STANNS COLLEGE OF ENGINEERING AND TECHNOLOGY
SOFTWARE TESTING METHODOLOGIES

UNIT-III

Dynamic Testing II: White-Box Testing: need, Logic coverage criteria, Basis path testing, Graph matrices, Loop testing, data flow testing, mutation testing

Static Testing: inspections, Structured Walkthroughs, Technical reviews

Dynamic Testing II (White Box Testing)

Need:

--White box testing (WBT) is also called Structural or Glass box testing.
--White box testing involves looking at the structure of the code.
--We need WBT to ensure:
-->That all independent paths within a module have been exercised at least once.
-->All logical decisions verified on their true and false values.
-->All loops executed at their boundaries and within their operational bounds internal data structures validity.

Logic Coverage Criteria:

Structural or White-Box Testing considers the program code and test cases are designed based on the logic of the program such that every element of the logic is covered. Therefore the intention in WTB is to cover the whole logic. Differetn forms of logic coverage are:
--Statement Coverage
--Decision or Branch Coverage
--Condition Coverage
--Decision / Condition Coverage
--Multiple condition Coverage

Example:

```
scanf("%d%d",&x,&y);
while(x!=y)
{
        if(x>y)
                x=x-y;
        else
                y=y-x;
}
printf("x=%d  y=%d",x,y);
```

Statement Coverage:

It is assumed that if all the statements of the module are executed once, every bug will be notified. If we want to cover every statement in the above code, then the following test cases must be designed:
Test case 1: x = y = n, where n is any number
Test case 2: x = n, y = n', where n and n' are different numbers.
Test case 3: x>y
Test case 4: x<y

Decision or Branch Coverage:

Branch Coverage states that each decision takes on all possible outcomes. In other words each branch direction must be traversed atleast once. The different test cases that can be designed are:

Test case 1: x = y	Test case 3: x < y
Test case 2: x != y	Test case 4: x > y

Condition Coverage:

It states that each condition in a decision takes on all possible outcomes at least once. For example consider the following statement: while((I<5) && (J<COUNT))
The different test cases are:

Test case 1: I <= 5, J < COUNT
Test case 2: I > 5, J > COUNT

Decision / Condition Coverage:
If the decision *If (A && B)* is being tested, then the test cases would be like:

Test Case 1: A is True, B is False.
Test Case 2: A is False, B is True.

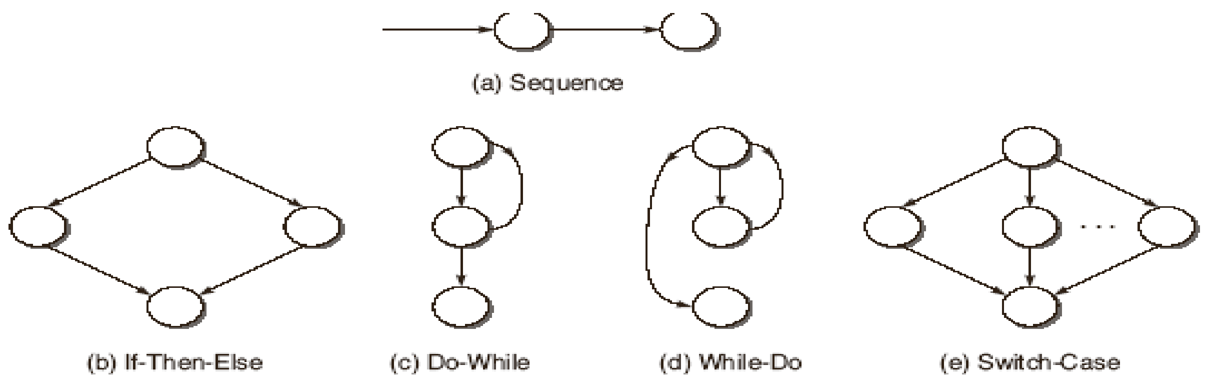
Multiple condition Coverage:
In case of multiple conditions, for the sattement *If (A && B)* the test cases would be:

Test Case 1: A = TRUE, B = TRUE
Test Case 2: A = TRUE, B = FALSE
Test Case 3: A = FALSE, B = TRUE
Test Case 4: A = FALSE, B = FALSE

Basis Path Testing:
Basis Path Testing (BPT) is the oldest structural testing technique . The technique is based on the control strucure of the program. Based on the control structure, a flow graph is prepared and all the possible paths can be covered and executed during testing. The guidelines for effective path testing are:
--Path Testing is based on control structure of the program for which flow graph is prepared.
--It requires complete knowledge of the program’s structure.
--It is closer to the developer and used by him to test his module.
--The effectiveness of path testing is reduced with the increase in size of software under test.
--Choose enough paths in a program such that maximum logic coverage is achieved.

Control Flow Graph:
The control flow graph is a graphical representation of control structure of a program. Flow graph can be prpepared as a directed graph. A Directed graph (V,E) consists of set of vertices V and edges E. Flow graph notations are:
--Node, Edges or links, Decision Node,
--Junction node :A node with more than one arrow entering it is called junction node
--Regions: Areas bounded by edges and nodes

Flow Graph Notations fro Different Programming Constructs: Flow graph is also known as **Decision to Decision (D&D) graph**.



Path Testing Terminology:

Path: A path through a program is a sequence of instructions or statements that starts at an entry, junction, or decision and ends at another or possibly the same junction, decision or exit.

Segment: Path consists of segments. The smallest segment is a link, that is, a single process that lies between two nodes.

Path Segment: A path segment is a succession of consecutive links that belongs to some path.

Length of a Path: The length of a path is measured by the number of links in it and not by the number of instructions or statements executed along the path.

Independent Path: An independent path is any path through the graph that introduces at least one new set of processing statements or new conditions.

Cyclomatic Complexity:

--It provides a quantitative measure of the logical complexity of a program
--Defines the number of independent paths in the basis set
--Provides an upper bound for the number of tests that must be conducted to ensure all statements have been executed at least once
--Can be computed three ways
-->The number of regions
-->V(G) = E – N + 2, E is the number of edges and N is the number of nodes in graph G
-->V(G) = P + 1, where P is the number of predicate nodes in the flow graph G

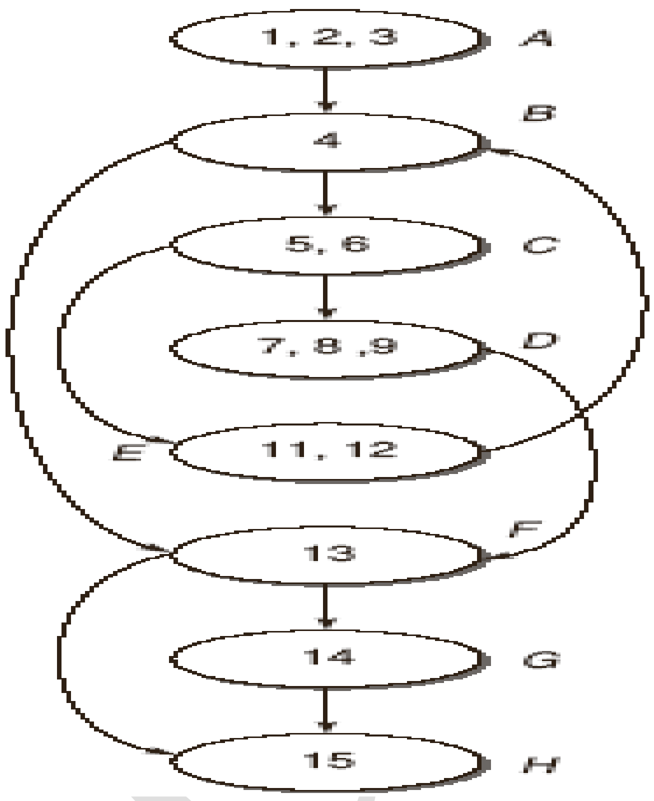
Guidelines for Basis Path Testing:

-- Draw the flow graph using the code provided for which we have to write test cases.
--Determine the cyclomatic complexity of the flow graph.
--Cyclomatic complexity provides the number of independent paths.
--Based on every independent path, choose the data such that this psth is executed.

Example:

```
main()
{
        int number, index;
1.      printf("Enter a number");
2.      scanf("%d, &number);
3.      index = 2;
4.      while(index <= number - 1)
5.      {
6.              if (number % index == 0)
7.              {
8.                      printf("Not a prime number");
9.                      break;
10.             }
11.             index++;
12.             }
13.             if(index == number)
14.                     printf("Prime number");
15.     } //end main
```

Flow graph for above example is:



Cyclomatic Complexity is:

-->V(G) = e – n + 2
= 10 – 8 +2
= 4
-->V(G) = Number of predicate nodes + 1
= 3 (Nodes B,C and F) + 1
= 4

-->V(G) = No. of Regions
= 4 (R1, R2, R3, R4)

Applications of path testing:
--Through testing / More coverage
--Unit testing
--Integration testing
--Maintenance testing
--Testing effort is proportional to complexity of the software.

Graph Matrices:

Flow graph is an effective aid in path testing. However, path tracing with the use of flow graphs may be a cubersome and time consuming activity. Graph Matric ,a data structure is the solution which can assist in developing a tool for automation of path tracing.

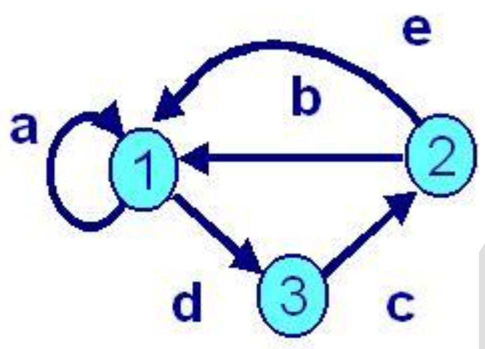
Graph Matrix:

A Graph Matrix is a square matrix whose size is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node, and matrix entries correspond to connections between nodes.

Connection Matrix:

If we add link weights to each cell entry, then graph matrix can be used as a powerfull tool in testing. The links between two nodes are assigned a link weight which becomes the entry in the cell of matrix. The link weight provides information about control flow.

Example:



Flow Graph

	1	2	3
1	a		d
2	b+e		
3		c	

Graph Matrix

	1	2	3	4
1			1	1
2				
3		1		
4		1		

Connection Matrix

1			1	1	2-1=1
2					
3		1			1-1=0
4		1			1-1=0

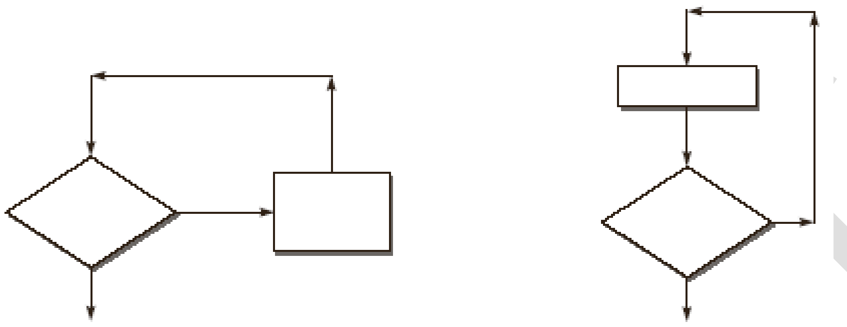
1+1=2=V(G)

Calculation of V(G)

Loop Testing:

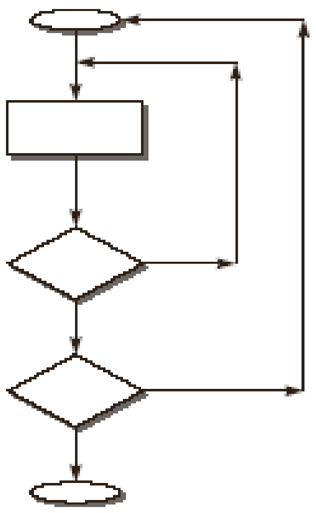
If loops are not tested properly, bugs can go undetected. Loop testing can be done effectively while performing unit testing by the developer. Different kinds of loops available for testing are:

Simple loops:



The following test cases should be considered for simple loops while testing them:
--> Check whether the loop control variable is negative.
--> Write one test case that executes the statements inside the loop.
--> Write test cases for a typical number of iterations through the loop.
--> Write test cases for checking the boundary values of maximum and minimum number of iterations defined (say min and max) in the loop. It means we should test for the min, min+1, min-1, max-1, max and max+1 number of iterations through the loop.

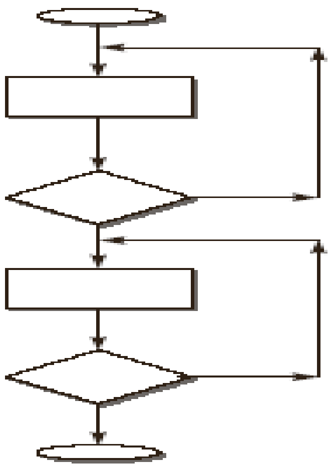
Nested Loops:



If we have nested loops in the program it is difficult to test. If we adopt the approach of simple tests to test the nested loops, then the number of possible test cases grow geometrically. Thus the strategy is to start with the innermost loops while holding outer loops to their minimum values. Continue this outward in this manner until all loops have been covered

Concatenated loops:

Two loops are concatenated if it is possible to reach one after exiting the other while still on a path from entry to exit. If two loops are not on the same path, then they are not coancatenated.



Data flow Testing:

Data Flow Testing is a white-box testing technique that can be used to detect improper use of data values due to coding errors. Errors may be unintentionally introduced in a program by programmers. For instance:
--A programmer might use a variable without defining it.
--He may define a vraible but not initialize it.

Example: int a;   if (a==100) { }

In this way, data flow testing gives a chance to look out of inappropriate data definition, its use in predicates, computations and termination. It defies potential bugs by examining the patterns in which that a peice of data is used.

State of a Data Object: A data object can be in the following state:
--Defined(d:) A data object is called defined when it is initialized.
--Killed / Undefined / Released (k): When the data has been reinitialized ot the scope of a loop control variable is finished.
--Usage (u): When the data object is on the right side of assignemt or used as a control variable in a loop. In general, we say that the usage is either **computaional use** (c-use) or **predicate use** (p-use).

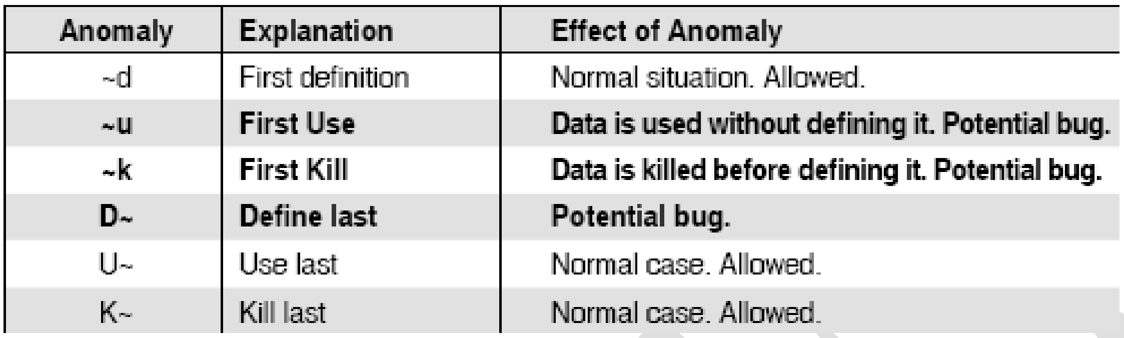
Data Flow Anomalies:
Data flow anomalies represent the patterns of data usgae which may lead to an incorrect execution of the code. An anomaly is denoted by a two character sequence of actions.

Anomaly	Explanation	Effect of Anomaly
du	Define-use	Allowed. Normal case.
dk	**Define-kill**	**Potential bug. Data is killed without use after definition.**
ud	Use-define	Data is used and then redefined. Allowed. Usually not a bug because the language permits reassignment at almost any time.
uk	Use-kill	Allowed. Normal situation.
ku	**Kill-use**	**Serious bug because the data is used after being killed.**
kd	Kill-define	Data is killed and then redefined. Allowed.
dd	**Define-define**	**Redefining a variable without using it. Harmless bug, but not allowed.**
uu	Use-use	Allowed. Normal case.
kk	**Kill-kill**	**Harmless bug, but not allowed.**

In addition to the above two character data anomalies, there may be a single character data anomalies. To represent these types of anomalies, we take the following conventions:

~x: indicates all prior actions are not of interest to x.
x~: indicates all post actions are not of interest to x.

All single-character data anomalies are listed below:



Anomaly	Explanation	Effect of Anomaly
~d	First definition	Normal situation. Allowed.
~u	**First Use**	**Data is used without defining it. Potential bug.**
~k	**First Kill**	**Data is killed before defining it. Potential bug.**
D~	**Define last**	**Potential bug.**
U~	Use last	Normal case. Allowed.
K~	Kill last	Normal case. Allowed.

Terminology used in Data Flow Testing:
-->Definition Node: Input statements, Assignment statements, Loop control statements, Procedure calls, etc.
-->Usage Node: Output statements, Assignment statements (Right), Conditional statements, Loop control statements, etc.
-->Loop Free Path Segment: It is a path segment for which every node is visited once at most.
-->Simple Path Segment: It is a path segment in which at most one node is visited twice.
-->Definition-Use Path (du-path):A du-path with respect to a variable v is a path between definition node and usage node of that variable. Usgae node can be p-usage or c-usgae node.
-->Definition-Clear path(dc-path): A dc-path with respect to a variable v is a path between definition node and usage node such that no other node in the path is a defining node of variable v.

Static Flow Testing: With static analysis, the source code is analysed without executing it. Let us consider an example of an application given below:
Example:

```
main()
 {
 int a=10,b=10,c;
 c=a+b;
 printf(“Value=%d”,c);
 }
```

For Variable 'a':

Pattern	Line Number	Explanation
-d	3	Normal Case: Allowed
du	3-4	Normal Case: Allowed
uk	4	Normal Case: Allowed

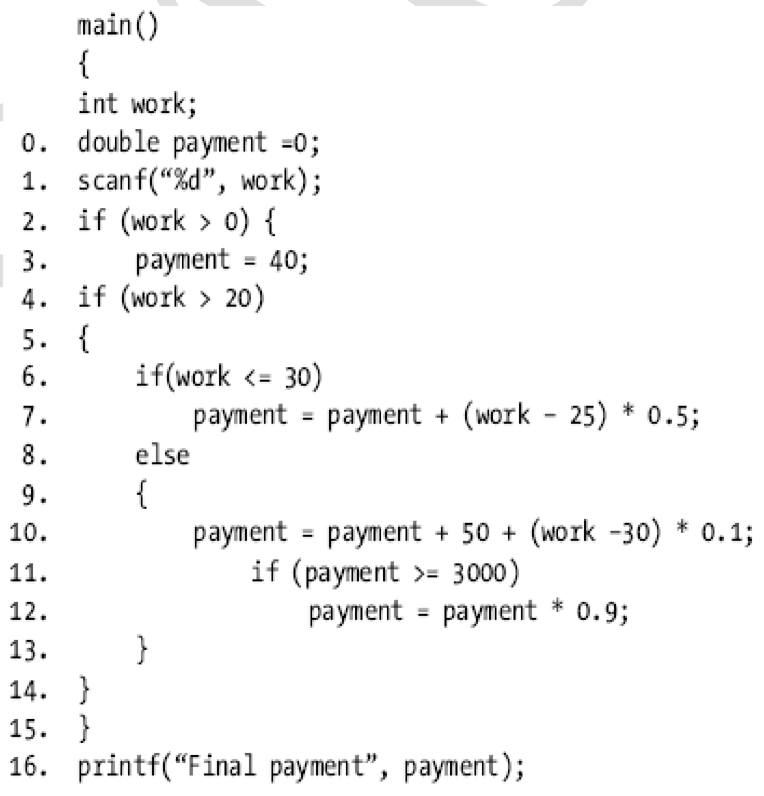
For variable 'c':

Pattern	Line Number	Explanation
-d	3	Normal Case: Allowed
du	3-4	Normal Case: Allowed
uk	5	Normal Case: Allowed

It is not always possible to determine the state of a data variable by just ststic analysis of the code. For example, if the data variable in an array is used as an index for a collection of data elements, we cannot determine its state by static analysis.

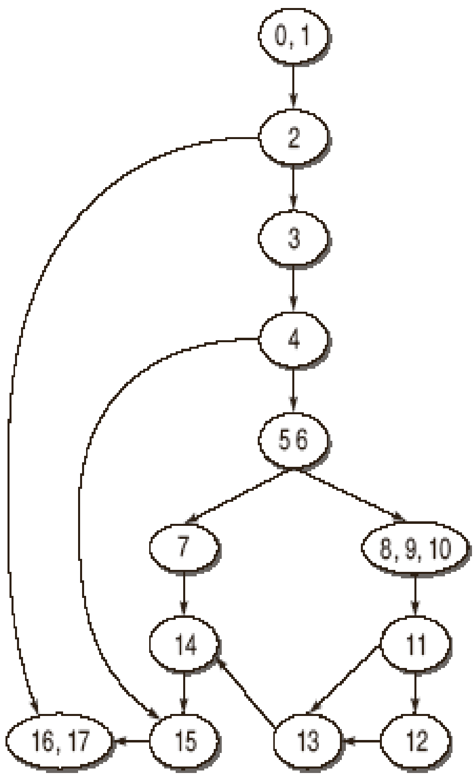
Dynamic Data Flow Testing:
Dynamic data flow testing is performed withthe intention to uncover possible bugs in data usage during execution of the code. Various strategies are employed for the creation of test cases.All these test strategies are defined below:
-->All-du Paths (ADUP): IT states that every du path from every definition of every variable to every use of that definition should be excercised under some test.
-->All-uses (AU): This states that for every use of the variable, there is a path from the definition of that variable to the use.
-->All-p-uses / Some-c-uses (APU + C): This stratagey states that for every variable and every definition of that variable, include at least one dc- path from the definition to every predicate use.
-->All-c-uses / Some-p-uses (ACU + P): This stratagey states that for every variable and every definition of that variable, include at least one dc- path from the definition to every computational use.
-->All-Predicate-Uses(APU): It is derieved from the the APU+C stratagey and states that for every variable there is a path from every definition to every p-use of that definition.
-->All-Computational-Uses(ACU): It is derieved from the the ACU+P stratagey and states that for every variable there is a path from every definition to every c-use of that definition.
-->All-Definition (AD): It states that every definition of every variable should be covered by atleast one use of that variable.

Example:



```
    main()
    {
    int work;
0.  double payment =0;
1.  scanf("%d", work);
2.  if (work > 0) {
3.      payment = 40;
4.  if (work > 20)
5.  {
6.      if(work <= 30)
7.          payment = payment + (work - 25) * 0.5;
8.      else
9.      {
10.         payment = payment + 50 + (work -30) * 0.1;
11.             if (payment >= 3000)
12.                 payment = payment * 0.9;
13.     }
14. }
15. }
16. printf("Final payment", payment);
```

Flow graph:



Data Flow Graph

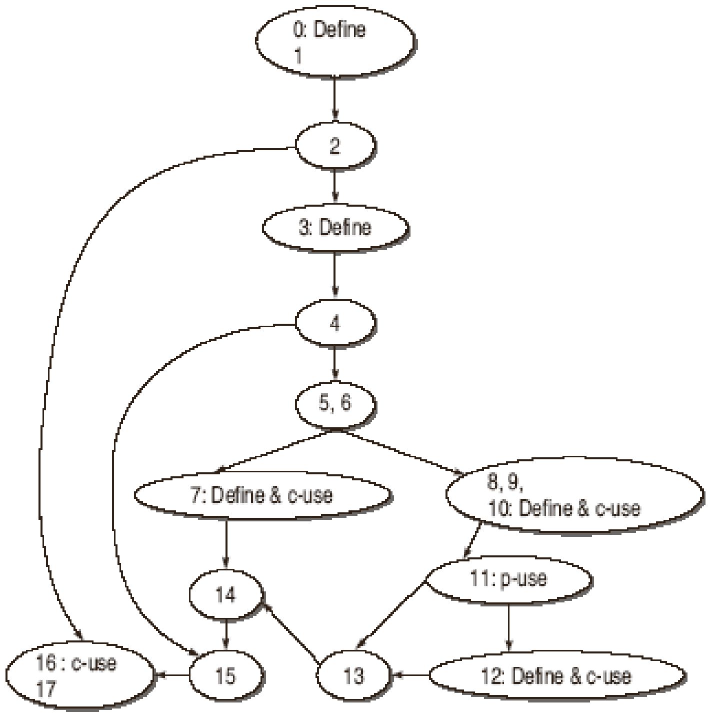
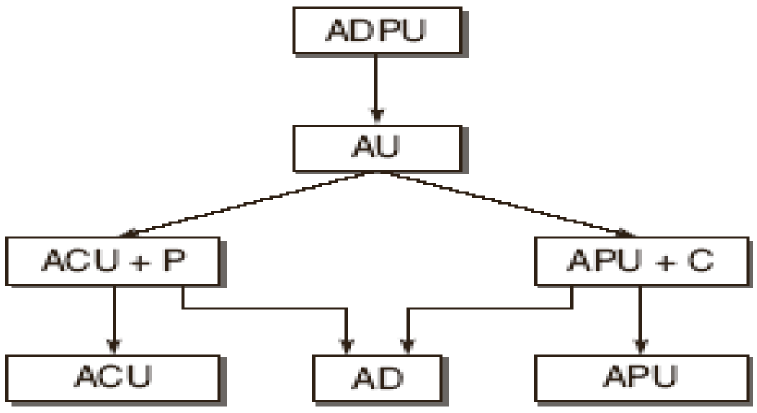


Figure 5.13 Data flow graph for 'payment'

Ordering of Data flow testing Strategies:



Mutation Testing

Mutation testing is the process of mutating some segment of code(putting some error in the code) and then testing this mutated code with some test data. If the test data is able to detect the mutations in the code. Mutation testing helps a user create test data by interacting with the user to iteratively strengthen the quality of test data. During mutation testing, faults are introduced into a program by creating many versions of the program, each of which contains one fault. Test data are used to execute these faulty programs with the goal of causing each faulty program to fail.

Faulty programs are called *mutants* of the original program and a mutant is said to be *killed* when a test case causes it to fail. When this happens, the mutant is considered *dead*

Primary Mutants:

When the mutants are single they are called as primary nutants. Mutation operators are dependent on programming languages.

Example:
```
if(a>b)
 x = x+y;
 else
x=x-y;
printf("%d",x);
```

Example mutants are:

M1: x = x – y;   M2: x = x / y;   M3: x = x+1;M4:   printf("%d",y);

Secondary Mutants:

Example:
```
if(a<b)
 c=a;
M1: if(a<=b-1)
 c=a;

M2: if(a+1 <=b)
 c=a;

M3: if(a==b)
 c=a+1;
```

Static Testing

Features of Software Testing:

--Static testing techniques do not demonstrate that the software is operational or one function of software is working;

--They check the software product at each SDLC stage for conformance with the required specifications or standards. Requirements, design specifications, test plans, source code, user's manuals, maintenance procedures are some of the items that can be statically tested.

--Static testing has proved to be a cost-effective technique of error detection.

--Another advantage in static testing is that a bug is found at its exact location whereas a bug found in dynamic testing provides no indication to the exact source code location.

Types of Static Testing

-->Software Inspections
-->Walkthroughs
-->Technical Reviews

Inspections:

-->Inspection process is an in-process manual examination of an item to detect bugs.
-->Inspection process is carried out by a group of peers. The group of peers first inspects the product at individual level. After this, they discuss potential defects of the product observed in a formal meeting.
-->It is a very formal process to verify a software product. The documents which can be inspected

are SRS, SDD, code and test plan.
-->Inspection process involves the interaction of the following elements:
a) Inspection steps b) Roles for participants c)Item being inspected

Inspection Team:

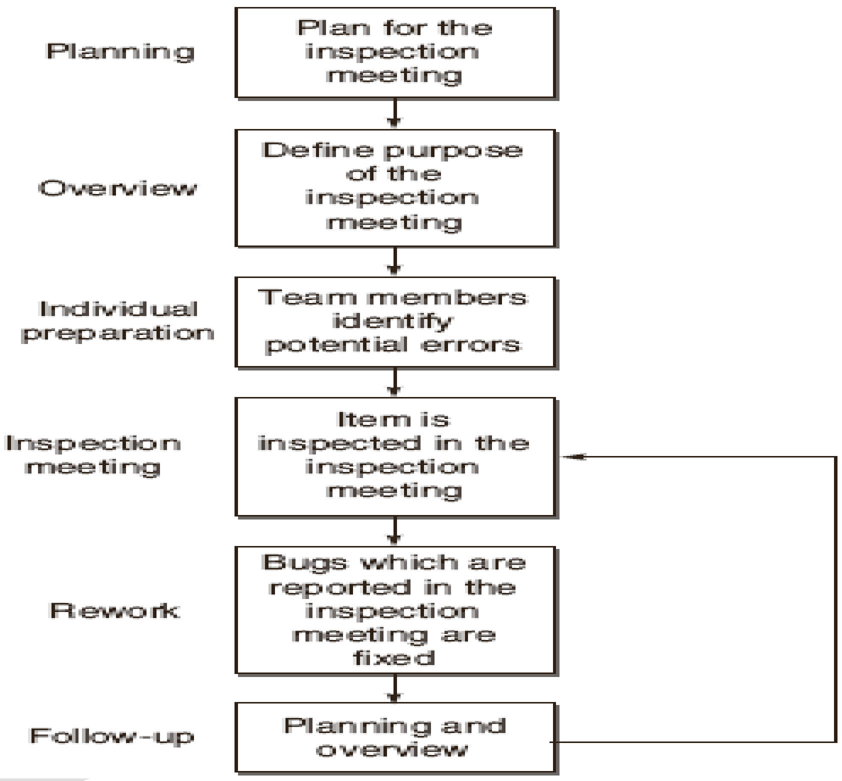
-->Author / Owner / Producer: A programmeror designer responsible for producing the program or document.

-->Inspector: A peer member of the team, i.e he is not a manager or supervisor. He is not directly related to the product under supervision and may be concerned with some other products.

-->Moderator: A team member who manages the whole inspection process. He schedules, leads, and controls the inspection session.

-->Recorder: One who records all the results of the inspection meeting.

Inspection Process:


Planning
Plan for the inspection meeting
Overview
Define purpose of the inspection meeting
Individual preparation
Team members identify potential errors
Inspection meeting
Item is inspected in the inspection meeting
Rework
Bugs which are reported in the inspection meeting are fixed
Follow-up
Planning and overview

Planning: During this phase the following is executed:
-->The product to be inspected is being identified.
-->A moderator is assigned.
-->The objective of the inspection is stated i.e whether the inspection is to be conducted for defect detection or something else.

During planning, the moderator performs the following activities:
--Assures that the product is ready for inspection.
--Selects the inspection team and assigns their roles.
--Schedules the meeting venue and time.
--Distributes the inspection material like the item to be inspected, client lists etc.

Overview: In this stage, the inspection team is provided with the background information for inspection. The author presents the rationle of the product, its relationship to the rest of the products being developed, its function and intended use and the approach used to develop it.
Individual Preparation: After the overview, the reviewers individually prepare themselves for the inspection process by studying the documents provided to them in the overview session. They point out potential errors or problems found and record in a log. This log is then submitted to the moderator. The moderator compiles the logs of different members and gives a copy of this compiled list to the author of the inspected item.

Inspection Meeting: Once all the initial preparation is complete, the actual inspection meeting can start. The inspection meeitng starts with the author of the inspected item who has created it. The author first discusses every issue raised by different members in the compiled log file. After the discussion, all the members arrive at a consensus whether the issues pointed out are in fact errors and if they are errors, should they be admitted by the author.

Rework: The summary list of the bugs that arise during the inspection meeting needs to be reworked by the author. The auhor fixes all these bugs and reports back to the moderator.

Follow-Up: It is the responsibility of the moderator to check that all the bugs found in the last meeting have been resolved. The document is then approved for release.

Benefits of Inspection Process:

-->Bug Reduction: Accorrding to the report that through the inspection process in IBM, the number of bugs per thousand lines of code has been reduced by two thirds.

-->Bug Prevention: Based on the experience of previous inspections, analysis can be made for future inspections or projects, therby preventing the bugs which have appeared earlier.

-->Productivity: Since all phases of SDLC may be inspected without waiting for code development and its execution, the cost of finding bugs decreases and increases productivity.

-->Real-time Feedback to Software Engineers: Developers find out the type of mistakes they make and what is the error density. Since they get this feedback in the early stage of the development, they may improve their capability.

-->Reduction in Development Resource: Inspections reduce the effort required for dynamic testing and any rework during design and code, thereby causing an overall net reduction in the development resource.

-->Quality Improvement: The direct consequence of static testing also results in the improvement of quality of the final product.

-->Project Management
-->Checking Coupling and Cohesion
-->Learning through Inspection
-->Process Improvement

Effectiveness of Inspection Process:
In an analysis, the inspection process was found to be effective as compared to structural testing because the inspection process alone found 52% errors. So the error detection ratio can be

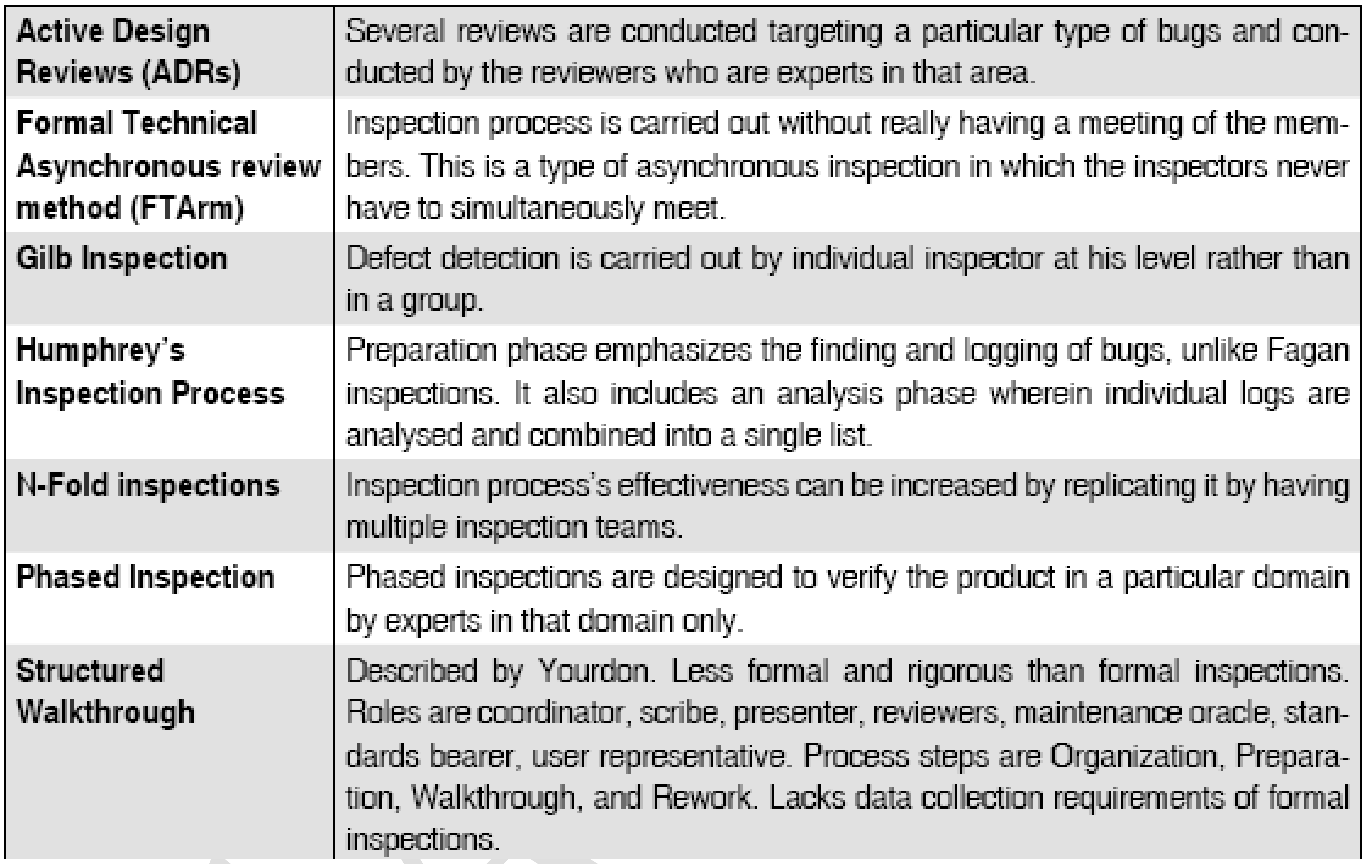
specified as:

Error found by an inspection
Error detection efficiency = -------------------------------------- * 100
Totla errors in the team before inspection

Variants of Inspection Process:

--> "**For deatialed diagrams refer the ppt**"

--> "**Based on the diagram of each variant you can explain in your own words**"



Active Design Reviews (ADRs)	Several reviews are conducted targeting a particular type of bugs and conducted by the reviewers who are experts in that area.
Formal Technical Asynchronous review method (FTArm)	Inspection process is carried out without really having a meeting of the members. This is a type of asynchronous inspection in which the inspectors never have to simultaneously meet.
Gilb Inspection	Defect detection is carried out by individual inspector at his level rather than in a group.
Humphrey's Inspection Process	Preparation phase emphasizes the finding and logging of bugs, unlike Fagan inspections. It also includes an analysis phase wherein individual logs are analysed and combined into a single list.
N-Fold inspections	Inspection process's effectiveness can be increased by replicating it by having multiple inspection teams.
Phased Inspection	Phased inspections are designed to verify the product in a particular domain by experts in that domain only.
Structured Walkthrough	Described by Yourdon. Less formal and rigorous than formal inspections. Roles are coordinator, scribe, presenter, reviewers, maintenance oracle, standards bearer, user representative. Process steps are Organization, Preparation, Walkthrough, and Rework. Lacks data collection requirements of formal inspections.

Reading Techniques:

A reading technique can be defined as a series of steps or procedures whose purpose is to guide an inspector to accquire a deep understanding of the inspected software product. Thus reading technique can be regarded as a mechanism for the individual inspector to detect defects in the inspected product. The various reading techniques are:

Ad-hoc Method: The word ad-hoc only refers to the fact that no technical support on how to detect defects in a software artifact is given them. In this case, defect detection fully depends on the skills, knowledge, and experience of an inspector.

Checklists: A checklist is a list of items that focus the inspectors attention on specific topics, such as common defects or organizational rules, while reviewing a software document.

Scenario – Based Reading: Different methods developed based on scenario based reading are:

-->***Perspective based Reading:*** **S**oftware item should be inspected from the perspective of different stakeholders Inspectors of an inspection team have to check software quality as well as the software quality factors of a software artifact from different perspectives.

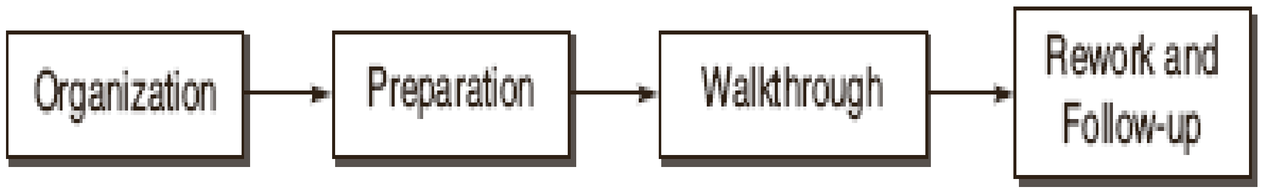
-->*Usage based Reading*: This method given is applied in design inspections. Design documentation is inspected based on use cases, which are documented in requirements specification.

-->*Abstraction driven Reading:* This method is designed for code inspections. In this method, an inspector reads a sequence of statements in the code and abstracts the functions these statements compute.

-->*Task driven Reading:* This method is also for code inspections . In this method, the inspector has to create a data dictionary, a complete description of the logic and a cross-reference between the code and the specifications.

->*Function-point based Scenarios:* This is based on scenarios for defect detection in requirements documents [103]. The scenarios, designed around function-points are known as the Function Point Scenarios. A Function Point Scenario consists of questions and directs the focus of an inspector to a specific function-point item within the inspected requirements document.

Structured Walkthroughs:



-->It is a less formal and less rigorous technique as compared to inspection. The very common term used in the literature for static testing is Inspection but it is for very formal process. If you want to go for a less formal having no bars of organized meeting, then walkthroughs are a good option.

-->A review is similar to an inspection or walkthrough, except that the review team also includes management. Therefore, it is considered a higher-level technique than inspection or walkthrough.

-->A technical review team is generally comprised of management-level representatives of the User and Project Management. Review agendas should focus less on technical issues and more on oversight than an inspection.

--> A typical structured walkthrough team consists of:
- --*Coordinator*: Organizes , moderates and follows up the walkthrough
- --*Presenter / Developers*: Introduces the item being inspected.
- --*Scrib / Recorder*: Notes down the defects
- --*Reviewer / Tester*: Finds the defects in the item.
- --*Maintenance Oracle*: Focuses on future maintenance of the project.
- --*Standards Bearer*: Assesses adherence to standards
- --*User Representative / Accredation Agent*: Reflects the needs of the user.

Technical Review:

--A review is similar to an inspection or walkthrough, except that the review team also includes management. Therefore, it is considered a higher-level technique than inspection or walkthrough.

--A technical review team is generally comprised of management-level representatives of the User and Project Management. Review agendas should focus less on technical issues and more on oversight than an inspection.