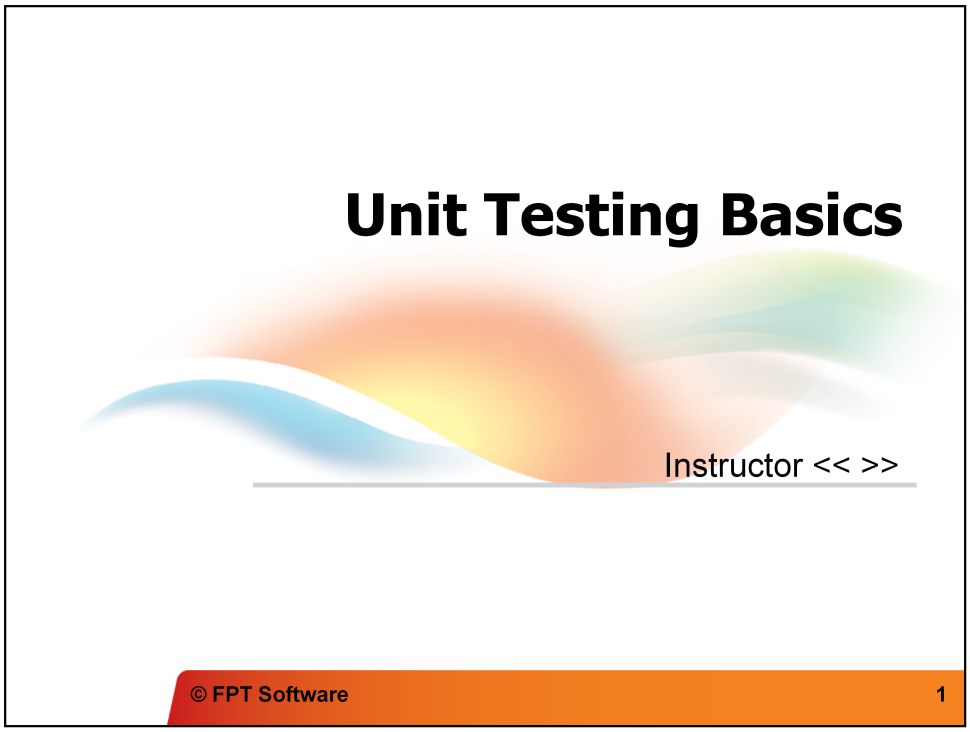

Unit Testing Basics
Instructor << >>
© FPT Software
1

Agenda

- Unit Test - What and Who?
- Unit Test - Why?
- Unit Test - How? (method, technique)
- Unit Test - Practice (3-5 labs) + Answer

Purpose: Practical Guide to Software Unit Testing

Objectives

The course helps attendees understand:

- Unit Test Fundamentals: Answer the question of what, why, when and how to do unit test
- Unit Test Technique: Introduce approaches and techniques to do unit test

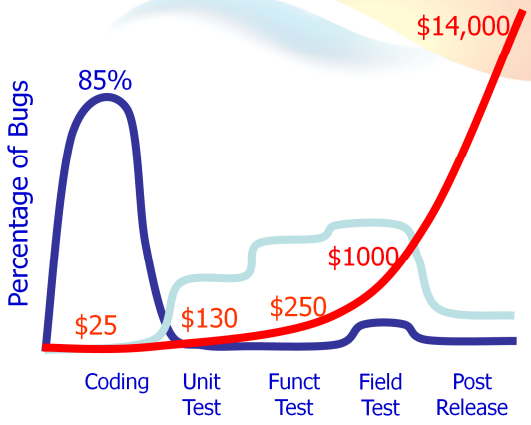
Unit Test – What and Who ?

- Unit Testing Actions:
  - Validate that individual units of software program are working properly
  - A unit is the smallest testable part of an application (In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is always a method)
  - Units are distinguished from modules in that modules are typically made up of units
- ★ Unit Testing Deliverables:
  - Tested software units
  - Related documents (Unit Test case, Unit Test Report)
- ★ Unit Testing Conductor: Development team

Unit Test – Why ?

- Ensure quality of software unit
- ★ Detect defects and issues early
- ★ Reduce the Quality Effort & Correction Cost

Cost of bugs



% Defects Introduced in this phase

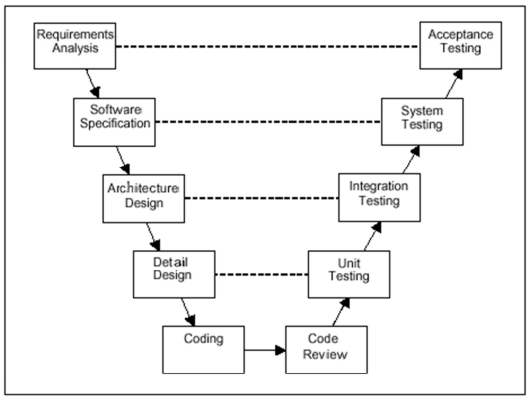
% Defects found in in this phase

$ Cost to repair defect in this phase

Source: Applied Software Measurement, Capers Jones, 1996

Unit Test – When ?

- After Coding
- Before Integration Test



Unit Test – How ?(1) Methodologies

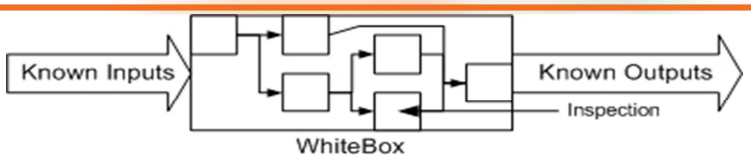
- UT processes
  - Follow the UT process defined in FSOFT UT Guidelines

Unit Test – How ?(2) Methodologies

A Simple Black box Specification

- Black-box testing
  - Functional testing: ensure each unit acts right as its design
  - Business testing: ensure the software program acts right as user requirement

Unit Test – How ?(3) Methodologies



- White-box testing
  - Developer does himself
    - Check syntax of code by compiler to avoid syntax errors
    - Run code in debug mode, line by line, through all independent paths of program to ensure that all statement of codes has been executed at least one time
    - Examine local data structure to ensure that data stored temporarily maintains its integrity during all all steps of code execution
    - Check boundary conditions to ensure that code will run properly at the boundaries established as requirements
    - Review all error handling paths
  - Apply WB Test techniques
    - See next slides

Unit Test – How ? Techniques

- Black box test (Functional)
  - Specification derived tests
  - Equivalence partitioning
  - Boundary value analysis
- White box (Structural)
  - Statement coverage
  - Decision (branch) coverage
  - Path coverage

Black box test – Specification derived test

- You can choose all or some statements in the specification of software
- Create test cases for each statements of specification
- Execute test cases to check test result will output as the specification

Example Specification

- Input - real number
- Output - real number

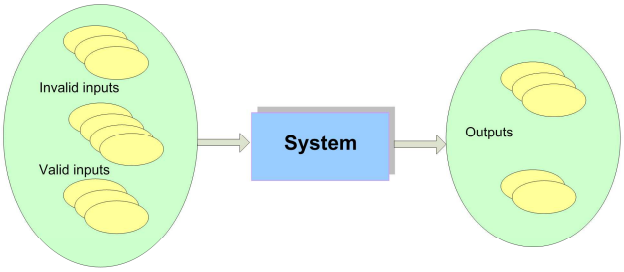
- When given an input of 0 or greater, the positive square root of the input shall be returned.
- When given an input of less than 0, the error message "Square root error - illegal negative input" shall be displayed and a value of 0 returned.

Example Test Cases

- **Test Case 1**: Input 4, Return 2
  - Use the first statement in the specification
  - ("When given an input of 0 or greater, the positive square root of the input shall be returned.").
- **Test Case 2**: Input -10, Return 0, Output "Square root error - illegal negative input"
  - Use the second and third statements in the specification
  - ("When given an input of less than 0, the error message "Square root error - illegal negative input" shall be displayed and a value of 0 returned.").

Black box test: Equivalence partitioning

- Divide the input of a program into classes of data from which test cases can be derived. This might help you to reduce number of test cases that must be developed.
- Behavior of software is equivalent for any value within particular partition
- A limited number of representative test cases should be chosen from each partition



Example Test Cases

Input partitions		Output partitions	
1	>= 0	**a**	>= 0
2	< 0	**b**	Error

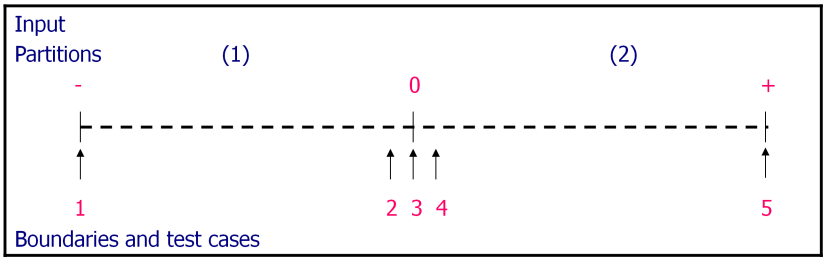
- **Test Case 1**: Input 4, Return 2
  - Use the >=0 input partition (1)
  - Use the >=0 output partition (a)
- **Test Case 2**: Input -10, Return 0, Output "Square root error - illegal negative input"
  - Use the < 0 input partition (2)
  - Use the "Error" output partition (b)

Black box test: Boundary value analysis

- It is similar to equivalence partitioning, is a selection of test cases, test input that check bounding values
- Anticipate that errors are most likely to exist at the boundaries between partitions
- Test the software at either side of boundary values

Example Test Cases

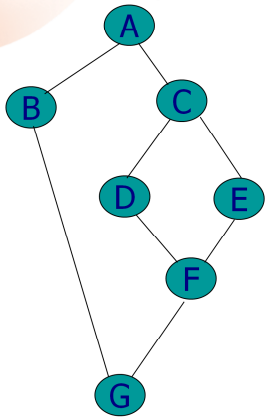
Input partitions		Output partitions	
1	>= 0	**a**	>= 0
2	< 0	**b**	Error



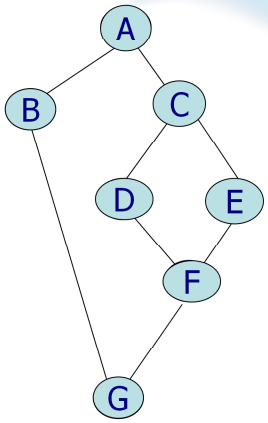
White box test : Node

Example:

```
buy(dress, bag) {
A  if dress already in bag
B    display "already bought it"
   else
C    if bag is full
D      display "bag is full"
     else
E      buy dress
       display "buy successful"
F    end if
G  end if
   }
```



White box test: Statement coverage



Total Nodes = 7;

Test case ABG covers 3 = 43%

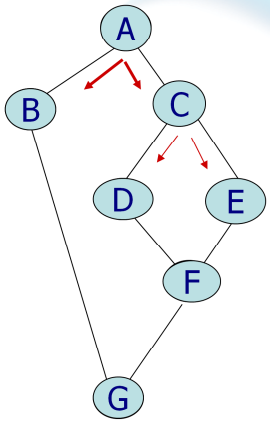
+

Test case ACDFG

Now covers 6/7 = 86%

Need 1 more for 100% statement coverage – ACEFG

White box test: Decision (branch) coverage



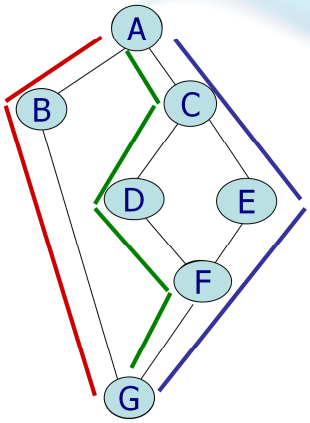
What branch coverage is achieved by ABG, ACDFG, ACEFG?

4 in total.

4 covered

So 4/4 = 100% branch coverage

White box test: Path coverage



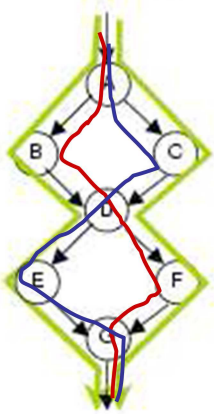
What path coverage is achieved by ABG, ACDFG, ACEFG?

3 in total.

3 covered

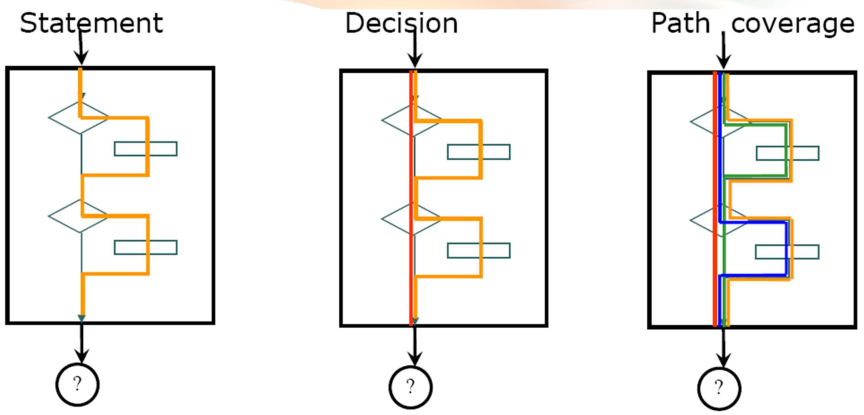
So 3/3 = 100% path coverage

Example: White box test case



- Test cases covering ABDEG and ACDFG cover 4/4 branches (100%) and 7/7 statements (100%)
- They, however, only cover 2/4 paths (50%).
- 2 more tests are required to achieve 100% path coverage
  - ABDFG
  - ACDEG

White box test: Comparison



Practice 1

```
public bool ValidPassword(string password)
{
    bool validPassword = false;
    // Check valid password length
    if(IsValidLength(password, minLength, maxLength))
    {
            validPassword = true;
            // Check valid password mix between lowcase and upcase
            if(!IsMixedCase(password))
               return false;
            // Check valid password mix between alpha & numeric
            if(!IsAphaNumeric(password))
            return false;
    }
    return validPassword;
}
```

How many unit test cases you must do to check this function?

IsValidLength() :function to check the length of passwod is valid or nots

IsMixedCase() : function to check the uppercase or lowercase

IsAphaNumeric() : functin to check the passwork is valid or not when it has both alpha and numeric

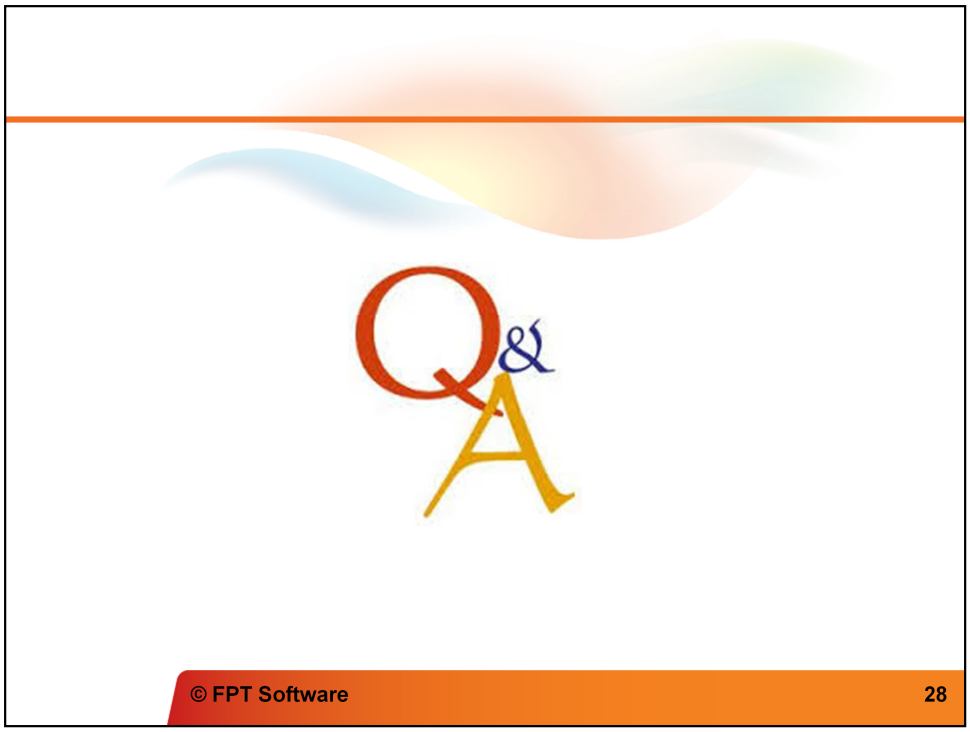
Result

6: There are 4 cases:

1.Test (ValidPasswordLength)
2.Test (ValidPasswordAlphaNumeric)
3.Test (ValidPasswordMixedCase)
4.Test với điều kiện đúng của 3 trường hợp trên (ValidPasswordLengthAlphaNumericMixedCase)

Discussion

- You find bugs while coding, is it unit test activity? Why?
- Which cases we apply Black-box and White-box testing?
- What are the result if we do not perform Unit test?
- As your opinion, What is the most difficult in Unit test activity?
- What are the products of Unit test activity?
- (Good question from student)


Q&A
© FPT Software
28