Software Design Specification

for

CMD TypeSwift Typing Tutor

Prepared by

Athul Babu K – NIE22CS019

Sobin Seby – NIE22CS047

Sojo Jenson – NIE22CS048

DATE: 27-02-25

Table of Contents

1. Abstract of the Project

CMD TypeSwift is a programmer-oriented typing tutor designed to help users improve their typing skills while simultaneously learning programming concepts. The application is written in C and is compatible with various terminal environments such as GNOME Terminal, xterm, and uxterm. CMD TypeSwift offers a range of lessons tailored for both programmers and non-programmers, providing real-time feedback on typing accuracy and speed. The tool is designed to be fast, efficient, and user-friendly, with features such as error tracking, speed analysis, and customizable lessons.

The primary goal of CMD TypeSwift is to provide a lightweight, terminal-based typing tutor that can be used by programmers to improve their typing speed and accuracy while practicing coding. The application is particularly useful for individuals who spend a significant amount of time working in terminal environments, as it allows them to practice typing in a context that closely resembles their daily workflow. The application is designed to be customizable, with users able to create their own lessons and modify existing ones to suit their specific needs.

2. Proposed Tools and Brief Description About Their Selection

2.1 Tools:

- GCC Compiler: The GNU Compiler Collection (GCC) is used for compiling the C code. GCC is a widely-used, open-source compiler that supports multiple platforms and is highly optimized for performance. It is the standard compiler for most Unix-like operating systems, making it an ideal choice for this project.

- Git: For version control and repository management. Git is the industry standard for version control, allowing for efficient collaboration and code management. It is used to manage the source code and track changes over time.

- GNOME Terminal, xterm, uxterm: These terminal emulators are chosen for their compatibility and widespread use in Unix-like operating systems. CMD TypeSwift is designed to work seamlessly with these terminals, ensuring a consistent user experience across different environments.

- Bash Scripting: Used for installation and uninstallation scripts. Bash is a powerful scripting language that is native to Unix-like systems, making it an ideal choice for automating the installation and uninstallation processes.

2.2 Selection Rationale:

- GCC was chosen for its robustness and compatibility with C programming. It is widely supported and offers excellent performance, making it the best choice for compiling the application.

- Git is the industry standard for version control, making it ideal for collaborative development. It allows multiple developers to work on the project simultaneously and track changes efficiently.

- The terminal emulators (GNOME Terminal, xterm, uxterm) were selected for their widespread adoption and compatibility with the application's design. These terminals are commonly used by programmers, ensuring that CMD TypeSwift will be accessible to its target audience.

- Bash scripting is used for its simplicity and effectiveness in automating installation and uninstallation processes. Bash scripts are easy to write and maintain, making them a practical choice for this project.

3. Introduction About the Proposed Project

CMD TypeSwift is designed to be a lightweight, efficient typing tutor that caters specifically to programmers. The application provides a set of lessons that help users improve their typing speed and accuracy while simultaneously learning programming concepts. The tool is terminal-based, making it accessible to users who prefer command-line interfaces. CMD TypeSwift offers real-time feedback, error tracking, and customizable lessons, making it a versatile tool for both beginners and experienced typists.

The application is particularly useful for programmers who spend a significant amount of time working in terminal environments. By providing lessons that focus on programming syntax and common coding patterns, CMD TypeSwift helps users become more proficient typists while also reinforcing their programming skills. The application is designed to be highly customizable, with users able to create their own lessons and modify existing ones to suit their specific needs.

4. System Overview

4.1 Functionality:

- Typing Lessons: Provides a set of pre-defined lessons for both programmers and non-programmers. The lessons are designed to help users improve their typing speed and accuracy while learning programming concepts.

- Real-Time Feedback: Displays correct and incorrect characters in green and red, respectively. This provides immediate visual feedback to the user, helping them identify and correct errors as they type.

- Speed Tracking: Tracks typing speed in characters per minute (CPM) and words per minute (WPM). The application calculates both accurate and approximate speeds, allowing users to track their progress over time.

- Error Tracking: Keeps a record of the number of errors made during each session. The application tracks both correct and incorrect characters, providing users with detailed feedback on their performance.

- Customizable Lessons: Allows users to create and use their own lessons via a `myown.txt` file. This feature enables users to tailor the application to their specific needs, making it a versatile tool for both beginners and experienced typists.

- Session Logging: Logs session details such as date, time, errors, and speed for future reference. The application creates a text file to store this information temporarily, allowing users to track their progress.

4.2 Design Considerations:

- The system is designed to be lightweight and efficient, with minimal dependencies. The application is optimized for performance, ensuring that it runs smoothly on a wide range of hardware configurations.

- The application is terminal-based, ensuring compatibility with a wide range of Unix-like systems. This makes it accessible to users who prefer command-line interfaces and ensures that it can be used in a variety of environments.

- The design focuses on user feedback and customization, allowing users to tailor the tool to their specific needs. The application provides a range of features that enable users to create their own lessons, track their progress, and receive real-time feedback on their performance.

5. Design Considerations

5.1 Assumptions and Dependencies:

- Operating System: The application is designed to run on Unix-like operating systems (e.g., Linux, macOS). It assumes that the user has access to a compatible terminal emulator and the necessary tools for compiling and running the application.

- Terminal Emulator: The application assumes the use of a compatible terminal emulator such as GNOME Terminal, xterm, or uxterm. These terminals are widely used in Unix-like environments, ensuring that the application will be accessible to its target audience.

- GCC Compiler: The application requires GCC for compilation. It assumes that the user has access to a compatible version of GCC and the necessary tools for compiling C code.

5.2 General Constraints:

- Hardware Constraints: The application is designed to run on standard hardware with minimal resource requirements. It is optimized for performance, ensuring that it can run smoothly on a wide range of hardware configurations.

- Software Environment: The application is dependent on a Unix-like environment and a compatible terminal emulator. It assumes that the user has access to the necessary tools for compiling and running the application.

- End-User Environment: The application is designed for users familiar with command-line interfaces. It assumes that the user has a basic understanding of how to navigate and use terminal emulators.

6. Development Methods

The development of CMD TypeSwift follows an iterative and incremental approach. The project is divided into several modules, each developed and tested independently before integration. The following methods were used:

6.1 Modular Programming:

The application is divided into modules such as lesson management, error tracking, and user interface. Each module is developed independently, allowing for easier testing and maintenance.

6.2 Agile Development:

The project follows Agile principles, with regular iterations and continuous feedback. This approach allows for flexibility and ensures that the application meets the needs of its users.

6.3 Test-Driven Development (TDD):

Each module is developed with corresponding unit tests to ensure reliability and correctness. This approach helps identify and fix issues early in the development process, reducing the risk of bugs in the final product.

7. System Architecture

7.1 High-Level Overview:

The system is divided into the following components:

7.1.1 User Interface (UI):

Handles user interaction and displays lessons and feedback. The UI is terminal-based, with text displayed in different colours to indicate correct and incorrect characters.

7.1.2 Lesson Manager:

Manages the loading and selection of lessons. The Lesson Manager loads lessons from predefined files or user-defined files (`myown.txt`).

7.1.3 Error Tracker:

Tracks and logs errors made during typing sessions. The Error Tracker monitors user input and logs errors in real-time.

7.1.4 Speed Analyzer:

Calculates and displays typing speed in CPM and WPM. The Speed Analyzer calculates speed based on the number of characters typed and the time taken.

7.1.5 Session Logger:

Logs session details for future reference. The Session Logger records session details such as date, time, errors, and speed.

7.2 Component Interaction:

- The User Interface interacts with the Lesson Manager to load and display lessons.

- The Error Tracker and Speed Analyzer provide real-time feedback to the user via the User Interface.

- The Session Logger records session details at the end of each lesson.

7.3 Diagrams:

Diagrams are enclosed as annexure 1.

8. Detailed System Design

8.1 User Interface (UI):

- The UI is terminal-based, with text displayed in different colors to indicate correct and incorrect characters. The UI provides a command-line interface for user interaction, with commands such as *cmd typswt --help* and *cmd typswt mkuser*.

- The UI is designed to be intuitive and user-friendly, with clear prompts and feedback to guide the user through the application.

8.2 Lesson Manager:

- The Lesson Manager loads lessons from predefined files or user-defined files (`myown.txt`). Lessons can be selected via the command line or through the UI.

- The Lesson Manager is responsible for managing the loading, selection, and display of lessons. It ensures that the correct lesson is displayed to the user based on their selection.

8.3 Error Tracker:

- The Error Tracker monitors user input and logs errors in real-time. Errors are displayed in red, and the total number of errors is displayed at the end of each session.

- The Error Tracker is designed to be accurate and efficient, ensuring that errors are logged correctly and in real-time.

8.4 Speed Analyzer:

- The Speed Analyzer calculates typing speed in CPM and WPM. Speed is calculated based on the number of characters typed and the time taken.

- The Speed Analyzer provides both accurate and approximate speed calculations, allowing users to track their progress over time.

8.5 Session Logger:

- The Session Logger records session details such as date, time, errors, and speed. Session logs are stored in a text file for future reference.

- The Session Logger is designed to be efficient and reliable, ensuring that session details are recorded accurately and stored securely.

9. Interfaces [UI]

9.1 User Interface Screens:

9.1.1 Main Menu:

Displays available commands and options. The Main Menu provides a list of commands that the user can execute, such as creating a new user or selecting a lesson.

9.1.2 Lesson Selection:

Allows users to select a lesson from the available list. The Lesson Selection screen displays a list of available lessons and allows the user to select one.

9.1.3 Typing Session:

Displays the lesson text and provides real-time feedback. The Typing Session screen displays the lesson text and provides feedback on the user's typing accuracy and speed.

9.1.4 Session Summary:

Displays session details such as errors, speed, and time. The Session Summary screen provides a summary of the user's performance during the session.

9.2 Interface models:

Interface models of the software is enclosed as annexure 2.

10. User Roles

10.1 Users:

- New Users: Can create a new user profile and start with basic lessons. New users are guided through the process of creating a profile and selecting their first lesson.

- Experienced Users: Can customize lessons and track their progress over time. Experienced users have access to advanced features such as custom lessons and detailed performance tracking.

- Administrators: Can manage lesson files and system settings. Administrators have access to system-level settings and can modify lesson files as needed.

11. Glossary

- CPM: Characters Per Minute

- WPM: Words Per Minute

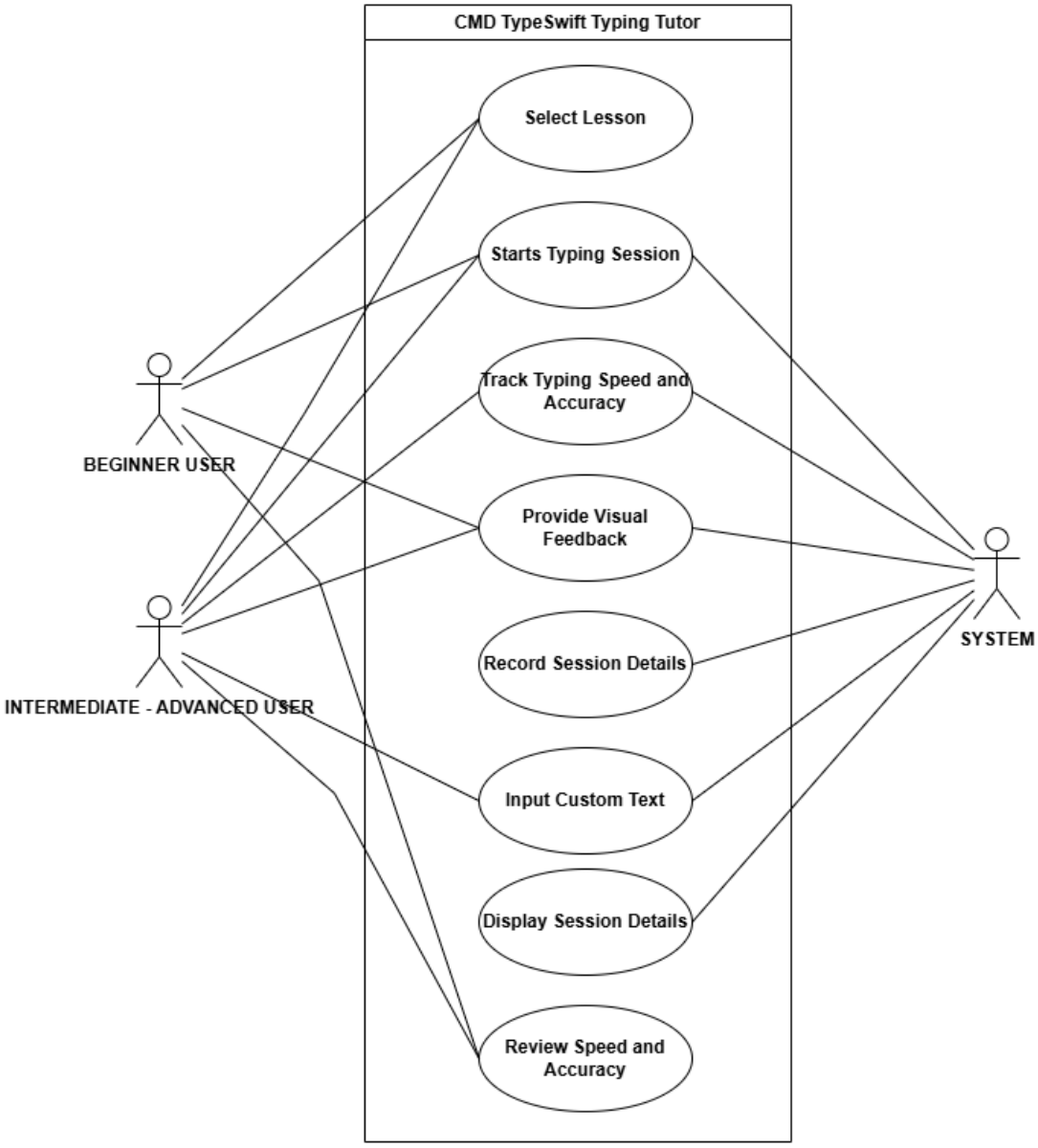
- UI: User Interface

- TDD: Test-Driven Development

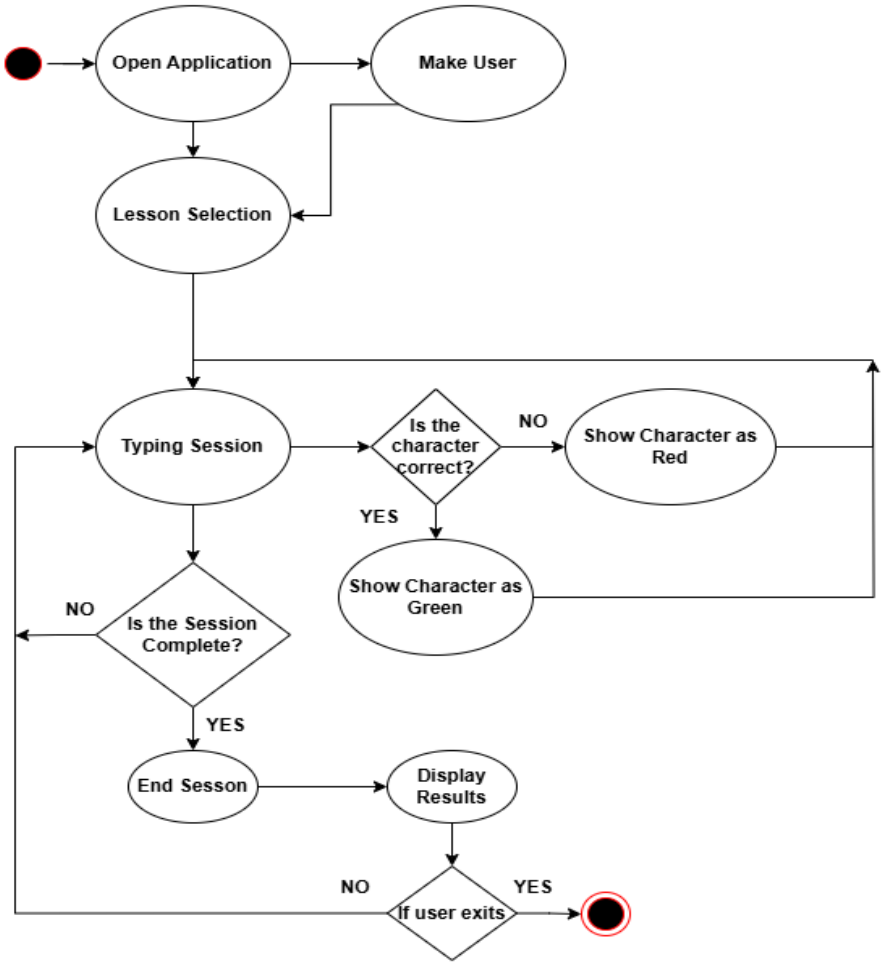
- UML Diagram: Unified Modelling Language Diagram

I. Annexure 1: UML Diagrams

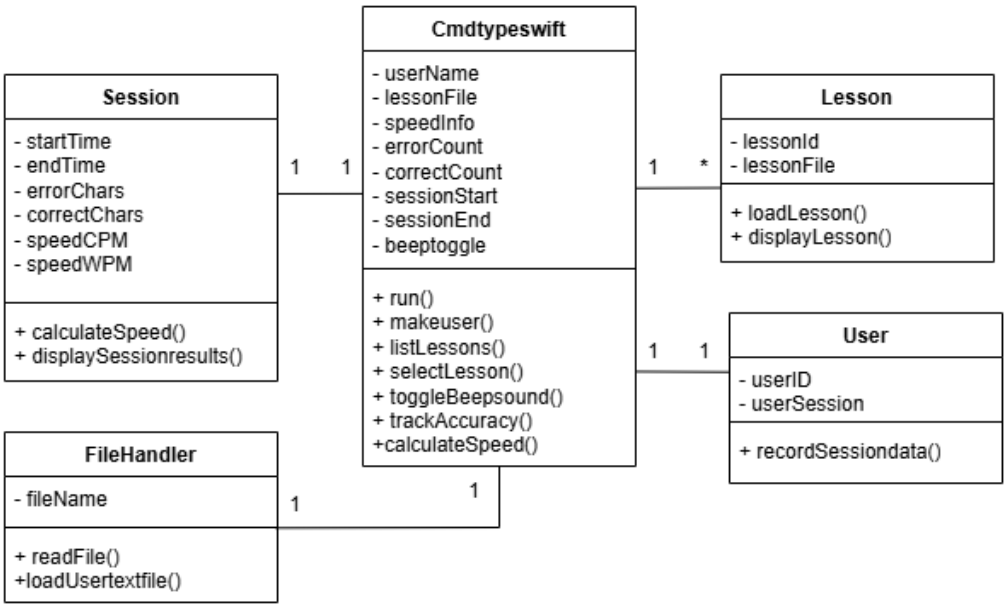
1. Use Case Diagram:



2. Activity Diagram:

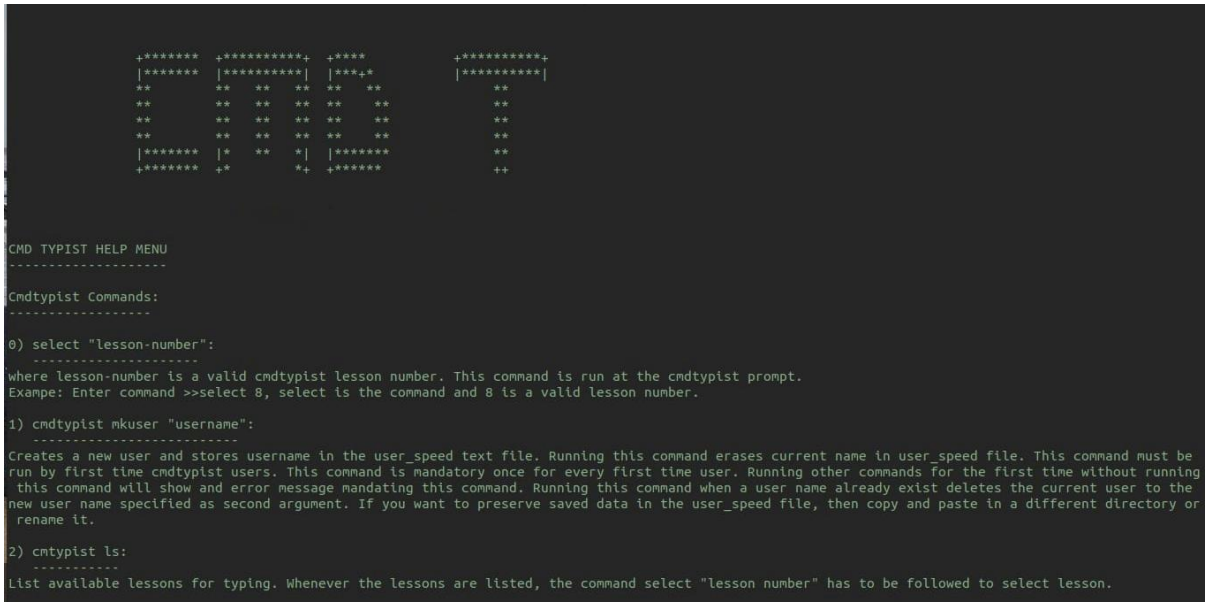


3. Class Diagram:



II. Annexure 2: User Interface Models

1. Start Screen:



2. Lesson selection: