SE 3XA3: Test Plan
ReTouch

Team #7, ReTouchers
Abrar Attia - attiaa1
Susan Fayez - fayezs
Mediha Munim - munimm

December 7, 2017

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
October 27	1.0	PoC, and Unit Tests added
October 27	1.1	Added general information
October 27	1.2	Added System Test Description
December 5	2.0	Updated scope, testing approach, and testing tools
December 5	2.1	Edited test cases for functional requirements)
December 5	2.2	Added links for symbolic parameters
December 6	2.3	Edited test cases for non-functional requirements
December 6	2.4	Updated naming conventions for non-functional tests to match srs doc
December 6	2.5	Added test cases for FREQ16 and FREQ17

This document describes the plan for testing ReTouch.

1 General Information

1.1 Purpose

The purpose of this project is to re-implement the open source project K-Touch. K-Touch is a utility that allows users to track their speed and accuracy in typing, and results in improved typing skills through practice and repetition. The re-implementation will improve upon the original project by making it more user friendly and providing more comprehensive documentation. The purpose of this document is to make the testing for ReTouch as efficient as possible to ensure a robust, complete, and fully-functional program.

1.2 Scope

The scope of ReTouch will be to improve KTouch by making it available on a wider variety of operating systems, while improving much of the functionality and user experience and still providing an intuitive GUI and a user guide. Therefore, testing ReTouch will involve testing the GUI, the user input, file input and output, and the other algorithms.

1.3 Overview of Document

This document outlines several types of tests that the ReTouchers team plans to use, potential testing tools, a testing schedule, and specific test cases that will cover the functional and non-functional requirements of ReTouch.

2 Plan

2.1 Software Description

ReTouch will be implemented very similarly to K-Touch, the original application, in its functionality. The software shall begin by giving the user a list of lessons to choose from, where each lesson consists of a different combination of keyboard characters that the user can practice typing. The main program
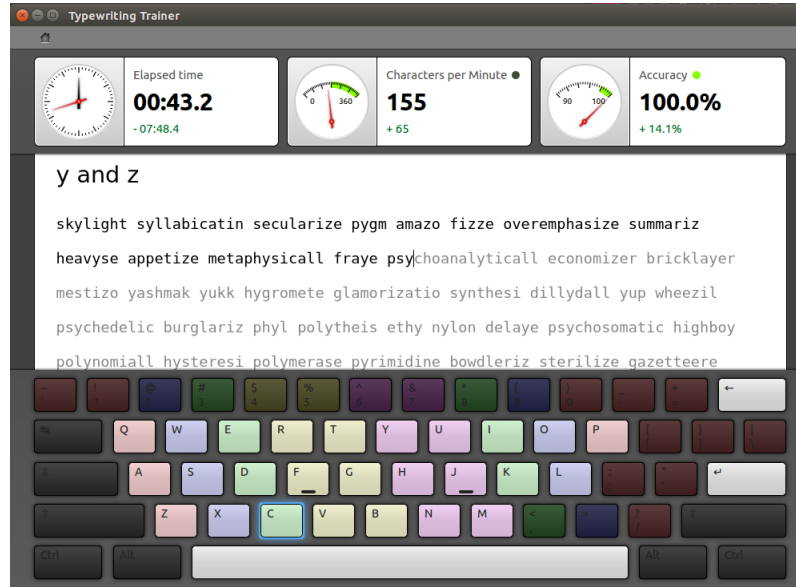
Figure 1: Visual Sample of K-Touch

will begin at that point. A sequence of characters will appear on screen, and the user will be prompted to type those characters on the keyboard (as is seen in Figure 1). The user must accurately type all of them (incorrectly typed characters can be erased and then typed again). The software will calculate the elapsed time, the user's typing accuracy, and the user's typing speed during the program's execution. Once the user has completed the lesson, the results of the lesson will appear on screen. The software will be implemented using Java.

2.2 Test Team

The testing will be divided roughly equally among the ReTouchers team (Abrar Attia, Susan Fayez and Mediha Munim). Ideally, every member will be able to look over all of the components that need to be tested and contribute new cases to ensure robustness of the software, but this may not be possible due to the time constraints on the project. See the Testing Schedule for details.

2.3 Testing Approach

Many different testing techniques will be utilized in the testing process. Black box testing will involve unit testing and integration testing. To automate these tests, the team will take advantage of JUnit's functionalities as well as AssertJ. Private methods will be tested as well, and ideally testing will take place throughout the implementation process to ensure that the tests that passed early on in the process will still pass after finalizing the code. Regression testing will be used should we make any major changes to the implementation. To test the internal framework of the software, white box testing will be used.The user input from the interface will be simulated using AssertJ's functionalities to make sure everything runs as expected. Not all of the testing will be automatic - manual system testing will be vital to the final product because the project is expected to have a detailed GUI. Both static and dynamic test cases will be covered using tools described in Testing Tools. In system testing, the functionalities of KTouch can be compared to ReTouch to ensure that it works just as it was intended to by the original programmers. All in all, the ReTouchers team strives to test as efficiently and effectively as possible.

2.4 Testing Tools

As mentioned earlier, the main tools for testing the Java code will be JUnit and AssertJ. Static testing can take place automatically as the project will be implemented in Eclipse, which has static testing capabilities. JaCoCo is a code coverage tool that works well with Eclipse. However, due to the time constraints associated with the project and the potential learning curve with the library, it may not be used. Finally, Apache JMeter will be used for load testing.

2.5 Testing Schedule

The Gantt Project, which illustrates the project schedule, can be found here or in the ProjectSchedule directory.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

Lesson Selection

1. FREQ1

   Type: Functional, Dynamic, Manual.

   Initial State: Lesson selection (start) screen.

   Input: The desired Lesson.

   Output: User redirected to beginning of Lesson screen.

   How test will be performed: The function that runs the selected Lesson will be run. We will check that the correct Lesson was opened.

2. FREQ16
   Type: Functional, Dynamic, Manual.

   Initial State: The user is in the middle of a Lesson.

   Input: The return to Lesson Selection button is pressed.

   Output: The Lesson the user is working on is terminated and the Lesson Selection page is opened.

   How test will be performed: The function that runs the selected Lesson will be run. A Lesson will be selected and then the return to Lesson Selection page button will be pressed. We will check that Lesson is closed and the Lesson Selection page is opened.

3. FREQ17
   Type: Functional, Dynamic, Manual.

   Initial State: The User has completed a Lesson and is on the Results page.

   Input: The choose another Lesson button is pressed.

   Output: The Results page is terminated and the Lesson Selection page is opened.

How test will be performed: The function that runs the selected Lesson will be run. A Lesson will be selected and the Lesson will be completed. The user will be redirected to the Results page and the select another Lesson button will be pressed. We will check that Results page is closed and the Lesson Selection page is opened.

Lesson Execution

1. FREQ4

   Type: Functional, Dynamic, Manual.

   Initial State: Beginning of the Lesson screen.

   Input: A desired Lesson.

   Output: The Lesson is run, the system sets the current character to the first character in the Lesson.

   How test will be performed: The function that runs the selected Lesson will be run. We will check that the current character value is set to the first character in the given Lesson.

2. FREQ5

   Type: Functional, Dynamic, Manual.

   Initial State: During a Lesson.

   Input: User types any character.

   Output: The system moves the current character value to the next one in the Lesson.

   How test will be performed: The function that runs the Lesson will be run, user input will be provided. We will check the current character value for each time a character is entered to make sure it is being updated properly.

3. FREQ6

   Type: Functional, Dynamic, Manual.

   Initial State: During a Lesson

Input: User types any character in the Lesson.

Output: The system sets that particular character in the Lesson as completed and the cursor moves on to the next character.

How test will be performed: The function that runs the Lesson will be run and user input will be entered. We will check that the system only marked the characters entered correctly as such and moves on to the next character when any character is entered.

4. FREQ8(i)

Type: Functional, Dynamic, Manual.

Initial State: During the Lesson, when the user reaches the end of a line having completed the correct characters.

Input: The user hits the enter key.

Output: If the user can move on to the next line.

How test will be performed: The function that runs the Lesson will be run and the correct user input for the first line will be entered. When the enter button is pressed, we will check if the system gives the proper response, allowing the user to continue on to the next line.

5. FREQ8(ii) Type: Functional, Dynamic, Manual.

Initial State: During the Lesson, when the user reaches the end of a line, having made mistakes.

Input: The user hits the enter key.

Output: The user may not move on the the next line.

How test will be performed: The function that runs the Lesson will be run and user input for the first line will be entered with mistakes. When the enter button is pressed, we will check if the system gives the proper response, preventing the user from moving on to the next line.

6. FREQ14(i)
Type: Functional, Dynamic, Manual.

Initial State: During the Lesson, when the user has finished typing everything correctly.

Output: The Lesson page is closed

How test will be performed: The function that runs the Lesson will be run and the user input will be entered for the entire Lesson correctly. We will check to make sure the Lesson page terminates.

7. FREQ14(ii)

Type: Functional, Dynamic, Manual.

Initial State: During the Lesson, when the user has finished typing everything but has made uncorrected mistakes.

Input: The user hits the enter key.

Output: The Lesson page does not terminate until the user goes back and corrects their mistakes.

How test will be performed: The function that runs the Lesson will be run and the user input will be entered for the entire Lesson with mistakes made on the last line. We will check to make sure the Lesson page does not terminate the Lesson page until after the mistakes are corrected and the user hits enter.

Lesson Mistakes and Corrections

1. FREQ7

   Type: Functional, Dynamic, Manual.

   Initial State: During the Lesson.

   Input: The user enters an incorrect character.

   Output: The system highlights the the character that was entered incorrectly, and if the user continues typing, each character after the initial incorrect character is highlighted as incorrect.

   How test will be performed: The function that runs the Lesson will be run and the user input will be entered incorrectly. We will check that the appropriate characters will be highlighted as incorrect.

2. FREQ9

Type: Functional, Dynamic, Manual.

Initial State: During the Lesson.

Input: The user enters the backspace key.

Output: The current character is moved to the previous character in the Lesson. If the current character is the first in the Lesson and backspace is entered, nothing happens.

How test will be performed: The function that runs the Lesson will be run and user input will be entered. The backspace key will be entered and we will check that the current character has moved appropriately.

3. FREQ10

   Type: Functional, Dynamic, Automatic.

   Initial State: During the Lesson.

   Input: User input for the Lesson, with some mistakes.

   Output: The number of characters entered incorrectly is stored in a variable.

   How test will be performed: The function that runs the Lesson will be run and user input will be simulated, with several mistakes. We will check that the counter variable for the number of mistakes is equal to the actual number of mistakes.

3.1.2 Display

Lesson Generation

1. FREQ2

   Type: Functional, Dynamic, Manual.

   Initial State: Lesson selection screen

   Input: The user selects a Lesson

   Output: The system will generate a Lesson - a list of characters of less than MAX_LESSON characters, including spaces.

   How test will be performed: The function that runs the Lesson will be run and we will check that the Lesson generated is appropriate.

Lesson Display

1. FREQ3

   Type: Structural

   Initial State: The beginning of the Lesson.

   Input: The user works on the Lesson.

   Output: The system displays the list of characters in the Lesson. The list of characters will be presented on separate lines, with each line being no greater than constant MAX_LINE.

   How test will be performed: The function that runs the Lesson will be run and we will check that the Lesson is displayed.

2. FREQ11

   Type: Structural

   Initial State: During the Lesson.

   Input: The user works on the Lesson.

   Output: The system displays the elapsed time, from when the user began the Lesson.

   How test will be performed: The function that runs the Lesson will be run and we will check that the timer is displayed.

3. FREQ12

   Type: Functional, Dynamic, Automatic.

   Initial State: During the Lesson.

   Input: The user works on the Lesson.

   Output: The system displays the user's typing accuracy.

   How test will be performed: The function that runs the Lesson will be run and user input will be simulated with mistakes. We will check that the value for the accuracy is what we expect it to be and displayed as such.

4. FREQ13

Type: Structural

Initial State: During the Lesson

Input: The user works on the Lesson

Output: The system displays the user's typing speed

How test will be performed: The function that runs the Lesson will be run and we will check that the user's typing speed is displayed correctly.

5. FREQ15

Type: Structural

Initial State: The Lesson is completed

Input: The user hits enter after they finish the Lesson

Output: The results (time, typing accuracy, and typing speed) are displayed to the user.

How test will be performed: The function that runs the Lesson will be run and the user input will be entered correctly. When the user has completed the Lesson, we will check that the Results page is opened with the correct results.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. LF3

Type: Structural, Static, Manual

Initial State: Any screen of the application (Lesson Selection, Lesson, Results).

Input/Condition: The user navigates the application.

Output/Result: The GUI's components will be consistent in colour, text fonts, and window properties in order to keep the application organized and easy to navigate.

How test will be performed: We will navigate all three stages of the application to ensure that the aesthetic is consistent throughout.

3.2.2   Usability and Humanity

1. UH1

   Type: Structural, Static, Manual

   Initial State: The Lesson selected and about to begin.

   Input/Condition: The user asked to complete the lesson, with no further guidance.

   Output/Result: The user should be able to complete the lesson.

   How test will be performed: A test group of users unfamiliar with the application will be asked to test it out. They will be asked to rate intuitiveness of the application and whether they could complete the lesson.

2. UH2

   Type: Structural, Static, Manual

   Initial State: The application unopened.

   Input/Condition: The user, asked to complete the process of opening the application, selecting a Lesson, and viewing the results without further guidance.

   Output/Result: The user should be able to easily navigate the three stages of the application.

   How test will be performed: A test group of users unfamiliar with the application will be asked to test it out. They will be asked to rate intuitiveness of the application and whether they were able to visit every page of the application.

3. UH3

   Type: Structural, Static, Manual

   Initial State: The application opened, the Lesson unselected.

   Input/Condition: User asked to try various Lessons of varying difficulty.

Output/Result: The user should find at least one Lesson challenging/helpful in improving their typing skills.

How test will be performed: A test group of users unfamiliar with the application and of varying levels of typing proficiency will be asked to test it out. They will be asked if they found any of the lessons challenging.

4. UH4

Type: Structural, Static, Manual

Initial State: The application not yet installed.

Input/Condition: The user asked to install the application.

Output/Result: The application should be easy for for the user to install.

How test will be performed: A test group of users unfamiliar with the application will be asked to compile and run the program. They will be timed and rate the difficulty of this process.

3.2.3   Performance

1. P1

Type: Structural, Dynamic, Manual

Initial State: During the Lesson.

Input/Condition: The user works on the Lesson.

Output/Result: The system will respond to the user input within 1 second.

How test will be performed: A user will work through a Lesson and we will check the response time as they go. We will also ask them if they experienced any lagging.

2. P2

Type: Structural, Dynamic, Manual

Initial State: During the Lesson.

Input/Condition: The user works on the Lesson.

Output/Result: The timer displayed to the user will be accurate and begin once the lesson loads and will end once the user hits enter on the last line to complete the Lesson.

How test will be performed: A user will work on a Lesson and we will check that the timer is accurate.

3. P5

Type: Structural, Static, Manual

Initial State: Any point in using the application.

Input/Condition: The user uses the application.

Output/Result: The application will not interfere with the user's machine.

How test will be performed: We will check on the user's machine during and after the application runs to ensure everything is running properly.

3.2.4 Operation and Environment

1. OE1

Type: Structural, Static, Manual

Initial State: Application compiled on a Windows, Mac, and Linux machine.

Input/Condition: User runs the application on both machines.

Output/Result: The application should run smoothly on all machines.

How test will be performed: Run the application on all machines.

2. OE2

Type: Structural, Static, Manual

Initial State: The application is unopened.

Input/Condition: The user opens the application.

Output/Result: The application depends on no other applications to run properly.

How test will be performed: We will open the application by itself, independent of any other applications.

3.2.5 Security

1. S1

Type: Structural, Static, Manual

Initial State: During a Lesson.

Input/Condition: The user completes the Lesson.

Output/Result: The system will not store any information on the user's results.

How test will be performed: When the user completes their lesson, we will check to make sure none of the data is stored.

3.2.6 Cultural

1. C1

Type: Structural, Static, Manual

Initial State: The application is unopened.

Input/Condition: A user will be asked to open and go through the three stages of the application (Lesson Selection, Lesson, and Results).

Output/Result: The user shall not be offended by any of the content within the application.

How test will be performed: A test group of users unfamiliar with the application will be asked to use the application. We will ask them if they were offended by anything in the application.

4 Tests for Proof of Concept

The Proof of Concept testing will focus on validating that all the identified risks in the development of the application can be overcome. The determined

risks that can be tested focus on running the code concurrently. The current program implements a constant timer alongside a main program that requests and waits for user keyboard input.

4.1   Running Programs Concurrently

Constant Timer and User Input Verification

1. FC-1

   Type: Functional, Dynamic, Manual

   Initial State: No input is set.

   Input: User inputs characters.

   Output: Calculated accuracy of user to match the expected characters.

   How test will be performed: The user will be requested to input a certain character as the timer waits for their input. The final characters inputted will be compared to the initial requested characters to determine the user's accuracy. Therefore, half of the program that tests the users keyboard inputs will be tested.

2. FC-2

   Type: Functional, Dynamic, Manual

   Initial State: No input is set.

   Input: User inputs characters.

   Output: The total time taken in comparison to the expected time and the user's speed calculation (the total characters per second).

   How test will be performed: As the user inputs characters that are requested, the timer will be tracked. Once the lesson is run through, the total time the program records will be compared to the expected time. Also, the user's speed calculation will be compared to the expected characters per second to confirm that the timer is running efficiently. Therefore, such a test will confirm that the two programs are running concurrently and both sections are being completed successfully.

5   Comparison to Existing Implementation

There are currently no required tests to be done to compare the application to the existing implementation.

6   Unit Testing Plan

The JUnit unit testing framework as well as AssertJ will be used to implement the unit testing for the ReTouch application.

6.1   Unit testing of internal functions

When testing the internal functions of the application, the unit tests will be based on each method's input parameters and their expected versus derived outputs. Each method will be testing with standard expected inputs as well as exceptions to improve the program's robustness and reduce bugs. Additionally, every class in the program will include all the required import statements. Therefore, the unit tests will not require any stubs or drivers when testing individual methods or components that depend on multiple methods. In addition, coverage metrics will be used to determine how much of the implemented code was tested. The goal to reach is 80% code coverage.

6.2   Unit testing of output files

When testing the output files and program outputs, there will need to be a comparison between the required output and the generated output. However, at this time, there are no expected output files that will be created when the program is run.

7 Appendix

7.1 Symbolic Parameters

- MAX_LESSON: The maximum number of characters per lesson.

- MAX_LINE: The maximum number of characters per line.