Time Your Trip: Preliminary Test Plan

Johannes Gallmann
Navjashan Singh
James Zhou
Sirine Trigui
Yixin Zhao

Introduction

This document outlines the testing techniques our group will be using for our project: Time Your Trip. The expected product is an Android Application which will require the user to enter a bus stop number, then choose the bus number as well as the destination stop. After this, our app will use GPS as well as real-time TransLink data to generated an expected arrival time, and will then set an alarm to notify the user.
The App includes a front-end, an embedded SQL database as well as the back-end. It uses TransLink API, Google API and interacts with the GPS signal, as well as the notification function of the device.

Verification Strategy

To be certain that our mobile App meets the user‘s real need, we need to make sure that we cover all the methods of requirement verification. There are four fundamental methods of verification which are somewhat hierarchical in nature, as each verifies requirements of the App with increasing accuracy:

- **Inspection:** is the nondestructive evaluation of our product using at least one of the five senses(visual, tactile,..). It can include simple physical manipulation and measurements. For example, we can visually examine our App for different UIs that were requested and/or check for the fields needed for data entry(Bus stop number in our case) and/or verify that the necessary buttons exist for initiating required functionality (NEXT Button to display the buses at a certain bus stop number).

- **Demonstration:** is the manipulation of the App as it is intended to be used to verify that the outputs are as expected. For example,

when the user enters the Bus Stop Number and clicks on the “NEXT” button (or select one of the provided choices if a ListView is returned) , a specific report is returned with the type of data needed (valid bus numbers or destination stops).

- **Test:** is the verification of our product using a controlled and predefined series of inputs or stimuli to ensure that our App will produce a very specific output as specified by the requirements. For example, when the user enters a predefined bus stop number,say 50043, the ListView that will be returned will contain a specific bus numbers that stop at that bus stop and another ListView containing the destination stops depended on the chosen bus number.

- **Analysis:** is the verification of the App using models,calculations and testing equipment. Such fundamental method allows someone to make predictive statements about the typical performance of the product based on the confirmed test results of a sample set or by combining the outcome of individual tests to conclude something new about the product. Analysis is often used to predict the failure of a product by using nondestructive tests to extrapolate the failure point.For example, complete a series of tests in which a specified number of users input their bus stop number where they are at and initiate the feature that provide them with the bus numbers that they can take at the same time. Then, we need to measure the response of our product to ensure that the bus numbers are returned within the time specified. After doing this, we can analyse the relationship between increasing number of our product users and the time it takes to return the outcome. When the results will be recorded to capture the product degradation, we will use those to predict the maximum allowable time to return the info as defined by the requirements.

As soon as we merge the different parts of our App so that we have a solid relationship between the database, the back-end and the User Interface and just after providing the users with the beta release, then we will be able to get feedback from them.

As long as we provide a description and let the users feel what kind of information our App is going to provide them with, then we will be able to obtain different feedbacks depending on the level of user experience:

- High level: in this case, the artifact that will be used is the UI mock-ups and will be provided to non-technical users

- Low level: in this case, the requirements document and the actual software itself will be used and provided to technical users.

Non-Functional Testing and Results

Non functional requirements deal with the quality attributes of the application and the following areas are a part of consideration

- Performance
- Scalability
- Maintainability
- Reliability

Performance

Performance depends on the factors such as the app's responsiveness and battery efficiency. In order to test the responsiveness of our app, we will be taking multiple usage tests on the app. For improvements in responsiveness, we are following standard coding practices and doing code reviewing at regular intervals so as to remove the redundant code and improve the app experience. Apart from that, battery consumed by an app is also a critical area for the user. In order to tackle such issues, we are taking various measures in how our application will be working so as not consume user's phone's battery continuously and we will keep testing the battery states of the phone while using our app regularly.

Scalability

For scalability of our app, the working of our app is significantly dependent on Translink's data when the user will be using his/her phone to get the real time data from translink. As our app will be creating separate queries for all the users based on their input and will be sending it to Translink's database and will be receiving data from them, so it will be able to handle the scalability side of the app. As we send those queries using an API key generated by Translink for a user which has a restriction of less than

1000 requests a day. Whereas, it is possible to increase this number by requesting Translink. On the other hand, when the user is using the app offline, then our app uses the embedded database to generate the expected timings. In this case, as every user installs the database on their phone when they install the app, it will increase the scalability of the app.

Maintainability

Maintainability is one of the most important aspect to be dealt with after the release of the application. For this issue, our app won't have much problems when the user is using the app on the data. Whereas, on offline version, as the translink's static database changes after every two weeks, therefore in order to handle this, we will be updating our app when the static translink's database changes. We will be providing these changes in the form of an update of the app to the user so where we will delete the previous database and will update it with the latest database. This will allow user to remain up to date with current timings if the user wants to use the app offline. We will regularly test our app when it's operating offline to check if the app is being updated properly and is using the updated database.

Reliability

A user will use an app only if he/she feels that the app is serving some purpose to their needs and is reliable enough to use it. In order to test the reliability of our app, we will testing our apps regularly both using the data the and the offline database. When the user will be using the app on data, then we will also use to GPS coordinates to calculate the estimated arrival time using Google applications.

Non Functional Testing

Test #	Requirement Purpose	Action/ Input	Expected Result	Actual Result	P/ F	Notes
1	Responsiveness of the app	Improving the code at the backend by following standard practices and deleting redundant code	App will respond faster			

2	Battery Life	Measures on how the app will be working in the background	App will not affect much of user's phone battery			
3	Scalability of the app	1) Increasing the number of requests sent to Translink per API from 1000 2) Installing an embedded translink database on the phone	More number of users will be able to take advantage of the app			
4	Keeping the user up-to-date with the offline version of app	Updating the app whenever translink's static database changes to provide user with the latest timings	App and user will be able to estimate the timings of the trip correctly.			

Functional Testing Strategy

This project will utilize all the major categories of testing. We will first utilize Unit testing for individual classes that perform certain functions, such as the SQLite handler class that retrieves information from an embedded SQLite database, the TransLink API class that constructs queries via TransLink API and receives information from it in JSON, and the GPS class that utilizes Google Maps API to compute estimated arrival times between two input GPS coordinates.

All of the aforementioned classes take in inputs and returns outputs, so we will be using the Black-box testing method for detecting bugs, by first giving specific inputs (both valid and invalid, where we know what the expected outputs are) and comparing the outputs received from the tests with the expected outputs. For unit testing test cases, we will focus on inputs that are expected for the specific classes to handle. For example, if we want to test the SQLite database handler class, we will create the inputs based on specific queries, and the class should be able to retrieve the correct data based on the query.

We will be using another method to test out the classes responsible for UI. An example would be for the class that handles displaying a list of items (ie bus objects or bus stop objects in our overall product). For testing, we can simply input an array of strings, and test out if the ListView UI class can generate a list that displays the strings and see if the display would be satisfactory for the purposes of our overall product.

Once the individual classes are finished with unit testing, we will move on to integration testing. For this, we will first test smaller subsystems and slowly move on to larger systems. For example, we will first test a functional class (a class that's meant to perform a certain task like connecting to TransLink API) and a UI class. For this, we will input a specific bus stop number that will allow the TransLink API class to generate an array of buses. The UI should be able to display a list of buses appropriately, and can update itself whenever the TransLink API class generates a new array of buses. We will do this for all the smaller subsystems. Our overall project will contain two features: online mode and offline mode, and so we will implement these two features as two subsystems of the overall system.

Once that's done, the overall system testing will begin. The overall system should start off with a bus stop number as the input, and from there the system can set an estimated arrival time. We give the system a series of inputs, some valid and others not valid, and we will see if the estimated time calculated matches our expected output.

Overall, we will conduct unit testing every time there is a change it a particular class's code (ie adding or removing code). We will conduct integrated tests every week just to make sure that our overall project is doing ok, and we will begin systems testing when all the classes are functional. We will also be using potential customers to test out our app (ie. family, friends, etc) by installing our application onto their phones and asking for their feedback on accuracy, user-friendliness, and other parameters. Lastly, we will be using a spreadsheet to document the bugs encountered during testing, and we will have markers that will indicate their status.

Adequacy Criterion

The Back-end(for each model)

- Each path should be executed at least once
- Each branch should be executed at least once
- Make sure fault-based tests exist for each method that takes inputs from user, (such as invalid input as letters or buffer overflow etc.)
- Make sure at least one test exists for catching each exception

Reason: the back-end performance is crucial for our overall application. We want to achieve a relatively high path coverage as well as path coverage to avoid having bugs in it. Since the back-end processes input from the user, we need to make sure it has the ability to handle various invalid input.

The Front-end

- Make sure that at least one test exists for testing the input box can be used in each page
- Make sure that at least one test exists for testing the scrolling function on each ListView
- Make sure that at least one test exists for testing the press action on every ListView
- Make sure that sufficient tests exist that each path of the code is executed by at least one test.

Reason: the front-end interacts with the user therefore it is critical that all the widgets work as expected. The user should be able to enter, choose, or scroll on different pages. We are planning to model our front-end as a Finite State Machine and achieve 100% path coverage since we have relatively small number of pages.

The SQLite database

- Make sure that at least one test exists for handling an invalid input
- Make sure that at least one test exists for creating a query based on user input
- Make sure that at least one test exists for obtaining a correct output for a certain query.

Reason: the SQLite database stores static transit information (such as bus stop locations, bus leave time estimates, etc.) that will be used to provide an offline feature for our app. The different parameters in the database are linked by ID’s via the relational database model, and to retrieve correct information from the database it is crucial to test

that the tables are linked properly and that queries from user input is properly constructed.

Subsystem testing

- Make sure that subsystem is tested following the adequacy criterion of individual model

Reason: for each subsystem, we want to make sure every model in the system interacts as expected as well as being able to handle invalid inputs. We may use regression testing after we divide subsystem.

System testing

- Make sure that every use case is covered
- Make sure that at least one faulted-test exists for each input

Reason: as a system, we expect our App to perform as we expected in the design document, therefore we want to cover all the use cases. And for each model we tested the ability of handling invalid input therefore we want to make sure as a system, the App still can handle it. We may use regression testing.

Test Cases and Results

Unit testing
--for different handler/model
--for database
--for UI
System testing
--for subsystem(if we can divide)
--for the interaction of different parts
--performance testing

SQLite Database Testing (SQLite handler class)

Test #	Requirement Purpose	Action/Input	Expected Result	Actual Result	P/F	Notes
1	Invalid	Valid	Error			

	input	query that searches database using invalid search parameter	saying no results found			
2	Valid input (bus number)	Valid query that searches database	Returns an array of bus objects			
3	Valid input (bus number)	Valid query that searches database	Returns an array of bus stop objects			

Translink database testing

Test #	Requirement Purpose	Action/Input	Expected Result	Actual Result	P/F	Notes
1	Invalid API key error code	Call getStop with invalid API key	Error message with code 10001			
2	Database connection error	Call getStop without internet connection	Error message with code 10002			
3	Invalid stop number error	Call getStop with invalid stop number	Error message with code 1001			
4	Stop number not found error	Call getStop with stop number of right format but not used	Error message with code 1002			

5	Handle other possible but unlikely errors of Translink database for getStop method	Simulate returned error messages by getStop	Error message with corresponding error code			
6	Database connection error	Call getNextBuses without internet connection	Error message with code 10002			
7	Invalid API key error code	Call getNextBuses with invalid API key	Error message with code 10001			
8	Invalid stop number error	Call getNextBuses with invalid stop number	Error message with code 3001			

9	Invalid count error	Call getNextBuses with invalid count numbers	Error message with error code 3007			
10	Invalid time frame error	Call getNextBuses with invalid time frame	Error message with code 3006			
11	Handle other possible but unlikely errors of Translink database for getNextBuses method	CalSimulate returned error messages by getNextBuses	Error messages with corresponding error codes			

UI testing

Test #	Requirement Purpose	Action/ Input	Expected Result	Actual Result	P/ F	Notes
1	Check the validity of the input box	Tap the input line on the launcher page	The keyboard shows up as well as being able to input.	The keyboard shows up and being able to input.	P	
2	Check the ability of selecting one option and only one at a time from the provided ListView containing bus # and another containing a dest stop	Select the first, the last, and one in the middle of the list , one at a time.	The screen should direct to the next page after I select one item	The user on the next page		
3	Scrolling down and up functions on the ListView for both bus number page and destination stop page	Scrolling down and up on the bus number page and/or destination stop page(depending on which page the user is)	If there is not a long enough ListView then we can't scroll down or up, otherwise we should be able to reach the top or the			

			bottom of the list			
4	Notifying the user with the alarm.	Press the “set alarm” button	Direct the user to the "alarm is set" page			
5	Not setting up the alarm on the “Estimated Arrival Time” page	Press “no alarm” button	The screen should show “No alarm set” page			
6	Cancel the alarm	Press “cancel” button	The app should go back to the launcher page			