Lecture 20: Testing (3)

Nonexecution-based Testing (Static Testing)

- Walkthroughs
- Inspections
- Walkthroughs are shorted and more informal than inspection.
- Goal of both is to record faults, not to correct them.

Walkthrough

- A *walkthrough* is an informal way of presenting a technical document in a meeting.
  - Unlike other kinds of reviews, the author runs the walkthrough: calling the meeting, inviting the reviewers, soliciting comments and ensuring that everyone present understands the work product.
  - Walkthroughs are used when the author of a work product needs to take into account the perspective of someone who does not have the technical expertise to review the document.
  - After the meeting, the author should follow up with individual attendees who may have had additional information or insights. The document should then be corrected to reflect any issues that were raised.

Walkthrough (2)

- Guidelines for a Successful Walkthrough
  - Verify that everyone is present who needs to review the work. This can include users, stakeholders, engineering leads, managers and other interested people.
  - Verify that everyone present understands the purpose of the walkthrough meeting and how the material is going to be presented.
  - Describe each section of the material to be covered by the walkthrough.

Walkthrough (3)

- Guidelines for a Successful Walkthrough
  - Present the material in each section, ensure that everyone present understands the material.
  - Lead a discussion to identify any missing sections or material.
  - Document all issues that are raised by walkthrough attendees.

Inspections

- Inspections usually based on a comprehensive infrastructure:
  - Development of inspection checklists for each type of design document as well as coding languages, which are periodically updated.
  - Development of typical defect type frequency tables, based on past findings to direct inspectors to potential ‘defect concentration areas.’
  - Training of competent professionals in inspection process issues – making it possible for them to serve as inspection leaders (moderators) or inspection team members

Inspections (2)

– Periodic analysis of the effectiveness of past inspections to improve the inspection methodology

– Introduction of scheduled inspections into project activity plan and allocation of the required resources, including resources for corrections

Inspection (3)

- Checklist
  - Clarity
    - Is the code clear and easy to understand?
    - Did the programmer unnecessarily obfuscate any part of it?
    - Can the code be refactored to make it clearer?
  - Maintainability
    - Will other programmers be able to maintain this code?
    - Is it well commented and documented properly?
  - Accuracy
    - Does the code accomplish what it is meant to do?
    - If an algorithm is being implemented, is it implemented correctly?
  - Readability and Robustness
    - Is the code fault-tolerant? Is the code error-tolerant?
    - Will it handle abnormal conditions or malformed input?
    - Does it fail gracefully if it encounters an unexpended condition?

Inspections (4)

- Checklist
  - Security
    - Is the code vulnerable to unauthorized access, malicious use, or modification?
  - Scalability
    - Could the code be a bottleneck that prevents the system from growing to accommodate increase load, data, users, or input?
  - Reusability
    - Could this code be reused in other applications?
    - Can it be made more general?
  - Efficiency
    - Does the code make efficient use if memory, CPU cycles, bandwidth, or other system resources?
    - Can it be optimized?

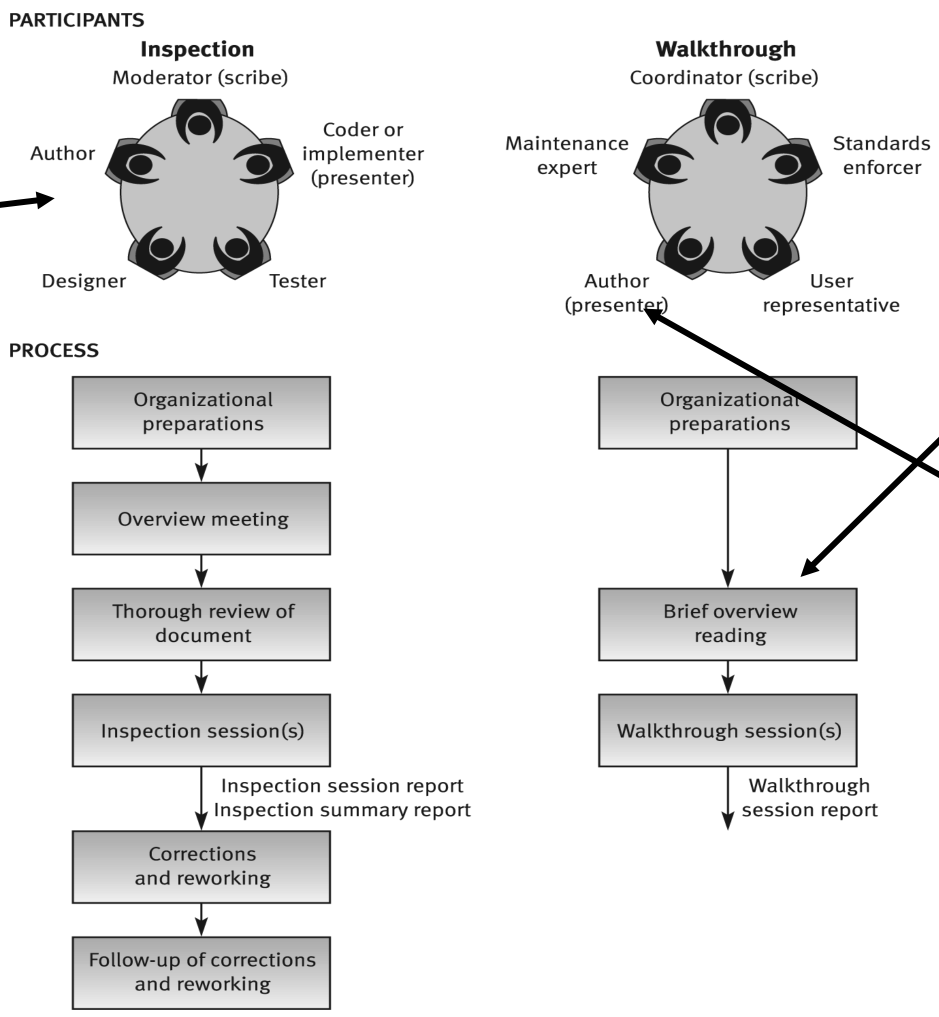
Inspections VS. Walk-Throughs Formality

- Walkthroughs and inspection differ in formality
- Inspections emphasize the objective of corrective action; more formal
- Walkthroughs limited to comments on document reviewed.

- Inspections also look to improve methods as well.
- Inspections are considered to contribute more to general level of software quality assurance.

Inspections VS. Walk-Throughs

Note: author is **not** the presenter


PARTICIPANTS
Inspection
Moderator (scribe)
Author
Coder or implementer (presenter)
Designer
Tester
Walkthrough
Coordinator (scribe)
Maintenance expert
Standards enforcer
Author (presenter)
User representative
PROCESS
Organizational preparations
Overview meeting
Thorough review of document
Inspection session(s)
Inspection session report
Inspection summary report
Corrections and reworking
Follow-up of corrections and reworking
Organizational preparations
Brief overview reading
Walkthrough session(s)
Walkthrough session report

Much less formal…

author is the presenter

Inspections VS. Walk-Throughs Participants

Inspections

- Leader
- Author
- Professionals:
  - Designer;
  - Coder/Implementer
  - Tester

Walkthrough

- Leader
- Author
- Professionals:
  - Standards enforcer
  - Maintenance expert
  - User representative

Inspections VS. Walk-Throughs Participants (2)

- Leader
  - Moderator in inspections; Coordinator in walkthroughs;
  - Must be well-versed in project development and current technologies;
  - Have good relationships with author and development team;
  - Come from outside the project team;
  - History of proven experience in coordination / leadership settings like this.
  - For inspections, training as moderator is required.

Inspections VS. Walk-Throughs Participants (3)

- Specialized Professionals for Inspections:
  - A designer – generally the system analyst responsible for analysis and design of software system reviewed;
  - A coder or implementer – one who is thoroughly acquainted with coding tasks, preferably the leader of the designate coding team.
    - Able to detect defects leading to coding errors and other software implementation issues.
  - A tester – experienced professional, preferably leader of assigned testing team who focuses on identification of design errors usually detected during the testing phase.

Inspections VS. Walk-Throughs Participants (4)

- Specialized Professionals for Walkthroughs:
  - A standards enforcer – team member specialized in development standards and procedures;
    - Locate deviations from these standards and procedures. (coding standards, data naming, program organizations)
    - These problems substantially affect the team's long-term effectiveness for both development and follow-on maintenance.
  - A maintenance expert – focus on maintainability / testability issues
    - To detect design defects that may hinder bug correction and impact performance of future changes.

Inspections VS. Walk-Throughs Participants (5)

  - A maintenance expert – Focuses also on documentation (completeness / correctness) vital for maintenance activity.

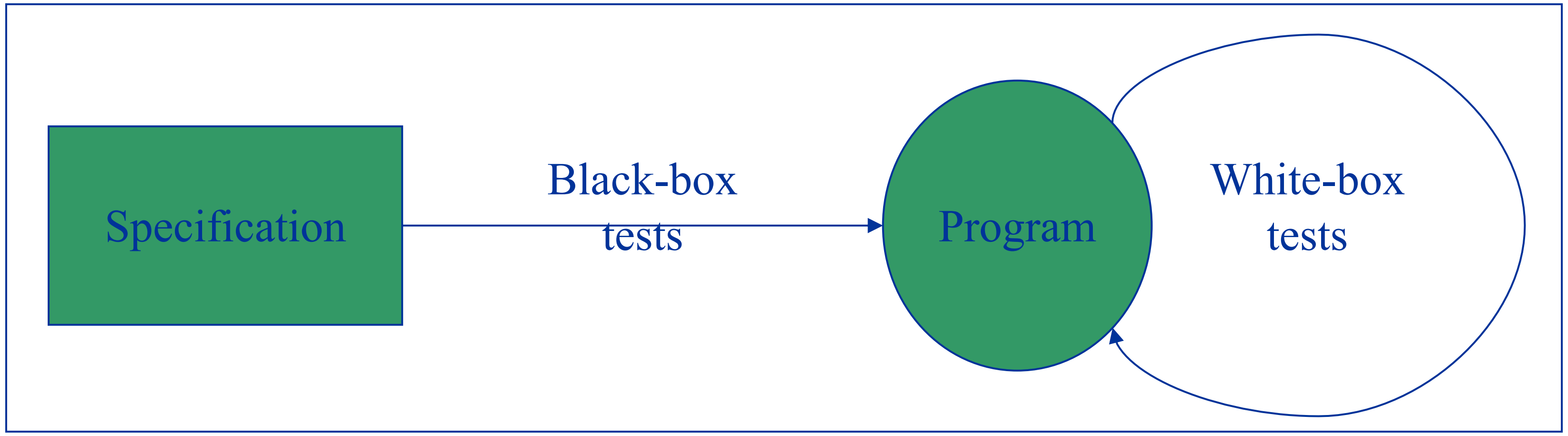
- A user representation – need an internal user (if customer is in the unit) or an external representative – review's validity due to his/her point of view as user-customer rather than the designer-supplier.

Execution-Base Testing (Dynamic Testing)

- Specification-based Testing
  - Black-box Testing
- Structure-based Testing
  - White-box Testing
- Experience-based Testing

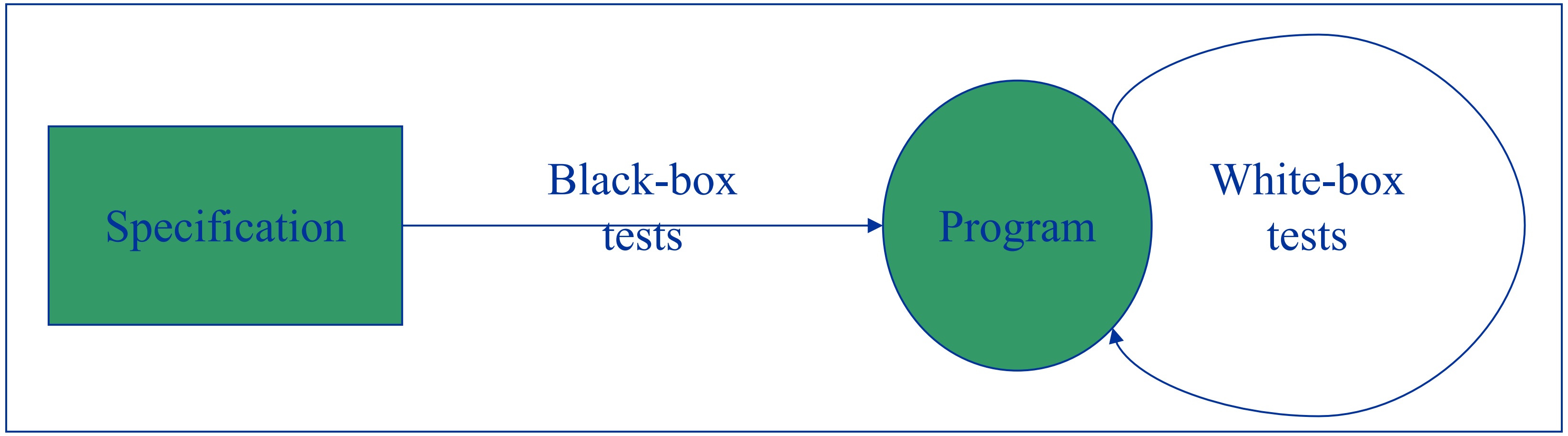
Black-box / White-box Testing

- Black-box tests are driven by the program’s specification.
- White-box tests are driven by the program’s implementation.



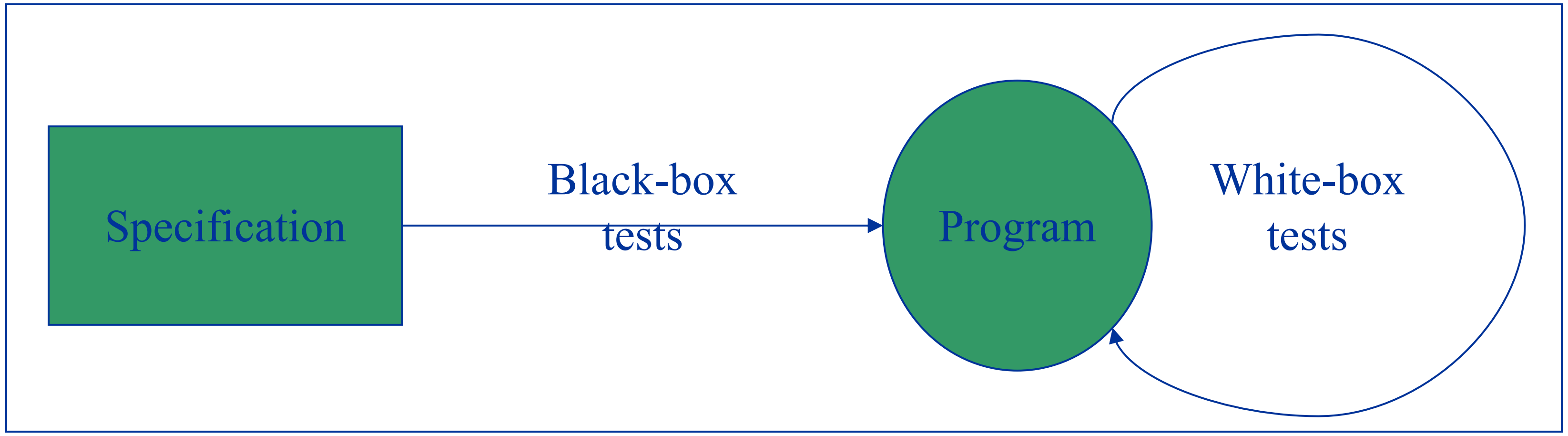
Black-box Testing

- Checks that the product conforms to specifications.
- Cannot determine how much code has been tested



White-box Testing

- Allows tester to be sure every statement has been tested.
- Difficult to discover missing functionality.



Execution-Base Testing (Dynamic Testing)

- Specification-based Testing (Lecture 18: Testing)
  - Black-box Testing
- Structure-based Testing
  - White-box Testing
- Experience-based Testing

Execution-Base Testing (Dynamic Testing)

- Specification-based Testing
  - Black-box Testing
- Structure-based Testing
  - White-box Testing
- Experience-based Testing

White-Box Testing/Structure-based Testing

- There exist several popular white-box testing methodologies:
  - Statement coverage
  - Branch coverage
  - Condition coverage
  - Path coverage
    - Control path
    - Data path

Statement Coverage

- Statement coverage methodology:
  - Design test cases so that
    - Every statement in a program is executed at least once.

Statement Coverage

- The principal idea:
  - Unless a statement is executed, we have no way of knowing if an error exists in that statement.

Example

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

a = 4, b = 5

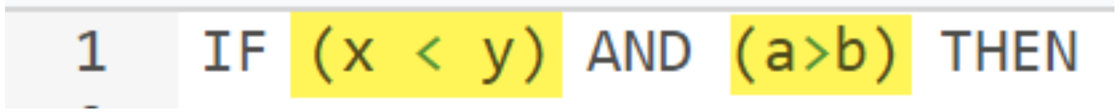
Branch Coverage

- *Branch coverage* is a requirement that, for each branch in the program (e.g., if statements, loops), each branch have been executed at least once during testing. (It is sometimes also described as saying that each branch condition must have been true at least once and false at least once during testing.)
- Test cases are designed such that:
  - different branch conditions
    - given true and false values in turn.

Condition Coverage

- Test cases are designed such that:
  - Each component of a composite conditional expression
    - Given both true and false values.

Example

- Consider the conditional expression
  - 

```
IF (x < y) AND (a>b) THEN
```

- Branch coverage
  - (x<y) and (a>b) == true
  - (x<y) and (a>b) == false
- Condition coverage
  - Each of x<y, and a>b is evaluated to true and false

Path Coverage

- Design test cases such that:
  - all linearly independent paths in the program are executed at least once.
  - Combination of branches

Linearly Independent Paths

- Defined in terms of
  - control flow graph (CFG) of a program.

Control Flow Graph (CFG)

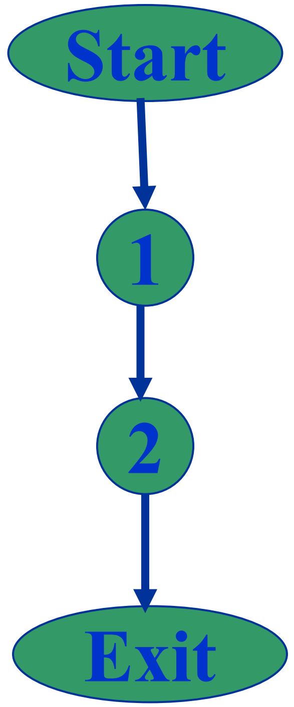
- A control flow graph (CFG) describes:
  - the sequence in which different instructions of a program get executed.
  - the way control flows through the program.

How to Draw Control Flow Graph?

- Control flow graph is process oriented.
- Control flow graph shows all the paths that can be traversed during a program execution.
- Control flow graph is a directed graph.
- Edges in CFG portray control flow paths and the nodes in CFG portray basic statements/blocks.

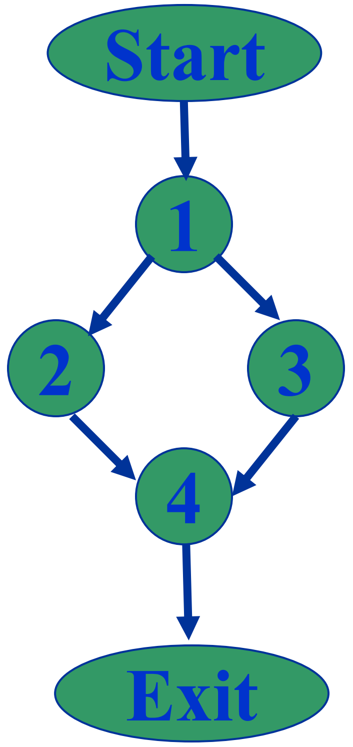
How to draw Control flow graph?

- Sequence:
  - 1 a=5;
  - 2 b=a*b-1;


$$\begin{tikzcd}
\text{Start} \arrow[d] \\
1 \arrow[d] \\
2 \arrow[d] \\
\text{Exit}
\end{tikzcd}$$

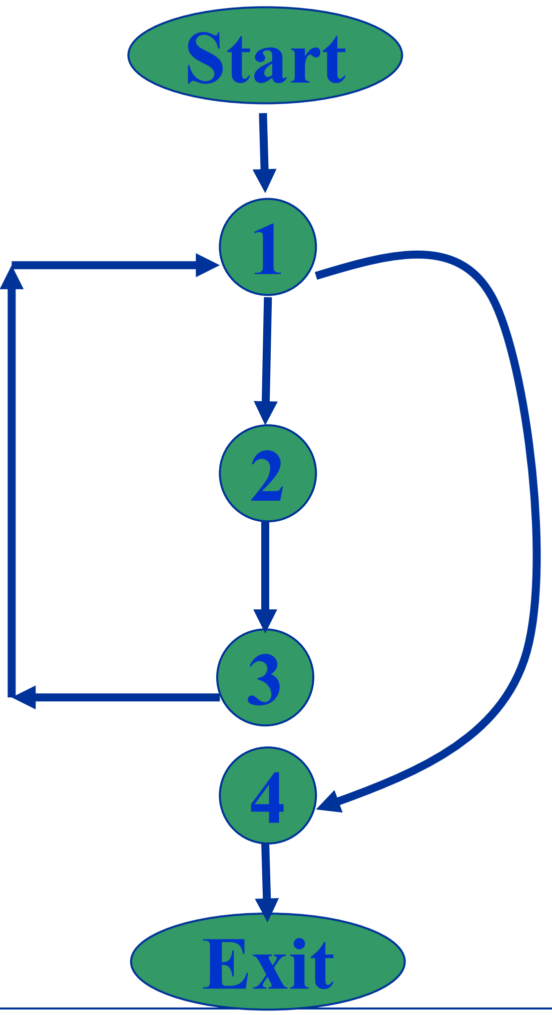
How to draw Control flow graph?

- Selection:
  - 1 if(a>b) then
  - 2 c=3;
  - 3 else c=5;
  - 4 c=c*c;



How to draw Control flow graph?

- Iteration:
  - 1 while(a>b){
  - 2 b=b*a;
  - 3 b=b-1;}
  - 4 c=b+d;



Path

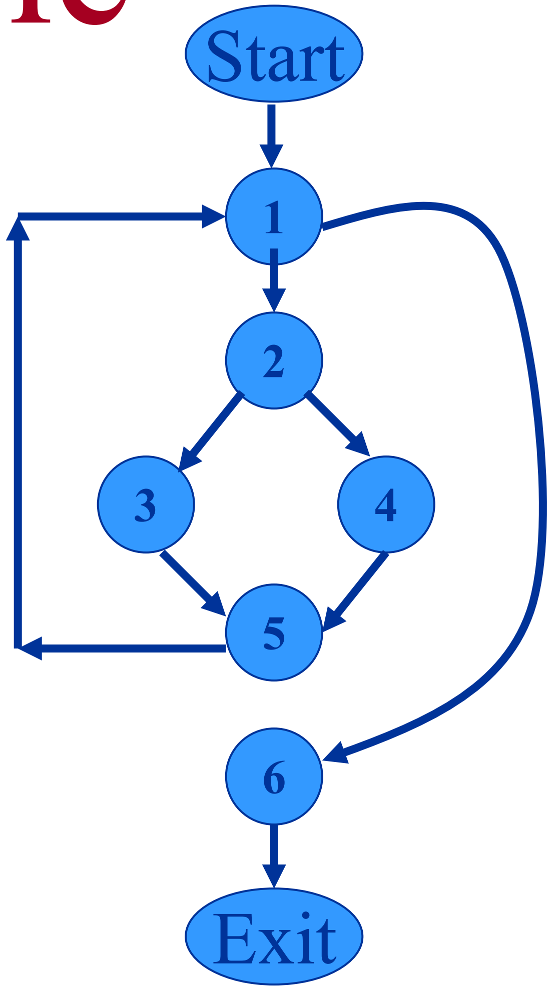
- A path through a program:
  - a node and edge sequence from the starting node to a terminal node of the control flow graph

Derivation of Test Cases

- Draw control flow graph.
- Determine V(G).
- Determine the set of linearly independent paths.
- Prepare test cases:
  - to force execution along each path.

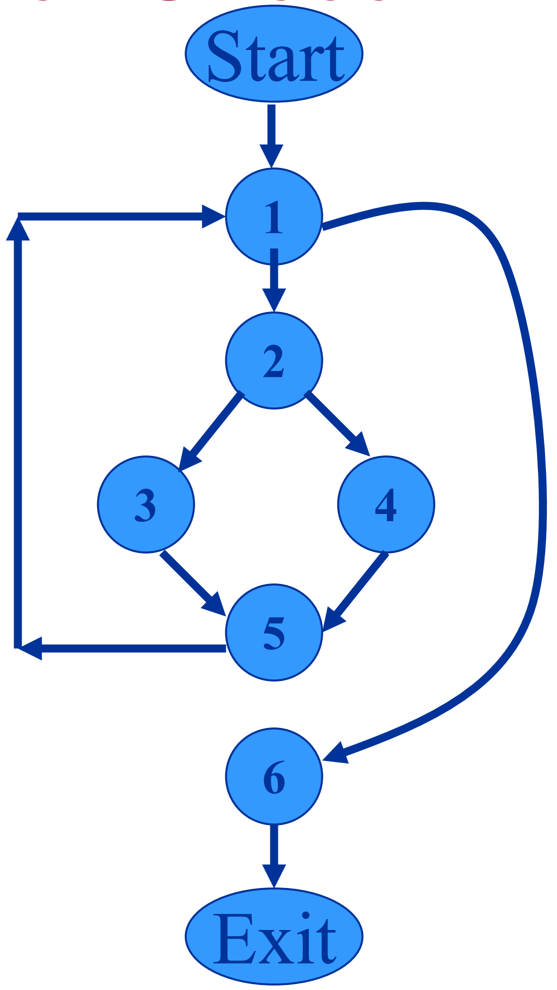
Example

- int f1(int x,int y){
- 1 while (x != y){
- 2   if (x>y) then
- 3   x=x-y;
- 4   else y=y-x;
- 5 }
- 6 return x;   }



Derivation of Test Cases

- Number of independent paths: 3
  - 1,6 test case (x=1, y=1)
  - 1,2,3,5,1,6 test case(x=1, y=2)
  - 1,2,4,5,1,6 test case(x=2, y=1)


Start
1
2
3
4
5
6
Exit