A Serious Game Approach to Introduce the Code Review Practice

Ardic, Baris; Tuzun, Eray

DOI
10.1002/smr.2750

Publication date
2024

Document Version
Final published version

Published in
Journal of Software: Evolution and Process

Journal of Software: Evolution and Process

WILEY

Journal of Software: Evolution and Process

RESEARCH ARTICLE - EMPIRICAL OPEN ACCESS

A Serious Game Approach to Introduce the Code Review Practice

Baris Ardic[1] | Eray Tuzun[2]

[1]Department of Computer Science, Delft University of Technology, Delft, The Netherlands | [2]Department of Computer Engineering, Bilkent University, Ankara, Turkey

Correspondence: Baris Ardic (b.ardic@tudelft.nl)

Received: 22 June 2022 | **Revised:** 20 June 2024 | **Accepted:** 24 November 2024

Funding: This study was supported by the Schweizerischer Nationalfonds zur Förderung der Wissenschaftlichen Forschung.

Keywords: code inspection | code review | serious games | software engineering education

ABSTRACT

Code review is a widely utilized 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 the introduction of the code review process. To realize these objectives, we design, build, and test the serious game. We then conduct three case studies with a total of 280 students. We evaluated the results by comparing the student's knowledge and confidence about code review before and after case studies, as well as evaluating how they performed in code review quizzes and game levels themselves. Our analysis indicates that students had a positive experience during gameplay, and an in-depth examination suggests that playing the game also enhanced their knowledge. We conclude that the game had a positive impact on introducing the code review process. This study represents a step taken toward moving code review education from industry starting positions to higher education. The game and its auxiliary materials are available online.

1 | Introduction

The code review process is established as an essential part of the application lifecycle management and is frequently applied in modern software development [1]. Performing code reviews properly has been shown to play an important role in reducing software defects and improving software quality [2]. Despite widespread usage and the emphasis given in the industry [3], code review practice is often not adequately addressed in typical Software Engineering or Computer Science curricula [4, 5]. Furthermore, during a literature review on peer code review education, Indriasari et al. [6] reported the lack of student learning engagement and consistency in the review quality to be major barriers to establishing an effective code review process. Our personal experiences with teaching software engineering courses are paralleled by these barriers; we noticed a lack of maturity in the students' code review processes, attributing this to their limited training on the subject when using tool-supported reviews. Therefore, we have the following overall research goal:

> To evaluate how a serious game approach improves software engineering students' comprehension and execution of the code review process, particularly in addressing engagement and quality issues.

Addressing these barriers requires teaching best practices, workflow, and potential code quality improvements in the code review process with greater student engagement. Serious games are a viable way to address these barriers because game-based formats are proven to increase user engagement [7]. Unlike

traditional games, serious games have educational purposes and can be designed around learning objectives [8].

To address these barriers, educators would require a platform that conveys the basics of the code review process while allowing participants to practice their defect detection skills. Moreover, any additional exercise demonstrating the author'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 begin participating in a more mature code review process.

Another useful effect of such a platform is to help with moving code review–related education and training from the industry to higher education. This shift could decrease the orientation load of new engineers while obtaining code review–related knowledge earlier should help students during their undergraduate software projects.

We already proposed a prototype serious game–based approach as the platform for introducing the code review process in our previous study [9] and proceeded with a design overhaul [10]. The initial design of the code review serious game (CRSG) was promising based on a small trial run.

Moreover, using a custom exercise platform allows us to have more control over how the process is presented and how interaction data can be collected. Therefore, to introduce the code review process in a class environment, this platform might be preferable to existing industrial code review tools. Another strength of a game approach is the perceived simplicity compared to more industry-oriented options.

To further validate our approach, we present the following research questions:

RQ1. How effective is CRSG for introducing students to code review and its related concepts?

RQ2. How feasible it is to use CRSG to introduce code review concepts within a course curriculum?

This study reports on the latest stage of our larger project in delivering this platform. The main objective of this stage of the overall study is to evaluate CRSG to see if it can be integrated into software engineering–related courses without occupying lecture time. The overall study made the following contributions:

- Developed the first publicly available CRSG[1]
- Designed quizzes and surveys to be able to evaluate the effectiveness of CRSG.[2]

This study's main contribution is the following:

- Conducted multiple case studies with a total of 280 unique students to evaluate CRSG on scale, and then, we shared our results and findings. We also made improvements to the platform between the case studies by developing a new game mode.

This paper reports on the last version that has a new game mode. This additional game mode allows us to approach the code review from the author's perspective, complementing the reviewer's viewpoint in the original mode. However, the actual focus is on the evaluation of CRSG. Building upon the previous studies [9, 10] that detailed the prototype, this paper reports the analysis of all three case studies conducted throughout a period of two years. The scope of this paper is to analyze all the data that we gathered throughout the overall process and to determine the role of CRSG in our arsenal of software engineering education. We also reflect on our experiences regarding building an in-house learning tool and threats to its effectiveness.

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

2 | Background

Code review is a manual inspection of source code by developers other than the author of the source code [2]. A simplified overview of the code review process is provided in Figure 1, which is adapted from [11]. The process starts with the initial code segment, which is altered until all concerns of the reviewer 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 seen as a vital part by the leading software companies in the industry [3, 12]. Most of the foundational knowledge in the literature on modern code review comes from analyzing open-source projects [13].

Collaborations between researchers and companies are not absent from the literature either because a better understanding of the process is directly beneficial for practitioners. A case study done in Google [2] shows that a mature and mandatory code review process was accepted and its benefits were acknowledged by the engineers. Another study by Microsoft [14] compiles the challenges to expect and the best practices to apply while establishing a modern code review process. In addition to such collaborative studies, the benefits of code review have been a topic of empirical research in software engineering. McIntosh et al. [15]

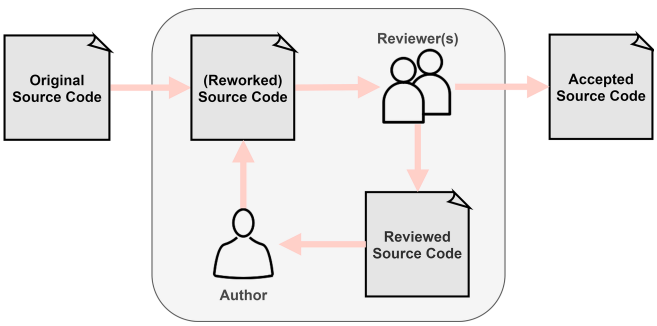

FIGURE 1 | An overview of the code review process [11].

have provided empirical evidence regarding the relationship between code review coverage and postrelease defects. Using code review data from popular and well-documented open-source projects such as Qt,[3] they show that both low code review coverage and participation produce artifacts with up to five additional postrelease defects. Similarly, Doğan and Tüzün [16] show that popular open-source projects are prone to code review–related process smells as they detect at least one process smell in around 72% of the open-source code reviews. Conducting the process correctly seems to be a nontrivial problem increasing the importance of code review education for the next generations of practitioners.

The rest of this section 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.

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. [17] investigates 156 studies between 1974 and 2016 about the use of game concepts to teach software engineering practices. A common motivating theme for these studies is the shortcomings in university lectures on software development–specific concepts.

There are two main approaches regarding the scope of these studies. First are the studies that combine multiple development concepts or oversee software development as a whole in their game flow; therefore, focus on multiple aspects of software engineering at once. A leading study in this regard is simSE by Navarro and van der Hoek [18] where the player controls a software engineering project by making decisions from a managerial perspective. Second are the studies that focus on a specific software engineering process or skill. For example, Sonchan and Ramingwong [19] tackle software engineering risk management skills with the card-based game ARMI where players identify and analyze the risks to strategize correctly for risk management during a software development project.

It is also possible to integrate a traditional course format with an interactive platform where the platform itself facilitates better learning by helping the instructor convey information. For instance, Farah et al. [20] utilize code review notebooks to introduce the process via code snippets in an online setting. Similarly, Haendler et al. [21] proposed a tutoring system for code refactoring training where they integrated feedback for the correctness and design quality.

2.2 | Serious Games for Code Review

Serious games are learning tools that aim to increase player engagement by leveraging game elements. They are learner-centered approaches, where the user controls the learning process in an interactive manner [22]. Unlike a traditional game, they are designed around learning objectives [23]. We have compiled the existing studies that were the closest to this one via purpose or execution. The purpose is to teach the code review practice in a higher education setting, and a serious game is used to fulfill this goal.

Pex4Fun is a serious game designed by Xie et al. [24, 25]. The gameplay consists of coding duels where players aim to reach the correct behavior by introducing code modifications. Players are provided feedback for these changes and the overall process indirectly simulates code review. Skills like software testing, debugging, and code inspection are practiced throughout the game.

Anukarna is a decision-making-based serious game by Atal and Sureka [26]. The players are presented with events that might occur during the code review workflow of a project. These events are resolved by players' decisions, where the intention is to manage their overall resources (time, budget, and labor) by moving the project forward. These events are implemented as a decision tree that moves players to the nodes with higher reward points when they continue making desirable decisions.

InspectorX is a code review simulation game featuring three types of players. Authors submit software artifacts to the system; inspectors review these artifacts, and moderators manage the assignment of artifacts among players. A ranking system is used to dynamically evaluate players regarding their defect detection capabilities in order to help moderators with their tasks. Additionally, a list of reasons can be provided with the software artifacts for inspectors to mark the defects they have found. The correctness of the review is decided by the moderators [27].

Lastly, Guimarães [28] developed a game flow where players identify intentionally planted mistakes or undesirable practices in a code snippet. Players then select a reason for the defect from a list provided by the tool. The gameplay supports a collaborative multiplayer mode in which players vote for the existence and reasons for these defects. After the defect list is refined by team discussions, the captain submits the final shared review.

3 | Game Design

This section consists of three subsections. 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.

3.1 | Learning Objectives

The overall research goal is to address the barriers mentioned by Indriasari et al. [6]. These barriers are review quality consistency and student learning engagement. The serious game

nature of the proposed platform intends to handle the engagement aspect. Therefore, we construct our learning objectives to specifically address the review quality consistency barrier. We intend to overcome this by increasing code review competency, both in process knowledge and practical skills required to carry out the process effectively.

Therefore, CRSG 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 (used interchangeably with students and participants in this work) to train the skills that 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 and components around these smaller objectives following the gamified design principles developed in [29].

The first objective consists of code review roles, duties, and workflow while the second objective consists of finding code errors, learning to classify these defects to better communicate them, and practicing reviewing code. After this initial design, we manually 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. Several features in the final version of the game originated from this developmental phase. To improve engagement in defect detection exercises, we incorporated elements such as "submission feedback" and "answer explanations." These additions, which are elaborated on later in this section, significantly enhance the game's interactivity and educational value.

This process is represented by Steps 2 and 3 in the overall workflow diagram in Figure 2. The mapping between the base objectives and game components can be seen in Figure 3, 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. The author mode feature was added after the initial design was completed to simulate both roles involved in code review. Because it is a standalone game mode, it also inherits most of the features of the reviewer mode.

3.2 | Prototype, Preliminary Experiment, and Feedback

After defining our learning objectives, we constructed a prototype and prepared quizzes and surveys for evaluation. Furthermore, we carried out a preliminary experiment with seven senior or graduate-level students to gather feedback and address the shortcomings of both the prototype and the initial evaluation experiment. The details of this process are shared in [9]. 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 postquiz and a postsurvey.

The main goal of this preliminary experiment was to gather feedback. We also conducted follow-up interviews where we

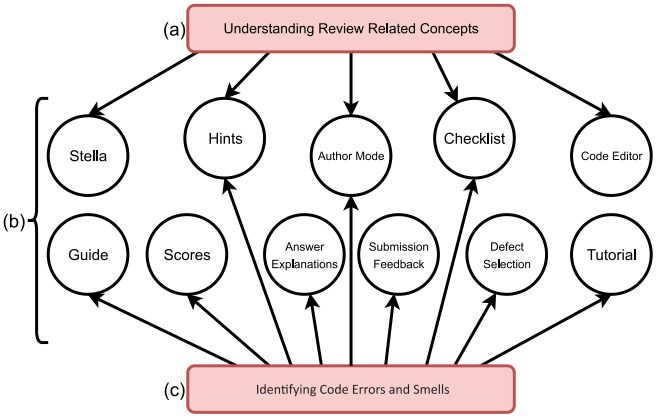

FIGURE 3 | The mapping between game features and learning objectives.

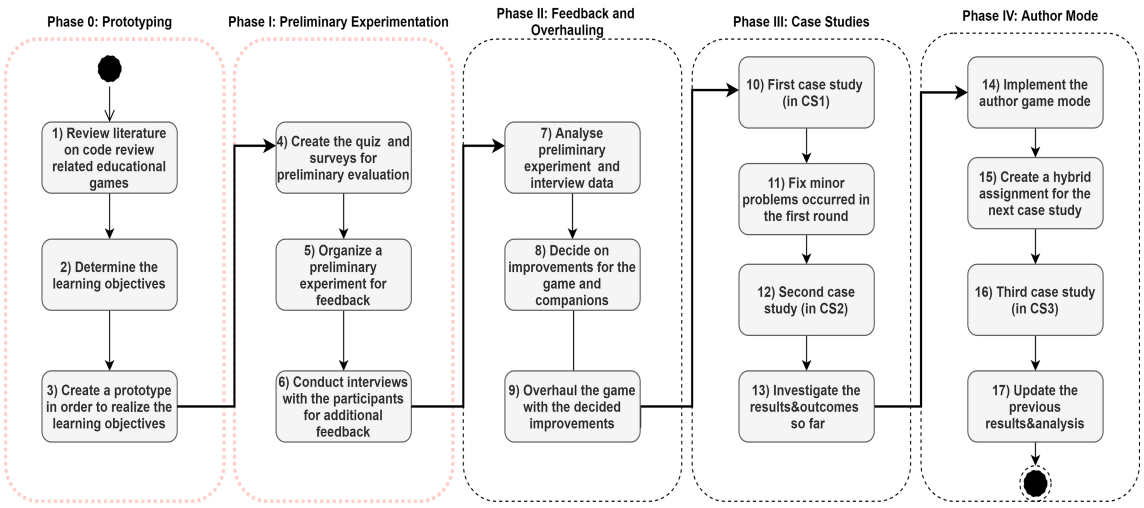

FIGURE 2 | Workflow of the overall study. *Note:* This study aims to report on steps [9–17], but for completeness, we reiterate earlier steps from the previous studies.

asked participants to evaluate the quiz on a per-question basis. The experiment took about 3 h from start to finish, because we wanted the participants to raise their opinions as they arose while we took notes. The follow-up interviews took about 30 min 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, 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. By doing so, we improved the quality of the whole experiment and reduced the experiment duration to a more feasible interval which was necessary for larger scale case studies. These steps are represented with Steps 3–7 in Figure 2. Before starting the large-scale case studies, we also overhauled the game by improving its theme and added several features that were desired by the participants of the preliminary experiment. These additions are indicated by Steps 8 and 9 in the overall workflow (Figure 2). Furthermore, the content related to Steps 10–17 covers our case studies, evaluations, and adjustments in between.

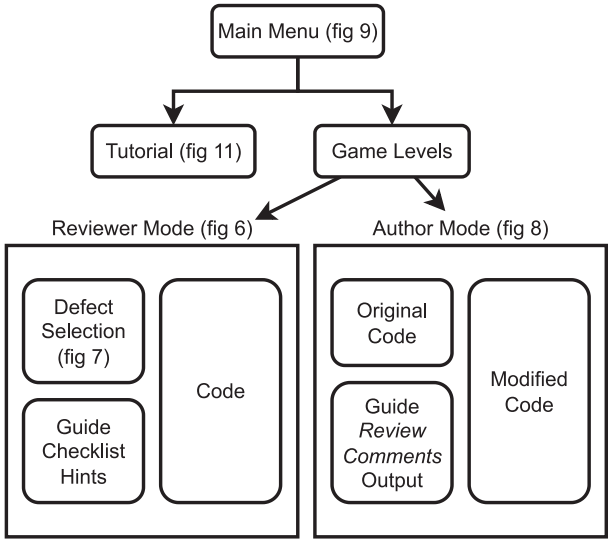

FIGURE 4 | Layout of game components.

3.3 | Game Components and Flow

This subsection starts with a short overall description of CRSG and the rest of it presents the details of each component and game flow. To provide a broader view, we depict the overall layout of the game components in Figure 4.

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 2 had four reviewer mode levels. The tutorial consists of various short code snippets, each of which demonstrates a defect type. The initial level in the game is a practice level where the aim is to demonstrate how the game interface works.

The game's actual levels feature lengthier code snippets with various manually introduced defects. In the reviewer mode levels, players aim to identify all defects and their classifications to fully complete a level. However, moving on to the next level doesn't necessitate complete success. In the author mode levels, players must effectively address comments provided by the reviewer. Although players can access answers for a level, once viewed, they cannot resubmit answers for that level. If players find themselves stuck during a level, hints and other resources we have prepared can be used for assistance. The typical flow for each game mode is illustrated in Figure 5. An older version of CRSG's design choices and key components have been detailed in [10].

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 it

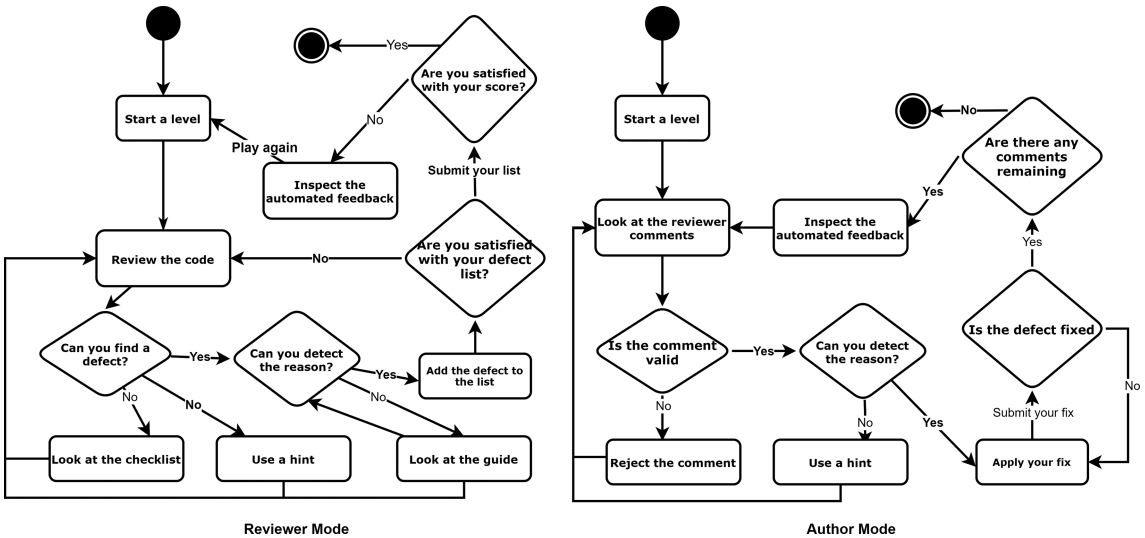

FIGURE 5 | Typical level flow for both game modes.

while also determining the reason for the defect by utilizing our pre-existing taxonomy adapted from [30]. Repeating this process, the players create a list of defects that they found during the code review. Once satisfied with their defect list, they can submit it for evaluation. Our scoring algorithm checks each defect in the submission against our answer key. A defect consists of its starting line, reason, and ending line. As one can see in Algorithm 1, our scoring method awards the players a 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 in 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 in our scale, 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, which means 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. The state of the game with the progress feedback after a defect list submission can be seen in Figure 6.

The right-hand side shows the defective code, and the left-hand side is divided between the defect list and the support menu. The left-hand side of Figure 5 demonstrates how a reviewer level is typically played.

Algorithm 1 Reviewer Mode Scoring

```
score ⇐ 0
for each sub in submittedList do
    for each defect in groundTruth do
        if sub.start == defect.start AND sub.end == defect.end then
            score ⇐ score + 1
            if sub.reason == defect.reason then
                score ⇐ score + 2
            end if
        end if
    end for
end for
```

3.3.1.1 | 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 [30]. We further trim and adapt it to be used in CRSG. 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.

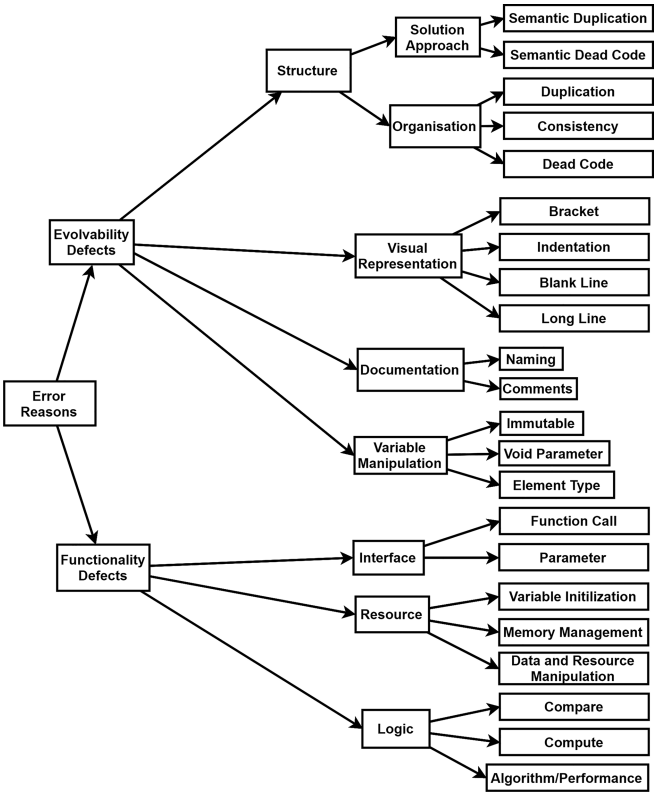

FIGURE 7 | The defect taxonomy used in CRSG.

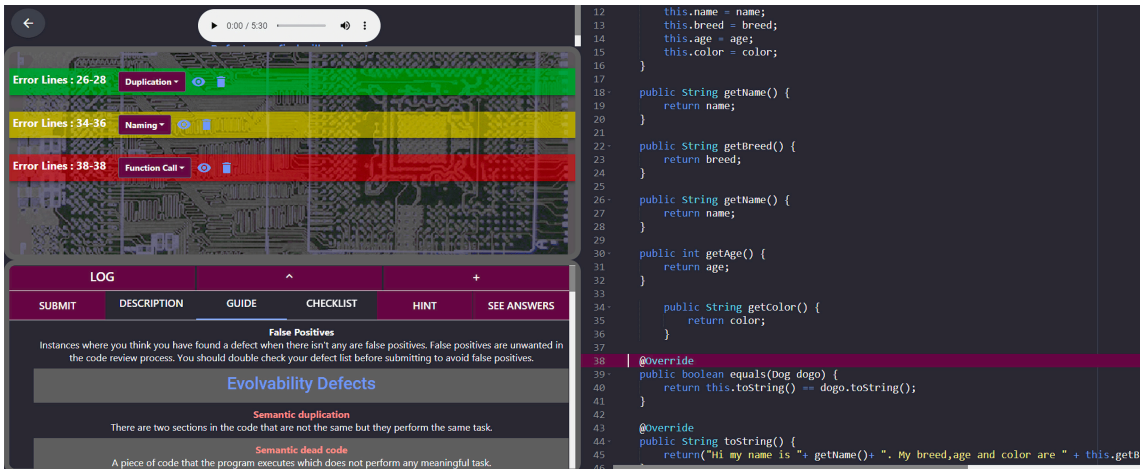

FIGURE 6 | A reviewer mode level.

Figure 7 represents the overall taxonomy that we use in the current version of CRSG.

3.3.1.2 | Auxiliary Game Components. The reviewer game mode has small built-in components that help the player in case they are stuck with a level. These are not as detailed as the tutorial, but they are a good way to help the player without leaving the review screen. These helpers are arranged as tabs located in the bottom left part of the screen as can be seen in Figure 6.

Guide offers a comprehensive view of the defect taxonomy, akin to Figure 7. It is color coded for players' convenience and presents concise definitions for each defect type.

Checklist fulfills the same purpose as a code review checklist that is used in some code review workflows. Checklists are shown to help beginners with the review process [31]. In our context, the list provides players pointers regarding where to look in the code snippet. The items are ordered in a specific way to help players start with easier-to-detect defects like style, variable naming, and magic numbers proceed with more contextual defects.

Hints are another helpful way to keep players from getting stuck on a particular level. They are written by hand and differ between levels. Each level has at least three hints ranging from "the number of defects in a level" to a vague statement meant to nudge the player in the right direction regarding hunting a particular defect.

See Answers is pressed, the level ends officially, and the players cannot change their defect lists anymore. The correct defects are displayed to the player on the left-hand side, and the defective lines are highlighted. We also switch the "description" tab with explanations of the defects so the player can read the solutions before proceeding to the next level.

3.3.2 | Author Mode

This study aimed to obtain an accurate representation of the code review process. The previous 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; therefore, we introduced a new game mode that approaches the process from the code change author's perspective.

Every level of this author game mode begins with a previously reviewed code segment, complete with comments and context from the initial review iteration. Using the information given, players examine both the code and reviewer comments, determining the validity of the remarks. We have provided a straightforward mechanism to dismiss 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 8, 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.

Using the reviewer comments, players try to apply their fixes to the source code for the reviewer comments that 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 CRSG. 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 that the problem persists.

At any point during the level, players can also utilize hints that are specific to each comment and are created by us similarly to the reviewer mode. This approach ensures that players can smoothly apply the review comments without getting stuck easily. Differences in the flow of the two game modes can be observed by comparing the respective parts of Figure 5.

3.3.3 | UI Components

The UI of the game is made to look like the player is controlling a character named Stella through space. We utilized this character to convey code review–related advice. The advice is a

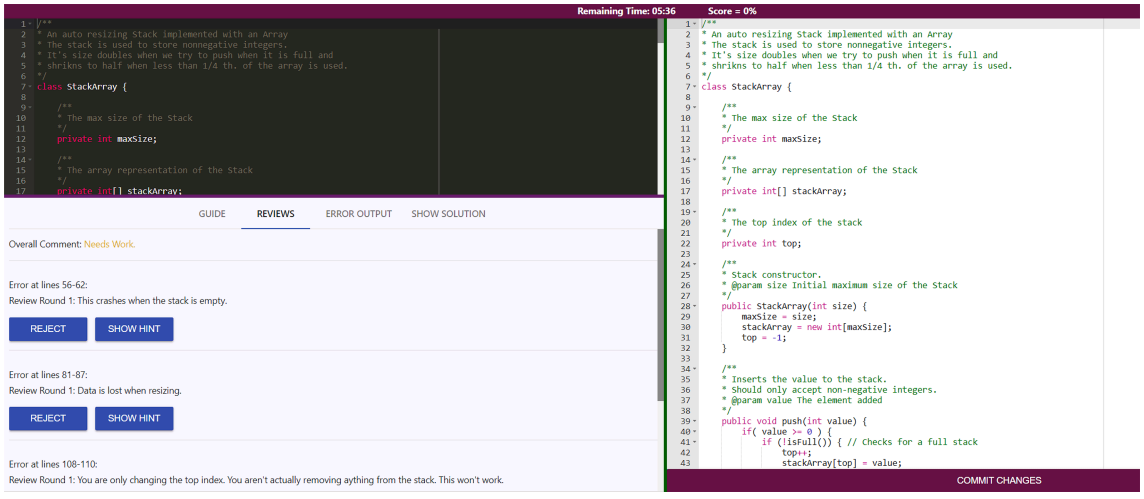

FIGURE 8 | An author mode level.

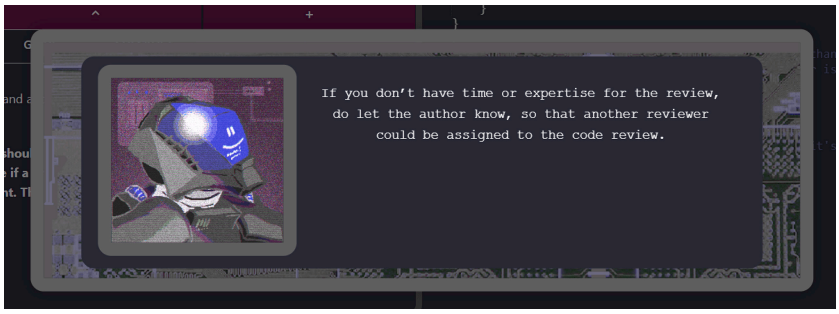

FIGURE 9 | An interaction with Stella.

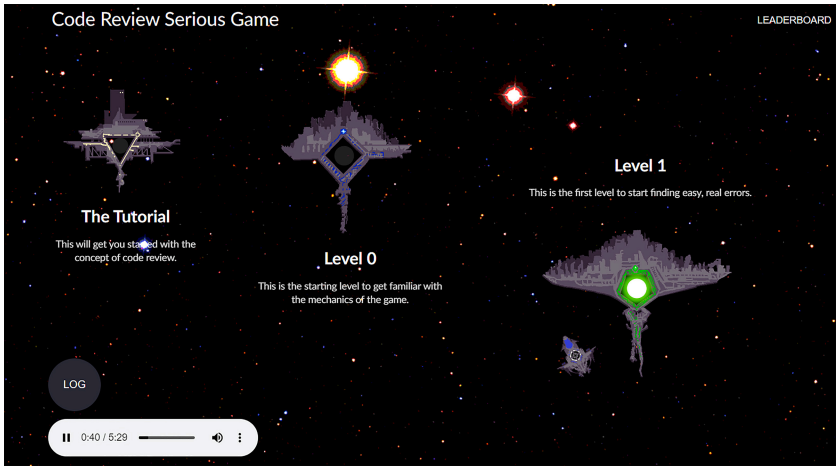

FIGURE 10 | Main menu screen.

compilation of our experiences and code review guidelines of Thoughtbot [32]. An interaction with Stella from the game is shown in Figure 9.

Each level in the game is represented by a space station including the practice level and the tutorial. This is done to comply with the overall theme of the game, where Stella helps these stations by reviewing their code. The individual space stations reflect the increases in difficulty by their representations. Each successive level is a shape with a larger number of vertices. For example, we start with three vertices for the tutorial and proceed with four vertices for the practice level as can be seen in Figure 10. The visual assets for the game were prepared using Adobe Photoshop software.

3.3.4 | Tutorial

The tutorial intends to familiarize the players via defect definitions and examples with the defective and nondefective versions of small code segments. In its current version, there are 10 panels, each with a different minicode segment planted with a single defect. The player can interact with the tutorial to discover the classification and the reason for each defect alongside definitions for the defect category. The "Show the Defect" button on the screen allows them to see a nondefected version of the same snippet while explaining how to avoid or fix the defect. One of the tutorial panels is shown in Figure 11.

4 | Research Design

In this section, we describe the research design format for evaluating CRSG using our main research questions on effectiveness and feasibility. Our methodology blends elements of case study and action research, as discussed by Staron [33]. The integration of action research is the result of the researchers' dual roles as course staff. Consequently, since its inception, the platform has been intended for our students. Therefore, the overall research design is not exactly a case study; however, because we use the guidelines from Runeson and Höst [34] to carry out and report on our research, we prefer to use the term case study for our lab sessions with students. The rest of this section focuses on the details of our research design by reporting on objectives, case study setting, ethics, data sources, and data analysis hypotheses. Subsequently, we also addressed issues related to validity to ensure adherence to these guidelines.

4.1 | Case Study Setting

This section details the setting and structure of the case studies conducted to evaluate the effectiveness of the CRSG. These sessions were adapted for remote delivery due to the COVID-19 pandemic and integrated into specific software engineering courses. The following subsections describe the integration with courses, adjustments made for remote delivery, participant details, and the general flow of each case study.

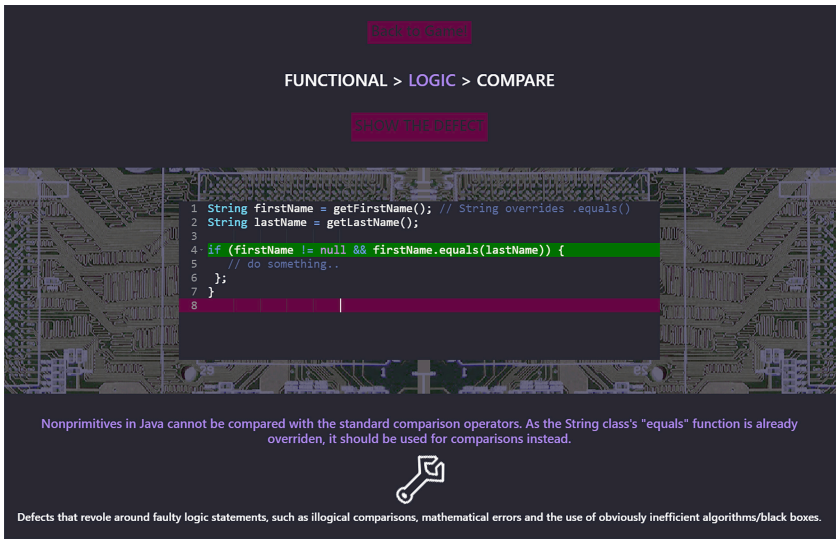

FIGURE 11 | A panel from the tutorial.

4.1.1 | Adjustment to Remote Delivery

The original intention of case studies was to conduct laboratory sessions in person. Due to the university's transition to online education due to COVID-19 pandemic regulations, we performed these case studies with video conferencing. There are three case studies, which are represented by Steps 10, 12, and 16 in Figure 2.

4.1.2 | Case Studies and Course Integration

From here on, we mention each case study by chronologically numbering them as CS1, CS2, and CS3. CS1 was done in CS319 [35] which is a must-course taken by third-year students. CS2 was done in CS453 [36] which is an elective course generally taken by fourth-year students. Similarly, CS3 was done in CS319 of the next year.

- *CS319 Object-Oriented Software Engineering* is a must course for third-year bachelor students. Students are required to have achieved passing marks in fundamental courses on programming and data structures. Key objectives include learning the basics of the software engineering process lifecycle, understanding the object-oriented approach through principles and design patterns, learning UML, and developing practical skills in visual modeling. Additionally, the course emphasizes the development of teamwork and communication skills, particularly through a group project that allows students to practice the application of object-oriented software development principles.

- *CS453 Application Lifecycle Management (ALM).* This fourth-year elective requires passing marks from CS319. The course offers an in-depth study of ALM in the context of large-scale IT software development. It explores the entire ALM process, including agile software development, project management, requirements management, architecture and design, software test management, change management, and more. The curriculum is designed to provide a comprehensive understanding of how these components interact in real-world scenarios, preparing students to effectively manage and optimize the software development lifecycle in professional environments.

4.1.3 | Participation and Ethics

In all case studies, the CRSG was integrated as a mandatory code review lab session within the course. Participation was required for all students, but their performance in these sessions was not graded to ensure a stress-free environment. An attendance-based grade, similar to an attendance quiz, was assigned. Exemptions were provided for students who were unable to participate in the lab sessions.

The total number of participants in all case studies was 280, although not all completed the study. Out of 276 complete submissions, the distribution was as follows: CS1 had 52 participants, CS2 had 80, and CS3 had 144. Table 1 provides detailed information on each case study.

The study received approval from our institution's ethics board and informed consent was obtained from all students. Personally identifiable information was used solely to match pregameplay and postgameplay materials, and all data were permanently anonymized before analysis.

4.1.4 | Flow and Structure of a Case Study

We have compiled various auxiliary materials aiming for both numerical evaluations and players' personal evaluations regarding their experience with the CRSG.

The flow of a case study can be seen in Figure 12, which starts with a presurvey that was sent out a day before. In all case studies, the scheduled meeting started with a short verbal tutorial

TABLE 1 | Case study details.

Case Study No.	Class	Participants	Reviewer levels	Author level
CS0 (preliminary)	—	7	4	—
CS1	CS319	52	4	—
CS2	CS453	80	4	—
CS3	CS319	144	2	2

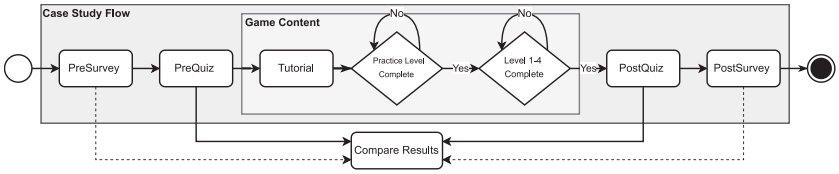

FIGURE 12 | The flow of the case studies.

and was followed by the prequiz component. After that, the players proceeded to play the tutorial and the practice level. At any point, they could ask us questions using the private chat function of the video conferencing application Zoom.[4] The study then continued with the actual gameplay of Levels 1–4. The players who have completed the main gameplay levels completed the postquiz before signing off. We sent out the postsurvey a day after the main session. We closed any access to the game or the material between case studies.

4.2 | Data Collection

In this study, we used three different sources to collect data. The first source consists of our surveys done before and after the virtual lab session. The presurvey aims to determine the CR knowledge and overall experience of the participants before coming into the case study. The postsurvey aims to measure the overall participant impressions and individuals' subjective familiarity and confidence after the case study. They share common Likert scale questions for participants to self-reflect which we then utilize for evaluations. The surveys are available in Appendix A (Tables A1–A3).

The second source consists of the prequizzes and postquizzes. They are identical; therefore, we can directly observe how the participants' answers change after playing the game. This "pre–post testing" is the most popular method of evaluation in digital games for engineering education [37].

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 from the participants of the preliminary experiment interviews (Step 6 in Figure 2). These are accompanied by a few questions that we added after the preliminary experiment, totaling up to 24 questions [9]. The quizzes were the same for all case studies.

Our third source for data collection are the player logs of the game. Our cloud database logs most of the player activity during gameplay. From these data, we can deduce the time spent 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. These are complemented by player IDs to recognize players and differentiate between participants of different rounds to conduct a more in-depth analysis.

4.3 | Data Analysis

To address our primary research questions:

RQ1. How effective is CRSG for introducing students to code review and its related concepts?

RQ2. How feasible is it to use CRSG to introduce code review concepts within a course curriculum?

We have formulated the following hypotheses to guide our data analysis and detail how each hypothesis is tested, which variables were used, and the source of the data:

4.3.1 | Assessing the Effectiveness of CRSG (RQ1)

1. **Hypothesis 1.1 (H1.1)**: Players' self-reported knowledge of code review will significantly improve after playing CRSG.
   - *Reason*: Improvement in self-reported knowledge indicates effective learning.
   - *Data source*: Presurvey and postsurvey responses.
   - *Variables*: Likert scale data on CR knowledge question (Q1) in Code Review Knowledge Sections in presurvey and postsurvey.
   - *Testing method*: Median and mod measures.
2. **Hypothesis 1.2 (H1.2)**: Players' self-reported confidence in performing code reviews will significantly improve after playing CRSG.
   - *Reason*: Improvement in self-reported confidence indicates development in practical application of code review–related skills.

- *Data source*: Presurvey and postsurvey responses.
- *Variables*: Likert scale data on CR confidence question (Q2) in Code Review Knowledge Sections in presurvey and postsurvey.
- *Testing method*: Median and mod measures.

3. **Hypothesis 1.3 (H1.3)**: Players' perceived importance of code review will significantly increase after playing CRSG.
   - *Reason*: Understanding the importance of code review is important for long-term retention.
   - *Data source*: Presurvey and postsurvey responses.
   - *Variables*: Likert scale data on CR confidence question (Q3) in Code Review Knowledge Sections in presurvey and postsurvey.
   - *Testing method*: Median and mod measures.
4. **Hypothesis 1.4 (H1.4)**: Players' familiarity with code review concepts will significantly increase after playing CRSG.
   - *Reason*: Familiarity with code review–related concepts is a requirement for a mature code review process.
   - *Data source*: Presurvey and postsurvey responses.
   - *Variables*: Likert scale data on CR concepts question (Q4) in Code Review Knowledge Sections in presurvey and postsurvey.
   - *Testing method*: Median and mod measures.
5. **Hypothesis 2 (H2)**: Players' scores on a comprehensive code review quiz will significantly improve after playing CRSG.
   - *Reason*: Objective performance metrics provide evidence of learning.
   - *Data source*: Prequiz and postquiz scores.
   - *Variables*: Average number of correct answers on quiz questions.
   - *Testing method*: Paired-sample t test to compare prequiz and postquiz scores.

4.3.2 | Feasibility of CRSG (RQ2)

6. **Hypothesis 3 (H3)**: Players will maintain a consistent level of engagement throughout gameplay.
   - *Reason*: To demonstrate the usability of CRSG.
   - *Data source*: Time metrics from gameplay.
   - *Variables for analysis*: Time spent on each level.
   - *Testing method*: Similarity of time spent per defect for each level.
7. **Hypothesis 4 (H4)**: The auxiliary components of the game will be positively received by players.
   - *Reason*: To better understand the player experience.
   - *Data source*: Postsurvey component evaluations.
   - *Variables for analysis*: Players' ratings of game components in Likert scale.
   - *Testing method*: Average and standard deviation statistics for the usefulness of each auxiliary game component.

The results section will follow this subsection, presenting findings related to the efficacy and feasibility of CRSG as an educational tool. Contrary to this section, the results are organized by data sources instead of research questions.

5 | Case Study Evaluation

5.1 | Survey Results

To provide a better understanding of the demographics of participants, we asked questions regarding their industry and Java language experience in the presurvey. Most of the industry experience consists of summer internships and part-time jobs. 46% of the participants have 0–3 months, while 18% have 3–6 months and 12% have more than 6 months of experience. Ninety-four percent 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 (Section 4.3, H1). These questions were asked in both surveys to evaluate the changes in the answers. A detailed view of the answers and differences can be seen in Figure 13. Each bar in the figure represents the percentage of participants that chose the respective option from a 1–5 Likert scale. For the majority of the questions, we can see the shift to more positive choices on the Likert scale from presurvey to postsurvey.

The findings presented in this figure are summarized in Table 2. Here, we observe a consistent increase in the median of responses toward more positive options across most questions, with the notable exception of the question about the importance of code review. A more elaborate breakdown of this data and the original surveys are available in Appendix A.

Upon analyzing both presurvey and postsurvey results, the most striking change noted pertains to participants' confidence in their code review knowledge and skills. Initially, in the presurvey, 46% of the participants predominantly selected the most negative option for this question. However, in the postsurvey, there was a notable shift with the majority now choosing the neutral, middle option. This change underscores a significant improvement in participants' self-perceived proficiency and confidence in code reviews.

5.2 | Quiz Results

The prequizzes and postquizzes consisted of the same 24 questions. A list of the quiz questions can be found in Appendix B (Tables B1–B2). The average scores on the prequiz and postquiz are 61 and 70, respectively. The overall score improvement between quiz installations is around 10 points out of 100. However, because the number of suboptions varies among quiz questions, affecting the points a question is worth, we use only the quiz scores as an indicator. Thus, the analysis of the quiz is based on the difference in the number of correct options between prequizzes and postquizzes (Section 4.3, H2).

To condense the quiz results, we have created three question groups, which are "CR Knowledge," "Defect Taxonomy," and "Programming."

The "Defect Taxonomy" category consists of Questions 6–8. Here, we ask the participants to differentiate between functional

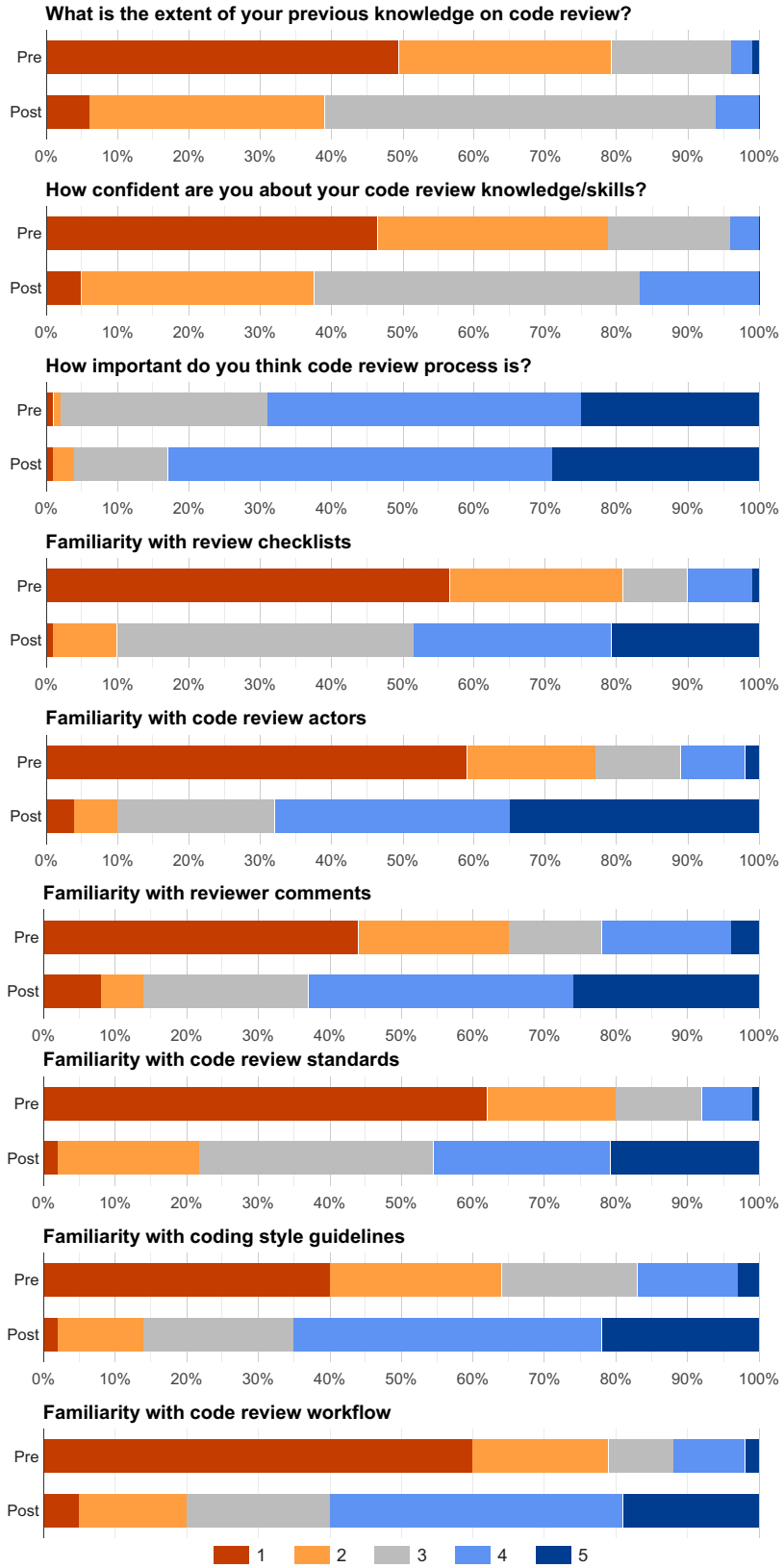

FIGURE 13 | Results of the overlapping questions in the presurveys and postsurveys from worst (1) to best (5) option.

and nonfunctional defect types (Question 6), organization-related structural defects (Question 7), and logic-related functional defects (Question 8).

Similarly, questions between 14 and 23 besides Question 20 are grouped under "Programming" where we ask about the behaviors and defects of small code segments.

TABLE 2 | Summary of code review knowledge sections in surveys.

Question (Table A3)	Premedian	Mod	Postmedian	Mod
CR Knowledge (Q1)	2	1	3	3
CR Confidence (Q2)	2	1	3	3
CR Importance (Q3)	4	4	4	4
CR Concepts (Q4)				
Review checklists	1	1	3	3
Code review actors	1	1	4	5
Reviewer comments	2	1	4	4
Code review standards	1	1	3	3
Coding style guidelines	2	1	4	4
Code review workflow	1	1	4	4

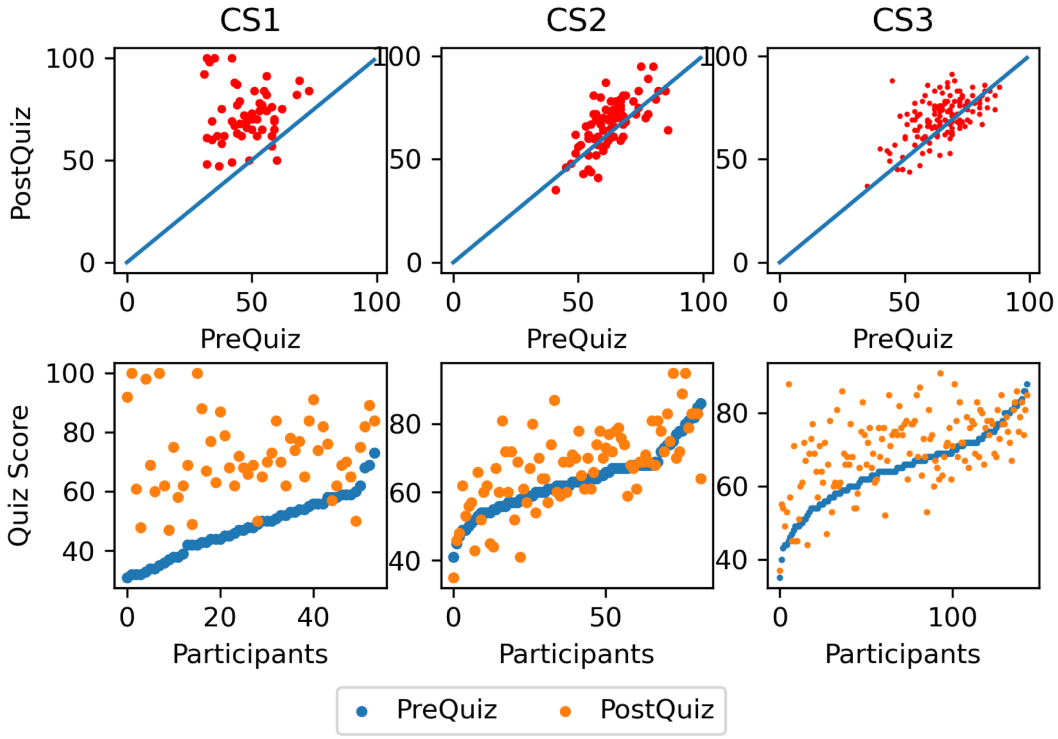

FIGURE 14 | Changes in quiz scores.

The remaining questions are grouped as "CR Knowledge" because they are about the overall knowledge of the code review process. The quiz questions themselves are available in Appendix B, and the original quizzes including the code segments can be accessed from our online resource package.[5]

For each question group, we calculated the average number of correct answers in the prequiz and postquiz. The improvement is the difference between these averages.

Figure 14 demonstrates the distribution of the quiz scores in each round. Each column in the figure corresponds to a case study. The top row compares the prequizzes and postquizzes. Each point belongs to a participant, and the 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 14 are obtained by sorting the participants regarding their prequiz scores in ascending order where each vertical line drawn 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 postquiz and prequiz scores of a participant, respectively. Compared to other case studies, for example, we can see that CS1 shows a clearer pattern of improvement and the lowest average postquiz score.

A paired-sample t test was performed to evaluate the significance of the quiz results by comparing the success of the quiz before and after playing the game. Because CS3 includes a different game mode, we perform this test between CS1 and CS2. First, to confirm that a paired-sample t test is applicable, we performed the Shapiro–Wilk test [38] to verify that the data for each quiz in each case study are normally distributed. The p values for this test are [0. 247,0. 299,0. 344,0.279] for each case study and prequizzes and postquizzes, respectively. As the p values are greater than $\alpha = 0.05$, the quiz results are normally distributed.

Then, we performed the paired-sample t test for case studies followed by Bonferroni correction [39] with $\alpha = 0.05$ to avoid false positives. We find that the corrected p values for CS1 and CS2 are $[9.350 \times 10^{-14}, 1.239 \times 10^{-5}]$. Therefore, we can reject the null hypothesis and conclude that our results between prequizzes and postquizzes are statistically significant.

5.3 | Player Score Analysis

We used a cloud database to record 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 min on average. More detailed information on scores and time spent per level is available in Table 3. We see that the time spent on a singular defect is similar between all levels, indicating that there were no major drops of attention during the case study period (Section 4.3, H3). 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 15. The linearity of the plots in the figure summarizes how consistently a player's effort is converted into points. The data points of this section came from CS1 and CS2, which were later used to help determine the maximum number of submissions in CS3.

5.4 | Player Satisfaction and Feedback

5.4.1 | Component Analysis

Component analysis data are gathered from each player in the postsurvey in the 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 (Section 4.3, H4). The distributions of the responses of the participants and the averages for each component can be seen in Figure 16. According to the participant responses, the most useful component is "answer explanations," while the least useful component is "Stella." The standard deviations for all components are around 1, meaning that most 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 postsurvey 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 16 include all case studies except the author mode because it was developed between CS2 and CS3 as can be traced in Phase IV of Figure 2.

TABLE 3 | Player averages per level in CS1 and CS2.

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

5.4.2 | Free-Text Answers

The postsurvey included two free-text questions. In the first question, we asked participants to write down three things they had learned during the case study. We compiled their answers and performed open coding [40], where we organized each item from each participant into categories according to the concept the item was about. The final concepts we came 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 17. Note that each participant accounted for about three items in the figure.

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

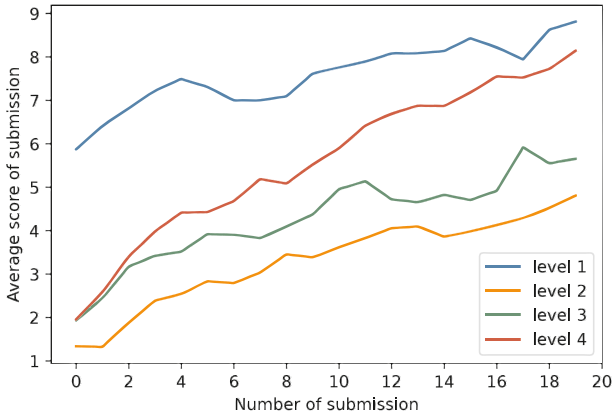

FIGURE 15 | Average player scores per submission in reviewer mode.

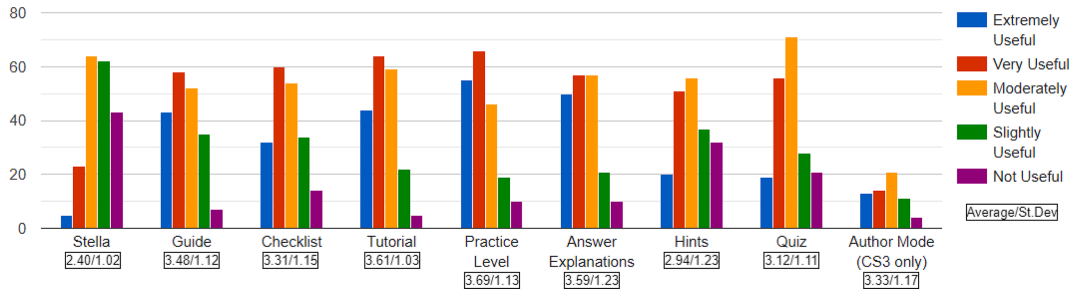

FIGURE 16 | Evaluation of game components by participants.

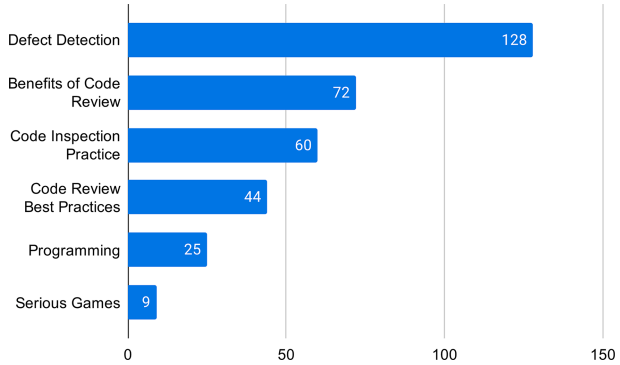

FIGURE 17 | Open coding concepts extracted from the first free-text question.

> 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 is 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 postquiz for that matter, might be a good idea.

6 | Discussion

6.1 | RQ1: How Effective is CRSG 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 that they increased their code review knowledge and are more confident about conducting reviews. The shift to more positive options on the Likert scale can be examined in Figure 13 while the medians of these options for all case studies can be observed in Table 2. We were able to observe a drastic decline in the lowest option on the Likert scale besides the question on the importance of code review (H1.3). This question was different from the rest, as the presurvey answers were already positive. There is some noise with the second lowest option as we saw a very small bump from one person to three people.

For the rest of the questions, we saw an improvement in code review knowledge (Section 4.3, H1.1). The most prevalent option changed from "1" to "3" on the Likert scale, but there is no real increase in options "4" and "5." This result is appropriate for the scope of CRSG as it is an introductory activity. A large increase in option "5" could have meant that the students underestimated the expertise required to conduct code reviews effectively. Similar statements can be made about CR confidence (H1.2) questions because the answer patterns are extremely similar. It is plausible that self-reported confidence and knowledge are similar.

In the presurvey, answers to all of the six different code review concept familiarity (H1.4) questions are mostly "1" or "2." Asking about a relatively intuitive concept like the actors of code review might seem redundant at first; however, it turned out to be a worthwhile question given the students' apparent confusion in the presurvey answers. After engaging with CRSG, reported familiarity with all of the concepts increased by exposure to the code review process. The least improved concepts are "review checklists" and "code review standards." The CR checklist in CRSG does not truly reflect an actual list that could be used by a reviewer. Our list was designed to help with the game levels as there is no universal code review checklist; however, students still have a rough idea of what an actual checklist will resemble. Teaching code review standards is not the main objective of CRSG, but we were still able to convey some information by changing the most prevalent answer from "1" to "3."

In general, we found the survey data to be satisfactory regarding self-evaluations of players. The increase in the confidence of students in Table 2 is indicative of the effectiveness of CRSG in introducing code review. The quiz-related data again shows that the players improved after playing; however, it is not as straightforward as the survey data. While interpreting this portion, one first needs to realize the differences between rounds. Participants in CS1 and CS3 are much more likely to have no prior knowledge about code review, as the syllabus of CS453 involves a brief chapter on code review. We observed the effects of these fundamental differences in rounds in the

prequiz scores. The average of CS1 regarding the prequiz is around 50 while CS2 averaged around 65. However, the postquiz averages are much more similar.

There were a couple of variables that collectively resulted in CS1 showing more improvement. Firstly, a significant part of the points collected from the quiz is not directly related to the game contents because the quiz was designed to be a standalone evaluation. 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 case study 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 the data we collected indicate, the timing of our introductory activity seems to affect the results. Due to scheduling and course load factors, it is not always possible to perform this activity in its ideal setting, which is an in-person, in-class activity.

Improvements in quiz results are achievable when the ideal setting is provided. We chose not to compare CS3 with prior versions because our author mode component did not utilize the defect taxonomy, reducing player exposure to it. Considering that this was the portion of the quiz with the greatest improvement in the previous cases, the comparison would not be fair. We think of this as a trade-off for future users. They can choose to use the author mode and have a complete simulation of the code review process, or they can use the original version to give players more time with the defect taxonomy.

Moreover, our students favored the interactive session over a regular class. In the CS3 postsurvey, 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?

Seventy-seven percent of the respondents to this question agreed or strongly agreed with the above question on a 1–5 Likert scale. Students favoring these types of activities is enough of a reason to pursue them. Moreover, the general results indicate that the activity is more beneficial as an introduction to CR, which supports our claims.

Taking into account the results of case studies and the factors mentioned above, we deduce that CRSG has merit in introducing code review. The extent of learning is not 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.

6.2 | RQ2: How Feasible It Is to Use CRSG to Introduce Code Review Concepts Within a Course Curriculum?

We have already addressed the benefits of CRSG in the previous research question. The ease of application and time-wise feasibility remain to be discussed. So far, we have presented CRSG as the main component of case studies in a setting that we use to evaluate the outcomes of gameplay sessions. 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 h. In all rounds, most of the students completed the session (from the prequiz to the postquiz in Figure 12) around 2 h. We deduced that the gameplay (the game content portion of Figure 12) takes 45 min to an hour for the majority, while the rest of the time was spent on the other components.

To perform the activity in a classroom setting, the quiz and survey components of our case studies 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 that the students were largely autonomous, which means that they had little to no confusion or technical problems during the game. This was also considered to be a factor that increases the applicability of CRSG 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 that the time spent per participant interacting with the game content (demonstrated in Figure 12) is mostly consistent between levels. This indicates that the activity we presented to the students was engaging.

6.3 | Defect Detection Skills

On its own, theoretical code review knowledge is not enough for the development lifecycle to benefit from code review because reviewer experience is a contributing factor for a successful review process [41]. Practical benefits, in addition to educational purposes, are related to the reviewer's ability to detect defects or other unwanted attributes of the code in review.

We have broken down the process of detecting defects into two stages consisting of detecting the defect and determining the reason. The reason behind the defect was important to us because we 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

TABLE 4 | Quiz summary.

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

communication process between the author and the reviewer, we integrate the defect taxonomy into our game. The quiz had direct questions on taxonomy and some programming questions that addressed both detection and reasoning, the results can be examined in Table 4.

The participants seem to do well on 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.

6.4 | Content Creation Challenges

Creating additional levels for CRSG proved to be a more difficult task than expected. Here, we elaborate on the aspects that make level creation challenging and the reasons behind them.

Game levels use code snippets that are equivalent to an actual source code file at best. This limited context is intentional to keep the overall complexity of the process low. We want players to focus on experiencing the code review process without worrying about understanding a complex piece of code. Therefore, the levels we designed are simple and the purpose of the code is very apparent even at a single glance. This intentional simplicity has a side effect. It is harder to represent a realistic code review scenario using a simple snippet. In real-world applications of code review, the scope of the code in review is larger. Because our players do not see an actual repository, each level has to be self-contained. This aspect is good for overall gameplay but in turn makes designing levels harder.

Moreover, defects that are planted into a level cannot produce compile-time errors. In a realistic code review scenario, a piece of code that is sent for review is complete [2] and a compile-time error would indicate the opposite. Therefore, we refrain from using defects that would introduce compile-time errors as well as very apparent behavioral changes. Our attempts to keep this aspect of level design is another limiting factor.

Having these limiting factors causes the content to move toward simplistic and static errors, for instance, bad comments or indentation-related defects. These types of error occur in real-world applications of code review [30], but it is also possible to detect them with static analysis tools. Although they are not the ideal type of defects to demonstrate the unique benefits of code review, we have decided to use them.

Trying to avoid simplistic defects can also result in a level consisting of complex defects, which makes the challenges too difficult or detail oriented. More demanding content might be desirable for some settings, but CRSG does not intend to test programming skill. To be able to focus on introducing the code review process, we attempted to create a nice balance using the defects that made sense for a realistic code review setting.

6.5 | Comparing CRSG to Related Work

There are a variety of approaches to delivering an introduction to code review. The dominant method is to use a traditional lecture or capstone project. In this study, we explore the extent of what CRSG is capable of and whether it can replace traditional lectures on the topic as an autonomous activity. As we demonstrated in Section 2, there are other studies that create serious games and other interactive material. The four studies that we were able to find in serious games for code review were Pex4Fun [25], Anukarna [26], InspectorX [27], and Guimarães [28]. Table 5 summarizes these studies.

Among CRSG and the four studies we identified, Anukarna [26] is the only one that does not simulate the review process with code editing. Instead, it focuses on the bigger picture decisions and resource management. These concepts could be integrated with the other games to increase the scope. This type of decision-based resource management technique might be a more explicit way to convey information on code review concept familiarity questions in Figure 13. The rest of the studies had more similar objectives and execution to CRSG. InspectorX [27] provided a game flow in which players provide the content to be reviewed. This is an alternative approach to the content that we provide ourselves. Creating content for CRSG came with a fair amount of challenges mainly because we wanted to have realistic code review scenarios. It is unlikely that exercises provided by the student could preserve this design aspect. Choosing between player-provided content and curated content is a trade-off

TABLE 5 | Comparison of code review–related serious games.

Name	Focus	Game elements
Pex4Fun [25]	Coding duels covering debugging, testing, and inspection	Points, player rankings, and feedback during gameplay
Anukarna [26]	Decision-making during code review	Scoring based on technical debt
InspectorX [27]	Player-provided code snippet inspection for error detection	Player rankings, multiple player roles, and points
Guimarães [28]	Code snippet inspection for error detection	Badges, points, and player collaboration via voting
CRSG	Code snippet inspection for error detection and application of reviewer comments	Points, leaderboards, feedback during gameplay, multiple game modes, and story elements

between realism and repeatability. The replay value could be preferable in situations where a game is not utilized as a lab session in a software engineering course.

The serious game created by Guimarães [28] has the closest design to CRSG; however, 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. Previous studies on the topic lacked detailed evaluations of their methods. Furthermore, the process of applying review comments is not addressed. We believe this to be an important element because the actual benefits of code review are obtained from successfully applied review comments.

The main advantages of CRSG over the tools from previous studies are mostly from a more in-depth view of the code review process because it includes dynamic feedback and guides for gameplay while also allowing players to experience author and reviewer roles separately.

7 | Threats to Validity

7.1 | Internal Validity

To decrease the uncontrollable variables and to expand our control over the case studies, we intended to perform laboratory sessions in person. By doing so, we would make sure that there would not be any communication between the participants or that they would not make use of online resources. Because we were unable to perform the case studies in our preferred setting due to the COVID-19 pandemic, it was performed in a video conferencing format using Zoom [42] which was the preferred application by the university administration. Due to this change, we have taken some measures to protect the evaluation results. We prepared in-game resources so that 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 prompted to the players after the level screen was back in focus.

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 that their numbers are small enough (two 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. Furthermore, because we created the game content and its auxiliary components by ourselves, human error could always affect the process negatively. We tried to negate this effect with feedback from the preliminary experiment and interviews with its participants. This process is denoted in Steps 5–9 in Figure 2.

There is another threat to internal validity that stems from the use of different versions of CRSG in our case studies. The experiences of conducting CS1 and CS2 highlighted opportunities to improve our platform. As a result, we developed the author mode and replaced less effective content in the reviewer mode, culminating in a more comprehensive version of CRSG. Consequently, CS3 shares only about half of its gameplay content with its predecessors. This variation means that the CS3 participants were engaged in an updated version of the game, which could potentially influence our findings and conclusions.

To address this concern, we have strategically excluded CS3 from our more quantitatively focused evaluation methods. This includes the paired-sample t test used for quiz analysis and player score analysis, as half of the content in CS3 is evaluated differently. By doing so, we aim to maintain the integrity of our results and ensure a more accurate assessment of the impact of CRSG.

7.2 | External Validity

The rest of this section investigates the external validity of our results, which is defined as the generalizability of the sample results to the population of interest, across different measures, persons, settings, or times [43]. The target audience for CRSG 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 [35] and CS453 [36], we intend to match the case study audience with the population of interest of CRSG. Using these courses as the audience guarantees a uniform minimum background for participants, because 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 studies, 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.

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

Moreover, we also acknowledge that applying the same test twice could also affect the results gathered from the postquiz because participants interacted with the same questions in the prequiz. 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 think passively and improve their answers to the quiz questions on their own.

Furthermore, one could also say that a certain decrease in interest regarding the questions would have manifested because the participants are taking the same test a second time. This decrease in interest would have led 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 postquiz results by masking some portion of the participant improvement from the analysis or introducing some negative results. We believe that there is a correlation between negative results and interest decline in postquiz data because of the postsurvey results. The postsurvey was sent out the day following the lab session to give students time to reflect and we did not observe the same amount of "noise" in that portion of the data.

7.3 | Construct Validity

The biggest threat to construct validity in this study comes from the comprehensiveness of the code review knowledge quiz. We use quizzes as an alternative to the survey to look for evidence of participant improvement. However, we also wanted to observe the general level of competency of the participants regarding the overall code review process. This does not directly contribute to the evaluation of the game but allows us to better understand the knowledge gaps of the participants. The caveat of the comprehensive quiz is that it involves things that are not directly thought by CRSG. For example, there is a question about the reviewer's responsibility toward design patterns; however, this information is not explicit in-game content.

Consequently, this means that quizzes are not ideal for measuring the outcomes of the game. We mitigate this threat by making the quiz-related measurement an alternative to survey which is the more widely used data source in the literature.

Another threat to construct validity is our efforts in measuring engagement. There is no standard way to quantify engagement that applies to our case studies. However, we were able to measure whether the participants spent a consistent amount of time per defect (on average). This is only an indicator. We similarly mitigated this threat by directly asking the participants if they would prefer this activity over a traditional lecture as discussed in Section 6 and received highly positive feedback from the participants.

8 | Conclusion and Future Directions

Throughout this study, we have proposed, designed, implemented, and evaluated CRSG. Our motivation comes from the lack of representation regarding code review in curricula of computer science, software engineering, etc. We designed the game to directly address our learning objectives and performed a preliminary evaluation in order to gather feedback to be able to improve the game to better meet the requirements of the learning objectives. We then proceeded to the evaluation of the game, where we utilized three case studies with 280 students in total. During each case study, participants filled out pregameplay and postgameplay surveys and quizzes that we used as data for evaluating the game.

We present our data and analysis where the analysis attempts to answer our two research questions that refer to teaching CR practices and the general feasibility of using CRSG as an in-class activity. The results indicate that the game can be used as an introductory activity regarding the code review practice. Furthermore, the structure of CRSG can potentially be extended to demonstrate other software engineering processes (e.g., bug tracking) or other related soft skills regarding programming. 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.

To further document the study and allow educators to use CRSG, 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.

The current design of our code review tool extends its potential application beyond traditional course settings. For instance, integrating this tool or its design with a massive open online course (MOOC) focused on code review could significantly broaden its reach, overcoming limitations like game duration inherent to course-based implementation. Additionally, incorporating our material into an industrial orientation program presents another viable direction. However, given that our current version is primarily geared toward introducing the code review process, it may be oversimplistic for professional environments.

To adapt to these broader applications, a few essential enhancements are necessary. Currently, the process of curating game content is challenging. Using generative artificial intelligence technologies to automate content creation could dramatically increase replayability and appeal of CRSG. This advancement

is seen as a critical step in expanding the game's audience and adapting it for diverse contexts, including industrial training and large-scale online education platforms.

Besides the practical uses, the current design can also present more opportunities for researchers. Most aspects of CRSG are designed from scratch, but the point we managed to reach could allow for a valuable head start for other researchers who are looking to enhance software engineering education even outside of the code review process. With some reconsideration, it is plausible to utilize aspects of CRSG for other processes such as software testing or teaching programming, in general.

Data Availability Statement

The data that support the findings of this study are openly available in CRSG case study data at https://figshare.com/s/567912dbc8e39e41350c.

Endnotes

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

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

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

[4]www.zoom.us.

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

References

1. A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," in *2013 35th International Conference on Software Engineering (ICSE)* (San Francisco, CA, USA: IEEE, 2013), 712–721.

2. C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern Code Review: A Case Study at Google," in *ICSE (SEIP) Edited by* F. Paulisch and J. Bosch, (Gothenburg, Sweden: ACM, 2018), 181–190.

3. J. Klünder, R. Hebig, P. Tell, et al., "Catching Up With Method and Process Practice: An Industry-Informed Baseline for Researchers," in *ICSE (SEIP) Edited by* H. Sharp and M. Whalen, (Montreal, QC, Canada: IEEE/ACM, (2019), 255–264.

4. 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* (Florence, Italy: IEEE, 2015), 3–6.

5. V. Garousi, G. Giray, and E. Tuzun, "Understanding the Knowledge Gaps of Software Engineers: An Empirical Analysis Based on SWEBOK," *ACM Transactions on Computing Education* 20, no. 1 (2019): 1–33, https://doi.org/10.1145/3360497.

6. T. D. Indriasari, A. Luxton-Reilly, and P. Denny, "A Review of Peer Code Review in Higher Education," *ACM Transactions on Computing Education* 20, no. 3 (2020): 1–25, https://doi.org/10.1145/3403935.

7. T. M. Connolly, M. Stansfield, and T. Hainey, "An Application of Games-Based Learning Within Software Engineering," *British Journal of Educational Technology* 38, no. 3 (2007): 416–428, https://doi.org/10.1111/j.1467-8535.2007.00706.x.

8. T. Susi, M. Johannesson, and P. Backlund, "Serious Games: An Overview," (2007).

9. 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)* (Munich, Germany: IEEE, 2020), 1–5.

10. 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: Association for Computing Machinery, 2020), 1561–1565, https://doi.org/10.1145/3368089.3417932.

11. 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: Association for Computing Machinery, 2014), 202–211, https://doi.org/10.1145/2597073.2597082.

12. SMARTBEAR, "The State of Code Review in 2019: Trends, Tools, and Insights for dev Collaboration," (2019), https://smartbear.com/resources/ebooks/the-state-of-code-review-2019/.

13. N. Davila and I. Nunes, "A Systematic Literature Review and Taxonomy of Modern Code Review," *Journal of Systems and Software* 177 (2021): 110951, https://www.sciencedirect.com/science/article/pii/S0164121221000480.

14. L. MacLeod, M. Greiler, M.-A. D. Storey, C. Bird, and J. Czerwonka, "Code Reviewing in the Trenches: Challenges and Best Practices," *IEEE Software* 35, no. 4 (2018): 34–42.

15. 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: Association for Computing Machinery, 2014): 192–201, https://doi.org/10.1145/2597073.2597076.

16. E. Doğan and E. Tüzün, "Towards a Taxonomy of Code Review Smells," *Information and Software Technology* 142 (2022): 106737, https://www.sciencedirect.com/science/article/pii/S0950584921001877.

17. M. R. A. 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* 95 (2018): 201–218, http://www.sciencedirect.com/science/article/pii/S0950584917303518.

18. 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)* (Ottawa, ON, Canada: IEEE, 2005), 25–32.

19. 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)* (Hua Hin, Thailand: IEEE, 2015), 1–5.

20. J. C. Farah, B. Spaenlehauer, M. J. Rodríguez-Triana, S. Ingram, and D. Gillet, "Toward Code Review Notebooks," in *2022 International Conference on Advanced Learning Technologies (ICALT)* (2022), 209–211.

21. T. Haendler, G. Neumann, and F. Smirnov, "An Interactive Tutoring System for Training Software Refactoring," in *11th International Conference on Computer Supported Education*, Vol. 1 (Heraklion, Greece: SciTePress, 2019), 177–188.

22. F. Ricciardi and L. T. D. Paolis, "A Comprehensive Review of Serious Games in Health Professions," *International Journal of Computer Games Technology* 2014 (2014): 9–9.

23. M. Ulicsak and M. Wright, *Games in Education: Serious Games*, (Bristol, UK: Futurelab, 2010), https://www.nfer.ac.uk/games-in-education-serious-games.

24. 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)* (San Francisco, CA, USA: IEEE, 2013), 36–39.

25. 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* (San Francisco, CA, USA: IEEE Press, 2013), 1117–1126.

26. R. Atal and A. Sureka, "Anukarna: A Software Engineering Simulation Game for Teaching Practical Decision Making in Peer Code Review," in *1st International Workshop on Case Method for Computing Education (CMCE 2015)* (New Delhi, India, 2015), 63–70.

27. L. Andrade, E. Grynberg, M. Schots, and V. M. B. Werneck, "InspectorX 2.0: Developing a Multi-Device Game for Software Inspection Education," in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)* (Munich, Germany: IEEE, 2020), 1–4.

28. J. P. R. Guimarães. (2016):, "Serious Game for Learning Code Inspection Skills," Master's Thesis, Universidade Do Porto.

29. B. Morschheuser, L. Hassan, K. Werder, and J. Hamari, "How to Design Gamification? A Method for Engineering Gamified Software," *Information and Software Technology* 95 (2018): 219–237, https://www.sciencedirect.com/science/article/pii/S095058491730349X.

30. M. Mäntylä and C. Lassenius, "What Types of Defects Are Really Discovered in Code Reviews?," *IEEE Transactions on Software Engineering* 35, no. 3 (2009): 430–448.

31. G. Rong, J. Li, M. Xie, and T. Zheng, "The Effect of Checklist in Code Review for Inexperienced Students: An Empirical Study," in *2012 IEEE 25th Conference on Software Engineering Education and Training* (Nanjing, China: IEEE, 2012), 120–124.

32. inc. T., "Thoughtbot Guides," (2019), https://github.com/thoughtbot/guides/tree/master/code-review.

33. M. Staron, *Action Research in Software Engineering* (Cham: Springer, 2020).

34. P. Runeson and M. Höst, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *Empirical Software Engineering* 14 (2009): 131–164.

35. Bilkent University Department of Computer Engineering, "Syllabus to the Course Referred as CS319," (2022), https://stars.bilkent.edu.tr/syllabus/view/CS/319/.

36. Bilkent University Department of Computer Engineering, "Syllabus to the Course Referred as CS453," (2021), https://stars.bilkent.edu.tr/syllabus/view/CS/453/.

37. C. Udeozor, R. Toyoda, F. Russo Abegão, and J. Glassey, "Digital Games in Engineering Education: Systematic Review and Future Trends," *European Journal of Engineering Education* 48 (2022): 321–339.

38. S. Shaphiro and M. B. J. B. Wilk, "An Analysis of Variance Test for Normality," *Biometrika* 52, no. 3 (1965): 591–611.

39. C. Bonferroni, "Teoria Statistica Delle Classi e Calcolo Delle Probabilita," *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze* 8 (1936): 3–62.

40. A. Strauss and J. Corbin, *Basics of Qualitative Research* (Newbury Park, CA: Sage Publications, 1990).

41. 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)* (USA: IEEE Computer Society, 2015), 111–120, https://doi.org/10.1109/ICSM.2015.7332457.

42. Zoom Video Communications Inc, "Video Conferencing, Web Conferencing, Webinars, Screen Sharing," (2020), https://zoom.us/meetings.

43. W. R. King and J. He, "External Validity in IS Survey Research," *Communications of the Association for Information Systems* 16 (2005): 45, http://dblp.uni-trier.de/db/journals/cais/cais16.html#KingH05a.

Appendix A

Surveys

TABLE A1 | Presurvey questions.

Question	Answer options
1. Name	
2. Where are you currently in your studies?	1st year–2nd year–3rd year–4th year–graduate–master–PhD
3. How long is your current industry experience including internships and part-time jobs?	None–0–3 months–3–6 months–6–12 months–1–3 years–3+ years
4. What is your proficiency level related to Java programming language?	Theoretical knowledge, but no working experience–beginner with some working knowledge–intermediate with practical application–advanced with significant experience–excellent with ability to mentor others
5. Which classes did you take or are you taking? (*Note:* Select one or more choices)	Algorithms (CS473)–Object-Oriented Programming (CS319)–Application Lifecycle Management (CS453)–Software Verification and Validation (CS458)–Software Project Management (CS413)–Software Product Line Engineering (CS415)
6. What are your expectations from playing this game?	
Code review knowledge section	
1. What is the extent of your previous knowledge on code review?	None–I have basic familiarity with CR, but no experience applying it–I have knowledge of CR and used it in small or artificial projects–I have applied CR in real-world setting–I have applied CR extensively in a real-world setting
2. What is the definition of code review? Please explain it briefly. If you answered the above question with a "no," skip this question.	
3. If you participated in code review before, which code review system/method did you utilize?	
4. How confident are you about your code review knowledge/skills?	Not confident at all–slightly confident–somewhat confident–fairly confident–extremely confident
5. How important do you think code review process is?	Not important at all: I could not see any reason to make code review important–not very important: there are some points that makes the code review important, but in general, it is not–fairly important: although there are reasons for making the code review important, I could not list them–very important: there are obvious reasons in order to call code review important–fundamental: code review is one of the most important processes in software development
6. How familiar are you with the following code review concepts?	Review checklists, code review actors, reviewer comments, code review standards, coding style guidelines, code review workflow
Familiarity levels	Not familiar–slightly familiar–moderately familiar–familiar–very familiar

TABLE A2 | Postsurvey questions.

Question	Answer options
1. Name	
2. How much have you enjoyed the game elements in the serious game?	Very enjoyable–enjoyable–neutral–boring–very boring
3. How do you feel about the impact of game elements in the teaching of the code review process?	Very satisfied–satisfied–neutral–unsatisfied–very unsatisfied
4. How satisfied are you with the fairness of scoring in-game?	Very satisfied–satisfied–neutral–unsatisfied–very unsatisfied
5. Please provide explicit reasons for your dissatisfaction, if they exist.	
6. Please rate the game components according to their usefulness.	Stella's log, in-game guide, checklist, game tutorial, practice level (Level 0), answer explanations, hints, quiz–not useful at all–slightly useful–somewhat useful–fairly useful–extremely useful
7. What are the three things you learned while playing the game?	
8. Do you have any positive or negative feedback for the game?	
Code review knowledge section (identical to presurvey)	
1. What is the extent of your previous knowledge on code review?	None–I have basic familiarity with CR, but no experience applying it–I have knowledge of CR and used it in small or artificial projects–I have applied CR in real-world setting–I have applied CR extensively in a real-world setting
2. What is the definition of code review? Please explain it briefly. If you answered the above question with a "no," skip this question.	
3. If you participated in code review before, which code review system/method did you utilize?	
4. How confident are you about your code review knowledge/skills?	Not confident at all–slightly confident–somewhat confident–fairly confident–extremely confident
5. How important do you think the code review process is?	Not important at all: I could not see any reason to make code review important–not very important: there are some points that makes the code review important, but in general, it is not–fairly important: although there are reasons for making the code review important, I could not list them–very important: there are obvious reasons in order to call code review important–fundamental: code review is one of the most important processes in software development
6. How familiar are you with the following code review concepts?	Review checklists, code review actors, reviewer comments, code review standards, coding style guidelines, code review workflow
Familiarity levels	Not familiar–slightly familiar–moderately familiar–familiar–very familiar

TABLE A3 | Survey results (values are expressed as percentages).

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

Appendix B

Quiz Summary

TABLE B1 | Quiz result summary Part 1.

Question		Pre(correct)	Post(correct)	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 ensure 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 organization-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

(Continues)

TABLE B1 | (Continued)

Question		Pre(correct)	Post(correct)	Diff
8. Which of the following are logic-related functional defects?	Semantic duplication	124	260	136
	Bad variable initialization	114	174	60
	Suboptimal algorithm/performance	224	260	36
	Bad compare operations	253	273	20
	Semantic dead code	137	243	106
	Bad compute operations	254	270	16

TABLE B2 | Quiz result summary Part 2.

Question		Pre (correct)	Post (correct)	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 lifecycle 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)				