An Overview of Object Oriented Software Testability

N.Suresh
M.A.M Engineering College
Dept of computer science and Engineering
Trichy.

Abstract: - Building high quality and testable software is an essential requirement for software system. Software testability is a critical aspect during the software development life cycle. Software that is easily testable is known as testable software. Testability is an essential or distinctive aspect that is acquainted with the objective of predicting efforts needed for testing the program. Designing testability is a very important issue in software engineering. It is suggested to design software with high degree of testability. A program with high degree of testability illustrate that a selected testing criterion could be achieved with less effort and the existing faults can be revealed more easily during testing.
This paper gives the concept of software testability, previously defined by The IEEE standard Glossary**,** our measurement for testability and complexity and also shares our thought and understanding about the testability in the object oriented system. In this paper we have explained the concept of software testability and complexity in very sophisticated manner. The results are verified by suitable example and graph.

Keywords: Software Testing, Software Testability, Simplicity, Complexity.

1 Introduction

Necessity of Testing: In early days, application was manual but today mostly application are software based. So testing is very crucial. It is most crucial when human life is associated with the software in any sense. So our main emphasis will be on software quality and their reliability on the defined input and their desired output. So we need the effective testing to get adequate level of software quality and reliability. There are some reasons why do we test a system.

- Provide confidence in the system
- Identify areas of weakness
- Establish the degree of quality
- Establish the extent that the requirements have been met
- To prove it is both usable and operable

Methodology of Development: When we are going to develop a software application our main focus is on that which methodologies we use. There are lots of reasons for using object oriented methodologies as a benchmark for development of software system. Some reasons are as follows:

- Provide Reusability
- Reduce the complexity,
- Reduce the incidence of error
- OO has a unique feature, like inheritance, abstraction, information hiding, polymorphism, dynamic binding [2].

But studies mention that information hiding and abstraction can decrease the testability of software in object oriented methodologies [1]. The above mention feature also introduces some kind of error. And the testing issue in OO software is different from the conventional software testing. That why we cannot use the techniques of conventional software in the object oriented system.
It is impossible to conduct an exhaustive testing process for software product. To increase the quality process for software product we need.

- Economic test methods
- Effective test tools
- Software components with high testability.

How ever we are facing a dilemma that the complexity in the software system is growing rapidly while the testing resources are limited. To maximize the impact of testing, we need to design system so that their testability is apex.

This paper presents a testability and complexity analysis. Here testability shows that how much effort required for testing a system and complexity shows that how much system is complex for testing. The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives our view with example. In section 4 we review the benefits of the testability. Section 5 concludes paper.

2 Related Works

2.1 Software Testability

To get adequate level of software quality and reliability we used software testability as a quality attribute for determining the complexity and how much efforts required for testing to getting the desired results. Software testability has been defined and described in various literatures from different point of views.
The most common is the "*ease of performing testing* "[3].The IEEE Standard Glossary defines testability as the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met [3]. Testable software is one that can be tested easily, systematically and externally at the user interface level without any ad-hoc measure [4] [5].Robert V. Binder defines testability as "The relative ease and expense of revealing software faults [6]".

Testability is an important attribute to the Maintainability, Reliability, and Changeability of software. Testable software is affluent and less expansive to maintain.Testable software need two characteristics i.e. observability and controllability. Binder defines these two facets of testability succinctly [6]:

Observability: During the software testing process it is compulsion or requirement to observe the internal details of software execution, to diagnose errors which are discovered during this process. A system is said to be observable if the possible input state of the system can be observed Observable software makes it suitable for the tester to observe the internal behavior of the system.

Controllability: During the software testing process some of the condition is really difficult to test, like disk full, out of memory, resource failure, network link failure, communication failure etc. A system is said to be controllable if and only if the system states can be changed by changing the system input. Controllable software makes it possible to initialize the software to desired states, prior to the execution of various tests.

2.2 Testability Measurement

Several techniques have been made for development of meaningful testability [4, 7, 8] But here we are using the testability measurement techniques of John McGregor and S. Srinivas [10].
They mentioned that Testability of a method into the class depends upon the visibility component. Testability of method is

$$\text{ή=constant*(ζ)}$$

Where ζ is the visibility component

Testability of the class is

$$\text{θ=min (ή)}$$

The definition of the visibility component (VC) is

ζ= Possible Output/Possible Input

Inputs are as follows:

1. Number of explicit parameter in the method signature
2. Number of implicit parameter in the declared in the class

Outputs are as follows:

1. Explicit reference parameter in the method signature
2. Implicit parameter, object attribute in the class
3. Return value of the method
4. Any exception throws by the method.

2.3 Complexity Measurement

Cyclomatic complexity is software metric (measurement). It was developed by Thomas J. McCabe [11] and is used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to the commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command.
The cyclomatic complexity of a flow graph is as follows

$M = E - N + 2P$

Where

M = Cyclomatic complexity
E = Number of edges of the graph
N = Number of nodes of the graph
P = Number of connected components.

3. Examples

In this research our working definition of testability is “Testability of a program is a degree of simplicity of the program”. By using following example we are trying to understand the simplicity in the form of complexity. Means if system’s complexity is increase that means its simplicity decrease and the effort of testing (Testability) will increase.

For understanding and the analysis the role of complexity in the software testability we are taking examples of vending machine and coin dispenser. In these examples, in the first step we measure the testability by using the method of John McGregor and S. Srinivas [10]. Than in the second step we draw the Control flow graph and find the complexity of the program.

1.Vending Machine

```
1. public class VendingMachine
2. {
3. final private int COIN = 25;
4. final private int VALUE = 50;
5. private int totValue;
6. private int currValue;
7. private Dispenser d;
8. public VendingMachine()
9. {
10.totValue = 0;
11.currValue = 0;
12.d = new Dispenser();
13.}
14. public void insert()
15. {
16. currValue += COIN;
17. System.out.println("Current value = " +
currValue );
18. }
19. public void return()
20. {
21. if ( currValue == 0 )
22. System.err.println( "no coins to return" );
23. else
24. {
25. System.out.println("Take your coins");
26. currValue = 0;}
27. }
28. public void vend( int selection )

29. {
30. int expense;
31. expense = d.dispense( currValue, selection );
32. totValue += expense;
33. currValue -= expense;
34. System.out.println( "Current value = " +
currValue );
35. }
36. }
```

2. Coin Dispenser

```
1. public class Dispenser
2. {
3. final private int MAXSEL = 20;
4. final private int VAL = 50;
5. private int[] availSelectionVals =
{2,3,13};
6. public int dispense( int credit,
int sel )
7. {
8. int val=0;
9. if ( credit == 0 )
10.System.err.println("No coins
inserted");
11.else if ( sel > MAXSEL )
12.System.err.println("Wrong
selection "+sel);
13.else if ( !available( sel ) )
14.System.err.println("Selection
"+sel+" unavailable");
15.else
16.{
17.val = VAL;
18.if ( credit < val )
19.System.err.println("Enter "+(val-
credit)+" coins");
20.else
21.System.err.println("Take
selection"); }
22.return val;
23.}
24.private boolean available( int sel
)
25.{
26.for (int i = 0;
i<availSelectionVals.length; i++)
27.if (availSelectionVals[i] == sel)
28.return true;
29.else
30.return false;
31.}
32.}
```

Process

There are two examples first is vending machine and second one is coin dispenser. In both examples we do two stage processes, in first step we find out the testability [11] and in the second step we find the cyclometic complexity [12] of the program. Than we analyze that how complexity affects the testability of a program.

Table 1.Testability Analysis of Vending Machine.

S.No	Method Name	Visibility Component(ζ)	Method Testability(ή)	Class Testability (θ)
1	VendingMachine()	3/3=1	2*1=2	2
2	void insert()	3/3=1	2*1=2	
3	void return()	3/3=1	2*1=2	
4	void vend( )	4/4=1	2*1=2	

Table 2. Testability Analysis of Coin Dispenser

S.No	Method Name	Visibility Component(ζ)	Method Testability(ή)	Class Testability (θ)
1	int dispense()	5/4=1.25	1.25*2=2.5	2.5
2	private Boolean available(int sel)	4/3=1.33	1.33*2=2.66	

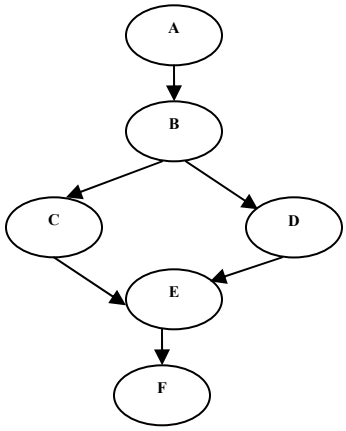

Fig. 1

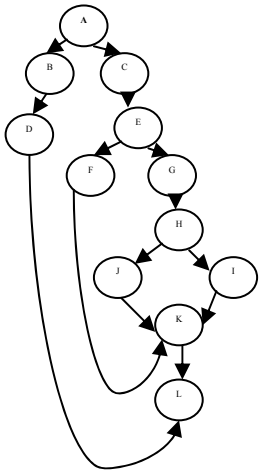

Fig.2

Fig 1 shows the flow graph of the vending machine, its complexity is 2 while the fig 2 shows the flow graph of coin dispenser and its complexity is 4 that means as the complexity of the program is increases as well as the testing effort is also increases. This result is verified by the graph. In fig 3 we show the graph which is drawn between complexity and the testability.

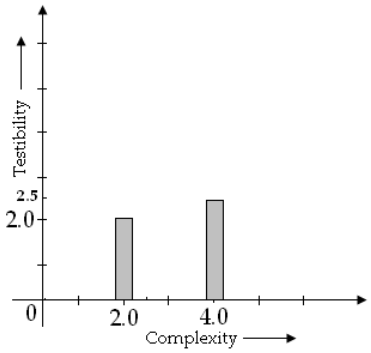

Fig.3

4. Benefits of Testability

There are a lot of other characteristics of design that are related to testability. In particular the lists below are benefits to testability [9].

- Understandability
- Modifiability
- Availability
- Flexibility
- Maintainability
- Reliability
- Usability
- Changeability
- Fault Tolerance

5. Conclusions

Software testability is an important factor during the software development life cycle. In this paper we are giving our view that testability is the degree of the simplicity of the program and it will increase as the complexity of the program will increases as shown in fig.3 and its complexity will depends on whole software development life cycle.

References:

[1]. Corporation, R.S.T., *Testability of Object-Oriented Systems*,.1995, National Institute of Standards and Technology: Gaithersburg, MD. Report Number: NIST GCR 95-675

[2]. J. E. Payne, R. T. Alexander, C. H. Hutchinson: "Design-for-Testability for Object Oriented Software", *Object Magazine, SIGS Publications Inc.,* vol. 7, no.5, 1997, pp. 34-43

[3]."IEEE standard Glossary of Software Engineering Terminology,"ANSI/IEEE Standard 610-12-1990, IEEE Press, New York,1990.

[4]. R. S. Freedman, "Testability of Software Components", *IEEE Transactions on Software Engineering,* vol. 17, No. 6, June 1991, pp. 553-563

[5]. S. C. Gupta, M. K. Sinha: "Improving Software Testability by Observability and Controllability Measures", *13th World Computer Congress,IFIP*, vol. 1, 1994, pp. 147-154

[6]. Binder, R.V., *Design for Testability with Object-Oriented Systems.* Communications of the ACM, 1994. 37(9): p. 87-101.

[7]. Voas, J.M., *PIE: a dynamic failure-based technique.* IEEE Transactions on Software Engineering, 1992. 18(8): p. 717-27.

[8]. Voas, J.M. and K.W. Miller, *Software Testability: The New Verification.* IEEE Software, 1995. 12(3): p. 17-28.

[9].Appendix D of Stefan Jungmayr's thesis Improving testability of object-oriented systems

[10]. A Measure of Testing Effort Conference on Object-Oriented Technologies Toronto, Ontario, Canada, June 1996.

[11]. T.J.McCabe, "A Complexity Measure", IEEE Tran. Software Eng., vol. SE-2, No.4, Dec.1976, pp. 308-320.