**Test Approach Document**

**For Mobile Application Manager**

By,

Bhargavi Madunala

Mitin Sharma

Surya Kiran Akula

Vishal Koosuri

Contents

[**1.0** **INTRODUCTION** 3](#_Toc479005555)

[**2.0** **SCOPE** 3](#_Toc479005556)

[**3.0** **Testing Strategy** 3](#_Toc479005557)

[**3.1** **Unit Testing** 3](#_Toc479005558)

[**3.2** **Performance Testing** 4](#_Toc479005559)

[**3.3** **Function Testing** 4](#_Toc479005560)

[**3.4** **Configuration Testing** 5](#_Toc479005561)

[**4.0** **Hardware Requirements** 5](#_Toc479005562)

[**5.0** **Environment Requirements** 5](#_Toc479005563)

[**6.0** **Control Procedures** 5](#_Toc479005564)

[**7.0** **Features to Be Tested** 5](#_Toc479005565)

[**8.0** **Risks/Assumptions** 5](#_Toc479005566)

[**9.0** **Appendix** 5](#_Toc479005567)

### **Introduction**

Mobile App Manger is a mobile based application which runs on the android operating system. The application incorporates *processor* stats, memory stats, Network stats. It helps users of the system to manage the processor throughput, and view network and memory stats allocated to the applications running. User can keep track of mobile performance and purge the applications causing difficulties.

### **Scope**

It is critical that all system and subsystem interfaces be tested as well as system performance at this early stage.

The interfaces between the following subsystems will be tested:

* Processor statistics
* Memory statistics.
* Network statistics

The most important performance measures to test is to check the response time for Statistics to be displayed on the UI

### **Testing Strategy**

The Test Strategy presents the approach to the testing of these applications. Test Requirements described *what* will be tested; *how* it will be tested.

### **Unit Testing**

Unit testing is performed by the developing team. They perform simple tests while the code is being developed. As the developers have a prior knowledge of how the code works, white box testing can be performed on the code.

Example code:

public void testChangeText\_sameActivity() {

// Type text and then press the button.

onView(withId(R.id.editTextUserInput))

.perform(typeText(STRING\_TO\_BE\_TYPED), closeSoftKeyboard());

onView(withId(R.id.changeTextButton)).perform(click());

// Check that the text was changed.

...

}

### **Performance Testing**

The goal of Performance testing is to verify and validate the performance requirements have been achieved by measuring response times and other time sensitive requirements. Performance testing is usually executed frequently.

Technique:

* Modify data or modify scripts to increase/decrease the number of iterations.
* The goal is for successful completion of the scripts without any failures and within the expected / required time allocation.
* Performance testing should be performed on a dedicated machine or at a dedicated time. This permits full control and accurate measurement.

### **Function Testing**

Function Testing should focus on any target requirements that can be traced directly to use cases (or business functions), and business rules. The goal is to test and verify data acceptance, processing, and retrieval and its appropriate implementation.

This type of testing is based upon black box techniques, that is, verifying the application by interacting with the application via the output interface and analyzing the output (results).

UI testing is the process of testing the Graphical User Interface of the Application under Test. UI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes and other popup windows, etc.

Technique:

The expected results occur when valid data is used. The appropriate error / warning messages are displayed when invalid data is used. All identified defects should be addressed.

### **Configuration Testing**

Configuration testing verifies operation of the application on different android mobile configurations. The goal is to achieve software functionality on any kind of android based operating system.

### **Hardware Requirements**

Physical machines : Desktop/Laptop

Operating systems : windows 7.

Emulating device : Nexus 5x.

### **Environment Requirements**

* Compilers            : JVM
* IDE                : Android studio
* Source code repository    : GitHub
* Build process        : Gradle
* Testing frame work : [Espresso](#_Appendix)
* Android test support library : [AndroidJUnitRunner, UI Automator](#_Appendix)

### **Control Procedures**

Every incident encountered during the testing process, are documented.

### **Features to Be Tested**

Software features will be tested namely Network statistics, processor statistics, and Memory statistics.

### **Risks/Assumptions**

Risks:

Manual Testing is performed throughout the project.

We do not have any automated testing plan.

Assumptions:

All the team members are dedicating their time on completing their tasks. The design document may be modified based on the project requirements that may change timely.

### **Appendix**

1. Espresso

Espresso is a testing framework for Android to make it easy to write reliable user interface tests.

Google released the Espresso framework in Oct. 2013. Since its 2.0 release Espresso is part of the Android Support Repository.

Espresso automatically synchronizes your test actions with the user interface of your application. The framework also ensures that your activity is started before the tests run. It also let the test wait until all observed background activities have finished.

It is intended to test a single application but can also be used to test across applications. If used for testing outside your application, you can only perform black box testing, as you cannot access the classes outside of your application.

Espresso has basically three components:

ViewMatchers - allows to find view in the current view hierarchy

ViewActions - allows to perform actions on the views

ViewAssertions - allows to assert state of a view

1. AndroidJUnitRunner

The AndroidJUnitRunner class is a JUnit test runner that lets you run JUnit 3 or JUnit 4-style test classes on Android devices, including those using the Espresso and UI Automator testing frameworks. The test runner handles loading your test package and the app under test to a device, running your tests, and reporting test results.

1. Android Testing Support Library

The "Android Testing Support Library" provides an extensive framework for testing Android apps. This library provides a set of APIs that allow you to quickly build and run test code for your apps, including JUnit 4 and functional user interface (UI) tests. You can run tests created using these APIs from the Android Studio IDE or from the command line.