Code Reading Dojo: Designing an Educationally-oriented Mobile Application Aimed at Promoting Code Reading Skills

Zahra Ghaed

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science and Application

Stephen H. Edwards, Chair
Donald S. McCrickard
Francisco J. Servant Cortes

February 15, 2017
Blacksburg, VA

Keywords: computer science education, code reading skills, educational game, mobile game
Copyright 2017, Zahra Ghaed
In recent years, much attention has been directed to the use of educational games for learning computer science concepts. The motivation of game-based learning with positive experience has been deeply studied in the literature, but game design for improving code reading skills have much room for improvement. Being good at the reading code is important to a professional developer. To address this issue, we defined a new educationally-oriented mobile game application, aimed at promoting the development of code reading skills in a new and fun way. The strategy of this game is to find errors in pieces of codes. At each level, students should find all syntactic and semantic errors in the code in a certain time in order to advance to the next level.

Of the numerous programming languages, we chose Java because it is one of the most popular programming languages. In many colleges, Java plays a major role in introductory courses. Our vision is to allow instructors to employ the game in their introduction to programming in Java course. In addition, we hope it could be adapted for use in introductory courses using different programming languages.

Data collected during the project helps us evaluate the impact of game-based learning on code reading in programming languages. We asked undergraduate students at the department of computer science at Virginia Tech to play with the game during Spring 2017 semester. The collected data analyzed, and it students believe that Code Reading Dojo improves their code reading skills in Java and overall programming ability, in additions to help them find errors in their own program.
General Audience Abstract

In recent years, much attention has been directed to the use of educational games for learning computer science concepts. The motivation of game-based learning with positive experience has been deeply studied in the literature, but game design for improving code reading skills have much room for improvement. Being good at the reading code is important to a professional developer. To address this issue, we defined a new educationally-oriented mobile game application, aimed at promoting the development of code reading skills in a new and fun way. The strategy of this game is to find errors in pieces of codes. At each level, students should find all syntactic and semantic errors in the code in a certain time in order to advance to the next level.

Of the numerous programming languages, we chose Java because it is one of the most popular programming languages. In many colleges, Java plays a major role in introductory courses. Our vision is to allow instructors to employ the game in their introduction to programming in Java course. In addition, we hope it could be adapted for use in introductory courses using different programming languages.

Data collected during the project helps us evaluate the impact of game-based learning on code reading in programming languages. We asked undergraduate students at the department of computer science at Virginia Tech to play with the game during Spring 2017 semester. The collected data analyzed, and students believe that Code Reading Dojo improves their code reading skills in Java and overall programming ability, in additions to help them find errors in their own program.
To my husband, MohammadReza and my parents
Acknowledgements

I would like to thank my advisor, Professor Stephen H. Edwards, for his encouragement, positive feedback and mentorship. His support has been a key part in my master’s defense. I learnt a lot from our conversations during the weekly meetings. I would also like to thank my committee members Dr. Scott McCrickard and Dr. Francisco Servant for being in my advisory committee and their support throughout the process.

I am thankful to Dr. Lujean Baab for her financial support throughout my master’s study. It was a great pleasure working in the Technology-enhanced Learning and Online Strategies (TLOS) at Virginia Tech during my master’s study.

I am beyond grateful to my soulmate and husband, MohammadReza for his love and encouragement. His love has always been a motivation for me. This thesis would have never been completed without his endless friendship and love.

Lastly, I am extremely grateful to my parents, Sorour and Mohammad for their unconditional support and great advice in my entire life.
Table of Contents

Chapter 1 Introduction ........................................................................................................... 1
  1.1 Motivation ..................................................................................................................... 1
  1.2 Background .................................................................................................................. 1
  1.3 Problem Statement ...................................................................................................... 2
  1.4 Outline Solution Strategy: Code Reading Dojo .............................................................. 2
  1.5 Thesis Organization ...................................................................................................... 4

Chapter 2 Literature Review .................................................................................................. 5
  2.1 Related Work .................................................................................................................. 5
    2.1.1 Using Mobile Devices in Education ......................................................................... 5
    2.1.2 Using Games in Introductory Computing Courses ................................................... 6
    2.1.3 Deliberate Practice .................................................................................................. 6
    2.1.4 Teaching Code Reading Skills ................................................................................ 7
  2.2 Summary of Key Differentiators .................................................................................... 7

Chapter 3 Code Reading Dojo Design Goals ......................................................................... 8
  3.1 Deliberate Practice ......................................................................................................... 8
  3.2 Gradual Engagement .................................................................................................... 9
  3.3 Accessibility ................................................................................................................ 9
  3.4 Learnability ................................................................................................................ 9
  3.5 Enjoyment ................................................................................................................... 10

Chapter 4 Implementation and Walkthrough ......................................................................... 11
  4.1 A Java Tool for Generating Content .......................................................................... 12
    4.1.1 Common Java Errors .............................................................................................. 12
    4.1.2 Source of the Original Code ............................................................................... 15
    4.1.3 Inject Errors in Original Pieces of Codes .............................................................. 17
  4.2 Client Side .................................................................................................................... 19
    4.2.1 Main Menu ............................................................................................................ 22
    4.2.2 List of Levels ........................................................................................................ 25
    4.2.3 Playing and Repeating a Level ............................................................................. 25
    4.2.4 Setting up Profile .................................................................................................. 28
    4.2.5 Survey ................................................................................................................... 29
4.3 Server Side .......................................................................................................................... 30
  4.3.1 Program Model ........................................................................................................ 31
  4.3.2 User Model................................................................................................................... 31
  4.3.3 Info Model ................................................................................................................... 31
  4.3.4 Tap Model ................................................................................................................... 32
  4.3.5 Survey Model ............................................................................................................. 32

Chapter 5 Evaluation ........................................................................................................... 34
  5.1 Analyzing Collected Data .............................................................................................. 34
    5.1.1 Levels ....................................................................................................................... 34
    5.1.2 Errors ...................................................................................................................... 36
    5.1.3 Taps ........................................................................................................................ 39
    5.1.4 Improvement ......................................................................................................... 45
  5.2 Survey Results ................................................................................................................. 45
    5.2.1 Survey 1 - Improving code reading skills ................................................................. 46
    5.2.2 Survey 2 - Improving code writing skills ................................................................. 46
    5.2.3 Survey 3 - Assisting in error-findings skills .............................................................. 47
    5.2.4 Survey 4 - Improving overall programming knowledge .......................................... 48

Chapter 6 Conclusion and Future Work .............................................................................. 49
  6.1 Contribution to CS Education ....................................................................................... 49
  6.2 Future Work ................................................................................................................ 50

References ............................................................................................................................. 51

Appendix A Virginia Tech IRB Approval ............................................................................ 53
Appendix B Original Source Code of Code Reading Dojo ................................................ 56
List of Figures

Figure 4.1 Code Reading Dojo’s architecture includes three components, a tool for generating content, a client code, and a server side ................................................................. 12
Figure 4.2 Some regular expressions that are used in the Java Tool .................................................. 18
Figure 4.3 An example of mutation for generating a faulty program with three errors .................. 18
Figure 4.4 The first low-fidelity prototype of Code Reading Dojo has the main functionalities of the final version ........................................................................................................... 20
Figure 4.5 The architecture of ember application contains five components and the model connects the client side to the server side ........................................................................ 21
Figure 4.6 The first screen players of the application see contains sign up, sign in, or quick play buttons ..................................................................................................................... 22
Figure 4.7 Main menu for logged in users contains play, instruction, complete profile, and logout buttons .......................................................................................................................... 23
Figure 4.8 Minimum information is asked from the user in the sign up page ................................ 24
Figure 4.9 Login page needs email as a username and password ......................................................... 24
Figure 4.10 List of levels for a user who has completed 8 levels ......................................................... 25
Figure 4.11 Quick instructions before each level provides a short description for the user ........... 26
Figure 4.12 Top: The user correctly identifies three of errors in this level and incorrectly clicks on one word Bottom left: The user finds all the errors, Bottom right: The user’s score becomes negative and user loses the game ................................................................. 27
Figure 4.13 After a user complete a level, a description is provided for all errors ......................... 28
Figure 4.14 User's profile page shows email address and level completed and a button for completing profile ................................................................................................................. 29
Figure 4.15 Code Reading Dojo database has five tables .................................................................... 30
Figure 5.1 In what level did players stop playing? ............................................................................... 35
Figure 5.2 Average time per play for levels 1-12 increase by level increase ..................................... 36
Figure 5.3 Errors success rates – First time ......................................................................................... 37
Figure 5.4 Success rate per error ........................................................................................................ 38
Figure 5.5 Average time to find errors – First time ............................................................................. 39
Figure 5.6 Success Rate per level for all players who completed at least level 6 ............................ 45
Figure 5.7 Survey 1 - Improving code reading skills ........................................................................... 46
Figure 5.8 Survey 2 - Improving code writing skills ......................................................................... 47
Figure 5.9 Survey 3 – Assisting in error-finding skills ....................................................................... 48
Figure 5.10 Survey 4 – Improving overall programming knowledge .................................................. 48
List of Tables

Table 1 Replacement Mutation ................................................................. 13
Table 2 Insertion mutation ................................................................. 14
Table 3 Deletion Mutation ................................................................. 14
Table 4 Top 10 Mistakes in All Levels ..................................................... 39
Table 5 Top 5 Mistakes in Level 1 ......................................................... 40
Table 6 Top 5 Mistakes in Level 2 ......................................................... 41
Table 7 Top 5 Mistakes in Level 3 ......................................................... 42
Table 8 Top 5 Mistakes in Level 4 ......................................................... 43
Table 9 Top 5 Mistakes in Level 5 ......................................................... 43
Table 10 Top 5 Mistakes in Level 6 ......................................................... 44
Chapter 1

Introduction

1.1 Motivation

Computer science education has invented several new and innovative ways of teaching programming languages in recent years. Many students find learning programming languages challenging. Game-based learning is considered an encouraging method to motivate students to learn programming languages. There are many games that can be used as assignments in computer science classrooms. Young people spend a substantial amount of their time playing with their phones. Therefore, having these educational games on mobile phones provides a convenient means to encourage students learning programming languages concepts.

Bayliss in her research introduces three different methods that introductory programming courses can be taught employing mobile games [1]. According to her paper, students may learn through programming a game, playing a game, or analyzing elements in a game. Programming a game or analyzing elements in a game needs to be included in a course material. However, games can be played by students outside of their classroom in their spare time. Additionally, mobile games are largely accessible for students, and they can play easily in any place. Therefore, our focus is on the larger area of computer science students who are willing to spend more time practicing programming on their mobile phone and improve their knowledge even if it is not as part of their course.

1.2 Background

There have been developments in educational mobile games with many of them teaching the basics of programming in a fun and accessible way through video games. Eagle and Barnes introduce a game to help students learn basic programming techniques: basic loops, arrays, and nested for loops [6]. Lee and Ko in another work developed an educational game aiming to teach a specific unit of learning which is named Gidget [11].

The most important aspect of an educational game in addition to being fun, is to contain useful content. There exist games covering an in depth knowledge of a specific area while there are number of games, in contrast to the mentioned category that are designed to include the general area of computer science. Gibson and Bell summarizes a number of offline and online educational games in computer science based on their availability, desktop, mobile, browser, or unplugged [2]. Based on the paper, there have been less developments in mobile games compared to other kinds of educational games. Many games are focused on programming concepts which cover a breadth of areas for introductory courses, while binary and network and security are the two largest topics
focused on a specific in-depth concept. Most mobile games for introductory courses such as Cargo-bot are mainly puzzles with simple tasks of moving forward or left to make correct instructions [3]. These games are mostly video games aimed at learning computer science concept. Nevertheless, there are fewer games designed for reading source codes of programs during the game, and concentration on improving code reading abilities still lacks in educational games.

An educationally-oriented game focusing on code reading skills, provides an environment for students to understand the meaning of their codes. Improving code reading skills will also cause people writing better code. Moreover, these kinds of games may reduce the reliance of the programmer to a compiler and other debugging tools and help them finding errors in their code faster. The primary purpose of improving code reading skills is to avoid common mistakes and misunderstandings in language features in future.

1.3 Problem Statement

Designing and evaluating an educationally oriented mobile game application aimed at promoting code reading skills is considered as the overall research goal in this study. In order to address this goal, we divided this goal into some more specific sub problems. The sub-problems include:

- Providing a collection of common mistakes and misunderstandings in Java
- Analyzing the difficulty of injected errors and providing a list of most difficult errors for Java users
- Analyzing the performance of players during the game
- Collecting top mistakes in finding bugs for beginners
- Evaluating the impact of game-based learning in improving code reading skills

By designing and evaluating a new educationally-oriented game, we plan to answer the sub-problem statements and reach our research goal. Collecting and categorizing common mistakes are the first steps through achieving our goal. After designing the game, collecting the data from participants will help us identify the most difficult errors that can be captured by Java users, and most common mistakes that Java users make in finding bugs. In addition, considering deliberate practice, obtaining the data on success rate of players in an educationally-oriented game is the other sub-goal of this study.

1.4 Outline Solution Strategy: Code Reading Dojo

Considering the shortfall of mobile-based applications for improving code reading skills at the college level, we believe that it’s time for developing an educationally-oriented mobile game for enhancing such skills. Code Reading Dojo is a new educationally-oriented Android and iOS game application aimed at promoting the development of reading codes and finding errors in a new and enjoyable way. It is accessible through Google Play and Apple Store for all students who are interested in improving their own code reading skills. Of the numerous programming languages, we have selected Java, because it is one of the most popular programming languages. In many colleges, Java plays a significant role in introductory courses. Our vision is to allow instructors to employ the game in their introduction to programming in Java courses. Furthermore,
our goal is to make it adaptable for use in introductory courses using different programming languages.

An essential part of Code Reading Dojo is to provide a full breadth of content that would be considered in introductory courses to Java. As a result, we include fundamental Java syntax and semantic. The main idea of the game is that players should find errors in the code and each level has a different difficulty of errors; lower levels consist of simple syntax errors and higher levels include more semantic and run-time errors. The vision of Code Reading Dojo is to enhance code reading skills in Java. Our game can be effective at countering the impact of both ordinary mistakes and misunderstood Java language features. By finding errors in the code, we hope that students can find errors easier in their code or team projects’ code.

The main source code that is included in the game is related to fundamentals of Object-Oriented programming in Java. We have covered breadth common mistakes in object-oriented programming such as classes, objects, methods, instantiation of a class, and messages sent to objects. We have developed Code Reading Dojo in a way that covers relevant materials of the course so that it contains a breadth of knowledge requires for introductory programming courses and students can benefit from playing it. We have covered 18 different types of common mistakes and provided more information about each error after completing each level. Some of the common mistakes for beginner levels include missing a semicolon or unbalanced brackets while for higher levels, the focus is on more advanced concepts of misunderstanding interface or abstract classes.

The target audience for Code Reading Dojo consists of students who have just started to learn Java or beginners with limited knowledge of Java. According to [1], beginners tend to make more mistakes in their programs and the earlier we teach students to avoid mistakes is better. For instance, students in their first years of college do not have enough programming experiences to know about all common mistakes. One way to decrease students’ common mistakes in programming assignments is to motivate students to practice reading and writing strong programs.

The application of educational software in undergraduate courses at the department of computer science at Virginia Tech has been tested before. The introductory courses at Virginia Tech include many programming assignments. However, these assignments are mostly concentrated on writing codes rather than examining the code reading skills. Thus, Code Reading Dojo could be a supplement for students in their first years of learning Computer Science.

After collecting a wide area of common mistakes and misunderstandings in Java, categorizing these errors helps us evaluate the game more easily. Nineteen different categories of errors are introduced in Code Reading Dojo. The 19 categories are classified into the following types: Having extra elements in Java, missing an element in Java, or misusing elements of Java. Most of the errors are repeated in multiple levels which help the user practice the same types of error many times.

In order to answer our problem statements, we capture every action of the users. By every click made by a user, the clicked word, time to click on the word, user's email, level, and successful or failed attempts are captured. Collecting this data helps us in analyzing difficulty of errors in each level and the users’ performance in finding these errors. Code Reading Dojo primary sub-goals are to find the difficulty of various types of errors and obtain users' mistakes along with evaluating users' performance. Evaluating errors that users’ take more time to find as well as errors that are identified by smaller number of users can help us in finding the most difficult errors.
Recording and keeping track of success taps over total taps for all players assists us in finding users' performance.

In addition to analyzing users' collected data, we want to know the players' opinions on Code Reading Dojo. Therefore, we added a survey after level six of the program to find whether players believe Code Reading Dojo helps them in improving their ability to read and write code, finding bugs in their own program, and improving overall programming knowledge.

### 1.5 Thesis Organization

The remainder of this thesis includes as follows. Chapter 2 goes through related works related to educational mobile game applications and what we are doing differently in Code Reading Dojo. Chapter 3 discuss the design goals of Code Reading Dojo. In Chapter 4, we discuss details of Code Reading Dojo implementation and different components included in the application. We use walkthroughs to explain how this game works. Chapter 5 we talk about how do we believe we solve different problems. We discuss our analysis and evaluation on students’ data in this chapter. Finally, in chapter 6, we explain our contribution to computer science education and points to some ways in which the work can be improved upon.
Chapter 2

Literature Review

In this chapter, related works to Code Reading Dojo will be reviewed, including the importance of deliberate practice in learning, using mobile devices in education, using games in education, and research focused on code reading skills. This chapter ends with a discussion on the main differences between Code Reading Dojo and related works.

2.1 Related Work

Code Reading Dojo's main design goal is focused on deliberate practice on improving code reading skills. Therefore, it would be useful to look at related research that has been done on deliberate practice and teaching code reading skills. Moreover, there exist many efforts in computer science education aimed at using a game in teaching computer science courses [6, 11, 12]. In addition, mobile devices play a major role in the higher education [4]. In this section, related works in using mobile devices and games in education will be studied. In the end, we explain that previous research focused on the importance of deliberate practice, and teaching code reading skills.

2.1.1 Using Mobile Devices in Education

There have been numerous studies in integrating mobile devices into the curriculum [4]. Students have easier access to mobile applications since they carry their mobile devices with them all the time. Martin and Ertzberger demonstrate a significant difference in comparing mobile devices and computers as learning tools in instructional design in their research.

In their study, they defined "here and now mobile learning" as a learning type for students since they have access to the information anytime, anywhere [8]. The use of mobile devices in a classroom could provide benefits to the students. Glackin, Rodenhiser, and Herzog illustrate that mobile devices improve student's educational experience [7].

Ituma demonstrates mobile learning as an active learning [9]. In Ituma's research, mobile learning is considered a way for enhancing student engagement and enriching learning environment. Gikas and Grant did a research study on three different universities, and their study shows a change in students' learning using mobile devices [10]. In Gikas’ and Grant’s research, they discover enhancement in students' learning regardless of possible limitations such as difficulty due to a mobile device small screen.
2.1.2 Using Games in Introductory Computing Courses

There has been the development of many games with a positive experience in enhancing learning in the other research practices [6, 11, 12]. Eagle and Barnes develop a game for promoting students’ attitude and learning in introductory to CS courses [6]. Wu's Castle is a game focus on teaching loops and array concepts where students need to change programs in a 2D environment. After each change, the game provides feedback and helps students execute the programs in a visual environment. In Eagle’s and Barnes’ research, small pieces of interactive instructions help players learn while playing the game. The researchers in the Wu's Castle research compare results of a control group using Wu's Castle before an assignment and a group of students playing Wu's Castle after the assignment. Their results show that students who played Wu's Castle before their assignment have a better understanding of loops and arrays concepts.

Lee and Ko have developed Gidget, an educationally-oriented game aimed at debugging pieces of code [12]. In this game, Gidget is a robot who is trying to debug a code and ask students to help him either by writing missing lines of code or fixing existing code. During the game, Gidget gives feedbacks to students regarding their progress. Their feedback helps students design and analyze basic algorithms. Lee’s and Ko’s Gidget research demonstrates that with the personable feedbacks of the Gidget robot in the game, participants completed more levels and gained more skills in programming.

In the Gidget research, Lee, Ko, and Kwan show that adding assessments to a game improves players' performance and engagement [11]. In the Gidget research, they add assessments to improve novice programmers’ engagement in learning programming. Their results indicate that for those players who have assessments in their game, they complete more levels in a shorter time which demonstrates improving in engagement and performance. Their study shows that adding assessments to a game improves players’ engagement and they are learning programming concepts with debugging incorrect codes included in the game.

2.1.3 Deliberate Practice

Deliberate practice is a specific type of practice that is purposeful and well-organized. In a research on deliberate practice, Ericsson shows that the process of deliberate practice involves activities beyond the comfort zone [13]. People need to define steps of goals and need an expert to help them make a plan for reaching their goal. Ericsson demonstrates that deliberate practice is the key to achieving high performance in any area. Deliberate practice needs feedback on efforts in order to find your performance and improve it accordingly. In his research, he illustrates that deliberate practice help people to achieve improvement in performance in any field. Code Reading Dojo main goal is to help students practice reading code using well-defined steps and continues feedback. We believe that following Ericsson’s research, deliberate practice on code reading skills can help students improve their ability to read code and find bugs easier.

Gladwell in the "Outlier: The Story of Success" mentions that top groups of violinists spent more hours practicing alone than the others [14]. In the Gladwell book, he shows that to become an expert in a field it takes more than 10,000 hours of practice. This practice is not just repeating skills; it needs to involve feedback from an expert and concentration during the practice.
2.1.4 Teaching Code Reading Skills

There have been many studies in other research that focus on improving learning how to program. However, most of these studies focused on code writing skills and there exists room for improving code reading skills along with furthering its research. Busjahn and Schulte, illustrate that understanding the existing source codes is also important in the ability to program [15]. In their research, code reading skill is considered as a separate skill on its own. It cannot be assumed that code reading skills are gained with learning how to write code.

In a research on an Introductory Java Course for non-technical students, Scaiano, Javadtalab, and Peyton demonstrate that students are more confident in writing code rather than reading code [16]. They find that students have a hard time understanding code because reading code requires a thorough understanding of details and it is hard to give feedback on reading code. Beginners in Java have a hard time resolve finding bugs in their five-line code. Scaiano, Javadtalab, and Peyton suggest that in designing introductory courses in Java, instructors should focus both on reading and writing code, and not just algorithm design.

2.2 Summary of Key Differentiators

There are many discussions and thoughts on how best to teach introductory programming in computer science courses. While there are inevitable benefits and drawbacks to any teaching approach, we feel that the use of mobile devices in teaching introductory programming could provide benefits to students. Students have easier access to these gaming applications since they carry their mobile devices at all times. The other motivations of game-based learning with positive experience has been deeply studied in other research, but game design for learning programming languages have much room for improvement. The aim of most gaming assignments taught in introductory computer science courses is focused on writing code rather than reading code.

Some games focus on limited skills in programming, like loops, arrays, and nested for loops [6], whereas Code Reading Dojo is not limited by specific skills. There exists evidence that shows that using mobile games in introductory courses at the college level of computer science has a positive impact. In our game, we develop a new educationally-oriented mobile game and provide it for students to play within the “Introduction to Programming in Java” course. However, our game is not limited to the mentioned course, and it will be expandable for other programming languages and courses.

In addition, unlike most related works, the target of our work is not limited to specific people, Code Reading Dojo is publicly accessable and free on Apple Store and Google Play so that everyone can benefit from playing the game.
Chapter 3

Code Reading Dojo Design Goals

As stated in the introduction chapter, Code Reading Dojo’s principal goal is promoting code reading skills in a new and fun way. We have tried to keep this game educational so that players would benefit most from playing with Code Reading Dojo. We have developed Code Reading Dojo based on the following goals:

- **Deliberate Practice**: We believe that practice is the crucial key to success. Therefore, practice is one of our goals in Code Reading Dojo design.
- **Gradual Engagement**: Code Reading Dojo is a mobile game that needs to get players involved. Therefore, continuous engagement is among our design goals.
- **Accessibility**: Any user in the world who have one of the two popular mobile operating systems of Android or iOS will have access to Code Reading Dojo. We believe Code Reading Dojo must be easily available to everyone who is interested in improving his knowledge of code reading skills in Java.
- **Learnability**: Code Reading Dojo has a user-friendly interface and learnability is among our design goals.
- **Enjoyment**: One of our design goals is to make this application fun and enjoyable.

3.1 Deliberate Practice

Deliberate practice is our primary target in designing Code Reading Dojo. For deliberate practice, a well-defined, measurable step toward improving knowledge of code reading skills is required. Therefore, we came up with 18 levels of game in three different stages of difficulty. The goals for each level are defined, and the reason for each error is shown to the user. While playing the game, when the user clicks on each word, we unambiguously tell the user whether the word is among errors in the program or not. Moreover, we tell the player if he achieved the goal of winning the game using a short notification of "You Won" or "You Lost."

Sufficient feedback is considered as a part of deliberate practice. Code Reading Dojo's design includes this by continuous feedback. After each level, a description of all errors listed is given, and players can take advantage of learning new common mistakes. Consequently, even if the player is not sure about the word that he just clicked, she learns about the scientific reason behind the error.

Deliberate practice is useless if it includes wrong practices. To achieve the best practices for promoting code reading skills, we included a large area of common mistakes that we collected during our research. The collection covers a wide range of errors in practical source code samples. These original pieces of code are derived from the real projects [20, 21].
Competitive practice can be also a method for deliberate practice. We added competition flavor to the game by encouraging students to advance at least level 6 to be entered into a raffle where 20 people each get a $10 Amazon Gift Card.

Another element of deliberate practice is that it can be repeated many times. Some of the errors included in Code Reading Dojo are repeated multiple times in different levels. Repeating practices helps players remember errors by asking the same mistake multiple time.

3.2 Gradual Engagement

One of our design goals is to make users interested in playing with Code Reading Dojo. With gradual engagement designing, we have shown our users why it would be a benefit to them to register in our application. We do not force users to sign up with Code Reading Dojo. The first screen a user sees is a logo with a short description and a chance to quick play with the application without having to logging in. Code Reading Dojo lets players interact with the game in order to motivate them to create an account with the application if they enjoy the game.

We also decided not to put all of our questions on the first page of sign up. These questions include age, gender, their knowledge of programming languages, and number of different programming language that they know. Instead, users can complete their profiles and answer these questions after they log into the system and playing the game.

Code Reading Dojo's core feature is playing and finding bugs in a game. Therefore, non-registered users can play and benefit from the game. Additional features for logged in users are:

- Keeping track of level completed
- Error descriptions after each level
- Completing a profile
- Filling a survey related to the game

3.3 Accessibility

It is important to design Code Reading Dojo to be usable and accessible. Code Reading Dojo’s design goal is to let users easily download and play with the game. Therefore, we decided to put our app in both of the two major operating systems app stores which are Google Play Store, and Apple Store. This makes our product useful and desirable to not only Virginia Tech students who can benefit from it, but also all the people around the world who are interested in improving their knowledge of code reading skills. It is also free through these two major application stores.

3.4 Learnability

Developing a learnable tool is also an important factor in designing Code Reading Dojo. We want users to understand how to use the interface. Instructions for playing the game is provided
and feedback is given to the user at each level. Code Reading Dojo's design is minimal as we tried to keep steps as less as we could so that players can focus on the content of code and find errors.

3.5 Enjoyment

Code Reading Dojo is fun. One of our players mentioned in social media that "Characters of this game are so cute". We worked with a student in animation department to create fun characters. From Code Reading Dojo’s feedback and review we realize that users like to play with Code Reading Dojo in their spare time and they like Code Reading Dojo as a practice for programming. Game elements that we added to Code Reading Dojo include timer and stars for scores of the user. Players need to find errors in a limited time and they have to get all stars of the game.
Chapter 4

Implementation and Walkthrough

In this section, we describe Code Reading Dojo’s implementation and components. We also present a walkthrough of the game.

One of the most important decisions in the design of Code Reading Dojo is how to generate a faulty code. There are many ways to inject errors in the original source code. One of these ways is to mutate a code using regular expressions. Mutation can be implemented by capturing different regular expressions in a piece of Java code and change the original code. Two possible mutations are statically injecting errors or dynamically injecting errors while players are playing. There are advantages and disadvantages associated with each method. Manually Adding errors to pieces of code is countered as a static way of injecting errors. This method of injecting error is more trivial to implement. However, with static mutation, there will be errors at each level and players may memorize errors by replaying the same level in order to pass.

Dynamic mutation is another way of injecting errors. In Code Reading Dojo’s design, we decided to implement a tool to add errors to original correct source code and generate faulty programs. The Java tool to inject errors is a valuable tool since there will be thousands of different mutations for each code. By having a tool to dynamically inject errors in the code, Code Reading Dojo will have different sets of errors at each level. Code Reading Dojo players will see different versions of a level by replaying it. In this implementation, the effect of memorizing errors will be suppressed.

Although dynamic mutation is more beneficial and closer to Code Reading Dojo design goals, due to the limited time and complexity of implementation of dynamic mutation, we decided to only have one version of each level. The tool for generating faulty code is implemented and used only once for each level to generate levels including errors.

In general, Code Reading Dojo includes three essential components:

- A tool for generating content: Code Reading Dojo is an educational game. Therefore, the content provided to students is the most important part of the game.
- The client side of the mobile application: This part includes the user interface of the application. The user is in direct interaction with this component and it is an essential part in ensuring satisfactory user experience with the software.
- server side and database: This part is the back-end of the mobile application which is responsible for storing all the data collected from the users playing the game. The user has no interaction with this part. Figure 4.1 illustrates how these components fit together for this application.
4.1 A Java Tool for Generating Content

We developed a tool for generating the content of the application. The tool is a Java API that generates codes with errors. Although this tool is not directly connected to the application, it plays an important role in the game. The Java tool creates content of the application.

In this section, we describe the process of implementing the tool. The section starts with covering a large area of errors that are included in the game, and the original code that we have used. The section concludes by explaining how the errors are injected into the original code.

4.1.1 Common Java Errors

As the first steps of designing the game, we started collecting a large area of common mistakes and misunderstood Java features. One way of avoiding these types of errors is to familiarize students with them. We developed a tool to inject common mistakes in Java into sets of small pieces of codes which automatically generated mutated code. There exist many automated tools for generating mutated code at the source or the byte-code level. µJava, is one example of source code mutation and method level mutations [20]. Offutt in the µJava study provides a list of method level operators that are available in µJava. In contrast to the paper, the target of our application is a basic mutation of both methods and class level operations. Additionally, our specific targets are Java beginners hence we focused only on the fundamentals of the Java language. Therefore, we have developed a tool to generate mutated faulty code at the source code automatically. We included three types of mutation in our tool:
- **Replacement mutations**: The most common type of error in writing code is confusing different elements of the language such as using the wrong operator [17]. There are different types of mutation errors at class or method level in Java which consist of mutation on operators, a mutation on using different data types, or mutation on object-oriented features such as inheritance. Offutt experimentally determines five mutation operators for various programming languages. Table 1 illustrates how we generate mutations in operators. In our 18 levels of Code Reading Dojo, we have 27 different types of errors. Twenty-two of these errors are considered as replacement mutation. The difficulty of these errors is different varies with the level of the game. In the lower level there are simple errors such as replacement in arithmetic statements (method level mutation). In the higher level errors such as confusing the use of extends versus implements (class-level mutation) are included.

<table>
<thead>
<tr>
<th>Mutation Name</th>
<th>Number of levels included</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types Replacement</td>
<td>Beginner: 5 Intermediate: 6 Advanced: 8</td>
<td>int fact (int num) (\rightarrow) float fact (int num) (level 7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arithmetic Operators Replacement</td>
<td>Beginner: 3 Intermediate: 1 Advanced: 1</td>
<td>num = num + 1 (\rightarrow) num = num -1 (level 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational Operators Replacement</td>
<td>Beginner: 2 Intermediate: 1 Advanced: 1</td>
<td>if (num (\leq) 2) (\rightarrow) (num (\geq) 2) (level 10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java Keywords Replacement</td>
<td>Beginner: 1 Intermediate: 2 Advanced: 2</td>
<td>public class Game (\rightarrow) public Class Game (level 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment Operators Replacement</td>
<td>Beginner: 0 Intermediate: 1 Advanced: 0</td>
<td>count += 1 (\rightarrow) count ++ 1 (level 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Level Replacement</td>
<td>Beginner: 0 Intermediate: 0 Advanced: 2</td>
<td>public interface publication item (\rightarrow) private interface publication item (level 17)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditions Replacement</td>
<td>Beginner: 0 Intermediate: 1 Advanced: 0</td>
<td>for (Book book:books) (\rightarrow) while (Book book:books) (level 11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>public class Game (\rightarrow) public interface card Game (level 12)</td>
</tr>
<tr>
<td>Inheritance Replacement</td>
<td>Beginner: 0 Intermediate: 1 Advanced: 4</td>
<td></td>
</tr>
</tbody>
</table>
• **Insertion mutations:** This type of mistake occurs when the programmer inserts extra elements such as curly brackets in the code. Another type of mutation is adding an extra element to the original code. Adding extra arithmetic or logical operators is one example of insertion mutation. For instance, using `++` instead of `+` in `X=X+1` is an insertion mutation. Another example of using unnecessarily elements is using a semicolon in a function declaration. We tried to cover important common mistakes in Java and added four different kinds of adding mutations in the game. Table 2 shows insertion mutations that we have employed in Code Reading Dojo.

<table>
<thead>
<tr>
<th>Mutation Name</th>
<th>Number of levels included</th>
<th>Example</th>
</tr>
</thead>
</table>
| Curly Bracket Insertion     | Beginner: 3               | } à }\}
|                            | Intermediate: 2           |                             |
|                            | Advanced: 3               | (level 1)                    |
| Semicolon Insertion         | Beginner: 2               | int fact (int num) à int fact (int num); { |
|                            | Intermediate: 4           |                             |
|                            | Advanced: 2               | (level 7)                    |
| Arithmetic Operators Insertion | Beginner: 0               | fact (num – 1) à fact (num - - 1) |
|                            | Intermediate: 1           |                             |
|                            | Advanced: 0               | (level 7)                    |
| Assignment Operators Insertion | Beginner: 0               | + à +=                      |
|                            | Intermediate: 2           |                             |
|                            | Advanced: 0               | (level 18)                   |
| Return Type Insertion       | Beginner: 0               | Public Game (Dice dice){ à |
|                            | Intermediate: 1           | Public boolean Game (Dice dice) |
|                            | Advanced: 0               | (level 8)                    |

• **Deletion mutations:** Forgetting to place fundamental components such as parentheses for function calls is another type of error in the code. In addition to replacing and inserting elements, removing necessary parts of the program also creates error. Forgetting to add a type of a variable for initialization is one of the common mistakes in Java. Deleting arithmetic, logical, and conditional operations also create a mutation in the program. There are five different deleting mutations that we used in Code Reading Dojo. Table 3 shows these errors.

<table>
<thead>
<tr>
<th>Mutation Name</th>
<th>Number of levels included</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Deletion</td>
<td>Beginner: 2</td>
<td><code>this.firstName = firstName</code> →</td>
</tr>
<tr>
<td></td>
<td>Intermediate: 0</td>
<td><code>firstName = firstName</code></td>
</tr>
<tr>
<td></td>
<td>Advanced: 0</td>
<td>(level 5)</td>
</tr>
<tr>
<td>Statement Deletion</td>
<td>Beginner: 0</td>
<td><code>return result;</code> → <code>return;</code></td>
</tr>
<tr>
<td></td>
<td>Intermediate: 1</td>
<td>(level 14)</td>
</tr>
</tbody>
</table>
All mutations from table 1, 2, and 3 either create new faulty versions of programs (mutated code), or they create new versions that do not achieve the intended goal of the program. These new mutated programs are not equivalent to the original programs they are mutated after.

4.1.2 Source of the Original Code

“Java Software Solutions: Foundations of Program Design” written by Lewis and Loftus and “Introduction to Java Programming, and Comprehensive Version” written by Liang books which are among of common books used in introductory courses as our main reference for collecting pieces of codes [21, 22]. We take advantage of these useful resources as well as Virginia Tech resources for introductory in order to Java courses to generate a rich collection of original codes. Below presents first six levels of Beginners difficulty. Appendix B provides source code of all 18 levels of the game. Lower levels consist of smaller pieces of code, and higher levels are more complex programs.

```java
//--------------------- Beginner - Level 1 ---------------------
/**
 * Print two statements.
 */
public static void main( String[] args) {
    System.out.println("Hello, World!");
    System.out.println("This is the first level!");
}

//--------------------- Beginner - Level 2 ---------------------
/**
 * Add two numbers and print the result.
 */
public void adder(int num1, int num2) {
    System.out.print(num1 + " + " + num2 + " = " + (num1 + num2) );
}
```
//------------------- Beginner – Level 3 -------------------
/**
 * Increase an input number by 1.
 */
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int num = scan.nextInt();
    num = adder(num);
    System.out.print(num);
}

static int adder(int num) {
    num = num + 1;
    return num;
}

//------------------- Beginner – Level 4 -------------------
/**
 * Convert number grade to letter grade function.
 */
public String findGrade(int points) {
    String grade = "";
    if (points <= 100 && points >= 90) {
        grade = "A";
    } else if (points < 90 && points >= 80) {
        grade = "B";
    } else if (points < 80 && points >= 70) {
        grade = "C";
    } else if (points < 70 && points >= 60) {
        grade = "D";
    } else if (points < 60 && points >= 0) {
        grade = "F";
    } else {
        grade = "NOT VALID";
    }
    return ("Your final grade is " + grade);
}

//------------------- Beginner – Level 5 -------------------
public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String fullName() {

public class FizzBuzz {
    /** A program that prints the numbers from 1 to 100.
     * For multiples of five print “Fizz”,
     * for multiples of seven print “Buzz”,
     * for numbers which are multiples of both five and seven print
     * “FizzBuzz”.
     */
    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++) {
            if (((i % 5) == 0) && ((i % 7) == 0)) {
                System.out.println("fizzbuzz");
            } else if ((i % 5) == 0) {
                System.out.println("fizz");
            } else if ((i % 7) == 0) {
                System.out.println("buzz");
            } else {
                System.out.println(i);
            }
        }
    }
}

4.1.3 Inject Errors in Original Pieces of Codes

Mutations that are described in section 3.1.1 in three different levels are classified. Some mutations are easier to find which are mostly method-level, while other mutations including most class-level mutations are harder to discover. Table 1, 2 and 3 illustrate how mutations were categorized in three levels based on their difficulty. Three different difficulty levels of beginners, intermediate, and advanced were defined with new types of mutation and faulty programs being introduced in each level.
In order to implement these mutations, we defined 18 regular expressions to capture patterns of different pieces of programming languages. Figure 4.2 illustrates some of our initial regular expressions for the Java tool. Each has a specific type of mutation. We used Java Pattern library to implement these regular expressions and defined a hierarchy of classes for each type of mutation. Then, we injected mutations that we collected into 18 pieces of code. At each level, some of these patterns are captured and 3, 4, or 5 of the mutations are injected to the original code based on difficulty level.

Figure 4.3 illustrates one of the examples of which starts with a simple program. Multiple common mutations matched with the program. We randomly selected 3 of them and we injected them into the code. This is result in a program with three different errors included.

```
/**
 * Calculate factorial of a number.
 */

public int fact(int num) {
    if (num <= 1)
        return 1;
    return temp * num;
}
```

```
/**
 * Calculate factorial of a number.
 */

public float fact(int num) {
    if (num >= 1)
        return 1;
    return temp * num;
}
```

Figure 4.2 Some regular expressions that are used in the Java Tool.

Figure 4.3 An example of mutation for generating a faulty program with three errors.
In the architecture of our Java tool, we included three types of mutations mentioned previously. We used an abstract class of Regular Expression, and all three types of mutations (replace, insert, and delete) extend this class since they have different implementation of the mutate method. As a result, we used programs generated but this game in Code Reading Dojo. Players should find injected mistakes in the faulty programs.

4.2 Client Side

For players of a game, the user interface plays an important role. Developing a good user interface for a mobile game is challenging due to the numerous amount of details in implementation. For our proposed educational game, we have anticipated those challenges and found possible solutions for each of them. Before starting the implementation of the game, we thought about different aspects of game development and established a prototype version of our proposed work. Figure 4.4 shows a screenshot of the first prototype of our game. The Figure shows an example in which the player found one mistype in a piece of a program. After finding the mistake, the final score increases (shown as a star in the prototype), and the mistyped word becomes green. The application will show tips and directions which explain strategies to win the game.
Figure 4.4 The first low-fidelity prototype of Code Reading Dojo has the main functionalities of the final version.

There are technologies available for implementing the game. We have selected writing Single Page Applications (SPAs) which allow users to experience native experience. SPA reload any part of the user interface without requiring a server to retrieve the full HTML. SPA was chosen to launch the game on time, and it dynamically updates pages as the user interacts with the application. Moreover, since our game is research based, we may change different parts of the game during the development which is often costlier, time-consuming and requires extra effort in traditional native applications. To make the development process easier and more structured, we decided to develop the game based on the Model View Controller (MVC) pattern and use the Ember.js framework. Ember.js is an open JavaScript library that makes it easier to write client-side web applications that are competitive to native applications [18]. Figure 4.5 illustrates the architecture of our ember application and its relationship with other components. Since our final goal was to develop a mobile game application, we decided to use ember-cordova to build the
mobile application. Ember-Cordova is a CLI/Addon for building native mobile applications with Ember [19]. In conclusion, we decided to develop Code Reading Dojo using Ember and Ember-Cordova.

After completing the prototype idea of our game and finding the suitable technology to use, we started developing the game and provided a list of all different game scenarios and different tasks for each scenario.

In this section, we explain various walkthroughs and details of implementation. We give some of the details about the main menu, sign up or log in, a walkthrough for playing and repeating the game, completing the profile and filling the survey.

Figure 4.5 The architecture of ember application contains five components and the model connects the client side to the server side.
4.2.1 Main Menu

Figure 4.6 demonstrates the first screen of the game when the user has not logged in. There are three options for the users when on the first screen:

- **Sign up**: The user can create a new account that they can use on any Android device.
- **Login**: The user can login with an existing username and password and retrieve user's data.
- **Quick play**: All players can start playing without logging in.

Figure 4.6 The first screen players of the application see contains sign up, sign in, or quick play buttons.

Figure 4.7 illustrates the same main menu for logged in users. There are four different options for the logged user.

- **Play**: This option navigates the user to the list of Code Reading Dojo levels.
- **Instructions**: There is a short description of how players should play with this game.
- **Profile**: For logged in users, there is a profile page that they can complete their information and keep track of levels that they have completed.
- **Logout**: Logged in users can logout at all states of the game.
One advantage for registering in Code Reading Dojo is that the data for each user, such as profile and level completed, are stored in the database. Logged in users can continue to play the levels that they have not completed yet without replaying the levels that they have completed before. Additionally, there are features in the application that only registered players can benefit from. One of these features is the information provided after completing each level which describes the errors in the game. On the other hand, if the user chooses not to register in the game, no data will be stored. We added this option so that those who are not willing to give us their email (username) and other information can still play with the game and improve their knowledge of code reading skills.

- **Sign up**: Figure 4.8 shows the sign up page, which user can create a new account in the application. We don't ask for the user's name or other personal information. However, there is an optional questionnaire after logging in to the game for completing the profile.

Another part of signing up which is critical, is asking players to give us the permission to store their data for research purposes. Participation in the study is voluntary, and players can uncheck the corresponding box. If they wish not to participate. They can read the complete consent form and decide if they agree or not. The consent form is accepted by the Virginia Tech Institutional Review Board (IRB). Appendix B is the agreement we put in the app. if the player changes her mind, she can modify her answer about the collection of her data.
• **Login:** Users that are already registered in Code Reading Dojo can login with their credentials and continue playing. Figure 4.9 shows a screenshot of the login page. Players can go to signup page from login page.

Figure 4.8 Minimum information is asked from the user in the sign up page.

Figure 4.9 Login page needs email as a username and password.
4.2.2 List of Levels

Code Reading Dojo has 18 levels in three different categories. Each category includes a new collection of errors in their programs. We designed level logos with square blocks where each level of difficulty has specific colors, and the number of blocks is the same as the level number. The level completed by logged in users is stored in the database so that they can continue playing from the level that they stopped at. For non-registered users, level completed is stored in the session until they close the application. Figure 4.10 shows a screenshot of a list of levels for a user that completed nine levels, and she can start playing level 10. All players can replay levels that they have previously completed.

![Image of a list of levels for a user who has completed 8 levels.](image)

4.2.3 Playing and Repeating a Level

Playing a level is the most important part of the game. First, we give short instructions of what players should do (Figure 4.11). Players should find mistakes by touching on words in a piece of code, and they need to find at least three errors. In the level page, players should confirm that they want to start the game. After the confirmation, a stopwatch starts, and a set of empty stars appears on top of the code. Figure 4.12 top shows a game when the user has found 3 out of 4 errors. Users must find all errors at each level to be able to progress further. These errors can be a compile or runtime error, or a part that is not intended for the program. If the selected word is chosen correctly, the color of that particular code becomes green which means that the player has successfully found one of the errors in the code and the score increases by one. If the player incorrectly locates a part of the code, the color of the selected word becomes red and the score
decreases by one. Players of the game should find all errors in the code and their final score must be positive. If their scores become negative, they will not be allowed to proceed to the next level. However, they can retry the same level. Here is the summary of rules for winning a level in Code Reading Dojo.

- At each level, the player should find all errors in the code. By finding each error the player's score increases by one.
- The player may incorrectly identify errors in the code, the score will be decreased by one per each wrong click.
- The total score of the player must be positive to win.

Figure 4.11 Quick instructions before each level provides a short description for the user.

There is a meaning behind each of these rules. First, we don't want a player to pass a level without finding all of the errors of the game. If the mistakes on code go unrecognized, the player believes that was not a mistake, and it remains in her mind. Second, we don't want to give unlimited chances to the player so that he randomly clicks on the words in the game to proceed to the next level. Finally, the total number of wrong clicks must be less than the number of errors in the level. In beginners’ level, there are three errors, and players may be wrong by locating on correct words three times. In intermediate level, there are four errors, and players can be wrong twice. In advanced levels, there are five errors and players can click on four more correct words.

After finding all the errors, users can click on the Next Level button to go to the next level. Figure 4.12 left shows a game that the player win the level. The user may also lose the game in a level and click on Retry to replay the game (Figure 4.12 right).
Figure 4.12 Top: The user correctly identifies three of errors in this level and incorrectly clicks on one word Bottom left: The user finds all the errors, Bottom right: The user’s score becomes negative and user loses the game.
When a user wins a level, there will be a short description for all errors and reasons for mistakes. Figure 4.13 shows an example of the description of level 7. We believe that this is an important part to describe common mistakes in Java to make sure that users understand why they are not correct.

Figure 4.13 After a user complete a level, a description is provided for all errors.

4.2.4 Setting up Profile

Code Reading Dojo is publicly available through Google Play and Apple Store. In this matter, one aspect of our research project is to analyze who our targets are. For logged in users, there is a profile including their email address and level completed. Moreover, they can change their response to the permission of using their data in the research project. At the end, there is a button for completing their profile (Figure 4.14).
In completing user profile, we are interested in knowledge of different programming languages and years of their experience. We also want to know their age and gender in order to classify them. We will explain more in evaluation chapter about questions included on the more information page. Below are questions that we include in the complete profile page:

1. Your Gender? Female or Male.
2. How old are you?
3. How many months/years do you have programming experience?
4. Between one to five, how much do you enjoy mobile games?
5. Which programming languages have you used before? (Java, C, C++, Python, JavaScript)

4.2.5 Survey

Every player who completes at least six levels of the game can access to a survey regarding their experience with the game. There are four questions in the survey. Questions of this survey is provided below and we will discuss more on how we came up with these questions in the evaluation chapter. Players should indicate how they strongly agree or disagree with all the following statements. We used likert scale between strongly disagree to strongly agree.

1. Using Code Reading Dojo improves my code reading skills in Java.
2. Using Code Reading Dojo improves my code writing skills in Java.
3. Using Code Reading Dojo improves my ability to find errors in my own program.
4. Using Code Reading Dojo improves my overall programming ability.

4.3 Server Side

The architecture of Code Reading Dojo is client-server. We need a server API for our client to store and retrieve data and communicate with the client. After doing research on different frameworks for our back-end, we decided to use Ruby on Rails as a framework for our server [23]. Models in Rails application are very close to models in Ember application, and we decided that Rails is a good match for our application since the database is one of the most important components to the functionality of the game. Both Ember and Rails follow convention over configuration. We decided to have the same models on both Ember and Rails API.

As mentioned, due to the importance of database in our rails API, we store different types of data from users and save it to the database. We also retrieve data from programs to display in our Ember client. Figure 4.15 illustrates how different models of Code Reading Dojo and their relationships build the database of our application. An explanation of models in our Rails API will be given in the following section.

Figure 4.15 Code Reading Dojo database has five tables.
4.3.1 Program Model

The initial model that we came up with is the program model. The first thing that we are required to have in our server side is the content of pieces of code for each level. We decided to store this data on the server in order to be able to manipulate the server or add more levels to the game. We do not want to limit the application to levels that we put on the client side. On the other hand, storage is another aspect of this decision. Storing all data in the client-side makes our client application larger. Since Code Reading Dojo is a mobile application, we tried to keep the size of the application as small as possible. The Program table consists of the following fields:

- Code: This field contains the heart of our application. We store all pieces of codes in the Program table.
- Difficulty: Each code has a difficulty level. The difficulty level can be beginner, intermediate, or advanced.
- Level: In addition to difficulty, each code has a level within its category of difficulty.
- Errors: Each instance of a program contains errors in the code. All errors are stored together in a field, and they are separated by commas.

4.3.2 User Model

The User model is responsible for storing and retrieving user data. We need user information for authentication. In order to perform authentication, we store email addresses as the username and the encrypted passwords. Additionally, we store level completed by players in the Users table and retrieve this data for our list page. Players do not need to complete levels that they have already completed in their previous login session. Furthermore, we store user's response to the consent form; whether they are willing to let us use their data in our research or not. Here is the summary for the User table.

- Email: Email is the username in our application. The reason for selecting email address as the username is to have the capability to contact the users in the future to announce winners of $10 Amazon gift card.
- Password: We store encrypted password and use username and password credentials for authentication.
- Level Completed: After a user completes a new level, we update level completed in the Users table.
- Consent: Players may modify their agreement to the consent form in their profile.

4.3.3 Info Model

In addition to storing the necessary data for each user, we provide an optional questionnaire for users to complete. The info table has a foreign key to the User table using the email address as
the foreign key. None of the fields in this table are mandatory, and if the user decides not to answer any of them, we store NULL instead. The following fields are included in the Info table.

- Email: Email is the foreign key to the User table. We decided to have a separate table for more information with a foreign key to the Users table since completing profile is a voluntary act for users. If we store these fields in the Users table, the Users table becomes a sparse table with many NULL variables in it.

- Gender: We store gender as a predefined string which can be “male” or “female”.

- Age: Age is one of the demographic questions asked in surveys. We store age in the Info table, and it's optional to answer.

- Experience: Years of programming experience is among questions that significantly affect our evaluation. We want to see how people with different years of experience play with Code Reading Dojo.

- Enjoy: In general, level of enjoyment with plying mobile games affects users to like Code Reading Dojo or not. We store this field as a Likert-typed approach, between strongly agree to strongly disagree.

- Languages: We are storing programming languages that users know as a string separated by commas.

### 4.3.4 Tap Model

Tap model is the model that we mostly use for our research. We collect all actions that are useful for our research such as times to click on each word and if they find an error or they clicked on a correct word. Every time that the user clicks on a word, it creates a new Tap instance. The Tap table has a foreign key to the User table using username as the point of communication. The relation between User and the Tap table is a one to many relationships. The following fields are stored in the Tap table.

- Email: Email or username is the foreign key of the Tap table to connect Info with users. We complete this field from the session.

- Word: This field contains the word in which the user clicks on. It can be correct or not. We will evaluate that in a separate success field.

- Success: Whether the clicked word is among one of the errors or not will be stored in this field. It is a predefined string "yes" or "no" based on the clicked word and errors existing in the level.

- Time: This field contains how long it takes for the user to click on the word that he just clicked.

- Level: It is important to track the level of each success or failed click, since some of the words may be an error in one level but correct in another level.

### 4.3.5 Survey Model

Players who reach level six, are eligible to complete an optional survey regarding their experience with the game. We will give an explanation of this survey in the evaluation chapter. The Survey model has a foreign key to the User table. We used Likert-typed scaling for our survey
and we store 1 to 5 for strongly disagree to strongly agree. Below are fields covered in the Surveys model.

- **Email**: Email is the foreign key from the Survey table to the Users table. We fill this field using current session.
- **S1**: S1 is the short format of Survey1. Based on the user's answer to the likert scale, we store from 1 to 5 or NULL for nothing in the table.
- **S2**: S2 is the short format of Survey2. It stores the result of the second question in the survey.
- **S3**: S3 is the short format of Survey3. It stores the result of the third question in the survey.
- **S4**: S4 is the short format of Survey4. It stores the result of the fourth question in the survey.
Chapter 5

Evaluation

In this chapter, we demonstrate the results of the Code Reading Dojo evaluation by addressing some of the critical questions in this research. There are two major methods for evaluating Code Reading Dojo including asking participants how they like the game and analyzing collected data to see how users play the game.

5.1 Analyzing Collected Data

In addition to users’ opinion of Code Reading Dojo, we collect every action of a user to examine the impact of Code Reading Dojo and gain additional information. At each level, for all users, each tap on the screen is captured. Based on the problem statement and questions which we are endeavor to answer, we collect the data for all tap actions in the Tap table. The following information is recorded to help us in our findings:

- **Email**: The player's email who clicked on the word
- **Time**: How long the user spends to decide and click on the word
- **Word**: The word which is clicked
- **Success**: Whether the clicked word is successful or not
- **Level**: Which level the user is playing (Some words are errors in a level, while they are correct in another level)

We collect the mentioned data to find answers to our questions. Our problem statement can be categorized into four different areas; levels of the game, information related to errors, data about individual non-errors taps, and user improvements.

5.1.1 Levels

Our primary design goal is deliberate practice. We defined 18 measurable small steps for improving the knowledge of code reading skills. These 18 levels are featured with 3 different stages of difficulty. Although we introduce new types of errors at each level of difficulty most errors are repeated in multiple levels. From the collected data, we analyze the following questions:

**In What Level Did Players Stop Playing?**

Code Reading Dojo's levels have three different stages of difficulty including beginners, intermediates, and advanced. We include six levels for each stage of difficulty. In order to motivate students playing, we had a raffle of twenty Amazon Gift Cards for people who completed until at
least level 6 (beginners’ level). Out of 84 people who registered in the game, 46 of them completed the game until level 6. We believe the Amazon Gift Card was a strong motivation for users to complete until at least level 6. Figure 5.1 illustrates the levels during which players stopped playing Code Reading Dojo until Feb 8th, 2017. 34 out of 84 players stopped playing at level 6, which is a remarkable number. Time is another factor that affects where players stop playing. All data for this game is collected within five days. We anticipate having more people reach higher levels when they have more time to play.

Figure 5.1 In what level did players stop playing?

Which Levels Take More Time per Play?

The level of difficulty in Code Reading Dojo is increasing as the level increases. We introduced new errors in each level of difficulty, and we provide more complex programs at higher levels. These two factors influence the average time spent on each level. Figure 5.2 displays the average duration per each play for all players in different levels. Time is one method to measure the difficulty of each level. Based on the collected data, the average time spent on each level and hence the difficulty is increased by the level.
5.1.2 Errors

One of the main contributions of Code Reading Dojo in computer science education is to collect common mistakes and misunderstandings in Java. We defined different types of Java mutations and injected errors in chapter 4. In this section, based on our collected data, we evaluate the kinds of errors that are harder to find.

Two different methods of measuring difficulty are the success rate for an error in a level and the average time to find an error. The following questions are analyzed and answered in this study:

**Which Errors Missed More the First Time that Players See the Error?**

Three types of mutations are introduced in chapter 4. There are 8 replacement mutations, 5 insertion mutations, and 6 deletion mutations that exist in the game. Some of these 19 mutations are repeated in multiple levels or are present at the same level but more than once. We defined success rate as:

\[
\text{Success rate} = \frac{\text{Number of success taps for all players in a level}}{\text{Total number of times level in played}}
\]

The success rate is one way to examine the difficulty of an error. Errors with lower success rates are harder to find than errors with higher success rates; since more player could not find the error in the total amount of time the level in played. Figure 5.3 illustrates the success rates for 15 errors in the first 12 levels. We analyze the data on the first 12 levels due to the presence of more players at these levels. Fewer people played the advanced level (13-18). The most important conclusion from the figure is that replacement errors have less success rates than insertion and deletion errors. As a result, replacement errors are harder to find than insertion and deletion errors. For 15 types of errors, we only considered the first time that these errors are introduced to the
users. The reason for not including later times is learning factor. We believe after the first time an error is introduced to a user, it will be straightforward for the user to identify the same error.

Did Players Become Better in Finding the Same Type of Error?

In addition to finding the success rate for the first time each error is made, it is important to see the improvement of students in finding the same error. 8 types of errors are repeated multiple times during the first 12 levels of the game. Type Replacement is the most common error among all errors present in the first 12 levels. This error is repeated 11 times in the game. Figure 5.4 displays the improvement in the success rate of same error in different levels. Each color points to a different type of error. The overall conclusion from this figure is that by showing the same type of error in different levels, players' success rates are increasing. The results show that the users are learning the same type of error and are finding the same error easier as they pass through the levels.
Which Errors in Average Take More Time to Find by Players?

Average time to find errors are counted as another way to define the difficulty of errors. Because players spend more time to finding them, these errors are considered more difficult.

We have used mean rather than median since we assumed that we don't have outliers in our data and the distribution of data is a normal distribution. However, in some levels, especially lower levels, there are outliers. These outliers include people who tried to click on the screen and find errors randomly. These outliers played each level many times and created skewed data. In the mentioned levels (mostly lower levels) the median is a better way to illustrate each level with a number.

Figure 5.5 shows the average time to find errors in the first time that an error is introduced. The results in this figure are close to the results shown in figure 5.3. The average time spent to find a replacement error is higher than the average time spent on insertion and deletion errors.
5.1.3 Taps

In order to advance to the next level, each player has to click on words that could result in correctly identifying a error. Words that people click on are captured and analyzed. Table 4 represents top 10 words players clicked on that were not actual errors in the program during five days of the study. Curly bracket insertion is the most wrong clicked word in all levels. One possible reason for clicking on an extra curly bracket is due to the fact that users have seen the same type of error in previous levels, and they have learned that one of the possible errors is an extra curly bracket. A possible reason for users clicking on “}” and “{” is curly bracket insertion mutation. We think that users click on “public” because of misunderstanding of access level which is a types of access level replacement mutation. Most of these errors have already been included in the game, but they are not considered errors in the levels that players clicked on them. There are two common mistakes by players that are not included in the game, but are interesting errors that will be beneficial to add in the next version of Code Reading Dojo.

Table 4 Top 10 Mistakes in All Levels

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>}</td>
<td>144</td>
</tr>
<tr>
<td>{</td>
<td>99</td>
</tr>
<tr>
<td>public</td>
<td>92</td>
</tr>
</tbody>
</table>
Below, we will present the top 5 mistakes at Beginners level along with the code and errors in the program.

### 5.1.3.1 Level 1:

Level one, in the current version of Code Reading Dojo, is a simple two lines of print to console. Below shows the program for level 1. The errors in this program are highlighted as shown below. "}}" with 69 correct taps, ");" with 58 correct taps, and "string[]" with 53 correct taps are errors in this program. Table 5 displays the top 5 mistakes at this level.

```java
/**
 * Print two statements.
 */
public static void main( string[] args ) { 
    System.out.println("Hello, World!");
    System.out.println("This is the first level!");
}
```

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>level&quot;);</td>
<td>17</td>
</tr>
<tr>
<td>System.out.println(&quot;This</td>
<td>16</td>
</tr>
<tr>
<td>static</td>
<td>13</td>
</tr>
<tr>
<td>void</td>
<td>12</td>
</tr>
<tr>
<td>args</td>
<td>11</td>
</tr>
</tbody>
</table>
5.1.3.2 Level 2:
Level two of the Beginners level is a function to check whether or not a number is prime. Below is the program for the second level. The three errors in this program include, "}}" with 50 correct taps, "<= " with 49, and "string" with 46 correct taps. Table 6 presents the top 5 mistakenly clicked words taps in this program.

```java
/**
 * Add two numbers and print the result.
 */
public int adder(int num1, int num2) { 
    System.out.print(num1 ++ "+" + num2 + "=" + (num1 + num2));
}
```

Table 6 Top 5 Mistakes in Level 2

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>40</td>
</tr>
<tr>
<td>System.out.println(num1)</td>
<td>37</td>
</tr>
<tr>
<td>num2)</td>
<td>21</td>
</tr>
<tr>
<td>(num1</td>
<td>20</td>
</tr>
<tr>
<td>}</td>
<td>17</td>
</tr>
<tr>
<td>&quot;=&quot;</td>
<td>17</td>
</tr>
</tbody>
</table>

5.1.3.3 Level 3:
Level three of the Beginners level is an added function which gets an input and increases it by one. The coding below is level three program with three errors highlighted in yellow. Errors in the level are "integer" with 67 correct taps, "String" with 63 correct taps, and "-" with 52 correct taps. Table 7 lists the top five mistakes in this level.

```java
/**
 * Increase an input number by 1.
 */
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int num = scan.nextInt();
    num = adder(num);
    System.out.print(num);
}

static String adder(int num) {
    num = num - 1;
    return num;
}
```
### Level 3:

Table 7 Top 5 Mistakes in Level 3

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>46</td>
</tr>
<tr>
<td>static</td>
<td>33</td>
</tr>
<tr>
<td>Scanner(System.in)</td>
<td>31</td>
</tr>
<tr>
<td>=</td>
<td>23</td>
</tr>
<tr>
<td>System.out.print(num)</td>
<td>22</td>
</tr>
</tbody>
</table>

### Level 4:

Level four in the Beginners level is a function which converts a number grade to letter grade. The coding below illustrates this level’s program along with the errors highlighted in yellow. Injected errors at this level "}" with 50 correct attempts has the most success rate, "<=" with 49 and "string" with 46 correct. Table 8 shows the top 5 mistakes in this level.

```java
/**
 * Convert number grade to letter grade function.
 */
public string findGrade(int points) {
    String grade = "";
    if (points <= 100 && points >= 90 ) {
        grade = "A";
    } else if (points <= 90 && points >= 80 ) {
        grade = "B";
    } else if (points < 80 && points >= 70 ) {
        grade = "C";
    } else if (points < 70 && points >= 60 ) {
        grade = "D";
    } else if (points < 60 && points >= 0 ) {
        grade = "F";
    } else {
        grade = "NOT VALID";
    }
    return ("Your final grade is " + grade);
}
```
<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade</td>
<td>30</td>
</tr>
<tr>
<td>else</td>
<td>19</td>
</tr>
<tr>
<td>}</td>
<td>18</td>
</tr>
<tr>
<td>}</td>
<td>15</td>
</tr>
<tr>
<td>points</td>
<td>14</td>
</tr>
</tbody>
</table>

### 5.1.3.5 Level 5:

Level 5 is a simple class with a constructor and a function to print the firstName and the lastName. Below shows the program. There are three errors in the program, two of them have the same type of error, this deletion. Errors in level 5 "firstName" with 50 correct attempts, "lastName" with 48 right taps and "}" with 46 correct taps. Table 9 illustrates the top 5 wrong taps in this level.

```java
public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        firstName = firstName;
        lastName = lastName;
    }

    public String fullName() {
        return firstName + " " + lastName;
    }
}
```

Table 9 Top 5 Mistakes in Level 5

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastName;</td>
<td>18</td>
</tr>
<tr>
<td>String</td>
<td>18</td>
</tr>
<tr>
<td>public</td>
<td>16</td>
</tr>
<tr>
<td>Person(String</td>
<td>12</td>
</tr>
</tbody>
</table>
5.1.3.6 Level 6:
Level 6 is a FizzBuzz game, which is a program that prints numbers from 1 to 100, for multiples of five print “Fizz”, for multiples of seven print “Buzz”, for numbers which are multiples of both five and seven prints “FizzBuzz”. Errors in this program are “!=” with 62 correct taps, “*” with 61, and “Args” with 44 correct taps. The coding below is the program and Table 10 shows the top 5 mistakes in the program.

```java
public class FizzBuzz {
    /** A program that prints the numbers from 1 to 100.
     * For multiples of five print “Fizz”,
     * for multiples of seven print “Buzz”,
     * for numbers which are multiples of both five and seven print “FizzBuzz”.
     */
    public static void main(String[] Args) {
        for (int i = 1; i <= 100; i++) {
            if (((i % 5) != 0) && ((i % 7) == 0)) {
                System.out.println("fizzbuzz");
            } else if ((i * 5) == 0) {
                System.out.println("fizz");
            } else if ((i % 7) == 0) {
                System.out.println("buzz");
            } else {
                System.out.println(i);
            }
        }
    }
}
```

Table 10 Top 5 Mistakes in Level 6

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>}</td>
<td>77</td>
</tr>
<tr>
<td>{</td>
<td>28</td>
</tr>
<tr>
<td>==</td>
<td>27</td>
</tr>
<tr>
<td>%</td>
<td>21</td>
</tr>
<tr>
<td>else</td>
<td>19</td>
</tr>
</tbody>
</table>
5.1.4 Improvement

We are interested in analyzing players' performance in playing with Code Reading Dojo. A formula of success rate for passing each level is defined as follow:

$$\text{Success rate for each level} = \frac{\text{Total number of success taps for a user}}{\text{Total taps for a user}}$$

The mentioned success rate for each level shows the rate of success taps for a user which can be considered as a way of measuring the performance for them. Figure 5.6 illustrates the final graph which show a different trend for each user. Our theory is that different people have a different way of learning and thus a different success rate. Additionally, the level of difficulty for each level, as well as the length and complexity of each level is increases in higher levels. Therefore, we cannot expect users to make fewer mistakes during the game. At each level, the user may see a new set of errors which leads to the success rate becoming lower than the previous level.

![Figure 5.6 Success Rate per level for all players who completed at least level 6](image)

5.2 Survey Results

The surveys, which are completed after playing some of the levels, are the best ways to evaluate how Code Reading Dojo has succeeded in reaching its final goals. Our primary target is to improve players’ knowledge of code reading skills. Therefore, we ask Code Reading Dojo players about their experience with the game inside the application.

Code Reading Dojo is publicly available on Google Play and Apple Store. An email regarding participation in the Code Reading Dojo study was sent out to the undergraduate students at the Department of Computer Science at Virginia Tech. After five days, 84 people registered in
the game. 46 people have completed at least level six and filled out a survey after finishing the level. The questions included in the survey are as follows:

- Using Code Reading Dojo improves my code reading skills in Java
- Using Code Reading Dojo improves my code writing skills in Java
- Using Code Reading Dojo improves my ability to find errors in my program
- Using Code Reading Dojo improves my overall programming ability

The questions are in five Likert scales, from strongly disagree to strongly agree. The first two questions are considered to be more detailed questions, while the other two are mostly general questions about the knowledge of programming languages. The survey is available for users after completing level six. We place the survey after players have finished one third of the program to ensure that people have had enough chances to play the game so that they can evaluate it. All four questions are optional. By Feb 8th, 2017, we had collected the data from 46 people. The remaining subsections are describing the results of the survey.

5.2.1 Survey 1 - Improving code reading skills

The first question in the survey is focused on the main goal of the application. We are interested in players’ opinion on whether Code Reading Dojo helps them improve their knowledge of code reading skills. Figure 5.7 depicts the results of the question. The results illustrate that most people agree on the impact that Code Reading Dojo has on code reading skills.

![Figure 5.7 Survey 1- Improving code reading skills](image)

5.2.2 Survey 2 - Improving code writing skills

Programming is an integral part of computer science comprised of writing and reading code. In this study, we are interested in users' opinion on the impact of Code Reading Dojo in
writing code as well. After the survey, we realized that the results in this question are less positive than code reading skills improvement. However, there are many players who agree that Code Reading Dojo improves their knowledge of code writing skills. Figure 5.8 shows the survey results for this question.

![Survey Results](image.png)

**Figure 5.8 Survey 2 - Improving code writing skills**

### 5.2.3 Survey 3 - Assisting in error-findings skills

The strategy of Code Reading Dojo is to find bugs in pieces of codes. We intentionally use solid pieces of code in our game using modern reference books and Virginia Tech lecture notes. Finding bugs is considered routine work for programmers. We are eager to know players' opinion on whether Code Reading Dojo helps them find bugs in their own programs. Our results conclude that most people agree on the usefulness of Code Reading Dojo to help them find errors in actual programming. Figure 5.9 shows the results for this question.
5.2.4. Survey 4 - Improving overall programming knowledge

The last question in the survey is a general question about knowledge of programming languages. Code Reading Dojo is an educationally-oriented computer science game. We ask players whether they believe this game has been useful for improving programming knowledge. The results for this question are positive, and we conclude that most people agree that Code Reading Dojo improves their knowledge of programming languages. Figure 5.10 displays the results for this question.
Chapter 6
Conclusion and Future Work

Code Reading Dojo is an educationally-oriented mobile application aimed at promoting the knowledge of code reading skills. Code Reading Dojo's main design goals include deliberate practice and competitive practice and continuous engagement. The goals are achieved through accessibility, learnability and enjoyment. We focused on our primary goals during the implementation of Code Reading Dojo. Code Reading Dojo includes six levels, each in three different levels of difficulty. In the game, we covered a wide variety of common mistakes and misunderstanding of Java features.

Code Reading Dojo mobile application has been developed and it is available for the public on Google Play and the Apple Store. 86 people signed up for the game by Feb 8th, 2017. We evaluated and analyzed collected data from students and asked their opinion about the game using a survey inside the game. In this chapter, we explain the contribution of this work and future improvement to the game.

6.1 Contribution to CS Education

Our main contribution in this work is the Code Reading Dojo system. Unlike other educational games in computer science, we did not just build another game in app stores; our goal was to analyze the data we collected from students. Some of our contributions in this field are listed below:

- We provide 18 categories of errors in three different types. These three types of errors are replacement mutation, insertion mutation, and deletion mutation.
- We provide a list of the hardest types of errors based on the time that students spent to find them. In addition, we also present a list of the most missed errors in the programs. According to our theory, the hardest types of errors are errors which take more time to be identified and errors that are missed more often by participants.
- We present improvement for the same types of errors in different levels. Our results illustrate players can detect the same types of errors more easily, if they have seen the errors previously.
- We provide an analysis of students’ performance. There is no unique trend shown for all players. However, we believe the reason is because the complexity of levels increases during playing, and at each level new errors are introduced to the players, hence we don’t see success taps improvement for all players.
Based on the survey results, students believe Code Reading Dojo improves knowledge of code reading skills in Java and overall programming languages for students. We find that there is a strong relationship between code reading skills and overall programming knowledge.

6.2 Future Work

In our experiment, we added our results from our Java Tool manually. This limits the program to have the same problem during different attempts by the same player. Our Java Tool can be more beneficial when it is able to generate faulty codes dynamically with unlimited positions for errors. This functionality will make the game more enjoyable and challenging.

Not every user of Code Reading Dojo has the same background. One possible extension to Code Reading Dojo is to make Code Reading Dojo customizable based on programming experience. Our aim is to have a short questionnaire or a short quiz to understand the user's programming knowledge, and on the questionnaire generate different levels. Another possible customization is to ask players which concepts do they want to focus on more. Based on the user's response, we can inject more errors from the mentioned concepts and better providing challenging problems for the user.

We developed Code Reading Dojo modular, and all data are stored in the server separated from the client-side mobile application. With this matter, it will be easy to expand Code Reading Dojo to other programming languages as well.

The Java Tool for generating faulty errors is extendable. Instructors or other researchers in this field can add their own regular expressions for other programming languages and generate a set of fault programs in their desired programming language.
References


Appendix A

Virginia Tech IRB Approval
MEMORANDUM

DATE: January 24, 2017
TO: Stephen H Edwards, Zahra Ghaed
FROM: Virginia Tech Institutional Review Board (FWA00000572, expires January 29, 2021)

PROTOCOL TITLE: CodeReadingDojo
IRB NUMBER: 16-1128

Effective January 24, 2017, the Virginia Tech Institution Review Board (IRB) Chair, David M Moore, approved the New Application request for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report within 5 business days to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at: http://www.irb.vt.edu/pages/responsibilities.htm

(Please review responsibilities before the commencement of your research.)

PROTOCOL INFORMATION:
Approved As: Exempt, under 45 CFR 46.110 category(ies) 2
Protocol Approval Date: January 24, 2017
Protocol Expiration Date: N/A
Continuing Review Due Date*: N/A

*Date of Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

FEDERALLY FUNDED RESEARCH REQUIREMENTS:
Per federal regulations, 45 CFR 46.103(f), the IRB is required to compare all federally funded grant proposals/work statements to the IRB protocol(s) which cover the human research activities included in the proposal/ work statement before funds are released. Note that this requirement does not apply to Exempt and Interim IRB protocols, or grants for which VT is not the primary awardee.

The table on the following page indicates whether grant proposals are related to this IRB protocol, and which of the listed proposals, if any, have been compared to this IRB protocol, if required.
<table>
<thead>
<tr>
<th>Date*</th>
<th>OSP Number</th>
<th>Sponsor</th>
<th>Grant Comparison Conducted?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Data this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this IRB protocol is to cover any other grant proposals, please contact the IRB office (irbadmin@vt.edu) immediately.
Appendix B

Original Source Code of Code Reading Dojo
//-------------------- Beginner - Level 1 --------------------
/**
 * Print two statements.
 */
public static void main( String[] args) {
    System.out.println("Hello, World!");
    System.out.println("This is the first level!");
}

//-------------------- Beginner - Level 2 --------------------
/**
 * Add two numbers and print the result.
 */
public void adder(int num1, int num2) {
    System.out.print(num1 + " + " + num2 + " = " + (num1 + num2) );
}

//-------------------- Beginner - Level 3 --------------------
/**
 * Increase an input number by 1.
 */
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int num = scan.nextInt();
    num = adder(num);
    System.out.print(num);
}
static int adder(int num) {
    num = num + 1;
    return num;
}

//-------------------- Beginner - Level 4 --------------------
/**
 * Convert number grade to letter grade function.
 */
public String findGrade(int points) {
    String grade = "";
    if (points <= 100 && points >= 90 ) {
        grade = "A";
    } else if (points < 90 && points >= 80 ) {
        grade = "B";
    } else if (points < 80 && points >= 70 ) {
        grade = "C";
    } else if (points < 70 && points >= 60 ) {
        grade = "D";
    } else if (points < 60 && points >= 50 ) {
        grade = "E";
    } else {
        grade = "F";
    }
    return grade;
}
else if (points < 60 && points >= 0 ) {
    grade = "F";
} else {
    grade = "NOT VALID";
}
return ("Your final grade is " + grade);

//------------------- Beginner - Level 5 -------------------
public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String fullName() {
        return firstName + " " + lastName;
    }
}

//------------------- Beginner - Level 6 -------------------
public class FizzBuzz {
    /** A program that prints the numbers from 1 to 100.
     * For multiples of five print “Fizz”,
     * for multiples of seven print “Buzz”,
     * for numbers which are multiples of both five and seven print “FizzBuzz”.
     */
    public static void main(String[] args ) {
        for (int i = 1; i <= 100; i++) {
            if (((i % 5) == 0) && ((i % 7) == 0)) {
                System.out.println("fizzbuzz");
            } else if (((i % 5) == 0)) {
                System.out.println("fizz");
            } else if (((i % 7) == 0)) {
                System.out.println("buzz");
            } else {
                System.out.println(i);
            }
        }
    }
}

//------------------- Intermediate - Level 1 -------------------
/**
 * Calculate factorial of a number.
 */

58
public int fact(int num) {
    if (num <= 1) {
        return 1;
    }
    int temp = fact(num - 1);
    return temp * num;
}

public class Game {
    private Dice dice;

    public void correctGame(Dice dice) {
        this.dice = dice;
    }

    public void play() {
        dice.roll();
        int cnt = 1;
        while (dice.isSix()) {
            dice.roll();
            cnt += 1;
        }
        if (cnt > 10) {
            System.out.println("You win!");
        }
    }
}

class Dice {
    private int face;

    public Dice() {
        roll();
    }

    public void roll() {
        face = (int)((Math.random() * 6) + 1);
    }

    public boolean isSix() {
        return (face == 6);
    }
}

public class Game {
    private Dice dice;

    //-------------------- Intermediate - Level 2 -------------------
    //-------------------- Intermediate - Level 3 -------------------
}
public void correctGame(Dice dice) {
    this.dice = dice;
}

public void play() {
    dice.roll();
    int cnt = 1;
    while (dice.isSix()) {
        dice.roll();
        cnt += 1;
    }
    if (cnt > 10) {
        System.out.println("You win!");
    }
}

class Dice {
    private int face;

    public Dice() {
        roll();
    }

    public void roll() {
        face = (int)((Math.random() * 6) + 1);
    }

    public boolean isSix() {
        return (face == 6);
    }
}

public boolean isPrime(int num) {
    if (num < 2) {
        return false;
    }
    int index;
    for (index = 2; index < num; index++) {
        if (num % index == 0) {
            return false;
        }
    }
    return (index == num);
}

//------------------- Intermediate - Level 4 -------------------

/**
 * Function to check if a number is prime.
 */
public boolean isPrime(int num) {
    if (num < 2) {
        return false;
    }
    int index;
    for (index = 2; index < num; index++) {
        if (num % index == 0) {
            return false;
        }
    }
    return (index == num);
}

//------------------- Intermediate - Level 5 -------------------
public class BookDatabase {
  private ArrayList<Book> books;

  public BookDatabase() {
    books = new ArrayList<Book>();
  }

  public void addItem(Book theBook) {
    books.add(theBook);
  }

  /**
   * Print all books in the database.
   */
  public void bookList() {
    for (Book book : books) {
      System.out.println(book.title);
    }
  }
}

class Book {
  public String title;
  private boolean read;

  public Book(String theTitle) {
    title = theTitle;
  }
}

//------------------- Intermediate - Level 6 -------------------
public class CardGame {
  private ArrayList<Card> deck = new ArrayList<Card>();
  private String name;

  /**
   * Function to initialize a card deck.
   */
  public void initializeDeck() {
    String[] ranks = "2", "3", "4", "5", "6", "7", "8", "9", "10",
          "Jack", "Queen", "King", "Ace";
    String[] suits = "Clubs", "Diamonds", "Hearts", "Spades";
    for (String rank : ranks) {
      for (String suit : suits) {
        deck.add(new Card(rank, suit));
      }
    }
  }
}

//--Advanced -- Level 1--

public class Minimum {
    /**
     * Print minimum of three numbers.
     */
    public static void main(String[] args) {
        int num1 = 2;
        int num2 = 3;
        int num3 = 4;
        System.out.print(min(num1,num2,num3));
    }

    private static int min(int i, int j) {
        if ( i < j ) {
            return i;
        } else {
            return j;
        }
    }

    private static int min(int i, int j, int k) {
        if ( i < j ) {
            return min(i,k);
        } else {
            return min(j,k);
        }
    }
}

//----------------------------- Advanced - Level 2 -----------------------------
public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String toString() {
        return firstName + " " + lastName;
    }
}

class Student extends Person {
    private int grade;

    public Student(String firstName, String lastName, int grade) {
        super(firstName, lastName);
        this.grade = grade;
    }
}
public String toString() {
    String result = super.toString();
    result += " with grade " + grade;
    return result;
}

//------------------- Advanced - Level 3 -------------------
interface Student {
    public void study();
    public void sleep();
}

public class UnderGraduateStudent implements Student {
    public void study() {
        System.out.println("Study time!");
    }
    public void sleep() {
        System.out.println("Sleep time!"ements);
    }
    public void party() {
        System.out.println("Party time!");
    }

    /**
     * Create an instance of undergraduate student
     * Call study and party methods.
     */
    public static void main(String[] args) {
        UnderGraduateStudent undergrad = new UnderGraduateStudent();
        undergrad.study();
        undergrad.party();
    }
}

//------------------- Advanced - Level 4 -------------------
public class VendingMachine {
    private static int balance;
    private static int[] Price = new int[6];
    private static String[] Description = new String[6];

    public static void setPrice(int itemNumber, int thePrice) {
        Price[itemNumber] = thePrice;
    }

    public static int getPrice(int itemNumber) {
        return Price[itemNumber];
    }
}
public static void setDescription(int itemNumber, String theDescription) {
    Description[itemNumber] = theDescription;
}

public static String getDescription(int itemNumber) {
    return Description[itemNumber];
}

public VendingMachine(int myBalance) {
    balance = myBalance;
}

public static boolean buyItem(int itemNumber) {
    if (balance < getPrice(itemNumber)) {
        System.out.println("Not enough money!");
        return false;
    } else {
        balance -= getPrice(itemNumber);
        System.out.println("Bought " + getDescription(itemNumber) + " for " + getPrice(itemNumber));
        return true;
    }
}

public static void addMoney(int amount) {
    balance = balance + amount;
}

// ------------------- Advanced - Level 5 -------------------
public class PublicationItem {
    private String title;

    public PublicationItem(String theTitle) {
        title = theTitle;
    }
}

class Book extends PublicationItem {
    private String author;

    public Book(String theTitle, String theAuthor) {
        super(theTitle);
        author = theAuthor;
    }
}

class Newspaper extends PublicationItem {
    private String frequency;
public Newspaper(String theTitle, String theFrequency) {
    super(theTitle);
    frequency = theFrequency;
}

class PublicationDatabase {
    private ArrayList<PublicationItem> items;

    public PublicationDatabase() {
        items = new ArrayList<PublicationItem>();
    }

    public void addItem(PublicationItem theItem) {
        items.add(theItem);
    }

    //--------------------- Advanced - Level 6 ---------------------
    public class Poll {
        public static final int MAX_CHOICES = 10;
        private int choiceNumber;
        private int[] choices;

        public Poll(int pollNumber) {
            this.choiceNumber = pollNumber;
            choices = new int[MAX_CHOICES];
        }

        public void voteFor(int choiceNumber) {
            choices[choiceNumber]++;
        }

        public int getFirstChoice() {
            return choices[0];
        }

        /**
         * Calculate vote percents for all choices.
         */
        public double[] getVotePercents() {
            int totalVotes = 0;
            for (int c : choices) {
                totalVotes += c;
            }

            double[] votePercents = new double[choices.length];
            if (totalVotes == 0) {
                return votePercents;
            }
        }
    }
}
for (int i = 0; i < votePercents.length; i++) {
    votePercents[i] = 100.0 * choices[i] / totalVotes;
}
return votePercents;