

A Project Report On

# **Software Testing and Quality Assurance (Mini Project I)**

SUBMITTED BY

**Pranav Kulkarni**  
**Maitraya Kakade**  
**Nachiket Erlekar**

**Roll No:41430**  
**Roll No: 41427**  
**Roll No: 41434**

**CLASS: BE-4**

GUIDED BY  
**Prof S.D. Kale**



**DEPARTMENT OF COMPUTER ENGINEERING**  
**PUNE INSTITUTE OF COMPUTER TECHNOLOGY**  
DHANKAWADI, PUNE-43

SAVITRIBAI PHULE PUNE UNIVERSITY  
2020-21

## **Title:**

Create a small web-based application by selecting relevant system environment/platform and programming languages. Narrate concise Test Plan consisting features to be tested and bug taxonomy. Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools. Prepare Test Reports based on Test Pass/Fail Criteria and judge the acceptance of application developed.

## **Problem Definition:**

Perform Desktop Application testing using unittest library in python.

## **Objective**

We are going to learn how to Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios.

Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools.

Prepare Test Reports based on Test Pass/Fail Criteria.

## **Theory**

### **Test Plan for Application Testing**

The Test Plan document is derived from the Product Description, Software Requirement Specification SRS, or Use Case Documents. The focus of the test is what to test, how to test, when to test, and who will test. Test plan document is used as a communication medium between test team and test managers.

A standard test plan for Application Testing should define following features;

- Define the scope of testing
- Define objective of testing
- Approach for testing activity
- Schedule for testing
- Bug tracking and reporting

**UNIT TESTING :** UNIT TESTING is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

**INTEGRATION TESTING :** INTEGRATION TESTING is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing

### **Extreme Programming & Unit Testing**

Unit testing in Extreme Programming involves the extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to extreme programming, but they are essential to it. Below we look at some of what extreme programming brings to the world of unit testing:

- Tests are written before the code
- Rely heavily on testing frameworks
- All classes in the applications are tested
- Quick and easy integration is made possible

### **Unittest library python**

The unittest unit testing framework was originally inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

## **Application Tested :**

A bookstore application with a simple graphical user interface (**GUI**) is built to support all 4 essential CRUD functionalities. These operations are tested using some test cases.

## Test Plan :

The main aim is to check if we are getting correct output for each given input and if the test is being executed correctly .

- We start with giving varied inputs to perform CRUD operations on database.
- Test cases include authentication for different values for Author, Book name, ISBN code and publishing year fields for our database.
- For this we test our file, a separate tab opens up wherein we write our test code for the different operations, assertEquals is used here.
- Finally we get the test reports, which indicate the overall performance of our tests, i.e. either test pass or test fail.

## SAMPLE UNIT TESTS:

```
def test_nonNumericYear_INSERT(self):
```

```
    flag = database.insert("bookt","i am auth","abcd","IBN000df")
```

```
    self.assertFalse(flag,"Non Numeric year should not be allowed.")
```

```
def test_floatYear_INSERT(self):
```

```
    flag = database.insert("bookt","i am auth","1967.45","IBN000df")
```

```
    self.assertFalse(flag,"Float year should not be allowed.")
```

```
def test_emptyFields_INSERT(self):
```

```
    flag = database.insert("", "", 1920, "")
```

```
    self.assertFalse(flag,"Incomplete or blank arguments in insert function are not valid.")
```

```
def test_shortTitle_INSERT(self):
```

```
    flag = database.insert("b","i am auth","1967","IBN000df")
```

```
    self.assertFalse(flag,"Title less than 2 letters should not be allowed.")
```

# SAMPLE INTEGRATION TESTS:

```
def test_INSERT(self):
```

```
    flag = self.wrapper.insert(self.title,self.author,self.year,self.isbn)
```

```
    self.assertTrue(flag,"INSERT operation failed.")
```

```
    print("INSERT integration passed.")
```

```
def test_SEARCH(self):
```

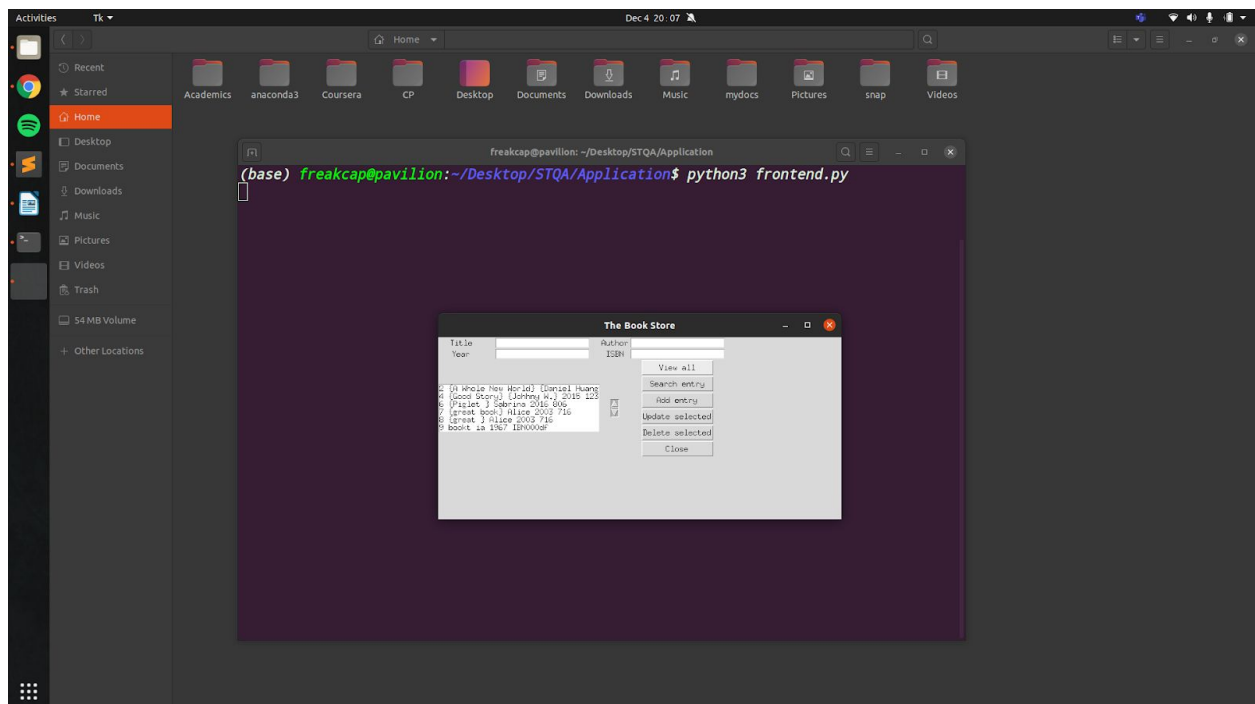
```
    flag = self.wrapper.search(self.title,self.author,self.year,self.isbn)
```

```
    self.assertTrue(flag,"SEARCH operation failed.")
```

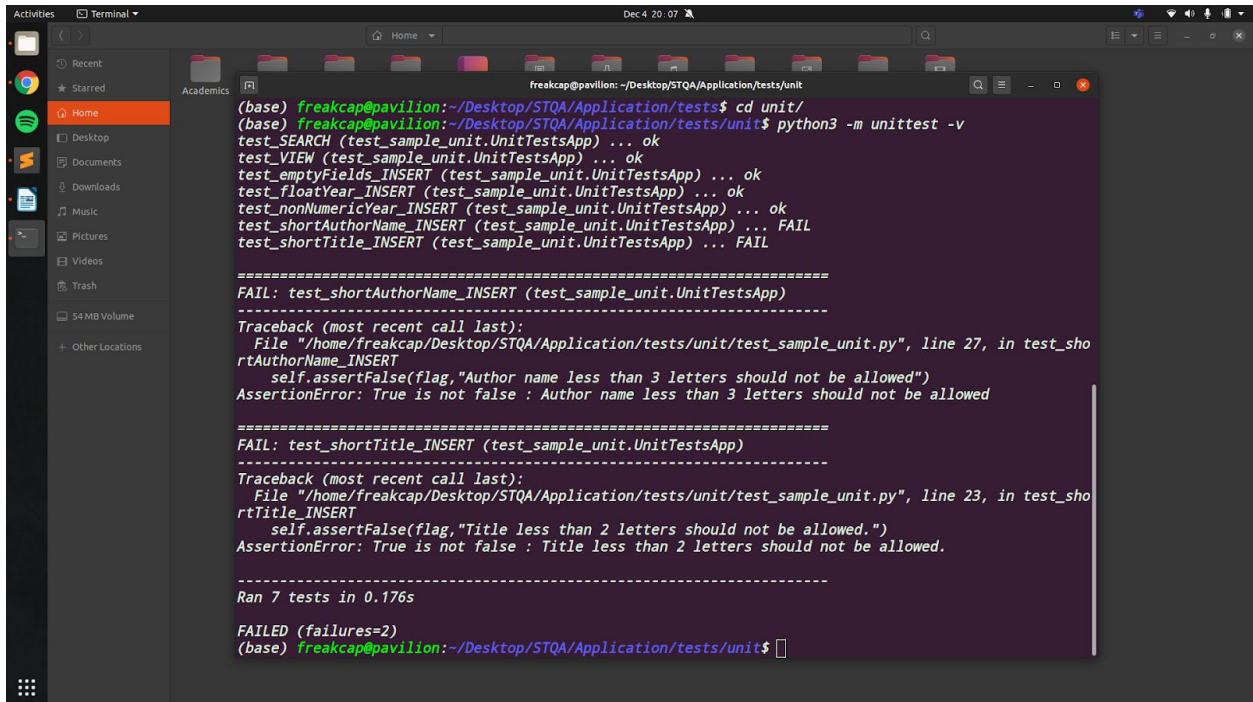
```
    print("SEARCH integration passed.")
```

## Screenshots of application

### 1. GUI of the application



## 2 . Execution of unit tests



```
(base) freakcap@pavilion:~/Desktop/STQA/Application/tests$ cd unit/
(base) freakcap@pavilion:~/Desktop/STQA/Application/tests/unit$ python3 -m unittest -v
test_SEARCH (test_sample_unit.UnitTestsApp) ... ok
test_VIEW (test_sample_unit.UnitTestsApp) ... ok
test_emptyFields_INSERT (test_sample_unit.UnitTestsApp) ... ok
test_floatYear_INSERT (test_sample_unit.UnitTestsApp) ... ok
test_nonNumericYear_INSERT (test_sample_unit.UnitTestsApp) ... ok
test_shortAuthorName_INSERT (test_sample_unit.UnitTestsApp) ... FAIL
test_shortTitle_INSERT (test_sample_unit.UnitTestsApp) ... FAIL

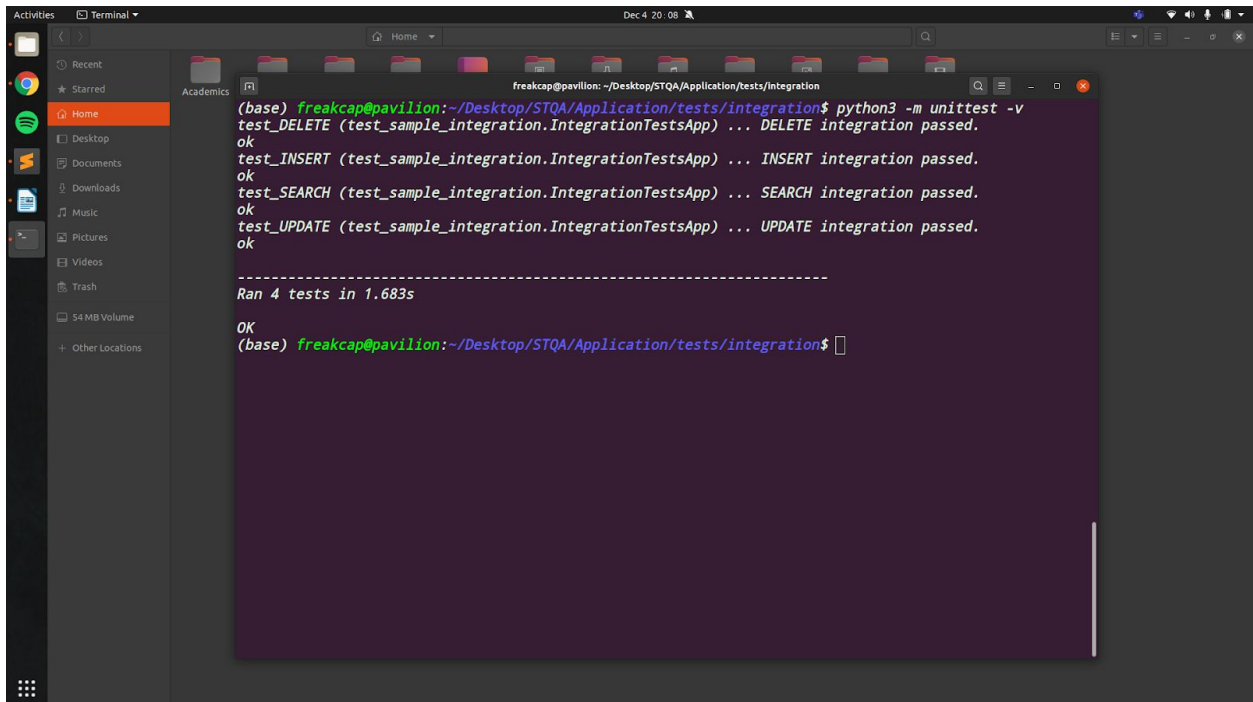
=====
FAIL: test_shortAuthorName_INSERT (test_sample_unit.UnitTestsApp)
-----
Traceback (most recent call last):
  File "/home/freakcap/Desktop/STQA/Application/tests/unit/test_sample_unit.py", line 27, in test_shortAuthorName_INSERT
    self.assertFalse(flag, "Author name less than 3 letters should not be allowed")
AssertionError: True is not false : Author name less than 3 letters should not be allowed

=====
FAIL: test_shortTitle_INSERT (test_sample_unit.UnitTestsApp)
-----
Traceback (most recent call last):
  File "/home/freakcap/Desktop/STQA/Application/tests/unit/test_sample_unit.py", line 23, in test_shortTitle_INSERT
    self.assertFalse(flag, "Title less than 2 letters should not be allowed.")
AssertionError: True is not false : Title less than 2 letters should not be allowed.

-----
Ran 7 tests in 0.176s

FAILED (failures=2)
(base) freakcap@pavilion:~/Desktop/STQA/Application/tests/unit$
```

## 3. Execution of integration tests



```
(base) freakcap@pavilion:~/Desktop/STQA/Application/tests/integration$ python3 -m unittest -v
test_DELETE (test_sample_integration.IntegrationTestsApp) ... DELETE integration passed.
ok
test_INSERT (test_sample_integration.IntegrationTestsApp) ... INSERT integration passed.
ok
test_SEARCH (test_sample_integration.IntegrationTestsApp) ... SEARCH integration passed.
ok
test_UPDATE (test_sample_integration.IntegrationTestsApp) ... UPDATE integration passed.
ok

-----
Ran 4 tests in 1.683s

OK
(base) freakcap@pavilion:~/Desktop/STQA/Application/tests/integration$
```

## Conclusion :

Hence, we have successfully performed Unit and Integration testing on a Bookstore application performing CRUD operations.