CMPT 276
FINAL REPORT

Group “golf”

PREPARED BY

Hieu Duong

Mrinal Goshalia

Lucy Wang

PREPARED ON

Apr 11, 2023

1. Introduction

In today's world, learning a new language has become an essential skill for personal and professional growth, leading to a rising demand for language learning tools. Throughout this semester the three members of group "golf" have used the agile development methodologies to collectively develop an interactive "Sudoku Language Practice App" using Android Studio and incorporating a Room Database.

The app aims to aid teachers as a language learning tool and helps students/users interactively learn and practice new languages by playing a modified, language-learning version of the traditional sudoku game. Currently, the game offers standard and listening comprehension modes in: Chinese (simplified), English, French, Hindi, and Vietnamese. The app's design ensures easy user navigation, provides clear gameplay instructions, and features several customization options to enhance user engagement and experience.

Throughout the project lifecycle, we have planned, designed, programmed, and tested our app at each iteration. This report reflects upon the challenges faced, steps taken to overcome them, as well as future challenges and objectives to address if the app is further developed. The following four sections will detail the requirements set for users/user-stories, app design and implementation, GitLab usage, testing and lessons learned throughout the agile development process.

2. Requirements

Throughout the four stages of the project lifecycle, clear and detailed app requirements established via user stories and TDD examples were essential. This process helped us create a clear, collective app vision, ensuring that all group members were working towards the same end-goals.

In the first iteration, we identified and categorized our user demographic into four main groups: novice users, expert users, language learners, and language teachers. User stories expanded each category, detailing what each user should be able to do while TDD examples provided specific steps to accomplish these objectives. We utilized Figma to design multiple mock-ups of our user interfaces while integrating user requirements into them. Although the process initially seemed tedious, setting user stories, TDD examples, and visualizing them through Figma simplified the implementation process and created a clear app vision.

In subsequent iterations, as we began programming and implementing requirements, we encountered unforeseen challenges. For example, we realized that to increase app engagement we needed to provide personalized features such as changing grid sizes,

listening comprehension mode, etc. Hence, we continued to update, add to, and refine our user stories and TDD examples. Given the timelines and new challenges encountered at each iteration, we adapted our approach by prioritizing the implementation of the most important requirements and incorporating new requirements while addressing previously set ones. Although we could not implement all set requirements, having a list of future requirements provided a solid foundation for further development.

Continuously reviewing and refining requirements kept us organized and prioritized our app's functionality. Moving forward, we recognize the importance of timelines and will factor them in when creating requirements, estimating when each requirement should be implemented to ensure realistic expectations for their achievement. Overall, focusing on user stories and TDD examples simplified the implementation process and created a clear, app vision.

3. Design and Implementation

Our team of three tackled the challenge of building this Android game from scratch, using a variety of tools and technologies to meet the project requirements. We started by creating a design mockup and prototype using Figma for the first iteration, which helped us visualize the app's look and feel. Throughout the project, we aligned our UI design with the mockup to ensure consistency.

As we progressed, we realized the importance of scalable solutions in adapting to changing requirements. Since we lacked prior experience with Android development and only a basic understanding of Java, the second iteration was the most challenging. We had to learn these skills on the fly to create a functional app while also avoiding technical debt that could hinder our ability to adapt to new requirements in the future.

One crucial lesson we learned was that selecting the appropriate tool was more important than piecing together a solution. For instance, we initially used a horizontal scroll view to display word buttons, but we found that it did not provide an optimal user experience. Later, we discovered the Flow widget in ConstraintLayout, which allowed the buttons to wrap around to the next line, resulting in a much-improved user interface.

Regarding architecture, we initially placed all controller logic inside the Activity/Fragment classes, but changing requirements reminded us of the need for separation of concerns. Refactoring with the Model-View-Controller (MVC) pattern allowed us to achieve that, resulting in a cleaner codebase and improved maintainability.

Lastly, we replaced all the hardcoded game data with a Room database, which was the most extensive refactoring we did. This made our code cleaner and allowed us to focus on designing the game logic, enabling us to add new features quickly. However, the rapid

addition of new code presented a challenge to maintainability, as our Model classes became larger and more coupled than before, which could limit the app's extensibility if development continued.

4. GitLab Usage

Our team learned a lot from using Git and GitLab for collaborative development. They offered useful features such as version control and issue tracking, which helped us organize and manage our work effectively. However, we also encountered some obstacles that hindered our progress.

We enjoyed making small and organized Git commits, which allowed us to quickly identify changes and their purposes. We also leveraged Git branches to work on our features independently and GitLab's merge request feature to review and improve each other's work.

However, we struggled with resolving merge conflicts that happened when we worked on the same files or issues. This required careful collaboration and regression testing to ensure quality. We also had trouble working on multiple issues at once when they were large and ambiguous or dependent on others.

Despite these challenges, we learned valuable lessons about proper planning and communication when using GitLab. We also realized that designing for maintainability and scalability from the beginning would have been critical to the project's success, and we should have invested more in this process before starting implementation.

5. Testing

During our development process, testing played a crucial role in ensuring that our codes and features were functional and that the old codes continued to work properly. Although we struggled with black box testing, we found success with regression testing to prevent the new codes from breaking the old ones. Test-driven development (TDD) was difficult for us initially due to a lack of clear design, but once we finalized classes and functions, white box testing became easier. We learned that achieving 100% coverage was not the ultimate goal of testing, but rather a tool to help us design better tests.

We then turned to manual testing and verified that all the TDD examples we wrote in the first iteration were properly implemented and tested. Although the manual tests were not comprehensive, we were able to replace them with instrumented tests to run automatically on Android devices since we were already working with UiAutomator. The Android tests

were a great success even though we could no longer rely on test coverage to guide our testing. We found that the Android tests could be used for unit testing as well, but running those tests required an Android device and were more time-consuming than the unit tests. We factored out tests to handle only certain sets of UI components, allowing us to quickly update them when our underlying UI implementation changed.

We initially tested on Android exhaustively, but random testing saved time. However, they were not reproducible and missed bugs. We solved this by setting a fixed random seed for tests, making them consistent and reliable. Overall, we learned that testing demands flexibility, adaptability, and learning from mistakes.

6. Conclusion

In conclusion, our team used agile development methodologies to successfully develop an interactive Sudoku Language Practice App, for an engaging language learning experience.

We simplified the implementation process and created a clear app vision by focusing on requirements through user stories and TDD examples. Additionally, we gained valuable software engineering skills such as utilizing design patterns, selecting appropriate tools, and refactoring code to ensure our app is scalable and adaptable to changing requirements. Collaborating on GitLab, we learned to delegate tasks using issues and to resolve merge conflicts. We also used regression and instrumented testing with UiAutomator and unit testing to resolve bugs and maintain functional codes and features.

Although we could not implement all the set requirements, having a list of future requirements provides a solid foundation for further development. Moving forward, we want to continue to develop more of our requirements, such as implementing a note taking mode, giving game hints, creating a tutorial, and other features to enhance user engagement.

Overall, developing the app has been an insightful and fulfilling experience. We are excited to continue developing the app to help teachers and users in their language learning journeys.