Software Testing

Definitions of “TESTING” (cont’d)

- Myers: The process of executing a program with the *intent* of finding errors.
- IEEE: The process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

Definitions of “TESTING” (cont’d)

- Testing undertaken to *demonstrate that a system performs correctly* is sometimes referred to as **validation testing**.
- Testing undertaken to *expose defects* is sometimes referred to as **defect testing**.

Fisherman’s Dilemma

- You have 3 days for fishing and 2 lakes to choose from. Day 1 at lake X nets 8 fish. Day 2 at lake Y nets 32 fish. Which lake do you return to for day 3?
- Does your answer depend on any assumptions?

Di Lemma

- In general, the probability of the existence of more errors in a section of a program is directly related to the number of errors already found in that section.

Invalid and Unexpected Inputs

- Test cases must be written for INVALID and UNEXPECTED, as well as valid and expected, input conditions.
- In many systems, MOST of the code is concerned with input error checking and handling.

Anatomy of a Test Case

- What are the *parts* of a test case?
  1. a description of input condition(s)
  2. a description of expected results
- Where do ‘‘expected results’’ come from?

Testing Techniques

- **Black-Box:** Testing based solely on analysis of requirements (unit/component specification, user documentation, etc.). Also know as *functional testing.*
- **White-Box:** Testing based on analysis of internal logic (design, code, etc.). (But *expected* results still come from requirements.) Also known as *structural testing.*

Levels or Phases of Testing

- **Unit:** testing of the smallest programmer work assignments that can reasonably be planned and tracked (e.g., function, procedure, module, object class, etc.)
- **Component:** testing a collection of units that make up a component (e.g., program, package, task, interacting object classes, etc.)

Levels or Phases of Testing (cont’d)

- **Product:** testing a collection of components that make up a product (e.g., subsystem, application, etc.)
- **System:** testing a collection of products that make up a deliverable system

Levels or Phases of Testing (cont'd)

- Testing usually:
  - begins with *functional* (black-box) tests,
  - is supplemented by *structural* (white-box) tests, and
  - progresses from the unit level toward the system level with one or more integration steps.

Plan-Based Testing Process Activities

Test Planning

Test Design

Test Implementation

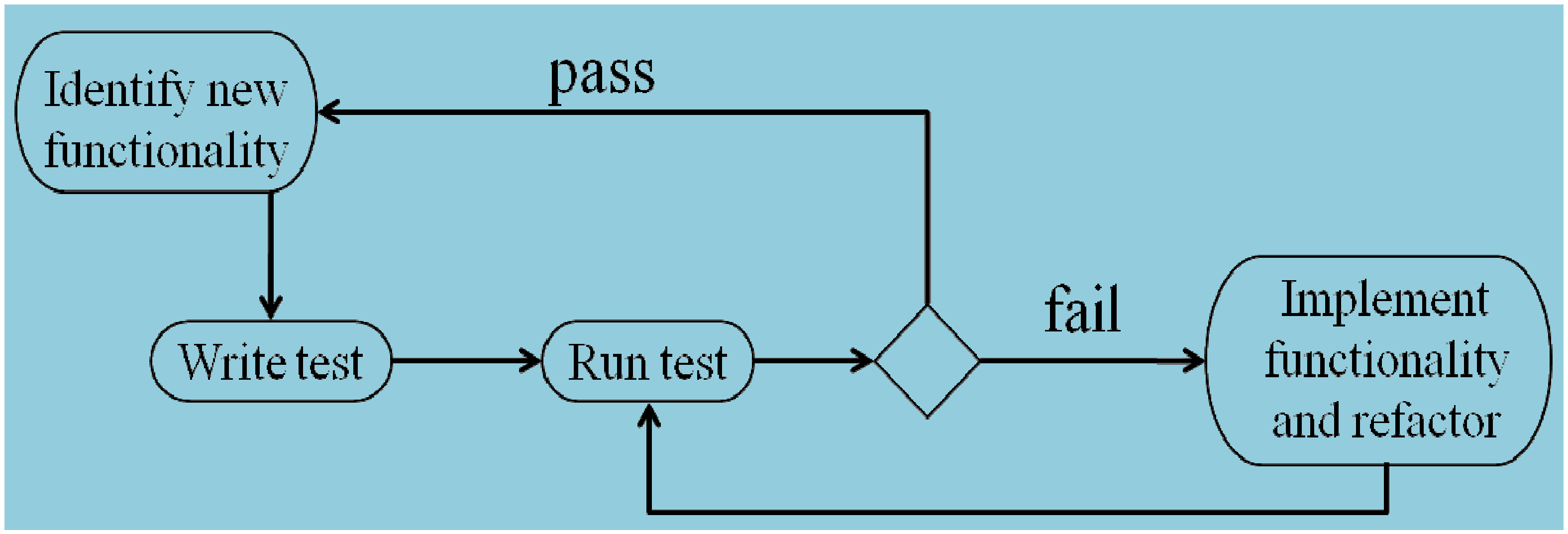
Test Execution

Execution Analysis

Result Documentation

Final Reporting

Test-Driven Development (TDD)


Identify new functionality
pass
fail
Write test
Run test
Implement functionality and refactor

Exhaustive Testing is Exhausting

- **Situation:**
  - A module has 2 input parameters.
  - Word size is 32 bits.
  - Testing is completely automated: 100 nanoseconds are required for each test case.
- **Question:** How long would it take to test this module *exhaustively*, i.e., covering every possible combination of input values?

Black-Box Testing Techniques

Definition of Black-Box Testing

- Testing based solely on **analysis of requirements** (specification, user documentation, etc.).
- Also know as *functional* testing.
- Black-box techniques apply to *all* levels of testing (e.g., unit, component, product, and system).

Test cases and Test suite

- Test a software using a set of carefully designed <u>test cases.</u>
- The set of all test cases is called the <u>test suite</u>

Test cases and Test suite

- A test case is a triplet [I,S,O]:
  - I is the data to be input to the system,
  - S is the state of the system at which the data is input,
  - O is the expected output from the system.

Design of Test Cases

- Two main approaches to design test cases:
  - Black-box approach
  - White-box (or glass-box) approach

White-box Testing

- Designing white-box test cases:
  - requires knowledge about the internal structure of software.
  - white-box testing is also called <u>structural testing</u>.

White-Box Testing

- There exist several popular white-box testing methodologies:
  - Statement coverage
  - branch coverage
  - path coverage
  - condition coverage
  - mutation testing
  - data flow-based testing

Coverage Testing

Statement Coverage

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

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

Statement coverage criterion

- Observing that a statement behaves properly for one input value:
  - no guarantee that it will behave correctly for all input values.
  - The statement coverage can be calculated as shown below:

$$\text{Statement coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

Example

- Consider code sample :

```
READ X
READ Y
I F X>Y THEN Z = 0
ENDIF
```

- To achieve 100% statement coverage of this code segment just one test case is required, one which ensures that variable A contains a value that is greater than the value of variable Y, for example,
- **X = 12 and Y = 10.**

-
  ```
  1 READ X
  2 READ Y
  3 Z =X + 2*Y
  4 IF Z> 50 THEN
  5 PRINT large Z
  6 ENDIF
  ```
- Although it isn’t completely correct, we **have numbered each** line and will regard each line as a statement. Let’s analyze the coverage of a set of tests on our six-statement program:
- TEST SET 1
  Test 1_1: X= 2, Y = 3
  Test 1_2: X =0, Y = 25
  Test 1_3: X =47, Y = 1

Which statements have we covered?

- In Test 1_1, the value of Z will be 8, so we will cover the statements on lines 1 to 4 and line 6.
- In Test 1_2, the value of Z will be 50, so we will cover exactly the same statements as Test 1_1.
- In Test 1_3, the value of Z will be 49, so again we will cover the same statements.
- Test 1_4: X = 20, Y = 25
- This time the value of Z is 70, = 100%.

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;       }
```

Euclid's GCD Algorithm

Euclid's GCD computation algorithm

- By choosing the test set {(x=3,y=3),(x=4,y=3), (x=3,y=4)}
  - all statements are executed at least once.

Branch Coverage

- Test cases are designed such that:
  - different branch conditions
    - given true and false values in turn.

Branch Coverage

- Branch testing guarantees statement coverage:
  - a stronger testing compared to the statement coverage-based testing.
  - Decision coverage also known as branch coverage or all-edges coverage.
  - A decision is an IF statement, a loop control statement (e.g. DO-WHILE or REPEAT-UNTIL), or a CASE statement, where there are two or more outcomes from the statement. With an IF statement, the exit can either be TRUE or FALSE, depending on the value of the logical condition that comes after IF.

- The decision coverage can be calculated as given below:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100\%$$

Example

-
  ```
  1 READ A
  2 READ B
  3 C = A – 2 *B
  4 IFC <0THEN
  5 PRINT “C negative”
  6 ENDIF
  ```

- Let’s suppose that we already have the following test, which gives us 100% statement coverage for code sample.

- TEST SET 2   Test 2_1: A = 20, B = 15

- The value of C is -10, so the condition ‘C < 0’ is True, so we will print ‘C negative’ .
- What other test would we need to exercise the False outcome and to achieve 100% decision coverage?

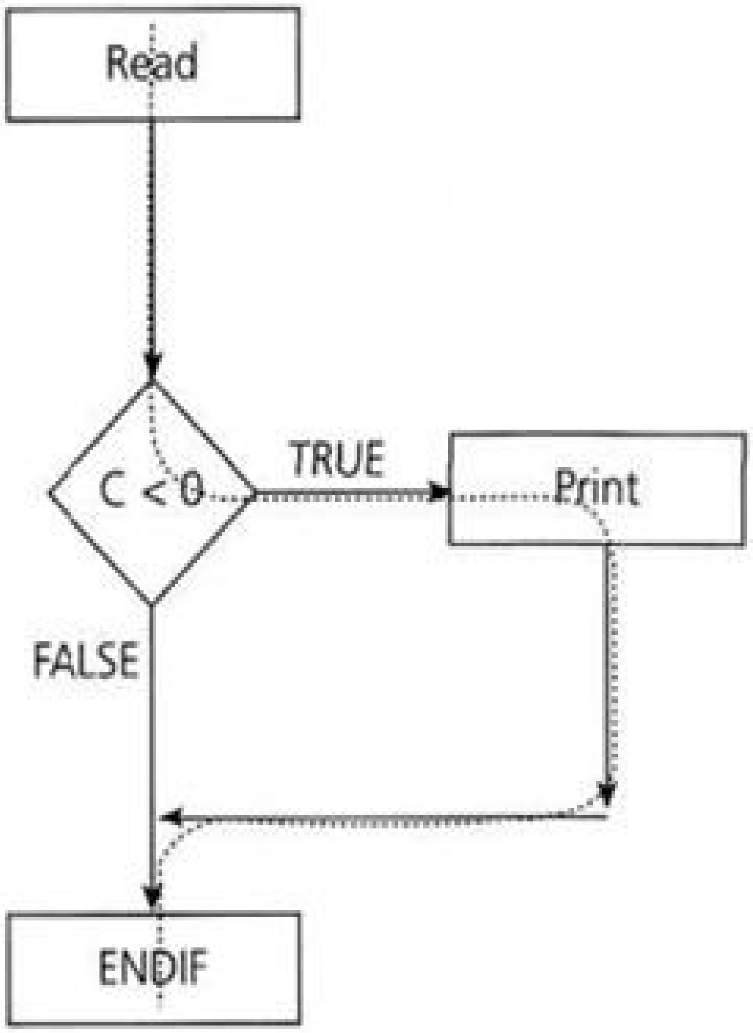


FIGURE 4.4 Control flow diagram for code sample 4.3

- The dotted line shows where Test 2_1 has gone and clearly shows.
-

  Let’s modify our existing test set by adding another test:
- **TEST SET 2**
  Test 2_1: A = 20, B = 15
  Test 2_2: A = 10, B = 2
- This now covers both of the decision outcomes, True (with Test 2_1) and False (with Test 2_2).

Stronger testing

- Test cases are a superset of a weaker testing:
  - discovers at least as many errors as a weaker testing
  - contains at least as many significant test cases as a weaker test.

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;   }

Example

- Test cases for branch coverage can be:
- {(x=3,y=3),(x=3,y=2), (x=4,y=3), (x=3,y=4)}

Condition Coverage

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

- This is closely related to decision coverage but has better sensitivity to the control flow.
- However, full condition coverage does not guarantee full decision coverage.
- Condition coverage reports the true or false outcome of each condition.
- Condition coverage measures the conditions independently of each other.

Example

- Consider the conditional expression
  - ((c1.and.c2).or.c3):
- Each of c1, c2, and c3 are exercised at least once,
  - i.e. given true and false values.

Condition coverage

- Consider a boolean expression having n components:
  - for condition coverage we require 2^n test cases.

Condition coverage

- Condition coverage-based testing technique:
  - practical only if n (the number of component conditions) is small.

Path Coverage

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

Linearly independent paths

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

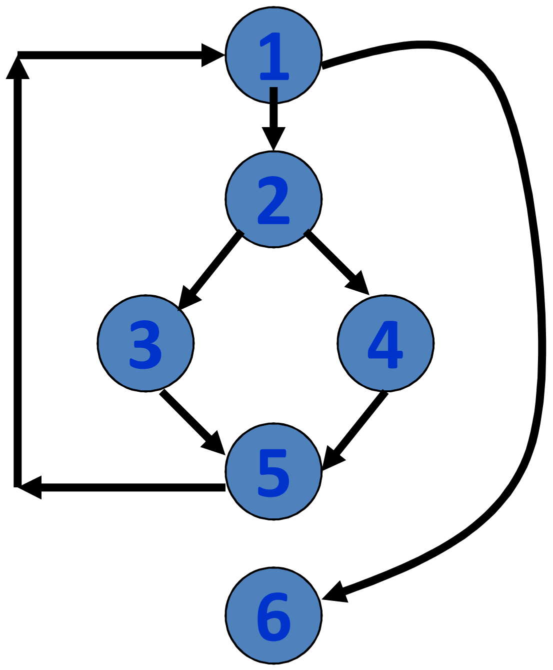
Control flow graph (CFG)

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

Example

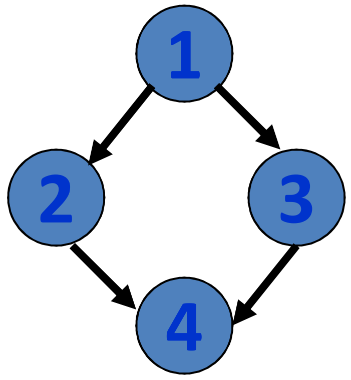
- int f1(int x,int y){
- 1 while (x != y){
- 2   if (x>y) then
- 3   x=x-y;
- 4   else y=y-x;
- 5 }
- 6 return x;   }

Example Control Flow Graph



How to draw Control flow graph?

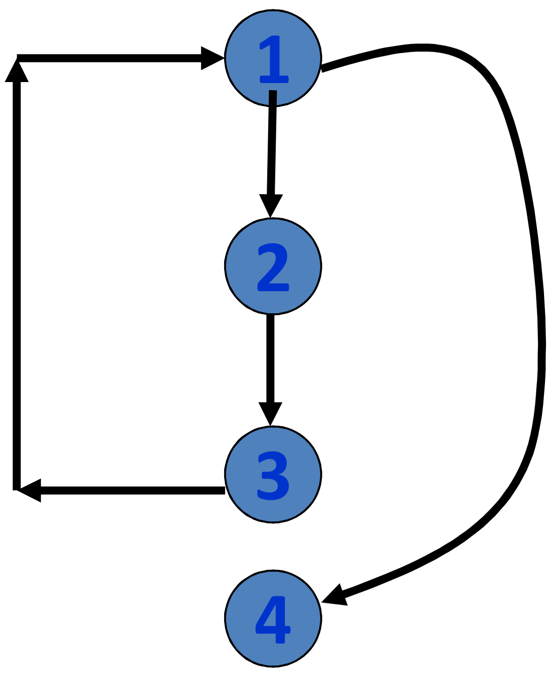
- Selection:
  - 1 if(a>b) then
  - 2   c=3;
  - 3 else   c=5;
  - 4 c=c*c;



$$\begin{tikzcd}
 & 1 \arrow[dl] \arrow[dr] & \\
2 \arrow[dr] & & 3 \arrow[dl] \\
 & 4 &
\end{tikzcd}$$

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.
  - There may be several terminal nodes for program.

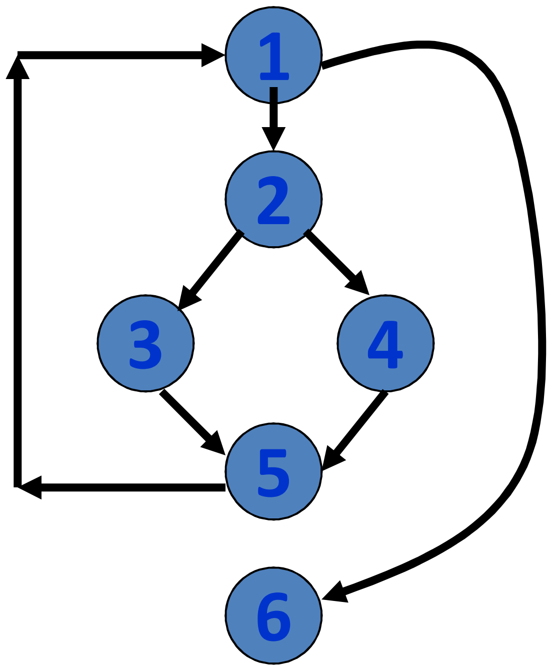
McCabe's cyclomatic metric

- An upper bound:
  - for the number of linearly independent paths of a program
- Provides a practical way of determining:
  - the maximum number of linearly independent paths in a program.

McCabe's cyclomatic metric

- Given a control flow graph G, cyclomatic complexity V(G):
  - V(G)= E-N+2
    - N is the number of nodes in G
    - E is the number of edges in G

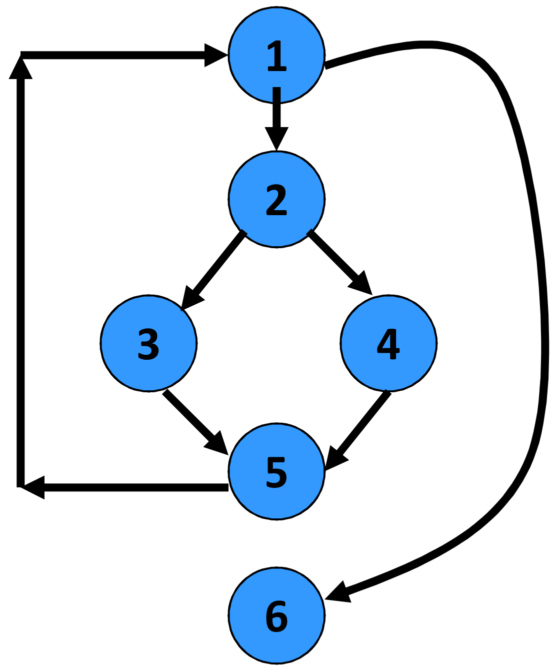
Example Control Flow Graph



Example

- Cyclomatic complexity = 7-6+2 = 3.

Example Control Flow Graph



Example

- From a visual examination of the CFG:
  - the number of bounded areas is 2.
  - cyclomatic complexity = 2+1=3.

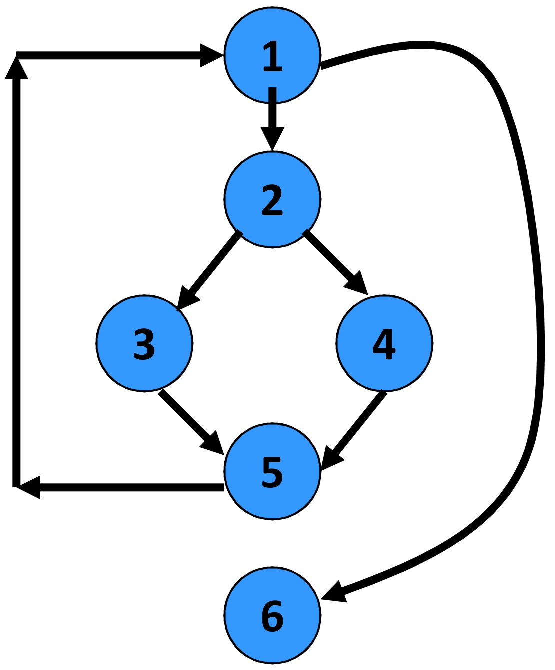
Cyclomatic complexity

- The cyclomatic complexity of a program provides:
  - a lower bound on the number of test cases to be designed
  - to guarantee coverage of all linearly independent paths.

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;        }
```

Example Control Flow Diagram



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)

Cyclomatic complexity

- From maintenance perspective,
  - limit cyclomatic complexity
    - of modules to some reasonable value.
  - Good software development organizations:
    - restrict cyclomatic complexity of functions to a maximum of ten or so.

Summary

- There are two approaches to testing:
  - black-box testing and
  - white-box testing.

Summary

- White box testing:
  - requires knowledge about internals of the software.
  - Design and code is required.

Summary

- We have discussed a few white-box test strategies.
  - Statement coverage
  - branch coverage
  - condition coverage
  - path coverage

Summary

- A stronger testing strategy:
  - provides more number of significant test cases than a weaker one.
  - Condition coverage is strongest among strategies we discussed.