Quality Assurance Plan

CMPT 276

Group 5

App: Mealify

Members: John Zheng, Justin Yu,

Juey Lew, Vincent Yu, Feng Wu

Table of Contents

1. Unit testing tools and method

There is a multitude of unit testing frameworks for iOS development in the Swift programming language. Luckily, xCode comes equipped with a "Test Navigator." XCTest is a class provided by xCode in Swift that would provide us with a general framework to run unit tests. Since XCTestCase is a class definition extending xCodes build in XCTest framework to take into account a single test case. Included in the class are methods for setting up (setUp), tearing down (tearDown), an assertion case (testExample) and performance test cases. By using xCode's XCTestCase, we can automatically run unit test cases on our project.

It has been emphasized that in order to build unit tests that satisfies the best practice, we would have to seperate each unit test case into three parts: 'given', 'when' and 'then'. The 'given' part should consist of any variable declarations to run the test. 'When' consists of code that needs to be tested and 'then' would be any assertions on the result of the executable code.

There are other frameworks available to us that would help formulate test cases such as EarlGrey and Calabash. However, XCTest and XCUITest is sufficient enough for our project as both are created/designed by Apple and uses the Swift programming language. The documentation for both frameworks is comprehensive and encourages test-driver development.

2. Unit/system testing internal deadlines

Assuming each major feature would take approximately one week to complete, the team would allocate 1-2 days performing unit tests and fixing any errors that arise. Each iteration of the project is over the course of two week and we planned to implement 2-3 features each iteration. It seems likely to finish 2-3 features in 7-9 days amongst the team members, which would allow us with the remaining 5-7 days to perform the system test with all the new features along with any existing features, and to fix any bugs that arise from the merging of multiple features.

Upon completion of a major feature, all members of the group would perform an evaluation on the UI/UX or 'flow' of the application for that particular feature. Any concerns regarding any possible confusion the user may experience in the application's 'flow' should be taken account and mentioned to the rest of the group. Changes will be made if the confusion of a certain application's aspect is unanimous. This form of testing would occur the day a major feature is completed and the testing would span over two days.

User acceptance testing of version 3

Since a majority of the application's core functionality will be implemented within the first two iterations, it would be ideal to perform a user acceptance test July 25 4:30pm which is a week before the due date of version 3 of the project. Here we will ask users to do the following:

- input the amount of food intake for multiple days and update us on whether the application properly and accurately tracked the total food intake on each nutrient (carbohydrates, fat and protein)
- after inputting the food intake for a meal, check to see if the the meal's ingredients, nutrients, and date/time of consumption is accurately recorded in the calendar
- input a meal plan into a form and find all possible restaurants that serve dishes that would satisfy the corresponding meal plan
- to notify us in any confusion the application has created in regards to the flow from one page to another (if the user interface makes sense to the user)

3. Integration testing

Integration Test 1. For our first version of the project, we will finish implementing the feature that would allow users to fill in a form to input the food intake for a certain meal whether it be for breakfast, lunch and dinner. The result would be stored in our database that way we can query the nutrient intake and use the Charts API to help the user visualize intake over a period of time. To perform an integration test between the form, database and Charts API the Quality Assurance member and other developers will have to input nutrient data over multiple days and check if the charts plot the correct values. It would also help to check the database if it input the same values as was originally input in the form. This testing of this feature and fixing bugs that arise would take place between June 29-July 1 which leaves us one week to implement this feature.

Integration Test 2. The second version of the project would include the ability to track meals via the Google Calendars API. The user's breakfast, lunch and dinner nutrient intake as well as its ingredients will be included in the calendar in the corresponding date of consumption. In order to test this feature, whenever the user inputs a meal into a form, then the meal and its ingredients will be placed in the calendar in the correct date of consumption. Ideally, we would test this July 13.

Integration Test 3. Mealify would allow new users to register an account to the application. In the registration form, the user would input username, name, phone and other health/fitness statistics. This data would be saved in the SQLite database. The testing procedure would include registering a fake user with data in the registration form and asserting if the correct values are inputted into the database. The

passwords for users would have to be encrypted for user's protection. This feature would be tested July 10.

Integration Test 4. The Food Asset Map will be implemented in iteration 2 and completed by July 11. The feature would allow users to pick a certain diet restriction and the map would mark restaurants that serve meals that would cater to those restrictions. A way to test this is to make sure that for each restaurant that was marked there exists at least one meal in the menu that satisfies the user's diet restriction. Also, the markers must be placed in the right restaurant's coordinates.

Integration Test 5. The third iteration of the project would focus on the ability to message and compare meals amongst friends of users. A way to test this is to simulate a user interaction with another user. A user would message the other user and the correct message would be sent. We have to assert that the correct message was sent to the specified person. The testing for this feature would take place in July 25.

The difference between the integration tests above and unit testing is that these integration requires multiple components and external APIs. Unit testing would only cover tests for each individual component rather than multiple components and their interactions. In order to prevent "big bang" integration tests, we perform integration tests for each feature implemented then proceed to add integration tests between multiple features for each iteration.

4. Measuring size and complexity

As more features are implemented into our project, it is inevitable that the size and complexity of the source code will increase. There is a tool called **cloc** (https://github.com/AlDanial/cloc/tree/master/Unix) which is provided by the github repository and created by AlDaniel which would help us keep track of the number of lines of code written as well as the number of files. To be more specific, cloc also counts the number of blank lines and comment lines all using the UNIX command line interface.

Another tool called apps-top allows us to keep track of how many lines of code in our xCode project for each file. The application also runs on the UNIX CLI.

5. Miscellaneous

For each feature implemented, the developers would create their own ‘git branch’. Before merging the branch to master, as the quality assurance analyst, I would have to perform integration testing and to see if the features implemented does what is specified in accordance to the requirements documentation. Any issues will be logged in a separate document that way the developers that is responsible for implementing the feature can change the source code in order to satisfy the actual requirements. When the feature implemented satisfies all the requirements specified, it is the quality assurance analyst’s responsibility to merge the branch back to master and log the feature as completed.