Binary Ninjaz

Harvest

User Manual

Letanyan Arumugam	14228123
Sizo Duma	15245579
Teboho Mokoena	14415888
John Ojo	15096794
Kevin Reid	15008739
Shaun Yates	16007493

Stakeholders

SAMAC: Barry Christie

Contents

List of Figures

1 Testing Process

In this section, the testing process, based on IEEE-829, will illustrated. This process will fulfil the requirements stated in this document.

Three phases within the testing process can be identified, namely:

1. Test specification,
2. Test implementation,
3. Test reporting.

Each phase is now elaborated.

1.0.1 Test Specification

The test specification can be subdivided into 2 parts. On the one hand, there is *planning and verification*, on the other there is *Analysis, design, and implementation*.

Planning and verification may consist of the following activities:

- Specifying the scope and the risks, and defining the objectives of the tests;
- Determining the strategy;
- Taking decisions with regard to what must be tested, what roles apply, how the test activities will take place and how the outcomes of the tests will be implemented;
- Planning of the test design activities;
- Planning of implementation and evaluation.

During the *analysis, design, and implementation* phase, the test objectives will be transformed into specific test conditions and test cases. This may involve the following activities:

- A review of the test base (such as requirements, architecture, design, and interfaces);
- Evaluation of the testability of the test base and the test objects;

- Identification and prioritisation of the test conditions;
- Designing of the test cases;
- Identifying the necessary test data in order to support the test cases;
- Designing the test environment and identifying the infrastructure and tools.

1.0.2 Implementation of Tests

The implementation of tests forms the phase in which the test procedures and scripts are implemented and the results of these (including incidents) are logged. This may involve the following activities:

- Implementation of the tests, manually or by means of a tool;
- Logging the results of the implementation;
- Comparing the results with the result that had been predicted;
- Reporting any setbacks in the form of incidents and analysing these in order to identify the cause;
- Repeating the tests as a result of the incidents identified.

1.0.3 Test Reporting

Test Reporting forms the phase in which the implementation of the tests is compared with the objectives. This may involve the following tasks:

- Checking of the test logs and the list of defects and comparing these to the exit criteria specified in the test plan;
- Investigating whether additional tests are required;
- Writing a summary report.

1.1 Testing Frequency

Since tests are used to validate the continued performance of a module—or the system as a whole—tests should be run after each module is finished being modified or created.

2 Testing Tools

There are many types of tests that can be run, in the case of **Harvest**, Unit and UI tests are run.

2.1 Android

2.1.1 JUnit

JUnit is a unit testing framework for the Java Environment, and as such is used for testing the Android—programmed in Java—implementation of **Harvest**. Android development is carried out using the Android Studio IDE, which facilitates easy use and set up of the JUnit framework. Android Studio also recommends the use of JUnit. For the reasons given, JUnit was selected to run unit tests on Android.

Android unit tests can be found here.

2.1.2 Espresso

Espresso, like JUnit, is integrated into, and recommended by, Android Studio, and therefore is used to facilitate UI testing on Android.

Android UI tests can be found here.

2.1.3 Execution

Both UI and Unit tests are executed in the same manner for Android, the primary difference being that a UI test will need to run on a connected device or virtual machine.

In order to run a unit or UI test for Android, one must open the file containing the desired test(s) using Android Studio as in figure 1. Once there, one can right click on the green 'test' button in the margin, then they can select to run the tests, the results of which will be seen at the bottom of the IDE.

2.2 iOS

2.2.1 XCTest

XCTest integrates seamlessly into Xcode's (the iOS IDE) testing workflow, and for that reason is used to facilitate all testing on iOS. XCTest is used for

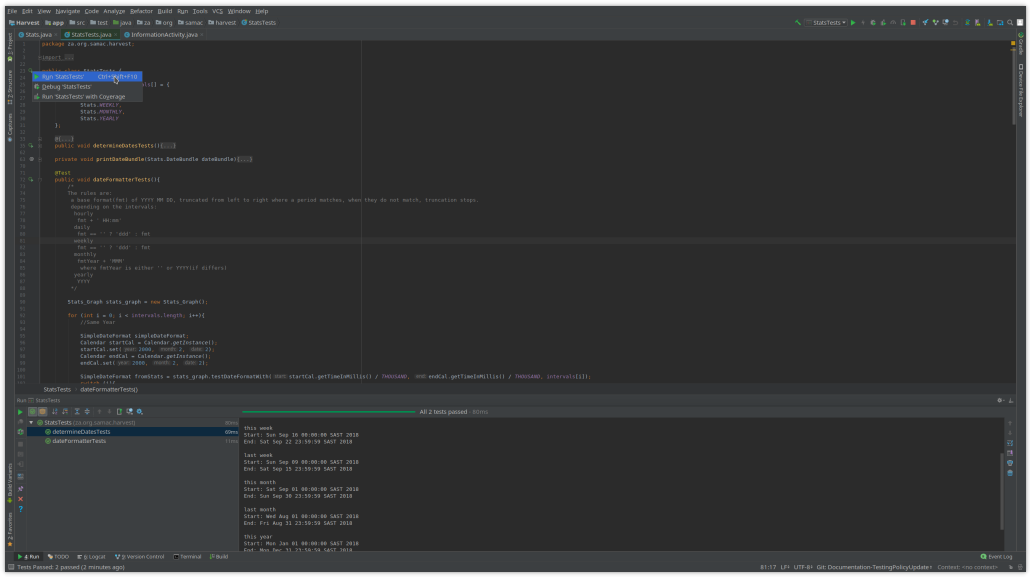

Figure 1: Android Studio Test Example

both UI and unit testing on iOS.

iOS unit tests can be found here, and UI tests can be found here.

2.2.2 Execution

To begin tests in Xcode, click on the play button in the top right, on the line that is highlighted in blue; as seen in figure 2. This will run the tests. The results will look like figure 9.

2.3 Web

2.3.1 QUnit

QUnit is used for unit testing javascript functions. It was chosen since it displays its results in a convenient way—by including them in the web page, and it's very simple to use.

QUnit tests can be found here.

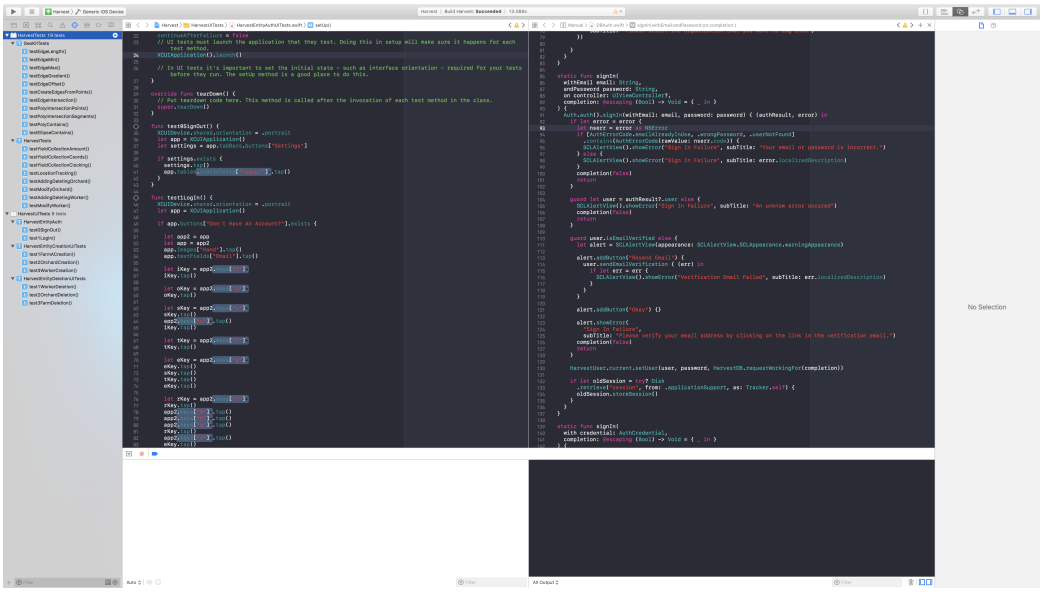

Figure 2: iOS Xcode Test Start

```
<!--QUnit testing-->
<!--Uncomment to enable-->
    <!-- <div id="qunit" style="background-color: white"></div>
    <div id="qunit-fixture"></div>
    <script src="https://code.jquery.com/qunit/qunit-2.6.0.js"></script>
    <script src="tests.js"></script> -->
```

Figure 3: Before Enabling Tests

```
<!--QUnit testing-->
<!--Uncomment to enable-->
    <div id="qunit" style="background-color: white"></div>
    <div id="qunit-fixture"></div>
    <script src="https://code.jquery.com/qunit/qunit-2.6.0.js"></script>
    <script src="tests.js"></script>
```

Figure 4: After Enabling Tests

2.3.2 Execution

QUnit To execute QUnit tests, go into the html file that uses the functions to be tested. At the bottom, find a segment that mentions `QUnit Testing`, such as in figure 3; then simply uncomment the code, as in figure 4, and refresh the page. The results are now displayed, such as in figure 5.

3 Test Cases

TODO when requirements done.

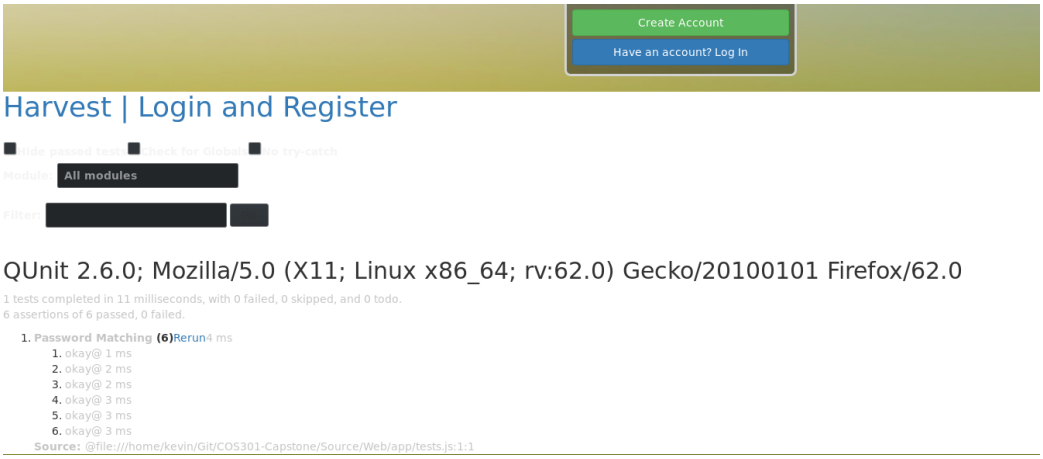

Figure 5: Results of QUnit test

4 History

4.1 Android

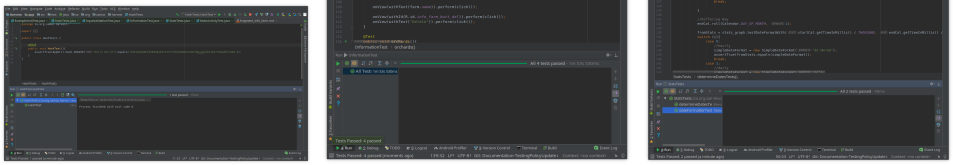

Figure 6: Unit Test to Verify Functioning of Hash Function

Figure 7: UI Test to Verify Functioning of Information Editing

Figure 8: Unit Test to Verify Formatting of Dates

Android test logs can found here. Refer to figures 6, 7, and 8 for examples

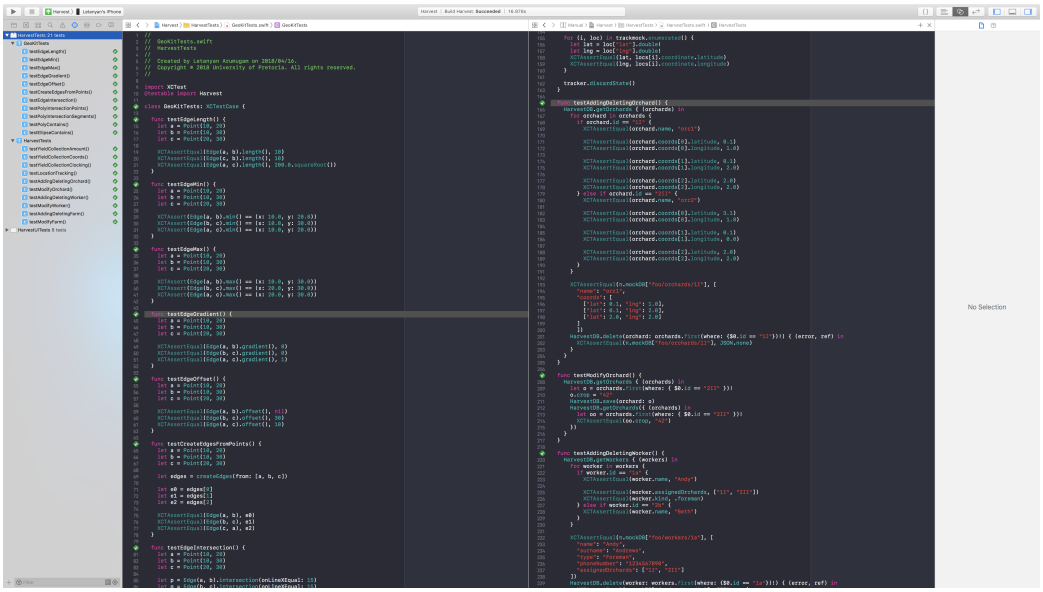

Figure 9: iOS Xcode Test Example

of tests.

4.2 iOS

iOS test logs can be found here.

Refer to figure 9 to see iOS tests.

4.3 Web

Web test logs can be found here.

Refer to figure 10 for an example of a QUnit test.

```
QUnit.test("Password Matching", function ( assert ) {
    showRegister();
    assert.ok(checkPass("123", "123"));
    assert.ok(checkPass("Hello World", "Hello World"));
    assert.notOk(checkPass("hello world", "Hello World"));
    assert.notOk(checkPass("123", ""));
    assert.ok(checkPass("", ""));
    assert.notOk(checkPass("", "123"));
});
```

Figure 10: QUnit Test to Verify Passwords are Rejected or Approved Appropriately