CMPT 275

Assignment 2B – Quality Assurance Plan

By Team X

Project Group #10

10/20/2017

Project Website
https://sites.google.com/view/blackbox210

Ivana Jovasevic | 301247058

Shayne Kelly II | 301193702

Aaron Nguyen | 301254559

Peter Saffold | 301228703

Ya Qi (Jerry) Liu | 301255583

Table of Contents

0. Revision History

Revision	Status	Date	By
1.0	Template of QA Plan	September 26th, 2017	Aaron
2.0	Added Introduction, Unit test/schedule	October 01st, 2107	Jerry
3.0	Added User acceptance test, integration test, and schedule, Project complexity	October 06th, 2017	Shayne, Peter, Jerry
4.0	Detailing and formatting	October 10th, 2017	All
5.0	Finalize document	October 17th, 2017	All
6.0	Version 1 Revision	November 4th, 2017	All
7.0	Version 2 Revision	November 19th, 2017	All

Table 1: Revision History

1. Introduction

This Quality Assurance document is a guideline for the prototype iOS application Personal Emotion Trainer (P.E.T.) in development by Team X (Group 10). The following document will provide an overview of unit testing, integration testing and user acceptance testing of the application. Project complexity will also be measured and displayed. Additionally, the document will provide dates, times, and location for each testing procedure to ensure all the verification of the application are meet before the delivery deadline.

2. Unit Testing

2.1 Software Tools for Unit Testing

Xcode 9 provides the XCTest testing framework which has sufficient capabilities for thorough unit and performance testing. The IDE has the ability to for developers to write and generate test cases that will test the application's different modules and create automate test cases. It can also run continuous integration testing via command-line scripts [1]. The framework automatically generates a test bundle for the project. Test bundles contain test classes used to sort test cases and each test case will be testing a targeted module. Test classes are made manually by the developers and the tests are done via the test navigator. The XCTest framework can also display results after each test with pass/fail indicators once a set of tests is finished [1]. Baseline testing and benchmarks are also provided by XCTest framework. This will allow for speed and performance evaluation of each module, and will ensure the application is meeting the performance requirements.

2.2 Testing method

Testing will be conducted in regular intervals throughout the development process by the QA and development team. The test navigator will call code in the project to produce output. Then the output is validated for each section of code. If the code fails to produce correct results a failure indicator will be reported for further debugging. Each module will be tested separately for unit testing, and the system, as a whole, will be tested as a whole after each module is implemented successfully. Testing will be conducted before and after each revision to the source code.

However, majority of the development and testing will be done through Mac computers as Xcode is only available on the iOS platform. Each version of the application will be pushed onto the source control repository - GitHub. Finally, the application will be tested manually in addition to automated testing, to ensure quality user experience and visual aesthetics. The final application will be installed onto an iPad (Apple tablet) to check for compatibility issues and to prepare for live demonstration.

2.3 Internal Unit Testing Schedule

Unit testing will be done by the developer and QA members of the group. Each version and its primary modules will be tested individually to ensure functionality. The table below describes the start, end, and due date for each version of the application.

Version	**Start Date**	**End Date**	**Due Date**
V1	Oct 20th, 2017	Oct 30th, 2017	Nov 06th, 2017
V2	Nov 8th, 2017	Nov 14th, 2017	Nov 20th, 2017
V3	Nov 24th, 2017	Nov 30th, 2017	Dec 04th,2017

Table 2: Internal Testing Deadlines

3. User Acceptance Testing

User Acceptance testing will be conducted throughout all version of the application to test each major feature functionality. Testing procedures begin three days before the delivery due date to ensure that all bugs and issues are resolved ahead of time. Testing will be done by a member of the team, and a individual that is not part of the development team. In version 3, if possible, user acceptance testing will be done by a member of the Autism Mentorship Initiative (AMI). Testing duration is uncertain at the writing of this document, it is estimated to be around two to three hours.

3.1 User Acceptance Testing Procedure

Version 1

1. User will download P.E.T. application from the app store onto an iPad or be given an iPad with the app pre-installed.
2. User will open the application and register a account to test the account management features
3. User will be given 20 - 30 minutes to play around with the menu navigation to familiarize themselves and to test out the UI, then the user will go through the tutorial.
4. User will be asked to change game configurations and to ensure desire output.
5. User will be asked to change some account information under settings.
6. User will navigate to level selection menu and select tier 1 of levels, then select a question.
7. User will be shown an image and question prompt. Then user is asked to select the correct corresponding button that describe the given facial emotion.
8. User will be asked to make an incorrect answer to see if the right response is made.
9. User will skip one or two questions and see if the questions shuffle correctly.
10. User will go through questionnaires and make correct and/or incorrect answers, until next tier (tier 2) is unlocked (when 50% of the questions in tier 1 is answered correctly).
11. User will play as much of tier 2 as them desire.
12. User will sign out of application once finished, reopen application and sign back in with changed credentials in step 5 to check saved information and progression.

Version 2

1. User will open the app and load previously created account.
2. User will navigate to the level selection menu and select a level.
3. User will be asked to answer as many questions as possible in 30-45 minutes.
4. User will be asked to return to main menu, and open Question Manager.
5. User will upload one picture from the photo gallery or directly from the camera.
6. User will be asked for permission to access the photo gallery or the camera
7. User will select one picture to upload.
8. User will supply the question with an appropriate question prompt.
9. User will label the custom question with the appropriate answer which must be one of the emotion types included in the application.
10. User will return to level selection and select the custom-made questions under the button “Question Bank”.
11. User will load up the question or questions made in Question Maker.
12. User will attempt to answer the question made previously.
13. User will return to the main menu.
14. User will once again open the Question Manager page, and add additional question, or edition existing question, or delete existing questions.

15. User will exit the application.

Version 3

1. As in previous versions, user will open the app and load previously created account.
2. User will access level tiers implemented with the emotion recognition API.
3. User will open the stand-alone tier designated for the testing of the emotion recognition API.
4. User will be asked to authorize app to access camera on the device.
5. User will see an image and be asked to replicate the emotion of the image shown.
6. User will use the camera as an on-screen indicator to perform emotion shown on the screen.
7. User will be given a score of the emotion made and either they will pass and move on to the next level or fail and will have to renter a facial image.
8. After testing of emotion API, user will play the game without any instructions to make sure the application prompts are understandable and easy to use.
9. User will be given one hour to play test the entire application while development team closely monitors actions and output.
10. User will be asked about the general aesthetics of the application and about the difficulties of questionnaires.
11. User will be asked about the behaviours that they learned and emotions that they recognized from the gameplay.
12. User will exit the application.

3.2 Database Testing

Database testing will be conducted alongside user acceptance testing for Version 1 and 2 of the application. There are two sets of databases; a local SQLite XCode Database, and a remote Firebase web-based database. Each set of testing will ensure the communication to and from each database is working as intended.

3.3 Database Testing Procedure

1. Respond to request for login credentials from application (server-side).
2. Respond to request for question bank (local)
3. Storing data from account creation (server-side)
4. Storing data from progression of the gameplay (local)
5. Respond to request for local statistics (local).

3.4 User Acceptance Testing Schedule

User acceptance testing will be conducted once the application is near completion to ensure it meets all user requirements. This will allow for feedback on user experience before the deadlines.

Version	**Time**	**Date**	**Location**
V1	10:00am	Nov 3th, 2017	ASB, SFU Burnaby
V2	12:00pm	Nov 17th, 2017	ASB, SFU Burnaby
V3	10:00am	Dec 01st, 2017	ASB, SFU Burnaby

Table 3: User Acceptance Testing Time and Location

4. Integration Test Plan

Integration testing will be conducted after unit testing as shown in the internal schedule. For each version of the application Integration tests will be performed to ensure all of the associated modules are working as intended as a combined system. This is a very important procedure as testing of the communications of each modules and class will determine the performance of the entire system. The following integration test plan will be implemented as development moves forward. This will be a starting guideline as it is expected to change as development continues.

4.1 Integration Test - Version 1

- **Main UI**: The main user interface will be tested first to determine if interaction between each Viewcontroller is valid and cohered to the functional and performance requirements. All subclasses of the main UI will be tested, items such as buttons, text file registration, view transitions and any other unexpected input from the user will be tested and handled. All assets such as pictures, custom made UI components, and narrative guide (voice recording or stock recording) will be checked to ensure system calls the correct asset when asked.
- **Authentication System**: The user account system will be tested along with the database system to ensure that user data is properly send to and received from the server. Accounts will be created and added to the server via the main UI, then a QA member will verify the account by check the database for discrepancies.
- **Gameplay System**: The core mechanics of the gameplay will be tested thoroughly to ensure that all boundary conditions are taken care of, and the user will not be meet with any unpredicted conditions such as getting stuck on a particular screen, progression not registering, and correct answers not registering. The game will be completed once from start to end and timed to determine difficulty of the questionnaire. And that the progression system is communicating correctly with the server.
- **Database System**: As mentioned before the database will be tested alongside main UI and the authentication system to ensure each module are communicating in unison. The question bank proportion of the database will be tested with the gameplay system. The database will be required to retrieve the correct questionnaires and relating assets when user place a selection. The user account database will be tested with the authentication system.

4.2 Integration Test - Version 2

- **Camera and Photo Gallery Integration**: As the Emotion Recognition API and the custom question maker system will need access to the camera and photo gallery features. These systems are integrated first with the core application (Main UI) for testing. Once completed each system that calls the camera and photo gallery functions will be integrated next.
- **Custom Question Maker System**: Once the camera and photo gallery systems are performing as expected, this system will be the next to be integrated and tested. This function will call the camera or photo gallery components through the main UI and retrieve either

select image from photo gallery or a photo directly taken from the camera. The questions created will be accessible through level section, as it will be its only level tier.

4.3 Integration Test - Version 3

- **Emotion Recognition API**: The Azure or Affectiva API (Choice of API will be finalized soon) will be first tested on its own as a standalone function. As a standalone function, it will be implemented with the camera function. Since the API will utilize images taken by the camera and process the results. The API will be tested with the range of emotions that the API can handle to ensure that the proper output is obtained. Then it will be incorporated into and be tested with the core gameplay system.
- **Other Minor Features**: The integration testing will involve testing system from both Version 1 and Version 2 extensively. Since the system will have many minor features implemented in additional to the Emotion API; the focus during this period of development will be to ensure quality of overall user experiences, runtime performance improvements, and ensure visual aesthetics of the overall application.

4.4 Integration Testing Schedule

Integrated testing will be conducted once the unit testing is complete and each module is ready to be implemented as a system. Testing will be done before the user acceptance testing in order to allocate time for debugging.

Version	**Start Date**	**End Date**	**Due Date**
V1	Oct 30th, 2017	Nov 05th, 2017	Nov 06th, 2017
V2	Nov 14th, 2017	Nov 19th, 2017	Nov 20th, 2017
V3	Nov 30th, 2017	Dec 03, 2017	Dec 04th,2017

Table 4: Integration Testing Schedule

5. Project Complexity Overview

Project complexity is an important statistic to keep track of throughout the lifecycle of the project. Project complexity analysis will help manage the software as it grows beyond control. It allows for better planning, coordination and keep project organized. To determine important metrics at each stage of the development cycle, tools must be considered for quick and easy analysis.

Tools available within the ***Xcode*** development platform will allow for project complexity measurements. Xcode itself would be the primary tool for code metrics measurement. It has an extensive list of tools to measure application performance and analysis. The IDE can also provide tracking for number of files, lines of code, and the number of classes used. Additionally, Xcode is able to evaluate and give an estimated size of the application before uploading onto the app store.

Instrument on the iOS is also able to measure many of the important metrics of the application during run time. Instrument is a specialized tool part of the Apple development environment that collects data as it profiles. It can profile a wide array of items such as performance, memory usage, energy usage, and memory problems, such as leaks, abandoned memory, and zombies [2]. Instrument is a standalone app that could be used either with or without Xcode. However, it is only available on the iOS.

The ***CLOC*** (Count Lines of Code) library is available to measure the amount blank lines, comment lines, and physical lines of source code for a wide range of programming language (Swift) [3]. CLOC is a single, self-contained file that will not be directly interacting with the source code to measure project size. CLOC is also cross-platform so the measurement can be done on Windows as well. CLOC is an open source tool.

Tailor is another measurement tool available to use. It is a Cross-platform static analyzer and linter for Swift. Tailor supports Swift directly out of the box and it enforce style guidelines outlined in the Swift Programming Language, GitHub, Ray Wenderlich, and Coursera style guides [4]. Tailor does however, require Java (JRE or JDK) Version 8 or above to run. Furthermore, Tailor have Xcode project integration which will be ideal for the development process. Tailor is released under the MIT license.

Primary metrics to be measured include but not limited to:

- lines of code
- Load time
- Memory usage
- Data usage
- Power usage

A simple estimation is done for the lines of code of P.E.T. to showcase development and estimated application growth. As figure 1 presents, Version 1 will be implementing most of the UI features and account management feature. The number of lines of code will increase steadily between Version 1 and Version 2. At this time, major features such as a custom question maker will be implemented to the program. Between Version 2 and Version 3 there is shown to be another steady increase, because for this release we will be adding an emotion recognition API. At this stage, the application is also in the polishing and detailing phase.

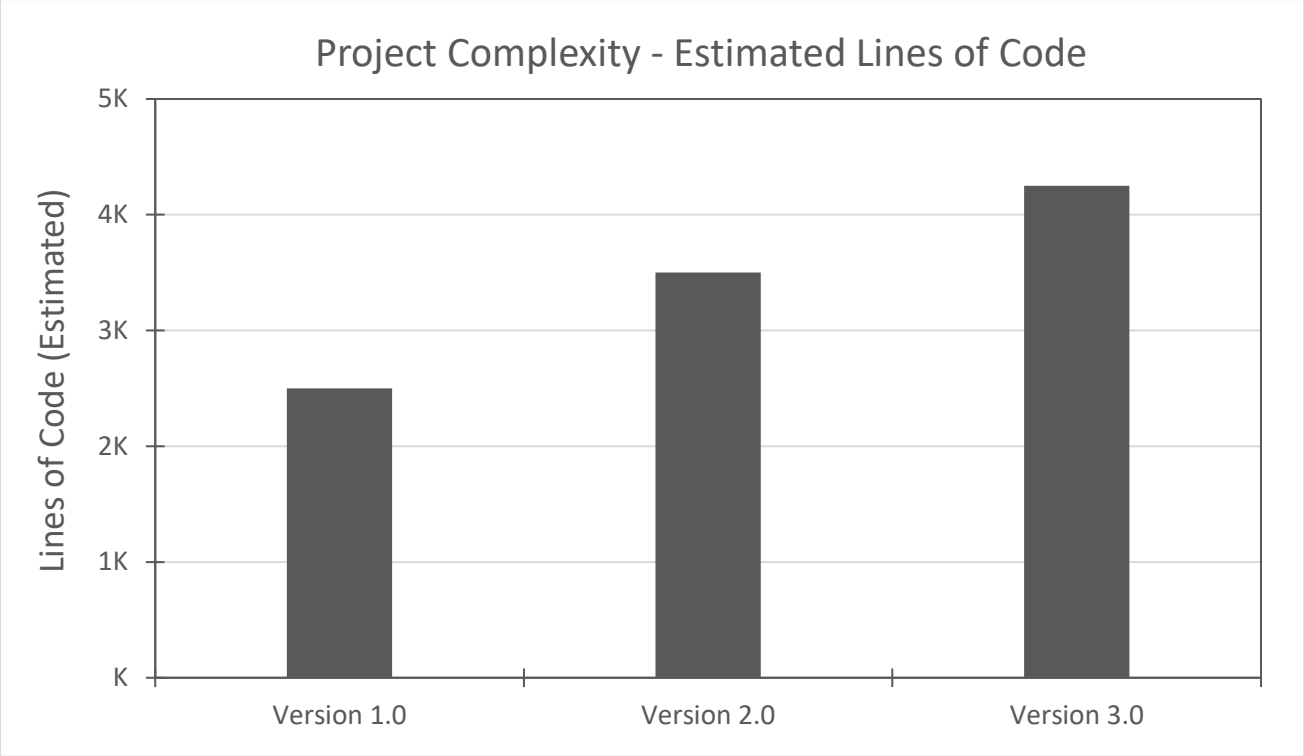

Figure 1: Project Complexity Estimation

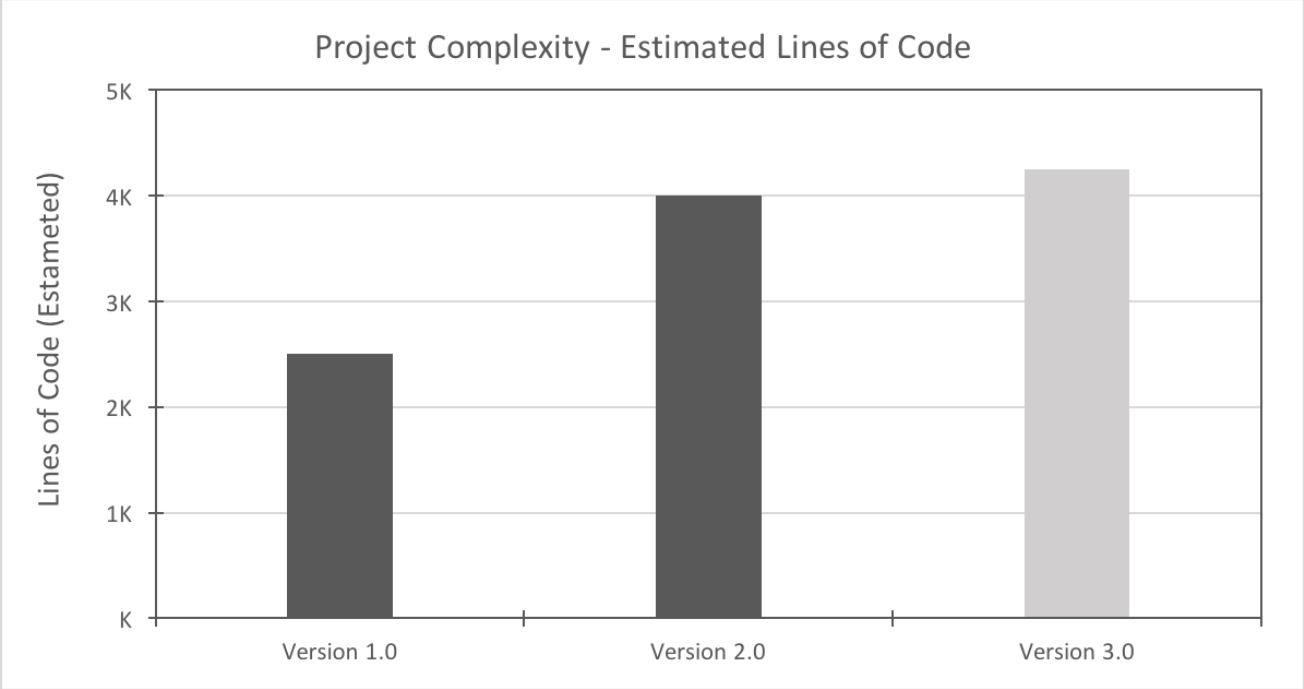

Figure 2: Project Complexity for Version 2 with estimate for Version 3

6. Additional QA Measurements

To further ensure the quality of the product additional QA measurements are listed below. This will be a development guideline for the team to follow in order to produce a high quality application.

1. Coding standards will be followed throughout development process. Source code must be commented in detail, and descriptive variable names must be used. Source code must be written in a neat and clean fashion. Classes and functions must have full description explaining in detail of the functionally, priority, type of input, and expected output.

2. Documentation member and/or project manager will inspect the source code weekly to see if coding standards are followed. Any issue that appear will need to be discussed during weekly meetings. Furthermore, application performance, major bugs, and development issue will also be discussed during weekly meetings. Development and QA reports will be highlighted during weekly meetings.

3. If developers want to make major changes to the code, they must contact project manager and CC rest of the team. Developer must provide a detailed outline of the changes they wish to make and the reason. The Development Team will keep a record of proposed changes indicating which module(s) or class(es) the changes belong to, dates of changes, and name of the author.

4. All bugs and issue arises from development of the application will be documented in detail. Each member performing testing of the application will keep their own individual QA documents. QA documents will include the type of error found, how to replicate the error, possible solutions for the error, and date/time for discovery of the error. Report document from the QA members will be given to the development team to correct the issues.

5. Multiple backup of the source code will be made and archived.

6. The whole team will meet to test the complete application manually to ensure functionally before every delivery deadline.

7. Reference

[1] Apple Developer Library, "*Testing with Xcode*" [online]. Available: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/01-introduction.html#//apple_ref/doc/uid/TP40014132-CH1-SW1. [Accessed: 01-Oct-2017].

[2] Instruments User Guide," *Instruments User Guide: About Instruments",* 13-Sep-2016. [Online]. Available: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/index.html. [Accessed: 30-Sept-2017].

[3] AlDanial, "AlDanial/cloc," *GitHub,* 28-Sep-2017. [Online]. Available: https://github.com/AlDanial/cloc#why_use. [Accessed: 01-Oct-2017].

[4] Sleekbyte, "sleekbyte/tailor," *GitHub,* 27-Mar-2017. [Online]. Available: https://github.com/sleekbyte/tailor. [Accessed: 05-Oct-2017].