Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Quality Assurance and Testing (SEng4112)

Chapter Five

Software Testing Techniques

Objective

- **After successful completion of this chapter, students will be able to understand**
  - ✓ The static and dynamic software testing techniques
  - ✓ White box and black box testing techniques
  - ✓ The benefits and limitations of those testing techniques

Software Testing Techniques

- Software testing technique includes the various strategies or approaches used to test an application to ensure how it functions and behaves as expected.
- Methods in which the internal structure, design or implementation of a software is tested
- These are software testing techniques which the organization must choose carefully, and implement on the software application.
- In order to get the most out of each type of testing, and choose the right tools for a given situation, it's crucial to understand the benefits and limitations of each type of testing technique.

Cont...

- There are two software testing techniques

1. Static Testing Techniques
2. Dynamic Testing Techniques

1. Static Testing Techniques

- Static software testing techniques are used for testing code, requirements and design documents and put review comments on the work document.
- In this type of testing, the code is not executed.
- It can be done manually or by a set of tools/software.
- With static testing, we try to find out the errors, code flaws and potentially malicious code in the software application.
- It starts earlier in development life cycle and hence, it is also called verification testing technique.

Cont...

- **In static testing technique, we use work documents such as:**
  - Requirement specifications
  - Design document
  - Source code
  - Test plans
  - Test cases
  - Test scripts
  - Help or user document

Static testing technique

- Static testing technique includes:
  - ✓ Informal Review
  - ✓ Walkthrough
  - ✓ Technical Review
  - ✓ Static Code Review
  - ✓ Management Review
  - ✓ Inspection
  - ✓ Audit

1.1. Informal Review

- Informal reviews are applied many times during the early stages of the life cycle of the document.
- A two persons team can conduct an informal review.
- In later stages, these reviews often involve more people and a meeting.
- The most important thing to keep in mind about the informal reviews is that they are not documented.

1.2. Walkthrough

- Walkthrough is not a formal process of review and is carried out by authors.
- The authors guide the participants through the document and try to understand their thought processes, in order to gain feedback and arrive at a common understanding.
- The author of the work product explains the product to his/her team.
- Participants can ask questions if any.
- Meeting is led by the author.

Cont...

- Scribe makes note of review comments
- Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
- Is especially useful for higher level documents like requirement specification, etc.

1.3. Technical review

- A technical review is carried out by a trained moderator or technical expert.
- It is less formal as compared to an inspection.
- It is usually carried out as a peer review without any participation from the management.
- A team consisting of your peers, review the technical specification of the software product and checks whether it is suitable for the project.
- They try to find any discrepancies in the specifications and standards followed.
- This review concentrates mainly on the technical documents related to the software such as test strategy, test plan and requirement specification documents.

1.4. Static code review

- This is a systematic review of the software source code, without executing the code.
- It checks the syntax of the code, coding standards, code optimization, etc.
- This review can be done at any point during development.
- It is basically applied in two ways:
    1. Rule-based Static Analysis
    2. Flow-based Static Analysis

1.5. Management Review

- The management review is a cross-functional review undertaken by an organization's top management.
- It includes analyses of
  - ✓ customer satisfaction,
  - ✓ determination of the cost of poor quality,
  - ✓ performance trends and
  - ✓ achievement of objectives defined in the business plan.

1.6. Inspection

- An inspection is the most formal type of review and is conducted by trained moderators.
- During an inspection, the documents and products are examined through a process of reviews and defects are identified following strict process to find the defects.
- The defects identified are documented in a logging list or issue log and a formal follow-up is carried out.
- The main purpose is to find defects, and the meeting is led by trained moderator.
- Reviewers have checklist to review the work products.
- They record the defect and inform the participants to rectify those errors.

1.7. Audit

- An audit is an independent examination of all the documents related to the software, carried out by an external agent.
- Audits provide increased assurance to the various stakeholders of the project that the document is up to standards and devoid of any defects.

2. Dynamic Test Techniques

- Dynamic testing involves execution of the test object (application) on a computer.
- This testing technique is also called execution technique or validation testing
- Dynamic testing is performed in runtime environment. When the code being executed is input with a value, the result or the output of the code is checked and compared with the expected output. With this we can observe the functional behavior of the software, monitor the system memory, CPU response time, performance of the system.

Cont...

- Dynamic testing is carried out during the validation process.
- Dynamic test design techniques can be further classified into:
  - **Specification-based (black-box, also known as behavioral techniques)**
  - **Structure-based (white-box or structural techniques)**
  - **Experience-based**

2.1. Specification-Based / Black Box Testing Technique

- Specification-based testing technique, also known as the black box testing technique uses the software's external descriptions such as the
  - technical specifications,
  - design,
  - customer requirements, etc.
- This implies that a tester who does not have any knowledge about the code or internal structure can also perform the test.

Cont...

- **Some of the well-known methods in this technique are:**
  - ❖Equivalence Partitioning (EP)
  - ❖Boundary Value Analysis (BVA)
  - ❖Decision Table Testing
  - ❖State Transition Testing

2.1.1. Equivalence Partitioning (EP)

- Equivalent partitioning (EP) or Equivalent Class Partitioning (ECP) is a black box testing technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system and UAT.
- The idea behind this technique is to divide/partition a set of test conditions into groups or sets that can be considered the same or equivalent. This is used for reducing the total number of test cases to a finite set of testable test cases, still covering maximum requirements.
- One test value is picked from each class while testing. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software.

Cont...

- If one condition in a partition works, we assume all of the conditions in that partition will work and if one condition fails, it is assumed that all others in the partition will fail and there is no point in testing others.
- In this technique, you divide the set of test conditions into a partition that can be considered the same.
  - It divides the input data of software into different equivalence data classes.
  - You can apply this technique, where there is a range in input field.

Cont...

- Let's consider the behavior of tickets in the flight reservation application, while booking a new flight.
- Ticket values 10 to 50 are considered valid & ticket is booked. While other values are considered invalid for reservation

Invalid	Valid	Invalid
<10	10-50	>50

Cont...

- The divided sets are called Equivalence Partitions or Equivalence Classes.
- Then we pick only one value from each partition for testing. ◦ The hypothesis behind this technique is that if one condition/value in a partition passes all others will also pass.
- Likewise, if one condition in a partition fails, all other conditions in that partition will fail.

2.1.2. Boundary Value Analysis (BVA)

- BVA is based on testing the boundary values of valid and invalid partitions.
- The behavior at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.
- Every partition has its maximum and minimum values and these values are the boundary values of a partition.
- A boundary value for a valid partition is a valid boundary value. Similarly a boundary value for an invalid partition is an invalid boundary value.

Cont...

- Tests can be designed to cover both valid and invalid boundary values.
- When designing test cases, a test for each boundary value is chosen.
- This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases.
- Boundary value analysis is most common when checking a range of numbers.
- We should design test cases which exercise the program functionality at the boundaries, with values just inside and just outside the boundaries.

Cont...

- Here, the assumption is that if the program works correctly for these extreme cases, then it will work correctly for all values in between the valid partition.
- Testing has shown that defects that arise when checking a range of values the most defects are near or at the boundaries.

Cont...

- For each boundary, we test +/-1 in the least significant digit of either side of the boundary.
- Boundary value analysis can be applied at all test levels.

Example: Assume, we have to test a field which accepts Age between 18 – 56,

- Min-1
- Min
- Min+1
- Nominal
- Max-1
- Max
- Max+1

AGE [Enter Age] *Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	**Valid (min, +min, -max, max)**	**Invalid (max +1)**
17	18, 19, 55, 56	57

2.1.3. Decision Table Testing

- A decision table is a tabular representation of inputs versus rules/cases/test conditions.
- Decision table testing is a testing technique used to test a system for different input combinations.
- This is a systematic approach where the different input combinations and their corresponding system outputs are captured in a tabular form.
- That is why it is also called as a Cause-Effect table where causes and effects are captured for better test coverage.

Cont...

- It also helps in better test coverage for complex business logic.
- The significance of this technique becomes immediately clear as the number of inputs increases.
- Number of possible combinations is given by 2^n, where n is the number of inputs.
- For n = 10, which is very common in the web based testing, having big input forms, the number of combinations will be 1024. Obviously, you cannot test all but you will choose a rich sub-set of the possible combinations using decision based testing technique

Cont...

- Let's create a decision table for a login screen.
- The condition is simple if the user provides correct username and password, the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Legend:

- T - Correct username/password
- F - Wrong username/password
- E - Error message is displayed
- H - Home screen is displayed

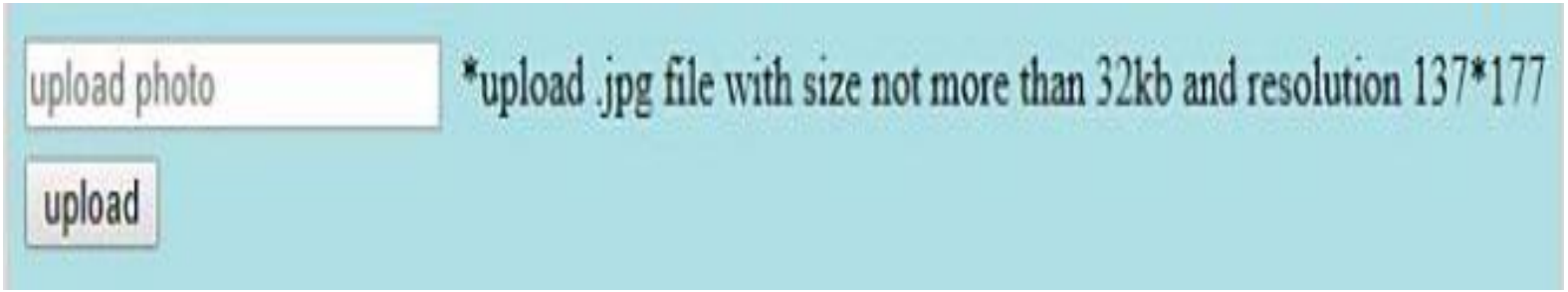
Conditions	Case 1	Case 2	Case 3	Case 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Cont...

- Interpretation:
- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

Cont...

- Decision Table for “Upload Screen”
- Now consider a dialogue box which asks the user to upload photo with certain conditions like
  1. You can upload only '.jpg' image format
  2. File size less than 32kb
  3. Resolution 137x177.
- If any of the conditions fails, the system will throw a corresponding error message stating the issue; and if all conditions are met, photo will be updated successfully



Cont...

- Let's create the decision table for this case.

Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	More than 32kb	More than 32kb	Less than 32kb	Less than 32kb	More than 32kb	More than 32kb
resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch	Error message for format, size, and resolution mismatch

Cont...

- For this condition, we can create 8 different test cases and ensure complete coverage based on the above table.

1. Upload a photo with format '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Photo should upload successfully

2. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message resolution mismatch should be displayed

3. Upload a photo with format '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message size mismatch should be displayed

4. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message size and resolution mismatch should be displayed

Cont...

5. Upload a photo with format other than '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format mismatch should be displayed

6. Upload a photo with format other than '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message format and resolution mismatch should be displayed

7. Upload a photo with format other than '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format and size mismatch should be displayed

8. Upload a photo with format other than '.jpg', size more than 32kb and resolution not 137*177 and click on upload. Expected result is Error message for format, size and resolution mismatch should be displayed

2.1.4. State Transition Testing

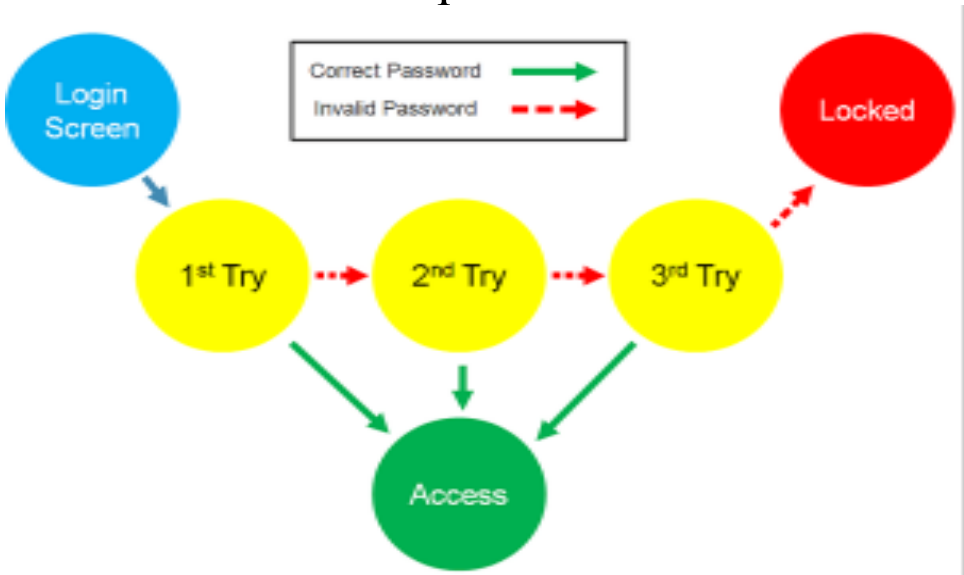
- In state transition testing technique, changes in input conditions cause state changes in the application under test.
- In this technique, the tester analyzes the behavior of an application under test for different input conditions in a sequence.
- The tester provides both valid and invalid input test values and record the system behavior.
- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
- This technique is helpful where you need to test different system transitions.

Cont...

- When to use state transition?
  - ✓ When the tester is trying to test sequence of events that occur in the application under test i.e., this will allow the tester to test the application behavior for a sequence of input values.
  - ✓ When the system under test has a dependency on the events/values in the past.
  - ✓ This can be used when a tester is testing the application for a finite set of input values.
- When to not rely on state transition?
  - ✓ When the testing is not done for sequential input combinations.
  - ✓ If the testing is to be done for different functionalities like exploratory testing

Cont...

- Example:- Let's consider an ATM system function where if the user enters the invalid password three times, the account will be locked.
- In this system,
  - If the user enters a valid password in any of the first three attempts the user will be logged in successfully.
  - If the user enters the invalid password in the first or second, the user will be asked to re-enter the password.
  - And finally, if the user enters incorrect password 3rd time, the account will be blocked.



2.2. Structure-based / White Box Test Technique

- The exact opposite of the black box test design technique, white box test design technique necessitates knowledge of the internal structure of a program.
- Some of the methods implemented in this type of technique are:
  - Statement coverage
  - Decision coverage
  - Condition coverage
  - Multiple condition coverage
  - Control flow based criteria
  - Data Flow Testing

2.2.1. Statement coverage

- Statement coverage technique involves execution of all statements of the source code at least once.
- It is used to calculate the total number of executed statements in the source code out of the total statements present in the source code.
- Based on the input given to the program, some code statements are executed and some may not be executed.
- The goal of statement coverage technique is to cover all the possible executing statements and path lines in the code.
- $$Statement\ coverage = \frac{Number\ of\ executed\ satements}{Total\ number\ of\ statements} * 100$$

- **Example**

```
Test (int a, int b) {
int sum = a+b;
if (sum>0)
Print ("This is a positive result")
else
Print ("This is negative result")
}
```

Scenario 1: If a = 5 and b = 4

Total number of statements = 7
Number of executed statements = 5
Statement coverage = 5/7*100
= 500/7
= 71%

Scenario 2: If a = -2 and b = -4

Statement coverage = 6/7*100
= 6/7 *100
= 85%

But, we can see all the statements are covered in both scenarios and we can consider that the overall statement coverage is 100%.

2.2.2. Decision coverage technique

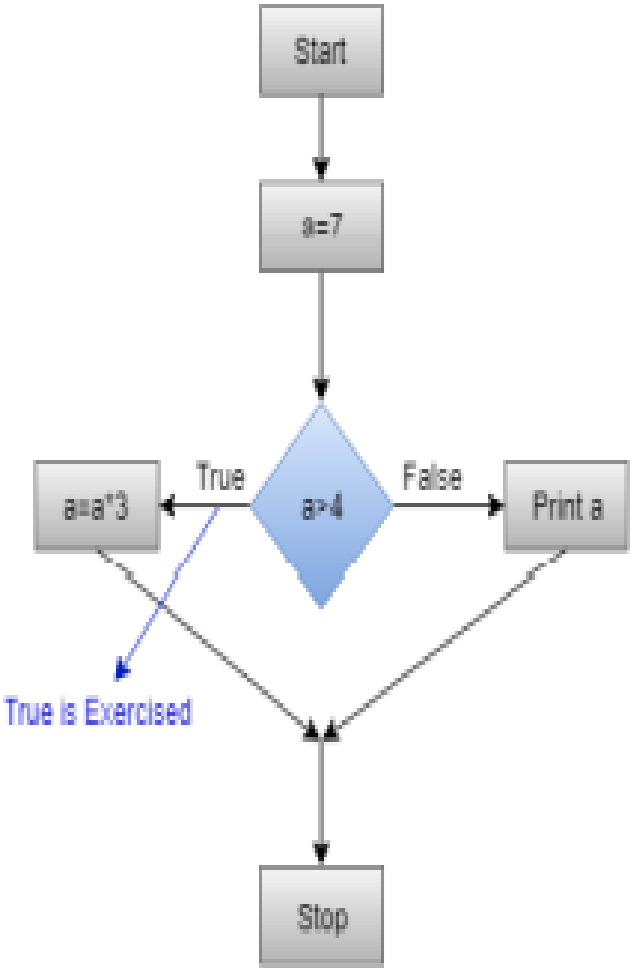
- This technique reports true and false outcomes of Boolean expressions.
- Whenever there is a possibility of two or more outcomes from the statements like **do while statement, if statement and case statement** (Control flow statements), it is considered as decision point because there are two outcomes either true or false.
- Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.
- $$\textbf{\textit{Decision coverage}} = \frac{\textbf{\textit{Number of Decision Outcomes Exercised}}}{\textbf{\textit{Total Number of Decision Outcomes}}} * 100$$

- Example 1

```
Test (int a)
{if (a>4)
a=a*3
Print (a)
}
```

Scenario 1: **If a=7**

The control flow graph when the value of a is 7



Only "True" is exercised

Decision Coverage = ½*100

Decision Coverage is 50%

2.2.3. Condition Coverage

- Condition coverage or predicate coverage ensures whether all the Boolean expressions have been evaluated to both TRUE and FALSE.
- In condition coverage the possible outcomes of ("true" or "false") for each condition are tested at least once. This means that each individual condition is one time true and false.
- Also the combinations of conditions are not relevant.
- Since there are only two possible outcomes of a condition (true or false), condition coverage results in 2 test situations per decision point.
- Example:

```
if (A OR B)
   {
      Print Z
    }
```

Cont...

Truth table

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Condition coverage
The possible outcomes ("true" or "false") of each condition are tested at least once.

IF number of books > 8 **OR** sum ≥ 100 THEN extra discount

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	0	1 (extra discount)
TS2	0	1	1 (extra discount)

For condition coverage, only two test situations are needed per decision point

The condition "Number of books > 8" is one time True and one time False

For condition "Sum ≥ 100" is also one time True and one time False

Notice that the outcomes of the decision do not need to vary

2.2.4. Multiple Condition Coverage

- Every combination of 'true' or 'false' for the conditions related to a decision has to be tested in this technique.

Example

- Assume we want to test the following code extract:

```
if ( (A || B) && C )
    {
      /* instructions */
    }
else
    {
      /* instructions */
    }
```

- where A, B and C represent atomic Boolean expressions (i.e. not divisible in other Boolean sub-expressions).

Cont...

- In order to ensure Multi-condition coverage criteria for this example, each combination of values for A, B and C should be tested at least one time.
- So, in our example, we need $2^3=8$ tests leading to the following evaluations of A, B and C to valid Multi-condition coverage:

A = true / B = true / C = true

A = true / B = true / C = false

A = true / B = false / C = true

A = true / B = false / C = false

A = false / B = true / C = true

A = false / B = true / C = false

A = false / B = false / C = true

A = false / B = false / C = false

Some languages evaluate the second operand of a logical AND only if the first operand is true (or evaluate the second operand of OR only if the first operand evaluates to false).

Cont...

- However, some programming languages such as C, C++, C#, Java Script etc, ensure multi-condition coverage using the results of tests in the following way:

  A = true / B = not evaluation / C = true

  A = true / B = not evaluation / C = false

  A = false / B = true / C = true

  A = false / B = true / C = false

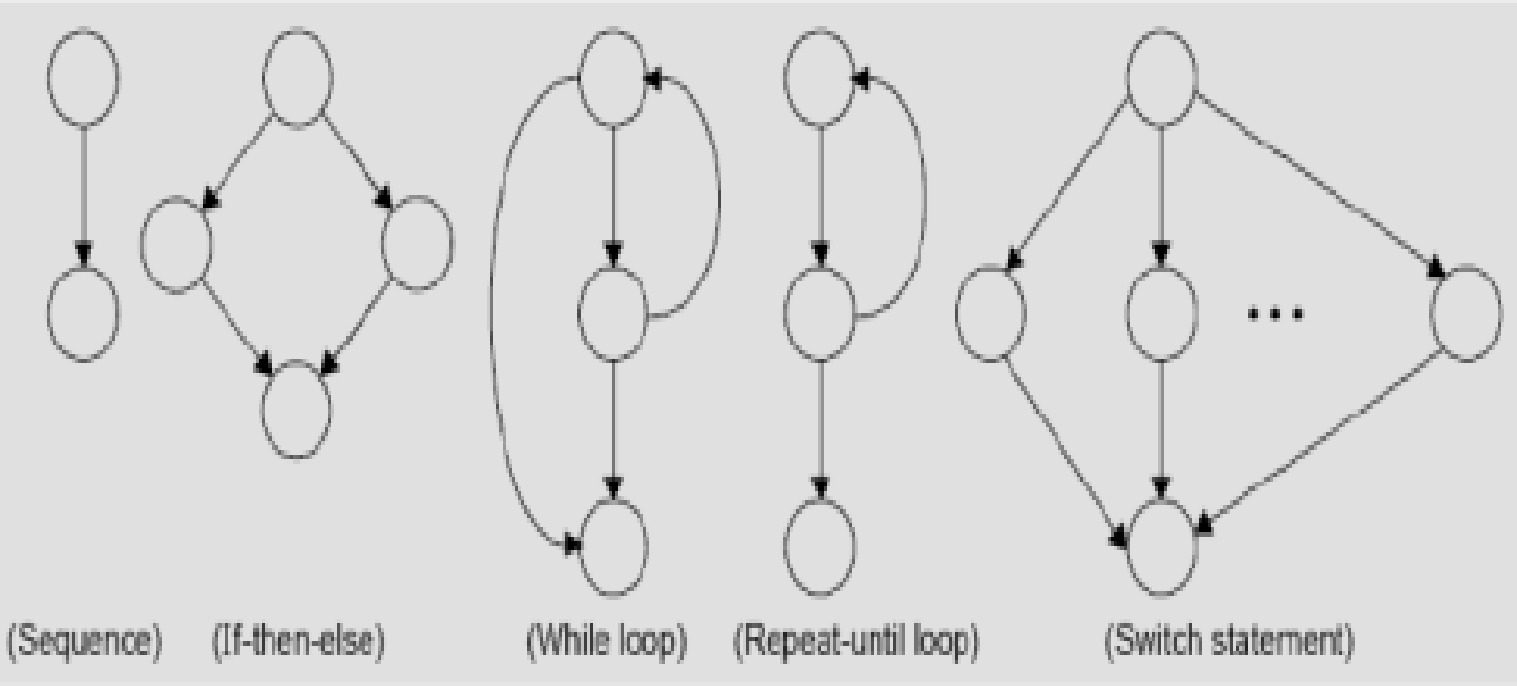
  A = false / B = false / C = not evaluation

- E.g., for the 1st case, if A is "true", system will not calculate the value of B since in every way (A or B) will be evaluated to "true". In fact, it means that the 1st and 2nd cases of the 1st list will be evaluated in the same way by the system and that will follow this logic while presenting the results.

2.2.5 Control Flow Based Criteria

- Control flow testing is **a form of white-box testing where the implementation of the code is known to the tester**.
- The development team often performs control flow testing.
- This process determines and observes the execution paths of a program in a structured way.
- The technique used to develop the test cases.
- The **control flow** of a program can be analyzed using a graphical representation known as flow graph.
- The **flow graph** is a directed graph in which nodes are either entire statements or fragments of a statement, and edges represents flow of control.

- **The basic construct of the flow graph**



Cont...

- **On a flow graph:**
  - Arrows called edges represent flow of control
  - Circles called nodes represent one or more actions.
  - Areas bounded by edges and nodes called regions.
  - A predicate node is a node containing a condition

Decision – to – Decision (DD) path Graph

- When we have a flow graph, we can easily draw another graph that is known as **decision-to-decision** or (DD) path graph, wherein we lay our main focus on the decision nodes only.
- The **nodes of the flow graph** are combined into a **single node** if they are in sequence.
- A path through a program is a node and edge sequence from the starting **node to a terminal** node of the control flow graph of a program.
- If a path has one new node compared to all other linearly independent paths, then the path is **linearly independent**.

Example 1

- Consider a program given for the identification of the greatest number. Draw the flow graph & DD Path graph. Also find the independent paths from the DD Path graph.

```
int main() {
1.    double n1, n2, n3;
2.   cin >> n1 >> n2 >> n3;
3.  if(n1 >= n2){
4.     if(n2>=n3){
5.      Max = n1;}
6.      else{ Max = n3;}}
7.    else if(n2 >=n3){
8            Max = n2;}
9.            else { Max = n3;}
10.        cout << "Largest number: " << max;}}
```

Cont...

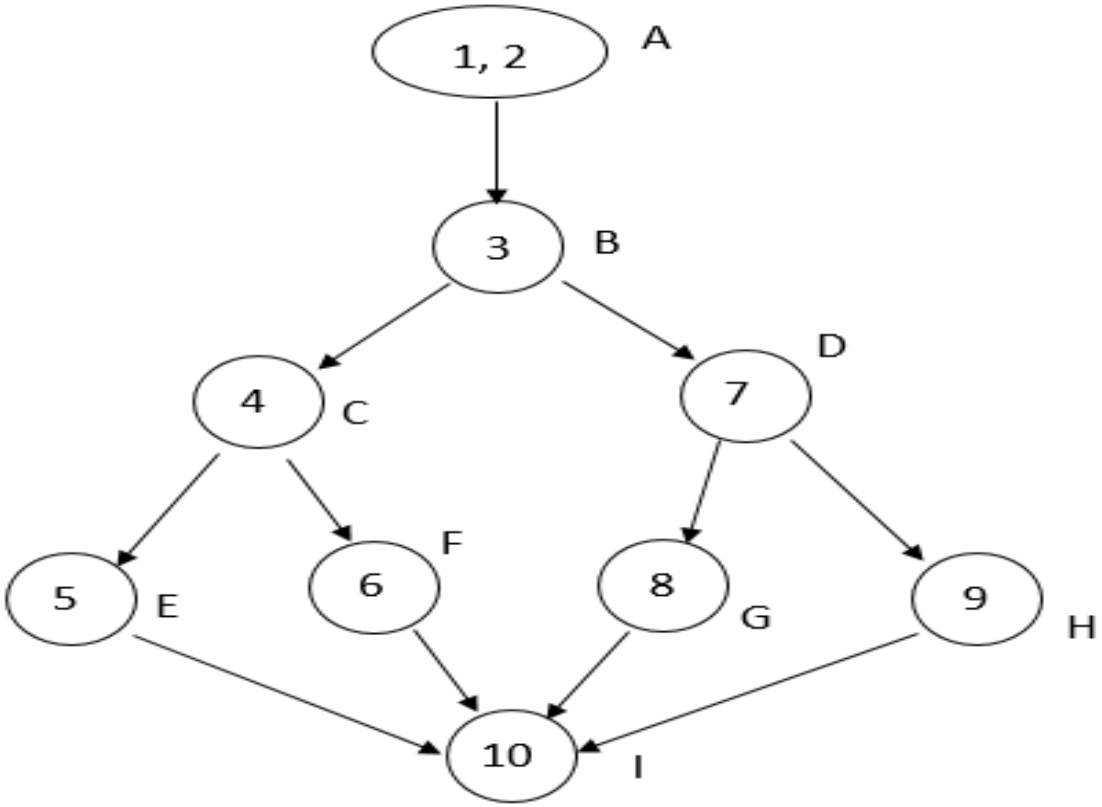
- The flow Graph of the above code and the independent path are:

i) ABDHI

ii) ABDGI

iii) ABCFI

iv) ABCEI



Example 2

- Consider a program given for the classification of a triangle. Its input is a triple of positive integers (say a,b,c) from the interval [1,100]. The output may be [Scalene, Isosceles, Equilateral, Not a triangle]. Draw the flow graph & DD Path graph. Also find the independent paths from the DD Path graph.

Cont...

```
#include <stdio.h>
#include <conio.h>
1    int main()
2    {
3      int a,b,c,validInput=0;
4      printf("Enter the side 'a' value: ");
5      scanf("%d",&a);
6      printf("Enter the side 'b' value: ")
7      scanf("%d",&b);
8      printf("Enter the side 'c' value:");
9      scanf("%d",&c);
10     if ((a > 0) && (a <= 100) && (b > 0) && (b <= 100) && (c > 0)
           && (c <= 100)) {
11       if ( (a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
12         validInput = 1;
13       }
14     }
15     else {
16       validInput = -1;
17     }
18     If (validInput==1) {
19       If ((a==b) && (b==c)) {
20         printf("The trinagle is equilateral");
21       }
22       else if ( (a == b) || (b == c) || (c == a) ) {
```

(Contd.)...

```
23          printf("The triangle is isosceles");
24        }
25        else {
26          printf("The trinagle is scalene");
27        }
28      }
29      else if (validInput == 0) {
30        printf("The values do not constitute a Triangle");
31      }
32      else {
33        printf("The inputs belong to invalid range");
34      }
35      getche();
36      return 1;
37  }
```

Solution :

Flow graph of triangle problem is:

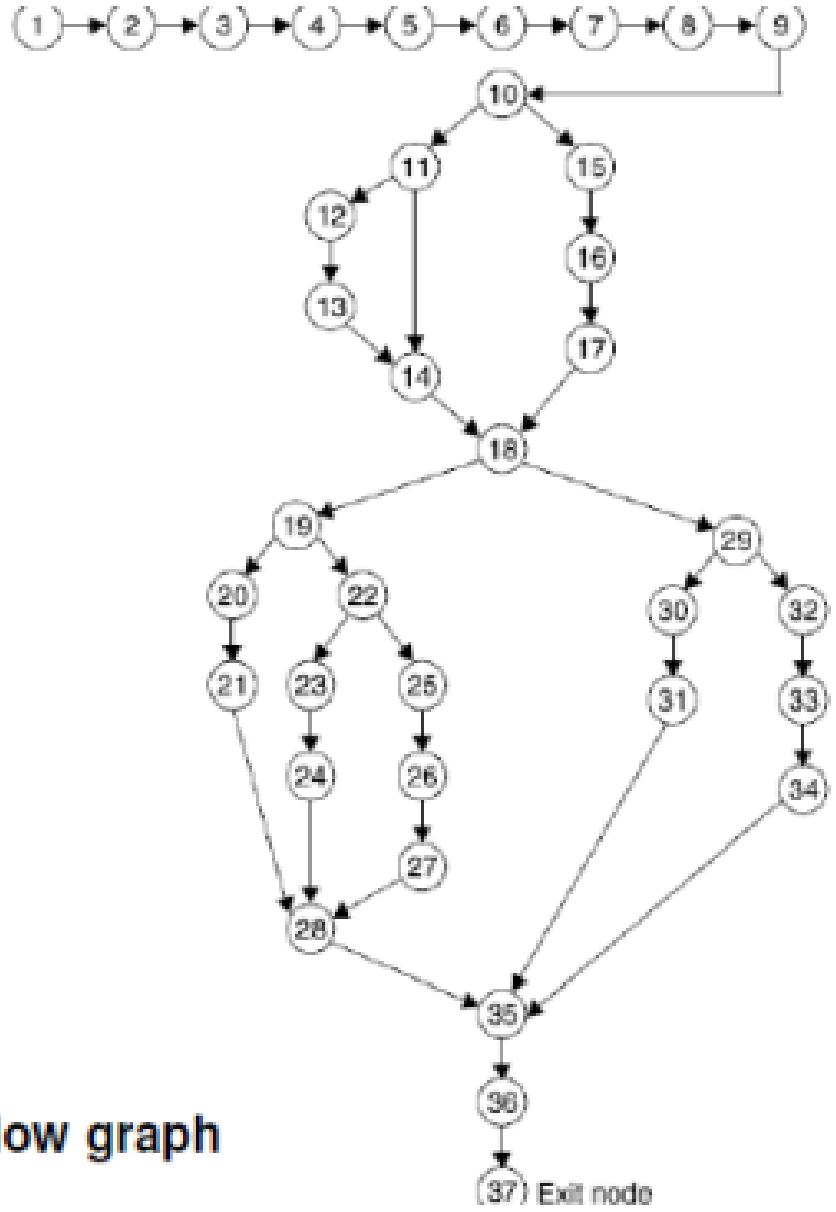


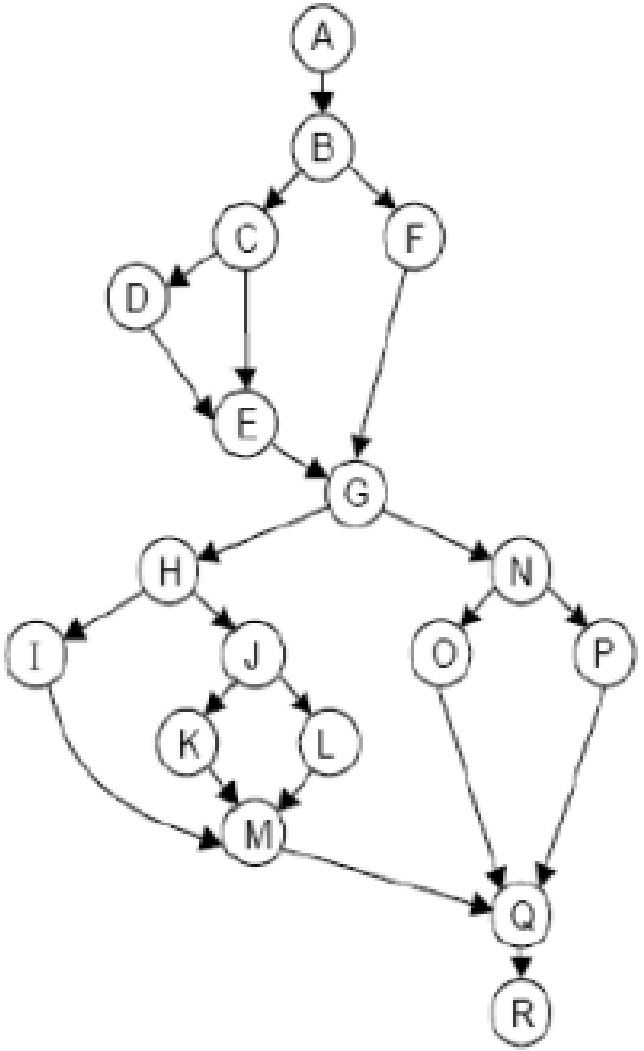
Fig.8. 20 (a): Program flow graph

The mapping table for DD path graph is:

Flow graph nodes	DD Path graph corresponding node	Remarks
1 TO 9	A	Sequential nodes
10	B	Decision node
11	C	Decision node
12, 13	D	Sequential nodes
14	E	Two edges are joined here
15, 16, 17	F	Sequential nodes
18	G	Decision nodes plus joining of two edges
19	H	Decision node
20, 21	I	Sequential nodes
22	J	Decision node
23, 24	K	Sequential nodes
25, 26, 27	L	Sequential nodes

Flow graph nodes	DD Path graph corresponding node	Remarks
28	M	Three edges are combined here
29	N	Decision node
30, 31	O	Sequential nodes
32, 33, 34	P	Sequential nodes
35	Q	Three edges are combined here
36, 37	R	Sequential nodes with exit node

DD Path graph is given in Fig. 20 (b)



Independent paths are:

(i) ABFGNPQR
(ii) ABFGNOQR
(iii) ABCEGNPQR
(iv) ABCDEGNOQR
(v) ABFGHIMQR
(vi) ABFGHJKMQR
(vii)ABFGHJMQR

Fig. 20 (b): DD Path graph

Cyclomatic Complexity

- Cyclomatic complexity is a source code complexity measurement.
- It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Cyclomatic complexity (V(G)) = **E - N + 2*P**

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

Note: for a single program (or subroutine or method), P is always equal to 1. so simple formula for a single subroutine is M = E-N+2 or

- Described (informally) as the number of decision points + 1

Cont...

- The cyclomatic complexity of the above example is

For Example 1

$$V(G) = E-N+2$$

$$= 11-9+2 = 4$$

For Example 2

$$V(G) = 23-18+2 = 7$$

Data Flow Testing

- **Data Flow Testing** is a type of structural testing.
- It is a method that is used to find the test paths of a program according to the locations of definitions and uses of **variables** in the program.
- It mainly **focuses** on the two **points:**
  - In which statement the variables are defined
  - In which statement the variables are used
- It uses the **control flow graph** to detect illogical things that can interrupt the flow of data.
- Anomalies in the flow of data are detected at the time of associations between values and variables due
  - If the variables are used without initialization.
  - If the initialized variables are not used at least once

Example

```
1. read a, b, c, d;
2. if(a>b)
3.     X = a+1;
4.     print x;
   else
5.     x = b-1
6.     Y = b+1
7.     print Z;
8.     PrintY:
```

Variable	Define	Use
a	1	2,3
b	1	2,5,6
c	1	NA
d	1	NA
X	3,5	4
Y	6	8
Z	NA	7

Advantages of Data Flow Testing

- To identify a variable that is declared but never used with in the program.
- To identify a variable that is used but never declared.
- To identify a variable that is defined multiple times before it is used.
- Deallocating a variable before it is used.

2.3. Experience-Based Test Technique

- As the name suggests, the experience-based technique neither involves internal, nor the external structure, but is based on experience.
- This type of testing is most effective in the hands of experienced testers who will bring their knowledge to bear in the most productive ways possible. Why it is so?
  - Experienced testers know what needs to be documented to assess coverage and ensure some level of repeatability.
  - Experienced testers also know what to target to find the biggest bugs quickly and return a high yield during their testing time.
- These are not techniques for novices-but they are definitely appropriate and productive techniques for the advanced tester.

Cont...

- **Some of the methods followed are:**

1) Exploratory Testing Technique

2) Fault Attack Testing Technique

3) Error guessing Testing Technique

4) Checklist based Testing Technique

2.3.1. Exploratory testing technique

- Exploratory testing, is all about discovery, investigation and learning.
- It emphasizes on personal freedom and responsibility of the individual tester.
- Test cases are not created in advance but testers check system on the fly.
- They may take notes on ideas about what to test before test execution.
- The focus of exploratory testing is more on testing as a "thinking" activity.
- Normally, you design test cases first and later proceed with test execution. On the contrary, exploratory testing is a simultaneous process of test design and test execution.

2.3.2. Fault attack testing technique

- Software attacks, sometimes called fault attacks, are focused on trying to induce a specific type of failure.
- When performing attack testing, you should consider all areas of the software and its interaction with its environment as opportunities for failures.
- Attacks target the user interface, the operating system, interfacing systems, database interfaces and any file system interaction.
- Anytime data is being exchanged, it is potentially vulnerable to a failure and consequently is an excellent target for an attack.
- Coverage for attack testing is usually measured by determining if all the potentially vulnerable interfaces have been tested

2.3.3. Error guessing testing technique

- It doesn't sound very official, but it works.
- Error guessing as a testing technique is employed by the tester to determine the potential errors that might have been introduced during the software development and to devise methods to detect those errors as they manifest into defects and failures.
- Error guessing is commonly used in risk analysis to "guess" where errors are likely to occur in the error-prone areas.
- Error guessing coverage is usually determined based on the types of defects that are being sought.
- If there is a defect (or bug) taxonomy available, that can be used as the guideline. If taxonomy is not employed, the experience of the tester and the time available for testing usually determine the level of coverage.

2.3.4. Checklist-based testing technique

- Checklist-based testing is used by experienced testers who are using checklists to guide their testing.
- The checklist is basically a high-level list, or a reminder list, of areas to be tested.
- This may include items to be checked, lists of rules, or particular criteria or data conditions to be verified.
- Checklists are usually developed over time and draw on the experience of the tester as well as on standards, previous trouble-areas, and known usage scenarios.
- Coverage is determined by the completion of the checklist

THE END

THANK YOU!!!