A White Box Testing Technique in Software Testing: Basis Path Testing

D. Madhavi
Assistant Professor
Department of Information Technology
Andhra Loyola Institute of Engineering and Technology

Abstract
Software Testing is the emerging and important field of IT industry because without the concept of software testing, there is no quality software which is produced in the industry. Verification and Validation are the two basic building blocks of software testing process. There are various testing tactics, strategies and methodologies to test the software. Path Testing is one such a methodology used to test the software. Basically, path testing is a type of White Box/ Glass Box/ Open Box/ Structural testing technique. It generates the test suite based on the number of independent paths that are presented in a program by drawing the Control Flow Graph of an application. The basic objective of this paper is to acquire the knowledge on the basis path testing by considering a sample of code and the implementation of path testing is described with its merits and demerits.

Keywords: Basis Path Testing, Control Flow Graph, Cyclomatic Complexity, White Box Testing

I. INTRODUCTION
Now-a-days, human beings are shifted from traditional way of doing things to modern way by using new technologies. People can’t imagine a day without using Mobile Phones, Internet, Television, Washing Machines, Cars, Robotics, Toys, and Personal Computers etc. All these machines are working based on the software. Therefore it is the responsibility of industry to release the good quality software into the market. This can be achieved through Testers in the organization. Otherwise there is a big failure in the software industry. Some of the software problems in software industry which can’t be solved are explosion of the Ariane 5 Rocket, the Y2K problem, USA star-wars program, Failure of London Ambulance system, WIN-XP problem, accounting software Failures, USS Yorktown Incident.

Good Testing implies more than just executing a program with desired inputs and executed output. It requires the planning of test cases, design of test cases, executing of test cases and test review, which is given as a feedback to planning team. The software testing activity is very expensive, critical and complex. Software without testing is most dangerous and leads to more expensiveness. So no one is doing software without testing, it is like driving a car without breaks or gears. Therefore software testing is essential activity before releasing any software in a software company.

II. BASIS PATH TESTING
Basis Path Testing is a type of Whit-Box or Transparent Box or Code-Based or Structural Testing or Clear-Box or Open-Box or Glass-Box Testing Technique. Tester must have the knowledge of the implementation of the code to design test cases. That’s why generally this type of testing is carried out by software developers. It focuses on internal states of objects and tries to cover all the paths in the program. White-Box Testing can be started based on the detailed design document. The main aim of Whit-Box Testing is to check on how the system is performing. The advantages of White-Box Testing are Introspection, Stability and Thoroughness. The disadvantages are Complexity, Fragility and Integration. The difference between path testing and basis path testing is that path testing executes all the paths or some selected paths in the program whereas basis path testing is designed to test all the independent paths in a program. Basis path testing covers the branch testing also. A path is a sequence of instructions or statements that starts at an entry point or junction or decision and ends at another exit point or junction or decision. A path through the program may go through one or more decision or junction points. An independent path is a path moved along the one edge at least and the edge is not traversed before the path is defined.

The advantages of path testing are
1) It generates independent paths presented in a program.
2) It checks the program logic by executing all independent paths.
3) It generates analyzed and arbitrary test cases design.

Basis Path Testing is the oldest Structural testing technique and based on the control structure of a program. The steps to perform the basis path testing are firstly, a flow graph is drawn, next all possible paths covered in a program are prepared; lastly all those paths are executed during the testing. It is more general coverage criteria than other coverage criteria. It is useful for detecting the more errors. The problem with this testing is if a program having loops with infinite number of possible paths and it
is not practical to test all the paths. The solution to problem is some criteria are used to select paths which are executed for maximum coverage of logic. This technique selects paths that provide the basis set of execution paths through the program.

The basic principal is to test each individual path in as many ways as possible in order to maximize the coverage of each test case. Different ways of doing path testing is to one using DD (Decision-to-Decision) paths another way is using Basis Path Testing. These methods give the best possible chance of discovering all faults with in a piece of code and make the testing process easier. The DD-path testing is introduced by Miller in 1977 and Basis Path Testing is introduced by Thomas McCube in mid- 1970’s. The different ways of test data generation for basis path testing is using Genetic Algorithms and Clonal Selection Algorithm, Aspect Oriented programming.

A. Cyclomatic Complexity:

The metric and measurement are the two different words in the software engineering filed. Measurement is the quantitative indication of dimension, size and capacity of an attribute of a software process, whereas a metric is defined as the quantitative measurement of a degree of an attribute of a software process such as size, cost, quality and schedule. The test metrics is used to understand the effectiveness of a testing process that can measure quantitatively. The Cyclomatic Complexity is a software quality metric used to find the logical complexity of a software by considering the Control Flow Graph (CFG). It was introduced by Thomos J. McCube in the year of 1976. This metric describes the non-linearity of the graph and measure the number of linearly independent paths through a program. Cyclomatic Complexity can be calculated with respect to the methods, functions classes or modules in the software. The Cyclomatic number can be defined by representing a program as a Flow Graph. A Flow graph consists of nodes, which represents the processing statements while the edges represent the control flow between the nodes. Let us consider the below figure which represents the CFG having both edges and nodes

![Flow Graph](image)

Fig. 1: Flow graph consists of nodes and edges.

1) Mathematical Representation of a Cyclomatic Complexity:

To find the Cyclomatic number, the tester considers the number of paths in a program. If there are cycles then number of paths is infinite. So he considers only independent paths. A strongly connected graph G in which each node can be reached from any other node. In graph theory, e is the number of edges in a graph G, and n is the number of nodes then the number of independent paths are represented by V(G) which is equal to e-n+1. Sometimes the graph may not be strongly connected, in this case to make the graph to be strongly connected; there is a need to add an arc from last node to first node. Therefore the number of independent paths i.e V(G) = e-n+1 +1 i.e is equal to e-n+2.

According to Miller’s Theorem, V (G) =d+p. where d=decision nodes and p= number of graphs.

A k-way decision point contributes for (k-1) number of choice points.

Generally, a program may contain several procedures also. Separate procedures can be represented as separate flow graphs. These procedures can be called from any point but connections for calling is not shown explicitly. In this case, the sum of the numbers of each graph = Cyclomatic complexity of whole graph = V (G) = e-n+2p. Where p= number of graphs, e=number of edges and n=number of nodes. Cyclomatic Complexity number can be derived from any of the following.

\[ V (G) = e-n+2p \]  
\[ V (G) = d+p \]

Where d= number of decision nodes in the graph and this can be proved by Miller.

The formula to calculate number of nodes in a graph i.e d=k-1. Where k= number of arrows leaving the node. If a graph consists of switch case or multiple if-else decision nodes then this formula is used.

If the program is having many number of connected components then V (G) can be calculated as union of all the connected components. For example, a program P having three components M,N and O then complexity of whole program is \( V(G) = V(P) + V(M) + V(N) + V(O) \) or \( V (P \cup M \cup N \cup O) = V (P) + V (M) + V (N) + V (O) \).

Properties of Cyclomatic Complexity:

1) The independent paths in a Graph can be represented by \( V(G) \)
2) Independent paths, \( V(G) >= 1 \)
3) The graph G, can have one path if \( V(G)=1 \)
4) The Cyclomatic Complexity number is in between 0 to 10.

2) Cyclomatic Number:

If the number of Lines of Code is small then Cyclomatic Complexity can be easily calculated manually. The test automation tools can be used if the program is complex. If the program is large and complex then it involves more number of CFG’s. Based on the complexity number, the tester performs the corresponding actions. The complexity number varies based on the program complexity. The Cyclomatic Complexity number with its testability, cost and effort can be shown below.
Table – 1

<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Level of Complexity</th>
<th>Testability</th>
<th>Cost and Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -10</td>
<td>Less</td>
<td>High</td>
<td>Less</td>
</tr>
<tr>
<td>10-20</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>20-40</td>
<td>Very High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>&gt;40</td>
<td>-</td>
<td>Not at all Testable</td>
<td>Very High</td>
</tr>
</tbody>
</table>

3) **Tools used for Cyclomatic Complexity:**

Many tools are available to find the complexity of a program. Some tools are used for specific technologies. Example tools are
- devMetrics : used to analyse metrics for C+ projects
- GMetrics : used to find metrics in Java programs
- NDepends : also used for Java applications
- OCLint: It is a static code analyzer for C and Related structured programming languages.

For structured programming, the complexity is 1+ number of loops and if statements because there is no goto statement in structured programming. There are two ways to find the Cyclomatic Complexity, one is to calculate manually by using pen and paper another one is using tools. There are so many tools are existed. For example, a sample code snippet can give Cyclomatic Complexity as 2 by using Eclipse Metrics Plug-in, whereas the same sample code gives 5 by using SonarQube and by using GMetrics. Because the conjunctions is considered as a separate in one tool and the single if is considered as a single conjunction in another tool. Therefore to find the complexity stick with only one tool.

4) **Advantages of Calculating Cyclomatic Complexity:**

1) The Cyclomatic number is equivalent to the complexity.
2) Helps to the developers and testers to find the independent paths through the program.
3) All the independent paths must have been tested at least once.
4) Improves the code coverage.
5) Evaluates and eliminates the more risk associated with the program.
6) Helps to focus on uncovered paths
7) Helps to achieve number of test cases to achieve both branch coverage (Upper bound) and number of paths (Lower bound).

**B. Control Flow Graph (CFG):**

Control Flow Graph is a graphical representation of control structure of a program. CFG is also called as DD Graph or Program graph. It is a graphical representation of program’s source code. Flow graphs can be prepared as a directed graph. A directed graph having vertices, V and edges, E can be represented as a set of (V, E). Here, edges E are ordered pairs of elements of V. Based on the concepts of directed graph, following notations are used for a flow graph.

1) Node: Represents one or more procedural statements. These are numbered or labeled and is denoted by a circle. Nodes are of two types: source node and sink node. Source node is the starting node and sink node is the ending node of a CFG.
2) Edges/ Links: Represents flow off control in a program and is denoted by an arrow on the edge. The edge must terminate at a node.
3) Decision Node: A node with greater than one arrow leaving it called Decision Node.
4) Junction Node: A node with more than one arrow entering it called Junction node.
5) Regions: Areas bounded by edges and nodes are called regions. While counting regions, areas outside a graph are also considered a region.

1) **Flow Graph notations for different programming constructs:**

Flow graph can be prepared on the basis of control structure of a program. Here sequential statements can be merged into a single node. Therefore a flow graph is also called D-D graph or Decision-to-Decision graph. The fundamental graphical notations for programming constructs are shown in Fig.2. It depicts the constructs for sequential statements, If-then-Else construct, Do-while construct, While-Do construct and Switch Case.

![Flow Graph notations for different programming constructs](image-url)
2) **Differences between CFG and Flow Chart:**
   1) A program flow chart resembles a Control Flow Graph.
   2) In CFG, don’t show the details of what is in a process block. Whereas, flow chart every part of the process block is drawn.
   3) Flow chart focuses on process steps. Whereas CFG focus on control flow of a program.
   4) Act of drawing a CFG is useful tool. i.e CFG helps to clarify the Control Flow and data flow issues.

C. **Assumptions used in Path Testing:**
The assumptions of Path Testing have the following.
1) All the specifications are in correct manner.
2) Data is defined and accessed properly.
3) There are no bugs other than those that affect the Control Flow Graph.

D. **Terminology used in Path Testing:**
To represent the software as a Control Flow Graph some common terminology is used i.e path, segment, path segment, length of path, independent path. Generally, path is a sequence of instructions or statements through a program that starts at an entry or junction or decision and ends at another or same entry or decision or junction. A segment is a single process lies between two nodes. The smallest segment is a link. A path consists of segments. Path segment is a succession of two or more consecutive links that belongs to some path. The length of a path is measured by the number of links in it, not by the number of instructions or statements executed along the path. Another way to measure the length of the path is the number of nodes traversed. The number of links traversed is one less than the number of nodes traversed if the program is having an entry and an exit node. Independent path is any path through the graph that introduces at least one new set of processing statements or new conditions or new edges.

E. **Guidelines:**
Cyclomatic number can be used as an upper bound and lower bound for the number of test cases. Based on the upper limit of Cyclomatic number the number of test cases is designed. The set of independent paths are considered as the basis set for flow graph of the software. The following guidelines are used for designing of test cases in path testing.
1) Path Testing based on control structure of the program for which flow graph is prepared.
2) Path Testing requires complete knowledge of programs structure.
3) Path Testing is closer to developer and used by him to test the module.
4) Effectiveness of Path Testing $\propto 1/\text{Size of software under test}$.
5) Choose enough paths in such a way that maximum logic coverage is achieved.

F. **Applications of Path Testing:**
1) **Coverage Analysis:**
Path Testing is better suitable as compared to other testing methods such as statement coverage or branch coverage because basis path set provides the number of test cases covered. i.e Branch or statement coverage path set $\subseteq$ Basis Path set. It performs through testing and more coverage. Cyclomatic Complexity along with basis path analysis employees more comprehensive scrutiny of code structure and control flow, providing a far superior coverage technique.
2) **Applied in Unit Testing:**
Mainly used for structural testing of a module. Some of the decision outcome errors and control flow problems are hidden with branch testing. These can be uncovered with path testing in module testing and prepares then for integration.
3) **Applied in Integration Testing:**
There is a chance of interface errors during calling of one module to another module. Path testing analyses all paths on the interface and explores all errors.
4) **Used in Maintenance Testing:**
Path Testing is also necessary with modified version of software. Selected path testing is done as part of Regression Testing or earlier prepared unit testing should be run on the modified software.
5) **Path Testing takes care of Complexity of software:**
Basically, Testing Effort $\propto$ complexity of the Software
Based on the complexity of software, Cyclomatic Complexity provides the number of Test Cases executed on the software. The number of tests derived (carried out) $\propto$ complexity of Software
   - Path Testing take cares of Complexity of software.
6) **Path Testing effort is concentrated on the error prone software:**
It provides the number of tests as a measure of software Cyclomatic Complexity number, which signifies the testing effort. Testing effort is only on error-prone part thus minimizing testing effort.
G. Limitations of Path Testing:

1) Difficult to find errors in different areas
   1) Interface errors and errors are not caught
   2) Not all the initialization mistakes are covered.
   3) Specification mistakes in requirements are not covered.
   4) The data base and data flow errors are not caught
   5) Mistakes in CFG are not detected. Because of mistakes of designer or any errors in specifications.

2) Difficult process if the project is too complex.
3) Doing Path Testing is quite time consuming process.
4) It is an effort driven activity to cover all the planned Path Testing activities.
5) Successful Path Testing does not reveal any missing functionality and wrong implementations.

III. BASIS PATH TESTING APPROACH

Basis path testing follows the guidelines of path testing and it can be described using a sample pseudo code. The Basis path testing is performed on this example code for that need to find CFG, Cyclomatic Complexity, Independent paths and Test Suite. The sample pseudo code is as shown in the Fig. 3.

```
START
Variable y=0
INPUT x
IF x>0 then
   y=0
IF x>20 then
   IF x<=30 then
      y=y+(x-25)*0.5
   ELSE
      y=y+50+(x-30)*0.1
   IF x>=3000 then
      y=y*0.9
END IF
END IF
OUTPUT y
STOP
```

The Test Suite satisfies the all the above independent path. That is the test cases designed in a test suite should be in such a manner such that all paths are satisfied. Then test suite has a minimum of five test cases.
IV. Conclusion

Many researchers provided different techniques to perform path testing in different ways even though the basis path testing is the basic fundamental way of performing the White Box testing technique even it has its limitations. Cyclomatic complexity number and independent paths places a major role in performing path testing. Path testing catches 50 percent of all bugs caught during Unit Testing. It is more effective for unstructured code than for structured code follows structured programming. All the software organizations use this path testing to release the software product into the market or to customers. The study of optimization and prioritizations of paths is the future work for this paper. Many of the researchers find the optimizations of paths even though it is still a problem if the software is too complex.

REFERENCES