Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 5, Issue. 1, January 2016, pg.291 – 299

Investigation into Automated Testing Implementation Using MLUNIT FRAMEWORK IN MATLAB

Shweta [1], **Nikita Nain**[2]

1. M.Tech student 2. H.O.D. , IIET (Jind)

Abstract- A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Introduction to Automation Testing

Automation Testing means using an automation tool to execute your test case suite. The automation software can also enter test data into the System under Test, compare expected and actual results and generate detailed test reports.

Test Automation demands considerable investments of money and resources. Successive development cycles will require execution of same test suite repeatedly. Using a test automation tool it's possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. This improved ROI of Test Automation.

Goal of Automation is to reduce number of test cases to be run manually and not eliminate manual testing all together.

Test automation may be able to reduce or eliminate the cost of actual testing. A computer can follow a rote sequence of steps more quickly than a person, and it can run the tests overnight to present the results in the morning. However, the labor that is saved in actual testing must be spent instead authoring the test program. Depending on the type of application to be tested, and the automation tools that are chosen, this may require more labor than a manual approach. In addition, some testing tools present a very large amount of data, potentially creating a time consuming task of interpreting the results.

Things such as device drivers and software libraries must be tested using test programs. In addition, testing of large numbers of users (performance testing and load testing) is typically simulated in software rather than performed in practice.

Selection of Automation Tool

Selecting the right tool can be a tricky task. Following criterion will help you select the best tool for your requirement-

- Environment Support
- Ease of use
- Testing of Database
- Object identification
- Image Testing
- Error Recovery Testing
- Object Mapping
- Scripting Language Used
- Support for various types of test - including functional, test management, mobile, etc...
- Support for multiple testing frameworks
- Easy to debug the automation software scripts
- Ability to recognize objects in any environment
- Extensive test reports and results
- Minimize training cost of selected tools

Tool selection is one of biggest challenges to be tackled before going for automation. First, identify the requirements, explore various tools and its capabilities, set the expectation from the tool and go for a Proof of Concept.

Framework in Automation

A framework is set of automation guidelines which help in

- Maintaining consistency of Testing
- Improves test structuring
- Minimum usage of code
- Less Maintenance of code
- Improve re-usability
- Non Technical testers can be involved in code
- Training period of using the tool can be reduced
- Involves Data wherever appropriate

Types of Framework

There are four types of framework used in software automation testing:

1. Data Driven Automation Framework
2. Keyword Driven Automation Framework
3. Modular Automation Framework
4. Hybrid Automation Framework.

Benefits of automated testing

Following are benefits of automated testing:

- 70% faster than the manual testing
- Wider test coverage of application features
- Reliable in results
- Ensure Consistency
- Saves Time and Cost
- Improves accuracy
- Human Intervention is not required while execution
- Increases Efficiency
- Better speed in executing tests
- Re-usable test scripts
- Test Frequently and thoroughly
- More cycle of execution can be achieved through automation
- Early time to market

Proposed Implementation

Software engineering Automated Software Testing for Matlab Software testing can improve software quality. To test effectively, scientists and engineers should know how to write and run tests, define appropriate test cases, determine expected outputs, and correctly handle floating-point arithmetic.

Using Matlab mlUnit automated testing framework, scientists and engineers using Matlab can make software testing an integrated part of their software development routine.

1. **Write Unit Tests**

   Assemble test methods into test-case classes

Script-Based Unit Tests

- Write Script-Based Unit Tests

Function-Based Unit Tests

- Write Function-Based Unit Tests
- Write Simple Test Case Using Functions
- Write Test Using Setup and Teardown Functions

Class-Based Unit Tests

- Author Class-Based Unit Tests in MATLAB
- Write Simple Test Case Using Classes
- Write Setup and Teardown Code Using Classes
- Tag Unit Tests
- Write Tests Using Shared Fixtures
- Create Basic Custom Fixture
- Create Advanced Custom Fixture
- Create Basic Parameterized Test
- Create Advanced Parameterized Test

2. **Run Unit Tests**
   Run test suites in the testing framework

   - All tests in a package
   - All tests in a class
   - All tests in a folder

3. **Analyze Test Results**

    - Analyze Test Case Results
    - Analyze Failed Test Results

Implementation of Automated testing in matlab using mlunit

mlunit originally began as an update to mlUnit (http://sourceforge.net/projects/mlunit/), also available from MATLAB Central file exchange. The purpose was to add support for the new "classdef" style classes in MATLAB 2008a. Creating tests involves subclassing a class named TestCase, then adding methods whose names begin with "test". Inside each method you can use the inherited validation methods (assert, assertEquals, assertNotEquals) to check for success or failure. All tests are run automatically and their results recorded and reported after the run.

Testing Fibonacci function

We all know the Fibonacci series

0 1 1 2 3 5 8 13

In which we always consider the sum of last two number at third location

Loc1 0

Loc2 1

Loc3 1(Loc1 + Loc2)

Loc4 2(Loc2 + Loc3)

Loc5 3(Loc3 + Loc4)

Loc6 5(Loc4 + Loc5)

Loc7 8(Loc5 + Loc6)

Loc8 13(Loc6 + Loc7)

fib(x)

```
function y = fib(x)

% Simple queue implementation of Fibonacci function..

if x < 0 || (int64(x) ~= x)

   error('invalid input: please input only non-negative integers.');
```



```
end

if x < 2, y = x; return; end

q = [0 1];

for k = 2:x

   q = [q sum(q)];

   q(1) = [];

end

y = q(2);
```

when we call fib function

```
>> fib(6)

ans =

     8
```

When assert is used with fib function

```
>> assert_equals(1, fib(2));
>> assert_equals(0, fib(2));
??? Error using ==> mlunit_fail at 34
Data not equal:
  Expected : 0
  Actual   : 1

Error in ==> abstract_assert_equals at
   mlunit_fail(msg);

Error in ==> assert_equals at 42
abstract_assert_equals(true, expected,
```

Testing fib using mlunit

test_fib.m

```
function self = test_fib(name)
%test_fib constructor.
%
%  Class Info / Example
```



```
%  ====================
%  The class test_fib is the fixture for all tests of the test-driven
%  Fibonacci. The constructor shall not be called directly, but through
%  a test runner.


tc = test_case(name);
self = class(struct([]), 'test_fib', tc);
```

test_null.m

```
function self = test_null(self)
%test_null checks, whether the return value of fib(0) is 0.

assert_equals(0, fib(0));
```

test_value.m

```
function self = test_value(self)
%test_value tests different values of the fibonacci function (y = fib(x)).

assert_equals(1, fib(1));
assert_equals(1, fib(2));
assert_equals(2, fib(3));
assert_equals(3, fib(4));
assert_equals(5, fib(5));
assert_equals(8, fib(6));
assert_equals(13, fib(7));
assert_equals(21, fib(8));
assert_equals(34, fib(9));
assert_equals(55, fib(10));
```

test_value1.m

```
function self = test_value1(self)
%test_value1 tests different values of the fibonacci function (y = fib(x)).

assert_equals(0, fib(1));
```


After Running mlunit test result will be as follow

```
>> mlunit_test

---------------------------------------------------------
mlUnit 1.6.8
Started: 2015-04-23 14:33:45.
Test directory: C:\Users\aa\Documents\MATLAB\mlunit\test
---------------------------------------------------------

Running suite @test_fib

  test_value1 FAIL:
    Data not equal:
      Expected : 0
      Actual   : 1
    In test_value1.m at line 10
```

after testing following xml file is creating representing failures, errors, testcases, time take to test

TEST-@test_fib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="@test_fib" errors="0" failures="1" tests="3" time="0.428" hostname="unknown" timestamp="2015-04-23T14:33:46">
  <properties/>
  <testcase classname="@test_fib" name="test_null"/>
  <testcase classname="@test_fib" name="test_value"/>
  <testcase classname="@test_fib" name="test_value1">
    <failure><![CDATA[Data not equal:
  Expected : 0
  Actual   : 1
In <a href = "matlab:opentoline ('C:\Users\aa\Documents\MATLAB\mlunit\test\@test_fib\test_value1.m',10)">test_value1.m</a > at line 10]]></failure>
  </testcase>
  <system-out/>
  <system-err/>
</testsuite>
```


References

1. Artem, M., Abrahamsson, P., & Ihme, T. (2009). Long-Term Effects of Test-Driven Development A case study. In: Agile Processes in Software Engineering and Extreme Programming, 10th International Conference, XP 2009,. 31, pp. 13-22. Pula, Sardinia, Italy: Springer.

2. Bach, J. (2000, November). Session based test management. Software testing and quality engineering magzine(11/2000), (http://www.satisfice.com/articles/sbtm.pdf).

3. Bach, J. (2003). Exploratory Testing Explained, The Test Practitioner 2002, (http://www.satisfice.com/articles/et-article.pdf).

4. Bach, J. (2006). How to manage and measure exploratory testing. Quardev Inc., (http://www.quardev.com/content/whitepapers/how_measure_exploratory_testing.pdf).

5. Basilli, V., & Selby, R. (1987). Comparing the effectiveness of software testing strategies. IEEE Trans. Software Eng., 13(12), 1278-1296.

6. Berg, B. L. (2009). Qualitative Research Methods for the Social Sciences (7th International Edition) (7th ed.). Boston: Pearson Education.

7. Bernat, G., Gaundel, M. C., & Merre, B. (2007). Software testing based on formal specifications: a theory and tool. In:Testing Techniques in Software Engineering, Second Pernambuco Summer School on Software Engineering. 6153, pp. 215-242. Recife: Springer.

