© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | VOLUME: 06 ISSUE: 03 | JULY-SEPTEMBER 2015

Automated testing in Object oriented and Functional programming approach using Matlab

*Sarita R.N. College of education and technology
** Sangeeta (Asst. Prof. in CSE) R.N. College of education and technology

Abstract: Object-oriented languages are good when you have a fixed set of *operations* on *things*, and as your code evolves, you primarily add new things. This can be accomplished by adding new classes which implement existing methods, and the existing classes are left alone. Functional languages are good when you have a fixed set of *things*, and as your code evolves, you primarily add new *operations* on existing things. This can be accomplished by adding new functions which compute with existing data types, and the existing functions are left alone. Here in this paper we will compare object oriented and functional testing.

[I]Functional programming approach

In computer science, **functional programming** is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function *f* twice with the same value for an argument *x* will produce the same result *f(x)* each time. Eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional programming has its roots in lambda calculus, a formal system developed in the 1930s to investigate computability, the Entscheidung's problem, function definition, function application, and recursion.

Many functional programming languages can be viewed as elaborations on the lambda calculus. Another well-known declarative programming paradigm, *logic programming*, is based on relations.

[II] Object Oriented programming Approach

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as *attributes;* and code, in the form of procedures, often known as *methods.*

A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this").

In OO programming, computer programs are designed by making them out of objects that interact with one another.

There is significant diversity in object-oriented programming, but most popular languages are class-based, meaning that objects are instances of classes, which typically also determines their type.

Many of the most widely used programming languages are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative,

procedural programming. Significant object-oriented languages include Python, C++, Objective-C, Smalltalk, Delphi, Java, Swift, C#, Perl, Ruby and PHP.

[III] Real-world modeling and relationships

OOP can be used to associate real-world objects and processes with digital counterparts. However, not everyone agrees that OOP facilitates direct real-world mapping (see Criticism section) or that real-world mapping is even a worthy goal; Bertrand Meyer argues in *Object-Oriented Software Construction* that a program is not a model of the world but a model of some part of the world; "Reality is a cousin twice removed". At the same time, some principal limitations of OOP had been noted. For example, the circle-ellipse problem is difficult to handle using OOP's concept of inheritance.

However, Niklaus Wirth (who popularized the adage now known as Wirth's law: "Software is getting slower more rapidly than hardware becomes faster") said of OOP in his paper, "Good Ideas through the Looking Glass", "This paradigm closely reflects the structure of systems 'in the real world', and it is therefore well suited to model complex systems with complex behaviours".

Steve Yegge and others noted that natural languages lack the OOP approach of strictly prioritizing *things* (objects/nouns) before *actions* (methods/verbs). This problem may cause OOP to suffer more convoluted solutions than procedural programming.

[IV] Object Oriented Unit Testing

- smallest testable unit is the encapsulated class or object
- similar to system testing of conventional software
- do not test operations in isolation from one another
- driven by class operations and state behavior, not algorithmic detail or data flow across module interface

OO Integration Testing

- focuses on groups of classes that collaborate or communicate in some manner
- integration of operations one at a time into classes is often meaningless
- regression testing is important as each thread, cluster, or subsystem is added to the system
- thread-based testing
    - testing all classes required to respond to one system input or event
- use-based testing
    - test independent classes first
    - test dependent classes making use of them next
- cluster testing
    - groups of collaborating classes are tested for interaction errors

Object Oriented Validation Testing

- focuses on visible user actions and user recognizable outputs from the system
- validation tests are based on OOA
    - use-case scenarios
    - object-behavior model
    - event flow diagram
- conventional black-box testing methods can be used to drive the validation tests

OO Test Case Design

- Each test case should be uniquely identified and be explicitly associated with a class to be tested
- State the purpose of each test
- List the testing details for each test

OO Test Case Detail

- states to examine for each object involved
- messages and operations to exercised as a consequence of the test
- exceptions that may occur when the object is tested
- external conditions needed to be changed for the test
- supplementary information required to understand or implement the test

OO Test Design Issues

- White-box testing methods can be applied to testing the code used to implement class operations, but not much else
- Black-box testing methods are appropriate for testing OO systems

OOP Testing Concerns

- classes may contain operations that are inherited from super classes
- subclasses may contain operations that were redefined rather than inherited
- all classes derived from an previously tested base class need to be thoroughly tested

Interclass Test Case Design Multiple Class Testing

- for each client class use the list of class operators to generate random test sequences that send messages to other server classes
- for each message generated determine the collaborator class and the corresponding server object operator
- for each server class operator (invoked by a client object message) determine the message it transmits
- for each message, determine the next level of operators that are invoked and incorporate them into the test sequence

Interclass Test Case Design Behavior Model Testing

- test cases must cover all states in the state transition diagram
- breadth first traversal of the state model can be used (test one transition at a time and only make use of previously tested transitions when testing a new transition)
- test cases can also be derived to ensure that all behaviors for the class have been adequately exercised

[V] Function based testing

Code of Prime.m

```
function primalitytest = prime(n)
m = 2; % initialise factor to test
flag=0;
for m = 2:floor(sqrt(n))
   if mod(n,m) == 0 %m is a factor of n
      flag=1;
   end
end;
if(flag==1)
primalitytest='No';
else
primalitytest='Yes';

end
```

```
>> assert_equals('Yes',prime(3))
>> assert_equals('Yes',prime(4))
??? Error using ==> mlunit_fail at 34
Data not equal:
  Expected : 'Yes'
  Actual   : 'No'
  Changes  :  ^^^

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

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

[VI] Class based Testing

Testing of BasicClass.m

```
classdef BasicClass
  properties
    Value
  end
  methods
    function r = roundOff(obj)
      r = round([obj.Value],2);
    end
    function r = multiplyBy(obj,n)
      r = [obj.Value] * n;
    end
  end
end
```

Accessing method from class using object

Step1: create object of class.

Step2: assign value to instance variable of class using object.

Step3: access method from class and pass object as an argument.

```
>> x=BasicClass

x =

    BasicClass

>> x.Value=8

x =

    BasicClass

>> multiplyBy(x,4)

ans =

    32
```

Using assertion on object of class

```
>> assert_equals(32,multiplyBy(x,4))
>> assert_equals(22,multiplyBy(x,4))
??? Error using ==> mlunit_fail at 34
Data not equal:
  Expected : 22
  Actual   : 32

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

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

[VII] Conclusion

With the introduction of executable modeling tools 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.

References

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.