Software Test Plan

for

Iris: Email Reader Application

Version 1.20

Scott Arnette

Joseph LaCava

Derek Ouzia

Bryan Smith

University of Virginia’s College at Wise

Department of Mathematics and Computer Science

April 12th 2016

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Objectives 1

1.2 Testing Strategy 1

1.3 Scope 2

1.4 Reference Material 2

1.5 Definitions and Acronyms 3

2. Test Items 3

2.1 Program Modules 3

2.2 User Procedures 4

2.3 Operator Procedures 4

3. Features 4

3.1 Features to be Tested 4

3.2 Features Not to be Tested 4

4. Approach 4

4.1 Component Testing 5

4.2 Integration Testing 5

4.3 Interface Testing 5

4.4 Regression Testing 5

4.5 Beta Testing 5

5. Pass/Fail Criteria 5

5.1 Suspension Criteria 5

5.2 Resumption Criteria 6

5.3 Approval Criteria 6

6. Testing Process 6

6.1 Test Deliverables 6

6.2 Testing Tasks 6

6.3 Responsibilities 6

7. Environmental Requirements 6

7.1 Hardware 7

7.2 Software 7

7.3 Security 7

7.4 Tools 7

7.5 Publications 7

7.6 Risks and Assumptions 7

8. Change Management Procedures 7

9. Test Procedures 8

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Bryan Smith | 12/6/15 | Initial Creation | 1.00 |
| Scott Arnett | 4/12/16 | Sections 3 and 4 | 1.10 |
| Bryan Smith | 4/12/16 | Finish up rest of the sections. | 1.20 |

# Introduction

This Software Test Plan (STP) will provide an overview of testing procedures and information for the Email Reader Application, also known as Iris. It will be referred to as Iris, or simply, the application, henceforth. The plan will identify items to be tested, the features to be tested, and the types of testing to be performed.

Primary focus of the testing of the application will be making sure it is backwards compatible (works on older versions) with older versions of Android back to API 11 (Honeycomb).

## Objectives

* Identify existing project information and the software that should be tested.
* List the recommended test requirements (high level).
* Recommend and describe the testing strategies to be employed.
* Identify the required resources and provide an estimate of the test efforts.
* List the deliverable elements of the test activities.

## Testing Strategy

The Android framework includes an integrated testing framework that helps test all aspects of the application and the SDK tools include tools for setting up and running test applications. Whether you are working in Eclipse with ADT, Android Studio, or working from the command line, the SDK tools help you set up and run tests within an emulator or the physical devices being targeted.

The testing framework has these key features:

* Android test suites are based on JUnit. Plain Junit can be used as well to test a class that does not call the Android API, or Android's JUnit extensions to test Android components.
* The Android JUnit extensions provide component-specific test case classes. These classes provide helper methods for creating mock objects and methods that help control the lifecycle of a component.
* Test suites are contained in test packages that are similar to main application packages, so you don't need to learn a new set of tools or techniques for designing and building tests.
* The SDK tools for building and tests are available in Eclipse with ADT, Android Studio, and also in command-line form for use with other IDEs. These tools get information from the project of the application under test and use this information to automatically create the build files, manifest file, and directory structure for the test package.
* The SDK also provides monkeyrunner, an API for testing devices with Python programs, and UI/Application Exerciser Monkey, a command-line tool for stress-testing UIs by sending pseudo-random events to a device.

A large subset of testing for Iris will be White Box testing and Integration tests.

Refer to section 5 of this document for a detailed list of specific test plans.

## Scope

Scheduled and unscheduled changes will be managed via Issues and Pull requests integrated in BitBucket. BitBucket is the Git remote hosting service used for the Iris project. Git is a source control tool.

Updates to already released and distributed versions of Iris will be performed via the Google Play Store and will show up automatically if the user has Iris installed. A delayed or limited rollout can be defined in Google Play so that only small subsets of users see the update initially. This will ensure that if there was something that went unnoticed and made it in the release, we will receive feedback from this small subset of users and limit the impact on both user frustrations and/or network infrastructure for distribution

Lastly, Google Play Store may not be used in the event that licensing does not want to be paid in order to use Google Play Store. In this situation, the compiled APK can be distributed by common methods, including but not limited to: email, website download, content management systems, or manual command line installation using ADB.

## Reference Material

* Iris: Email Reader Application – Software Requirements Specification
* Iris: Email Reader Application – Software Design Document
* Android Testing <http://developer.android.com/tools/testing/index.html>

## Definitions and Acronyms

* ADB – Android Debug Bridge. Versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device
* APK - Android application package is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.
* API – Application Programming Interface. A set of routines, protocols, and tools that govern a software specification.
* GUI – Graphical User Interface. An interface that allows users to interact with electronic devices through icons and visual indicators.
* Iris – The name of the Email Reader Application. The project that this STP is for.
* OS – Operating System. The software that the product runs on.
* REST API - Representational state transfer application programming interface. A light weight web based API. The client does not need to know the structure of the API but the server provides the information the client needs to interface with the service.
* SDD – Software Design Document
* SRS – Software Requirements Specification.
* STP – Software Test Plan

# Test Items

## Program Modules

Due to constraints listed in this document, much of our automated testing is not feasible. Therefore much of our testing will be manually done by hand. This limits what we can do as much of these components cannot be tested manually without being integrated already. We will run integration an UI tests, white and black box testing. We will also have a separate independent testing team who will design and run their own tests for this project.

## User Procedures

Due to the small size of the Iris project, any user documentation will be re-read by the team and manually edited to reflect the changed or correct information.

## Operator Procedures

The application shall check for environment compatibility upon startup. Thus it is self-checking and no procedures are needed other than initial installation of the application. The only requirement for installation is having an Android Device with API 11+.

# Features

## Features to be Tested

Features of Iris to be tested originate from the functional requirements of the software system as defined in the SRS. The software features that shall be tested automatically include the getting/setting of user preferences/settings and the getting/setting of all information related to accounts and messages. Manual testing will be performed to ensure voice services can be launched and perform correctly, messages can be kept or removed as they arrive, multiple messages can be kept/removed in multiple quantities, and user preferences regarded screen timeouts and sync frequencies are performing as expected. Non-functional requirements to be tested include application performance (items such as load times and adherence to sync frequencies) and safety requirements (the application will not perform malicious tasks such as automatic removal of messages).

## Features Not to be Tested

Tests will not be performed for exceeding API calls, or for inboxes that contain a large amount of messages.

# Approach

Testing of the Iris application will be performed on a component level, an integration level, and an interface level. Test activities will be noted as passed only if the test is passed. If a test does not pass, it fails and will need to be revisited. Regression testing will also be performed as changes are made to the software system in development to ensure functionality does not break. Testing activities will be performed for each functional requirement defined in the SRS.

## Component Testing

Component testing will be assisted by automatic tests used to ensure appropriate variables and items can be created, set, and retrieved. Automated component testing will be denoted as passed or failed based on what is returned when automated testing is performed.

## Integration Testing

Integration testing will be performed as more components are combined to form the software system. All functional requirements of the application as denoted in the SRS will be tested to ensure the system functions as intended.

## Interface Testing

It has been determined that due to the limited GUI elements in the Iris project, that most UI elements will be tested manually by the team. It would be counterproductive to spend the time to setup the UI testing platform Google has for Android, as it would take longer and likely be less accurate due to the nature of the Iris project. The UI testing platform may be utilized to verify GUI states (does it save its state when it is moved from background to foreground and visa-versa?) however.

## Regression Testing

Regression testing will be performed after any changes to the system are made to ensure these changes have not adversely affected previously tested functionality.

## Beta Testing

Public beta testing is not currently planned for the Iris application due to time constraints, but may be considered at a later period.

# Pass/Fail Criteria

If an exception occurs or no defined passing value is encountered, the test automatically fails. The test will also fail after 5 seconds of running with no passing value or exception being encountered. A test will only fail if a passing value is encountered that was set by the test designer.

## Suspension Criteria

If a test fails, other tests that are separate will continue to attempt to run. However, their results will be thrown out due to possible effects from the failing test. Tests will then be suspended until the reason the test failed has been identified and remedied.

## Resumption Criteria

When the issue that caused a test or tests to fail is remedied, the tests are resumed. With the low amount of tests due to the size of this project, typically we will re-run all tests from the beginning.

## Approval Criteria

All tests should produce a value that is equal or in the range of the defined passing value.

# Testing Process

A select few tests for our process where automated using Android and Android Studio’s testing frameworks. However, much of the application is beyond our knowledge of how to test with regards to Android specifics. Since the applications is relatively small, we will manually test individual functions and the UI.

## Test Deliverables

Refer to [Section 9](#_Test_Procedures)

## Testing Tasks

An Android Emulator and Android Studio would need to be setup and installed correctly in order to begin automated tests on Iris. Tests can then be ran from test window or by executing a run configuration that specifies the tests to run. Refer to [Section 7.5](#_Publications) on details of skills needed for Android Testing.

## Responsibilities

The development team for this project was in charge of designing tests. However we have an independent testing team as well and they designed and ran their own tests on the application.

# Environmental Requirements

<Specify both the necessary and desired properties of the test environment including the physical characteristics, communications, mode of usage, and testing supplies. Also provide the levels of security required to perform test activities. Identify special test tools needed and other testing needs (space, machine time, and stationary supplies. Identify the source of all needs that is not currently available to the test group.>

## Hardware

* An Android device (phone or tablet) is needed for testing the application.
* The device must have touch support or mouse and keyboard support.
* The device must have connectivity to a network connection
* The network connection shall be speeds similar to that of Mobile 3G or higher (144Kb/s+).
* Testing can be performed on an emulated device so long as it meets the above specifications as well as the software specifications below.

## Software

* The device must be running Android OS 3 (API 11) or higher
* The device must have Google Play Services v8.3 or higher installed

## Security

No current testing environment security requirements.

## Tools

Refer to [Section 1.2](#_Testing_Strategy)

## Publications

* Android Testing Tools (<http://developer.android.com/tools/testing/testing-tools.html>)
* Android Testing Concepts (<http://developer.android.com/tools/testing/testing_android.html>)
* Testing in Android Studio (<http://developer.android.com/training/testing/start/index.html>)

## Risks and Assumptions

Our main risk and constraint is lack of knowledge on testing on the Android platform. It is highly specialized and outside the scope of our knowledge. We plan to main blackbox test each function manually in order to help mitigate the lack of Android testing knowledge.

# Change Management Procedures

There is very little to no change management process as we are a very small group. Typically there are only two working on development or fixes and it is in a pair programming approach. If a bug is found by our team or the separate testing team, it is logged in ClearQuest and the two developers work the issue simultaneously. It is then put through some component, integration, and regression tests and when verified is pushed to the source code repository.

# Test Procedures

We had one failed test. This was not due to the logic but with the test itself. The Iris Voice Service is highly dependent on Android OS components that are handled mostly automatically and are not touched by us. After doing much research, we concluded that we would skip writing tests for the IrisVoiceService and other highly dependent sections. Instead we manually tested them by hand in integrations tests, in which they all passed. Writing the tests that were highly dependent on the Android OS was outside of our knowledge range and after much research, we needed to move on in order to finish the project, and not sink too much time in a few automated tests. It is unfortunate however, as much of our logic falls inside theses sections. Since it is such a small project we could manage testing each function manually.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date Tested:** | | **4/12/16** | | |
| **Test Performed By:** | | **Bryan Smith** | | |
| **Project Name:** | | **Iris – SWEG Capstone 2016** | | |
| **Software Version:** | | **1.0** | | |
| **Related Requirements:** | | **4.0-4.10** | | |
| # | Test Step Description | | Expected Result | Passed |
| **1** | Test GMailAccount UserID Constructor | | Set the account objects User ID attribute to the value passed in the constructor. | P |
| **2** | Test GMailAccount current history ID Constructor | | Set the account objects current History ID attribute to the value passed in the constructor. | P |
| **3** | Test GMailAccount UserID Setter/Getter | | Set the account objects UserID attribute to the value passed in the setter function. Uses getter function to verify the value was set appropriately. | P |
| **4** | Test GmailAccount CurrentHistoryID Setter/Getter | | Set the account objects CurrentHistoryID attribute to the value passed in the constructor. | P |
| **5** | Test Get Shared Preferences | | Verifies that the shared preferences object produced by the android system is obtainable. This is what stores user settings. | P |
| **6** | Test Get/Set String preferences | | Verifies that a string user preference can be set and retrieved. | p |
| **7** | Test Get/Set Float preferences | | Verifies that a float user preference can be set and retrieved. | p |
| **8** | Test Get/Set Long preferences | | Verifies that a long user preference can be set and retrieved. | p |
| **9** | Test Get/Set Boolean preferences | | Verifies that a boolean user preference can be set and retrieved. | p |
| **10** | Test Get Key | | Verifies that a preference key can be found using its id | p |
| **11** | Test Get Token | | Verifies that the authentication token of a Gmail Account can be retrieved from the API | p |
| **12** | Test Get Gmail Service | | Verifies that the Gmail API is able to correctly be accessed | P |
| **13** | Test Get Gmail Account Credential | | Verify that the Gmail API can be accessed by an authorized user and the credential is returned that will allow for API data access. | P |
| **14** | Test Get Initial Gmail Account Credential | | Verifies a blank Gmail Credential can be obtained. This is used for methods that require them, but are run before any users are logged in. | P |
| **15** | Test IrisVoiceService | | Verifies that the IrisVoiceService was started and connected to correctly. | F |
| **16** | Test GMailAccount UserID Constructor | | Set the account objects User ID attribute to the value passed in the constructor. | P |
| **17** | Test GMailAccount current history ID Constructor | | Set the account objects current History ID attribute to the value passed in the constructor. | P |
| **18** | Test GMailAccount UserID Setter/Getter | | Set the account objects UserID attribute to the value passed in the setter function. Uses getter function to verify the value was set appropriately. | P |
| **19** | Test GmailAccount CurrentHistoryID Setter/Getter | | Set the account objects CurrentHistoryID attribute to the value passed in the constructor. | P |
| **20** | Test Get Shared Preferences | | Verifies that the shared preferences object produced by the android system is obtainable. This is what stores user settings. | P |
| **21** | Test Get/Set String preferences | | Verifies that a string user preference can be set and retrieved. | P |
| **22** | Test Get/Set Float preferences | | Verifies that a float user preference can be set and retrieved. | P |
| **23** | Test Get/Set Long preferences | | Verifies that a long user preference can be set and retrieved. | P |
| **24** | Test Get/Set Boolean preferences | | Verifies that a boolean user preference can be set and retrieved. | P |
| **25** | Test Get Key | | Verifies that a preference key can be found using its id | P |
| **26** | Test Get Token | | Verifies that the authentication token of a Gmail Account can be retrieved from the API | P |
| **27** | Test Get Gmail Service | | Verifies that the Gmail API is able to correctly be accessed | P |
| **28** | Test Get Gmail Account Credential | | Verify that the Gmail API can be accessed by an authorized user and the credential is returned that will allow for API data access. | P |
| **29** | Test Get Initial Gmail Account Credential | | Verifies a blank Gmail Credential can be obtained. This is used for methods that require them, but are run before any users are logged in. | P |
| **30** | Test IrisVoiceService | | Verifies that the IrisVoiceService was started and connected to correctly. | F |