Week-4

Unit 4: Structure-based techniques/ Whitebox testing/ glass box techniques

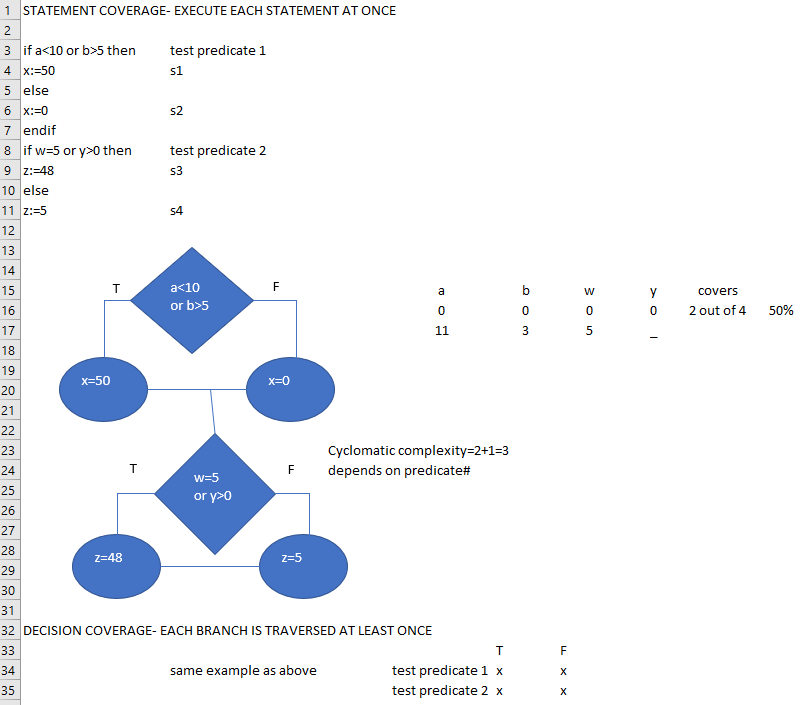
# Introduction

* White box testing is used for verification
* It provides statement coverage, decision coverage, decision/condition coverage, multiple condition coverage, path coverage
* Applicable at unit level, service level and little bit at integration level. Not applicable at overall system level testing
* Find code anomalies
* Code coverage can be assessed in term of – control flow, data flow

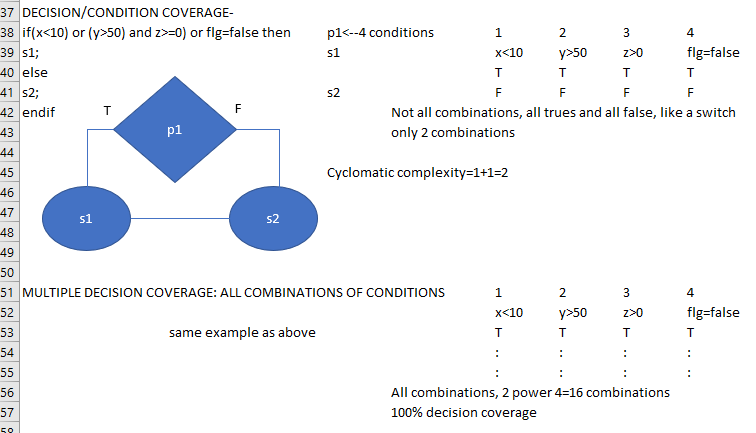
# Dynamic Analysis

## Control flow:

* + describes how well code is evaluated on different paths taken
  + Lower level testing done by individual developer
  + Analyze code coverage obtained by executing requirement based test cases
    - Undocumented requirements, dead code, incomplete test case requirement
  + Control flow coverage levels-
    1. Statement coverage:
       1. Create control flow diagram (CFG)- flow chart. Diamond shape if for test predicate and oval shape is for test statement
    2. Decision coverage:
       1. develop test cases such that each branch is traversed at least once. Not combination, but test each branch. True & false decision of the test predicate. Decision coverage satisfies statement coverage. Vice versa might be true or false. So, decision coverage is stronger criteria than statement coverage.



* + 1. Decision/condition coverage:
       1. Develop test cases such that each condition in a decision takes on all possible outcomes at least once & each decision takes on all possible outcomes at least once.
       2. Ex: if (x<10 or y>50 and z>=0 or Flg=F)🡪 behave like a “switch”. Only 2 test cases: TTTT and FFFF
    2. Multiple condition coverage:
       1. Develop test cases such that all combinations of conditions in a decision are tested.
       2. Ex: if(x<10) or (y>50) and (z>=0) or FA6=F🡨 4 condition or predicate, each with true & false🡪 2 power 4-=16.



# Static Analysis

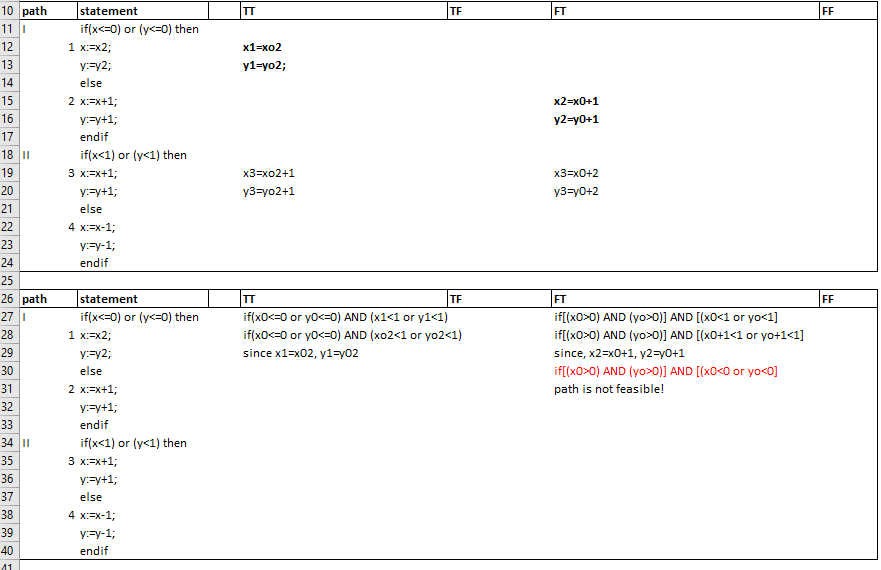
* static analysis does NOT require execution of a program to create and evaluate test cases.
* Static analysis attempts to model the flow of data in a program
  + Where are variables defined
  + Where are variables used
* and provide insight into areas such as data flow anomalies.

## Data flow:

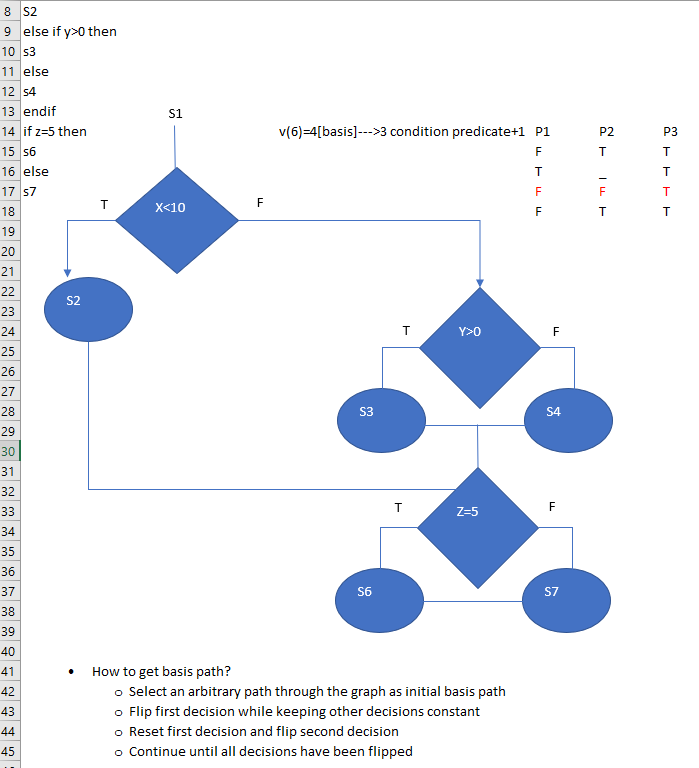
* Data flow anomalies example:
  + Variable defined and then redefined without being referenced
  + Referencing an undefined variable
  + Defining a variable but never using it
  + Numerous tools to perform anomaly detection
* Huang’s theorem

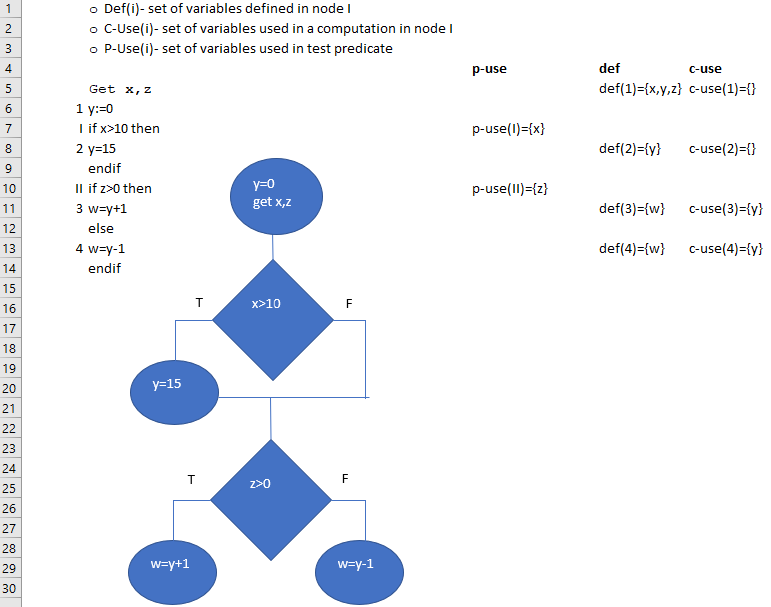
Symbolic execution: Symbolic execution is a technique, for formally characterizing a path domain, identifying a path condition.

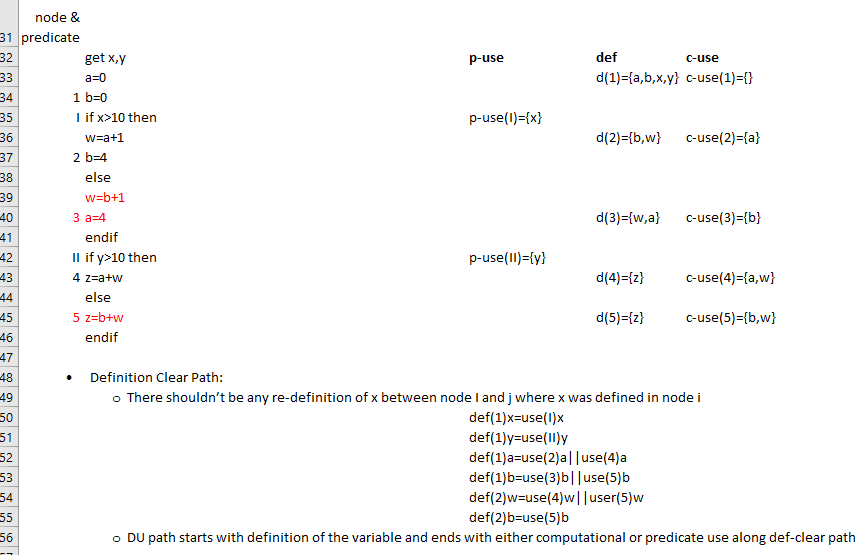
* + Execute a path that we want to get into program
  + Execute to determine/prove that a path in the program cannot be executed.
  + A variable x will have succession of symbolic values: A0, A2, A2…where subscripts refer to the number of the previous statement executed.



* Path condition: So, the symbolic path condition is going to be defined as an expression in terms of initial symbolic values that is necessary for us to achieve if we're going to try to go in and execute a particular path through the code.
* Branch coverage through structured testing: So that as we go through and select the subset of basis paths based on the cyclomatic complexity, it will guarantee for us branch coverage.
* How to get basis path?
  + Select an arbitrary path through the graph as initial basis path
  + Flip first decision while keeping other decisions constant
  + Reset first decision and flip second decision
  + Continue until all decisions have been flipped



* Data flow testing: annotate control flow graph with 3 sets for each node
  + Def(i)- set of variables defined in node I
  + C-Use(i)- set of variables used in a computation in node I
  + P-Use(i)- set of variables used in test predicate
* Definition Clear Path:
  + There shouldn’t be any re-definition of x between node I and j where x was defined in node i
* Definition Use(DU) path coverage:
  + DU path starts with definition of the variable and ends with either computational or predicate use along def-clear path
  + So what I've done is simply set the stage, all done by tools, analyzing the program, looking at where variables are defined, looking at where variables are used, thinking about definition clear paths, recognizing a path from where a variable is defined to where it's used, and then our last step here is ultimately to develop test cases, and to measure the coverage.
  + 



* + 