International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7 Issue-5, January 2019

Define – use Testing – An Example

S. Shanmuga Priya

Abstract: ***Variables plays significant role in programming and they ease programmers to write their programs flexibly. Variables are used to represent the data in a program. On executing a program, the variables are replaced with the real time values and hence it creates a possibility for a program to process different set of data. They are most predominant ones that involve in computation and bear the intermediate or final values in a program. These values are good enough in deciding the flow of the program and hence there is a necessity for analyzing flows. From the software tester's perspective, there grows a need to analyze and test those variables structurally. Data-flow testing is a white-box based testing technique that utilizes control flow and data flow through the program for testing. There are two main forms of data flow testing called as define/use testing and slice-based testing. This paper gives a focus on define/use based testing where it uses simple rules that aids in ensuring whether all the variables are defined and used at the appropriate points within the program. It helps the tester in chalking out the values of a variable, recording them and can trace the changes in values within the program. It is done by using a concept of a program graph, which is closely related to the path testing, selected on a specific variable.***

Index Terms: White-box Testing, Dataflow Testing, Define/Use Testing, du-path, dc-path.

I. INTRODUCTION

In procedural programming language like C, Java, variables plays a major role. The variables are defined by assigning values to them, and these values are used for various purposes throughout the program. Figure 1 shows the lifecycle of a variable in a program. A program comprises of statements (instructions) that may contain variable(s). There may be a need for one statement to interact with another, and this is achieved by using variable in the program. The variables are used for computations and to hold the values involved. Values computed at one part of the statement can be used by another part, thus leads to flow of data in a program.

Revised Manuscript Received on 30 January 2019.
***** Correspondence Author
S. Shanmuga Priya, Senior Assistant Professor, Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, Karnataka, India.

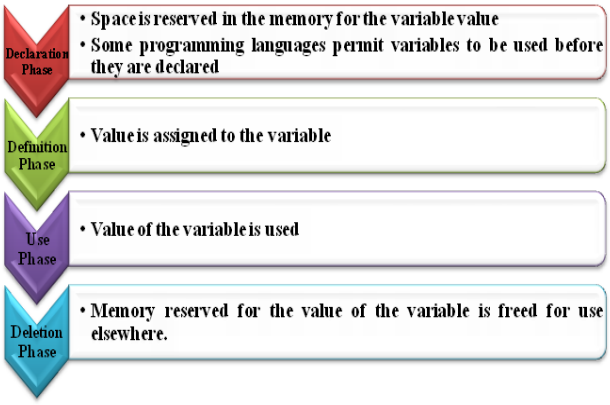


Figure 1 Lifecycle of a variable in a program

Potentially computed values decide the immediate flow of the system and hence it needs careful analysis. Correct or incorrect value computation is revealed only when it is used. Data flow testing aids in designing the test cases based on the definition and usage of variables that are used in the code i.e, it aims at testing the definition-use (du-) pair. It is done by testing the place where the variable receives the values (definition) and the place where the values are being used (usage). This testing is based on data-flow analysis and test cases are generated from the code [1] [2] [3]. In addition to test data generation, it can also be used for measuring the adequacy criterion, code optimization, anomaly detection [2] [4]. Moreover, the path-based nature of data flow analysis makes the infeasibility problem relevant to the software tester especially that helps in achieving a reasonable code coverage. Section 2 of this paper gives the basic idea about data flow testing, section 3 gives an example and describe about the data flow testing (du-path testing), section IV present the issues with data-flow testing, and section 5 gives conclusion.

II. DATA FLOW TESTING

Data flow testing techniques [5] [6] [4] is a white-box based testing technique was first proposed by Herman [7]. It is a form of control-flow testing technique that examines the variable's lifecycle. Based on the data flow testing, the occurrences of variables in a program is categorized namely as definition (DEF) and usage (USE) known as define/use testing, first formalized by Sandra Rapps and Elaine Weyuker [4]. The major advantage of data usage testing is it helps in identifying the risky areas of code can be found and more test cases can be applied to thoroughly test the application [8]. In addition to that, it helps in identifying the following anomalies [9],

- ✓ Variable that is declared, but never used in a program
- ✓ Variable if used, but never declared in any part of the program
- ✓ Variable if used, before it is declared
- ✓ Variable is used, before it is defined
- ✓ Variable being defined multiple times in a program before it is used
- ✓ Variable is deallocated before its usage
- ✓ Variable is used after deallocated of the memory

The definitions used in dataflow testing are as follows:

Define/Use Testing:

For a program P, a graph can be generated denoted as G (P), which consists of a single source node and single sink node. There are no edges from a node to itself (self loop). The set of variables used within the program is denoted as V, and the set of all paths within the program graph P (G) that can be generated is denoted as Paths (p) called as define/use testing.

Definition Node:

It is referred to as DEF (v,n) where v denotes the variable from the set V, n denotes the node in the program graph of P iff v is defined. Simply stated, a definition of a variable can be represented by the occurrence of a node (n) in which a variable (v) is assigned with a value. For example, a node z in a program P is considered as a defining node if it is stated as *int m=0;*.

Usage Node:

It is referred as USE (v,n), where v denotes the variable from the set V, n denotes the node in the program graph of P iff v is used. Simply stated, a usage of a variable can be represented by the occurrence of a node (n) in which a variable (v) is used or referenced. For example, a node z in a program P is considered as a usage node, if it is stated as *m=6+z;*.

Depending on the node's usage, it is further categorized into two types called as computational use (c-use) or predicate use (p-use) [10**]**.

c-use:

The c-use of a variable occurs in a computation statement or in output statement.

p-use:

The p-use of a variable occurs where a variable value is used in the condition (predicate) statement that determines the instant execution path.

du-Path:

A path p in a program graph P(G), is a du-path for any variable v, iff there exists DEF(v,n) and USE(v,n) nodes, such that i is the starting node and j is the ending node of the path.

Definition Clear (dc clear):

Any path p that is starting from a node n at which the variable v is defined and its ending at a node m at which the variable v is used, without being redefining of the variable v anywhere else along the path p, then the path is said to be definition clear for the variable v.

Usage Type	Meaning	Definition	Example(s)
c-use	computation-use	Variables used for computation	i) Variables on the right-hand side of a statement (c) ii) temp := c + 5; iii) Variables used in a write statement
p-use	predicate-use	Variables used for condition statements	i) if(c!= MAXLENGTH) {} ii) if (X > 0) then iii) if (a>0)

III. MATH

The example considered for define/use path testing is "The Commission Problem". The motive for selecting this problem is it's a typical commercial application that consists of a blend of computation and decisions involved. Also its rich in input stream, wide scope for dealing with the functional behavior of the system and few challenges are involved in producing the output table.

A. THE COMMISSION PROBLEM STATEMENT

A gunsmith in Missouri manufactures rifle that consists of locks, stocks and barrels. It must be sold by the salespersons in Arizona Territory. The cost of locks, stocks and barrels fixed by the manufacturer is $45.00, $30.00, and $25.00 respectively. The constraint to the salesperson is he/she has to sell at least one complete rifle per month and the constraint for the manufacturing unit it their production limit in a month is 70 locks, 80 stocks and 90 barrels. The company had four salespersons. Each rifle salesperson has to travel at least one town per month, can visit up to ten towns, and collect the sales details. At the end of each month, the salesperson must send a telegram to the Missouri Company by stating the total orders collected from each town. The company computes the commission as follows at the end of each month: 10% on sales up to $1000, 15% on the next $800, and 20% on any sales in excess of $1800. The telegrams from each salesperson were sorted into piles (by person) and at the end of each month a data file is prepared, containing the salesperson's name, followed by one line for
each telegram order, showing the number of locks, stocks, and barrels in that order. At the end of the sales data lines, there is an entry of "-1" in the position where the number of locks would be to signal the end of input for that salesperson. The program produces a monthly sales report that gives the salesperson's name, the total number of locks, stocks, and barrels sold, the salesperson's total dollar sales, and finally his/her commission [11].

B. PSEUDOCODE

The pseudocode for the commission problem [11] is considered for illustrating define-use testing.

```
Program Commission (INPUT, OUTPUT)

Dim locks, stocks, barrels As Integer
Dim lockPrice, stockPrice, barrelPrice As Real
Dim totalLocks, totalStocks, totalBarrels As Integer
Dim lockSales, stockSales, barrelSales As Real
Dim sales, commission: REAL

lockPrice = 45.0
stockPrice = 30.0
barrelPrice = 25.0
totalLocks = 0
totalStocks = 0
totalBarrels = 0

Input (locks)
While NOT (locks = -1) 'Input device uses -1 to indicate end of data
Input(stocks, barrels)
totalLocks = totalLocks + locks
totalStocks = totalStocks + stocks
totalBarrels = totalBarrels + barrels
Input (locks)
EndWhile

Output ("Locks sold:", totalLocks)
Output ("Stocks sold:", totalStocks)
Output ("Barrels sold:", totalBarrels)

lockSales = lockPrice * totalLocks
stockSales = stockPrice * totalStocks
barrelSales = barrelPrice * totalBarrels
sales = lockSales + stockSales + barrelSales
Output ("Total sales:", sales)

If (sales > 1800.0)
      Then
            commission = 0.10 * 1000.0
            commission = commission + 0.15 * 800.0
            commission = commission + 0.20 * (sales–1800.0)
      Else If (sales > 1000.0)
      Then
            commission = 0.10 * 1000.0
            commission = commission + 0.15*(sales–1000.0)
            Else commission = 0.10 * sales
      EndIf
EndIf
Output ("Commission is $", commission)
End Commission
```

C. STEPS TO BE FOLLOWED IN DATA FLOW TESTING

Given a code (program or pseudo-code), the steps are:

1. Number the lines in the program or pseudo-code that is considered.
2. Draw the program graph.
3. Draw the Decision-to-Decision (DD) Path Graph.
4. List the variables.
5. List occurrences and assign a category to each variable (define/use for variable).
6. Identify du-pairs and their use (p- or c-).
7. Define test cases, depending on the required coverage.

Step 1: Number the lines

```
1.  Program Commission (INPUT, OUTPUT)
2.  Dim locks, stocks, barrels As Integer
3.  Dim lockPrice, stockPrice, barrelPrice As Real
4.  Dim totalLocks, totalStocks, totalBarrels As Integer
5.  Dim lockSales, stockSales, barrelSales As Real
6.  Dim sales, commission: REAL
7.  lockPrice = 45.0
8.  stockPrice = 30.0
9.  barrelPrice = 25.0
10. totalLocks = 0
11. totalStocks = 0
12. totalBarrels = 0
13. Input (locks)
14. While NOT (locks = -1) 'Input device uses -1 to indicate end of data
15. Input(stocks, barrels)
16. totalLocks = totalLocks + locks
17. totalStocks = totalStocks + stocks
18. totalBarrels = totalBarrels + barrels
19. Input (locks)
20. EndWhile
21. Output ("Locks sold:", totalLocks)
22. Output ("Stocks sold:", totalStocks)
23. Output ("Barrels sold:", totalBarrels)
24. lockSales = lockPrice * totalLocks
25. stockSales = stockPrice * totalStocks
26. barrelSales = barrelPrice * totalBarrels
27. sales = lockSales + stockSales + barrelSales
28. Output ("Total sales:", sales)
29. If (sales > 1800.0)
30. Then
31. commission = 0.10 * 1000.0
32. commission = commission + 0.15 * 800.0
33. commission = commission + 0.20 * (sales–1800.0)
34. Else If (sales > 1000.0)
35. Then
36. commission = 0.10 * 1000.0
37. commission = commission + 0.15*(sales–1000.0)
38. Else
39. commission = 0.10 * sales
40. EndIf
41. EndIf
42. Output ("Commission is $", commission)
43. End Commission
```

Step 2: Draw the program graph

The program graph for the commission problem is shown in the figure 2.

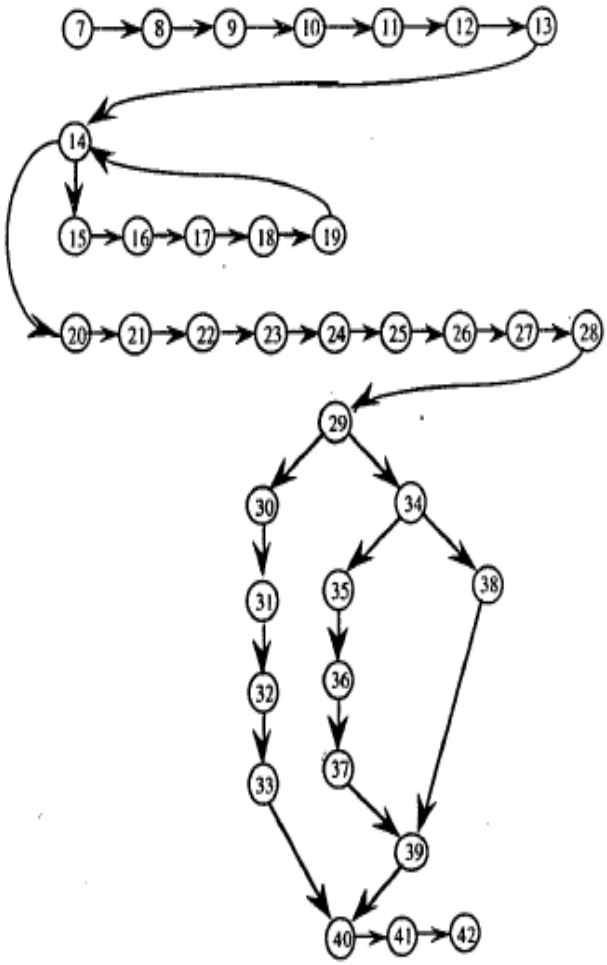


Figure 2 Program Graph for the example code

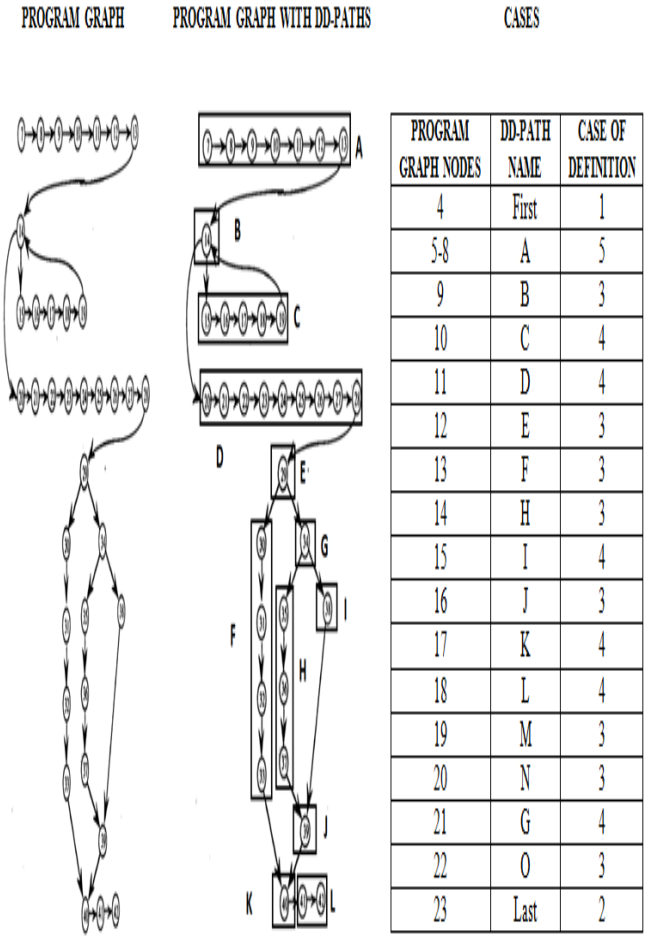
Step 3: Draw the Decision-to-Decision (DD) Path Graph

Decision-to-Decision path graph is a best-known structural testing. It is a path chain in a program graph such that, the initial and terminal nodes are distinct, every interior node has indeg =1 and outdeg = 1. The initial node is 2-connected to every other node in the path. The step to draw a DD Path Graph is:

i) Consider the program graph
   a. Divide program graph into DD-paths.
   b. Consider each and every node in the program graph and analyze to which particular case (any of the 5 cases in the definition) it falls under:
      Case 1: It consists of a single node with indegree=0, or
      Case 2: It consists of a single node with outdegree=0, or
      Case 3: It consists of a single node with indegree >=2 or outdegree >=2, or
      Case 4: It consists of a single node with indegree =1 and outdegree = 1, or
      Case 5: It is a maximal chain of length >=1
   c. Name each DD-path as A, B, C, etc.

ii) Build the DD-path graph.

Figure 3 shows the DD path graph for the commission problem.



PROGRAM GRAPH NODES	DD-PATH NAME	CASE OF DEFINITION
4	First	1
5-8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	Last	2

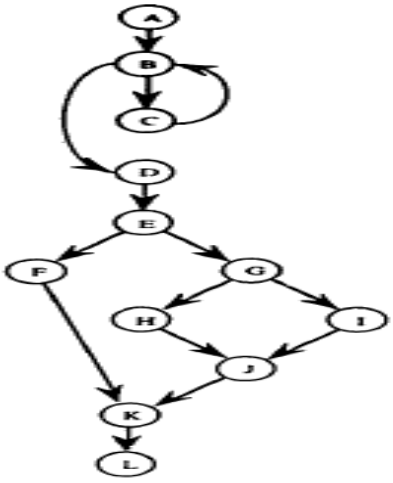


Figure 3 DD Path Graph

Step 4: List the variables

There are 14 different variables involved in the commission problem program. The pool of variables is represented by the character V.

V = {locks, stocks, barrels, lockPrice, stockPrice, barrelPrice, totalLocks, totalStocks, totalBarrels, lockSales, stockSales, barrelSales, sales, commission}

Step 5 & 6: List occurrences and assign a category to each variable (Define/Use for variable), and also identify du-pairs and their use (p- or c-)

a) du-path for the variable locks

Variable lock consists of two defining nodes 13, 19 and two usage nodes 14 (p-use), 16 (c-use), that gives four sub paths as follows. All four paths p1, p2, p3 and p4 are definition clear path.

DEF node	USE node	
	NODE	USE
13 19	14	p-use
	16	c-use

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
13 – 14	p1	<13, 14>	YES	-	YES	-
13 – 16	p2	<13, 14, 15, 16>	YES	-	YES	-
19 – 14	p3	<19, 20, 14>	YES	-	YES	-
19 – 16	p4	<19, 20, 14, 15, 16>	YES	-	YES	-

b) du-path for the variable stocks

Variable stock consists of one defining node 15 and one usage nodes 17, which give one sub path, that is definition clear as follows.

DEF node	USE node
15	17

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
15 – 17	p5	<15, 17>	YES	-	YES	-

c) du-path for the variable barrels

Variable barrels consists of one defining node 15 and one usage nodes 18, which give one sub path that is definition clear as follows.

DEF node	USE node
15	18

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
15 – 18	p6	<15, 18>	YES	-	YES	-

d) du-path for the variable lockPrice

Variable lockPrice consists of one defining node 7 and one usage nodes 24, which give two sub paths p7 and p8 as follows. Both the paths are definition clear.

DEF node	USE node
7	24

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
7 – 24	p7	<7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24>	YES	-	YES	-
	p8	<7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24>	YES	-	YES	-

e) du-path for the variable stockPrice

Variable stockPrice consists of one defining node 8 and one usage nodes 25, which give two sub paths p9 and p10 as follows. Both the paths are definition clear.

DEF node	USE node
8	25

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
8 – 25	p9	<8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24, 25>	YES	-	YES	-
	p10	<8, 9, 10, 11, 12, 13, 14, 21, 22, 23, 24, 25>	YES	-	YES	-

f) du-path for the variable barrelPrice

Variable barrelPrice consists of one defining node 9 and one usage nodes 26, which give two sub paths p11 and p12 as follows. Both the paths are definition clear.

DEF node	USE node
9	26

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
9 – 26	p11	<9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24, 25, 26>	YES	-	YES	-
	p12	<9, 10, 11, 12, 13, 14, 21, 22, 23, 24, 25, 26>	YES	-	YES	-

g) du-path for the variable totalLocks

Variable totalLocks consists of two defining node 10 and 16, three usage nodes 16, 21, 24, which give six sub paths p13, p14, p15, p16, p17, p18 as follows. Sub paths p13, 17 and p18 are definition clear, whereas path p14 and p15 are not definition clear as the node 16 that is present in the du-path is redefined. Sub path p16 is infeasible as it defines a path to itself and it can be disregarded.

DEF node	USE node
10 16	16 21 24

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
10 – 16	p13	<10, 11, 12, 13, 14, 15, 16>	YES	-	YES	-
10 – 21	p14	<10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21>	YES	-	NO	DEF node 16 is present in the du-path that may replace the value
10 – 24	p15	<10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24>	YES	-	NO	DEF node 16 is present in the du-path that may replace the value
16 – 16	p16	Disregard it, as a node to itself cannot form a path	-	-	-	-
16 – 21	p17	<16, 17, 18, 19, 20, 14, 21>	YES	-	YES	-
16 – 24	p18	<16, 17, 18, 19, 20, 14, 21, 22, 23, 24>	YES	-	YES	-

h) du-path for the variable totalStocks

Variable totalStocks consists of two defining node 11 and 17, three usage nodes 17, 22, 25, which give eight sub paths p19 to p26 as follows. Sub paths p19, p23 and p25 are definition clear, whereas, sub paths p20, p21, p24 and p26 are not definition clear. Sub path p22 is infeasible as it defines a path to itself and it can be disregarded.

DEF node	USE node
11 17	17 22 25

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
11 – 17	p19	<11, 12, 13, 14, 15, 16, 17>	YES	-	YES	-
11 – 22	p20	<11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22>	YES	-	NO	DEF node 17 is present in the du-path that may replace the value
11 – 25	p21	<11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24>	YES	-	NO	DEF node 17 is present in the du-path that may replace the value
17 – 17	p22	Disregard it, as a node to itself cannot form a path	-	-	-	-
17 – 22	p23	<17, 18, 19, 20, 14, 21, 22>	YES	-	YES	-
	p24	<17, 18, 19, 20, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22>	YES	-	NO	DEF node 17 is present in the du-path that may replace the value
17 – 25	p25	<17, 18, 19, 20, 14, 21, 22, 23, 24, 25>	YES	-	YES	-
	p26	<17, 18, 19, 20, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24, 25>	YES	-	NO	DEF node 17 is present in the du-path that may replace the value

i) du-path for the variable totalBarrel

Variable totalBarrel consists of two defining node 12 and 18, three usage nodes 18, 23, 26, which give eight sub paths p27 to p34 as follows. Sub paths p27, p31 and p33 are definition clear, whereas, sub paths p28, p29, p32 and p34 are not definition clear. Sub path p30 is infeasible as it defines a path to itself and it can be disregarded.

DEF node	USE node
12 18	18 23 26

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
12 – 18	p27	<12, 13, 14, 15, 16, 17, 18>	YES	-	YES	-
12 – 23	p28	<12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23>	YES	-	NO	DEF node 18 is present in the du-path that may replace the value
12 – 26	p29	<12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24, 25, 26>	YES	-	NO	DEF node 18 is present in the du-path that may replace the value
18 – 18	p30	Disregard it, as a node to itself cannot form a path	-	-	-	-
18 – 23	p31	<18, 19, 20, 14, 21, 22, 23>	YES	-	YES	-
	p32	<18, 19, 20, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23>	YES	-	NO	DEF node 18 is present in the du-path that may replace the value
18 – 26	p33	<17, 18, 19, 20, 14, 21, 22, 23, 24, 25, 26>	YES	-	YES	-
	p34	<17, 18, 19, 20, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26>	YES	-	NO	DEF node 18 is present in the du-path

j) du-path for the variable lockSales

Variable lockSales consists of one defining node 24, one usage node 27, which gives one sub path p35 as follows that is definition clear.

DEF node	USE node
24	27

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
24 – 27	p35	<24, 25, 26, 27>	YES	-	YES	-

k) du-path for the variable stockSales

Variable stockSales consists of one defining node 25, one usage node 27, which gives one sub path p36 as follows that is definition clear.

DEF node	USE node
25	27

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
25 – 27	p36	<25, 26, 27>	YES	-	YES	-

l) du-path for the variable barrelSales

Variable barrelSales consists of one defining node 26, one usage node 27, which gives one sub path p37 as follows that is definition clear.

DEF node	USE node
26	27

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
26 – 27	p37	<26, 27>	YES	-	YES	-

m) du-path for the variable sales

Variable sales consists of one defining node 27, and six usage node 28, 29, 33, 34, 37 and 39, which gives six sub paths p38 to p43 as follows that are definition clear.

DEF node	USE node
27	28 29 33 34 37 39

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
27 – 28	p38	<27, 28>	YES	-	YES	-
27 – 29	p39	<27, 28, 29>	YES	-	YES	-
27 – 33	p40	<27, 28, 29, 30, 31, 32, 33>	YES	-	YES	-
27 – 34	p41	<27, 28, 29, 34>	YES	-	YES	-
27 – 37	p42	<27, 28, 29, 34, 35, 36, 37>	YES	-	YES	-
27 – 39	p43	<27, 28, 29, 34, 38, 39>	YES	-	YES	-

n) du-path for the variable commission

Variable commission consists of six defining node 31, 32, 33, 36, 37 and 39, four usage node 32, 33, 37, 42, which gives one sub path p37 as follows that is definition clear. Sub paths p44, p49, p55, p58, p59, p63 are definition clear, sub paths p45, p47, p47, p51, p67 are not definition clear, sub paths p46, p50, p52, p54, p56, p57, p60, p61, p62, p64, p65, p66 are considered as not applicable as there is no feasible path between the source and sink node considered, and path p48, p53 are disregarded.

DEF node	USE node
31 32 33 36 37 39	32 33 37 42

Beginning Node – End Node	Path	du-path	Feasible Path?	Why not feasible?	dc-path?	Why not Definition clear path?
31, 32	p44	<31, 32>	YES	-	YES	-
31, 33	p45	<31, 32, 33>	YES	-	NO	DEF node 32 and 33 are in the du-path that may replace the value
31, 37	p46	No Path	-	-	N/A	-
31, 42	p47	<31, 32, 33, 41, 42>	YES	-	NO	DEF node 32 and 33 are in the du-path that may replace the value
32, 32	p48	Disregard it, as a node to itself cannot form a path	-	-	-	-
32, 33	p49	<32, 33>	YES	-	YES	-
32, 37	p50	No Path	-	-	N/A	-
32, 42	p51	<32, 33, 41, 42>	YES	-	NO	-
33, 32	p52	No Path	-	-	N/A	-
33, 33	p53	Disregard it, as a node to itself cannot form a path	-	-	-	-
33, 37	p54	No Path	-	-	N/A	-
33, 42	p55	<33, 41, 42>	YES	-	YES	-
36, 32	p56	No Path	-	-	N/A	-
36, 33	p57	No Path	-	-	N/A	-
36, 37	p58	<36, 37>	YES	-	N/A	-
36, 42	p59	<36, 37, 40, 41, 42>	YES	-	N/A	-
37, 32	p60	No Path	-	-	N/A	-
33, 33	p61	No Path	-	-	N/A	-
37, 37	p62	No Path	-	-	N/A	-
37, 42	p63	<37, 40, 41, 42>	YES	-	YES	-
39, 32	p64	No Path	-	-	N/A	-
39, 33	p65	No Path	-	-	N/A	-
39,37	p66	No Path	-	-	N/A	-
39,42	p67	<39, 40, 41, 42>	YES	-	YES	-

Step 7: Now based on the obtained du-path of each and every variable, test cases can be prepared.

After deriving the test paths, test cases can be prepared by considering the various input parameters involved in the corresponding path.

IV. ISSUES WITH DATA-FLOW TESTING

The following are a few pitfalls that have to be addressed while using data flow testing. They are as follows:

- ✓ Its bit complicated to conduct accurate data-flow analysis where array or pointer variables are used in a program.
- ✓ Bit difficult to test the define-use for programs that contains inter-procedures.
- ✓ When data-flow testing is done by using some static analysis tools, sometimes, results in arising "false alarms" that may hide the real problems. This may cause the tool to report with large number of anomalies, which are not caused by being wrong with anything.
- ✓ Cost involved in conduction of data-flow testing is quite higher when compared with other white box testing technique in connection with path testing.

V. CONCLUSION

As testing is a process of executing the program with an intent for finding the underlying errors, if any, this paper majorly concentrates on the data flow testing aspect that are described with an example of how to derive the test paths that needs to be tested. This technique can be followed when the application must be tested for ensuring the correctness of the data flow among the variables that is used in. Also, this approach gives test coverage metrics that ensures the percentage of code covered, that helps in assessing the defect rate of the software.

REFERENCES

1. Beizer, Boris "Software Testing Techniques," Van Nostrand Reinhold, 1990.
2. Harrold, Mary J. and Soffa, Mary L. "Inter Procedural Data Flow Testing," 3rd Testing, Analysis and Verification SYMP (SIGSUFT89), 1989, 158-167.
3. Siegel, Shel 1996, "Object Oriented Software Testing-A Hierarchical Approach," John Wiley & Sons, Inc. 1996.
4. Rapps, S. and Weyuker, E. J. "Selecting Software Test Data Using Data Flow Information," IEEE Transactions on Software Engineering, SE-11 (4), April 1985, 367-375.
5. Harrold, Mary J. and Rothermel, G. "Performing Data Flow Testing on Classes," 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering, Dec. 1994, 154-163.
6. Laski, J. and Korel, B. "A Data Flow Oriented Program Testing Strategy," IEEE Transactions on Software Engineering, SE-9(3), May 1983, 347-354.
7. P. M. Herman, "A Data Flow Analysis Approach to Program Testing", Australian Computer Journal, vol. 8, issue 3, pp. 92–96, 1976.
8. Srinivas Nidhra and Jagruthi Dondeti, "Black-box and White-box Testing Techniques-A Literature Review", International Journal of Embedded Systems and Applications (IJESA), Vol. 2, No. 2, June 2012.
9. Yogeshsingh, "Software Testing", Cambridge University Press, pp 173-175, 2012.
10. Frankl, Phyllis G. and Weyuker, Elaine J. "An Applicable Family of Data Flow Testing Criteria," IEEE Transactions on Software Engineering, 14(10), 1988, 1483-1498.
11. Paul C. Jorgensen: Software Testing, A Craftsman's Approach, 4th Edition, Auerbach Publications, 2013.

AUTHORS PROFILE

S. Shanmuga Priya is currently working as Senior Assistant Professor in the Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, Karnataka. She received her B.E. in Computer Science and Engineering from Bharathidasan University, and M.E in Computer Science and Engineering, from Anna University, Tamilnadu, India. She has around 14 years of experience in teaching. Her research interests include Big Data, Data Mining, Software Engineering and Software Testing.