Data Flow-Based Testing

- Data flow testing is used to analyze the flow of data in the program.
- Selects test paths of a program
  - According to the locations of
    - Definitions and uses of different variables in a program.
- It has nothing to do with data flow diagrams.

Data Flow-Based Testing

- For a statement numbered S,
  - DEF(S) = {X/statement S contains a definition of X}
  - USES(S)= {X/statement S contains a use of X}
  - Example: 1: a=b; DEF(1)={a}, USES(1)={b}.
  - Example: 2: a=a+b; DEF(1)={a}, USES(1)={a,b}.
- If a statement is a loop or if condition then its DEF set is empty and USE set is based on the condition of statement s.

Data Flow-Based Testing

- A variable X is said to be live at statement S1, if
  - X is defined at a statement S:
  - There exists a path from S to S1 not containing any definition of X.

- Data Flow Testing is to find the situations that can interrupt the flow of the program.
- It detects anomalies in the flow of the data by detecting associations between values and variables.
- These anomalies are:
  - A variable is defined but not used or referenced,
  - A variable is used but never defined,
  - A variable is defined twice before it is used

Disadvantages of Data Flow Testing

- Time consuming and costly process
- Requires knowledge of programming languages

DU Chain Example

```
X(){
 a=5; /* Defines variable a */
 While(C1) {
    if (C2)
          b=a*a;   /*Uses variable a */
          a=a-1; /* Defines variable a */
    }
  print(a); } /*Uses variable a */
```

Definition-use chain (DU chain)

- [X,S,S1],
  - S and S1 are statement numbers,
  - X in DEF(S)
  - X in USES(S1), and
  - the definition of X in the statement S is live at statement S1.

Test Criteria

- One simple data flow testing strategy:
  - Every DU chain in a program be covered at least once.
- Data flow testing strategies:
  - Useful for selecting test paths of a program containing nested if and loop statements.

- Predicate use (p-use)
  - If the value of a variable is used to decide an execution path is considered as predicate use (p-use).
- Computation use (c-use)
  - If the value of a variable is used to compute a value for output or for defining another variable.

Example

1. read x;	
2. If(x>0)	(1, (2, t), x), (1, (2, f), x)
3. a= x+1	(1, 3, x)
4. if (x<=0) {	(1, (4, t), x), (1, (4, f), x)
5. if (x<1)	(1, (5, t), x), (1, (5, f), x)
6. x=x+1; (go to 5)	(1, 6, x)
else	
7. a=x+1	(1, 7, x)
8. print a;	(6,(5, f)x), (6,(5,t)x)
	(6, 6, x)

Test criteria

- **All c-use coverage**
- **All c-use some p-use coverage**
- **All p-use some c-use coverage**

Mutation Testing

- The software is first tested:
  - using an initial testing method based on white-box strategies we already discussed.
- After the initial testing is complete,
  - mutation testing is taken up.
- The idea behind mutation testing:
  - make a few arbitrary small changes to a program at a time.

Mutation Testing

- Each time the program is changed,
  - it is called a mutated program
  - the change is called a mutant.

Mutation Testing

- A mutated program:
  - tested against the full test suite of the program.
- If there exists at least one test case in the test suite for which:
  - a mutant gives an incorrect result,
  - then the mutant is said to be dead.

Mutation Testing

- If a mutant remains alive:
  - even after all test cases have been exhausted,
  - the test suite is enhanced to kill the mutant.
- The process of generation and killing of mutants:
  - can be automated by predefining a set of primitive changes that can be applied to the program.

Mutation Testing

- The primitive changes can be:
  - altering an arithmetic operator,
  - changing the value of a constant,
  - changing a data type, etc.

Mutation Testing

- A major disadvantage of mutation testing:
  - computationally very expensive,
  - a large number of possible mutants can be generated.