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, and
3. Test reporting.

Each phase is now elaborated.

1.1 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.

1.2 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

1.3 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

2 TESTING TOOLS - unit testing level

Unit tests (otherwise known as component tests) seek to identify defects in, and verify the functioning of, testable software modules, programs, items, classes etc.

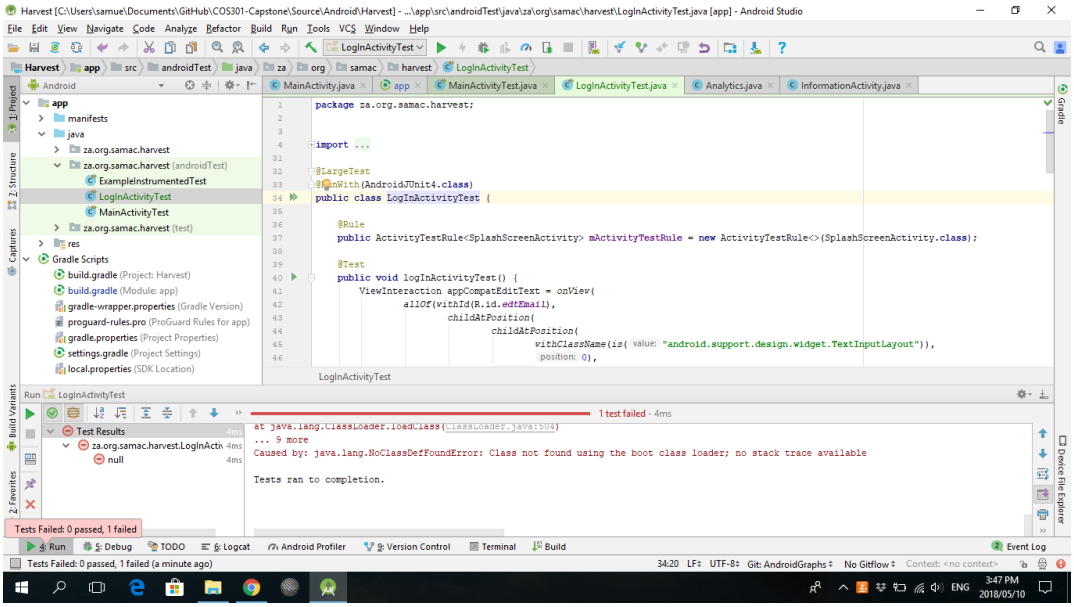
Unit tests are capable of verifying functionalities, as well as non-functional characteristics. The test cases are based on work products, such as the specifications of a component, the software design or the data model. Unit tests are used in order to be certain that the individual components are working correctly.

Unit tests mainly make use of white-box testing techniques and are therefore carried out with a knowledge of the code being tested and possibly also with the support of the development environment (the unit test framework, debugging tool, etc.). That is why these tests are often carried out by the developer that wrote the code or by a party of a similar calibre. Any defects identified are resolved as soon as they are detected.

The following testing tools will be used for the Harvest Software:

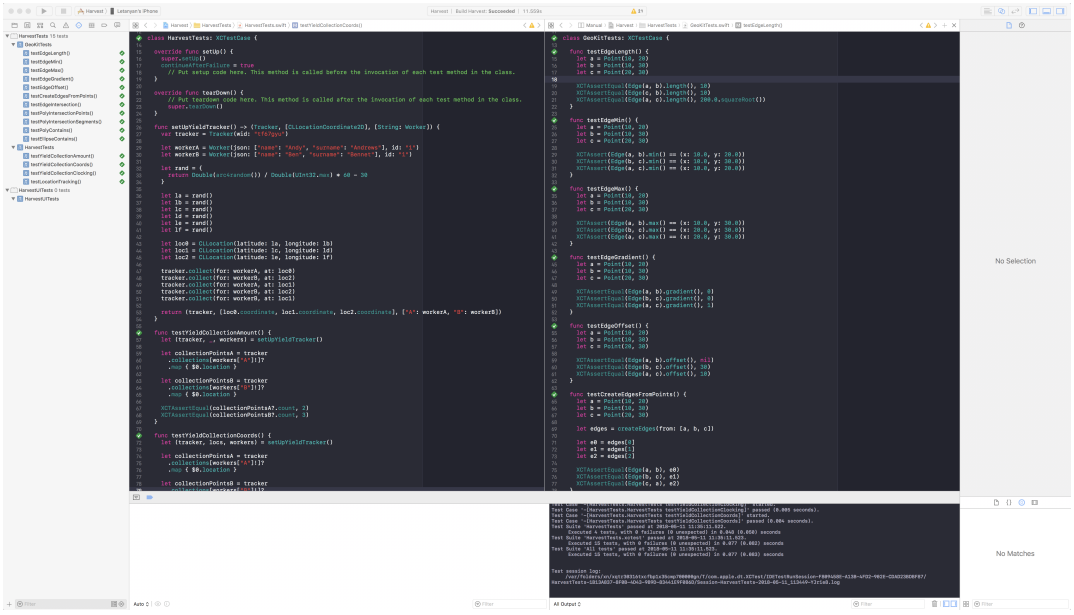
2.1 Android Studio

JUnit is a unit test for the Java development language. This is important in the case of a test-driven development. JUnit is an instance of the xUnit architecture for unit test frameworks. The JUnit framwork will be used for out automated testing purposes. To run tests one will need Android Studio. To execute the tests one can right click on the tests directory and select 'Run tests.'



Link to Android App Testing

2.2 XCTest framework example test on IOS App



Link to IOS App Testing

2.3 NPM

The Ava framework will be used for automated testing purposes. To run tests one will need to run "npm test" from the testing directory.

2.4 XCode

The XCTest framework will be used for out automated testing purposes. To run tests one will need XCode. To execute the tests one will need to use XCodes build command.

3 TESTING PHILOSOPHY

Implementation of both simple and holistic tests will be of concern. However, holistic testing will be of more interest. Testing workflows are of vital importance. The project does not have many simple tasks that could provide us with the safety guarantees we would like. We instead seek to find safety in implementing tests that check the whole interaction of these few unit cases.

3.1 Benefits

Automated test provide developers with the ability to have safe of mind that they have not made any changes that break the program in any way or introduce bugs in any part of the program. Forcing developers to write tests forces them to check every part of code they write making tests a form of documentation of a projects requirements.

4 TEST EVALUATION

Strict zero test case failure will be followed. Any branch with a failing test case will never be merged into master. Hence we will ensure that every branch runs tests before a pull request.