University of Pretoria
Software Engineering - COS 301

Defendr Testing Policy

Dark nITes

12 October 2019

Team members:

MI (Muhammed) Carrim	R (Ruslynn) Apanna	SNL (Sisa) Khoza	CP (Christiaan) Opperman	J (Jeandre) Botha

1 Testing mission

To provide a testing methodology to effectively error free software product within the given time and cost constraint that fulfills the specified requirements as well as provide quality risk management information.

2 Test phases

Phase	Owner	Objective	Key areas of testing
Unit	Development	• Detect errors in units • Reduce risk of unit failures during run time	• Functionality • Recourse utilisation
Integration	Development	• Detect defects in unit interfaces • Reduce risk data-flow failures	• Functionality • Performance • Interoperability • Compatibility
Acceptance	Software client (Advance)	• Ensures business criteria are met • Demonstrates that system works as intended in real world environment. • Detects problems and risks in user workflows.	• Pre-agreed acceptance criteria • Functionality in context of normal usage

Figure 1: Testing phases

The use of the various test phases (also known as levels) is to promote the mitigation of quality risk as soon as possible and to the highest possible extent. The table above describes the various test phases.

3 Evaluation of testing

We will evaluate the quality and efficiency of our testing at each stage in development life cycle including the transition to the live product. Testing will be measured in various methods including the monitoring of the live system as well as the quality of the system at each stage as well as formal and informal reviews of our current testing strategy at our weekly team meetings.

4 Approach to testing improvement

We will constantly be reviewing the effectiveness and quality of our testing as it will directly contribute to the quality of the software system. We will then improve on our current methods as we explore new avenues and discover new and better ways of doing things. We are using the Exploratory testing method. In conjunction with an Agile development methodology, exploratory allows the tests to change alongside requirements as newer and better methods for testing and deploying are found. A greater emphasis is placed on the client's expectations. The business need may evolve with environment, and a solution as specified may not be enough. Producing a solution that fits the client's requirements is key, and as such a methodology that allows for a greater freedom of movement is beneficial.

5 Testing technology

At this stage we are using custom built unit tests in python

6 Testing process

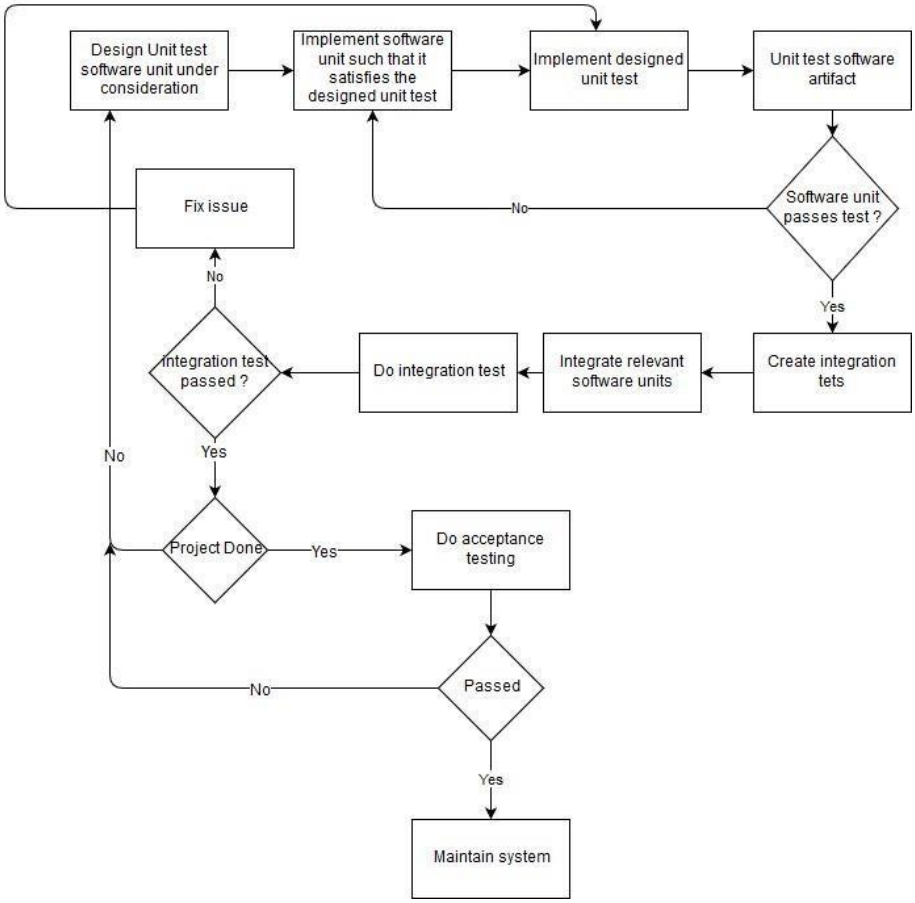

Figure 2: Testing process

7 Types of testing

Once a unit has been designed and is considered for use, tests on the software are run to ensure integrity. These tests focus both on the individual function as well as with the related functions.

7.1 – Unit tests

Unit tests make considerations based on a function's capability to perform their task individually. This is a test with the smallest possible scope and, typically consists of input(s) and an output.

An integral unit test for this phase was to ensure that ensure that the function handling blacklisting of IPs is working properly. This test was performed on the blacklist function; the test randomly generates IPs then blacklists them. The blacklist map is then queried to ensure it they are present. The results are also printed to a log file.

7.2 – Integration tests

Once a function has passed its unit test, it gets considered for integration. For a test to succeed in its integration test, the test must produce expected output from a range of efforts from multiple units. We regard integration testing as a cornerstone test as the system, in its entirety, is required for proper performance. Many of these tests considered the GUI, as it is the interface for the system. Tests would also cover both local and remote resources.

A test considered whether C controller, that allows interfacing with XDP functions, is executing correctly. This test depended on successful integration between the GUI, in Python, and the XDP program, in C. These functions included loading the XDP program or manipulating IP maps. A pass was awarded if a proper connection between XDP and GUI was present.

A test consisting of the Python database controller, the GUI and database was done. The test would require an IP to be manually blacklisted via the GUI. Said IP was then queried from the database. The test was a success if the IP was present. This validated a successful communication interaction between interface and remote database.

Another test case considered the GUI was behaving as intended when blacklisting an IP from the interface. The test would require the XDP controller as well as the XDP program and GUI. The test would need an IP to be enlisted into the blacklist, via the GUI, and then queried from the blacklist map. If the IP was present, a successful integration between GUI and XDP via the controller was established.

A test evaluating the correctness of removing an IP from the blacklist, checking XDP, the blacklist map and GUI were done. The test would nominate an IP to be removed from the blacklist map. The controller would then issue the command to XDP, which in turn would remove the IP. A query to the XDP program would then confirm the correctness of the interactions. A success was adopted if the IP was not listed in the maps.