

Môn học

ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Chương II

TIỀN DỰ ÁN VÀ VÒNG ĐỜI DỰ ÁN

(P2)

Nội dung

❑ Tiền dự án

1. Rà soát hợp đồng
2. Kế hoạch phát triển và kế hoạch chất lượng

❑ Vòng đời dự án

3. Giai đoạn phát triển(Tiếp theo)
4. Giai đoạn bảo trì

II.3 Giai đoạn phát triển(tiếp theo)

□ Nội dung:

1. Các hoạt động SQA
2. Các yếu tố ảnh hưởng đến cường độ của các hoạt động SQA trong quy trình phát triển PM
3. Rà soát phần mềm
4. Kiểm thử phần mềm

II.3.4 Kiểm thử phần mềm

- A. Khái niệm cơ bản
- B. Các mức kiểm thử
- C. Kỹ thuật kiểm thử
- D. Quy trình kiểm thử
- E. Tổ chức kiểm thử

Software testing

❑ Định nghĩa:

Kiểm thử là quá trình vận hành chương trình để tìm ra lỗi trước khi chuyển giao cho người dùng cuối

(Glen Myers)

Tại sao cần phải kiểm thử?

- ❑ Phần mềm luôn luôn có lỗi
- ❑ Thất bại phần mềm sẽ rất đắt
- ❑ Giúp kiểm tra về độ tin cậy của PM
- ❑ Đánh giá và quản lí rủi ro

Mục tiêu kiểm thử PM

- ❑ Phát hiện ra nhiều lỗi nhất có thể
- ❑ Có một PM **đạt được chất lượng ở mức có thể chấp nhận**
- ❑ Thực hiện các tests cần thiết và hiệu quả trong phạm vi ngân sách và lịch biểu đã đưa ra.

Test Case

- ❑ Gồm 3 thành phần chính:
 - Các bước thực hiện test
 - Dữ liệu test
 - Kết quả mong đợi
- ❑ Một test case tốt:
 - Hiệu quả
 - Tổng quát
 - Dễ cập nhật, sửa chữa
 - Kinh tế

Kiểm thử triệt để (Exhaustive testing)

- ❑ Thực hiện tổ hợp tất các đầu vào có thể có và điều kiện tiên quyết.
- ❑ Cần thực hiện một số lượng lớn các Test case → Chiếm thời gian rất lớn, không thực tế.

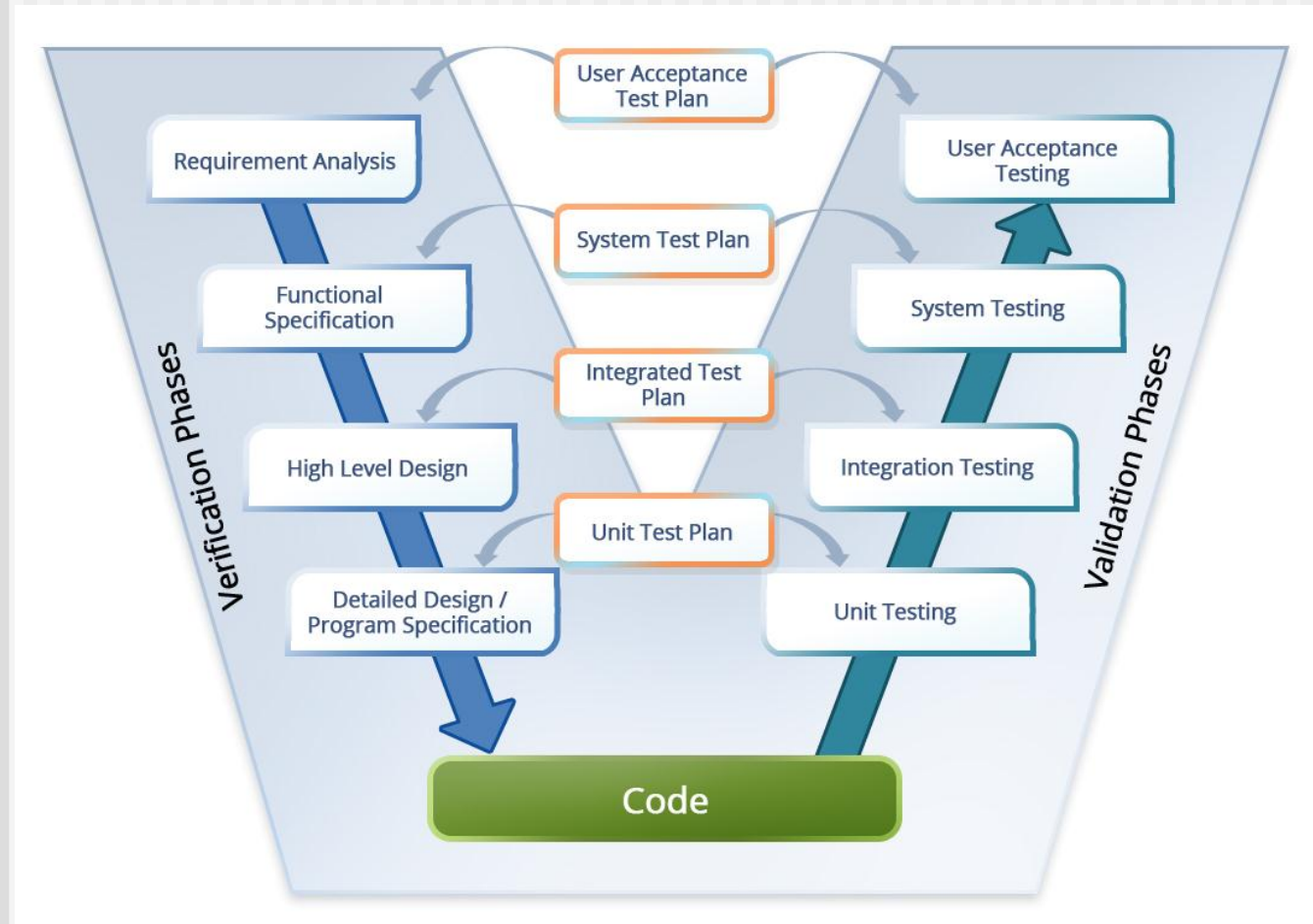
Kiểm thử bao nhiêu là đủ?

- ❑ Phụ thuộc vào kết quả test
- ❑ Phụ thuộc vào các rủi ro:
 - Bỏ lỡ các faults quan trọng
 - Chi phí phát sinh không cao
 - Phát hành phần mềm chưa được test
 - Thực thi các test không hiệu quả
 -

Kiểm thử bao nhiêu là đủ?

- ❑ Sử dụng các rủi ro để quyết định
 - Cái gì cần test trước
 - Cái gì cần nhấn mạnh
 - Cái gì chưa cần test trong thời điểm hiện tại
- Thực hiện sắp xếp độ ưu tiên cho các TCs

B. Các mức kiểm thử



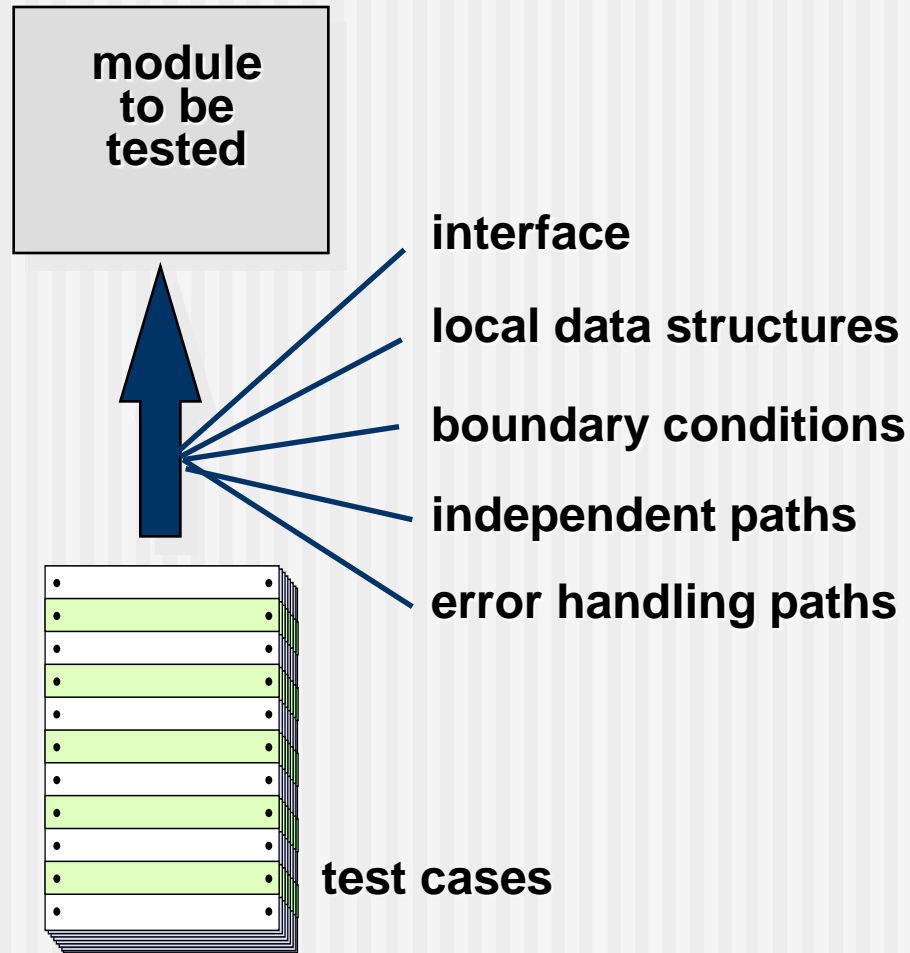
Kiểm thử đơn vị (Unit Testing)

- ❑ Mức thấp nhất
- ❑ Kiểm thử các thành phần nhỏ nhất có thể kiểm thử được như: function, class, module,...
- ❑ Các thành phần được test độc lập
- ❑ Do developer thực hiện

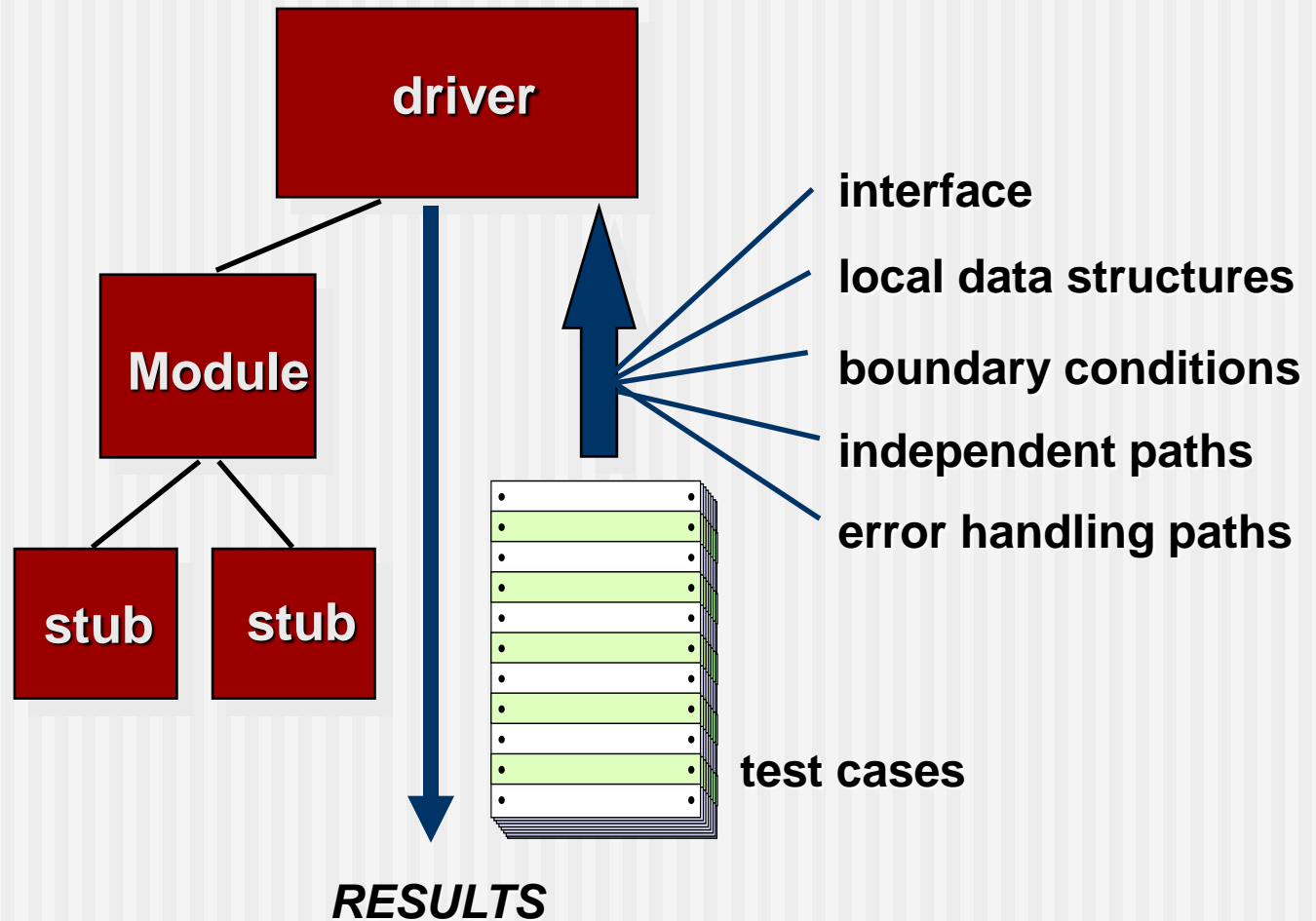
Chiến lược kiểm thử đơn vị

- ❑ Xác định các kĩ thuật kiểm thử(white-box)
- ❑ Đưa ra các tiêu chí để hoàn thành test
- ❑ Xác định mức độ độc lập khi thiết kế test
- ❑ Lập tài liệu về qui trình test và các hoạt động test(inputs và outputs)
- ❑ Những hoạt động lặp đi lặp lại sau mỗi lần fix lỗi hoặc thay đổi

Nội dung kiểm thử đơn vị



Môi trường kiểm thử đơn vị

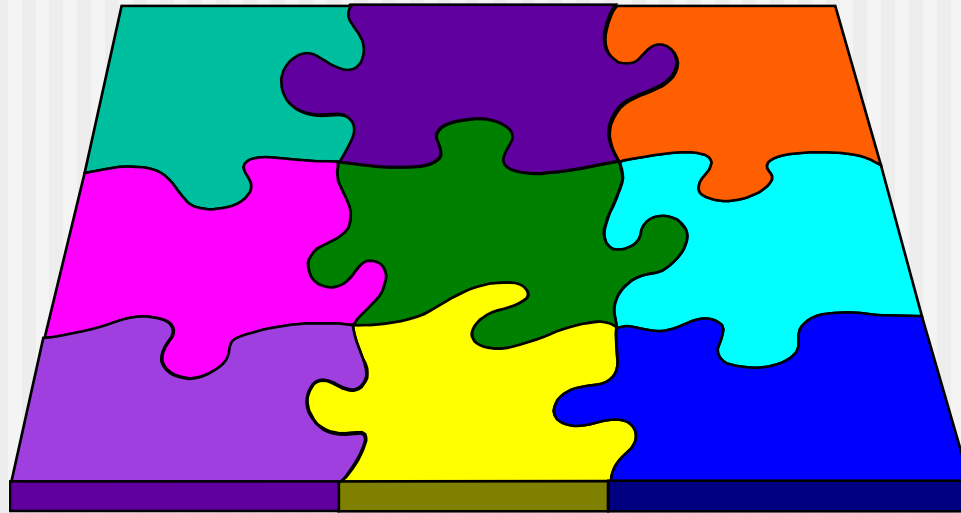


Kiểm thử tích hợp (Integration Testing)

- ❑ Tích hợp các thành phần phụ thuộc đã được test
- ❑ Tập trung tìm các lỗi:
 - Thiết kế và xây dựng kiến trúc PM
 - Các thành phần được tích hợp ở mức sub-system
 - Giao tiếp giữa các thành phần
- ❑ Do developers/testers thực hiện

Chiến lược kiểm thử tích hợp

- ❑ Có 2 hướng tiếp cận cơ bản:
 - Big-bang
 - Incremental (top-down, bottom-up)



Chiến lược kiểm thử tích hợp

❑ Big-bang:

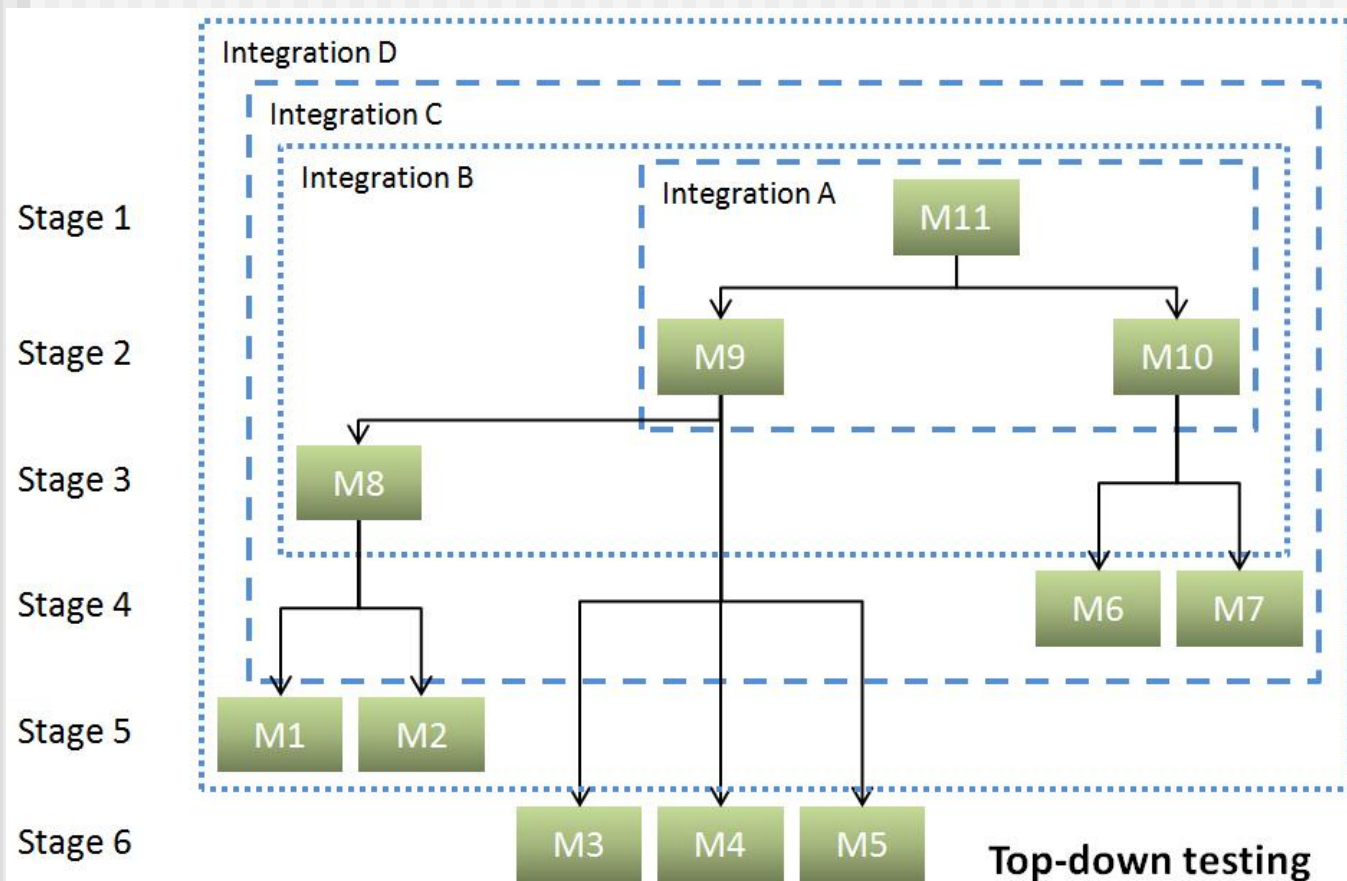
- Tại sao chúng ta lại kết hợp tất cả các thành phần và test cùng 1 lúc? → tiết kiệm thời gian
- Trong thực tế:
 - Mất nhiều thời gian để tìm lỗi và sửa lỗi
 - Test lại sau khi sửa lỗi sẽ phức tạp hơn nhiều

Chiến lược kiểm thử tích hợp

☐ Incremental

- Mức 0: các thành phần đã được test
 - Mức 1: 2 thành phần
 - Mức 2: 3 thành phần,
 -
- ☐ Giúp tìm ra các khiếm khuyết sớm → sửa lỗi sớm
 - ☐ Dễ dàng phục hồi sau lỗi

Tích hợp Top-down



Tích hợp top-down

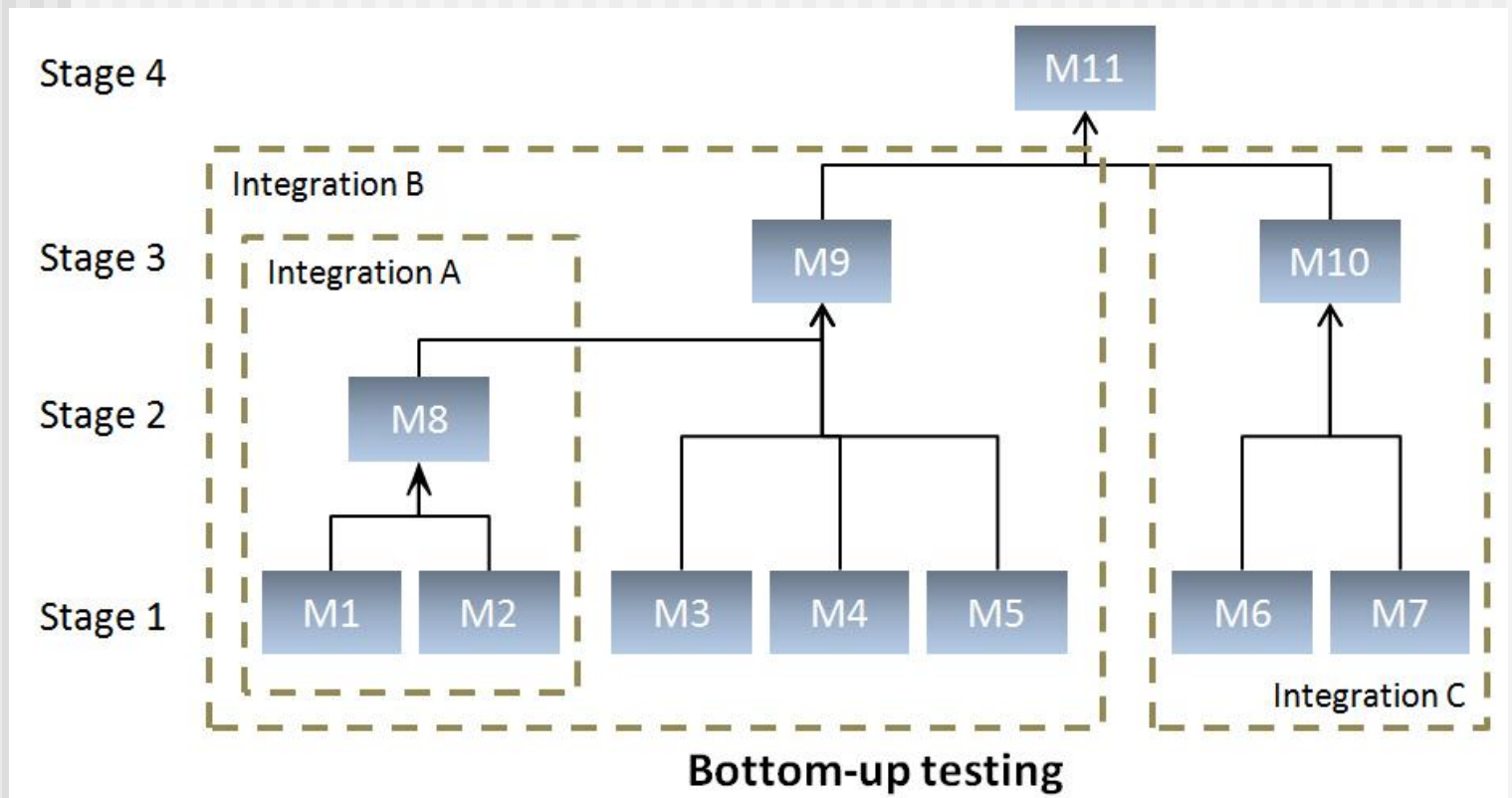
Ưu điểm

- ❑ Cấu trúc điều khiển quan trọng được test trước
- ❑ Hàm I/O được gọi sớm → viết test dễ
- ❑ Mô phỏng chức năng chính của PM sớm → nổi bật vấn đề liên quan đến yêu cầu

Khuyết điểm

- ❑ Phụ thuộc vào nhiều stub
- ❑ Khó khăn khi phân tích kết quả test

Tích hợp Bottom-up



Tích hợp Bottom-up

Ưu điểm

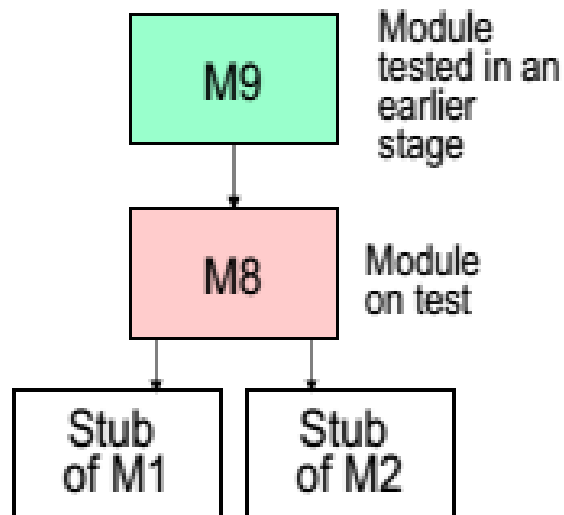
- ❑ Các mức thấp nhất được test đầu tiên
- ❑ Điều kiện test được tạo dễ dàng
- ❑ Dễ quan sát các kết quả test chi tiết

Khuyết điểm

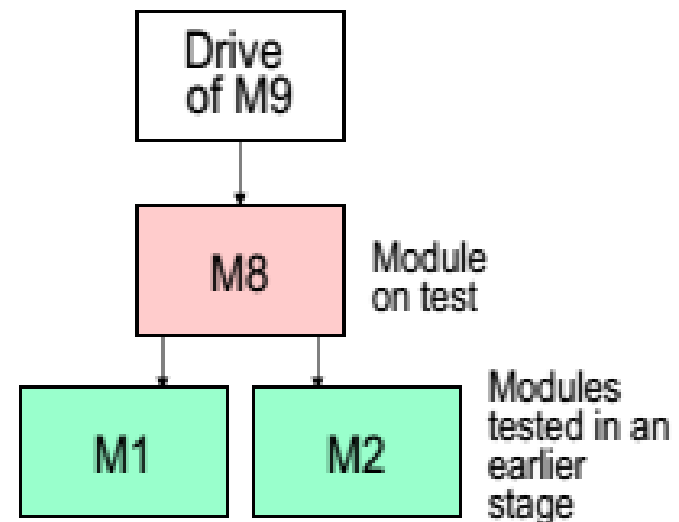
- ❑ Phải tạo các driver
- ❑ Chương trình tổng thể được quan sát trễ

Stubs and Drivers

Top-down testing of module M8



Bottom-up testing of module M8



Re-testing and regression testing

□ Re-testing:

- Chạy lại các test cases không thành công,
- Phiên bản mới chỉ bao gồm các khiếm khuyết cũ được fix
- Chạy lại chính xác test

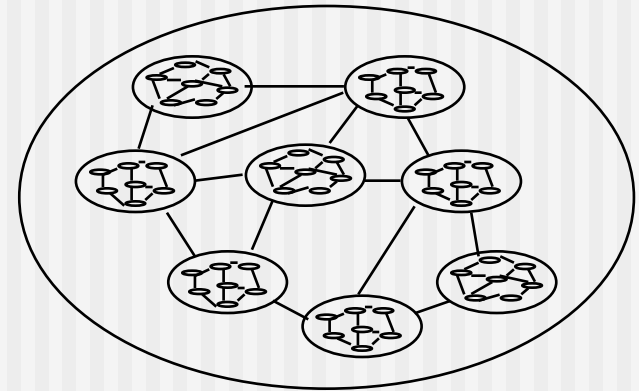
Re-testing and regression testing

□ Regression testing:

- Chạy lại tất cả các test cases đã được thực thi trước đó
- Phiên bản mới có thể bao gồm các khiếm khuyết cũ được fix, và các khiếm khuyết mới
- Có thể sử dụng các công cụ test tự động

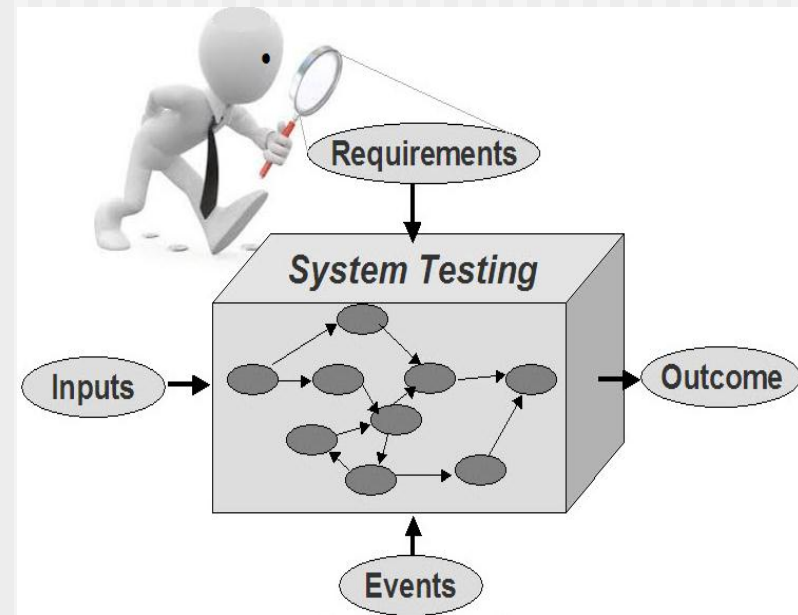
System testing

- ❑ Là bước tích hợp cuối cùng
- ❑ Kiểm thử chức năng (Functional)
- ❑ Kiểm thử phi chức năng (Non-Functional)
- ❑ Do các tester thực hiện



Functional System Testing

- ❑ Tìm các lỗi:
 - Input/Output
 - Giao diện người dùng
 - Giao tiếp của hệ thống với các phần khác
 - Hành vi của hệ thống



Non-functional System Testing

- ❑ Usability
- ❑ Security
- ❑ Documentation
- ❑ Storage
- ❑ Volume
- ❑ Configuration / installation
- ❑ Reliability
- ❑ Recovery
- ❑ Performance, load, stress

Usability Testing

- ❑ Đơn giản, hiệu quả khi sử dụng
- ❑ Giao diện nhất quán và phù hợp
- ❑ Message phù hợp và có ý nghĩa cho người sử dụng
- ❑ Hỗ trợ thông tin phản hồi
- ❑ Liên kết tắt

Security Testing

- ❑ Kiểm tra cơ chế bảo mật của hệ thống có hiệu quả không?
- ❑ Tester sẽ đóng vai trò là hacker
- ❑ Bài toán thiết kế hệ thống an ninh là:
 - Chi phí công cụ bảo vệ **nhỏ hơn** lợi ích bảo vệ khỏi đột nhập
 - Chi phí đột nhập **lớn hơn** lợi ích thu được từ đột nhập

Security Testing

- ❑ Ví dụ:
 - Passwords, encryption
 - Mức độ truy cập thông tin, quyền
- ❑ Một số kĩ thuật test lỗ hổng của ứng dụng Web:
 - SQL Injection
 - Cross-site Scripting(XSS)
 - ...

Documentation Testing

- ❑ Rà soát tài liệu
 - Kiểm tra định dạng, lỗi chính tả,
 - Độ chính xác về nội dung
- ❑ Kiểm tra tài liệu
 - Có làm việc không?
 - Tài liệu bảo trì
 - Hướng dẫn sử dụng

Performance Testing

- ❑ Xác định tốc độ, khả năng phân tải
- ❑ Tìm điểm “thắt cổ chai” → cải tiến nâng cao khả năng hoạt động của PM
- ❑ Timing Tests
 - Thời gian phục vụ và đáp ứng
 - Thời gian phục hồi CSDL
- ❑ Capacity & Volume Tests
 - Khối lượng/kích thước dữ liệu lớn nhất khả năng xử lý được

Stress/Load Testing

- ❑ Vận hành hệ thống khi sử dụng nguồn lực với số lượng, tần suất lớn
- ❑ **Load Testing:**
 - kiểm tra PM ở điều kiện liên tục tăng mức độ chịu tải, nhưng PM vẫn hoạt động được
- ❑ **Stress Testing:**
 - Kiểm tra PM ở trạng thái vận hành trong điều kiện bất thường

Configuration/Installation Testing

❑ Configuration tests

- Môi trường phần cứng, phần mềm khác nhau
- Nâng cấp 1 phần của hệ thống có thể dẫn đến xung đột với phần khác

❑ Installation Tests

- PM có thể cài đặt qua CD, networks,...
- Thời gian cài đặt
- Uninstall

Reliability Testing

- ❑ Mean Time Between Failures (MTBF)
để đo độ tin cậy
- ❑ Tạo ra một tập test đại diện, thu thập đủ thông tin để thống kê tỉ lệ thất bại

Recovery Testing

- ❑ Bắt phần mềm phải thất bại để xem khả năng phục hồi của nó đến đâu
- ❑ Có 2 cách phục hồi
 - Phục hồi tự động(back-up)
 - Phục hồi dựa trên sự can thiệp của con người

User acceptance testing

- ❑ Thẩm định xem các chức năng của PM có thỏa mãn sự mong đợi của khách hàng không?
- ❑ Do customer thực hiện

Tại sao người dùng nên tham gia test?

- ❑ Người dùng họ biết:
 - Những tình huống nghiệp vụ xảy ra trong thực tế
 - Cách người dùng thực hiện công việc của mình khi sử dụng hệ thống
 - Trường hợp đặc biệt có thể gây ra vấn đề
- ❑ Lợi ích:
 - Họ sẽ hiểu chi tiết về hệ thống mới

Kiểm thử Alpha và Beta

- ❑ Do người sử dụng đầu cuối thực hiện, không phải người đặt hàng
- ❑ Họ sẽ đưa ra các feedback về sản phẩm(phát hiện lỗi, đề nghị cải tiến,...)

Kiểm thử Alpha

- ❑ Do bên phát triển PM tiến hành
- ❑ Được tiến hành trong môi trường được điều khiển
- ❑ Dữ liệu thường là mô phỏng

Kiểm thử Beta

- ❑ Do bên khách hàng tiến hành
- ❑ Được tiến hành trong môi trường thực, không có sự kiểm soát của Dev
- ❑ Khách hàng sẽ báo cáo tất cả các vấn đề trong quá trình test một cách định kì

C.Kĩ thuật kiểm thử

- ❑ Static Testing
- ❑ Dynamic Testing

Kĩ thuật kiểm thử

- ❑ Là thủ tục chọn lựa hoặc thiết kế test tốt
- ❑ Dựa trên cấu trúc bên trong hoặc đặc tả của hệ thống PM
- ❑ Giúp tìm ra được nhiều lỗi
- ❑ Giúp đo nỗ lực test

Ưu điểm của các kĩ thuật

- ❑ Xác suất tìm ra lỗi của các tester là như nhau
- ❑ Kiểm thử hiệu quả hơn: Tìm ra được nhiều lỗi hơn với chi phí ít
 - Tập trung vào các loại lỗi nhất định
 - Kiểm thử đúng hướng
 - Tránh trùng lặp

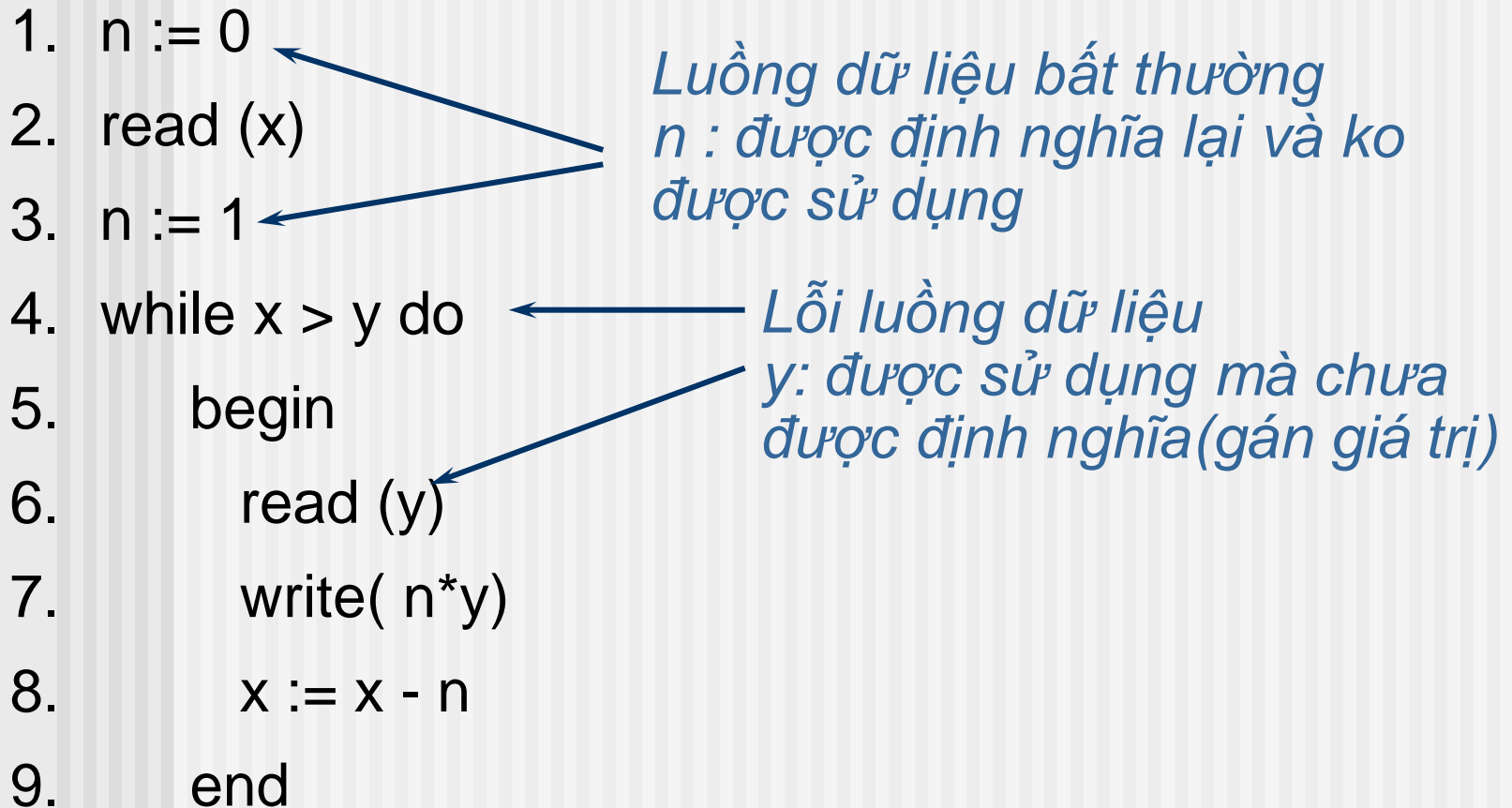
Static Testing

- ❑ Là kĩ thuật kiểm thử mà không thực thi code
- ❑ Kiểm thử dựa trên công cụ (**static code analysis**)
 - Data Flow Analysis
 - Control Flow Analysis

Phân tích luồng dữ liệu

- ❑ Xem xét các biến trong chương trình
 - Khai báo, định nghĩa biến
 - Sử dụng biến
 - Biến phải được khai báo trước khi sử dụng
 - Tham chiếu tới biến chưa được gán giá trị(defined)
 - Biến chưa bao giờ được sử dụng
 - Phạm vi tồn tại của biến

Phân tích luồng dữ liệu



Phân tích luồng điều khiển

- ❑ Vòng lặp vô hạn, đệ quy vô hạn
- ❑ Mã “chết”
- ❑ Nhảy đến nhãn không xác định
- ❑ Độ phức tạp của lưu đồ
- ❑

Unreachable code

```
int f(int x, int y)
{
    return x+y;
    int z=x*y;
}
```

```
double x = sqrt(2);
if (x > 5)
{
    printf("%d",x);
}
```

Ưu điểm và nhược điểm

❑ Ưu điểm:

- Tìm ra khiếm khuyết khó thấy
- Đưa ra những đánh giá về chất lượng code

❑ Nhược điểm:

- Không phân biệt được “fail-safe” code với lỗi thực khi lập trình → nhiều fail message
- Không thực thi code

Công cụ

❑ Multi-language

- Moose : C/C++, Java, Smalltalk, .NET,...
- Copy/Paste Detector (CPD): Java, JSP, C, C++ and PHP code.
- CheckKing: Java, JSP, Javascript, HTML, XML, .NET, PL/SQL, embedded SQL, C/C++, Cobol,...

Công cụ

- ❑ **C/C++**

- [Cppcheck](#), [Frama-C](#), ...

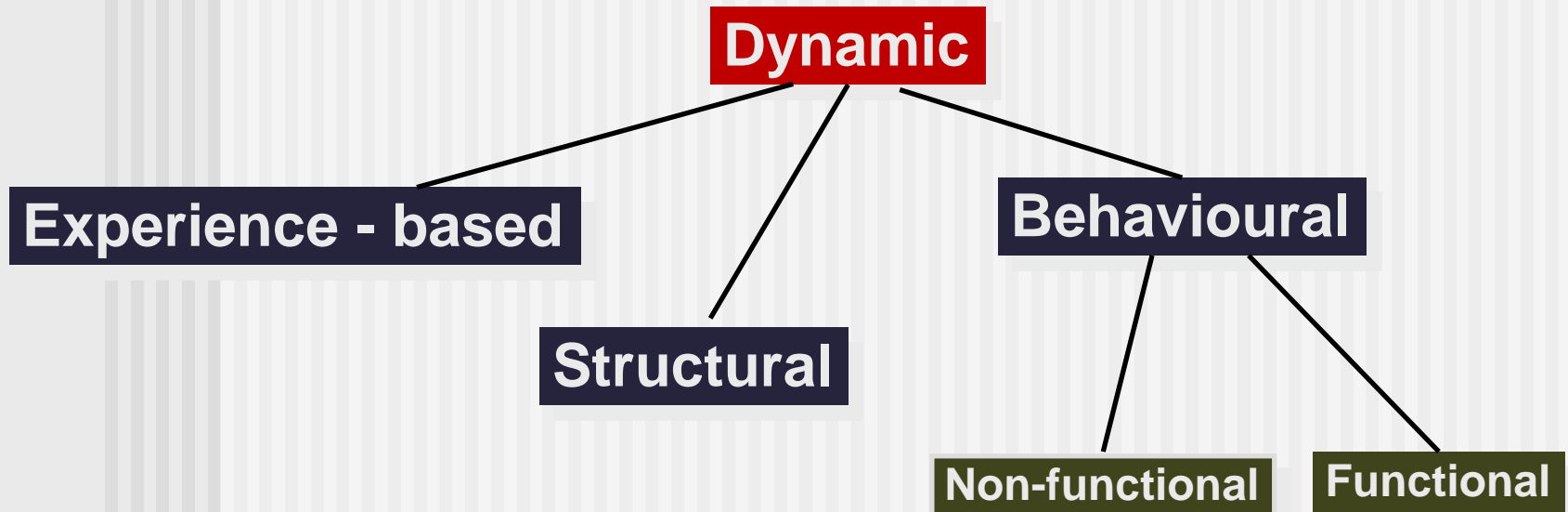
- ❑ **Java**

- [Checkstyle](#), [FindBugs](#), [Jtest](#),...

Dynamic Testing

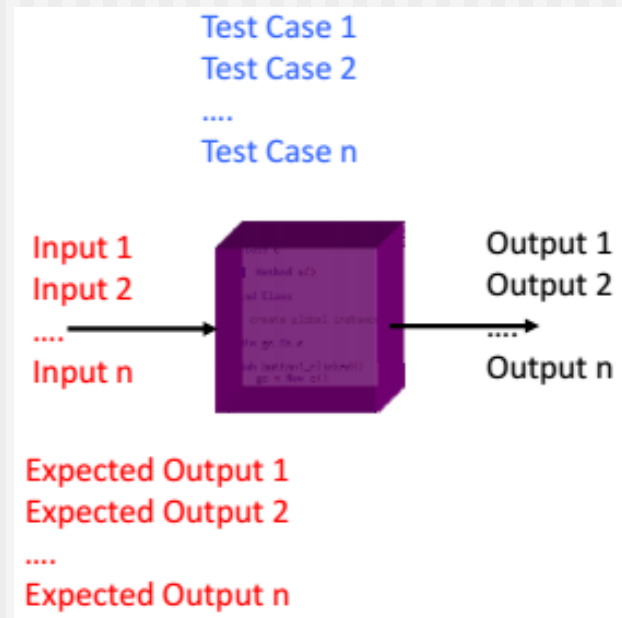
- ❑ Là kĩ thuật kiểm thử dựa trên thực thi code
- ❑ Tại sao phải sử dụng các kĩ thuật kiểm thử động?
 - Kiểm thử triệt để(Exhaustive testing) không thực tế
 - Chỉ sử dụng tập nhỏ nhất các test cases
 - Xác suất tìm ra lỗi lớn nhất
 - Cần chọn lựa các tests sử dụng các kĩ thuật thiết kế test cases

Các kĩ thuật kiểm thử động



Black-box testing

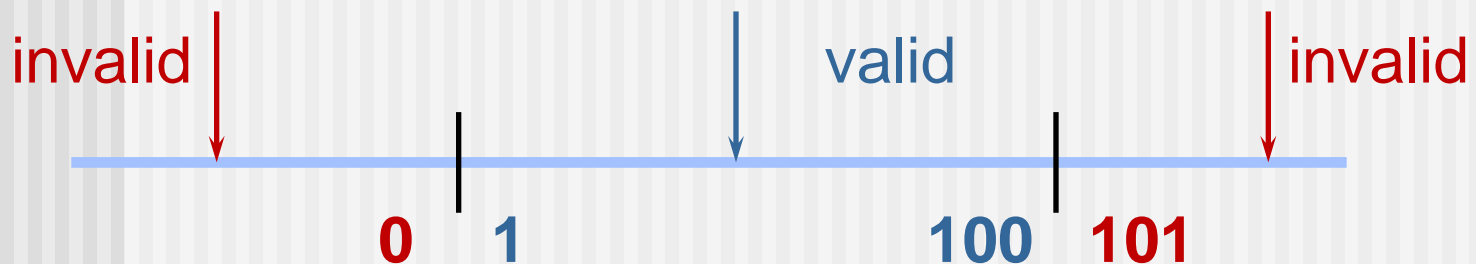
- ❑ Là phương pháp kiểm thử dựa trên hành vi hoặc chức năng của PM



Black-box testing

- ❑ Bao gồm các kĩ thuật:
 - Equivalence partitioning
 - Boundary value analysis
 - State transition testing
 - Decision table testing

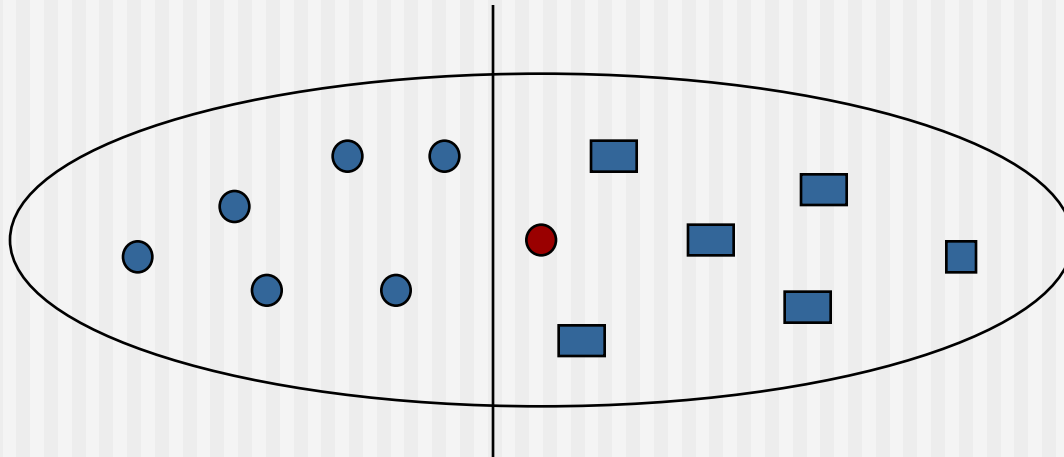
Phân hoạch lớp tương đương





- ❑ Phân chia tập hợp các điều kiện test (dữ liệu đầu vào) thành những vùng/lớp tương đương nhau.
- ❑ Lớp tương đương (equivalence partitions)
 - 2 tests thuộc cùng lớp tương đương nếu kết quả mong đợi của nó giống nhau
 - → thực hiện nhiều tests thuộc cùng 1 lớp tương đương thì dẫn đến dư thừa

Phân tích giá trị biên

- ❑ **Biên(boundary)** là điểm chuyển từ lớp tương đương này sang lớp tương đương khác
- ❑ Tìm ra được lỗi gần biên



Ví dụ:

Customer Name	<input type="text"/>	2-64 chars.
Account number	<input type="text"/>	6 digits, 1st non-zero
Loan amount requested	<input type="text"/>	£500 to £9000
 Term of loan	<input type="text"/>	1 to 30 years
 Monthly repayment	<input type="text"/>	Minimum £10

Term:

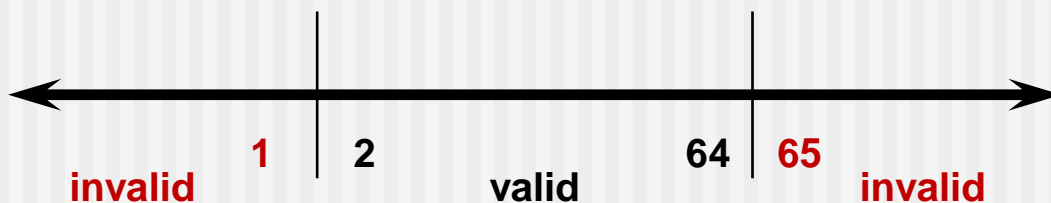
Repayment:

Interest rate:

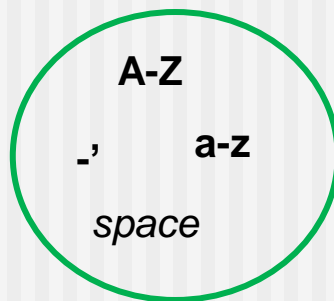
Total paid back:

Customer name

Số lượng kí tự:



Các kí tự hợp lệ:



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Customer name	2 to 64 chars valid chars	< 2 chars > 64 chars invalid chars	2 chars 64 chars	1 chars 65 chars 0 chars

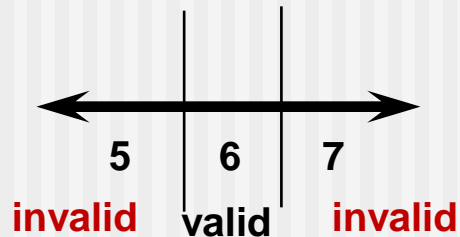
Account number

Chữ số đầu tiên:

valid: non-zero

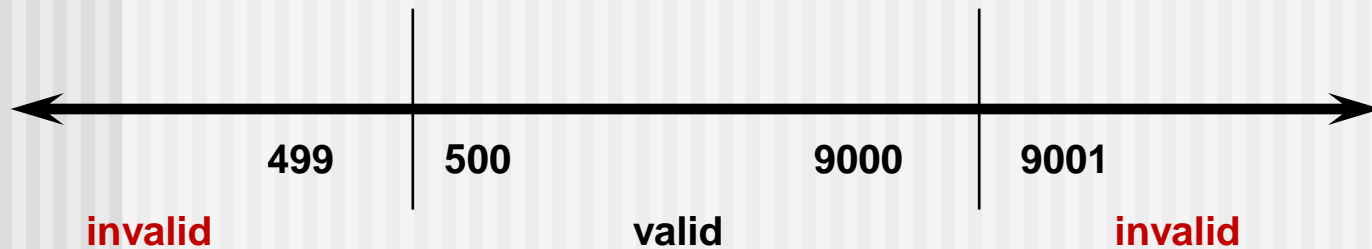
invalid: zero

Số lượng chữ số:



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Account number	6 digits 1 st non-zero	< 6 digits > 6 digits 1 st digit = 0 non-digit	100000 999999	5 digits 7 digits 0 digits

Loan amount



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Loan amount	500 - 9000	< 500 >9000 0 non-numeric null	500 9000	499 9001

Điều kiện test

Conditions	Valid Partitions	Tag	Invalid Partitions	Tag	Valid Boundaries	Tag	Invalid Boundaries	Tag
Customer name	2 - 64 chars valid chars	V1 V2	< 2 chars > 64 chars invalid char	X1 X2 X3	2 chars 64 chars	B1 B2	1 char 65 chars 0 chars	D1 D2 D3
Account number	6 digits 1 st non-zero	V3 V4	< 6 digits > 6 digits 1 st digit = 0 non-digit	X4 X5 X6 X7	100000 999999	B3 B4	5 digits 7 digits 0 digits	D4 D5 D6
Loan amount	500 - 9000	V5	< 500 >9000 0 non-integer null	X8 X9 X10 X11 X12	500 9000	B5 B6	499 9001	D7 D8

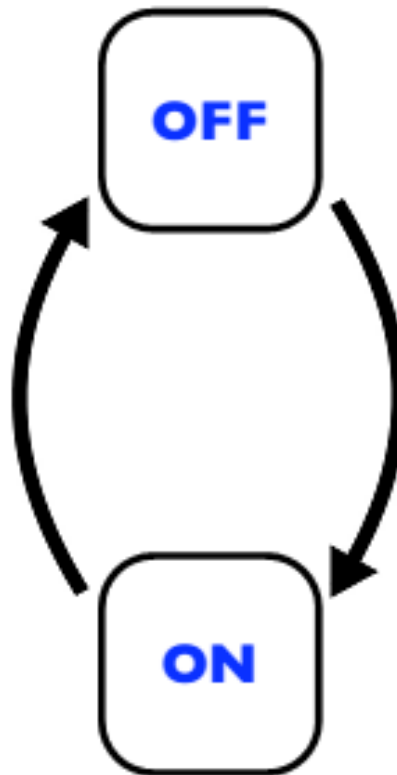
Thiết kế test cases

Test Case	Input	Expected Outcome		New Tags Covered
1	Name: John Smith Acc no: 123456 Loan: 2500 Term: 3 years	Term:	3 years	V1, V2, V3, V4, V5
		Repayment:	79.86	
		Interest rate:	10%	
		Total paid:	2874.96	
2	Name: AB Acc no: 100000 Loan: 500 Term: 1 year	Term:	1 year	B1, B3, B5,
		Repayment:	44.80	
		Interest rate:	7.5%	
		Total paid:	537.60	

Chuyển đổi trạng thái (state transition testing)

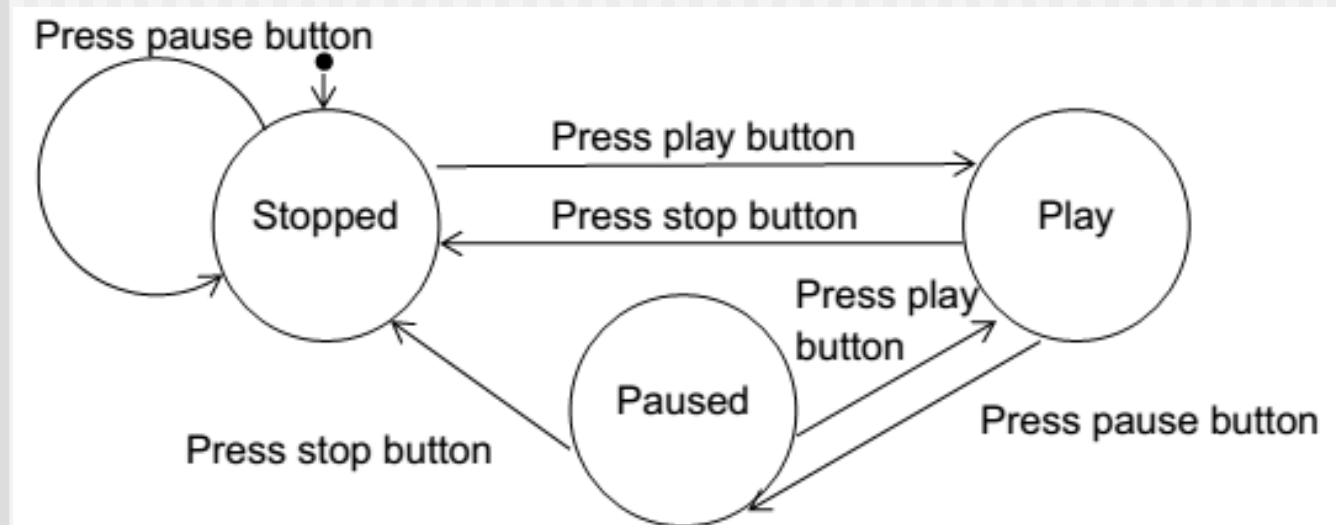
- Phân tích mối quan hệ giữa các trạng thái, sự kiện, hành động(action) gây ra sự chuyển đổi từ trạng thái này sang trạng thái khác
- **Các bước thực hiện**
 - Xác định tất cả các trạng thái
 - Với mỗi test, xác định
 - Start state
 - Input
 - Expected Output/End state

Ví dụ 1:



TEST 1	TEST 2
start state: off	start state: on
input: switch on	input: switch off
output: light on	output: light off
finish state: on	finish state: off

Ví dụ 2: Media Player



Ví dụ 2: Media Player

Test Case No.	Start State	Event/Input	End State/ Exp Output
1	Stopped	Press play button	Play
2	Stopped	Press pause button	Stopped
3	Play	Press stop button	Stopped
4	Play	Press pause button	Paused
5	Paused	Press stop button	Stopped
6	Paused	Press play button	Play

Bảng quyết định (Decision Table)

□ Gồm 3 phần:

■ Conditions

- Các điều kiện đầu vào để ra quyết định

■ Actions

- Kết quả của sự kết hợp các điều kiện đầu vào

■ Rules

- Luật kết hợp các điều kiện

Bảng quyết định (Decision Table)

❑ Các bước thực hiện tạo bảng:

1. Liệt kê tất cả các **conditions/inputs** và xác định giá trị của nó
2. Liệt kê tất cả các **actions**
3. Tính số **rules** lớn nhất
4. Xác định các **rules** từ sự tổ hợp giá trị của các **conditions/inputs**
5. Đánh dấu “X” vào mỗi **actions** tương ứng với mỗi **rule**
6. Rút gọn bảng (nếu có thể)

Ví dụ 1:

- Một ngân hàng sử dụng những luật sau để phân loại tài khoản mới của người gửi tiền:
 - Nếu tuổi ≥ 21 và số tiền gửi là ≥ 100 nghìn, thì loại tài khoản là A
 - Nếu tuổi < 21 và số tiền gửi là ≥ 100 nghìn, thì loại tài khoản là B
 - Nếu tuổi ≥ 21 và số tiền gửi là < 100 nghìn, thì loại tài khoản là C
 - Nếu tuổi < 21 và số tiền gửi là < 100 nghìn, thì không thể mở tài khoản

Ví dụ 1:

	Condition/Action	R1	R2	R3	R4
C1	Tuổi ≥ 21	Y	Y	N	N
C2	Tiền ≥ 100 nghìn	Y	N	Y	N
A1	A	X	-	-	-
A2	B	-		X	-
A3	C	-	X	-	-
A4	Ko thể mở TK	-	-	-	X

Ví dụ 2:

- ❑ Để tính thuế thu nhập, người ta có mô tả sau:

- *Người vô gia cư nộp 4% thuế thu nhập*
- *Người có nhà ở nộp thuế theo bảng sau:*

Tổng thu nhập	Thuế
$\leq 5.000.000$ đồng	4%
$> 5.000.000$ đồng	6%

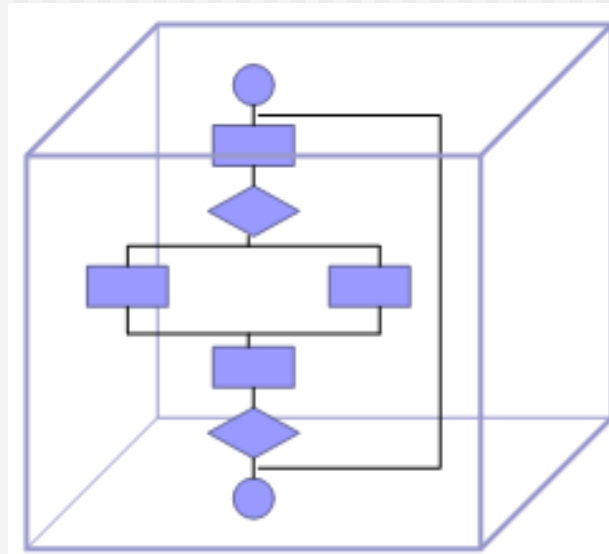
Ví dụ 2:

	Condition/Action	R1	R2	R3	R4
C1	Người có nhà ở	Y	Y	N	N
C2	Tổng thu nhập ≤ 500000	Y	N	Y	N
A1	Nộp thuế 4%	X	-	X	X
A2	Nộp thuế 6%	-	X	-	-

	Condition/Action	R1	R2	R3
C1	Người có nhà ở	Y	Y	N
C2	Tổng thu nhập ≤ 500000	Y	N	-
A1	Nộp thuế 4%	X	-	X
A2	Nộp thuế 6%	-	X	-

White-box testing

- ❑ Là phương pháp kiểm thử dựa trên cấu trúc logic của PM



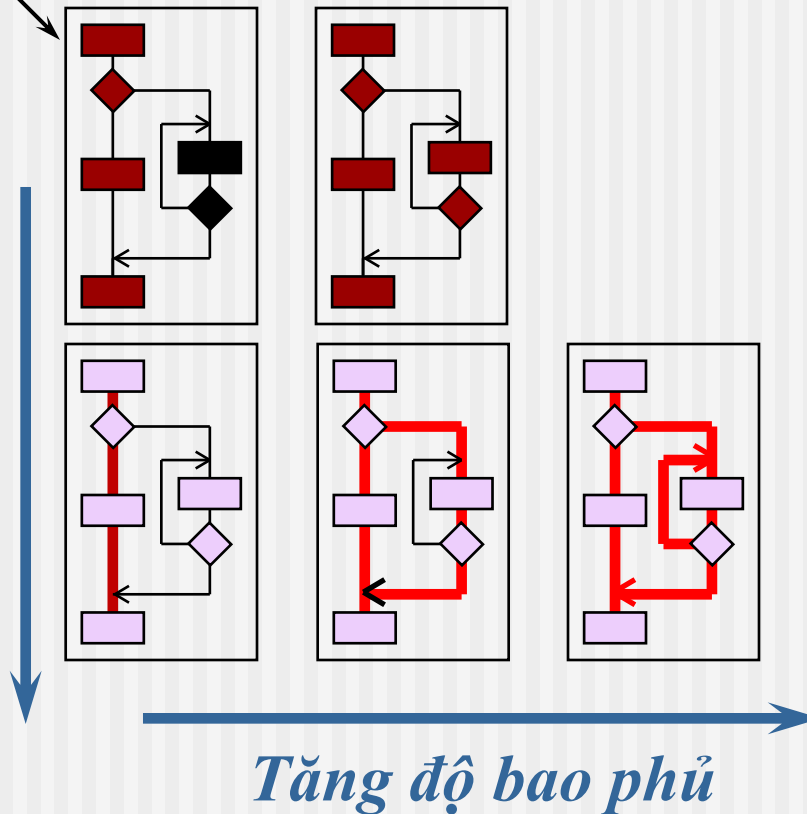
White-box testing

- ❑ Control Flow Testing
 - Statement testing
 - Branch / Decision testing
 - Branch condition testing
 - Branch condition combination testing
- ❑ Data flow testing

Kĩ thuật độ bao phủ cấu trúc (Coverage techniques)

Cái gì
được
bao
phủ?

*Các thành phần cấu
trúc khác nhau*



Kĩ thuật độ bao phủ cấu trúc (Coverage techniques)

- ❑ Đánh giá độ bao phủ của các thành phần cấu trúc
- ❑ Thiết kế các test bổ sung → có thể đạt được mức bao phủ 100%

*Độ bao phủ 100%
không có nghĩa là
100% được test*

Kiểm thử luồng điều khiển (Control Flow Testing)

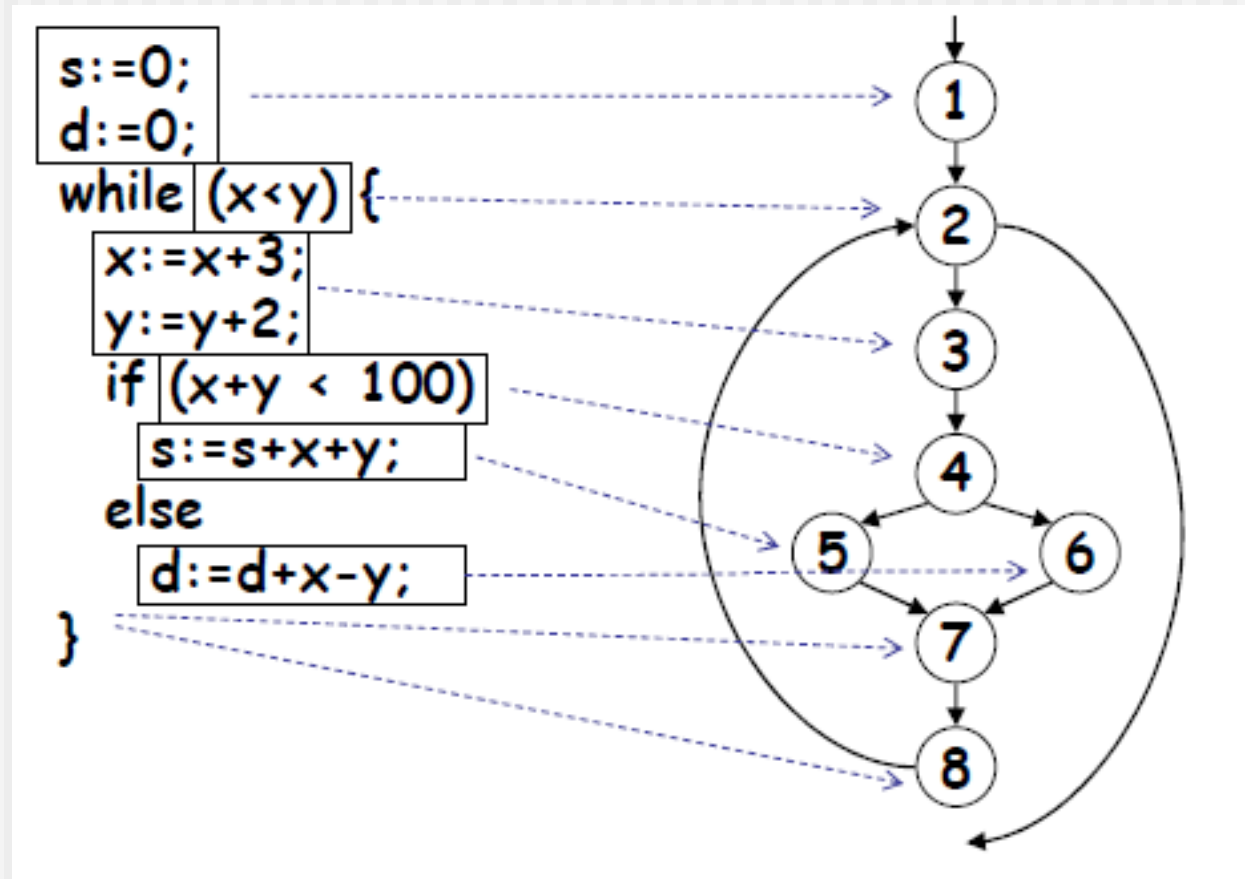
❑ Bước 1:

- Xây dựng đồ thị luồng từ source code
- Đồ thị bao gồm nút, cạnh

❑ Bước 2:

- Thiết kế các Test cases để bao phủ hết các thành phần của đồ thị

Ví dụ 1:



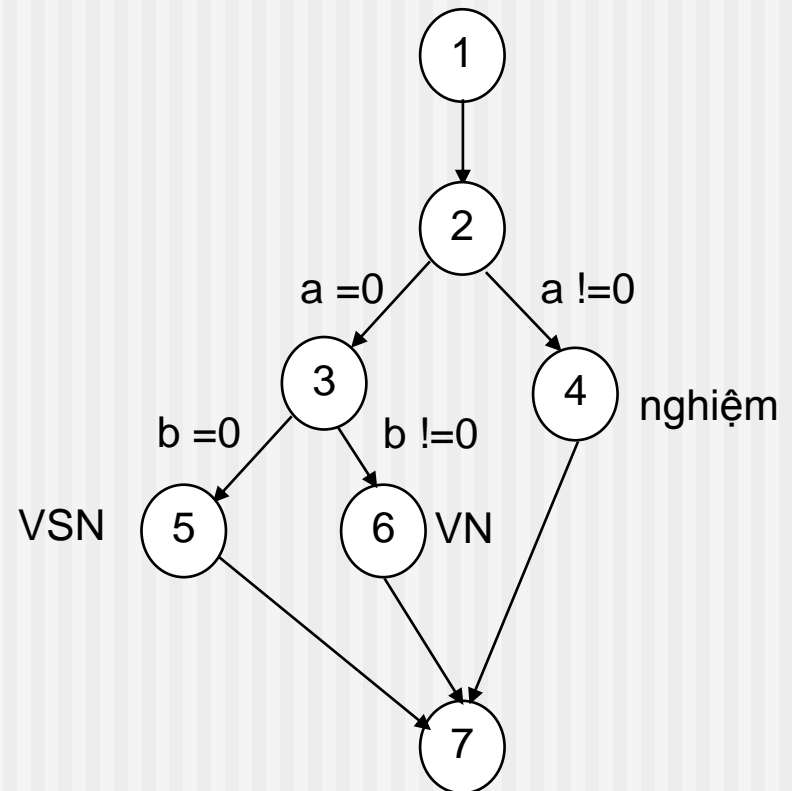
Thành phần của CFG

- ❑ Có 3 loại nút
 - Statement nodes
 - Decision nodes
 - Auxiliary nodes

Ví dụ 2:

Đồ thị luồng điều khiển của đoạn chương trình giải phương trình bậc nhất

1. if(a == 0)
2. if(b == 0)
3. cout<<"Vo so nghiem";
4. else
5. cout<<"Vo nghiem";
6. else
7. cout<<"x = "<<-b/a;



Độ phức tạp “Cyclomatic”

- $V(G) = E - N + 2$
- $V(G) = P + 1$
- $V(G) = R$

Trong đó:

- E: số cạnh
- N: số nút
- P: số nút điều kiện
- R: số miền

Vẽ đồ thị luồng

```
1. void prime(int n){
2.     cout<<"Cac so nguyen to nho hon n la:"<<endl;
3.     for(int i = 2; i < n;i++){
4.         int flag = 1;
5.         for(int j = 2; j <= sqrt(i);j++)
6.             if(i%j==0)
7.                 { flag = 0; break;}
8.         if(flag == 1){
9.             cout<<i<<"\t";
10.        }
11.    }
12. }
```

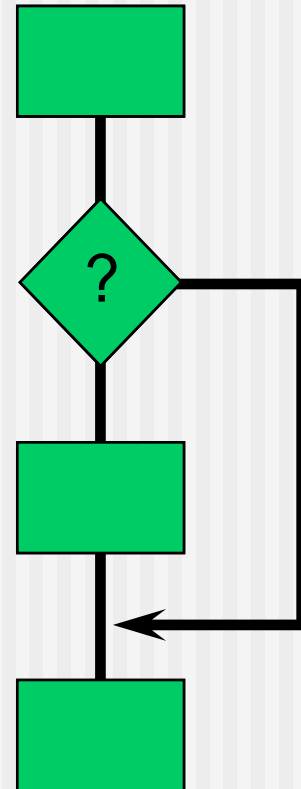
Bao phủ câu lệnh (Statement coverage)

- ❑ Tỷ lệ phần trăm các câu lệnh được thực thi bởi bộ test

$$= \frac{\text{Số câu lệnh thực thi}}{\text{Tổng số câu lệnh}}$$

- ❑ Ví dụ:

- ❑ Chương trình có 100 câu lệnh
- ❑ tests thực thi 87 câu lệnh
- ❑ Bao phủ câu lệnh = 87%



Ví dụ 1:

1	<code>cin>>a;</code>
2	<code>if (a > 6)</code>
3	<code> b = a;</code>
4	<code>cout<<b;</code>

Số câu lệnh

Test case	Input	Expected output
1	7	7

Với 1 test case này thì tất cả 4 câu lệnh đều được thực thi
→ đạt độ bao phủ dòng lệnh 100%

Bài tập 1

Đoạn chương trình giải phương trình bậc nhất $ax+b=0$

```
1.  {  
2.  if(a == 0)  
3.      if(b == 0)  
4.          cout<<"Vo so nghiem";  
5.      else  
6.          cout<<"Vo nghiem";  
7.  else  
8.      cout<<"x ="<<-b/a;  
9.  }
```

1. Có bao nhiêu câu lệnh?
2. Test case ($a=0, b=8$) bao phủ bao nhiêu % câu lệnh ?
3. Cần tối thiểu bao nhiêu test case để bao phủ 100% các câu lệnh ?

Bài tập 2

```
1. void func(int A, int B, int X )
2. {
3.     if (A > 1 && B == 0 )
4.         X = X / A;
5.     if ( A == 2 || X > 1)
6.         X = X + 1;
7. }
```

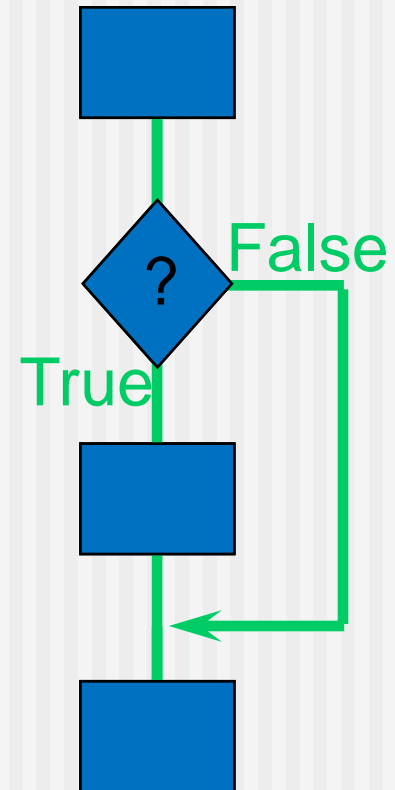
-
1. Có bao nhiêu câu lệnh ?
 2. Test case (A=2,B=1,X=3) bao phủ bao nhiêu % câu lệnh ?
 3. Test case (A=3,B=0,X=0) bao phủ bao nhiêu % câu lệnh ?
 4. Tối thiểu bao nhiêu test case để bao phủ 100% các câu lệnh ?

Bao phủ nhánh (Decision/Branch coverage)

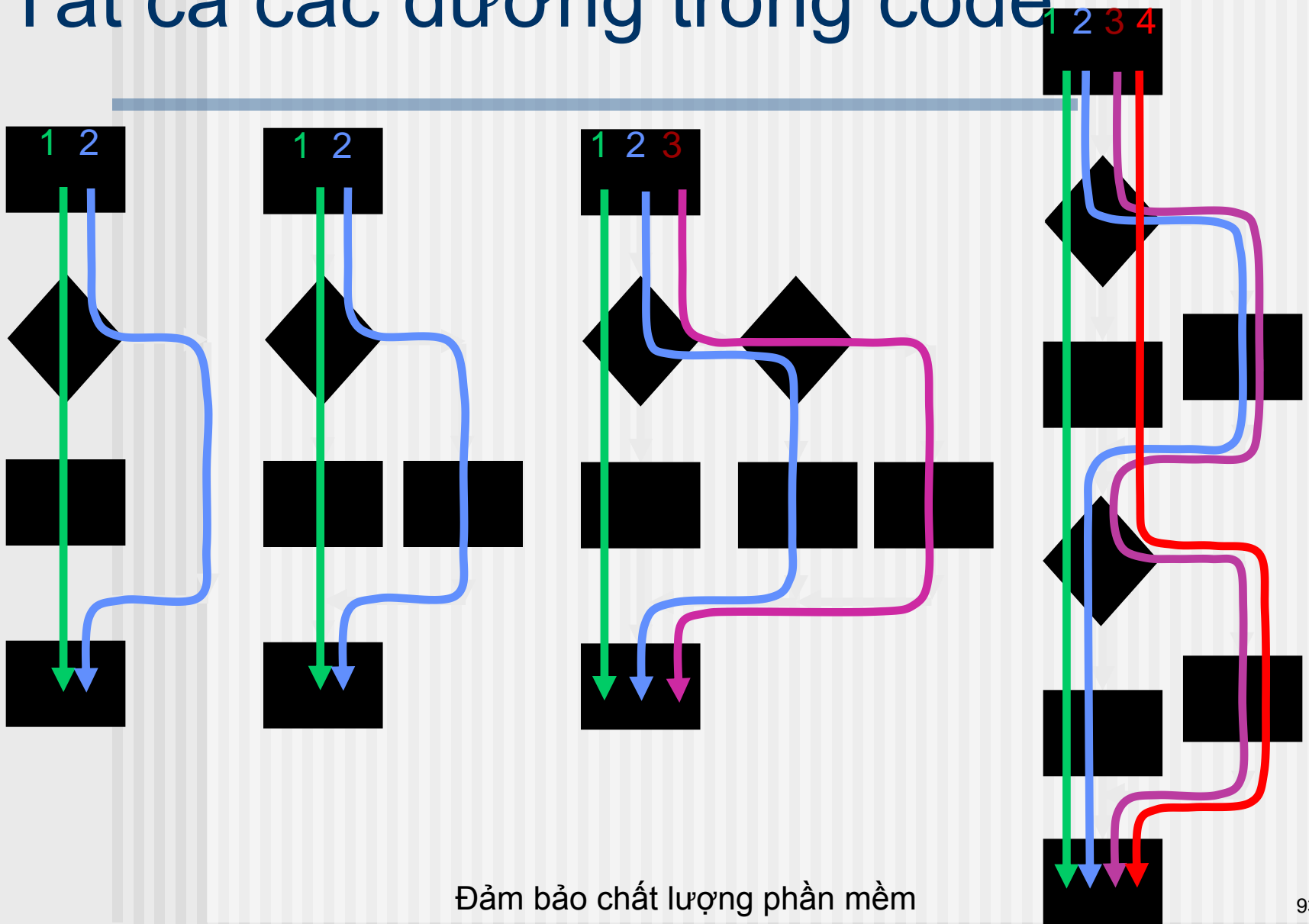
- ❑ Tỷ lệ phần trăm các nhánh được thực thi bởi bộ test

$$= \frac{\text{Số nhánh được thực thi}}{\text{Tổng số nhánh}}$$

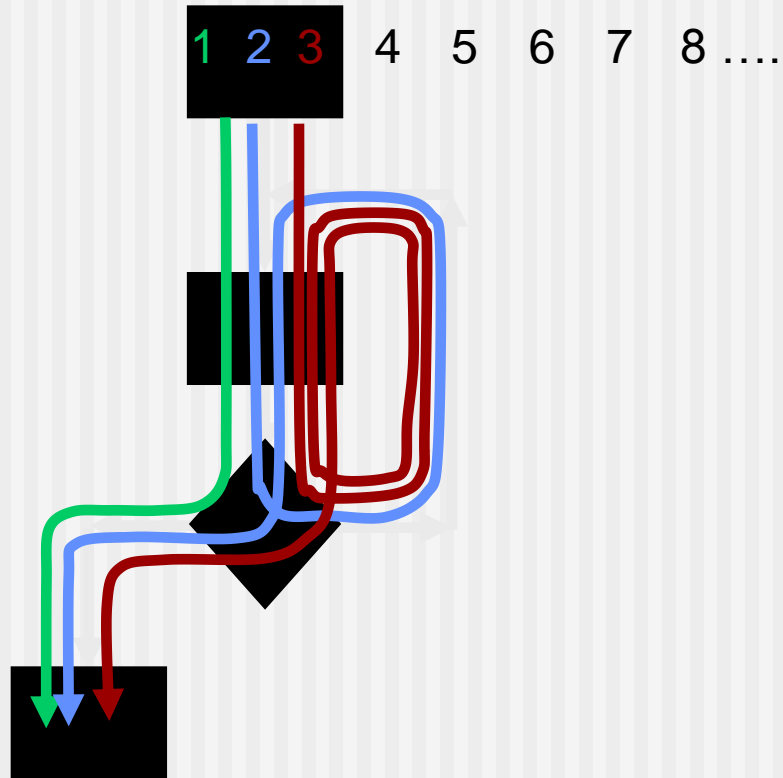
- ❑ Ví dụ:
 - Chương trình có 120 nhánh
 - tests thực thi 60 nhánh
 - Bao phủ nhánh = 50%



Tất cả các đường trong code



Các đường trong vòng lặp



Ví dụ 1:

Đoạn chương trình giải phương trình bậc nhất $ax+b=0$

```
1.  {  
2.    if(a == 0)  
3.        if(b == 0)  
4.            cout<<"Vo so nghiem";  
5.        else  
6.            cout<<"Vo nghiem";  
7.    else  
8.        cout<<"x =",-b/a;  
9.  }
```

1. Có bao nhiêu nhánh ?
2. Test case $(a=0,b=8), (a=0,b=0)$ bao phủ bao nhiêu % nhánh ?
3. Tối thiểu bao nhiêu test case để bao phủ 100% các nhánh?

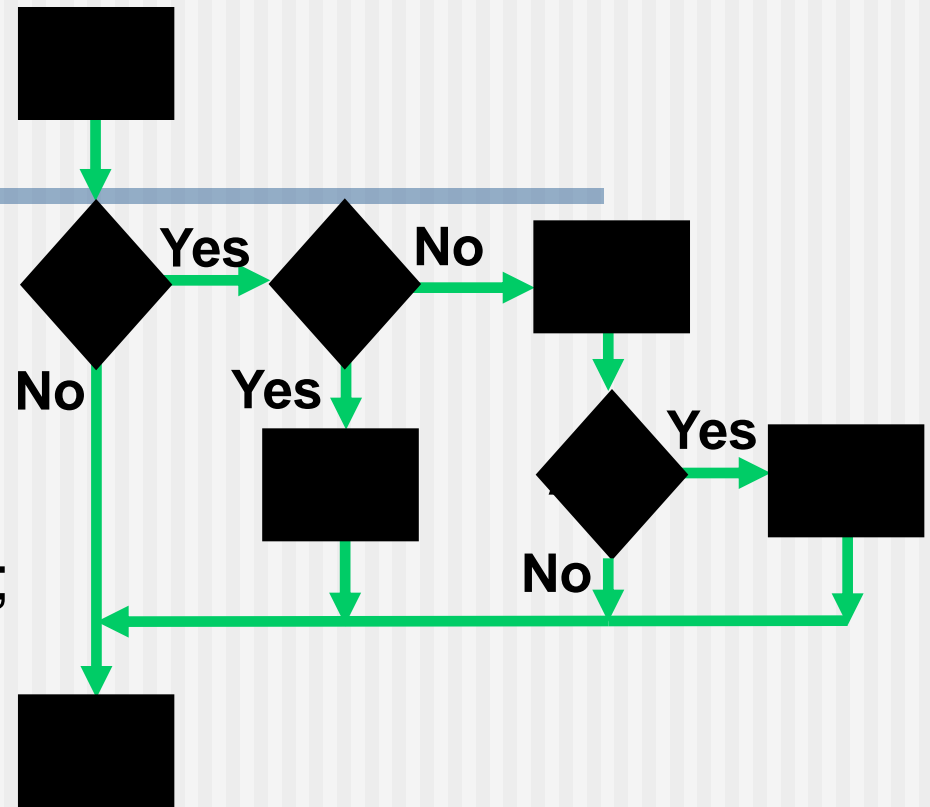
Ví dụ 2:

```
void func(int A, int B, int X )  
1.  {  
2.    if ( A > 1 && B == 0 )  
3.        X = X / A;  
4.    if ( A == 2 || X > 1)  
5.        X = X + 1;  
6. }
```

1. Có bao nhiêu nhánh?
2. Test case (A=2,B=0,X=3) bao phủ bao nhiêu % nhánh ?
3. Tối thiểu bao nhiêu test case để bao phủ 100% các nhánh ?

Ví dụ

```
1. cin>>A;
2. cin>> B;
3. if (A > 0)
4.     if (B == 0)
5.         cout<< "No values";
6.     else
7.     {
8.         cout<<"B"<<endl;
9.         if (A > 21)
10.            cout<<"A";
11.     }
```



- Độ phức tạp Cyclomatic : 4
- Số tests nhỏ nhất:
 - Statement coverage: 2
 - Branch coverage: 4

Bao phủ điều kiện (Condition coverage)

□ Tỷ lệ phần trăm các điều kiện được thực thi bởi bộ test

$$= \frac{\text{Số điều kiện được thực thi}}{\text{Tổng số điều kiện}}$$

Ví dụ: Bao phủ điều kiện

- Cần thiết kế các test cases bao phủ hết các điều kiện sau:

$A > 1$, $A \leq 1$, $B = 0$, $B \neq 0$

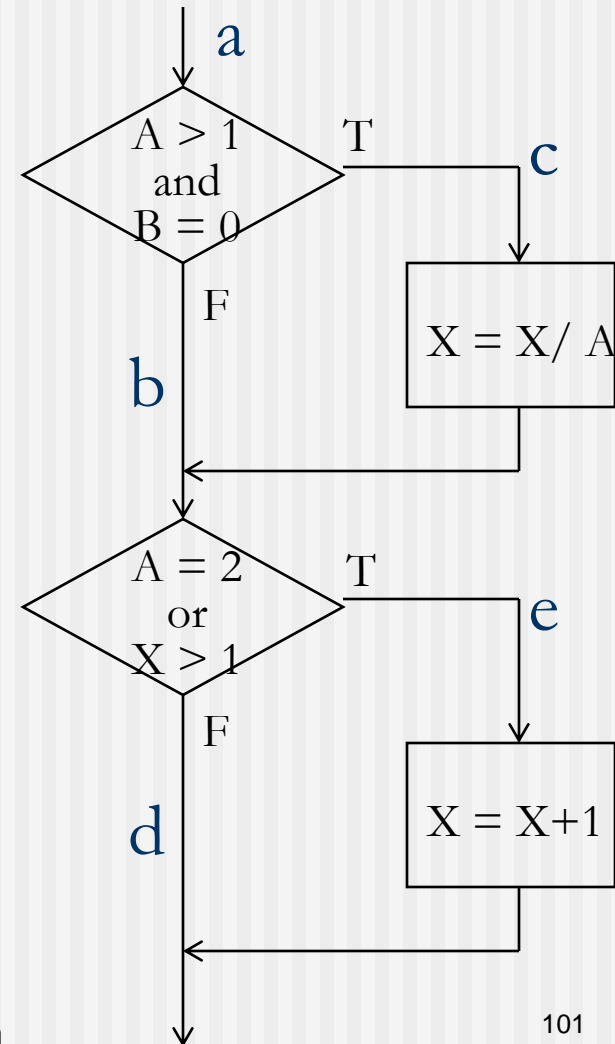
$A = 2$, $A \neq 2$, $X > 1$, $X \leq 1$

- Test cases:

1) $A = 1$, $B = 0$, $X = 3$ (abe)

2) $A = 2$, $B = 1$, $X = 1$ (abe)

- Không thỏa mãn bao phủ nhánh



Bài tập 1

```
1. float A, B, C;
2. printf("Enter three values\n");
3. scanf("%f%f%f", &A, &B, &C);
4. printf("\n largest value is: ");
5. if( A>B)
6. {
7.     if (A>C)  printf("%f\n",A);
8.     else    printf("%f\n",C);
9. }
10. else
11. {
12.     if (C>B)  printf("%f\n",C);
13.     else    printf("%f\n",B);
14. }
```

Bài tập 2

```
1. void PTA(int a[], int n)
2. {
3.     int i = 0;
4.     while ((i < n) && (a[i] >= 0))
5.         i++;
6.     if (i < n)
7.         printf("Phan tu am a[%d] = %d", i, a[i]);
8.     else
9.         printf("Khong co phan tu am.");
10. }
```

Bài tập 3

```
1.  const int SCALENE = 1;
2.  const int ISOSCELES = 2;
3.  const int EQUILATERAL = 3;
4.  const int ERROR = 4;
5.  int TriangleType(int x, int y, int z){
6.      if (x<=0 || y<=0 || z<=0)
7.          return ERROR;
8.      if ((x + y <= z) || (x + z <= y) || (z + y <= x))
9.          return ERROR;
10.     if (x == y && y == z)
11.         return EQUILATERAL;
12.     if (x == y || y == z || z == x)
13.         return ISOSCELES;
14.     return SCALENE;
15. }
```

Bao phủ nhánh – điều kiện

□ Bao phủ nhánh - điều kiện

- Viết các TCs sao cho mỗi điều kiện (đơn) trong mỗi nhánh nhận được 2 giá trị(T và F) ít nhất 1 lần và mỗi nhánh được thực hiện ít nhất 1 lần

□ Bao phủ tổ hợp các điều kiện

- Viết các TCs để thực thi được tất cả các tổ hợp giá trị(T và F) của các điều kiện (đơn) trong 1 nhánh

Ví dụ: Bao phủ nhánh-điều kiện

- ❑ Các TCs cần bao phủ tất các điều kiện

$A > 1, A \leq 1, B = 0, B \neq 0$

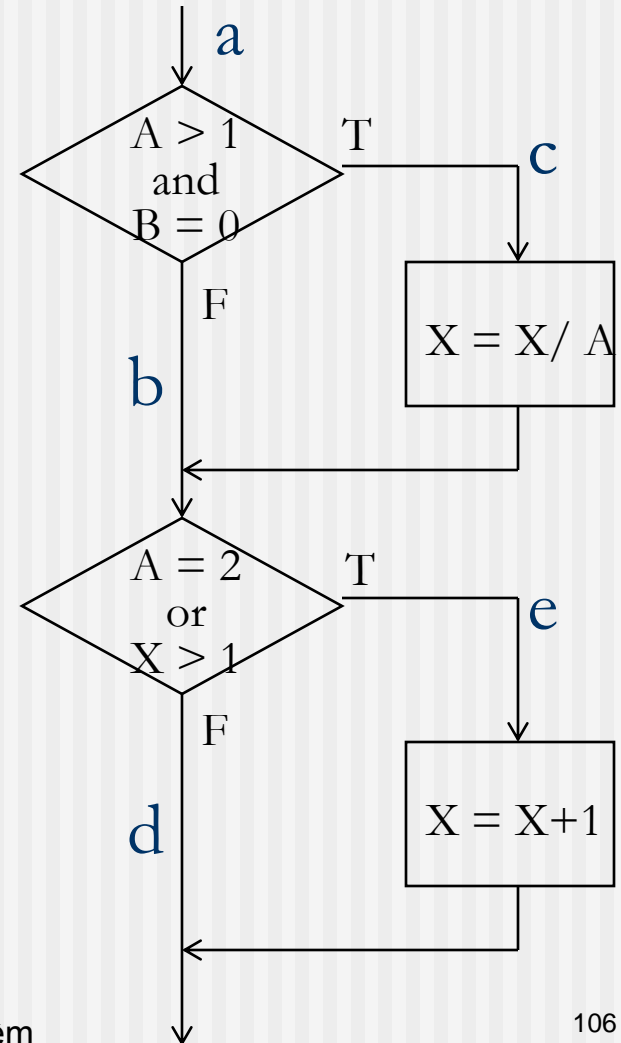
$A = 2, A \neq 2, X > 1, X \leq 1$

và $(A > 1 \text{ and } B = 0) \text{ T, F}$
 $(A = 2 \text{ or } X > 1) \text{ T, F}$

- ❑ Test cases:

1) $A = 2, B = 0, X = 4$ (ace)

2) $A = 1, B = 1, X = 1$ (abd)



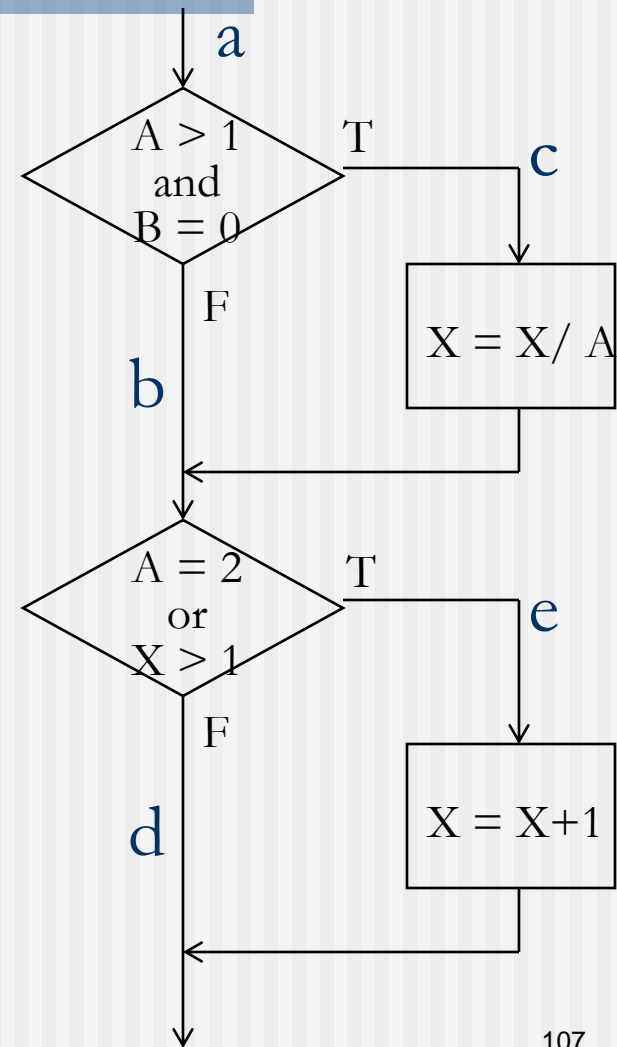
Ví dụ: Bao phủ tổ hợp điều kiện

❑ TCs phải bao phủ các điều kiện

- | | |
|-------------------------|-------------------------|
| 1) $A > 1, B = 0$ | 5) $A = 2, X > 1$ |
| 2) $A > 1, B \neq 0$ | 6) $A = 2, X \leq 1$ |
| 3) $A \leq 1, B = 0$ | 7) $A \neq 2, X > 1$ |
| 4) $A \leq 1, B \neq 0$ | 8) $A \neq 2, X \leq 1$ |

❑ Test cases:

- | | |
|--------------------------|---------------|
| 1) $A = 2, B = 0, X = 4$ | (bao phủ 1,5) |
| 2) $A = 2, B = 1, X = 1$ | (bao phủ 2,6) |
| 3) $A = 1, B = 0, X = 2$ | (bao phủ 3,7) |
| 4) $A = 1, B = 1, X = 1$ | (bao phủ 4,8) |



Kĩ thuật kiểm thử dựa trên kinh nghiệm

- ❑ Error guessing
- ❑ Exploratory testing

Đoán lỗi(Error guessing)

- ❑ Được sử dụng sau khi áp dụng các kĩ thuật hình thức khác
- ❑ Có thể tìm ra 1 số lỗi mà các kĩ thuật khác có thể bỏ lỡ
- ❑ Không có qui tắc chung

Kiểm thử thăm dò (Exploratory testing)

- ❑ Hoạt động thiết kế test và thực thi test được thực hiện song song
- ❑ Các chú ý sẽ được ghi chép lại để báo cáo sau này
- ❑ Khía cạnh chủ chốt: learning(cách sử dụng, điểm mạnh, điểm yếu của PM)

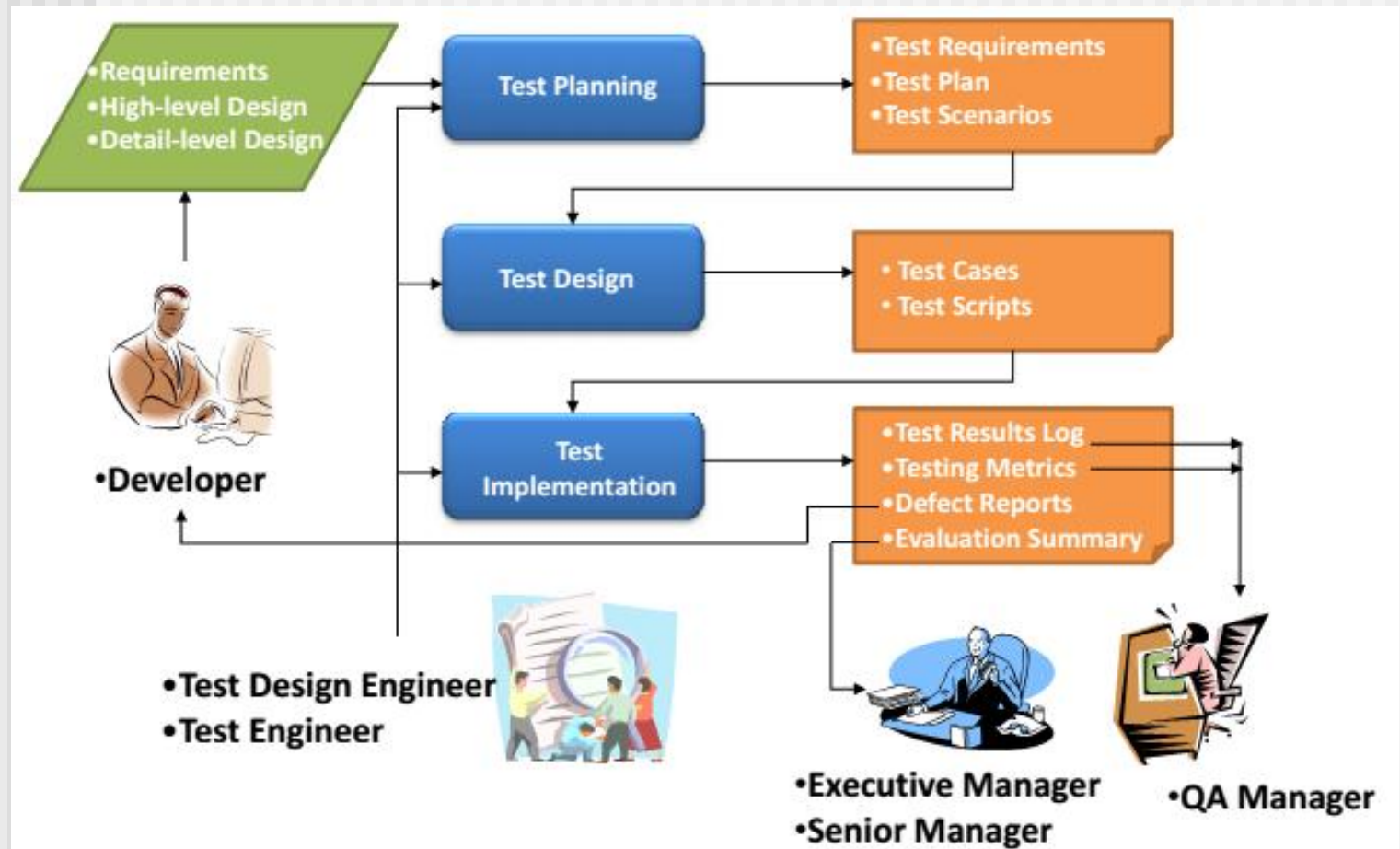
Chọn kĩ thuật kiểm thử

- ❑ Mỗi kĩ thuật riêng lẻ chỉ hiệu quả với 1 nhóm lỗi cụ thể
- ❑ Các yếu tố ảnh hưởng đến việc chọn kĩ thuật kiểm thử:
 - Loại hệ thống
 - Tiêu chuẩn quy định
 - Yêu cầu khách hàng hoặc hợp đồng
 - Mức độ rủi ro, loại rủi ro
 - Mục tiêu kiểm thử
 - Tài liệu có sẵn

Chọn kĩ thuật kiểm thử

- Kiến thức của testers
- Thời gian và ngân sách
- Mô hình phát triển PM
- Kinh nghiệm về các loại lỗi đã được tìm ra trước đó

D. Quy trình kiểm thử



Quy trình kiểm thử

- ❑ Bao gồm các hoạt động:
 1. Lập kế hoạch
 2. Phân tích xác định các điều kiện, thiết kế TCs
 3. Thực thi tests
 4. Đánh giá các tiêu chuẩn và báo cáo
 5. Kiểm tra hoàn thành tests

Quy trình kiểm thử

Bước 1: Vạch kế hoạch

- Thiết lập chiến lược test
- Xác định những người tham gia: testers, QA, development, support,...
- Xác định các yêu cầu, đặc tả về chức năng
- Chuẩn bị cơ sở hạ tầng cho test

Quy trình kiểm thử

Bước 2: Phân tích, thiết kế test

- Xác định điều kiện test, độ ưu tiên
- Thiết kế các Test cases
- Chuẩn bị dữ liệu kịch bản

Quy trình kiểm thử

Bước 3: Thực thi tests

- Thực thi các TCs đã được thiết kế:
 - Thực hiện trước các TCs quan trọng nhất
 - Có thể thực thi bằng tay hoặc tự động
 - Không thực thi tất cả các TCs nếu:
 - Có quá nhiều lỗi ở các TCs trước
 - Áp lực thời gian

Quy trình kiểm thử

Bước 4: Đánh giá và báo cáo

- Ghi lại các phiên bản của PM trong khi test
- Với mỗi TCS, ghi lại kết quả thực tế, độ bao phủ test
- So sánh kết quả thực tế với kết quả mong đợi
- Sau khi lỗi được fix thì cần phải test lại
- Viết báo cáo trạng thái kiểm thử

Quy trình kiểm thử

Bước 5: Kiểm tra hoàn thành tests

- Tiêu chuẩn kết thúc test đã được đưa vào test plan
- Nếu các tiêu chuẩn chưa được đáp ứng thì thực hiện lại từ việc thiết kế thêm các TCs
- Các tiêu chuẩn có thể ứng dụng với tất cả các mức test

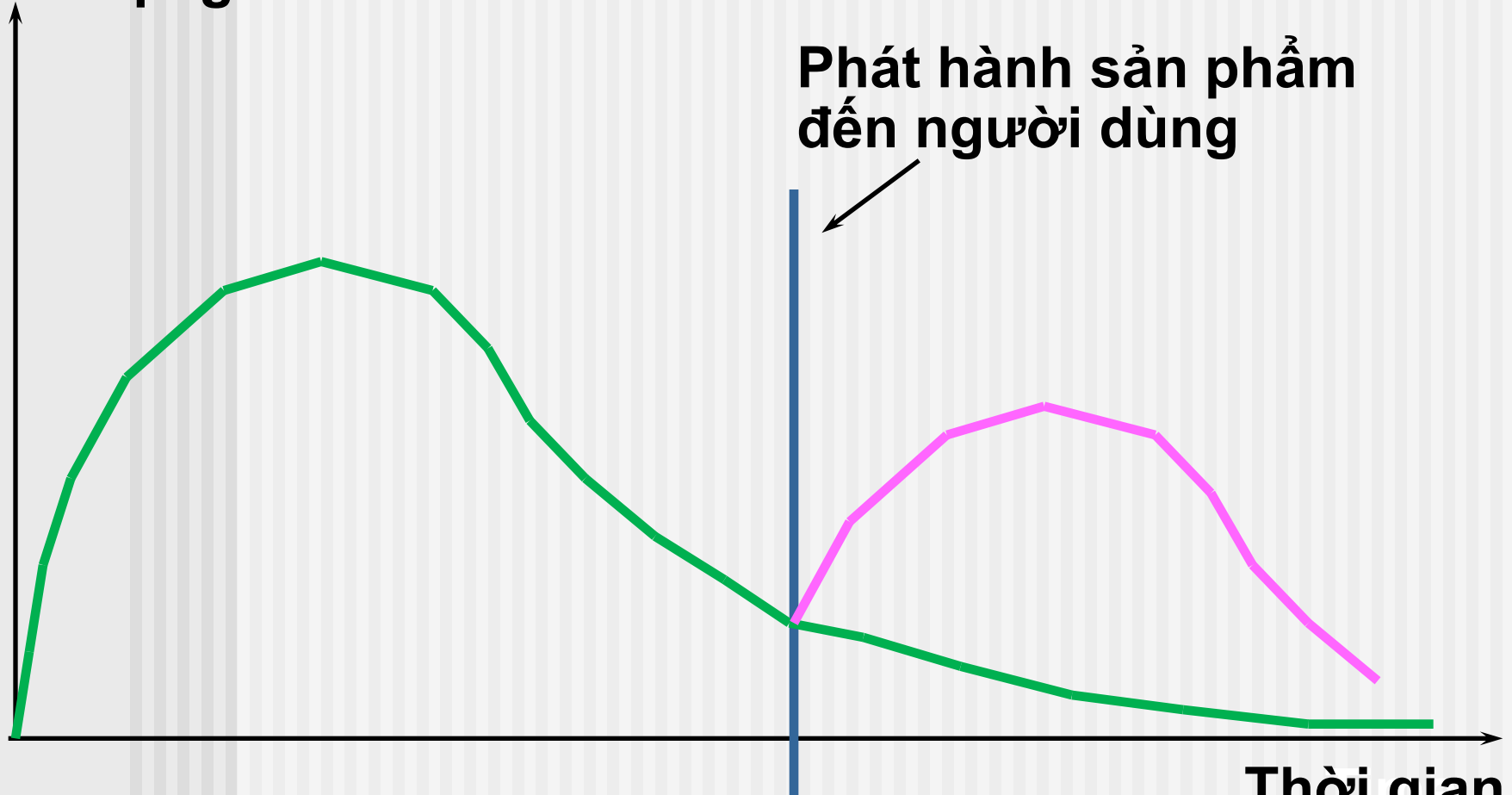
D. Tổ chức kiểm thử

- ❑ Tầm quan trọng của tính độc lập trong kiểm thử
- ❑ Cấu trúc tổ chức kiểm thử

Tầm quan trọng của tính độc lập trong kiểm thử

Số lượng lỗi

Phát hành sản phẩm đến người dùng



Đảm bảo chất lượng phần mềm

Thời gian

Cấu trúc tổ chức kiểm thử

1. Trách nhiệm của mỗi developer
2. Trách nhiệm của cả nhóm phát triển(Development team)
3. Tester nằm trong nhóm phát triển
4. Nhóm testers riêng biệt
5. Cố vấn cho kiểm thử
6. Nhóm testers thuộc công ty thứ 3

Mỗi developer thực hiện kiểm thử

❖ Ưu điểm

- Hiểu rõ code
- Tìm được các lỗi kĩ thuật mà tester có thể bỏ lỡ
- Tìm lỗi và sửa lỗi ít tốn ngân sách

❖ Nhược điểm

- Đánh giá chủ quan
- Khó phá hủy công việc của chính họ
- Xu hướng thấy được kết quả mong đợi hơn là kết quả thực tế

Nhóm phát triển thực hiện kiểm thử

❑ Ưu điểm

- Có sự độc lập nhất định
- Tìm được các lỗi kỹ thuật mà tester có thể bỏ lỡ
- Thân thiện, dễ dàng giao tiếp

❑ Nhược điểm

- Thiếu kỹ năng test
- Bị áp lực công việc

Tester nằm trong nhóm phát triển

❑ Ưu điểm

- Tester chỉ có trách nhiệm kiểm thử
- Độc lập trong kiểm thử
- Tester là thành viên của nhóm → có chung 1 mục đích

❑ Nhược điểm

- Ít được tôn trọng
- Làm việc 1 mình
- Quan điểm đưa ra đơn lẻ

Đội ngũ tester độc lập

❑ Ưu điểm

- Gồm những tester có kinh nghiệm, chỉ làm nhiệm vụ kiểm thử
- Kiểm thử khách quan hơn và nhất quán hơn

❑ Nhược điểm

- Phụ thuộc vào đội ngũ tester → test thiếu
- Cô lập với nhóm phát triển

Chuyên gia tư vấn kiểm thử nội bộ

❑ Ưu điểm

- Gồm các chuyên gia trong lĩnh vực kiểm thử → cung cấp, hỗ trợ giúp cải tiến quá trình kiểm thử
- Vạch kế hoạch, ước lượng, điều khiển quá trình kiểm thử tốt hơn

❑ Nhược điểm

- Mức chuyên gia đã đủ chưa??
- Lời khuyên, hướng dẫn chỉ ảnh hưởng đến thử tục test, chứ không quyết định hoàn toàn
- Một số chuyên gia cũng phải thực thi test

Nhóm testers thuộc công ty thứ 3

❑ Ưu điểm

- Gồm các chuyên gia trong lĩnh vực kiểm thử
- Độc lập với các chính sách nội bộ

❑ Nhược điểm

- Chi phí đắt
- Thiếu kiến thức về sản phẩm

Những kĩ năng cần thiết trong đội ngũ kiểm thử

- ❑ Chuyên gia kĩ thuật
- ❑ Chuyên gia về kiểm thử tự động
- ❑ Chuyên gia database
- ❑ Kĩ năng business
- ❑ Hiểu biết tốt về tính khả dụng
- ❑ Chuyên gia về môi trường test
- ❑ Người quản lí test

Vai trò của Test leader

- ❑ Lập kế hoạch, giám sát và kiểm soát các hoạt động kiểm thử
- ❑ Lập các mục tiêu, tổ chức chính sách, chiến lược kiểm thử
- ❑ Ước tính test cần thực hiện và đàm phán với quản lý để có được các nguồn lực cần thiết
- ❑ Nhận biết thời điểm cần test tự động, chọn lựa tools, tổ chức training
- ❑ Tham khảo ý kiến với nhóm khác

Vai trò của Test leader

- Hướng dẫn, giám sát việc phân tích, thiết kế, thực thi các TCs, thủ tục kiểm thử, kịch bản kiểm thử
- Đảm bảo quản lí cấu hình testware
- Đảm bảo thực hiện môi trường test
- Lập lịch thực hiện kiểm thử
- Theo dõi, đo lường, kiểm soát và báo cáo về tiến độ kiểm thử, tình trạng chất lượng sản phẩm và kết quả kiểm thử
- Viết báo cáo tóm tắt về trạng thái test

Vai trò của tester

- ❑ Rà soát và góp ý cho kế hoạch kiểm thử
- ❑ Phân tích, đánh giá các yêu cầu của người dùng, các đặc tả thiết kế.
- ❑ Xác định điều kiện test, thiết kế TCs, thủ tục test, hỗ trợ test tự động
- ❑ Cài đặt môi trường test

Vai trò của tester

- ❑ Thực hiện kiểm thử trên tất cả các mức test
- ❑ Thực thi, ghi nhận, đánh giá kết quả test, lập tài liệu báo cáo lỗi
- ❑ Rà soát đặc tả test, các báo cáo lỗi, kết quả test.
- ❑ Theo dõi quá trình kiểm thử, môi trường test sử dụng các tools