SE 3XA3: Test Plan
TouchTime

Team 13, ELM
Matthew Po - 40007877 - pom
Evan Ansell - 1415992 - ansellea
Livia Kelle - 1419709 - kellel

December 7, 2017

Contents

List of Tables

Table 1: **Revision History**

Date	**Version**	**Notes**
October 23	1.0	Initial Draft
October 26	1.0.1	Livia created non-functional tests and Gantt chart
October 26	1.0.2	Matt created functional tests
October 27	1.0.4	Evan created PoC tests and unit tests and parts of section 1
October 27	1.0.5	Livia edited parts of section 1 and 2
October 27	1.0.6	Final revision - everything else by everyone
December 6	1.1	Livia edited test cases to match current implementation, fixed JUnit spelling
December 6	1.1.1	Evan edited unit testing plan to match current implementation
December 6	1.1.1	Evan edited Table 3 to be more readable, added JUnit to testing tools

1 General Information

1.1 Purpose

This document will serve as the basis for the verification and validation of our program through testing. During and after the implementation of our software we will use this plan as a reference for which tests we will perform, how the tests will be performed, and the results that we expect for our tests.

1.2 Scope

Tests of the functional and non-functional requirements in the SRS will be completed in this project and outlined in this document. The aim is to perform a variety of tests, from functional vs structural, static vs dynamic, and automatic vs manual. On a smaller level, unit testing will be completed to test the functionality of individual methods. Utilizing many different testing styles has the advantage of testing different aspects of the application.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
PoC	Proof of Concept
SRS	Software Requirements Specification

Table 3: **Table of Definitions**

Term	**Definition**
TouchTime	Name of the Android smartwatch application.
Functional Test (Black Box)	Analyzes program by probing with various inputs; not concerned with how processing occurs, only results
Structural Test (White Box)	Analyzes program with test data with knowledge of implementation; concerned with how processing occurs
Static Test	Testing that does not involving code execution
Dynamic Test	Testing involving execution of code.
Automatic Test	Tests completed by testing tool such as JUnit
Manual Test	Tests conducted by people, by hand

1.4 Overview of Document

This document will be outlining our plans for testing of the TouchTime software. Described below are our approaches to testing, tools to be used, and plans for system and unit tests. TouchTime is not an indredibly complex software so there is a limited amount of functions that can be tested, however a large portion of our testing will involve non-functional testing to make sure that it meets the requirements and expectations of our users.

2 Plan

2.1 Software Description

TouchTime is an Android smart-watch application that enables users to know the time through vibration patterns, for the visually impaired or anyone that would like to be able to tell the time without having to look at the screen. Users interact with the application through tapping the watch face. Tapping on the left side of the screen will result in the vibration patterns corresponding to the hour; the right side of the screen will indicate the minutes.

2.2 Test Team

The software will be tested by Matthew Po, Evan Ansell, and Livia Kelle.

2.3 Automated Testing Approach

The software will be tested using the JUnit testing framework against an emulator that is running the TouchTime software. Each vibration sent by the code to the watch emulator will be logged into logcat in Android Studio. We can use JUnit to assert that the ~~logs are being sent~~ output of our funtions matches our expected result, which is an implication that the code has successfully executed and the watch is vibrating. Note that the test cases are being used to test against the software telling the watch to vibrate, and not used to test the emulators response (i.e. the physical vibration). We have to assume that the watch's hardware will be functioning correctly.

2.4 Testing Tools

Laptop running Android Studio with an Android Watch Emulator, as well as on an Android Wear 2.0 watch. We will use JUnit for our software-based unit testing.

2.5 Testing Schedule

See Gantt Chart in the Project Schedule folder. The bright green indicates the tasks related to testing.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Vibration Test on Screen Touch

1. test-f1

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

   Input: User touches watch face twice at a given time, on the hour.

   Output: Distinct vibration pattern for current time. Logcat log to indicate the screen has responded to the touch and vibration has occurred.

How test will be performed: Compare Logcat log pattern to pattern that should have occurred at the given hour. Repeat at various hour times, such as 12, 1, 2, 3, 9, 10.

2. test-f2

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

   Input: User touches watch face twice at a given time, at between 1-9 minutes past the hour.

   Output: Distinct vibration pattern for current time. Logcat log to indicate the screen has responded to the touch and vibration has occurred.

   How test will be performed: Compare Logcat log pattern to pattern that should have occurred at the given time. Repeat at various times, such as 1:01, 2:05, 3:09.

3. test-f3

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

   Input: User touches watch face twice at a given time, at a multiple of 10 past the hour.

   Output: Distinct vibration pattern for current time. Logcat log to indicate the screen has responded to the touch and vibration has occurred.

   How test will be performed: Compare Logcat log pattern to pattern that should have occurred at the given time. Repeat at various times, such as 1:10, 2:50, 3:30.

4. test-f3

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

Input: User touches watch face twice at a given time, at a time where the digit in the tens and ones place for the minutes is not 0.

Output: Distinct vibration pattern for current time. Logcat log to indicate the screen has responded to the touch and vibration has occurred.

How test will be performed: Compare Logcat log pattern to pattern that should have occurred at the given time. Repeat at various times, such as 1:11, 2:52, 3:37.

3.1.2 Abnormal Situations

1. test-fa1

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

   Input: User very quickly double taps watch face.

   Output: Distinct vibration pattern for current time. Logcat log to indicate the screen has responded to the touch and vibration has occurred.

   How test will be performed: Depending on how fast the user taps, the watch may not register it as two separate taps. This test will consist of the user tapping the watch twice at smaller and smaller time intervals to see where the cut off is for the watch to only register a single tap.

2. test-fa2

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

   Input: User touches the watch during daylight savings time when the time skips an hour.

   Output: Distinct vibration pattern for current time. Logcat log to indicate the screen has responded to the touch and vibration has occurred.

   How test will be performed: Compare Logcat log pattern to pattern that should have occurred at the given time. The time is received directly from the smartwatch itself, so it should be indicated correctly.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability

1. test-nf-u1

   Type: Functional, Dynamic, Manual

   Initial State: Application installed on Android smartwatch but not launched.

   Input/Condition: Users must tap icon to launch TouchTime.

   Output/Result: Percentage of users capable of launching TouchTime watch face.

   How test will be performed: A group of users will tap the smartwatch application to launch it and the percentage that succeed is recorded. This will ensure that an acceptable amount of users are capable of launching the application.

2. test-nf-u2

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch.

   Input/Condition: Users must tap the watch face twice, with a brief pause in between.

   Output/Result: Percentage of users capable of correctly tapping the watch screen at appropriate times.

   How test will be performed: Users must tap the watch face once to get out of ambient mode. Users must tap watch again to receive vibration feedback. The percentage of users capable of performing these steps is recorded.

3. test-nf-u3

   Type: Functional, Dynamic, Manual

Initial State: Application installed and launched on Android smartwatch. Users have been taught how the vibration patterns correspond to the time.

Input/Condition: Users must tap the watch face twice, with a brief pause in between.

Output/Result: Percentage of users capable of correctly identifying the time based off of the vibrations of the smartwatch.

How test will be performed: Users are taught how to use the application with a 2 minute tutorial by the tester. Users must tap the watch face once to get out of ambient mode. Users must tap watch again to receive vibration feedback. They are allowed to practice this for 1 minute. The users are then asked to perform the functions again and are asked what time was indicated by the watch. The percentage of users that answer this correctly is recorded.

3.2.2 Performance

1. test-nf-p1

   Type: Functional, Dynamic, Manual

   Initial State: Application installed on Android smartwatch but not launched.

   Input/Condition: Users must tap icon to launch TouchTime.

   Output/Result: TouchTime watch face is displayed within 5 seconds.

   How test will be performed: A group of users will tap the smartwatch application to launch it. The time between the application being tapped and the watch face being displayed is recorded and checked if it meets the time requirements.

2. test-nf-p2

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smart watch.

   Input: Users must tap the watch face once.

Output: The watch exits ambient mode.

How test will be performed: The user will tap the watch face. The watch must respond to the user's tap and exit ambient mode within one second.

3. test-nf-p3

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smart watch.

   Input: Users must tap the watch face twice, with a brief pause in between.

   Output: The watch indicates the time through vibration.

   How test will be performed: The user will tap the watch face. The watch must respond to the user's tap and exit ambient mode within one second. The watch must respond to the user's second tap and begin vibrating within one second.

4. test-nf-p4

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smart watch.

   Input: Users must tap the watch face twice, with a brief pause in between.

   Output: The watch indicates the time through vibration.

   How test will be performed: The user must tap the watch face once to get out of ambient mode and must tap watch again to receive vibration feedback. The time indicated must be accurate to the current time within one second in real-time.

3.3 Traceability Between Test Cases and Requirements

The test cases for the functional requirements relating to vibration patterns can be traced directly to SRS 2.4, 2.5, and 2.6, which indicate the vibration

patterns for the hours and minutes. The test cases relating to ambient mode and taps can be traced back to SRS 2.2, 2.3 and 2.6.

For the non-functional requirements, the usability test cases can be traced back to the original usability requirements indicated in the SRS. Tests test-nf-u1 and test-nf-u2 can be traced back to the ease of use requirement 3.2.1, stating that the application shall be usable by anyone capable of touching the screen of a smart watch. The test case test-nf-u3 can be traced back to the ease of learning requirement 3.2.2, stating that the application shall be easy for anyone capable of using a smart watch to use.

In the case of performance, tests test-nf-p1,test-nf-p2, and test-nf-p3 can be traced to the speed requirements 3.3.1. Test-nf-p4 and test-nf-p5 can be traced to the precision requirement 3.3.4, stating the time must be accurate to within one second.

4 Tests for Proof of Concept

For our Proof of Concept, our objectives were to create an Android Wear application that displays a clock face and responds to distinct touches on the left and right side of the face with a unique vibration pattern for each side. Since it is only a basic application of which we are mostly concerned with the functionality, the early proof of concept tests to be performed are simple manual functional tests.

4.1 Vibration Test on Screen Touch

Vibration Pattern

1. test-pc-v1

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch. Smartwatch connected to laptop running Android Studio.

   Input: Tap on left side of face (x $<$ 150).

   Output: Long vibration pattern from watch. Long vibration pattern output to Logcat log.

How test will be performed: We will tap the left side of the watch face on either the emulator or physical android watch. The expected result is a vibration pattern: long vibration, pause, long vibration, pause. We will be able to verify that this was the expected vibration pattern by looking at the Logcat logs in our Android Studio window.

2. test-pc-v2

   Type: Functional, Dynamic, Manual

   Initial State: Application installed and launched on Android smartwatch. Smartwatch connected to laptop running Android Studio.

   Input: Tap on right side of face ($x \geq 150$).

   Output: Short vibration pattern from watch. Short vibration pattern output to Logcat log.

   How test will be performed: We will tap the right side of the watch face on either the emulator or physical android watch. The expected result is a vibration pattern: short vibration, pause, short vibration, pause. We will be able to verify that this was the expected vibration pattern by looking at the Logcat logs in our Android Studio window.

5 Comparison to Existing Implementation

Our TouchTime application will have a different functional implementation than the existing software - indicating time with a vibration pattern rather than by the position of the watch hands. As such, direct functional comparisons will be difficult. Despite the different functionalities, our software has the same goal as the existing software, and the nonfunctional requirements (particularly usability and performance) that effect the user's experience will help us compare between the programs.

6 Unit Testing Plan

The JUnit framework supplied with Android Studio will be used to complete unit testing. We will use JUnit testing in order to quickly and efficiently perform structural tests, particularly on boundary conditions which may be

hard to test manually such as exact times which would be hard to test manually. Since our application accepts only one kind of input - a tap - we decided that it would be unneccessary to do unit testing based on user input and that the input testing could be accounted for in manual tests.

6.1 Unit testing of internal functions

Unit tests for internal functions will be performed by sending predefined input to the functions and comparing the output to the expected result. The internal functions that we will focus on will be the functions responsible for calculating the required number of vibrations according to the time and for creating the vibration pattern that will be output to the user.

7 Appendix

7.1 Usability Survey Questions

Some of our tests will be usability tests. Due to the nature of the application, we will have to provide users with instructions on how to tell the time based on the vibration patterns. These usability tests will be helpful to determine if changes need to be made in order to make our application more user-friendly. Some such questions that would follow system tests with groups of users could be:

- Were you able to launch the application on the smartwatch without issue?
- What was the hour that the watch indicated?
- What was the minute that the watch indicated?
- Was the application responsive enough for your needs?