



EVALUATING TESTABILITY OF OBJECT ORIENTED SYSTEM

Mohit Chaudhary, Himali Chopra, Ritesh Rastogi

Faculty at NIET School of Computer Science & Applications

19, Knowledge Park-II, Institutional Area, Phase - II, Greater Noida, Uttar Pradesh, INDIA

Abstract

Edifice high quality and testable software is an crucial requirement for software system. Software testability is a significant aspect during the software development life cycle. Software that is easily testable is known as testable software. Testability is an indispensable or idiosyncratic aspect that is familiar with the objective of predicting efforts needed for testing the program. Designing testability is a very important concern in software engineering. It is recommended to design software with high degree of testability. A program with high degree of testability demonstrates that a preferred testing criterion could be achieved with less effort and the existing faults can be exposed more easily during testing.

Our dimension for testability and complexity and also shares our thought and perceptive about the testability in the object oriented system.

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

I. Introduction

Development in object –oriented (OO) languages and methodologies have helped in the design of better and modular software, thereby reducing the complexity and software development methodologies reduce the incidence of error, but the need to test the software remains.

As software applications grow more complex and become a necessity in almost every day activities, more emphasis has been placed on software quality and reliability.

To maximize the impact of testing, we need to design systems so that their testability is optimal. Software testability is an external software attribute that evaluates the complexity and the effort required for software testing.

In this paper our working definition of testability is “Testability of a program is a degree of simplicity of the program”. In this work 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.

A. Testability

Software testability is an external software attribute that evaluates the complexity and the effort required for software testing. Software testability has been defined and described in literature from different point of views. 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 [1].

A software system's testability is defined by the ISO model [9] as “attributes of software that bear on the effort needed to validate the software product.”

Testable software is one that can be tested easily, systematically and externally at the user interface level without any ad-hoc measure [7] [8].

There are a lot of other characteristics of design that are related to testability [5].

Observability

In the process of testing, there is a need to observe the internal details of software execution, to ascertain correctness of processing and to diagnose errors discovered during this process.

Controllability

Controllable software makes it possible to initialize the software to desired states, prior to the execution of various tests.

Representation

Requirements captures the need and exceptions of customers thus test find out whether the implementation is correct and complete.

A specification details define the architecture and design pattern that that are used to implements the requirements.

Implementation

Implementation refers to post-sales process of guiding a client from purchase to use of the software or hardware that was purchased. The major part of the implementation of an application consists of source code expressed in one or more programming languages. An implementation that is driven by an external interface can be hard to test.

Test Suite

A test suite often also contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

A test suite for a primality testing subroutine might consist of a list of numbers and their primality (prime or composite), along with a testing subroutine. The testing subroutine would supply each number in the list to the primality tester, and verify that the result of each test is correct.

Test Tools

Easy-to-use tools will demand less of the staff responsible for testing. Test case definition in the presence of graphical user interfaces is another example where tooling can significantly reduce the required effort.

A good set of test tools is capable of interoperating with related tools.

Process Capability

The Process Capability is a measurable property of a process to the specification, expressed as a process capability index or as a process performance index. Process capability compares the output of an in-control process to the specification limits by using capability indices.

II. Related Work

In the past few years, a number of methods have been proposed to measure and analyze the testability of software [2, 3,5]. They can be classified into the following groups:

- Program-based measurement methods for software testability [3];
- Model-based measurement methods for software testability [3,4];
- Dependability assessment methods for software testability [2].

A. Program Based Testability Measurement

Since a fault can lie anywhere in a program, all places in the source code are taken into consideration while estimating the program testability. J.-C. Lin et al. [3] proposed a program-based method to measure software testability by considering the single faults in a program.

B. Model Based Testability Measurement

Another measurement approach of software testability is proposed based on a well-defined model: such as a data flow model [3].

C. Dependency Based Testability Measurement

Clearly, the two previous approaches need program source code and/or a program-based model to support software testability measurement. A. Bertolino and L. Strigni [2] proposed a black-box approach, where the software testability measurement is performed based on the dependency relationships between program inputs and outputs.

III. Object-Oriented Technology

The fundamental idea behind Object –Oriented language is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object. Object Oriented means that we organize software as a collection of discrete objects that incorporate both data structure and behavior.

These characteristics are required by an object oriented technology.

- Identity
- Classification
- Inheritance
- Polymorphism

Identity

In object oriented approach every object is distinguished with other objects in other words we can say every object has a unique identity.

Classification

As real world, object oriented technology provide a concept of classification. Objects of same properties and behavior should be defined in same class. Encapsulation define this characteristic of object oriented system. "Encapsulation consists of separating the external aspects of an object, which are accessible to other objects from the internal implementation details of the object, which are hidden from other objects".

Inheritance

Inheritance is somewhat analogous to using functions to simplify a traditional procedural program. It supports the concept of hierarchical classification. In object oriented technology, the concept of inheritance provides the idea of reusability.

Polymorphism

Polymorphism, a Greek term, means the ability to take more than one form.

IV. Approach

In this research our working definition of testability is “Testability of a program is a degree of simplicity of the program”. In this work we are trying to understand the simplicity in the form of complexity.

A. Testability Measurement

Several techniques have been made for development of meaningful testability [7, 10, 11] but here we are using the testability measurement techniques of John McGregor and S. Srinivas [12]. They mentioned that Testability of a method into the class depends upon the visibility component. Testability of method is

$$\eta = \text{constant} * (\zeta) \text{ Where } \zeta \text{ is the visibility component}$$

Testability of the class is

$$\theta = \min(\eta)$$

The definition of the visibility component (VC) is

$$\zeta = \text{Possible Output/Possible Input}$$

Before doing implementation we are defining our input, output and constant for testability analysis work and also taking some assumption for this work.

Assumption:

- Not consider system parameter
- Consider only concrete class.
- All method overloading and over ridding allow.
- Not consider static method but treat public static void main as a starting point.
- Not consider abstract method.

The input, output and constant for the java class will be as follows

Input:

- All parameter into the class.
- Parameters pass into the method signature.
- All class method parameter of the parent class excluding system parameter.
- All method of interface implementation.

Output:

- The return value of the method
- Any exception either checked or unchecked by the method
- All implicit parameter & object attribute define in the class
- Object reference in the method signature.

Constant :

- Final
- Literal
- Static final variable is also effectively used as a constant

B. Complexity Measurement

Cyclomatic complexity is also known as structural complexity because it gives internal view of code. Cyclomatic complexity is software metric (measurement). It was developed by Thomas J. McCabe [6] and is used to measure the complexity of a program .This approach is used to find the number of independent paths through a program. Although it is possible to define a set of algebraic expressions that gives the total number of possible paths through a program however using total number of paths has been found to be impractical.

M McCabe’s cyclomatic metric $V(G)$ of a graph with n vertices , e edges and P connected components is

$$V(G) = e - n + 2P.$$

Where

$V(G)$ = Cyclomatic complexity

E = Number of edges of the graph

N = Number of nodes of the graph

P = Number of connected components

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.}

```

Step 1. Testability Analysis

Table 4.1

S.No	Method Name	Visibility Component(ζ)	Method Testability(η)	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	

Step 2. Cyclomatic Complexity

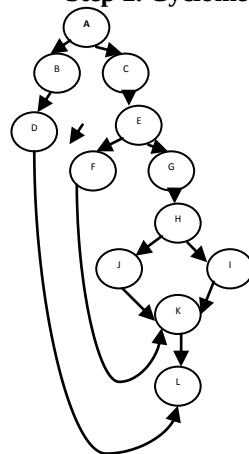


Figure 4.2 Control graph of coin dispenser

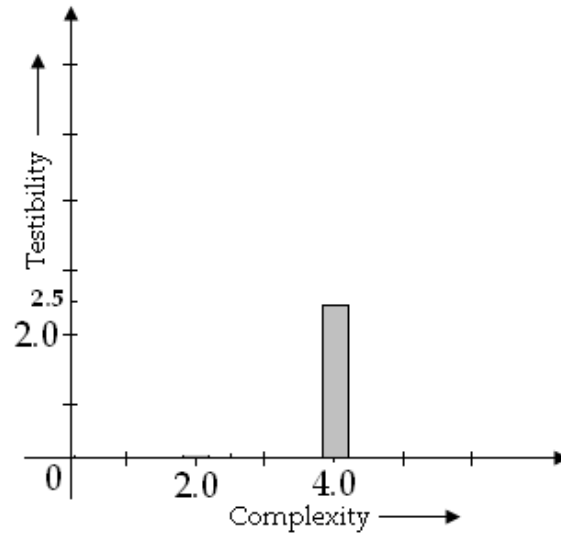


Figure 4.3 Testability and complexity graph of coin dispenser

V. Conclusions

We have a conclusion that testability is a worth factor that attempts to expect how much effort will be required for software testing and to estimate the difficulty of causing a liability to result in a collapse (called revealing the fault). Software testing is a decisive element in the software development life cycle and has the prospective to save time and money by identifying problems early and to improve customer contentment by delivering a more defect-free product.

Software testability is an significant factor during the software development life cycle. In this thesis we are giving our observation that testability is the degree of the ease of the program and it will increase as the intricacy of the program will increases and its intricacy will depends on whole software development life cycle.

References

1. J. Voas and K. Miller. Semantic metrics for software testability. *Journal of Systems and Software*, 20:207–216, March 1993.
2. Bertolino, A., and L. Strigini. —On the Use of Testability Measurement for Dependability Assessment, *IEEE Trans. on Software Engineering*, Vol. 22, No. 2, February 1996, pp. 97–108.
3. Lin, J.-C., I. Ho, and S.-W. Lin. —An Estimated Method for Software Testability Measurement, *Proc. of 8th International Workshop on Software Technology and Engineering Practice (STEP '97)*, 1997.
4. Robach, C., and Y. Le Traon. —Testability Analysis of Co-Designed Systems, *Proc. of 4th Asian Test Symposium (ATS'95)*, 1995.
5. Appendix D of Stefan Jungmayr's thesis Improving testability of object-oriented systems.
6. T.J.McCabe. —A Complexity Measure, *IEEE Tran*
7. R. S. Freedman, "Testability of Software Components", *IEEE Transactions on Software Engineering*, vol. 17, No. 6, June 1991, pp. 553-563.
8. 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.
9. B. van Zeist, P. Hendriks, R. Paulussen, and J. Trienekens. *Quality of Software Products*. Software Engineering Research Center, Utrecht, the Netherlands, 1996.
10. J. Voas, PIE: A dynamic failure-based technique, *IEEE Transactions on Software Engineering* 18 (8) (1992) 717–727.
11. J. Voas, K. Miller, Software testability: The new verification, *IEEE Software* 12 (1995) 17–28.
12. J. McGregor and S. Srinivas. A measure of testing effort. In *Proceedings of the Conference on Object-Oriented Technologies*, pages 129{142}. USENIX Association, June 1996.