Testing Policy and Guidelines

Developers
Tegan Carton-Barber
Emma Coetzer
Aeron Land
Luveshan Marimuthu
Kendra Riddle

May 2019

Contents

1 Introduction **3**

1.1 Purpose . . . 3

2 Testing Policy **3**

2.1 Purpose of Policy . . . 3

2.2 Benefits of Policy . . . 3

2.3 Test Policy Objectives . . . 3

3 Testing Guidelines **4**

3.1 Purpose of Guidelines . . . 4

3.2 Definition of Testing . . . 4

3.3 Testing Approach . . . 4

3.4 Test Levels . . . 5

3.5 Guiding Principles . . . 5

3.5.1 Test Planning . . . 5

3.5.2 Test Design . . . 5

3.5.3 Test Execution . . . 5

3.5.4 Test Closure . . . 5

3.6 Approach to Automated Testing . . . 5

3.7 Test Priorities . . . 6

1 Introduction

1.1 Purpose

The purpose of this document serves to provide an overview of the testing policies and guidelines that are to be implemented and followed. This document contains two sections which define the foundations of the testing culture followed with regard to the FABI Mobile system.

2 Testing Policy

2.1 Purpose of Policy

The purpose of a well defined testing policy is to provide an overview of how testing is to be conducted. This section describes the benefits of a test policy, the objectives, evaluation, and general testing approach.

2.2 Benefits of Policy

The primary benefits of a testing policy are the following:

- Define a structured policy and guideline which provides structure
- Promotes transparency
- Encourages testing to be done continually and consistently
- Promotes effective team work

2.3 Test Policy Objectives

The objectives of the testing policy aim to accomplish the following:

- Plan tests in advance in order to ensure that testing can commence soon into the development
- Provide an indication as to the priority of certain use cases
- Promote early detection of defects
- Encourage the documentation of test cases

3 Testing Guidelines

3.1 Purpose of Guidelines

These guideline set out to inform interested stakeholders about the fundamental structure, implementation, and evaluation regarding the testing process.

3.2 Definition of Testing

Testing is a process which tests the actual results against those results which were expected. Testing is used, not only to test for errors and gaps in logic, but also internal and external aspects which relate to the system. Testing can also help identify missing functional requirements or requirements which have been incorrectly transposed into code. Testing also ensure that system integration is done correctly.

3.3 Testing Approach

The approach used is Test Driven Development(TDD). The use of TDD will ensure that the development and testing team achieve good quality code and good test coverage.

The rules of TDD are:

- You are not allowed to write any production code unless it is to make a failing unit test pass.
- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
- You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

This approach uses a Red Green Refactor cycle:

- Red Phase
  - In this phase the team must write a test on the behaviour that is about to be implemented.
  - The test must be written before any production code is done
- Green Phase
  - In this phase the team must write straightforward production code that makes the appropriate test case pass
  - The solution must be as simple as possible. Any duplicate or dirty code will be handled in the refactor phase.
- Refactor Phase
  - In this phase the team is allowed to change code.
  - The team must be hypercritical in this phase to ensure performance and clean code is achieved.

3.4 Test Levels

The purpose of test levels is to promote decomposition in terms of tests. This allows testing to be broken into levels which start at a the lowest level, unit testing, to the highest level, acceptance testing.

Level	Owner	Objective	Key areas of Testing
Unit	Development	Detect defects code in units	Functionality and resource utilization
System	Development	Detect defects in end-to-end communications	Functionality, resource utilization, performance, reliability, portability, and interoperability
System Integration	Development	Detects defects in unit interfaces and reduces data and work flow failures	Functionality, data quality, and compatibility
Acceptance	Business	Demonstrate that the product works as expected in the target environment and detects defects in user work flow	Functionality in context of target environment

Table 1: Test Level Description

3.5 Guiding Principles

This section sets out to ensure that each phase in the testing process is performed effectively but providing an overview of each phase that is followed.

3.5.1 Test Planning

The planning phase ensures that all required resources and test cases are ready to be used in the design and execution phases.

3.5.2 Test Design

The design phase ensure that appropriate test cases are deigned in order to maximize test coverage of the application. These tests need to be designed, reviewed, and approved before testing execution can take place.

3.5.3 Test Execution

The execution phase is when the testing team run all tests that are set out in the above phases. An informal meeting should be held in order to clarify any concerns or problems prior to the execution of this phase.

3.5.4 Test Closure

The closure phase encapsulates the review of the execution phase in order to find any defects that occurred. These defects are then prioritized.

3.6 Approach to Automated Testing

Automated testing is done in two ways:

- Automatic Unit Testing
  - This is done using the Jasmine Framework and the Karma task runner.
- Automatic Integration Testing
  - This is done using TravisCI. Each time code is pushed to the repository TravisCI runs a series of tests ensuring that the newly added code is compatible and integrates with the rest of the system.

3.7 Test Priorities

Every test that is created and executed has a priority. These are used in order to manage task prioritization, specifically with regard to regression testing.

The priorities are classified as follows:

- High
  - This priority is used for tests that affect the core functionality of the system.
  - These must be resolved first.
- Medium
  - This priority is used for tests that, if failed do not severely impact the system.
  - These must be handled only after the high priority.
- Low
  - This priority is used for tests that represent ”nice-to-haves” and are not critically important.