Phone UI Test Automation Design Document

Jan-15, 2016
Version 1.0
Jan-04, 2018
Version 2.0

Copyright

Contents

Copyright

1.0 Overview and Purpose

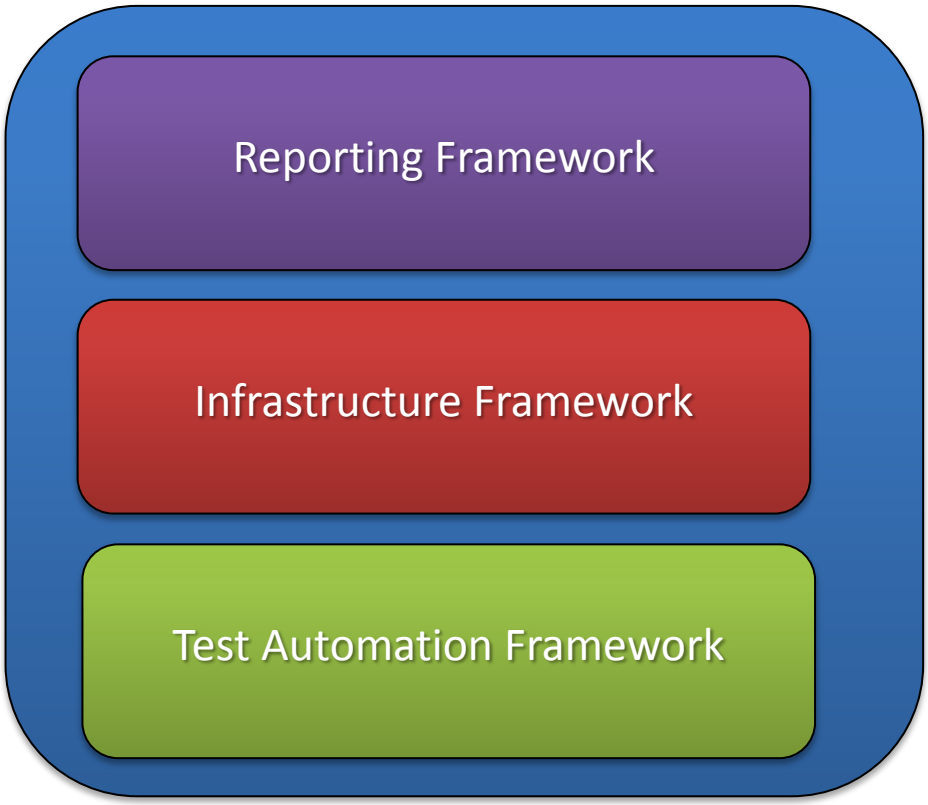
1.1 Introduction

This design document will provide an overview of how Phone UI automation testing will be covered for VISP, VzCloud and other Phone based Services. It aims to capture all test automation architecture and requirements.

2.0 Phone UI Test Architecture

The diagram below shows how the test architecture will be built on three different layers.

1. Test Automation Framework – This will be a multi-layered automation framework based on Appium for writing test scripts to be run on IOS and Android phones.
2. Test Infrastructure Framework – This Continuous Integration framework is based on Jenkins and Robot Framework
3. Test Reporting Framework – This reporting framework is based on Jenkins based Email reporting, Test Reports dashboard for generating Test Metrics and graphs.

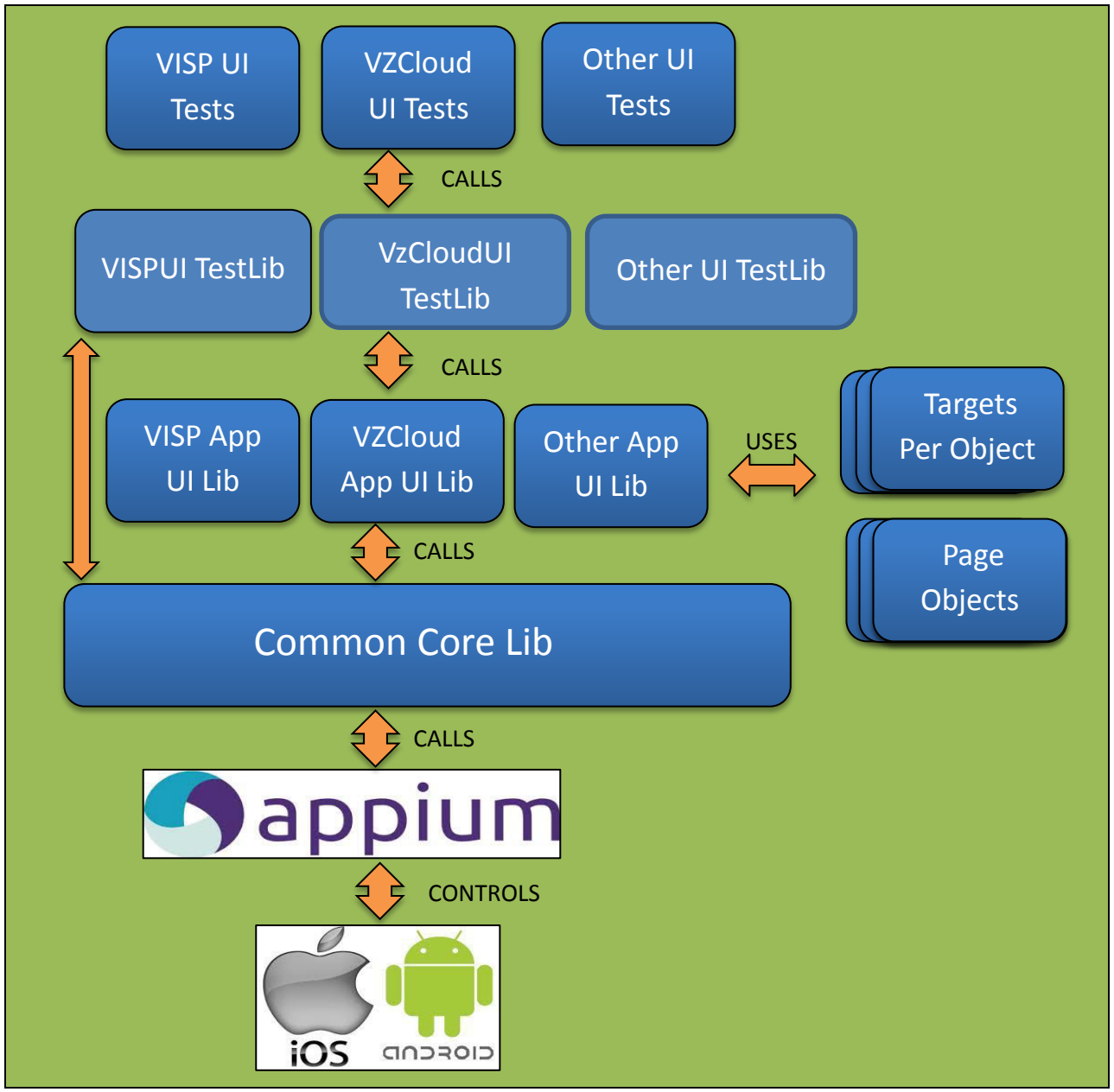


2.1 Test Automation Framework

The Test Automation Framework is based on Appium. Appium is based on the web driver protocol and is used to drive native, hybrid and mobile web apps on IOS and Android devices. The architecture diagram below shows the multilayered framework that is being proposed.

One of the main design goals of the framework is to ensure that we have a generic framework which can be used for any Phone UI testing to avoid duplicating automation code as we test different phone based services.

The framework relies on a design pattern called Page Objects. The Page Object pattern represents the screens of a particular app on the device as a series of objects. Within an app's UI there are areas that the tests interact with. A Page Object simply models these as objects within the test code. This reduces the amount of duplicated code and means that if the UI changes, the fix need only be applied in one place. This also ensures the code is modular which helps with reuse, debugging and maintainability.



The proposed test framework above shows a multi-tiered architecture.

Appium: It is the web driver protocol based automation tool to drive native, hybrid and mobile web apps on IOS and Android devices. It has a set of APIs used to drive Phone UIs.

Common Core Lib: It is a common set of generic actions across Apps that are used for UI actions such as click element, press menu and etc. , Test setup actions (Appium server setup, screen record, device related actions), and debugging (logging and exception handling) in order to reuse it and speed up test automation. It uses **Appium API** for UI actions, **ADB** tool for **Android** device related actions and **XCUITest** Driver for **iOS** device related Actions.

App UIs: These are object representations of apps, which use page objects and their associated targets. App UI objects are the test entry points to the App.

Page Objects: These represent the services offered by a particular UI page rather than exposing the details or mechanics of a UI page. Since tests are using Page Object APIs, they will not to be exposed to the underlying Appium API's. Page object APIs will return other page objects as a test traverses thru different UIs. It allows all the UI targets elements to be specified in one location as Class variable and per page. This means if any target elements change in the UI, it can be changed easily in one place rather than doing it across test scripts.

Targets: These are actual User actions offered by a particular UI page and supported for Test Case like play video, pause video, find freebee Icon.

UI Test Lib: These are high-level wrapper APIs which use several page object APIs together to form a set of user actions. There is UI Test Lib per each App and idea of using it to reduce the test case complexity and allow cross-pollination between different test suites across different testing applications. It also calls UI common core library directly for operations like screen record operations, logging, and open app.

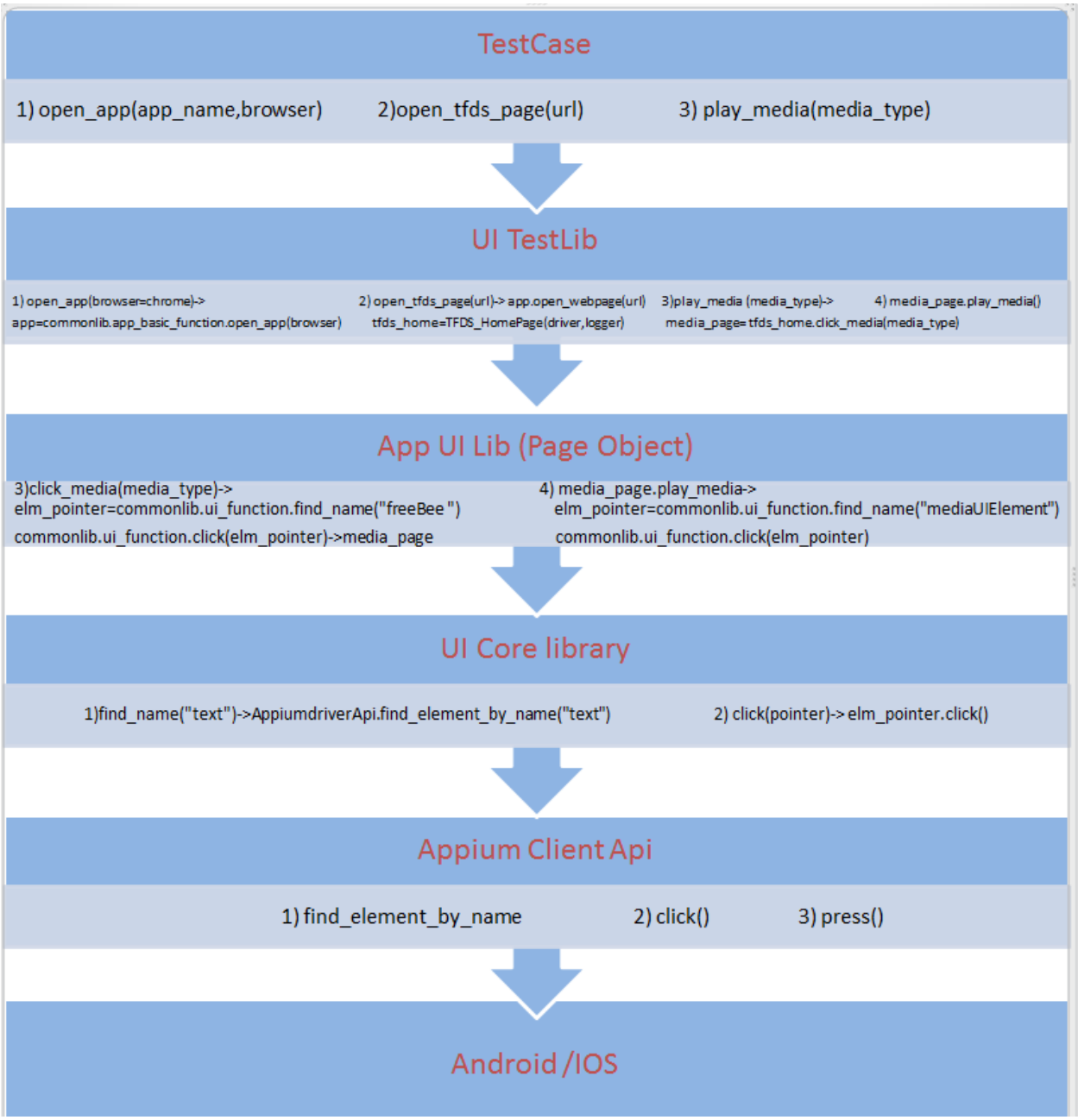
App UI Tests: These are tests, which primarily use UI helper methods to construct test cases along with configuration files to define hardware elements like IPs, usernames, and passwords.

2.2 UI Test Case Flow

The flow chart below attempts to capture a toll free data service test case on the web traversing through the different layers in order to show the test automation framework flow.

A test case in this particular case has two methods. One is open_app which has two input parameters (android/ios endpoint, chrome/safari/Firefox browser). "Open app" will use core lib "app_basic_function" api and opens chrome app if browser passed as Chrome. The next part of the test case if open_tfds_page and play_media as part of the toll free data service. These methods require a handle to the common core api library and a set of urls to traverse.

As we follow the test case through the different layers, when the play_media is called, it calls the UI test library APIs which in turn call the page object APIs, which in turn call the UI core library APIs, which in turn call the Appium client APIs which interact with the IOS/Android device to control it as shown below.



2.3 Test Automation Framework Benefits

Some of the benefits of using this framework are

- Object oriented code is modular and maintainable. Changes to targets and page objects functionality is done in one place rather than across different test scripts
- Test case are simple with parameterized input of device type and browser type
- Improved debug ability as we can find exact layer test failed at.
- Generic APIs to allow control of multiple native, hybrid and mobile web apps

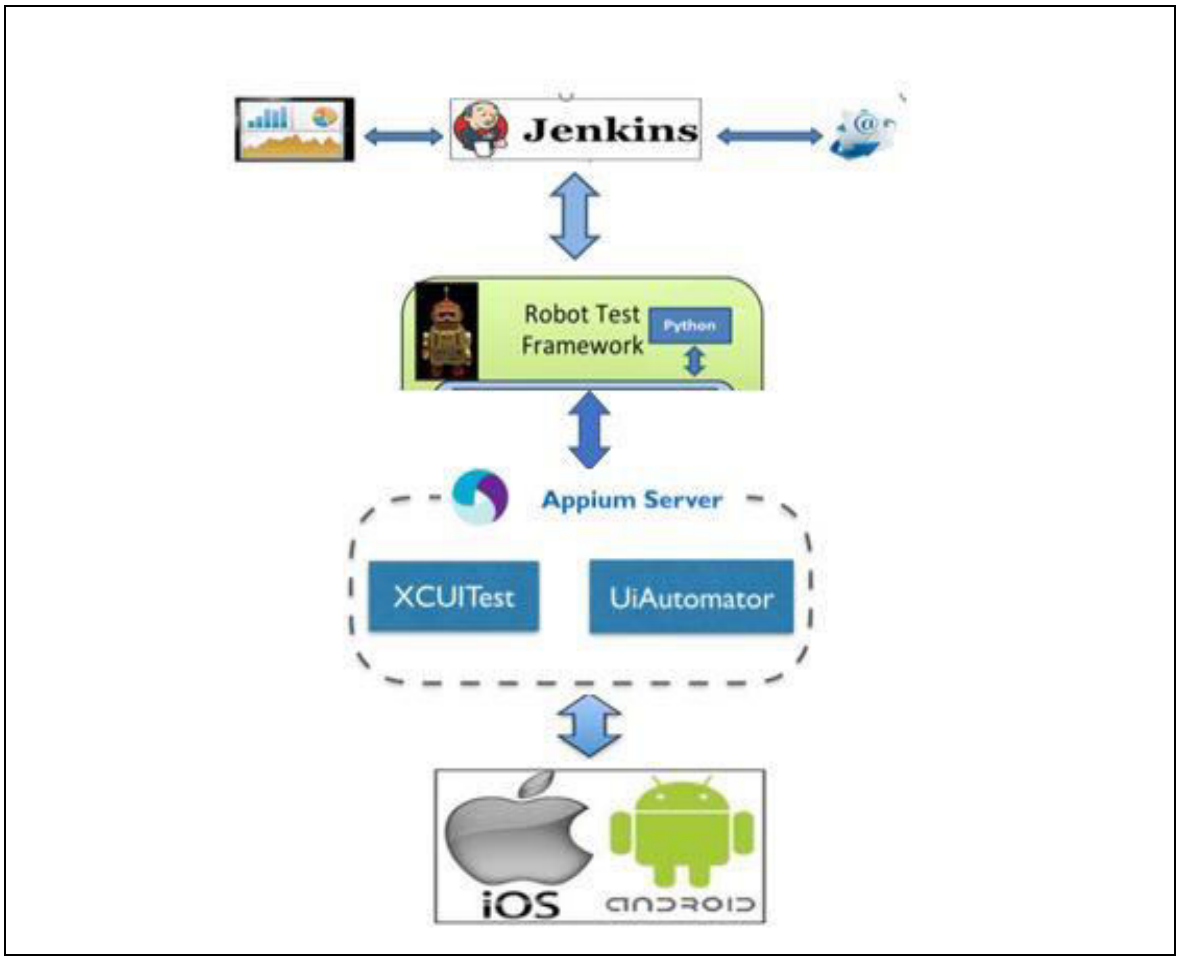
2.4 How to use and Limitations

Below things tell you how to integrate your app into this framework

- Test cases, UI Test Lib and Page objects modules are required to be developed for each app individually since each app had its own page actions and different UI elements.
- Common core libs can be reusable.

3.0 Test Infrastructure Framework

The test infrastructure framework is a Jenkins based continuous integration framework for device tests. The framework is as shown below.



Jenkins: Continuous Integration server to schedule, execute and store test suite and results after each test job.

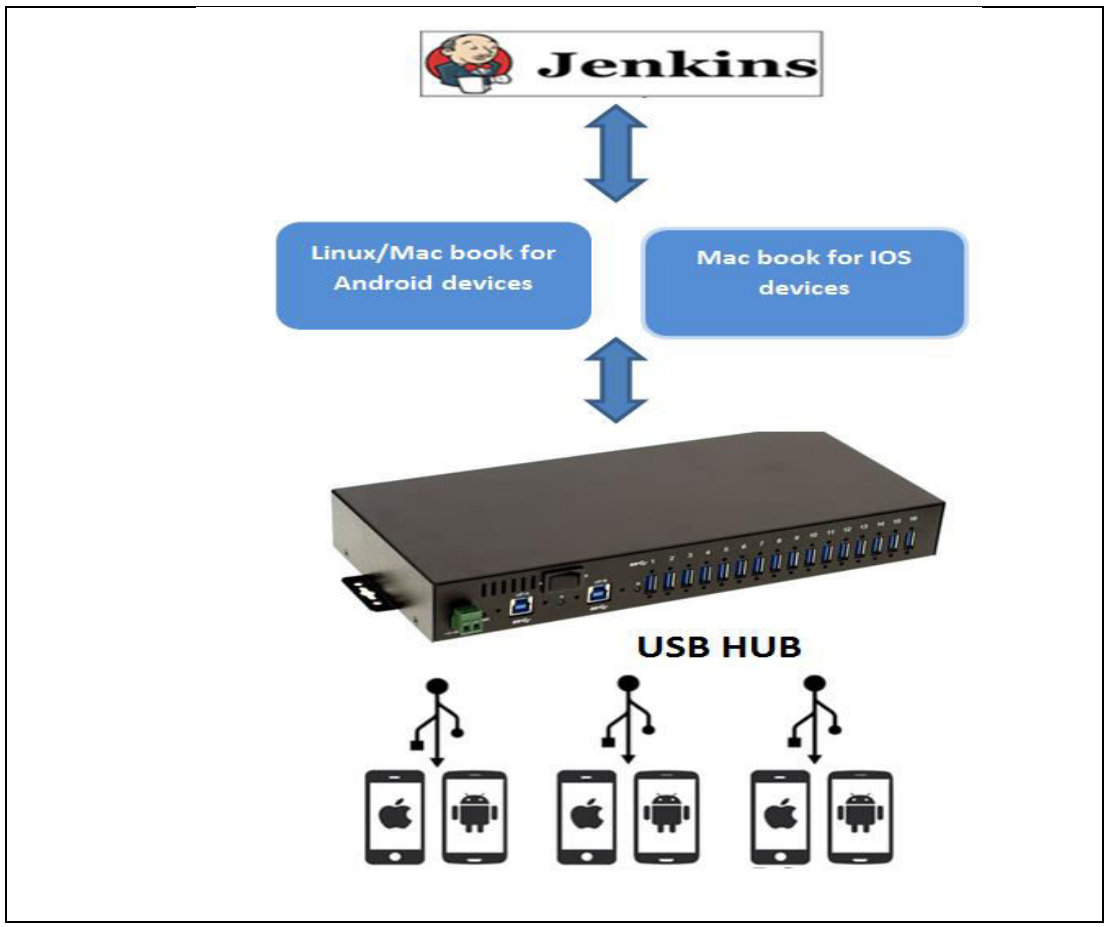
Test Reports Dashboard: It's Jenkins Plugin and uses Dc.js JavaScript library to generate test metrics per distro and generate all relevant test reports per test environment.

Robot Test Framework: A operating system and application independent generic test automation framework, which allows us to wrap Appium tests in order to get complete test reporting, test case tagging, Jenkins integration and other test libraries.

Appium Framework: It's a mobile automation framework used to automate user actions on IOS, Android handsets using Instruments and ADB tool.

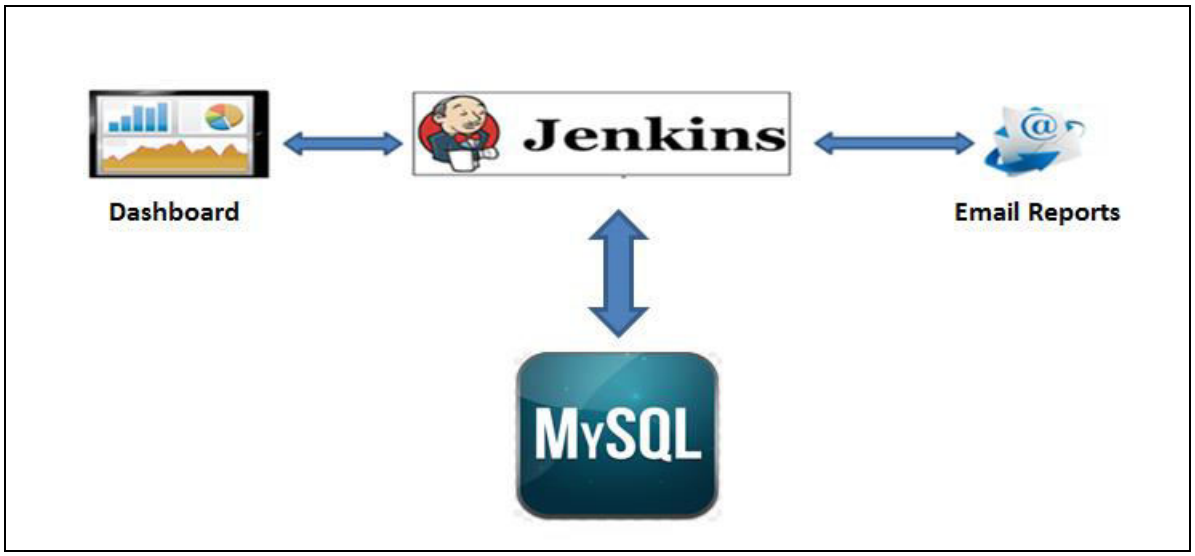
3.1 Test Infrastructure Setup

The following diagram shows the proposed hardware setup for the IOS/Android devices. Each set of phones homed to a datacenter will be connected to a USB hub, which will be controlled by a server. Tests will be run from the server, which will control the phones via the hub. The server will be connected to Jenkins in a master-slave configuration



4.0 Test Reporting Framework

The test-reporting framework will have three types of reports that will be generated for each test run as shown below.



Email Reports: As part of each Jenkins test run, email reports will be generated and sent out to a mailer with a list of tests which passed or failed. This email will provide immediate status of the tests and help trigger a triage chain as required.

MySQL DB: All relevant test metrics as part of test execution will be continuously send to MySQL DB and persistently stored in appropriate databases. Examples of test metrics include average response times, response codes etc.

Dashboard: It's a Jenkins plugin and developed on top of DC.js java script Library. It feeds the data from MySQL dB about Test stats and generates different kind of Graphs.