© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 03 | 17 | April-June 2016
Paper is available at www.jrps.in | Email : info@jrps.in

Implementation of Object Oriented Automated Testing in Matlab using MLUnit Tool

[1]Manoj, Department of CSE RN Engineering College, Computer Science Engineering
[2]Ms. Pooja Ahlawat, Department of CSE , Assistant Professor, RN Engineering College

Abstract: *Validation and verification of the code is required because of ever-increasing complexity of embedded software applications, and the emergence of safety critical applications. Several embedded software development groups are using models and doing up front engineering before testing on final product to address this need. Use of old style of testing late in the development cycle resulted in very expensive release cycles.*

I. Introduction[1]

Object-oriented technology has become more and more popular in several various contexts. The Object-oriented paradigm is applied in the areas of programming languages, user interfaces, databases, design and specification methodologies.

OOPS based languages are widely applied in industry, and several commercial applications are developed and designed and with object oriented technology.

Object-oriented software quality has undergone a rapid change during the last years as a consequence, the attitude towards
Several analysis and design methodologies state that a well-designed object-oriented system would only need minimal testing. The object oriented paradigm has been considered powerful enough to assure software quality without any additional effort.

It is not enough to guarantee the quality of software products although object-orientation enforces many important programming principles, such as modularity, encapsulation, and information hiding,

Object oriented software contains errors just like traditional code it is known to both practitioners and researchers. Due to their peculiarities object oriented systems present new and different problems with respect to traditional programs.

II. Quality Assessment

Research addressing quality assessment lead to the definition of specific object-oriented metrics. These metrics provide quality indicators for identifying parts of the system which are more likely to be error-prone.
Quality of object-oriented software has been addressed from two different viewpoints, namely, quality assessment and quality achievement in the last years,

When the level of quality of a class, a cluster of classes, or a system is inadequate, we need a way of

improving it, Quality assessment methods are complementary to quality achieving techniques. As far as quality achievement is concerned, it is possible to identify two main approaches:
Methodology based: These methodologies pay little attention to verification of the developed system, according to the underlying hypothesis that a suitable application of the methodology should lead to well designed systems, which are easy to maintain.
This methodology involves using techniques and methodologies that aim at improving the software development process and specifically address the analysis, design, and development of object-oriented systems.

Verification based: using static or dynamic analysis techniques that targets revealing faults. The underlying idea is that, despite the effectiveness of the process, human beings are error-prone and program will always contain faults. Examples of static analysis techniques are formal proofs of correctness and code inspections and testing techniques are examples of dynamic techniques.

III. Benefits of OOPS

The object-oriented paradigm introduces novel aspects that have to be specifically addressed while

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 03 | 17 | April-June 2016
Paper is available at www.jrps.in | Email : info@jrps.in

sharing some commonalities with traditional programming languages,

Inheritance, encapsulation and data hiding raise visibility problems imply incremental testing concerns, and polymorphism and dynamic binding introduce undesirability related issues. The structure of object-oriented software is different from that of traditional codes.

In object-oriented codes, procedures (methods) tend to be small and well understood. The complexity tends to move from within code modules to the interfaces between them. Testing at the unit level tends to be less complex in the object-oriented case than for traditional procedural systems, and integration testing becomes necessarily more expensive as a consequence.

IV. Automated Testing: Process, Planning, Selection of tools[2]

Manual testing is performed by a human in front of a computer carefully executing the test steps. Using an automation tool to execute your test case suite is Automation Testing.

The automation software can also enter test data into the System under Test, compare expected and actual results and generate test reports.

Test Automation demands considerable investments of money and resources. Successive development cycles will require execution of same test suite again and again.

Using a test automation tool it's possible to record this test suite and re-play it as required. No human intervention is required once the test suite is automated. This improved ROI of Test Automation.

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

V. Benefits of Automated Testing

Automated testing is essential due to following reasons:

- Manual Testing is time and cost consuming
- It's **difficult to test for multi lingual sites manually**
- Automation does **not** need **Human intervention**. You can run automated test unattended (overnight)
- Automation **boosts speed of test execution**
- Automation helps **boosting Test Coverage**
- **Manual Testing** can become **boring** and **error prone.**

Test Cases to Automate

Test cases to be automated can be selected using the following criterion to increase the automation ROI

- **High Risk** - Business **test cases**
- Test cases that are **executed again and again**
- Test Cases that are very difficult to perform manually
- Test Cases are **time consuming**

The following category of test cases are not suitable for automation:

- **Test Cases that are newly designed** and not executed manually at least once
- **Test Cases** for which the **requirements are changing frequently**
- **Test cases** which are executed on **ad-hoc basis**.

VI. Implementation of Class in MATLAB[5]

Classification systems and design patterns enable engineers and scientists to make sense of complex systems and to reuse efforts by others.

Object-oriented programming (OO) applies to software development the standard science and engineering practice of identifying patterns and defining a classification system describing those patterns.

The OO approach improves your ability to manage software complexity—particularly important when developing and maintaining large applications and data structures by applying classification systems and design patterns to programming,

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 03 | 17 | April-June 2016
Paper is available at www.jrps.in | Email : info@jrps.in

Class[3]

classdef Syntax
Class definitions are blocks of code that are denoted by the classdef keyword at the beginning and the end keyword at the end. Files can contain only one class definition.
The following diagram shows the syntax of a classdef block. Only comments and blank lines can precede the classdef key word.

Sample code to define class

```
classdef clas1
   properties
      x
   end
   methods
      function p=sq(obj)
        p= obj.x*obj.x
      end
    end
   end
```

when we run above code then result is as follow

Create object of class

```
>> y=clas1

y =

        clas1
```

Assign value of property

```
>> y.x=9

y =

        clas1
```

accessing member function of class and passing object as parameter

```
>> sq(y)

p =

   81


ans =

   81

>>
```

VIII. Testing using assert keyword

```
assert_equals(81,sq(y))

p =

   81
```

Testing by passing wrong value

```
assert_equals(82,sq(y))

p =

   81


??? Error using ==> mlunit_fail at 34
Data not equal:
  Expected : 82
  Actual   : 81

Error in ==> abstract_assert_equals at 115
   mlunit_fail(msg);

Error in ==> assert_equals at 42
abstract_assert_equals(true, expected, actual,
varargin{:});
```

VII**. Creating Test Case for MLUnit [4]**

test_cl1.m

```
function self = test_cl1(name)
%test_cl1 constructor.
%
%  Class Info / Example
%  ====================
%  The class test_cl1 is the fixture for all tests of test-
driven
%  cl1. The constructor shall not be called , but
through
%  a test runner.
tc = test_case(name);
self = class(struct([]), 'test_cl1', tc);
```

test_v1

```
function self = test_v1(self)
```

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 03 | 17 | April-June 2016
Paper is available at www.jrps.in | Email : info@jrps.in

```
y=clas1;
y.x=9;

assert_equals(81,sq(y))
assert_equals(80,sq(y))
```

Output :

```
Running suite @test_cl1


p =

    81


p =

    81

  test_v1 FAIL:
    Data not equal:
      Expected : 80
      Actual   : 81
    In test_v1.m at line 6
```

Fig 2.

VIII. Conclusions

There is a major need for more upfront engineering in today's embedded software design process. Very little upfront testing has been done within the automotive area. With the introduction of executable modeling tools such as MLUnit this upfront testing is more feasible. It is the work of the tool vendors to make this testing technology available and practical to the user.

Reference

1.Object Oriented software testing by Devid C. Kung http://www.ecs.csun.edu/~rlingard/COMP595VAV/OOSWTesting.pdf

2. Automated Testing tools http://www.guru99.com/automation-testing.html

3. Matlab Documentation http://in.mathworks.com/help/matlab/matlab_oop/getting-familiar-with-classes.html

4.ML-Unit Matlab unit Test Framework

http://sourceforge.net/p/mlunit/mlunit/HEAD/tree/trunk/

5. Object Oriented programming in Matlab http://www.ce.berkeley.edu/~sanjay/e7/oop.pdf

6. 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.

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

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

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

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

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

12. 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.