# Prioritization Of Implementation Based Testing Technique For Class Under Test

Prof. Rajkumar V. Panchal
Assistant Professor,
Department of Computer Engineering,
VPCOE,Baramati
Pune University(MH), India
*e-mail: panchal.rajit@gmail.com*

Prof. Gyankamal J. Chhajed
Assistant Professor,
Department of Computer Engineering
VPCOE, Baramati
Pune University(MH), India
*e-mail: gjchhajed@gmail.com*

***Abstract***— The basic unit of testing in an Object-Oriented (OO) software is a class. The Object-Oriented (OO) classes' exhibits different characteristic. Due to this different characteristics of class testing become more difficult. To address the difficulty of testing the different features of a class a plethora of Implementation Based Testing Techniques (IBTTs) have been developed. However no one IBTT has been emerged as the accepted approach. So to find most suitable IBTT to test the class mapping of IBTTs to class is performed. This mapping process provides feedback to the tester describing the characteristics of class that are suitably tested by testing technique.
In this paper we describe prioritization of Implementation Based Testing Techniques. To give Priority to IBTTs the control flow of each method is captured, with the help of control flow, complexity of each method is calculated and used. Depending on the complexity of each method and its suitable testing technique priority is assigned to Implementation Based Testing Techniques.

***Keywords***- *Implementation Based Testing Techniques (IBTTs) ; Object-Oriented (OO) ; control flow ; complexity.*

*****

## I. INTRODUCTION

The trend in the development of large scale object oriented system has shifted towards testable, robust models with a focus on the prevention of faults and system failure. One process that supports the construction of robust software is testing. The widespread use of the Object-Oriented (OO) paradigm has lead many developers to treat the class or class cluster as the basic test unit in an OO system [12]. However object oriented classes exhibit different characteristics like abstraction, encapsulation, inheritance, polymorphism, concurrency and Exception handling complicate class-based testing.

To address the difficulty of testing the different features of a class a plethora of Implementation Based Testing Techniques (IBTTs) have been developed. However no one IBTTs has emerged as the accepted approach, so to find most suitable testing technique to test the class mapping of Implementation Based Testing Techniques to a class is performed. This mapping process provides feedback to the tester describing the characteristics of the class that are suitably tested by Implementation Based Testing Technique together with the characteristics that are not suitably tested by any of the testing technique.

In this paper we have proposed a technique for prioritizing the Implementation Based Testing Techniques. The priority is assigned to the Implementation Based Testing Techniques by using the complexity of each method and its suitable testing technique. To calculate the complexity of each method control flow of each method is extracted and complexity is calculated from this control flow. Due to prioritization of Implementation Based Testing Techniques number of entities to be analyzed by a tester when selecting testing techniques can be reduced by 30%.

In the next section we overview terminology and concepts about object technology and mapping of Implementation Based Testing Techniques. Section 3 describes the proposed approach. In section 4 we describe prioritization of Implementation Based Testing Techniques. Section 5 describes the Validation and Result Analysis. Section 6 describes the conclusion.

## II. BACKGROUND

This section describes overview terminology and concepts about object technology and mapping of Implementation Based Testing Techniques.

Meyer defines a class as a static entity that represents an abstract data type with a partial or total implementation [13]. The static description supplied by a class includes a specification of the features that each object will contain. These features fall into two Categories: (1) attributes and (2) routines. Attributes are referred to as data items and instance variables in other OO languages while routines are referred to as member functions and methods. Throughout this paper the terms attributes and routines are used. Each attribute and routine local (variable or parameter) of a class can be declared as one of many possible types. These types include: built in types, user defined types and types provided by specialized libraries. Some OO languages also permit the use of parameterized types also referred to as generics that allow the actual type of the attribute or routine local to be known when an object of the class is instantiated. Accessibility to class attributes and routines provides a class with the ability to deny its clients access to certain features. In Java this mechanism is implemented using the access specifiers private, protected and public.

Inheritance allows features of the class to be reused in another class and permits the class to be extended to include new features. Polymorphism is the ability of a feature (attribute or routine) of a class to have many forms; that is the form of the feature invoked is dependent on the type of the object instantiated. As a consequence of polymorphism features can be bound to the object at runtime referred to a dynamic binding.

### A. Implementation Based Testing Techniques

Implementation based testing of an object oriented class is defined as the process of operating a class under specified conditions, observing or recording the results and making an evaluation of the class. Testing techniques that generate test information based on the implementation of these techniques are refer as Implementation Based Testing Techniques (IBTTs) .

Table 1 summarizes the IBTTs. Column 1 of Table 1 identifies the main researcher that developed the IBTT described in the corresponding row and the name of the respective IBTT. Column 2 identifies the class characteristics that can be suitably tested by the respective IBTT listed in Column 1 and Column 3 the class characteristics not suitably tested by the respective IBTT listed in Column 1.The class characteristics listed in Columns 2 and 3 are deduced from the references cited in Column 1. Each row in the table delimited by a hard line represents the information pertaining to a given IBTT. For example Row 2 represents the information for the IBTT developed by Buy et al [4]. The name assigned to the IBTT developed by Buy et al. is Automated Column 1. Column 2 of Row 2 states that the class characteristics (Private Attributes, Primitive Types and Simple Control Flow) can be suitably tested by the Automated technique. Column 3 lists the characteristics (Recursive Protected Attributes and Public Attributes) that cannot be suitably tested by the Automated IBTT. Note that the set of class characteristics listed in Column 3 of Table 1 i.e. those characteristics not suited to an IBTT is not exhaustive. These are the Implementation Based Testing Technique but no one technique has emerged as the accepted approach because no single technique addresses all of the complication that class may possess. Thus, the developer is facing with an ever-expanding choice of testing approaches and strategies from which to choose. As there is no single technique that addresses all of the complication of classes there are many testing strategies that are appropriate for some kind of classes but inappropriate for other kind classes complicate the choice of testing for classes.

### B. Mapping IBTTs to Object Oriented Classes

The mapping process accepts a summary of the class under test and a list summarizing the IBTTs. The output of the mapping process consists of a list of 3-tuples representing those class characteristics that can be suitably tested by some IBTT. In addition the output of the mapping process provides feedback information informing the tester of any class under test (CUT) characteristic that cannot be suitably tested by any IBTT. Each 4-tuple output in the list of characteristics that can be suitably tested by some IBTT consists of: (1) a characteristic of the CUT (2) a list of feature pairs (3) a unique identifier of the IBTT and (4) a priority assigned to the IBTT by the tester.

| Researchers (IBTT Name) | Class Characteristics | |
|---|---|---|
| | Suited To | Not Suited To |
| Alexander et al. [3] (Polymorphic Relationships) | Primitive and user defined types, polymorphism, dynamic binding | Local variables of routines assigned return values |
| Buy et al. [4] (Automated) | Primitive types, simple control flow | Complex variables e.g. arrays, structs; references |
| Harrold et al. [5] (Incremental) | Inherited classes | Classes with no parents |
| Harrold et al. [6] (Data-Flow) | Primitive types, new attributes | Complex variables e.g. arrays, structs; references, polymorphism, dynamic binding |
| Koppol et al. [7] (Concurrent Programs) | Features exhibiting concurrency and synchronization | Features not exhibiting concurrency and synchronization |
| Kung et al. [8] (Object State) | Primitive types, simple control flow | Large number of attributes |
| Sinha et al. [9] (Exception-Handling) | Exception objects and variables, references to exception objects | Attributes/ local variables outside exception mechanism |
| Souter et al. [10] (OMEN) | Objects in the presence of polymorphism, aliasing and inheritance | Primitive types, references to primitive types |

Table 1: Summary of IBTTs [2].

Each feature pair consists of the feature's name and whether it is an attribute or routine.

The proposed approach prioritizes the Implementation Based Testing Techniques automatically. In the mapping process priority is assigned but by the tester, manually according to the tester view point.

## III. THE PROPOSED APPROACH

In this section proposed approach to prioritize the IBTT is presented. Our approach is based on the control flow and complexity of method of class under test. To assign priority to the IBTTs firstly the control flow of each method of class under test is captured. By using the control flow of each method complexity of each method is calculated. Depending on the complexity of each method and its suitable IBTT priority is assigned to Implementation Based Testing Technique.

### A. Control Flow

The goals of software testing are to assess and improve the quality of software. Software testing has proven to be difficult in the absence of design information. Without an adequate understanding of a programs structure, it is difficult to test it properly. Program recognition is a technology that can help testers to recover a programs design and consequently, make software testing effective. Syntactically, a program is a sequence of statements. If the flow of the program can be recovered and used to analyze the testing paths automatically then generating test data based on adequate testing criteria will help testers to understand the program structure and efficiently improve the software quality.

Control flow of any method consists of following three types of control structure.

1) Sequence
2) Repetition
3) Selection

Which are the elementary building blocks for all programs.

1. Sequential -: The most straightforward control structure is the sequence. A list of what to do, step by step.
2. Repetition -: Repetition is also called iteration and is used when something is repeated over and over again, so anything where the program goes round in a loop. Typically programmed using code such as WHILE, REPEAT and FOR statements.
3. Selection -: Making Decision Selection is used to make a decision to go down one path or another, often programmed using code such as an IF statement or a CASE statement.

### B. Complexity

Software complexity is one branch of software metrics that is focused on direct measurement of software attributes. There are hundreds of software complexity measures ranging from the simple such as source lines of code to the esoteric such as the number of variable definition/usage associations. The most basic complexity measure the number of lines of code does not meet the open reengineering criterion since it is extremely sensitive to programming language, coding style and textual formatting of the source code. The cyclomatic complexity measure which measures the amount of decision logic in a source code function does meet the open reengineering criterion. It is completely independent of text formatting and is nearly independent of programming language.

Ideally complexity measures should have both descriptive and prescriptive components. Descriptive measures identify software that is error-prone, hard to understand, hard to modify, hard to test, and so on. Prescriptive measures identify operational steps to help control software for example splitting complex modules into several simpler ones or indicating the amount of testing that should be performed on given modules.

## IV. PRIORITIZATION OF IBTTs

For priority assignment first complexity of each routine is calculated and then depending on the complexity, priority is assigned. There is a strong connection between complexity and testing and the structured testing methodology makes this connection explicit.

### A. Relationship between Complexity and Testing

First complexity is a common source of error in software. This is true in both an abstract and a concrete sense. In the abstract sense complexity beyond a certain point defeats the human minds ability to perform accurate symbolic manipulations and errors result. The same psychological factors that limit people’s ability to do mental manipulations of more than the infamous "7 + /- 2" objects simultaneously [MILLER] apply to software. Structured programming techniques can push this barrier further away, but not eliminate it entirely. The cyclomatic complexity measure correlates with errors in software modules. Other factors being equal the more complex a module is the more likely it is to contain errors. Also beyond a certain threshold of complexity the likelihood that a module contains errors increases sharply. Because of this reason many developer limit the cyclomatic complexity of their software modules in an attempt to increase overall reliability. A detailed recommendation for complexity limitation is given in table 2.

| Cyclomatic Complexity | Risk Summary |
|---|---|
| 1-10 | Simple, low risk |
| 11-20 | Moderate complexity, medium risk |
| 21-50 | Complex, high risk |
| 51+ | Very high risk |

Table 2: Complexity limitation

Second, complexity can be used directly to allocate testing effort by leveraging the connection between complexity and error to concentrate testing effort on the most error-prone software.

### B. Prioritization

McCabe proposed a way to measuring flow complexity of a method which basically counts one for each place where the flow changes from a linear flow. His algorithm translated at least approximately into Java terms is as follows. His measurement was designed before exceptions and threads were used in programming languages.

Start with a count of one for the method. Add one for each of the following flow-related elements that are found in the method.

| Category | Add one for each of the following |
|---|---|
| Methods | Each return that isn't the last statement of a method |
| Selection | if, else, case, default |
| Loops | for, while, do-while, break and continue |
| Operators | &&,\|\|,? and : |
| Exceptions | catch, finally, throw or throws clause |
| Threads | start() call on a thread |

Table 3: flow-related elements

To calculate the complexity of each routine in class information given above is used. When method start count is set to one then one is added to the complexity count whenever flow-related elements found in the method such as if, for, while etc. as information given above. After calculation of complexity of each method in a class priority is assigned to the Implementation Based Testing Technique as follow.

First IBTT and methods that are suitably tested by that IBTT is observed. Then sum of complexity of this method is calculated. This step is repeated until all the IBTT present in the output of suitable testing technique is over (i.e. until Tuple List is complete).

As information given in table 2 more complex a module or method more likely it is to contain errors. So there is need to test these more complex module or method to remove errors. That's why IBTT whose sum of complexity is highest have given the first priority and this Implementation Based Testing Technique is the best testing technique to test the Class Under Test. Less than that complexity has given second priority and so on.

## V. VALIDATION AND RESULT ANALYSIS

In this section we discus Validation and Result Analysis of our prioritization of IBTTs

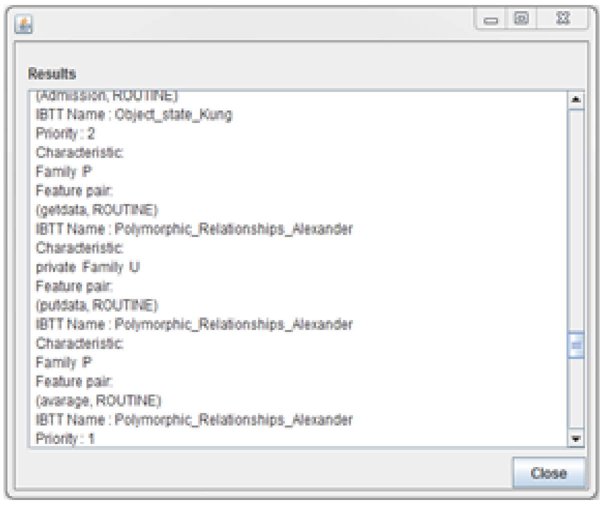
### A. Result Analysis

For result analysis we have tested the class Admission written in java language. Class Admission is implemented to display list of eligible candidate. Class Admission declares six variables, four methods and one constructor.

We tested class Admission by using tool developed for prioritization of IBTTs. The output generated by class Admission is shown in the figure 1.

### B. Validation

In order to validate the proposed approach of prioritization of IBTTs, tool is checked by using different java application. Here six different java applications having various object oriented characteristics is taken. Input these classes to mapping tool generate classtest and control flow of each method in class. Classtest shows list of attribute and routine that are suitably tested by Implementation Based Testing Technique. Analyzing the result of classtest gives observation that Polymorphic Relationship, Automated, Data Flow and Object State testing technique mostly used. Figure 2 shows the graph of validation.



```
(Admission, ROUTINE)
IBTT Name : Object_state_Kung
Priority : 2
Characteristic:
Family P
Feature pair:
(getdata, ROUTINE)
IBTT Name : Polymorphic_Relationships_Alexander
Characteristic:
private Family U
Feature pair:
(putdata, ROUTINE)
IBTT Name : Polymorphic_Relationships_Alexander
Characteristic:
Family P
Feature pair:
(avarage, ROUTINE)
IBTT Name : Polymorphic_Relationships_Alexander
Priority : 1
```

Figure 1: Classtest with priority.

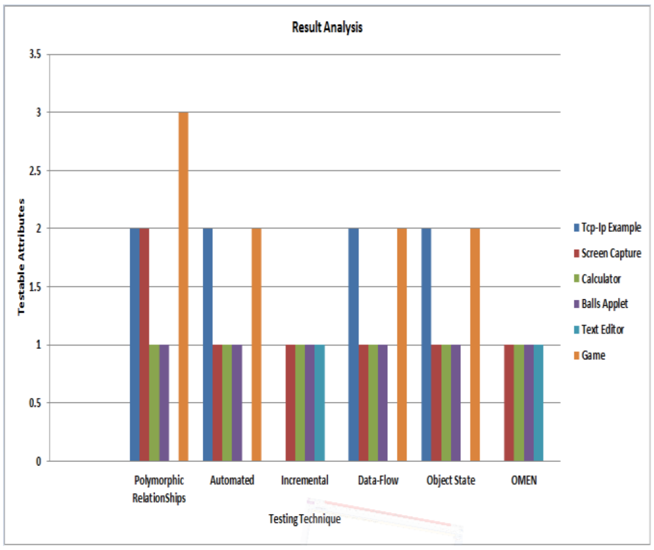


Figure 2: Result validation

## VI. CONCLUSION

In this paper we presented a prioritization of Implementation Based Testing Techniques. This approach captures the control flow of each method of class under test, calculate complexity and assign priority IBTTs. The control flow of method gives information about control structure used in the method and flow of data in method. By using the control flow of each method of class under test complexity of each method is calculated.

The complexity is measured by using Cyclomatic complexity. Cyclomatic complexity measures the amount of decision logic in a source code. Using this Cyclomatic complexity, complexity of each method is measured.

Depending on the complexity of each method and its IBTTs priority is assigned to Implementation Based Testing Techniques. Due to priority assigned to Implementation Based Testing Techniques the work of tester to select testing technique is much more reduced.

## REFERENCES

[1] Peter J. Clarke, Brian A. Malloy. A Taxonomy of OO Classes to Support the Mapping of Testing Techniques to a Class, in Journal of Object Technology, vol.4, no. 5,JulyAugust 2005.

[2] Peter J.Clarke and Brian Malloy. "Mapping Implementation Based Testing Techniques to Object-Oriented Classes", Technical Report, 2004-08.

[3] R. T. Alexander and A. J. Offutt. "Criteria for testing polymorphic relationships", In Proceedings of the 11th International Symposium on Software Reliability Engineering, pages 15-24.

[4] U. Buy, A. Orso, and M. Pezze. Automated Testing of Classes, In Proc. of ISSTA.Portland, OR, August 2000.

[5] M. J. Harrold, J. D. McGregor, and K. J.Fitzpatrick. Incremental Testing of Object-Oriented Class Structures, InProc. of ICSE,Melbourne, Australia, pages 6880.

[6] M. J. Harrold and G. Rothermel. Peforming Data Flow Testing on Classes, In Proc.of ACM SIGSOFT Symp. FSE.

[7] P. V. Koppol, R. H. Carver, and K. C. Tai. "Incremental integration testing ofconcurrent programs", IEEE Transactions on Software Engineering, 28(6):607-623.

[8] D. Kung, Y. Lu, N. Venugopalan, P. Hsia, Y. Toyoshima, C. Chen, and J. Gao."Object state testing and fault analysis for reliable software systems", In Proceedings of the 7th International Symposium on Reliability Engineering, pages 239-244.

[9] S. Sinha and M. J. Harrold. "Analysis and testing of programs with exception handling"' constructs. IEEE Transactions on Software Engineering, 26(9):849-871.

[10] A. L. Souter and L. L. Pollock. OMEN: A strategy for testing object-oriented software.

[11] Peter Clarke and Brian Malloy. A Unifed Approach to Implementation-Based Testing of Classes.

[12] R. V. Binder. ”Testing Object-Oriented Systems: Models, Patterns, and Tools”. Addison-Wesley, Reading, Massachusetts, 2000.

[13] B. Meyer. Object-Oriented Software Construction. Prentice Hall PTR, 1997.

## AUTHORS

**Panchal Rajkumar V.** received his B.E. degree in Information Technology (First Class with Distinction) in the year 2009 from SRTMU university and M.Tech Degree (First Class) in computer Engineering in 2012 From Dr.BATU university. He is GATE qualified. He has 2 years of teaching experience at undergraduate level. Currently he is working as assistant professor in Department of Computer Engineering of VPCOE, Baramati, Pune University. His research interests are Software Testing and Image Processing.

**Gyankamal Chhajed** obtained Engineering degree (B.E.) in Computer Science and Engineering in the year 1991-95 from S.G.G.S.I.E.T, Nanded and Postgraduate degree ( M.Tech. ) in Computer Engineering from College of Engineering, Pune (COEP) in the year 2005-2007 both with distinction . She is approved Undergraduate and Postgraduate teacher of Pune university and having about 18 yrs. of experience. She guided many projects at Undergraduate and Postgraduate Level. Gyankamal authored a book and has 23 publications at the national, international level for Conferences and Journal. She is life member of the ISTE & International Association IACSIT. Her research interests include Steganography and Watermarking, Image processing , Data mining and Information Retrieval , Biomedical Engineering.