HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

SOICT

School of Information and Communication Technology

ONE LOVE. ONE FUTURE.

IT3180 – Introduction to Software Engineering

15 – Verification and Testing

Testing

- A *software test* executes a program to determine whether a property of the program holds or doesn't hold
- A test ***passes*** [***fails***] if the property **holds** [**doesn't hold**] on that run

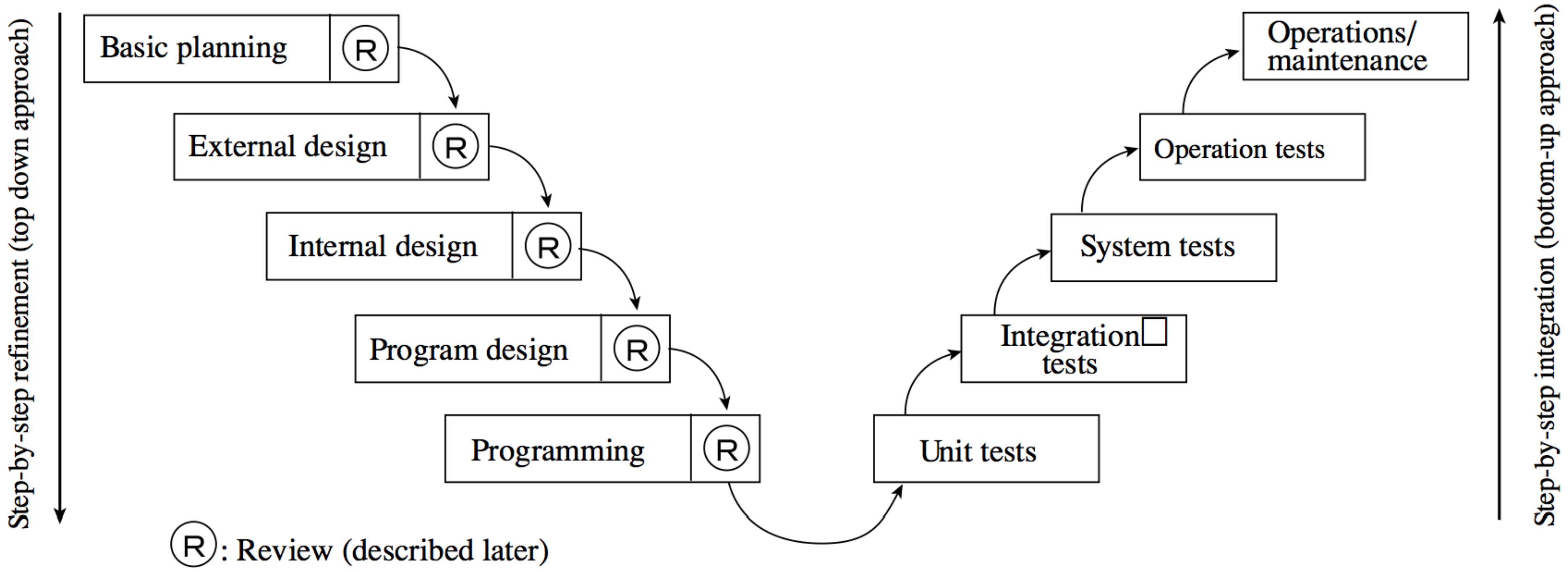
- *"[T]he means by which the presence, quality, or genuineness of anything is determined; a means of trial." –dictionary.com*

Software Quality Assurance

- Static analysis (assessing code without executing it)
- Proofs of correctness (theorems about program properties)
- Code reviews (people reviewing others' code)
- Software process (placing structure on the development lifecycle)
- …and many more ways to find problems and to increase confidence

V Model – Different levels of Test

- **Unit test**: ONE module at a time
- **Integration test**: The linking modules
- **System test**: The whole (entire) system
- **Acceptance test**: test from the user point of view



Test Levels – Unit Testing

- Unit Testing: Does each unit (class, method, etc.) do what it supposed to do?
  - Smallest programming units
  - Approaches: Black box and white box testing
  - Techniques, Tools

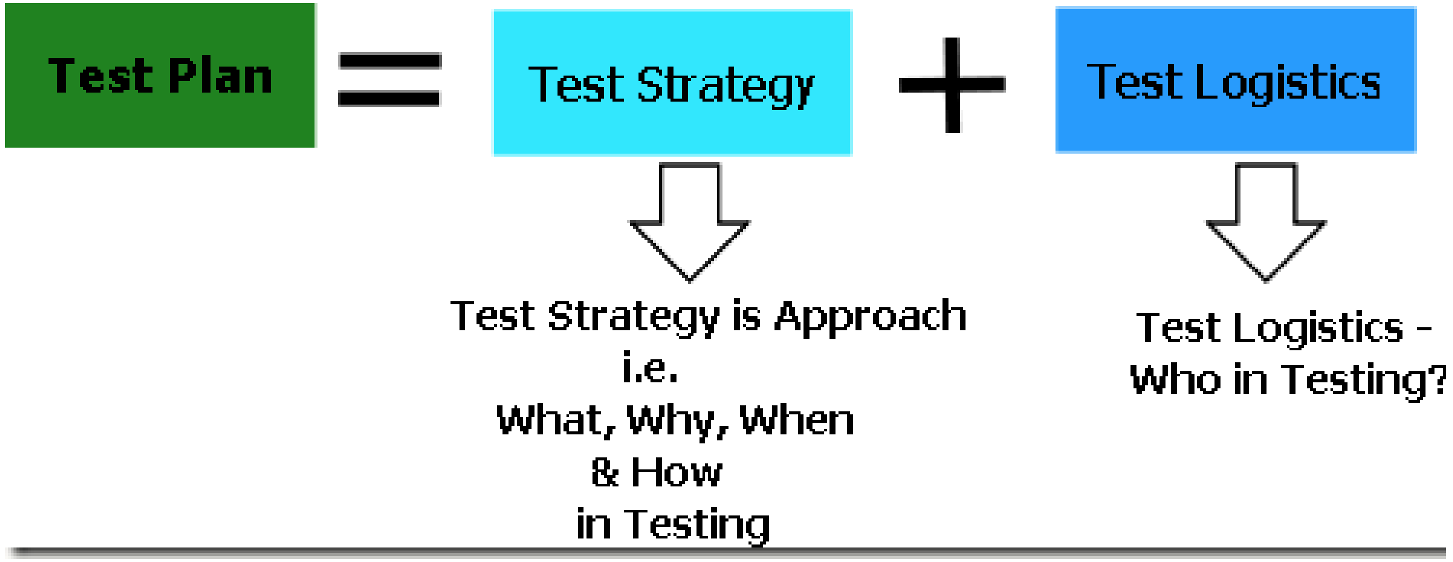
Test Levels – Integration Testing

- Integration Testing: do you get the expected results when the parts are put together?
  - Approaches: Bottom-up, top-down testing

Test Levels – System Testing

- System Testing: does it work within the overall system?
  - Approaches: Black box testing
- Acceptance Testing: does it match to user needs?

Terms

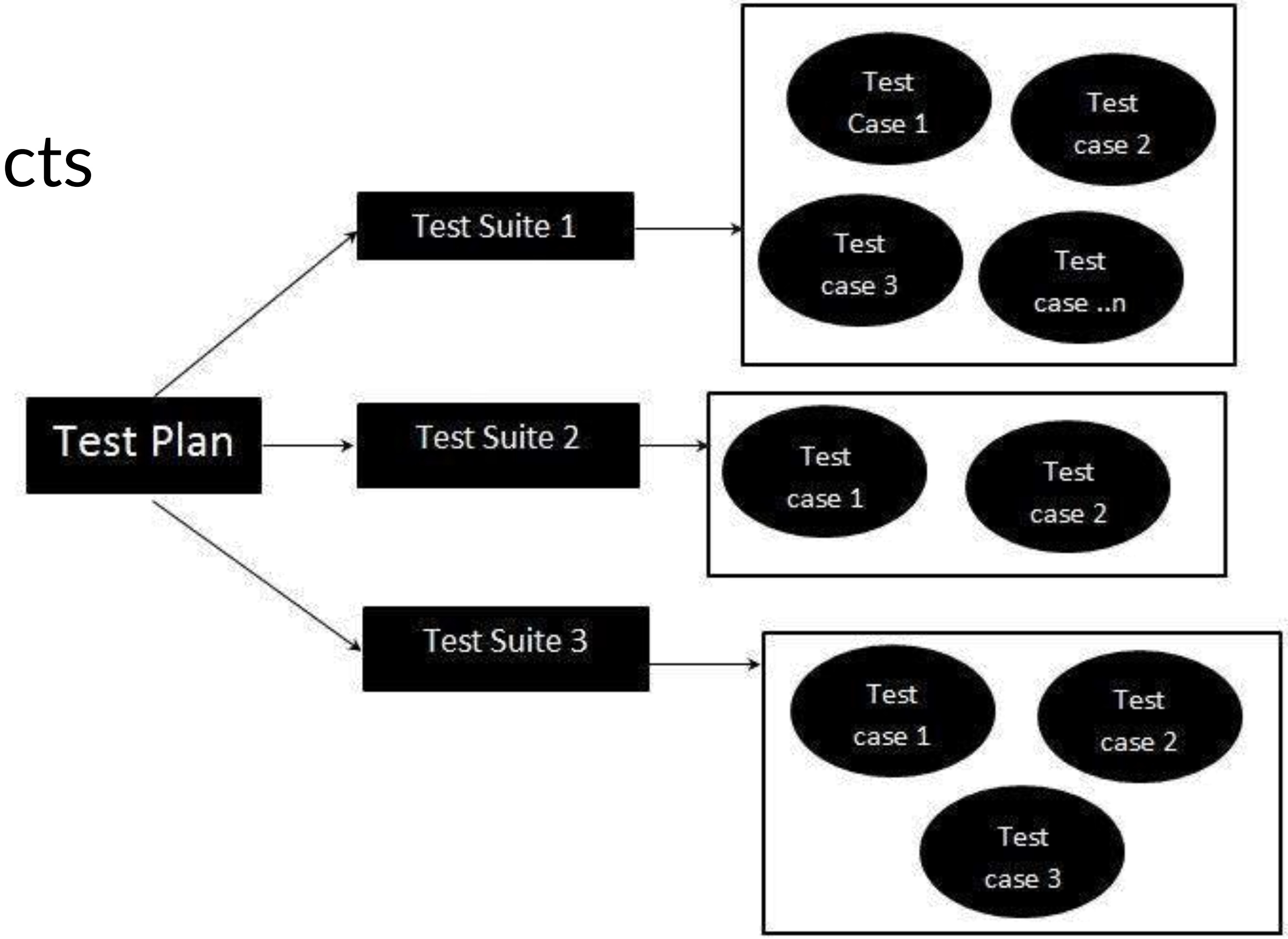

Test Plan
Test Strategy
Test Logistics
Test Strategy is Approach i.e. What, Why, When & How in Testing
Test Logistics - Who in Testing?

Terms (2)

- Test case
  - a set of conditions/variables to determine whether a system under test satisfies requirements or works correctly
- Test suite
  - a collection of test cases related to the same test work
- Test plan
  - a document which describes testing approach and methodologies being used for testing the project, risks, scope of testing, specific tools

Test suite

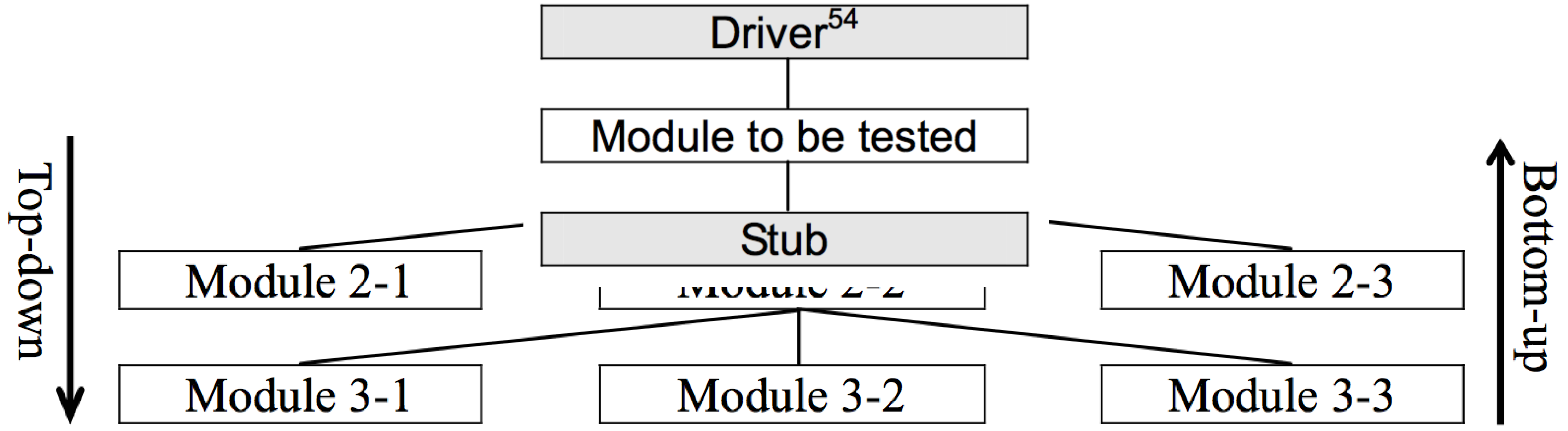
- Example of test suite
  - Test case 1: Login
  - Test case 2: Add New Products
  - Test case 3: Checkout
  - Test case 4: Logout



Integration Testing (1)

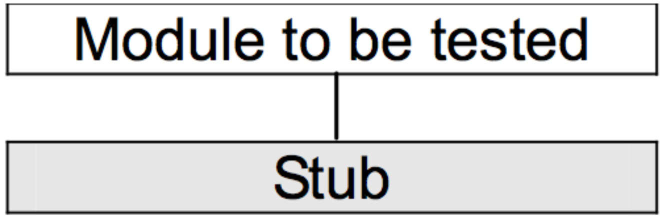
Examine the interface between modules as well as the input and output

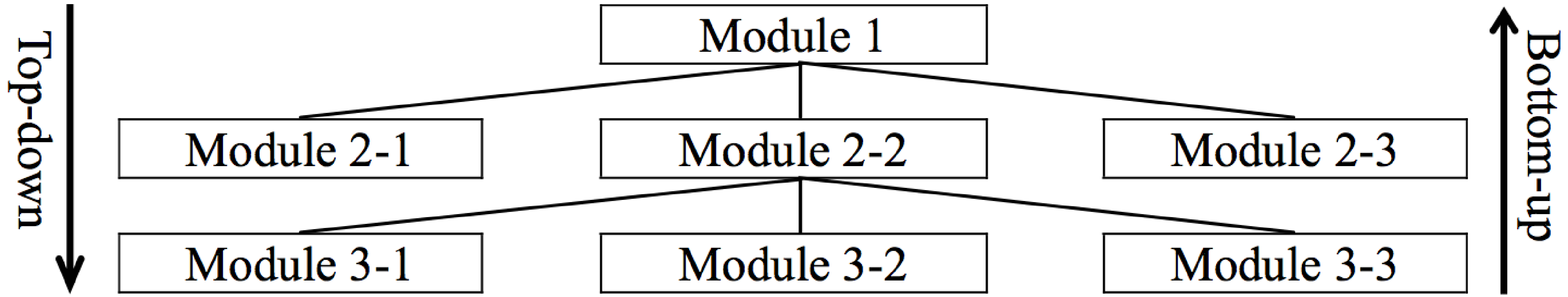
- Stub/Driver:
  - A program that simulates functions of a lower-level/upper-level module



Integration Testing (2) – Top-down Approach

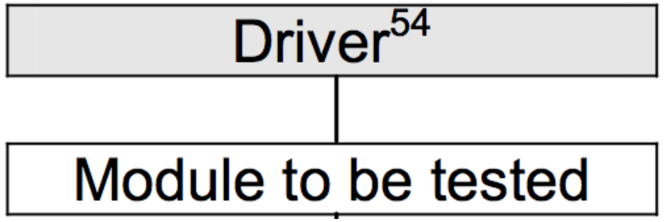
- Defects based on misunderstanding of specification can be detected early
- Effective in newly developed systems
- Need test stubs (can be simply returning a value)

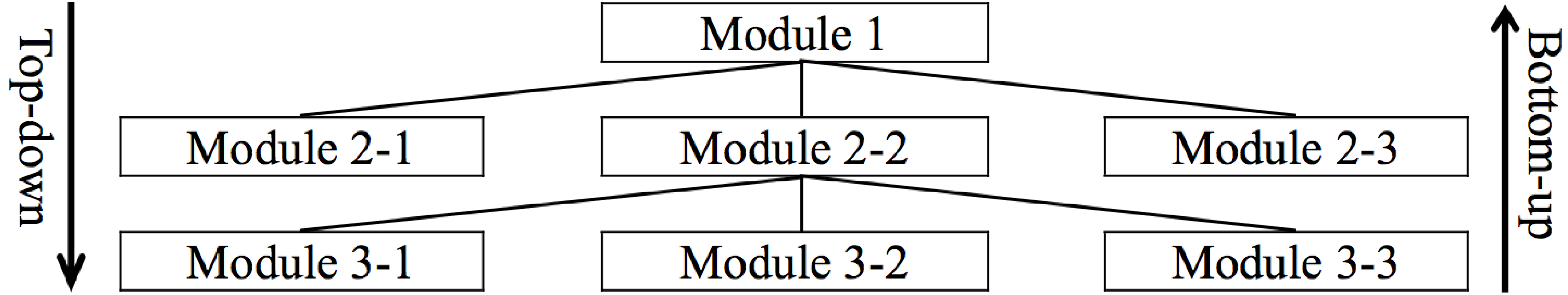




Integration Testing (2) – Bottom-up Approach

- Lower modules are independent => test independently and on a parallel
- Effective in developing systems by modifying existing systems
- Need test drivers (more complex with controlling)





Other integration test techniques

- Big-bang test
  - Wherein all the modules that have completed the unit tests are linked all at once and tested
  - Reducing the number of testing procedures in small-scale program; but not easy to locate errors
- Sandwich test
  - Where lower-level modules are tested bottom-up and higher-level modules are tested top-down

Regression test

"When you fix one bug, you introduce several new bugs"

- Re-testing an application after its code has been modified to verify that it still functions correctly
  - Re-running existing test cases
  - Checking that code changes did not break any previously working functions (side-effect)
- Run as often as possible
- With an automated regression testing tool

Test-case Design Techniques

A. Choose input data ("test inputs")
B. Define the expected outcome ("soict")
C. Run the unit ("SUT" or "software under test") on the input and record the results
D. Examine results against the expected outcome ("soict")

Specification	
Precondition	Postcondition
Implementation	

Black box

Must choose inputs *without knowledge* of the implementation

White box

Can choose inputs *with knowledge* of the implementation

Black-box vs. White box

Black box
Must choose inputs *without knowledge* of the implementation

- Has to focus on the behavior of the SUT
- Needs an “soict”
  - Or at least an **expectation** of whether or not an exception is thrown

White box
Can choose inputs *with knowledge* of the implementation

- Common use: *coverage*
- Basic idea: if your test suite never causes a statement to be executed, then that statement might be buggy

Unit & System Testing Techniques

For test case design

- Test Techniques for Black Box Test
  - Equivalence Partitioning Analysis
  - Boundary-value Analysis
  - Decision Table
  - Use Case-based Test
- Test Techniques for White Box Test
  - Control Flow Test
  - Data flow testing
  - Predicate testing

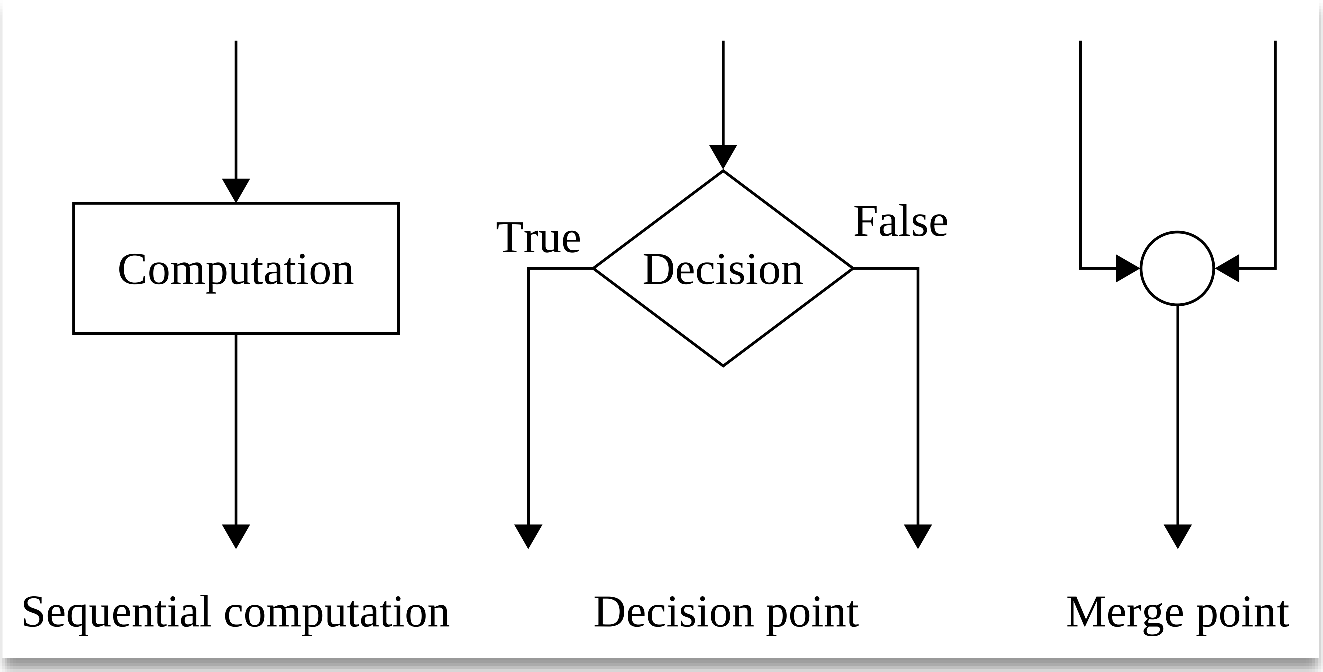
White-box Testing

Whitebox testing techniques

- Control Flow Testing
  - All-paths testing
  - Statement testing
  - Branch testing
- Data Flow Testing
  - All-defs coverage
  - All-uses coverage

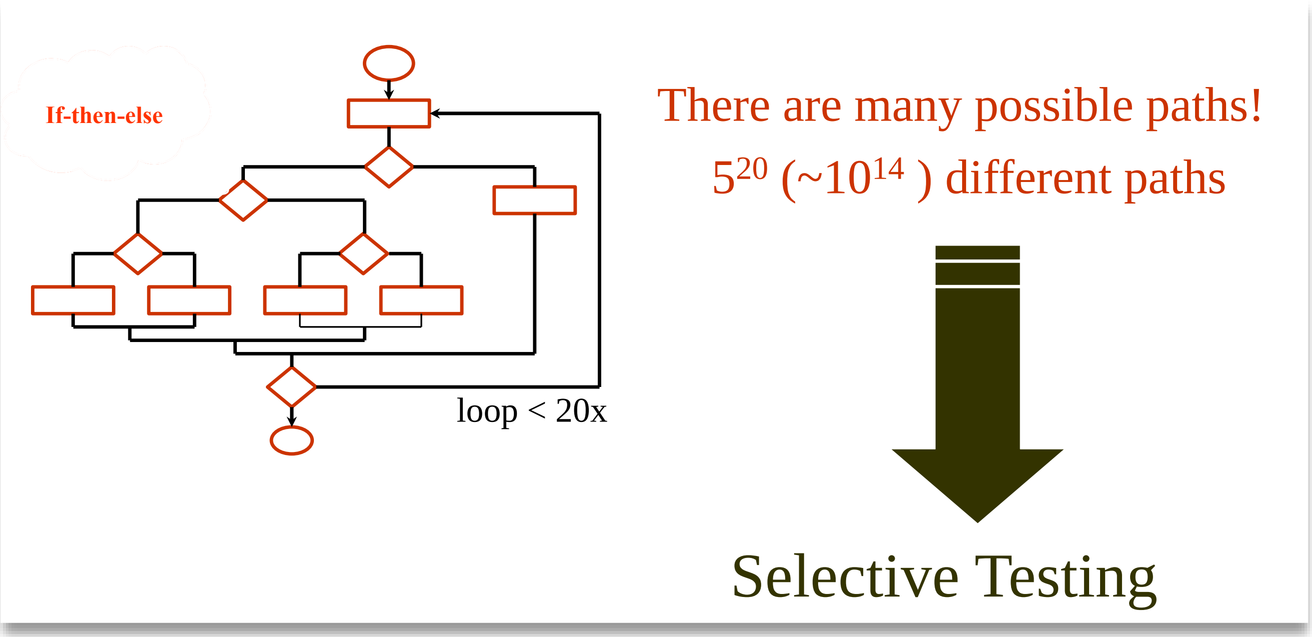
Control Flow Graph

- Represent the graphical structure of a program unit
- A sequence of statements from entry point to exit point of the unit



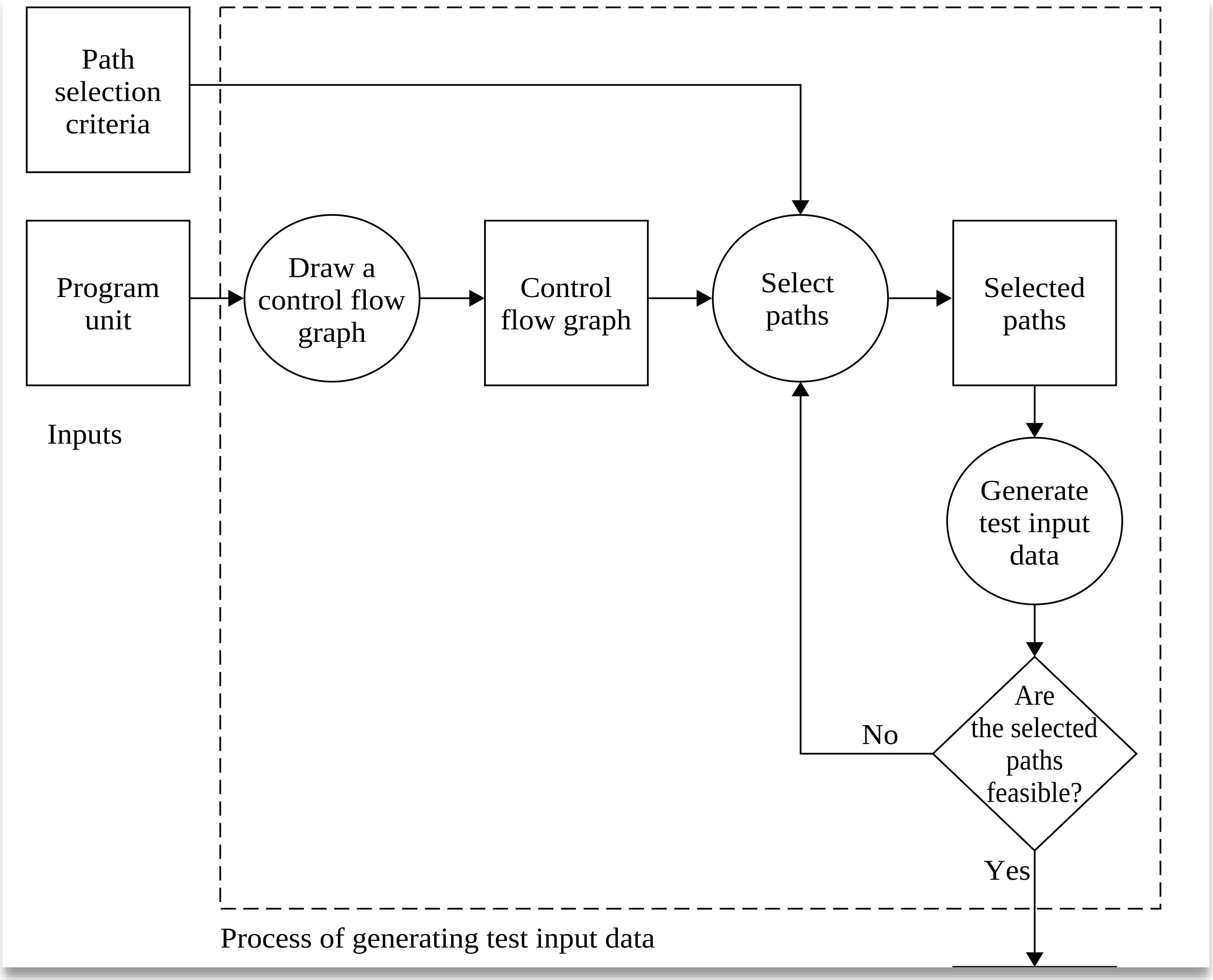
Control Flow Testing

- Main idea: select a few paths in a program unit and observe whether or not the selected paths produce the expected outcome
- Executing a few paths while trying to assess the behavior of the entire program unit



Outline of Control Flow Testing

- Inputs
  - Source code of unit
  - Path selection criteria
- Generate CFG: draw CFG from source code of the unit
- Selection of paths: selected paths to satisfy path selection criteria
- Generation of test input data

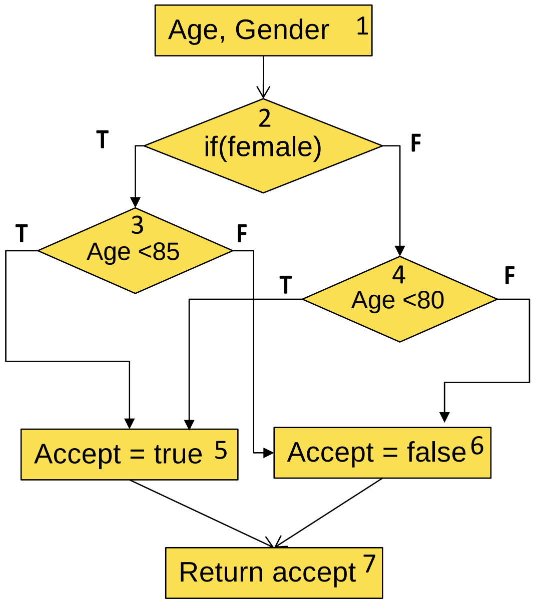


Path selection criteria

- Example:
  - Given the source code of the function `AccClient`
  - Draw the CFG

Life Insurance Example

```
bool AccClient(agetype
  age; gndrtype gender)
bool accept
  if(gender=female)
     accept := age < 85;
    else
     accept := age < 80;
return accept
```



All path coverage

- Objective: Design all possible test cases so that all paths of the program are executed
- 4 test cases satisfy the all paths coverage criterion

All paths

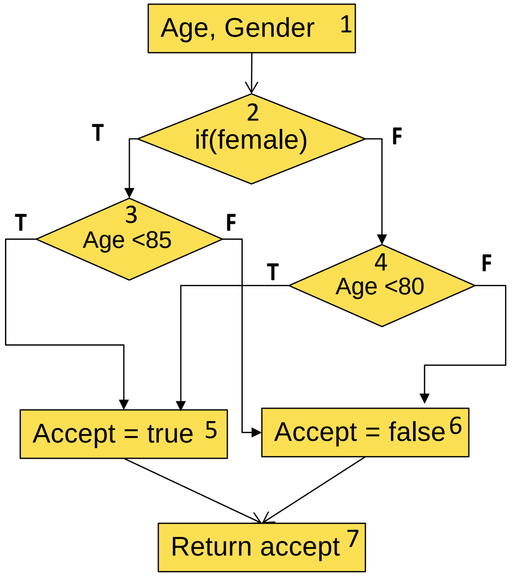
Female	Age < 85	Age < 80
Yes	Yes	Yes
Yes	Yes	No
~~Yes~~	~~No~~	~~Yes~~
Yes	No	No
No	Yes	Yes
No	Yes	No
~~No~~	~~No~~	~~Yes~~
No	No	No

<Yes,Yes,*> 1-2(T)-3(T)-5-7

<Yes,No,No> 1-2(T)-3(F)-6-7

<No,Yes,Yes> 1-2(F)-4(T)-5-7

<No,*,No> 1-2(F)-4(F)-6-7



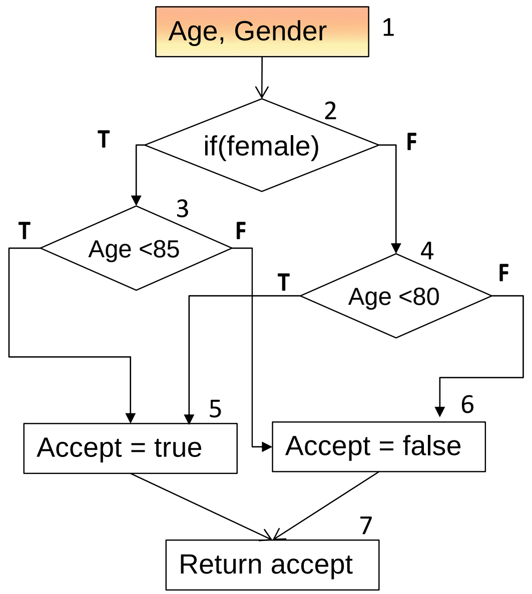
Statement Coverage

- Main idea: Execute each statement at least once
- A possible concern may be:
  - dead code

```
bool AccClient(agetype
  age; gndrtype gender)
bool accept
  if(gender=female)
    accept := age < 85;
   else
    accept := age < 80;
return accept
```

AccClient(83, female)->accept

AccClient(83, male) ->reject



Branch coverage

- Also called Decision Coverage
- A branch is an outgoing edge from a node
  - A rectangle node has at most one out going branch
  - All diamond nodes have 2 outgoint branches
- A decision element in a program may be one of
  - If – then
  - Switch – case
  - Loop
- Main idea: selecting paths such that every branch is included in at least one path

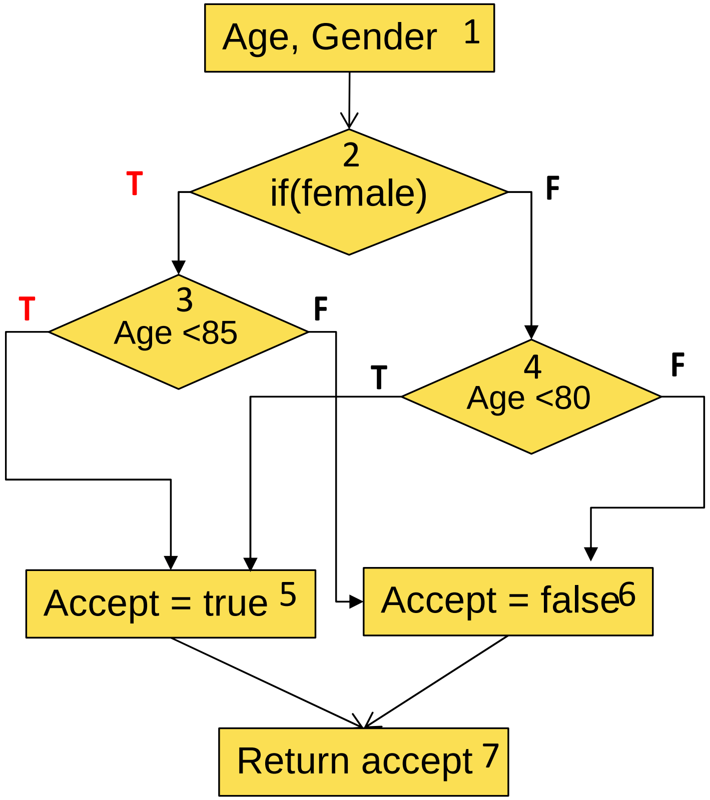
Example

Branch Coverage /1

AccClient(83,
female)->accept

```
bool AccClient(agetype
  age; gndrtype gender)
bool accept
  if(gender=female)
    accept := age < 85;
   else
    accept := age < 80;
return accept
```

true

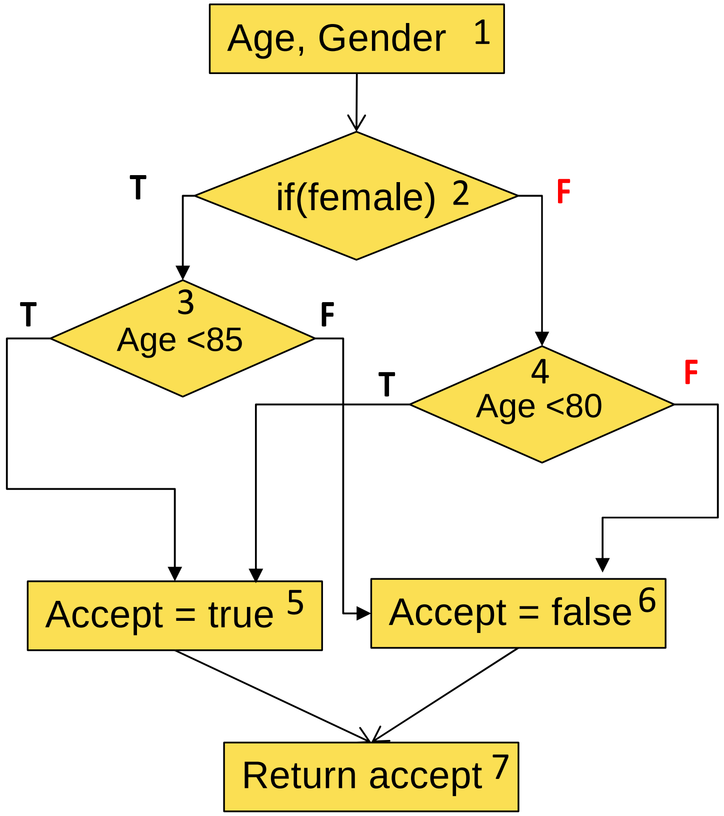


Branch Coverage /2

AccClient(83, male)
->reject

```
bool AccClient(agetype
  age; gndrtype gender)
bool accept
  if(gender=female)
     accept := age < 85;
   else
     accept := age < 80;
return accept
```

false



Example

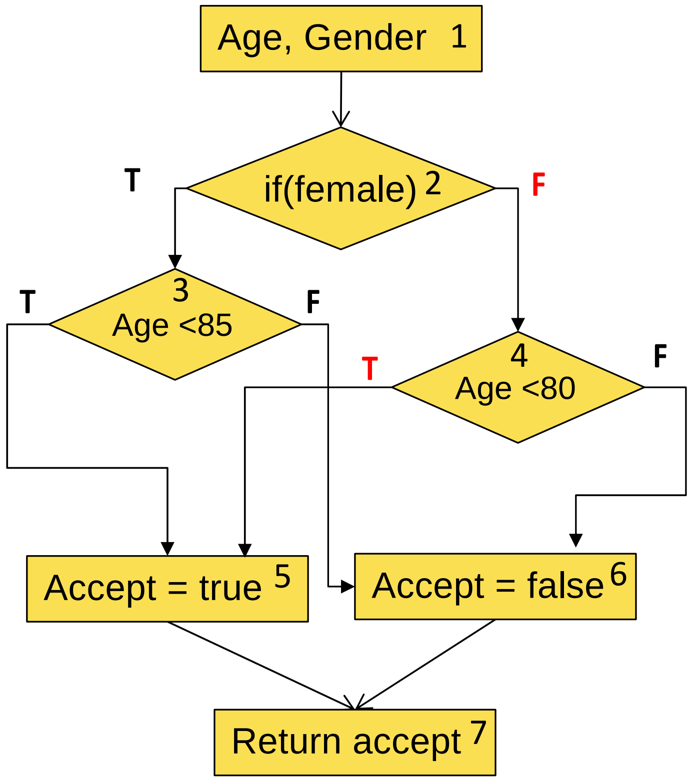
Branch Coverage /3

AccClient(78, male)->accept

```
bool AccClient(agetype
  age; gndrtype gender)
bool accept
  if(gender=female)
     accept := age < 85;
   else
     accept := age < 80;
return accept
```

true

true



Example

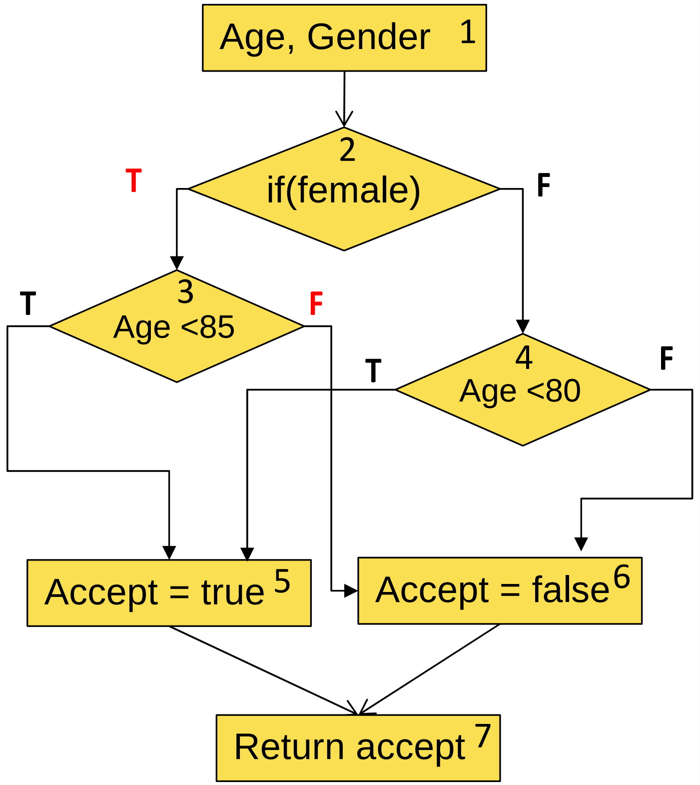
Branch Coverage /4

AccClient(88, female) ->reject

```
bool AccClient(agetype
  age; gndrtype gender)
bool accept
  if(gender=female)
     accept := age < 85;
   else
     accept := age < 80;
return accept
```

false

false



Comparing 3 criteria

- (1) All path coverage: assure 100% paths executed
- (2) Statement coverage: pick enough paths to assure that every source statement is executed at least once
- (3) Branch coverage: assure that every branch has been exercised at least once under some test
- (1) implies (3), (3) implies (2)
- These 3 criteria are also called as **Path Testing Techniques**

Example 2: Exponential Function

```
1   scanf("%d %d",&x, &y);
2   if (y < 0)
        pow = -y;
    else
        pow = y;
3   z = 1.0;
4   while (pow != 0) {
         z = z * x;
         pow = pow - 1;
5        }
6   if (y < 0)
         z = 1.0 / z;
7   printf ("%f",z);
```

Limitations of path testing

- Path Testing is applicable to new unit
- Limitations
  - Interface mismatches and mistakes are not taken
  - Not all initialization mistakes are caught by path testing
  - Specification mistakes are not caught

Black-box Testing

Black-box Techniques

- Equivalence Partitioning
- Boundary Analysis
- Table Decision

Equivalence Partitioning

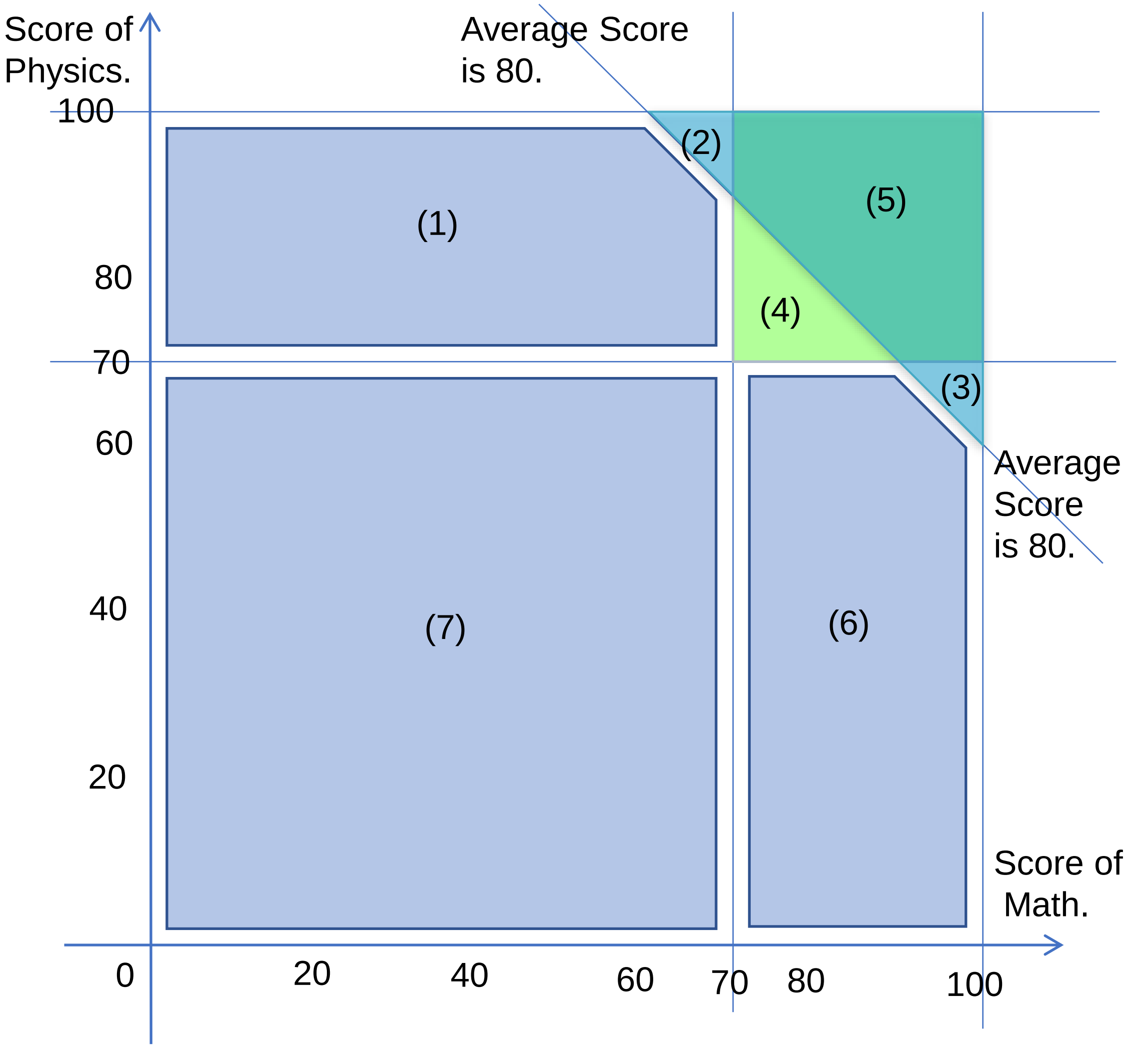
- Create the encompassing test cases by analyzing the input data space and dividing into equivalence classes
  - Input condition space is partitioned into equivalence classes
  - Every input taken from a equivalence class produces the same result

Example: Examination Judgment Program

- Program Title: “Examination Judgment Program”
- Subject: Two subjects as Mathematics, and Physics Judgment
- Specification:
  - Passed if
    - scores of both mathematics and physics are greater than or equal to 70 out of 100

      or,
    - average of mathematics and physics is greater than or equal to 80 out of 100
  - Failed => Otherwise

Example: Examination Judgment Program (2)



- How many equivalent classes?

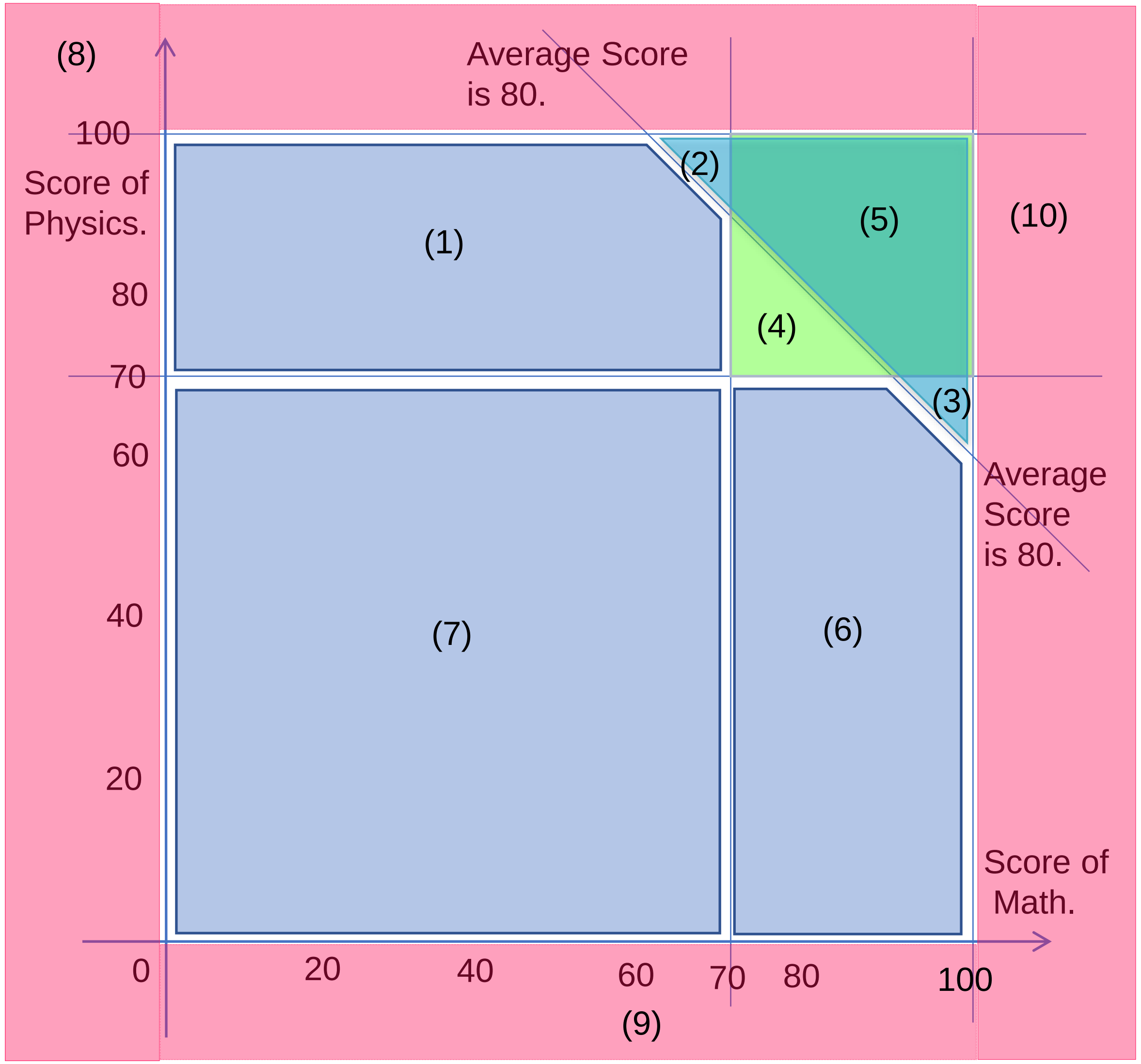
Score	Math.	Physics	Result
(1)	55	85	Failed
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	Failed
(7)	52	58	Failed

Equivalence Partitioning - Discussion

- What's about invalid data of the input?

- (8) Math = -15, Physics = 120 Both score are invalid.
- (9) Math = 68, Physics = -66 Physics score is invalid.
- (10) Math = 118, Physics = 85 Math score is invalid.

Example: Examination Judgment Program (3)



Some invalid data are added.

Score	Math.	Physics	Result
(1)	55	85	**Failed**
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	**Failed**
(7)	52	58	**Failed**
(8)	-15	120	**Invalid**
(9)	68	-66	**Invalid**
(10)	118	85	**Invalid**

Table Decision

- Relations between the conditions for and the contents of the processing are expressed in the form of a table
- A decision table is a tabular form tool used when complex conditions are combined
- Example: The conditions for creating reports from employee files

Under age 30	Y	Y	N	N
Male	Y	N	Y	N
Married	N	Y	Y	N
Output Report 1	-	X	-	-
Output Report 2	-	-	-	X
Output Report 3	X	-	-	-
Output Report 4	-	-	X	-

Example: Examination Judgment Program (4)

- Condition1: Mathematics score=>70
- Condition2: Physics score=>70
- Condition3: Average of Mathematics, and Physics =>80

	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7
Condition1	True	True	True	True	False	False	False	False
Condition2	True	True	False	False	True	True	False	False
Condition3	True	False	True	False	True	False	True(none)	False
"Passed"	Yes	Yes	Yes	---	Yes	---	N/A	--
"Failed"	---	---	---	Yes	---	Yes	N/A	Yes

Example: Examination Judgment Program (5)

- Invalid input data (integer)
  - Condition4: Mathematics score = valid that means “0=< the score =< 100”
  - Condition5: Physics score = valid that means “0=< the score =< 100”

	TCI1	TCI2	TCI3	TCI4
Condition4	Valid	Invalid	Valid.	Invalid
Condition5	Valid	Valid	Invalid	Invalid
“Normal results”	Yes	---	---	---
“Error message math”	---	Yes	---	Yes
“Error message phys”	---	---	Yes	Yes

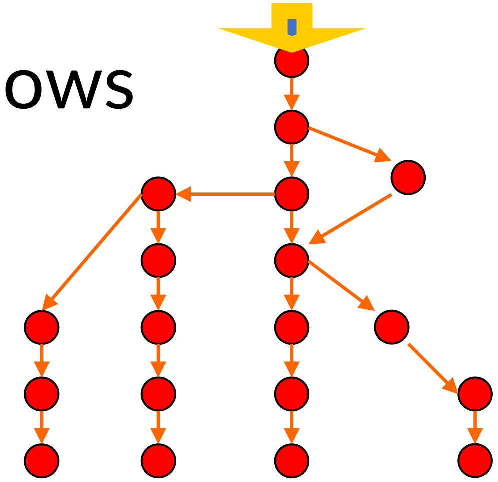
If both of mathematics score and physics score are invalid ➔ Two messages are expected to be output. Is it correct specifications?

Example: TriangleType

- Input: a, b, c >0
- (a+b)>c, (a+c)>b, (b+c)>a
- (a==b)||(b==c)||(c==a): Tam giac can
- (a==b)&&(b==c)&&(a==c): tam giac deu
- Tam giac thường
- Không phải tam giac nếu không thoả mãn các điều kiện bất đẳng thức.

Create Test case from Use case

- Identify all of the scenarios for the given use case
- Alternative scenarios should be drawn in a graph fo each action
- Create scenarios for
  - a basic flow,
  - one scenario covering each alternative flow,
  - and some reasonable combinations of alternative flows
- Create infinite loops



Test case for UC "Login"

- "Thành công"
  - Mã PIN đúng
- "Thất bại"
  - Mã PIN sai và số lần sai < 3
- "Khoá tài khoản"
  - Mã PIN sai và số lần sai >= 3

Mã PIN đúng	**Y**	**Y**	**N**	**N**
Số lần sai < 3	**Y**	**N**	**Y**	**N**
"Thành công"	x	N/A	-	-
"Thất bại"	-	N/A	x	-
"Khoá tài khoản"	-	N/A	-	x

15 – Verification and Testing

(end of lecture)

ONE LOVE. ONE FUTURE.