview data allowed us to trim the quizzes to meet our desired experiment duration.

We then proceeded to the large scale evaluations of the game CS1 and CS2 with 132 students. Inline with the student feedback from these case studies, we

designed and implemented the author mode. This new gameplay mode allowed us to fully reflect the duties of both code review roles in CRSG thus completing all major desired features of CRSG. To observe the author mode and all other improvements in action we conducted the third case study (CS3) with 144 students.

During each case study, participants filled pre and post gameplay surveys and quizzes that we use to collect data for evaluating the game. We also bring in an additional data collection method by recording the player activity during gameplay. We present our data from all sources and our analysis where we attempt to answer our 2 research questions that refer to teaching CR practices and the overall feasibility of using the code review serious game as an in class activity. The results indicate that the game can be used as an introductory activity regarding the code review practice. The benefits of the game for students can be summarized as follows:

- Understanding the workflow of the code review process.
- Learning the benefits and best practices of the code review process.
- Learning different categories of errors that can come up while reviewing code.

In order to further document the research and enable educators to utilize the code review serious game, we share our case study materials as well as the data and the game itself as online resources. In the future, we would like to add a collaborative multiplayer section to the game to address team building and knowledge transfer during the code review process, making the platform an even more complete package. Additionally, we would like to increase the scope of the game levels by making levels depend on multiple files or other development related artifacts like requirements. This improvement alongside a way to add more content to the game without modifying its source code would sharply increase the viability of CRSG as an industry oriented training tool.

Bibliography

[1] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at google," in *International Conference on Software Engineering (Software Engineering in Practice)* (F. Paulisch and J. Bosch, eds.), pp. 181–190, ACM, 2018.

[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 *International Conference on Software Engineering (Software Engineering in Practice)* (H. Sharp and M. Whalen, eds.), pp. 255–264, IEEE / ACM, 2019.

[3] S. Sripada, Y. R. Reddy, and A. Sureka, "In support of peer code review and inspection in an undergraduate software engineering course," in *2015 IEEE 28th Conference on Software Engineering Education and Training*, pp. 3–6, 2015.

[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.

[5] J. Hamari, D. Shernoff, E. Rowe, B. Coller, J. Asbell-Clarke, and T. Edwards, "Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning," *Computers in Human Behavior*, 08 2016.

[6] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, (New York, NY, USA), p. 202–211, Association for Computing Machinery, 2014.

[7] SMARTBEAR, "The state of code review in 2019: Trends, tools, and insights for dev collaboration." `https://smartbear.com/resources/ebooks/the-state-of-code-review-2019/`, 2019. (Accessed on 08/28/2020).

[8] 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.

[9] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, (New York, NY, USA), p. 192–201, Association for Computing Machinery, 2014.

[10] M. Souza, L. Veado, R. T. Moreira, E. Figueiredo, and H. Costa, "A systematic mapping study on game-related methods for software engineering education," *Information and Software Technology*, vol. 95, pp. 201 – 218, 2018.

[11] E. O. Navarro and A. van der Hoek, "Design and evaluation of an educational software process simulation environment and associated model," in *18th Conference on Software Engineering Education Training (CSEET'05)*, pp. 25–32, 2005.

[12] K. Berkling and C. Thomas, "Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure," in *2013 International Conference on Interactive Collaborative Learning (ICL)*, pp. 525–530, 2013.

[13] C. Thomas and K. Berkling, "Redesign of a gamified software engineering course," in *2013 International Conference on Interactive Collaborative Learning (ICL)*, pp. 778–786, 2013.

[14] P. Sonchan and S. Ramingwong, "Armi 2.0: An online risk management simulation," in *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 1–5, 2015.

[15] J. M. Rojas and G. Fraser, "Code defenders: A mutation testing game," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 162–167, 2016.

[16] C. G. von Wangenheim, R. Savi, and A. F. Borgatto, "Scrumia—an educational game for teaching scrum in computing courses," *Journal of Systems and Software*, vol. 86, no. 10, pp. 2675–2687, 2013.

[17] A. Baker, E. Navarro, and A. van der Hoek, "Problems and programmers: an educational software engineering card game," in *25th International Conference on Software Engineering, 2003. Proceedings.*, pp. 614–619, 2003.

[18] M. Ulicsak and M. Wright, "Games in education: Serious games," *Futurelab*, 2010.

[19] 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)*, pp. 36–39, IEEE, 2013.

[20] 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*, pp. 1117–1126, IEEE Press, 2013.

[21] R. Atal and A. Sureka, "Anukarna: A software engineering simulation game for teaching practical decision making in peer code review," in *Joint Proceedings of the 3rd International Workshop on Quantitative Approaches to Software Quality (QuASoQ), the Workshop on Alternate Workforces for Software Engineering (WAWSE) and the 1st International Workshop on Case*

Method for Computing Education (CMCE) co-located with APSEC 2015, pp. 63–70, 2015.

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

[23] B. Ardiç, I. Yurdakul, and E. Tüzün, "Creation of a serious game for teaching code review: An experience report," in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)*, pp. 1–5, 2020.

[24] K. Ünlü, B. Ardıç, and E. Tüzün, "Crsg: A serious game for teaching code review," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, (New York, NY, USA), p. 1561–1565, Association for Computing Machinery, 2020.

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

[26] G. Rong, J. Li, M. Xie, and T. Zheng, "The effect of checklist in code review for inexperienced students: An empirical study," in *Conference on Software Engineering Education and Training (CSEET)* (D. Chen, M. Barker, and L. Huang, eds.), pp. 120–124, IEEE Computer Society, 2012.

[27] T. inc., "Thoughtbot guides." `https://github.com/thoughtbot/guides/tree/master/code-review`, 2019. Accessed on 08/09/2020.

[28] B. A. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case studies for method and tool evaluation.," *IEEE Software*, vol. 12, no. 4, pp. 52–62, 1995.

[29] Bilkent University, "Syllabus of cs 319 - object-oriented software engineering." `https://stars.bilkent.edu.tr/syllabus/view/CS/319/`, 2020. Accessed on 08/09/2020.

[30] Bilkent University, "Syllabus of cs 453 - application lifecycle management." `https://stars.bilkent.edu.tr/syllabus/view/CS/453/`, 2020. Accessed on 08/09/2020.

[31] C. Robson, *Real world research: A resource for social scientists and practitioner-researchers*. Wiley-Blackwell, 2002.

[32] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[33] C. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[34] R. E. Stake, *The art of case study research*. sage, 1995.

[35] A. Strauss and J. Corbin, *Basics of qualitative research*. Sage publications, 1990.

[36] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?," in *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ICSME '15, (USA), p. 111–120, IEEE Computer Society, 2015.

[37] Zoom Video Communications Inc, "Video conferencing, web conferencing, webinars, screen sharing." `https://zoom.us/meetings`, 2020. Accessed on 08/09/2020.

[38] W. R. King and J. He, "External validity in is survey research.," *Commun. Assoc. Inf. Syst.*, vol. 16, p. 45, 2005.

Appendix A

Quiz and Survey Questions

A.1 Survey Questions

---

1 How long is your current industry experience including internships and part-time jobs?

1. None
2. 0-3 Months
3. 3-6 Months
4. 6-12 Months
5. 1-3 Years
6. 3+ Years

---

2 What is your proficiency level related to Java programming language?

1. Theoretical knowledge, but no working experience
2. Beginner with some working knowledge

3. Intermediate with practical application
4. Advanced with significant experience
5. Guru with ability to mentor others

---

3 What are your expectations for playing this game?

1. $\boxed{Free-text\ \ Question...}$

---

4 What is the extent of your previous knowledge on code review?

1. None.
2. I have basic familiarity with CR, but no experience applying it.
3. I have knowledge of CR and used it in small or artificial projects.
4. I have applied CR in real-world setting.
5. I have applied CR extensively in a real-world setting.

---

5 What is the definition of code review? Please explain it briefly. If you answered the above question with a "no", skip this question.

1. $\boxed{Free-Text\ \ Question...}$

---

6 If you participated in code review before, which code review system/method did you utilize?

1. I did not participate..
2. Over the shoulder / In person
3. Gerrit
4. GitHub Code Review

5. Microsoft TFS
6. Crucible
7. E-mail
8. Other..

---

7 How confident are you about your code review knowledge/skills?

1. Not confident at all.
2. Slightly confident.
3. Somewhat confident.
4. Fairly confident.
5. Extremely confident.

---

8 How important do you think code review process is?

1. Not important at all: I could not see any reason to make code review important.
2. Not very important: There are some points that makes the code review important, but in general it is not.
3. Fairly important: Although there are reasons for making the code review important I could not list them.
4. Very Important: There are obvious reasons in order to call code review important.
5. Fundamental: Code review is one of the most important processes in software development.

---

9 How familiar are you with the following code review concepts?

	Unfamiliar	Slightly familiar	Moderately familiar	Familiar	Very familiar
Review Checklists					
CR Actors					
Reviewer Comments					
CR Standards					
Style Guidelines					
CR Workflow					

A.2 Post-Survey Questions

1 How much have you enjoyed the game elements in the serious game?

1. Very Enjoyable: The game elements were highly motivating and encouraged further game-play.
2. Enjoyable: The game elements were fairly enjoyable and somehow encouraged further game-play.
3. Neutral: The game elements were pretty, but weren't really effective besides that.
4. Boring: The game elements were not attractive or enjoyable, didn't incentivise me to continue playing.
5. Very Boring: The game elements were dull and heavily detracted from my concentration.

---

2 How do you feel about the impact of game elements in the teaching of the code review process?

1. Very Satisfied: Game elements significantly simplified the code review learning process and made it more enjoyable.
2. Satisfied: Game elements somehow simplified the learning process, but cannot become a staple for the learning process.
3. Neutral: Game elements helped the learning process but is redundant compared to other learning methods.
4. Unsatisfied: Game elements made the learning process inconvenient and watered down, and shouldn't have been there.
5. Very Unsatisfied: Game elements detracted from the learning experience past the point of uselessness to the point to harmful, giving wrong information.

---

3 Do you think that this interactive session was more enjoyable and beneficial to you than a regular class?

1. Strongly Agree
2. Agree
3. Neither agree nor disagree
4. Disagree
5. Strongly Disagree

---

4 Please provide explicit reasons for your dissatisfaction if they exist.

1. $\boxed{Free-Text\ \ Question...}$

---

5 Please rate the game components according to their usefulness.

	Not useful	Slightly useful	Moderately useful	Very useful	Extremely useful
Stella's log					
In game guide					
Checklist					
Game Tutorial					
Practice level [level 0]					
Answer Explanations					
Hints					
Quiz					
Author Mode [level 2,3]					
Reviewer Mode [level1,4]					

---

6 What are the 2 things you learned while playing the game? [Please number the items and use full sentences]

1. $\boxed{Free-Text\ \ Question...}$

---

7 Do you have any positive or negative feedback for the game?[Please number the items and use full sentences]

1. $\boxed{Free-Text\ \ Question...}$

8 What is the extent of your previous knowledge on code review?

1. None.
2. I have basic familiarity with CR, but no experience applying it.
3. I have knowledge of CR and used it in small or artificial projects.
4. I have applied CR in real-world setting.
5. I have applied CR extensively in a real-world setting.

---

9 How confident are you about your code review knowledge/skills?

1. Not confident at all.
2. Slightly confident.
3. Somewhat confident.
4. Fairly confident.
5. Extremely confident.

---

10 What is the definition of code review? Please explain briefly.

1. $\boxed{Free-Text\ \ Question...}$

---

11 How important do you think code review process is?

1. Not important at all
2. Not very important
3. Fairly important
4. Very Important
5. Fundamental

12 How familiar are you with the following code review concepts?

-	Unfamiliar	Slightly familiar	Moderately familiar	Familiar	Very familiar
Review Checklists					
CR Actors					
Reviewer Comments					
CR Standards					
Style Guidelines					
CR Workflow					

A.3 Quiz Questions

1 Which of the following are among the benefits of code review (True/False) ?

1. Code improvement
2. Finding of defects
3. Transference of knowledge to managers
4. Exploration of alternative solutions
5. Testing of code
6. Sharing of code's responsibility

---

2 Which of the following are the duties of a code reviewer (True/False) ?

1. Help improve code quality
2. Find bugs
3. Fix bugs
4. Implement alternative solutions
5. Make compilable code
6. Help insure design patterns

---

3 Which of the following statements are true?

1. Reviewer should edit the code in order to fix it after understanding the defect description.
2. There is a preferable time constraint for an effective code review attempt.

---

4 Which of the following are fundamentals of the code review process (Yes/No) ?

1. Checklists

2. Command Terminal
3. Understanding the code's purpose
4. Development IDE

---

5 Assume that a function is technically correct (working as expected) but the reviewer thinks there are things that need to be changed in order for it to be easier to understand and maintain. What changes might this reviewer be talking about (True/False) ?

1. Naming
2. Commenting
3. Duplication of helper functions
4. Completeness
5. Correcting the code's function

---

6 Which of the following are organisation related structural defects (True/False) ?

1. Variable initialization
2. Long subroutine
3. Duplication
4. Dead code
5. Indentation
6. Consistency
7. Element Type

---

7 Which of the following are logic related functional defects (Yes/No) ?

1. Semantic duplication
2. Bad/inconsistent variable initialisation

3. Sub-optimal algorithm/performance
4. Bad compare operations
5. Semantic dead code
6. Bad compute operations

---

8 What are the two main actors of the code review process (Yes/No) ?

1. Reviewer
2. Manager
3. Author
4. Investor
5. Tester

---

9 In the following code review definition which word should be changed or discarded? "An automated inspection of source code by developers other than the author which can be done both individually or as a group."

1. Automated
2. Individually
3. As a group
4. Developers

---

10 In which state of the software development life-cycle is finding bugs the least economically impactful on the project?

1. Test
2. Review
3. Release

11 Which of the following might be considered potential side benefits of the code review process? [Note: Select one or more answer choices]

1. Knowledge transfer among colleagues
2. Helping the others regularly
3. Higher code quality

---

12 Which of the following is not a good strategy to set up an effective and efficient code review session as the author?

1. Making sure commit messages and pull request descriptions are informative.
2. Studying your code to be able to defend and avoid changing it.
3. Using a static-code analysis tool to eliminate errors detectable by machines before the review.
4. Making an effort to find a person who is likely to find errors in the code.

---

13 Inspect the following code block. This piece of code obviously does not run, however, the compiler does not raise an error because of a defect. Find that "defect" and answer this question and the one below it.
Is there a functionality error or an evolvability error in this code?

```
List listOfNumbers = new ArrayList();
listOfNumbers.add(10);
listOfNumbers.add("Twenty");
listOfNumbers.forEach(n -> System.out.println((int) n * 2));
```

1. Functional
2. Evolvability

---

14 What do you think is the reason for the error?

1. Consistency
2. Void Parameter
3. Element Type
4. Variable Initialization
5. Compute

---

15 Inspect the following code block, then answer the following four questions. Is the actual output of this code different than the intended output?

```
String s = “abc”;
s.toUpperCase();
System.out.println(s);
```

16 What is the reason for the difference?

1. No difference
2. Immutable data type
3. Variable Initialization
4. Function Call

---

17 What is the actual output? [Use only letters only in the answer. Do not put “ or ’]

1. $\boxed{Free-Text\ \ Question...}$

---

18 What is the expected output?[Use only letters only in the answer. Do not put ” or ’]

1. $\boxed{Free-Text\ \ Question...}$

---

19 Which of the following is not a proper item for a code review checklist?

1. Pick one word per concept for variables (use consistent terminology for variable names)

2. Explain yourself in code
3. Avoid returning null
4. Avoid Immutable final variables

---

20 Inspect the following code piece, identify what the author is trying to achieve and answer the following two questions.
How many times will the statement inside the for loop execute?

```
public class Warmup {
        public static void main(String[] args) {
                int END = Integer.MAX_VALUE;
                int START = END - 100;
                int count = 0;
                for (int i = START; i <= END; i++) {
                        count++;
                }
        System.out.println(count);
        }
}
```

1. 0
2. 1
3. 100
4. 101
5. ∞

---

21 What would you change about this piece of code?

1. $\boxed{Free-Text\ \ Question...}$

---

22 Inspect the following code snippet. Unfortunately, its author did not test it well. Now, as the reviewer, you must find the defect that they have missed. In order to help you can use the test case provided below the line '// Test Code'. Answer the following two questions.
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 Code
public static void main(String[] args){
        for(int i = 0; i<4;i++){
                switchCasePrimer(i);
        }
}
```

1. 0 1 1 2 3
2. 0 1 2 3
3. 0 1 2 2 3
4. 1 1 2 3

---

23 What is the problem that you have identified?

1. $\boxed{Free - Text\ \ Question...}$

A.4 Result Summary Tables

Question	As %	5	4	3	2	1
What is the extent of your previous knowledge on CR?	Pre	1	3	17	30	50
	Post	0	6	55	33	6
	Diff	-1	3	38	3	-44
How confident are you about your CR knowledge/skills?	Pre	0	4	17	32	46
	Post	0	17	46	33	5
	Diff	0	13	29	1	-41
How important do you think the CR process is?	Pre	25	44	29	1	1
	Post	25	54	13	3	1
	Diff	0	10	-16	2	0
How familiar are you with the following CR concepts?						
Review Checklists	Pre	1	9	9	24	56
	Post	21	28	42	9	1
	Diff	20	19	33	-15	-55
Code Review Actors	Pre	2	9	12	18	59
	Post	35	33	22	6	4
	Diff	33	24	10	-12	-55
Reviewer Comments	Pre	4	18	13	21	44
	Post	26	37	23	6	8
	Diff	22	19	10	-15	-36
Code Review Standards	Pre	1	7	12	18	62
	Post	21	25	33	20	2
	Diff	20	18	21	2	-60
Coding Style Guidelines	Pre	3	14	19	24	40
	Post	22	43	21	12	2
	Diff	19	29	2	-12	-38
Code Review Workflow	Pre	2	10	9	19	60
	Post	19	41	20	15	5
	Diff	17	31	11	-4	-55

Table A.1: Survey Results

Question		Pre	Post	Diff
1 What is your full name?				
2 Which of the following are the benefits of code review?	Code Improvement	269	271	2
	Finding of defects	274	278	4
	Transference of knowledge to managers	155	197	42
	Exploration of alternative solutions	229	204	-25
	Testing of code	116	144	28
	Sharing of code's responsibility	165	167	2
3 Which of the following are the duties of a code reviewer?	Help improve code quality	266	276	10
	Find bugs	273	268	-5
	Fix bugs	215	225	10
	Implement alternative solutions	189	207	18
	Make compilable code	183	175	-8
	Help insure design patterns	215	165	-50
	Help improve code consistency	270	278	8
4 Which of the following statements are true?	Reviewer should edit the code in order to fix it after understanding the defect description.	198	211	13
	There is a preferable time constraint for an effective code review attempt.	232	224	-8
5 Which are fundamental for code review process?	Checklists	268	272	4
	Command Terminal	183	225	42
	Understanding the code's purpose	168	274	106
	Development IDE	206	215	9
6 Assume that a function is working as expected but the reviewer thinks that the code needs changes to be easier to understand and maintain. What changes might this reviewer be talking about?				
	Naming	269	274	5
	Commenting	266	273	7
	Duplication of helper functions	170	187	17
	Completeness	161	158	-3
	Correcting the code's function	248	222	-26
7 Which of the following are organisation related structural defects?	Variable initialization	137	213	76
	Long subroutine	93	175	82
	Duplication	224	251	27
	Dead code	173	238	65
	Indentation	88	136	48
	Consistency	180	233	53
	Element Type	160	192	32
8 Which of the following are logic related functional defects?	Semantic duplication	124	260	136
	Bad variable initialisation	114	174	60
	Sub-optimal algorithm/performance	224	260	36
	Bad compare operations	253	273	20
	Semantic dead code	137	243	106
	Bad compute operations	254	270	16

Table A.2: Quiz Result Summary Part 1

Question		Pre	Post	Diff
9 What are the two main actors of the code review process?	Reviewer	276	278	2
	Manager	260	272	12
	Author	227	259	32
	Investor	276	277	1
	Tester	228	259	31
10 In the following code review definition which word should be changed or discarded? "An automated inspection of source code by developers other than the author which can be done both individually or as a group."		208	217	9
11 In which state of the software development life-cycle is finding bugs the least economically impactful on the project?		182	200	18
12 Which of the following might be considered potential side benefits of the code review process?		9	13	4
13 Which of the following is not a good strategy to set up an effective and efficient code review session as the author?	Making sure commit messages and PR descriptions are informative.	235	240	5
	Studying your code to be able to defend and avoid changing it.	195	190	-5
	Using a static-code analysis tool to eliminate errors detectable by machines before the review.	198	208	10
	Making an effort to find a person who is likely to find errors in the code.	197	206	9
14 Coding Question 1.1		188	133	–55
15 Coding Question 1.2		227	221	-6
16 Coding Question 2.1		185	189	4
17 Coding Question 2.2		101	192	91
18 Coding Question 2.3		221	245	24
19 Coding Question 2.4		269	271	2
20 Which of the following is not a proper code review checklist item?		115	123	8
21 Coding Question 3.1		56	95	39
22 What would you change about this piece of code? (free-text question)				
23 Coding Question 4.1		226	233	7
24 What is the problem that you have identified? (free-text question)				

Table A.3: Quiz Result Summary Part 2