Ninad. A. Patil
Roll No : 33
MCA Sem III / II nd Years
VESIT
Afternoon

10/09/2020

PRACTICAL No :- 3

Description:

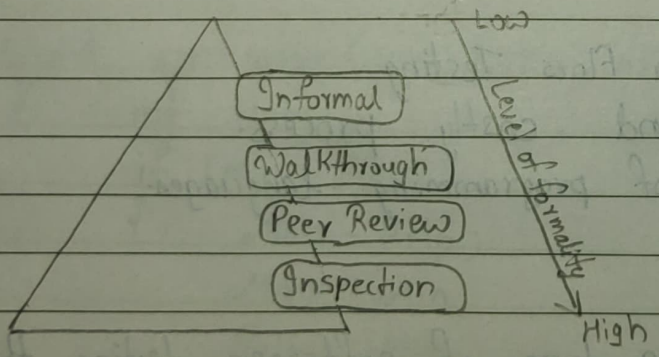
Static Testing is a software testing technique in which the software is tested without executing the code.

Aim: To perform Data Flow Analysis, Control Flow Analysis, and calculate cyclomatic complexity in static testing.

Theory:

- Static Testing is a testing technique in which software is tested without executing the code. Its has two parts :-

i) Reviews: Its typically used to find and eliminates error or ambiguites in documents such as requirements, designing, test cases etc. There are different types of reviews: given by a simple diagram



ii) Static Analysis: In this the code written by developers are ~~analysed~~ analysed for structural defect that may lead to defects. Following are the types of defects found by the tools during static analysis.

- A variable ~~is~~ with an undefined value.

- Inconsistent interface between modules and components.
- Variables that are declared but never used.
- Unreachable code (or) Dead code.
- Programming Standard violations
- Security vulnerabilities
- Syntax violations

(i) Data Flow Testing

It is a type of structural testing, which is used to find the test path of a program according to locations of definations and uses of variables in the programs.

- Advantages of Data Flow Testing
  - To find a variable that is used but never defined.
  - To find a variable that is defined but never used.
  - To find a variable that is defined multiple times before it used.
  - Deallocating a variable before it used.
- Disadvantages of Data Flow Testing
  - Time consuming and costly process.
  - Requires knowledge of programming languages.

(ii) Control Flow Testing

It is a type of software testing that uses program control flow as a model. Control flow testing is a structural testing strategy. This testing comes under white box ~~testing~~ testing. All structure design, code and implementation of the software should be known to the testing team. ~~Most~~ It is a graphical representation

of all path, which consist of two blocks (Entry and ~~Exist~~ Exit). The control enter into the flow graph through and entry block and leaves through the exit block. By this representative we can also find cyclomatic complexity and it is represented as V.

$V(G) = e - n + 2p$,

where $V(G)$: cyclomatic no of graph

e : no of edges in G, n: no of nodes in G

p: no of connected program parts

a] Perform data flow analysis on

i) The following function is supposed to exchange the integer value of the parameter 'Max' and 'Min' with the help of variable 'Help' if the value of variable 'Min' is greater than the value of the variable 'Max'.

```
void exchange (int &Min, int &M)
{   int Help;
    if (Min > Max)
    {   Max = Help;
        Max = Min;
        Help = Min;
    }
}
```

- Following anomalies ~~abnormalities~~ detected:

i] ur (undefined and read) - anamoly of variable Help:

- It is limited to function with the domain of this variable.
- It variable on the right side of the assignment and has undefined value.
- There was no initilization of variable when its checked.

ii] dd (defined and again defined) - anamoly of variable Max:
- The variable is assigned twice value and twice consecutively used on the left side of an assignment.
- The first assignment can either be omitted or the use of the first value has been forgetten.

iii] du (defined and unused) - anamoly of variable Help:
- The variable Help is assigned to another value that cannot be used anywhere in the last ~~function~~ assignment of the function.
- The ~~exact exact~~ exact reason is the variable is only void inside the function.

Corrected one:

```
void exchange (int &Min, int &Max)
{   int Help;
    if (Min > Max)
    {
      Help = Max;
      Max = Min;
      Min = Help;
    }
}
```

2] Billing rules

Usage (min)	Bill ($)
<100	400
101-200	50 cents for every additional minute
>200	10 cents for every additional minute

Source code for above application:

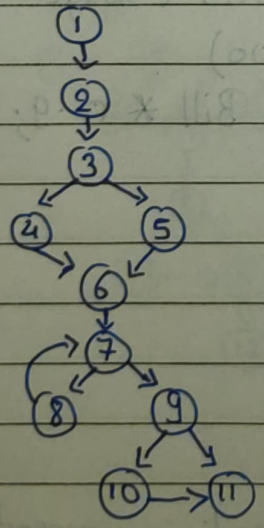
```
public static double calculateBill (int usage)
{   double Bill = 0;
    if (usage > 0)  { Bill = 40; }
    if (usage > 100)
    {   if (usage <= 200)
        { Bill = Bill + (usage - 100) * 0.5; }
        else
        { Bill = Bill + 50 + (usage - 200) * 0.1;
          if (Bill >= 100)
          {   Bill = Bill * 0.9;
          }
        }
    }
    return ~~att~~ Bill;
}
```

There is not a single anamolies present in the above program after performing data flow anaylsis.

6] Draw Control flow graph and calculate Cyclomatic Complexity

i] begin
int x, y, Power;
float z;
input (x,y);
if (x y < 0)
Power = -y;
else
Power = y;
z = 1;
while (Power != 0)
{ z = z * x;
Power = Power - 1; }
if (y < 0)
{ z = 1/z;
}

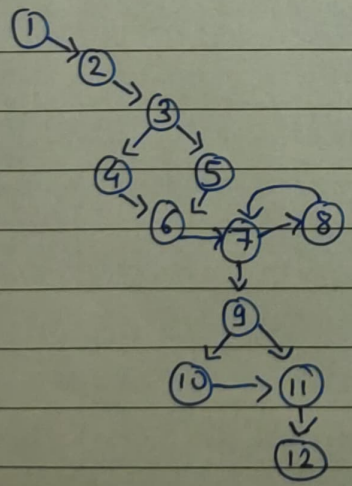
CFG:



Cyclomatic Complexity
$V(G) = e - n + 2(p)$, $e = 13, n = 11, p = 1$
$= 13 - 11 + 2(1)$
$= 2 + 2$
$\therefore V(G) = 4$

2]
```
void pow (int m, n)          — (1)
{ float q;                   > (2)
  int p;
  if (n < 0)                 — (3)
  p = 0 - n;                 — (4)
  else
  p = n;                     — (5)
  q = 1.0;                   — (6)
  while (p != 0)             — (7)
  { q = q * m;               > (8)
    p = p - 1; }
  if (n < 0)                 — (9)
   q = 1.0/q                 — (10)
   printf("%f", q)           — (11)
}                            — (12)
```

CFG:



Cyclomatic complexity

$V(G) = e - n + 2(p)$, $e = 14, n = 12, p = 1$

$= 14 - 12 + 2(1)$

$= 2 + 2$

$V(G) = 4$

Conclusion:

We have performed data and control flow analysis and calculated cyclomatic complexity as well.