

Chương 6. Kiểm chứng Phần mềm (Software Testing)



GVLT: Nguyễn Thị Thanh Trúc
ntruchcm@gmail.com



Nội dung

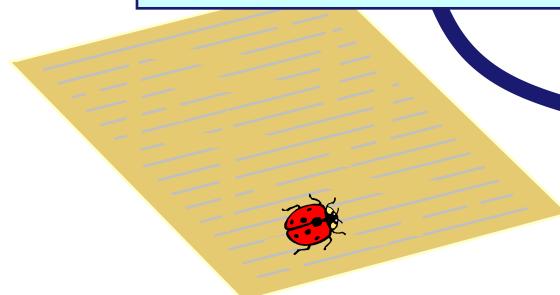
- ❖ Giới thiệu
- ❖ Khái niệm kiểm thử phần mềm
- ❖ Tại sao phải kiểm thử phần mềm
- ❖ Các nguyên lý trong kiểm thử phần mềm
- ❖ Các mức độ kiểm thử
- ❖ Các kỹ thuật kiểm thử
 - Kiểm thử hộp đen
 - Kiểm thử hộp trắng

Giới thiệu

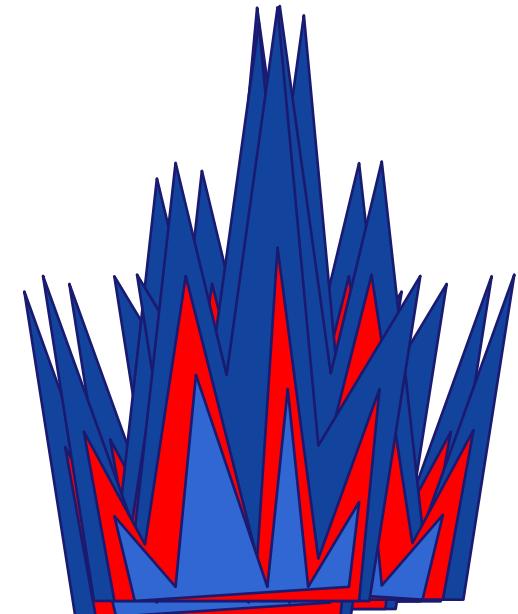
A person makes
an error ...



... that creates
a fault (bug,
defect) in the
software ...



... that can
cause a failure
in operation





Khái niệm kiểm thử phần mềm

- ❖ Kiểm thử phần mềm là quá trình thực thi phần mềm với mục tiêu tìm ra lỗi

Glen Myers, 1979

→ Khẳng định được chất lượng của phần mềm đang xây dựng

Hetzell, 1988



Một số đặc điểm kiểm thử PM

- ❖ Kiểm thử phần mềm giúp tìm ra được sự hiện diện của lỗi nhưng không thể chỉ ra sự vắng mặt của lỗi

Dijkstra

- ❖ Mọi phương pháp được dùng để ngăn ngừa hoặc tìm ra lỗi đều sót lại những lỗi khó phát hiện hơn

Beizer

- ❖ Điều gì xảy ra nếu việc kiểm thử không tìm được lỗi trong phần mềm hoặc phát hiện quá ít lỗi

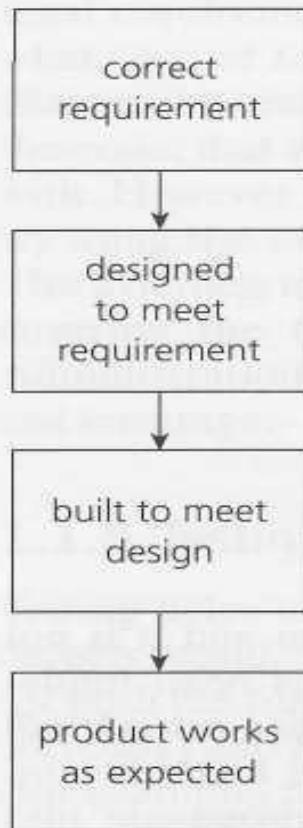


Tại sao kiểm thử lại cần thiết?

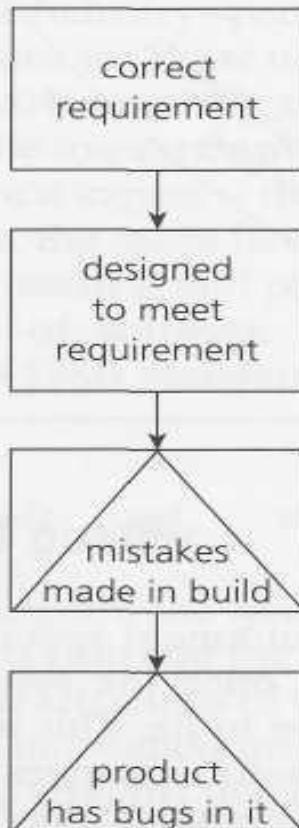
- ❖ Nhằm **tăng độ tin cậy** cũng như **chất lượng** của phần mềm.
- ❖ **Giảm chi phí** trong quá trình phát triển, nâng cấp, bảo trì phần mềm
- ❖ Ví dụ:
 - Website công ty có nhiều lỗi chính tả trong câu chữ → Khách hàng có thể lảng tránh công ty với lý do công ty trông có vẻ không chuyên nghiệp.
 - Một phần mềm tính toán lượng thuốc trừ sâu dùng cho cây trồng, vì lý do tính sai số lượng lên gấp 10 lần → Nông dân phải bỏ nhiều tiền mua, cây trồng hư hại, môi trường sống, nguồn nước bị ảnh hưởng,...

Lỗi tăng lên khi nào?

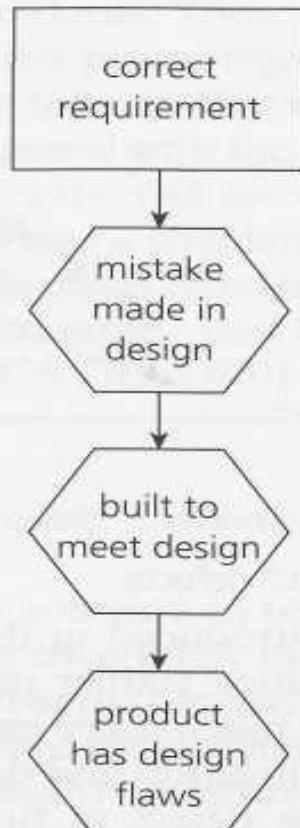
Requirement 1



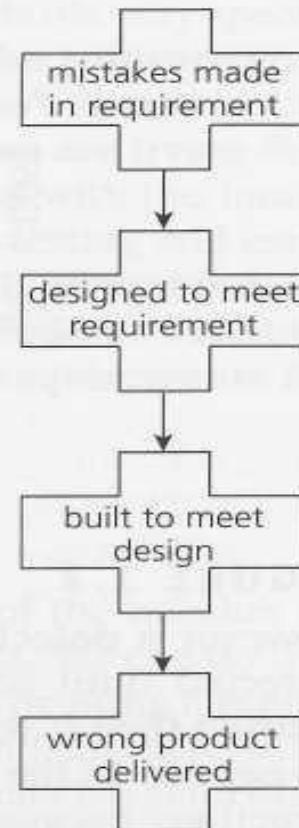
Requirement 2



Requirement 3



Requirement 4



correct functional
and non-functional
attributes delivered

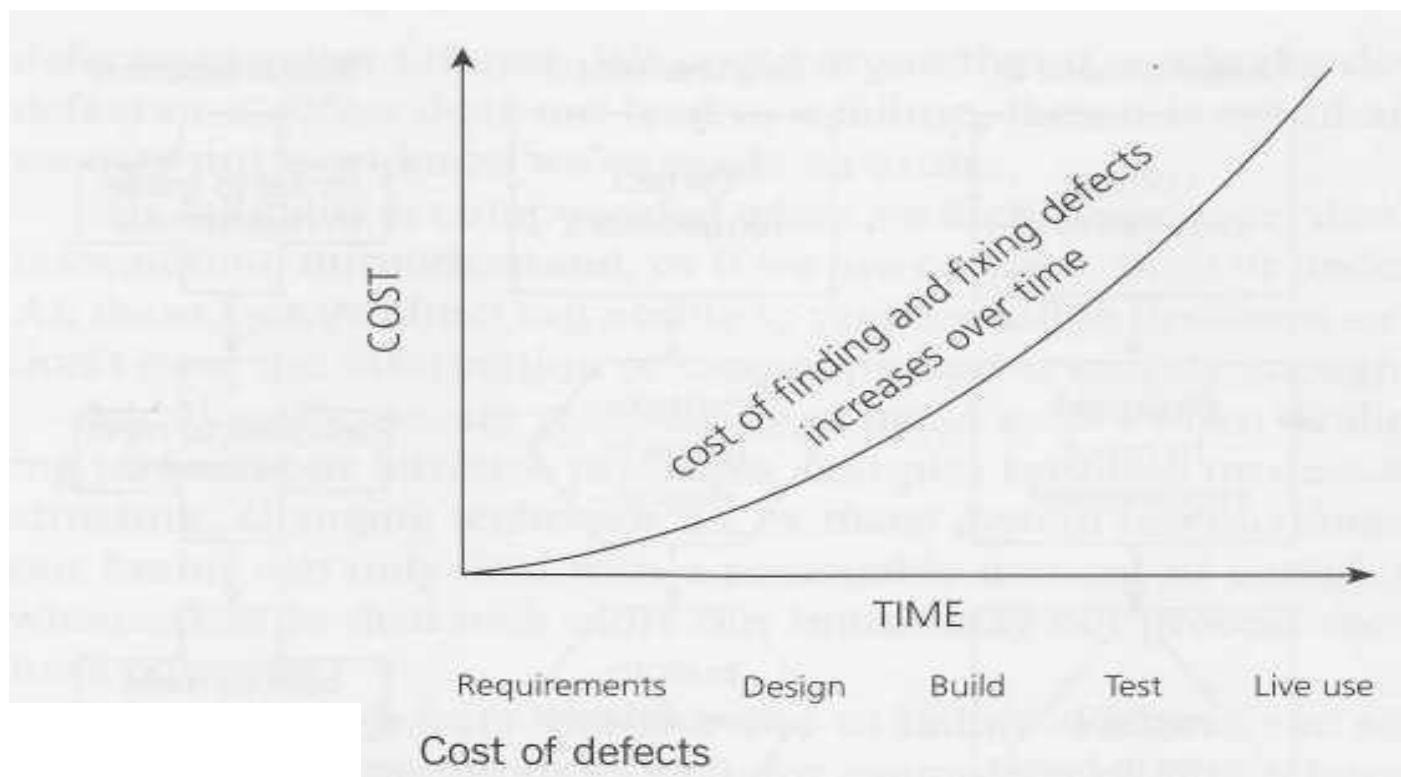
correctable defects

redesign to correct
defects

defects may be
hidden from the IT
team including testers

Lỗi tăng lên khi nào?

- ❖ Chi phí cho việc tìm thấy và sửa lỗi **tăng dần** trong suốt chu kỳ sống của phần mềm. Lỗi tìm thấy **càng sớm** thì chi phí để sửa **càng thấp** và ngược lại.





Các nguyên lý trong kiểm thử PM

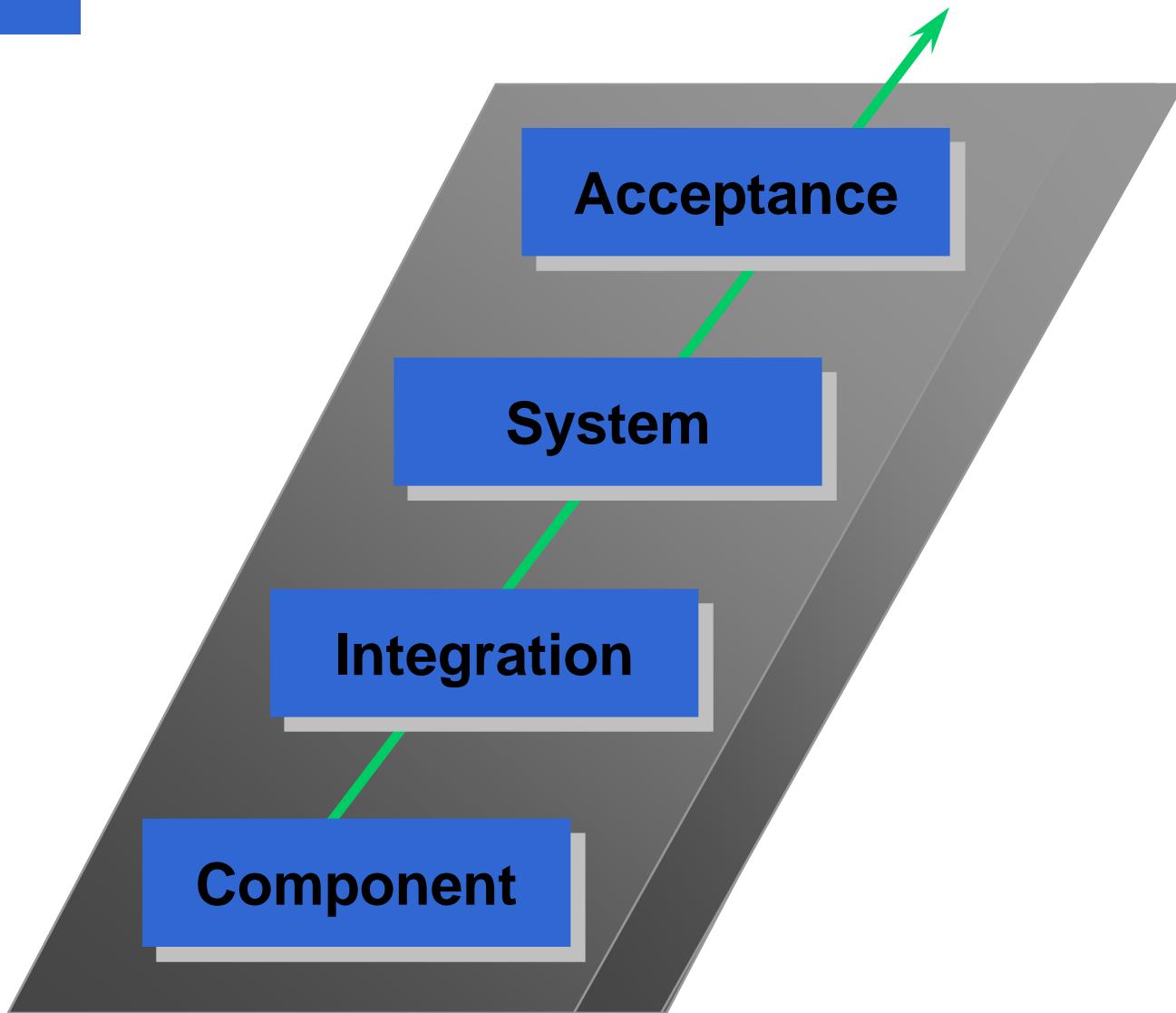
- ❖ Lập trình viên không nên thực hiện kiểm thử trên phần mềm mà mình đã viết
- ❖ Cần phải kiểm tra các chức năng mà phần mềm **không thực hiện**
- ❖ Tránh việc kiểm thử phần mềm với **giả định** rằng sẽ **không có lỗi** nào được tìm thấy
- ❖ Test case phải định nghĩa **kết quả đầu ra rõ ràng**
- ❖ Test case phải được lưu trữ và thực thi lại mỗi khi có sự thay đổi xảy ra trong hệ thống



Vai trò kiểm thử

- ❖ Vai trò kiểm thử trong suốt quy trình sống của phần mềm
 - Kiểm thử không tồn tại độc lập.
 - Các hoạt động của kiểm thử luôn gắn liền với các hoạt động phát triển phần mềm.
 - Các mô hình phát triển phần mềm khác nhau cần các cách tiếp cận kiểm thử khác nhau.

Các mức độ kiểm thử (Test levels)





Các mức độ kiểm thử (Test levels)

❖ Component testing (unit testing):

- Tìm lỗi trong các component của phần mềm như: modules, objects, classes,...
- Do có kích thước nhỏ nên việc tổ chức, kiểm tra, ghi nhận và phân tích kết quả trên Unit test **có thể thực hiện dễ dàng**
- Tiết kiệm thời gian, chi phí trong việc dò tìm và sửa lỗi trong các mức kiểm tra sau



Các mức độ kiểm thử (Test levels)

❖ Integration testing:

- Test sự kết hợp của các component, sự tác động của các phần khác nhau trong một hệ thống, sự kết hợp của các hệ thống với nhau,...



Các mức độ kiểm thử (Test levels)

❖ System testing:

- Đảm bảo rằng hệ thống (sau khi tích hợp) thỏa mãn tất cả các yêu cầu của người sử dụng
- Tập trung vào việc phát hiện các lỗi xảy ra trên toàn hệ thống

❖ Acceptance testing:

- Test phần mềm đứng dưới góc độ người dùng để xác định phần mềm có được chấp nhận hay không.



Các kỹ thuật kiểm thử

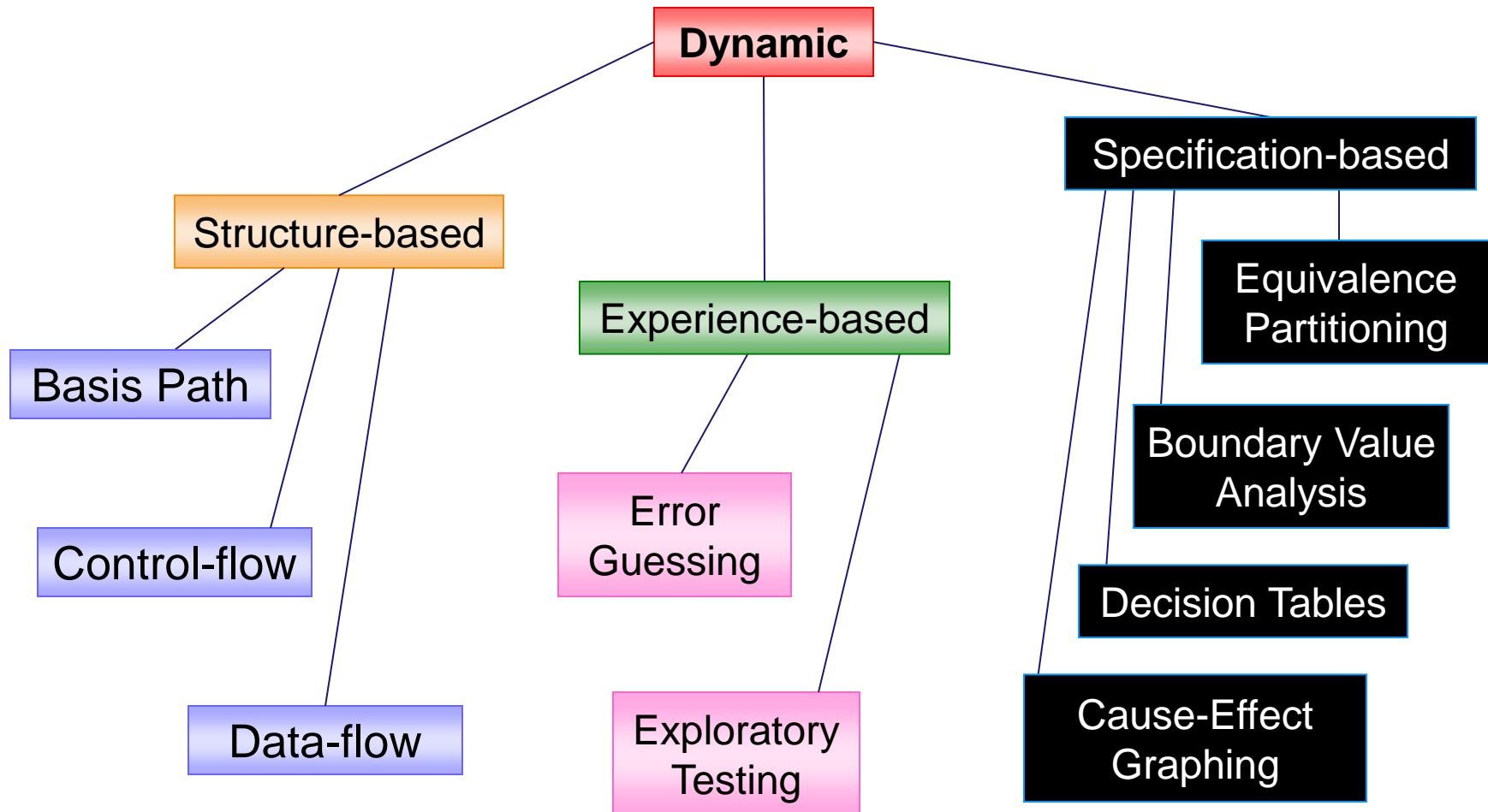
❖ Test tĩnh (Static Verification)

- Thực hiện kiểm chứng mà không cần thực thi chương trình
- Kiểm tra tính đúng đắn của các tài liệu có liên quan được tạo ra trong quá trình xây dựng ứng dụng
- Đạt được sự nhất quán và hiểu rõ hơn về hệ thống
- Giảm thời gian lập trình, thời gian và chi phí test,...

❖ Test động (Dynamic Testing)

- Thực hiện kiểm thử dựa trên việc thực thi chương trình

Dynamic Testing - Kiểm thử động





Các phương pháp kiểm thử (1)

❖ Funtional Testing (Black Box Testing):

- Test dựa trên mô tả, chúng ta xem xét phần mềm với các dữ liệu đầu vào và đầu ra mà không cần biết cấu trúc của phần mềm ra sao. Nghĩa là tester sẽ tập trung vào **những gì mà phần mềm làm**, không cần biết phần mềm làm như thế nào.
- Ưu điểm:
 - Không phụ thuộc vào việc thực hiện phần mềm
 - Việc phát triển test case có thể diễn ra song song với quá trình thực hiện phần mềm → Rút ngắn thời gian thực hiện dự án



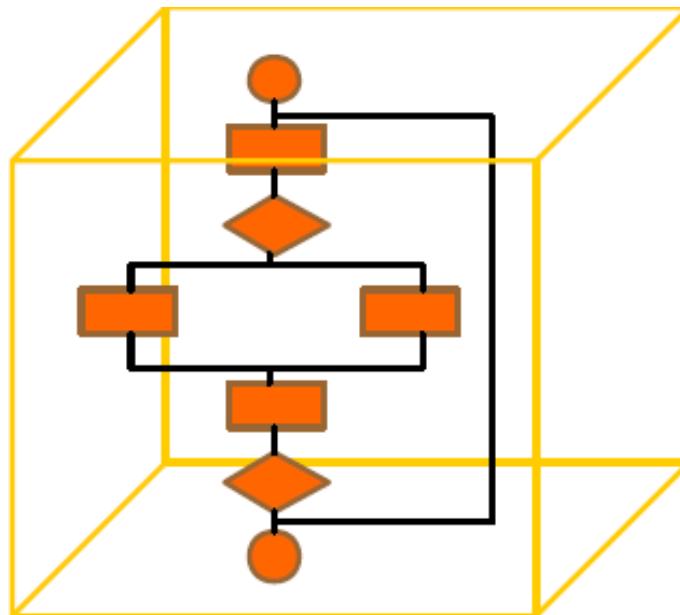
Các kỹ thuật kiểm thử hộp đen

- ❖ Kỹ thuật phân lớp tương đương (Equivalence Class Testing)
- ❖ Kỹ thuật dựa trên giá trị biên (Boundary Value Testing)
- ❖ Kỹ thuật dựa trên bảng quyết định (Decision Table-Based Testing)
- ❖ Kỹ thuật dựa trên đồ thị nguyên nhân – kết quả (causes-effects)
- ❖ ...

Các phương pháp kiểm thử (2)

❖ Structural Testing (White Box Testing):

- Test dựa trên cấu trúc còn được gọi là white-box hay glass-box bởi vì nó đòi hỏi sự hiểu biết về cấu trúc của phần mềm, nghĩa là phần mềm hoạt động như thế nào.





Các kỹ thuật kiểm thử hộp trắng

- ❖ Basis Path Testing
- ❖ Control-flow/Coverage Testing
- ❖ Data-flow Testing



Các phương pháp kiểm thử (3)

❖ Experience Testing (Test dựa trên kinh nghiệm)

- Kỹ thuật này đòi hỏi sự hiểu biết, kỹ năng và kinh nghiệm của người test.
- Dựa vào những kinh nghiệm thu thập được từ những hệ thống trước đó, tester có thể dễ dàng nhìn thấy được những điểm sai trong chương trình.



Các kỹ thuật kiểm thử hộp đen (1)

- ❖ Kỹ thuật phân lớp tương đương (Equivalence Class Testing)
- ❖ Kỹ thuật dựa trên giá trị biên (Boundary Value Testing)
- ❖ Kỹ thuật dựa trên bảng quyết định (Decision Table-Based Testing)
- ❖ Kỹ thuật dựa trên đồ thị nguyên nhân – kết quả (causes-effects)
- ❖ ...



Kỹ thuật phân lớp tương đương

- ❖ Ví dụ: Một textbox chỉ cho phép nhập số nguyên từ 1 đến 100
 - Ta không thể nhập tất cả các giá trị từ 1 đến 100
- ❖ **Ý tưởng của kỹ thuật này:** Chia (partition) đầu vào thành những nhóm tương đương nhau (equivalence).
- ❖ **Giảm đáng kể số lượng test case** cần phải thiết kế vì với mỗi lớp tương đương ta chỉ cần test trên các phần tử đại diện

Kỹ thuật phân lớp tương đương

- ❖ Có hai yếu tố ảnh hưởng đến việc thiết kế test case
 - Dựa trên giả định (Assumption)
 - Single fault assumption → Weak ECT (Equivalence Class Testing)
 - Multiple fault assumption → Strong ECT
 - Dựa trên loại dữ liệu inputs
 - Kiểm thử trên dữ liệu hợp lệ → Normal ECT
 - Kiểm thử trên dữ liệu không hợp lệ → Robust ECT

Assumption Data	Single Fault	Multiple Faults
Valid	Weak Normal	Strong Normal
Invalid	Weak Robust	Strong Robust



Kỹ thuật phân lớp tương đương

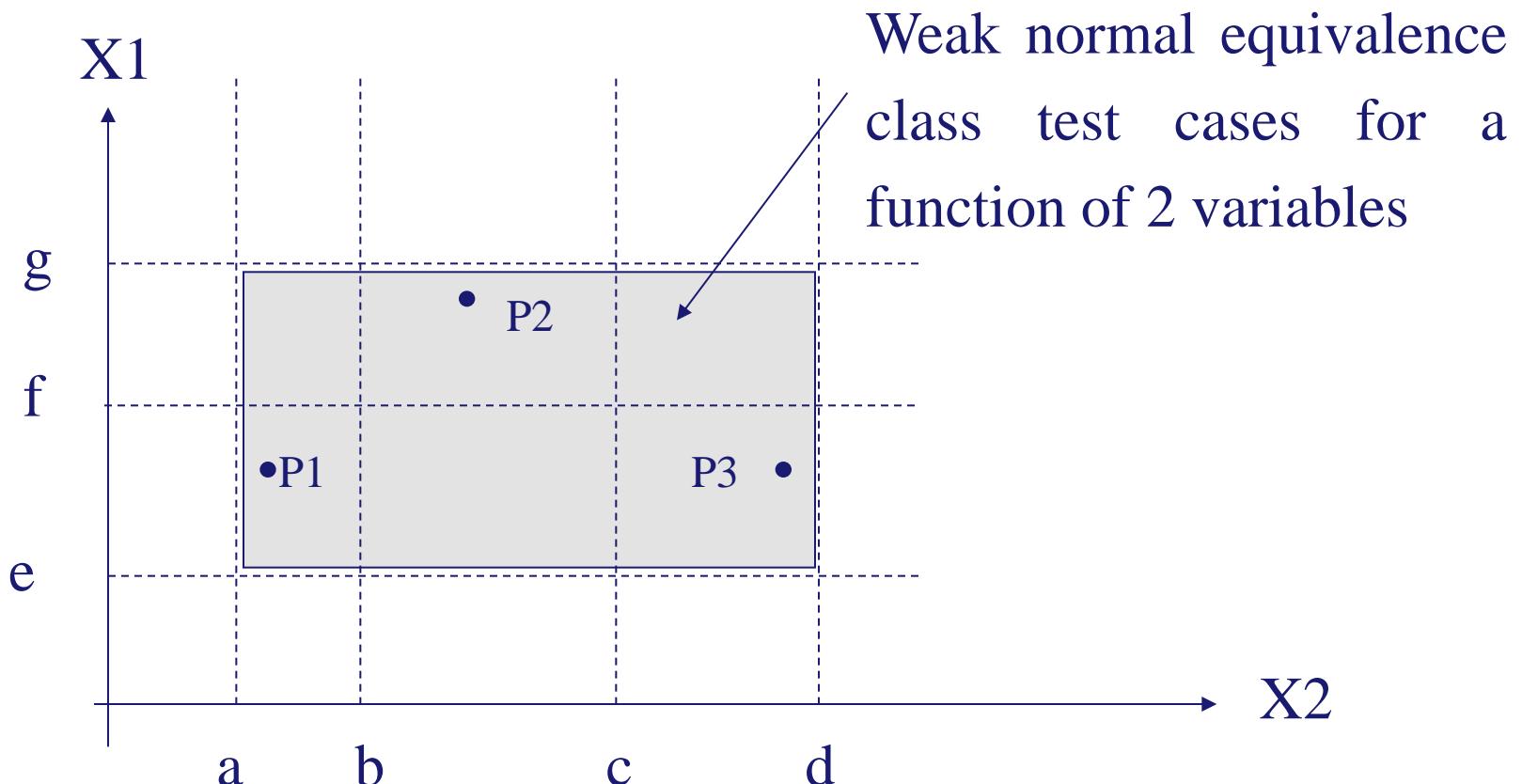
- ❖ Weak Normal Equivalence Class Testing
- ❖ Strong Normal Equivalence Class Testing
- ❖ Weak Robust Equivalence Class Testing
- ❖ Strong Robust Equivalence Class Testing



Weak Normal Equivalence Class Testing

- ❖ Dựa trên Single Fault Assumption
 - Một failure ít khi nào là kết quả của 2 hay nhiều faults xảy ra cùng 1 lúc
- ❖ Ví dụ:
 - $e \leq x_1 \leq g$, x_1 có 2 lớp tương đương $[e, f)$ $[f, g]$
 - $a \leq x_2 \leq d$, x_2 có 3 lớp tương đương $[a, b)$ $[b, c)$, $[c, d]$

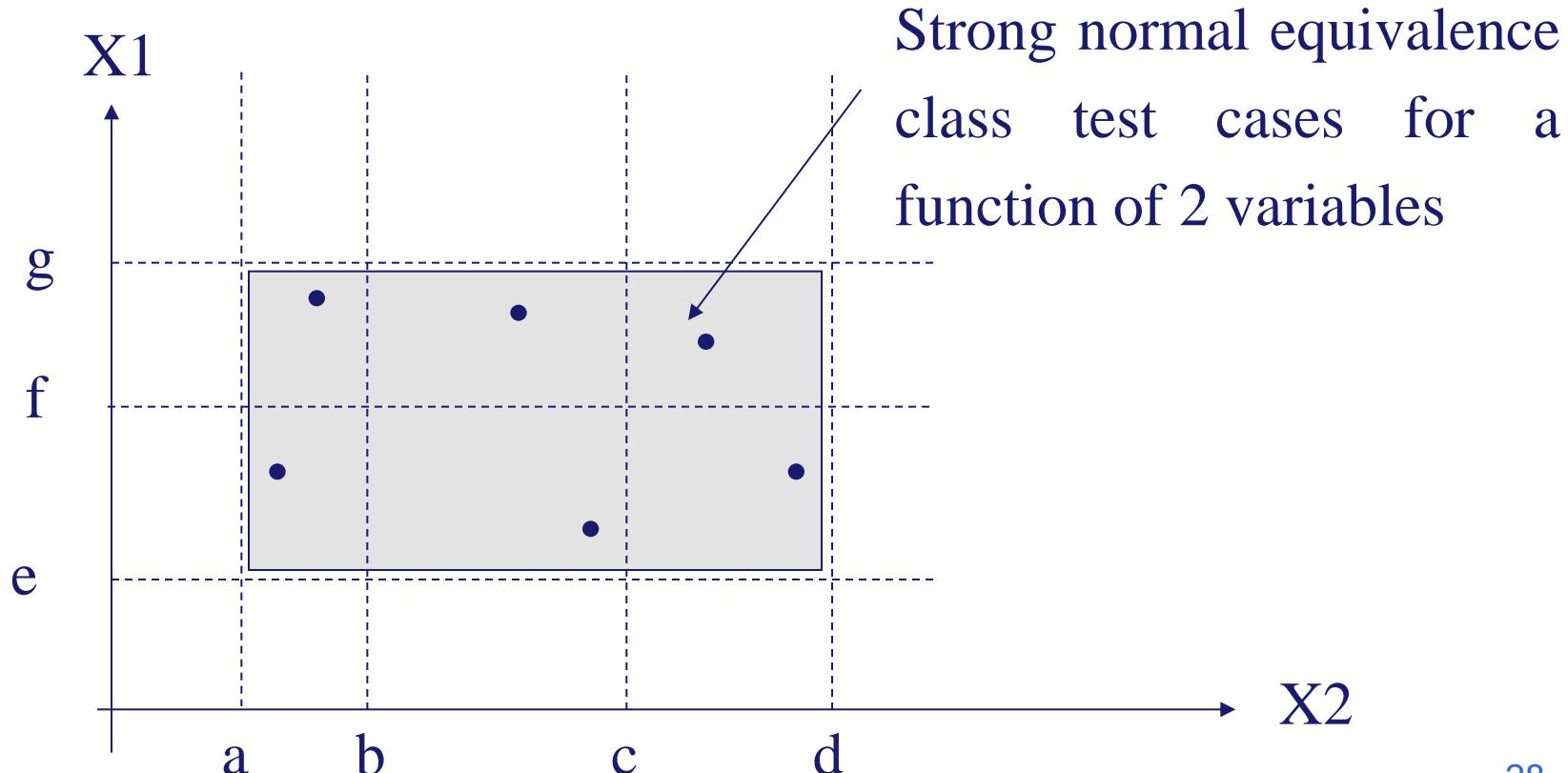
Weak Normal Equivalence Class Testing



Strong Normal Equivalence Class Testing

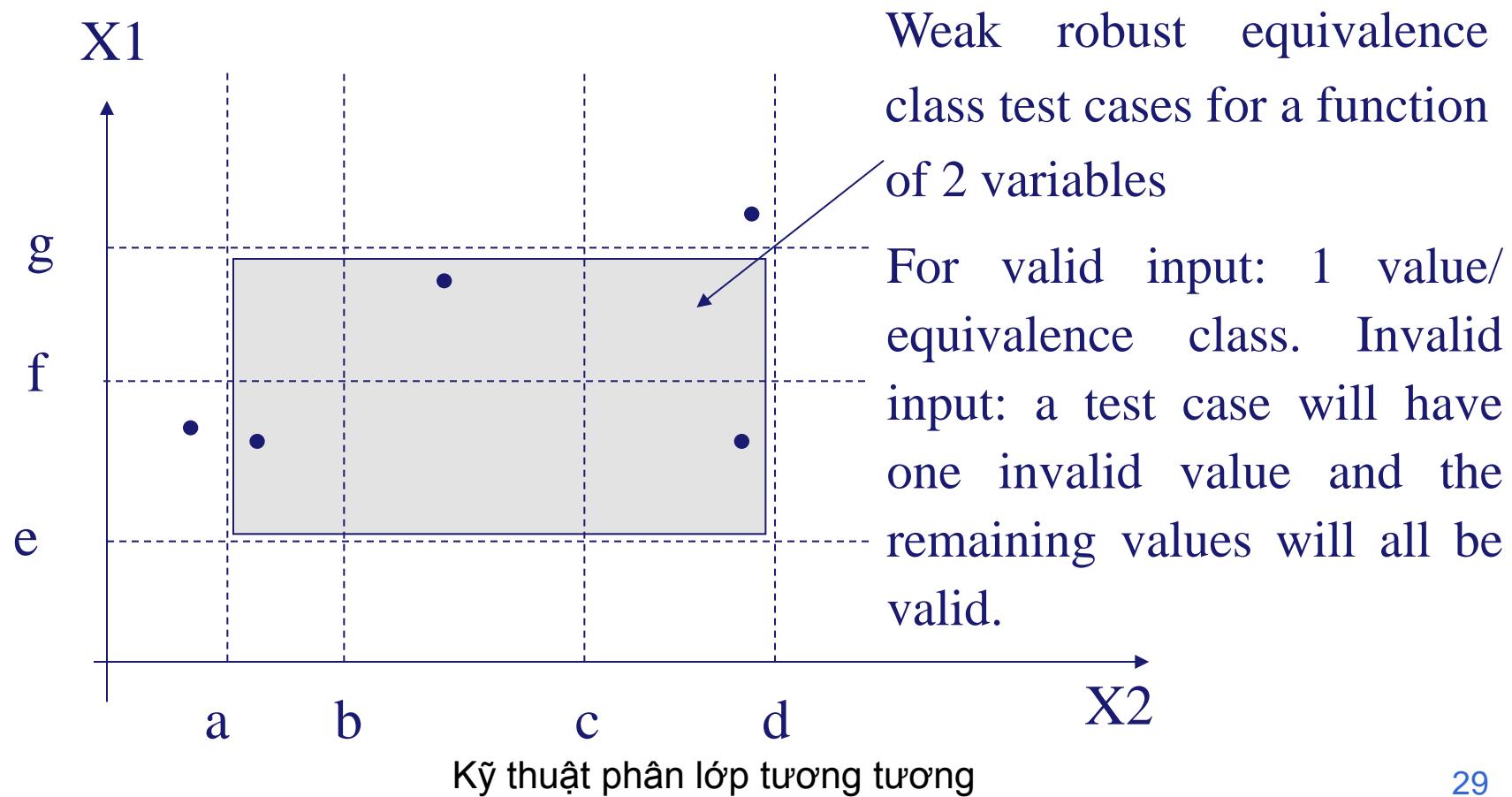
❖ Dựa trên Multiple Fault Assumption

- Một failure có thể là kết quả của 2 hay nhiều faults xảy ra cùng 1 lúc

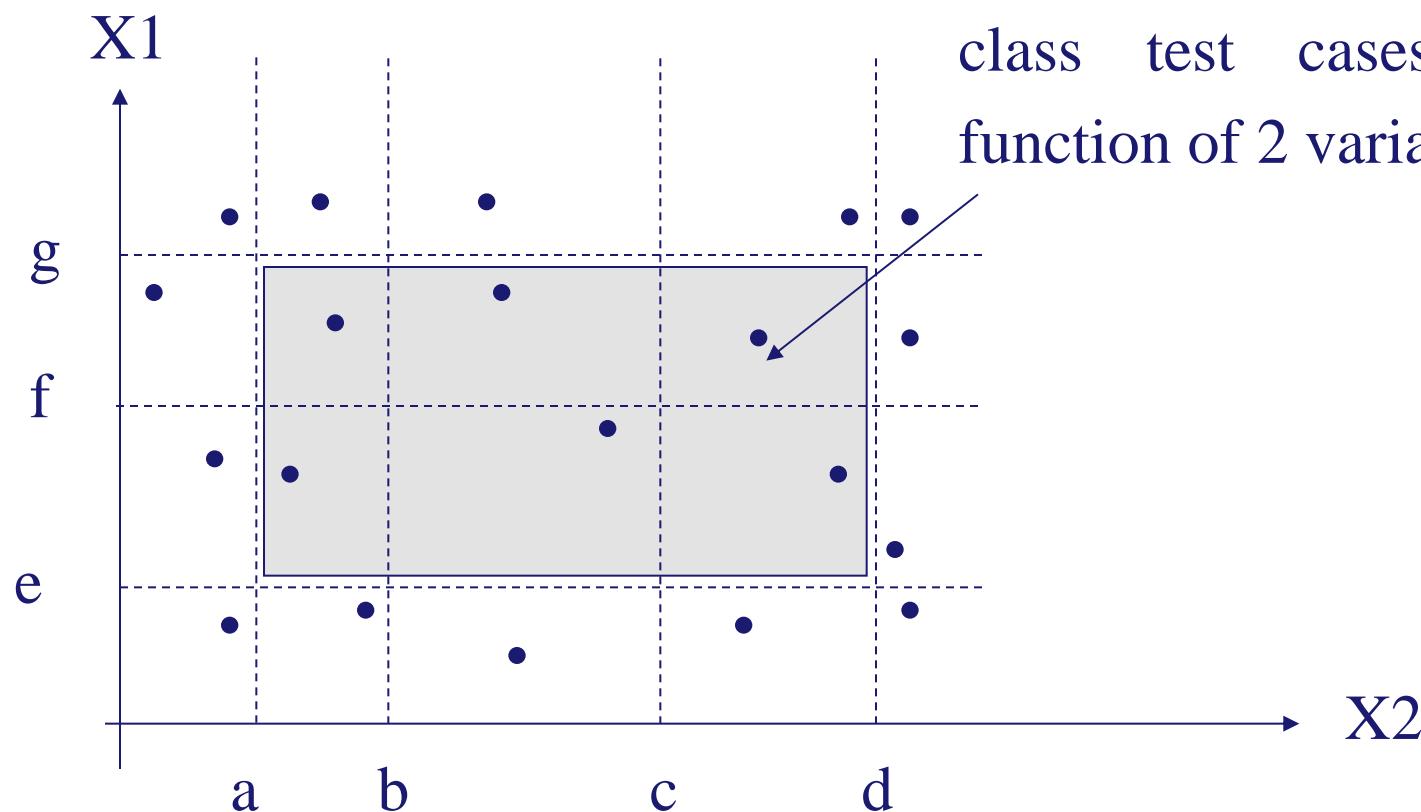


Weak Robust Equivalence Class Testing

- ❖ Tương tự Weak Equivalence Class Testing, tuy nhiên test thêm trường hợp 1 biến với giá trị không hợp lệ



Strong Robust Equivalence Class Testing



Strong robust equivalence
class test cases for a
function of 2 variables



Các kỹ thuật kiểm thử hộp đen (2)

- ❖ Kỹ thuật phân lớp tương đương (Equivalence Class Testing)
- ❖ Kỹ thuật dựa trên giá trị biên (Boundary Value Testing)
- ❖ Kỹ thuật dựa trên bảng quyết định (Decision Table-Based Testing)
- ❖ Kỹ thuật dựa trên đồ thị nguyên nhân – kết quả (causes-effects)
- ❖ ...



Kỹ thuật phân tích giá trị biên

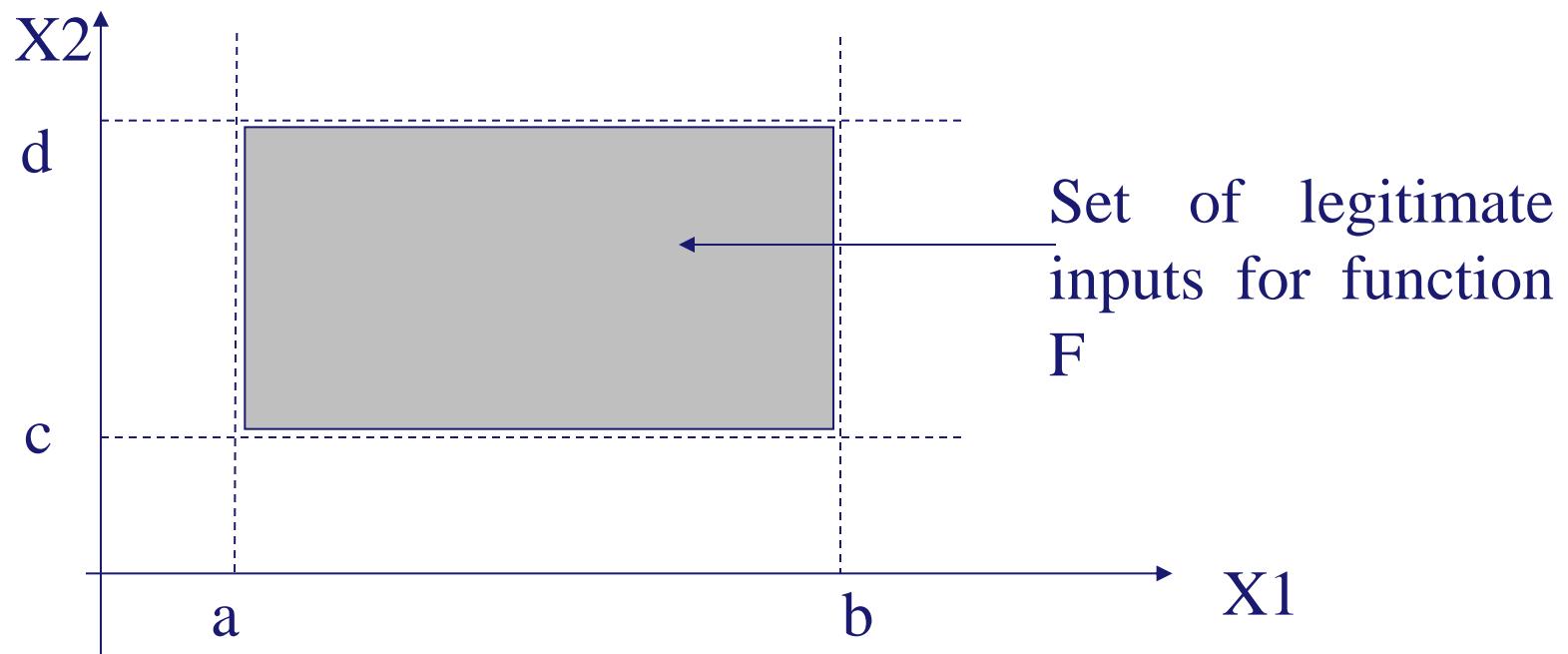
- ❖ Phân tích giá trị biên - Boundary Value Analysis
- ❖ Thường được áp dụng đối với các đối số của một phương thức
- ❖ Tập trung vào việc kiểm thử các giá trị biên của miền giá trị inputs để thiết kế test case do “lỗi thường tiềm ẩn lại các ngõ ngách và tập hợp tại biên” (Beizer)
- ❖ BVA hiệu quả nhất trong trường hợp “các đối số đầu vào (input variables) độc lập với nhau và mỗi đối số đều có một miền giá trị hữu hạn”

Kỹ thuật phân tích giá trị biên

❖ Giả sử hàm F có hai biến X_1, X_2 như sau:

- $a \leq X_1 \leq b$
- $c \leq X_2 \leq d$

❖ Input domain of a function of two variables:





Một số kỹ thuật kiểm thử giá trị biên

- ❖ Standard BVA (Boundary Value Analysis)
- ❖ Robustness testing
- ❖ Worst-case testing
- ❖ Robust worst-case testing

Standard BVA

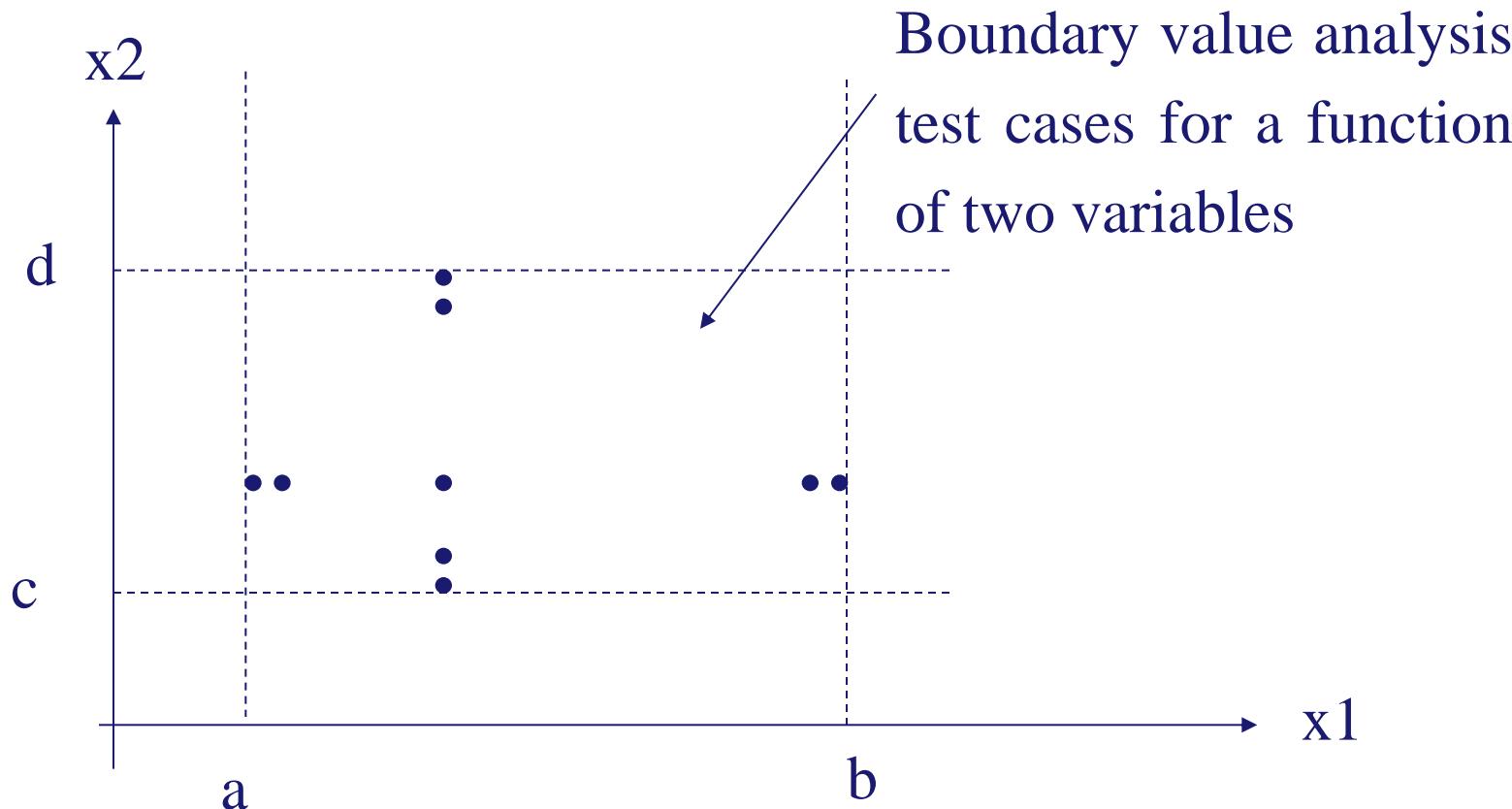
❖ Giả sử biến x có **miền giá trị** $[min, max]$

→ Các giá trị được chọn để kiểm tra

- Min - Minimal
- Min+ - Just above Minimal
- Nom - Average
- Max- - Just below Maximum
- Max - Maximum

Kỹ thuật phân tích trên giá trị biên

- ❖ Số test case là $4n+1$, với n là số lượng biến



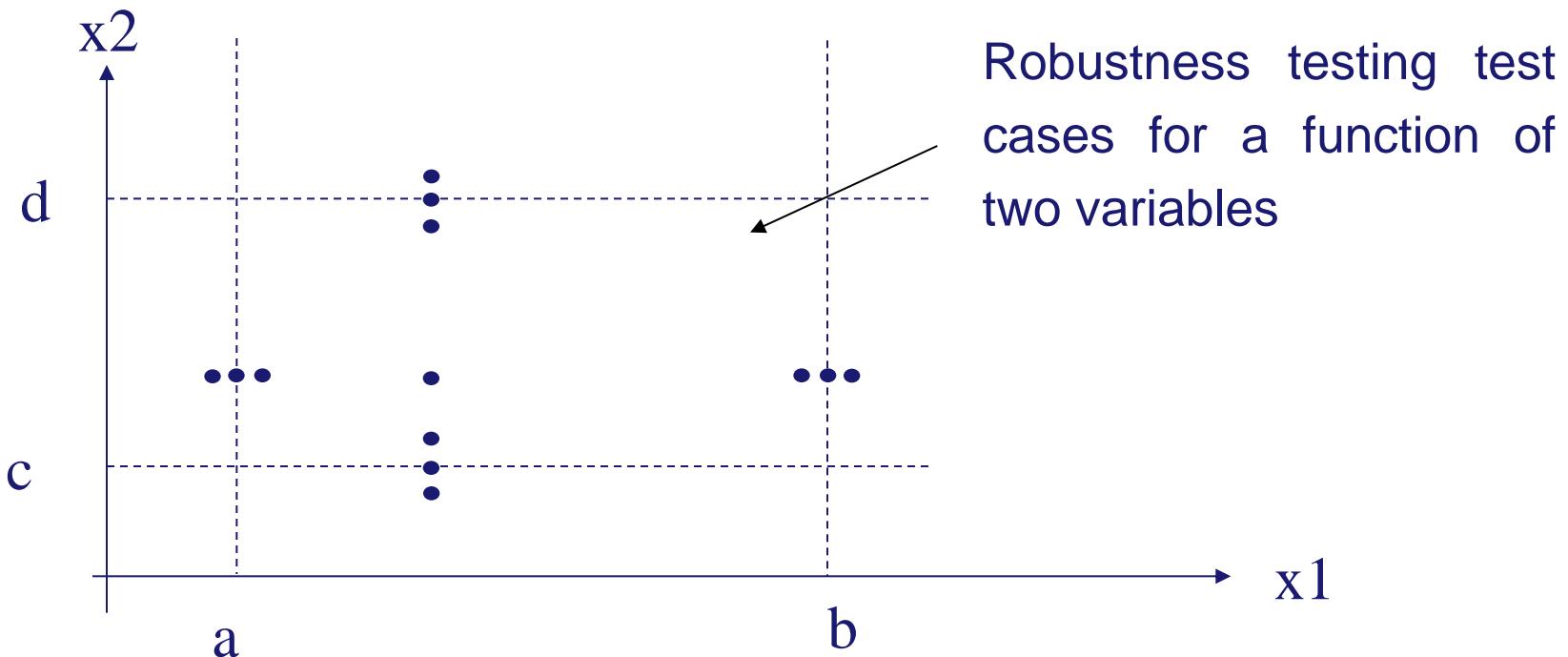


Robustness Testing

- ❖ Mở rộng của Standard BVA
- ❖ Kiểm thử cả hai trường hợp:
 - Input variable hợp lệ (clean test cases)
 - Kiểm thử tương tự như Standard BVA trên các giá trị (min, min+, average, max-, max)
 - Input variable không hợp lệ (dirty test cases)
 - Kiểm thử trên 2 giá trị: min-, max+ (nằm ngoài miền giá trị hợp lệ)

Robustness Testing

- ❖ Số lượng test case là $6n + 1$, với n là số lượng biến



- ❖ Tập trung vào việc kiểm thử trên các giá trị không hợp lệ và đòi hỏi ứng dụng phải xử lý ngoại lệ một cách đầy đủ

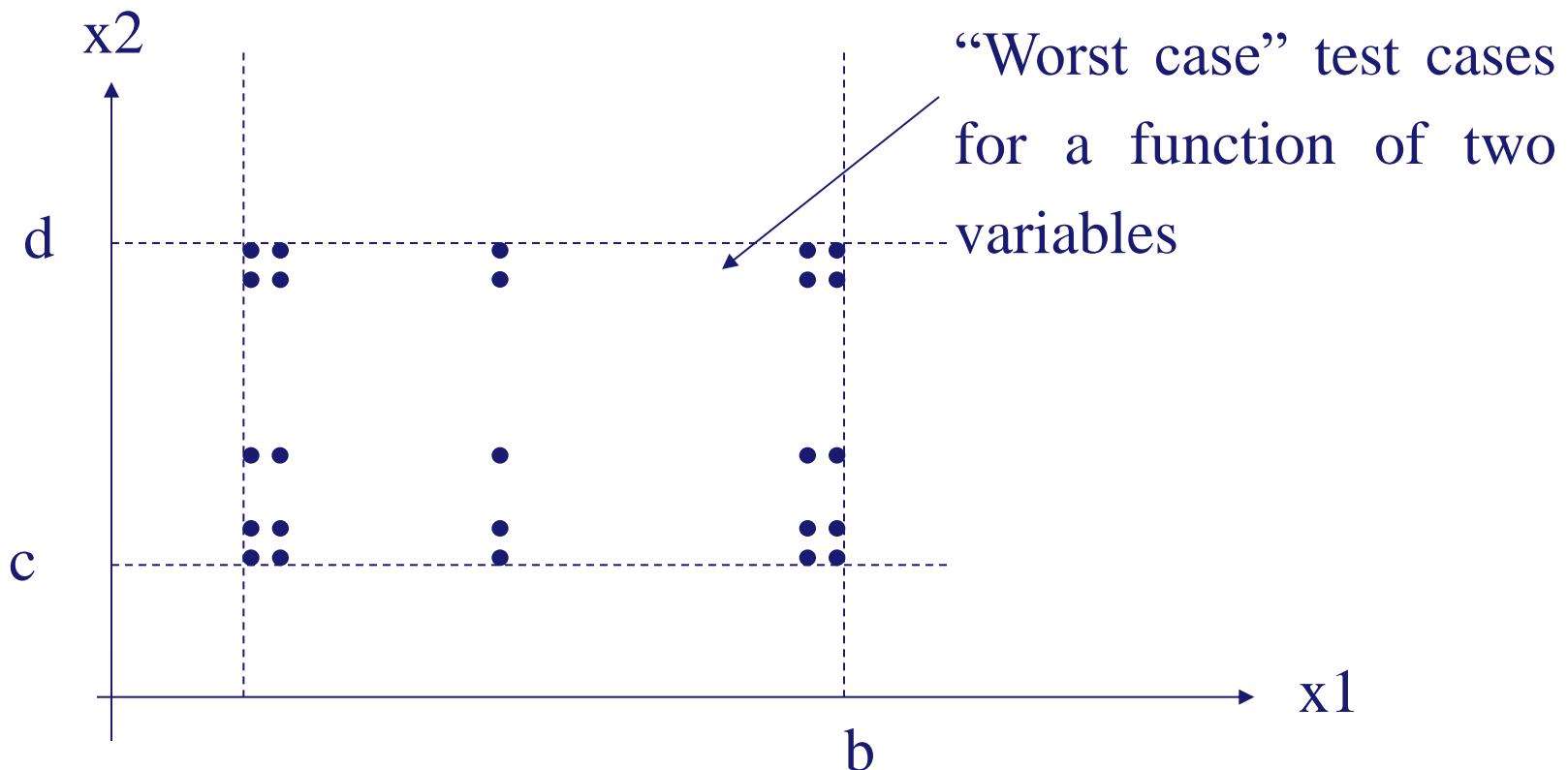


Worst-case testing

- ❖ Dựa trên **Multiple Fault Assumption** để thiết kế test case
- ❖ Các biến sẽ được kiểm tra đồng thời tại biên để dò lỗi
- ❖ Chúng ta không kiểm thử tại các giá trị không hợp lệ

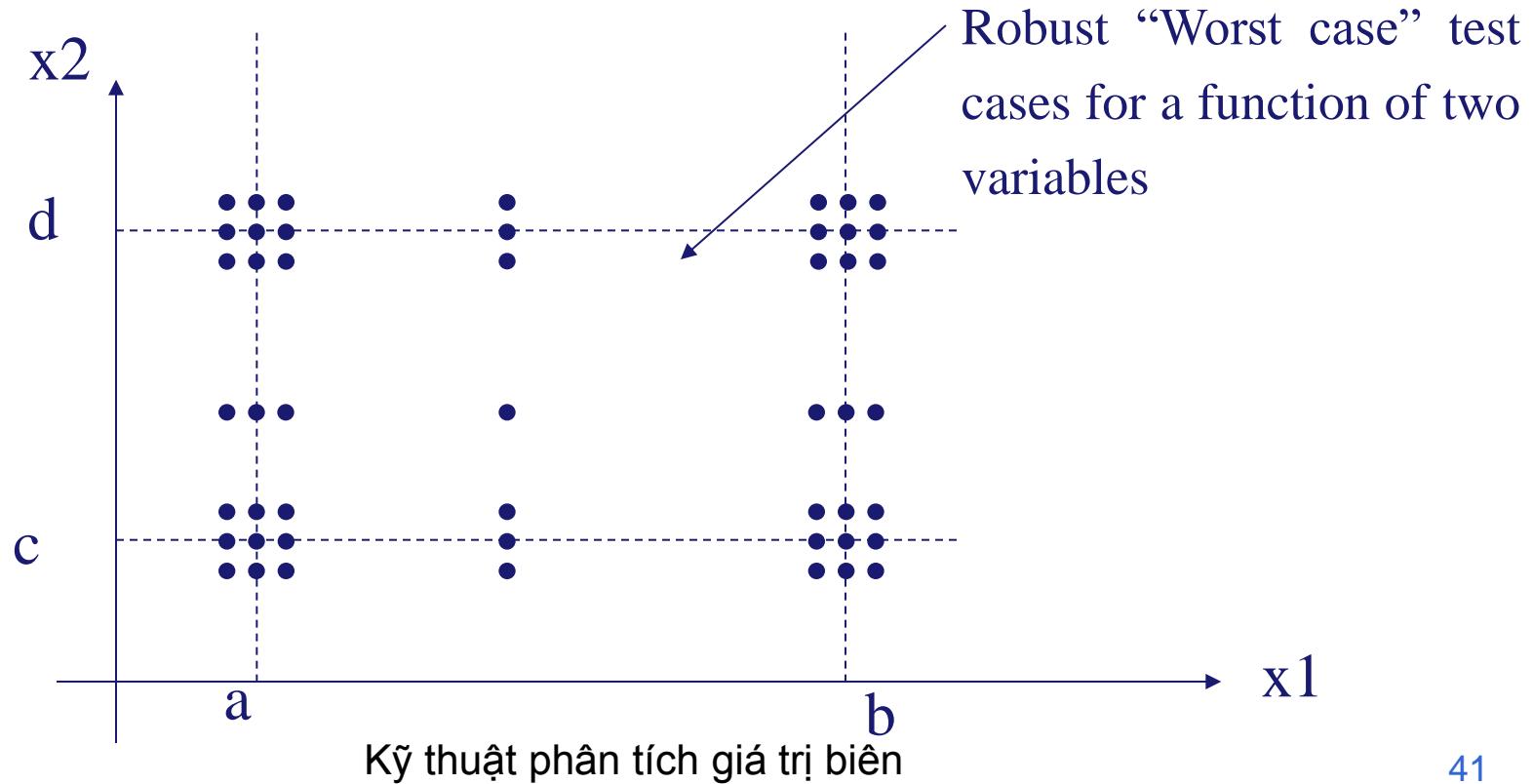
Worst-case testing

- ❖ Số lượng test case là 5^n , với n là số biến



Robust worst-case testing

- ❖ Tương tự Worst-case Testing nhưng kiểm tra thêm tại các giá trị không hợp lệ của input variables (min-, max+)
- ❖ Số lượng test case là 7^n , với n là số biến



Ví dụ hàm kiểm tra tam giác

- ❖ Ràng buộc: $1 \leq a, b, c \leq 200$.
- ❖ Áp dụng Standard BVA (số test case $4*3 + 1 = 13$)

- $\min = 1$
- $\min+ = 2$
- $\text{nom} = 100$
- $\max- = 199$
- $\max = 200$

Boundary Value Analysis Test Cases				
Case	a	b	c	Expected Output
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a Triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	199	100	Isosceles
9	100	200	100	Not a Triangle
10	1	100	100	Isosceles
11	2	100	100	Isosceles
12	199	100	100	Isosceles
13	200	100	100	Not a Triangle

Ví dụ hàm kiểm tra tam giác

❖ Áp dụng
Worst-case
testing

Worst Case Test Cases (60 of 125)				
Case	a	b	c	Expected Output
1	1	1	1	Equilateral
2	1	1	2	Not a Triangle
3	1	1	100	Not a Triangle
4	1	1	199	Not a Triangle
5	1	1	200	Not a Triangle
6	1	2	1	Not a Triangle
7	1	2	2	Isosceles
8	1	2	100	Not a Triangle
9	1	2	199	Not a Triangle
10	1	2	200	Not a Triangle
11	1	100	1	Not a Triangle
12	1	100	2	Not a Triangle
13	1	100	100	Isosceles
14	1	100	199	Not a Triangle
15	1	100	200	Not a Triangle
16	1	199	1	Not a Triangle
17	1	199	2	Not a Triangle
18	1	199	100	Not a Triangle
19	1	199	199	Isosceles
20	1	199	200	Not a Triangle
21	1	200	1	Not a Triangle
22	1	200	2	Not a Triangle
23	1	200	100	Not a Triangle
24	1	200	199	Not a Triangle
25	1	200	200	Isosceles
26	2	1	1	Not a Triangle
27	2	1	2	Isosceles
28	2	1	100	Not a Triangle
29	2	1	199	Not a Triangle
30	2	1	200	Not a Triangle
31	2	2	1	Isosceles
32	2	2	2	Equilateral
33	2	2	100	Not a Triangle
34	2	2	199	Not a Triangle
35	2	2	200	Not a Triangle
36	2	100	1	Not a Triangle
37	2	100	2	Not a Triangle
38	2	100	100	Isosceles
39	2	100	199	Not a Triangle
40	2	100	200	Not a Triangle
41	2	199	1	Not a Triangle
42	2	199	2	Not a Triangle
43	2	199	100	Not a Triangle
44	2	199	199	Isosceles
45	2	199	200	Scalene
46	2	200	1	Not a Triangle
47	2	200	2	Not a Triangle
48	2	200	100	Not a Triangle
49	2	200	199	Scalene
50	2	200	200	Isosceles
51	100	1	1	Not a Triangle
52	100	1	2	Not a Triangle
53	100	1	100	Isosceles
54	100	1	199	Not a Triangle
55	100	1	200	Not a Triangle
56	100	2	1	Not a Triangle
57	100	2	2	Not a Triangle
58	100	2	100	Isosceles
59	100	2	199	Not a Triangle
60	100	2	200	Not a Triangle



Ví dụ hàm tìm ngày kế tiếp

- ❖ Bài toán tìm ngày kế tiếp với các ràng buộc:
 - $1 \leq \text{Day} \leq 31$.
 - $1 \leq \text{month} \leq 12$.
 - $1812 \leq \text{Year} \leq 2012$
- ❖ Áp dụng Standard BVA (số test case $4^*3 + 1 = 13$)

Ví dụ hàm tìm ngày kế tiếp

<u>month</u>	<u>day</u>
min = 1	min = 1
min+ = 2	min+ = 2
nom = 6	nom = 15
max- = 11	max- = 30
max = 12	max = 31

<u>year</u>
min = 1812
min+ = 1813
nom = 1912
max- = 2011
max = 2012

Boundary Value Analysis Test Cases

Case	month	day	year	Expected Output
1	6	15	1812	June 16, 1812
2	6	15	1813	June 16, 1813
3	6	15	1912	June 16, 1912
4	6	15	2011	June 16, 2011
5	6	15	2012	June 16, 2012
6	6	1	1912	June 2, 1912
7	6	2	1912	June 3, 1912
8	6	30	1912	July 1, 1912
9	6	31	1912	error
10	1	15	1912	January 16, 1912
11	2	15	1912	February 16, 1912
12	11	15	1912	November 16, 1912
13	12	15	1912	December 16, 1912

Ví dụ hàm tìm ngày kế tiếp

- ❖ Áp dụng Worst-case testing, Số lượng test case: 5^3

Worst Case Test Cases (60 of 125)				
Case	month	day	year	Expected Output
1	1	1	1812	January 2, 1812
2	1	1	1813	January 2, 1813
3	1	1	1912	January 2, 1912
4	1	1	2011	January 2, 2011
5	1	1	2012	January 2, 2012
6	1	2	1812	January 3, 1812
7	1	2	1813	January 3, 1813
8	1	2	1912	January 3, 1912
9	1	2	2011	January 3, 2011
10	1	2	2012	January 3, 2012
11	1	15	1812	January 16, 1812
12	1	15	1813	January 16, 1813
13	1	15	1912	January 16, 1912
14	1	15	2011	January 16, 2011
15	1	15	2012	January 16, 2012
16	1	30	1812	January 31, 1812
17	1	30	1813	January 31, 1813
18	1	30	1912	January 31, 1912
19	1	30	2011	January 31, 2011
20	1	30	2012	January 31, 2012
21	1	31	1812	February 1, 1812
22	1	31	1813	February 1, 1813
23	1	31	1912	February 1, 1912
24	1	31	2011	February 1, 2011
25	1	31	2012	February 1, 2012
26	2	1	1812	February 2, 1812
27	2	1	1813	February 2, 1813
28	2	1	1912	February 2, 1912
29	2	1	2011	February 2, 2011
30	2	1	2012	February 2, 2012
31	2	2	1812	February 3, 1812
32	2	2	1813	February 3, 1813
33	2	2	1912	February 3, 1912
34	2	2	2011	February 3, 2011
35	2	2	2012	February 3, 2012
36	2	15	1812	February 16, 1812
37	2	15	1813	February 16, 1813
38	2	15	1912	February 16, 1912
39	2	15	2011	February 16, 2011
40	2	15	2012	February 16, 2012
41	2	30	1812	error
42	2	30	1813	error
43	2	30	1912	error
44	2	30	2011	error
45	2	30	2012	error
46	2	31	1812	error
47	2	31	1813	error
48	2	31	1912	error
49	2	31	2011	error
50	2	31	2012	error
51	6	1	1812	June 2, 1812
52	6	1	1813	June 2, 1813
53	6	1	1912	June 2, 1912
54	6	1	2011	June 2, 2011
55	6	1	2012	June 2, 2012
56	6	2	1812	June 3, 1812
57	6	2	1813	June 3, 1813
58	6	2	1912	June 3, 1912
59	6	2	2011	June 3, 2011
60	6	2	2012	June 3, 2012



Các kỹ thuật kiểm thử hộp đen (3)

- ❖ Kỹ thuật dựa trên giá trị biên (Boundary Value Testing)
- ❖ Kỹ thuật phân lớp tương đương (Equivalence Class Testing)
- ❖ Kỹ thuật dựa trên bảng quyết định (Decision Table-Based Testing)
- ❖ Kỹ thuật dựa trên đồ thị nguyên nhân – kết quả (cause-effect graphing)
- ❖ ...

Bảng quyết định

- ❖ Là kỹ thuật được áp dụng trong nhiều lĩnh vực:
 - Phân tích logic trong các hoạt động nghiệp vụ
 - Lập trình
 - Kiểm thử
 - ...
- ❖ Làm giảm số lượng test case không cần thiết so với 2 kỹ thuật Equivalence Class và Boundary Value Analysis vì nó loại trừ các phép kết hợp không cần thiết giữa các input variables

Bảng quyết định

- ❖ Liệt kê các **nguyên nhân (cause)** – **kết quả (effect)** trong 1 ma trận. Mỗi cột trong ma trận đại diện cho 1 phép kết hợp giữa các cause trong việc tạo ra 1 effect

		Combinations							
Causes	Values	1	2	3	4	5	6	7	8
Cause 1	Y, N	Y	Y	Y	Y	N	N	N	N
Cause 2	Y, N	Y	Y	N	N	Y	Y	N	N
Cause 3	Y, N	Y	N	Y	N	Y	N	Y	N
Effects									
Effect 1		X		X				X	
Effect 2			X			X		X	

Cause = Condition

Effect = Actions = Expected Results



Các bước để tạo ra Bảng quyết định

- ❖ Liệt kê tất cả các nguyên nhân (causes) trong bảng quyết định
- ❖ Tính tổng số lượng kết hợp giữa các cause
- ❖ Điền vào các cột với tất cả các kết hợp có thể có
- ❖ Rút bớt số lượng các phép kết hợp dư thừa
- ❖ Kiểm tra các phép kết hợp có bao phủ hết mọi trường hợp hay không
- ❖ Bổ sung kết quả (effects) vào bảng quyết định

B1: Liệt kê tất cả các nguyên nhân

Causes	Values	Combinations							
		1	2	3	4	5	6	7	8
Cause 1	Y, N	Y	Y	Y	Y	N	N	N	N
Cause 2	Y, N	Y	Y	N	N	Y	Y	N	N
Cause 3	Y, N	Y	N	Y	N	Y	N	Y	N
Effects									
Effect 1		X		X			X		X
Effect 2			X			X		X	

- ❖ Điền vào các giá trị trong từng cause
- ❖ Gom nhóm các causes có liên quan
- ❖ Sắp xếp các cause theo thứ tự giản

Ví dụ: xét bài toán kiểm tra loại của
chiều dài 3 cạnh a, b, c.

c1: $a < b+c?$

c2: $b < a+c?$

c3: $c < a+b?$

c4: $a = b?$

c5: $a = c?$

c6: $b = c?$

B2: Tính tổng số kết hợp giữa các causes

❖ Tổng số phép kết hợp

= (số lượng values của cause 1) * ... * (số lượng values của cause n)

c1: a < b+c?
c2: b < a+c?
c3: c < a+b?
c4: a = b?
c5: a = c?
c6: b = c?

Mỗi cause có 2 giá trị true, false
→ Tổng số phép kết hợp = $2^6 = 64$

B3: Điền giá trị các cột trong bảng

Causes	Values	Combinations							
		1	2	3	4	5	6	7	8
Cause 1	Y, N	Y	Y	Y	Y	N	N	N	N
Cause 2	Y, N	Y	Y	N	N	Y	Y	N	N
Cause 3	Y, N	Y	N	Y	N	Y	N	Y	N
Effects									
Effect 1		X		X				X	
Effect 2			X			X		X	

❖ Thuật toán:

- Xác định số lần lặp lại (RF) trong từng giá trị của cause bằng cách lấy tổng số phép kết hợp **còn lại** chia cho số values mà cause có thể nhận
- Điền dữ liệu cho dòng thứ i: điền RF lần **giá trị đầu tiên** của cause i, tiếp theo RF lần **giá trị tiếp theo** của cause i... cho đến khi dòng đầy
- Chuyển sang dòng kế tiếp, quay lại bước 1 và tiếp tục thực hiện

B3: Điene giá trị các cột trong bảng

❖ Ví dụ:

c1: $a < b+c$?		
c2: $b < a+c$?		
c3: $c < a+b$?		
c4: $a = b$?	

$$RF = 64 / 2 = 32$$

$$RF = 32 / 2 = 16$$

$$RF = 16 / 2 = 8$$

B4: Giảm số phép kết hợp

- ❖ Duyệt qua tất cả các ô trong từng cột, ô nào mà kết quả của nó không ảnh hưởng đến effect thì đặt giá trị trên ô này là “-” (don’t care entry)
- ❖ Ghép các cột với nội dung giống nhau thành 1 cột

F	T	T	T	T	T	T	T	T
-	F	T	T	T	T	T	T	T
-	-	F	T	T	T	T	T	T
-	-	-	T	T	T	F	F	F
-	-	-	T	T	F	T	T	F
-	-	-	T	F	T	F	T	F

B5: Kiểm tra độ bao phủ các phép kết hợp

- ❖ Tính rule-count trên từng cột (số lượng phép kết hợp mà cột này có thể thực hiện)
- ❖ Với các dòng có giá trị là ‘-’ thì luỹ thừa 2
- ❖ Nếu tổng của các rule-count bằng với tổng số kết hợp giữa các cause trong bước 2 thì bảng quyết định là đầy đủ

c1: a < b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a+c?	-	F	T	T	T	T	T	T	T	T	T
c3: c < a+b?	-	-	F	T	T	T	T	T	T	T	T
c4: a = b?	-	-	-	T	T	T	F	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F
c6: b = c?	-	-	-	T	F	T	F	T	F	T	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1

B6: Bổ sung kết quả (effect) vào trong bảng

- ❖ Duyệt qua từng cột và check vào kết quả (effect)
- ❖ Nhiều cột khác nhau có thể cho ra cùng 1 kết quả giống nhau

c1: $a < b+c?$	F	T	T	T	T	T	T	T	T	T	T	T
c2: $b < a+c?$	-	F	T	T	T	T	T	T	T	T	T	T
c3: $c < a+b?$	-	-	F	T	T	T	T	T	T	T	T	T
c4: $a = b?$	-	-	-	T	T	T	T	F	F	F	F	F
c5: $a = c?$	-	-	-	T	T	F	F	T	T	F	F	F
c6: $b = c?$	-	-	-	T	F	T	F	T	F	T	F	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1	1
a1: Not a Triangle	X	X	X									
a2: Scalene												X
a3: Isosceles							X		X	X		
a4: Equilateral				X								
a5: Impossible					X	X		X				

Ví dụ

❖ Bảng quyết định hoàn chỉnh

c1: a < b+c?	F	T	T	T	T	T	T	T	T	T	T	T
c2: b < a+c?	-	F	T	T	T	T	T	T	T	T	T	T
c3: c < a+b?	-	-	F	T	T	T	T	T	T	T	T	T
c4: a = b?	-	-	-	T	T	T	T	F	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F	F
c6: b = c?	-	-	-	T	F	T	F	T	F	T	F	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1	1
a1: Not a Triangle	X	X	X									
a2: Scalene												X
a3: Isosceles								X		X	X	
a4: Equilateral					X							
a5: Impossible						X	X		X			



Các kỹ thuật kiểm thử hộp đen (4)

- ❖ Kỹ thuật dựa trên giá trị biên (Boundary Value Testing)
- ❖ Kỹ thuật phân lớp tương đương (Equivalence Class Testing)
- ❖ Kỹ thuật dựa trên bảng quyết định (Decision Table-Based Testing)
- ❖ Kỹ thuật dựa trên đồ thị nguyên nhân – kết quả (causes-effects)
- ❖ ...



Đồ thị nguyên nhân – kết quả

- ❖ Là kỹ thuật thiết kế test case dựa trên đồ thị
- ❖ Tập trung vào việc xác định các mối kết hợp giữa các conditions và kết quả mà các mối kết hợp này mang lại



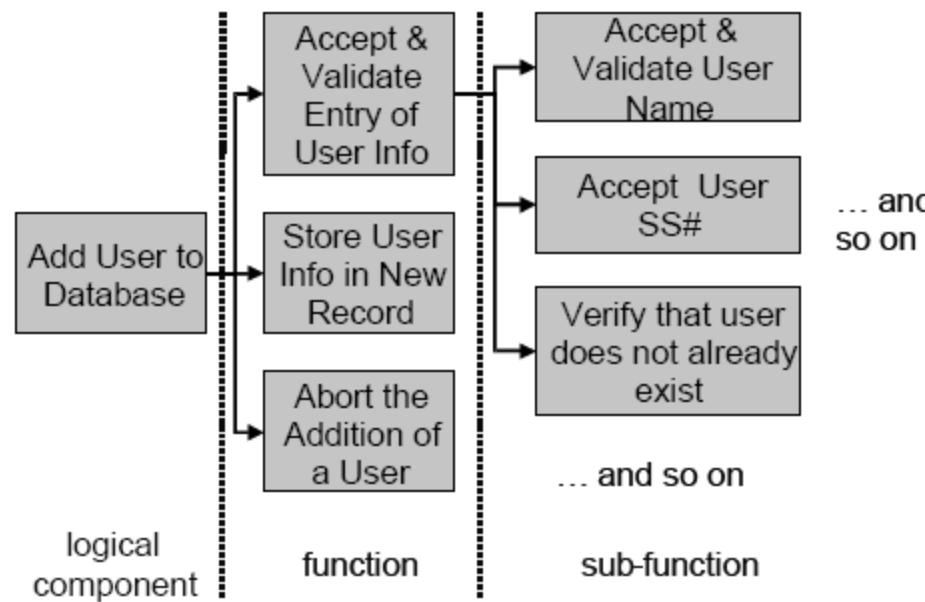
Các bước xây dựng đồ thị

- ❖ **Bước 1:** Phân chia hệ thống thành các vùng hoạt động
- ❖ **Bước 2:** Xác định các nguyên nhân (causes), kết quả (effects)
- ❖ **Bước 3:** Chuyển nội dung ngũ nghĩa trong đặc tả thành đồ thị
liên kết các cause và effects
- ❖ **Bước 4:** Chuyển đổi đồ thị thành bảng quyết định
- ❖ **Bước 5:** Thiết lập danh sách test case từ bảng quyết định.
Mỗi test case tương ứng với một cột trong bảng quyết định

Bước 1

❖ Phân chia hệ thống thành các vùng hoạt động

- Phân rã các yêu cầu chức năng thành danh sách các functions hay sub-functions



Bước 2

- ❖ **B 2.1:** Dựa vào đặc tả, xác định các causes và chỉ định mỗi causes này 1 định danh ID
 - Một cause có thể được xem như là 1 input conditions hoặc là đại diện của 1 lớp tương đương input conditions
- ❖ **B 2.2:** Dựa vào đặc tả, xác định effects hoặc sự thay đổi trạng thái của hệ thống và chỉ định mỗi effect 1 định danh ID
 - Effect có thể là output action, output condition hay là đại diện của 1 lớp tương đương output conditions

Xác định các causes, effects

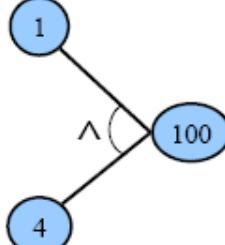
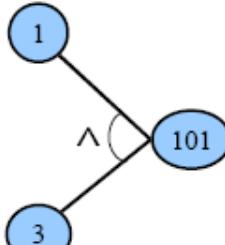
- ❖ Ví dụ: Xét đặc tả hệ thống tính phí bảo hiểm xe hơi
 - Đối với nữ < 65 tuổi, phí bảo hiểm là: 500\$
 - Đối với nam < 25 tuổi, phí bảo hiểm là: 3000\$
 - Đối với nam từ 25 đến 64, phí bảo hiểm là: 1000\$
 - Nếu tuổi từ 65 trở lên, phí bảo hiểm là: 1500\$
- Có 2 yếu tố xác định phí bảo hiểm: **giới tính và tuổi**

Causes (input conditions)	Effects (output conditions)
1. Sex is Male	100. Premium is \$1000
2. Sex is Female	101. Premium is \$3000
3. Age is <25	102. Premium is \$1500
4. Age is >=25 and < 65	103. Premium is \$500
5. Age is >= 65	

Bước 3

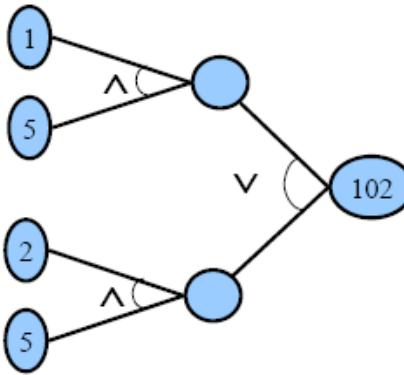
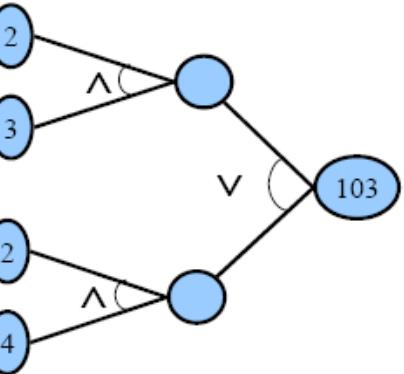
❖ Chuyển nội dung ngũ nghĩa trong đặc tả thành đồ thị liên kết các cause và effects

- CEG #1: Đối với nam từ 25 đến 64, phí bảo hiểm là 1000\$
- CEG #2: Đối với nam < 25 tuổi, phí bảo hiểm là 3000\$

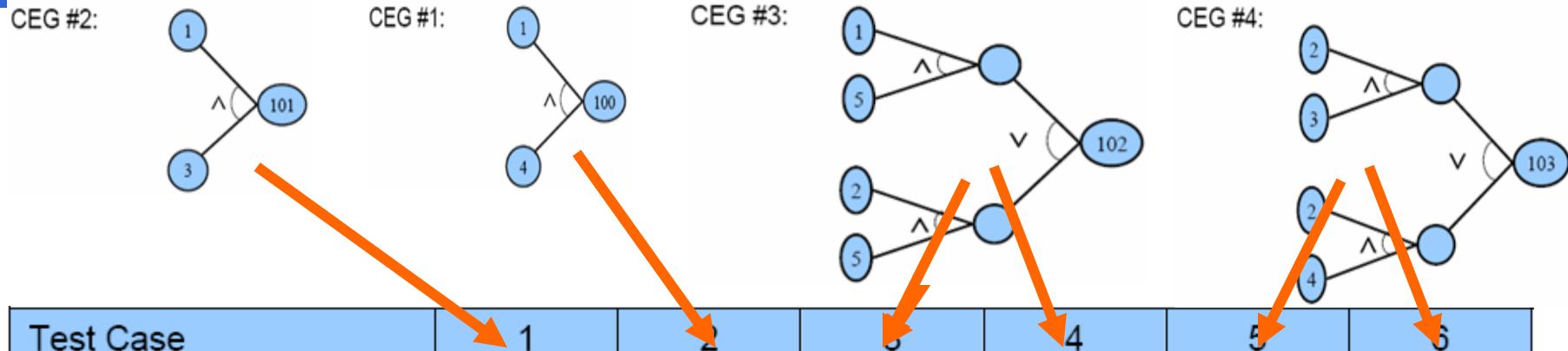
CEG	Interpretation
CEG #1: 	Causes: 1. Sex is Male and (Λ) 4. Age is ≥ 25 and < 65 Effect: 100: Premium is \$1000
CEG #2: 	Causes: 1. Sex is Male and (Λ) 3. Age is < 25 Effect: 101: Premium is \$3000

Bước 3

- ❖ CEG #3: Nếu tuổi từ 65 trở lên, phí bảo hiểm là: 1500\$
- ❖ CEG #4: Đối với nữ < 65 tuổi, phí bảo hiểm là: 500\$

CEG #3:  <pre>graph LR; 1((1)) --> C1(()); 2((2)) --> C2(()); 3((3)) --> C3(()); 4((4)) --> C4(()); C1 --- C5((5)); C2 --- C3 --- C5; C4 --- C5; C1 --- C2 --- C3 --- C5 --- 102((102)); C1 --- 102;</pre>	Causes: 1. Sex is Male and (^) 5. Age is ≥ 65 or (v) 2. Sex is Female and (^) 5. Age is ≥ 65 Effect: 102: Premium is \$1500
CEG #4:  <pre>graph LR; 2((2)) --> C2(()); 3((3)) --> C3(()); 4((4)) --> C4(()); 5((5)) --> C5(()); C2 --- C3; C4 --- C5 --- C2 --- C3 --- 103((103)); C4 --- C5 --- C2 --- C3 --- 103;</pre>	Causes: 2. Sex is Female and (^) 3. Age is < 25 or (v) 2. Sex is Female and (^) 4. Age is ≥ 25 and < 65 Effect: 103: Premium is \$500

Bước 4: Chuyển đổi đồ thị thành Bảng quyết định



Test Case	1	2	3	4	5	6
Causes:						
1 (male)	1	1	1	0	0	0
2 (female)	0	0	0	1	1	1
3 (<25)	1	0	0	0	1	0
4 (≥ 25 and < 65)	0	1	0	0	0	1
5 (≥ 65)	0	0	1	1	0	0
Effects:						
100 (Premium is \$1000)	0	1	0	0	0	0
101 (Premium is \$3000)	1	0	0	0	0	0
102 (Premium is \$1500)	0	0	1	1	0	0
103 (Premium is \$500)	0	0	0	0	1	1

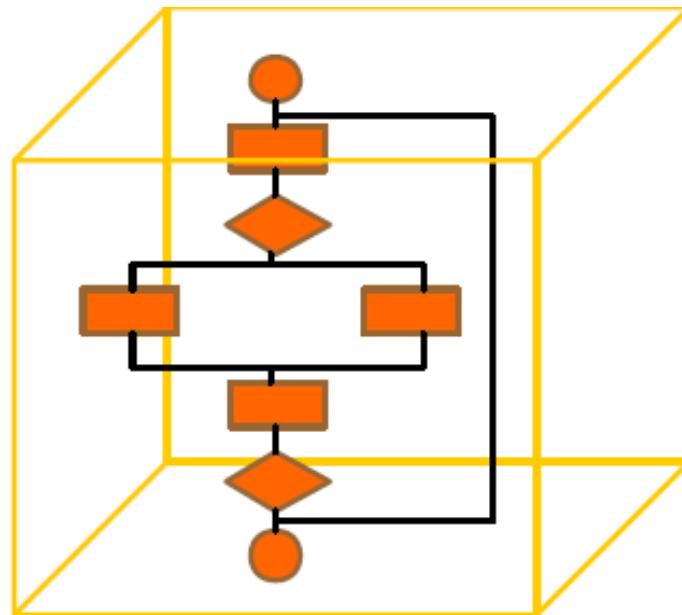
Bước 5: Lập danh sách test case từ Bảng quyết định

Test Case	1	2	3	4	5	6
Causes:						
1 (male)	1	1	1	0	0	0
2 (female)	0	0	0	1	1	1
3 (<25)	1	0	0	0	1	0
4 (>=25 and < 65)	0	1	0	0	0	1
5 (>= 65)	0	0	1	1	0	0
Effects:						
100 (Premium is \$1000)	0	1	0	0	0	0
101 (Premium is \$3000)	1	0	0	0	0	0
102 (Premium is \$1500)	0	0	1	1	0	0
103 (Premium is \$500)	0	0	0	0	1	1

Test Case #	Inputs (Causes)		Expected Output (Effects) Premium
	Sex	Age	
1	Male	<25	\$3000
2	Male	>=25 and < 65	\$1000
3	Male	>= 65	\$1500
4	Female	>= 65	\$1500
5	Female	<25	\$500
6	Female	>=25 and < 65	\$500

Chiến lược kiểm thử hộp trắng

- ❖ Thiết kế test case dựa vào cấu trúc nội tại bên trong của đối tượng cần kiểm thử
 - ❖ Đảm bảo tất cả các câu lệnh, các biểu thức điều kiện bên trong chương trình đều được thực hiện ít nhất một lần





Các kỹ thuật kiểm thử hộp trắng

- ❖ Basis Path Testing
- ❖ Control-flow/Coverage Testing
- ❖ Data-flow Testing



Basis Path Testing

- ❖ Được McCabe đưa ra vào năm 1976
- ❖ Là phương pháp thiết kế **test case** đảm bảo rằng tất cả các **independent path** trong một code module đều được thực thi ít nhất một lần
- ❖ **Independent path:** là bất kỳ path nào trong code mà bổ sung vào ít nhất một tập các lệnh xử lý hay một biểu thức điều kiện (Pressman 2001)
- ❖ Cho biết số lượng **test case tối thiểu** cần phải thiết kế khi kiểm thử một code module

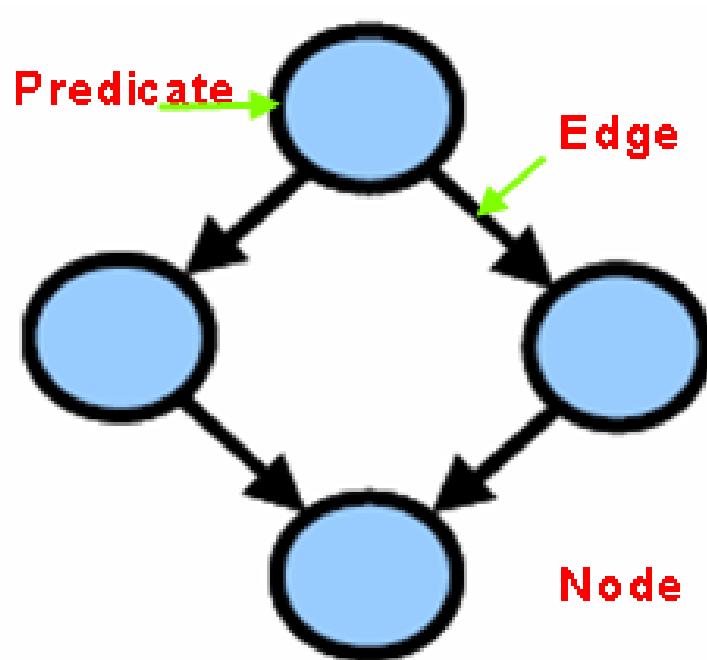


Các bước thực hiện

- ❖ Xây dựng đồ thị luồng điều khiển
- ❖ Tính toán độ phức tạp Cyclomatic
- ❖ Chọn ra tập path cơ sở cần test
- ❖ Phát sinh test case thực hiện kiểm tra từng path trong tập path cơ sở

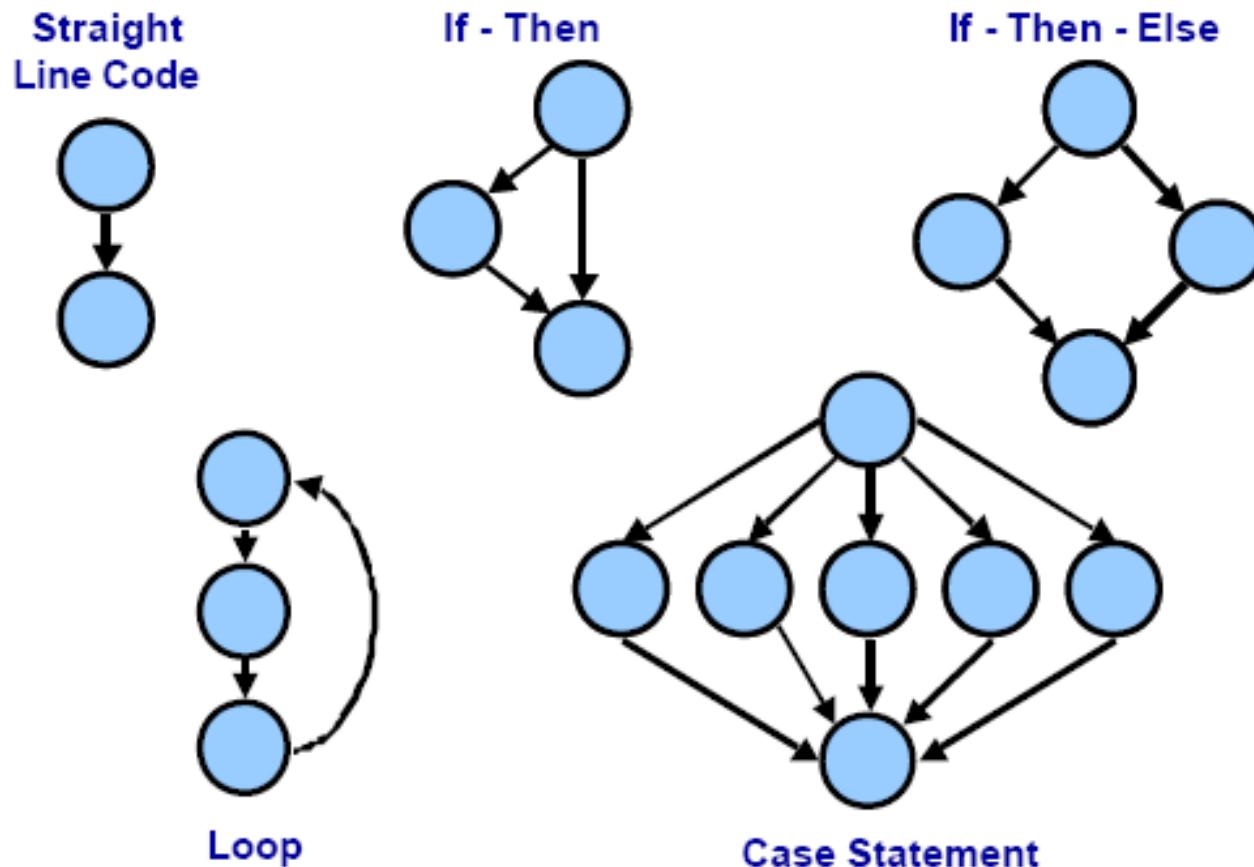
Xây dựng đồ thị luồng điều khiển

- ❖ Edge: đại diện cho một luồng điều khiển
- ❖ Node: đại diện cho một hoặc nhiều câu lệnh xử lý
- ❖ Predicate node: đại diện cho một biểu thức điều kiện



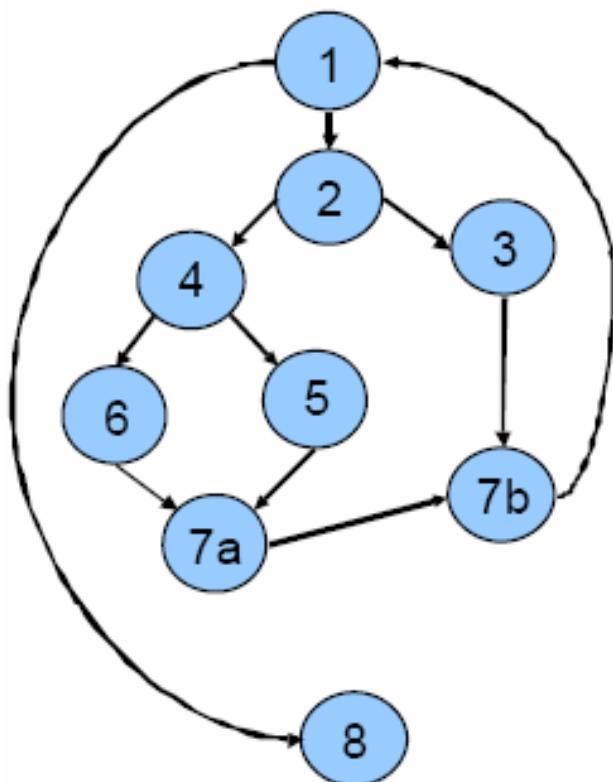
Xây dựng đồ thị luồng điều khiển

- ❖ Một số cấu trúc luồng điều khiển cơ bản



Xây dựng đồ thị luồng điều khiển

- ❖ Đồ thị luồng điều khiển từ một đoạn chương trình

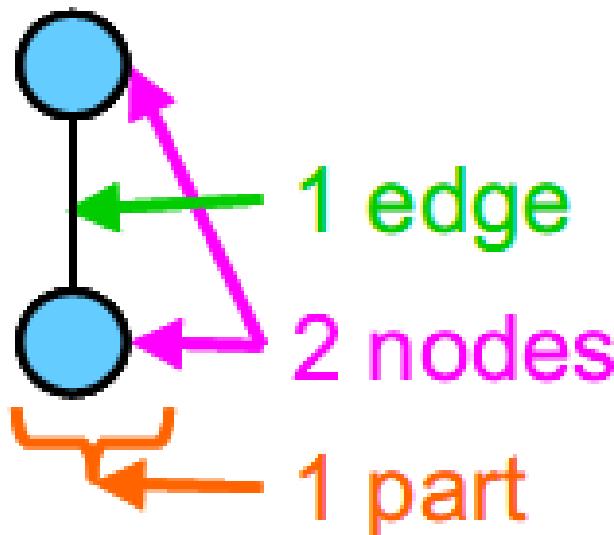


```
1. do while ( records remain )
   read record;
2.   if ( record field 1 = 0 ) then
3.     process record;
   store in buffer;
   increment counter;
4.   elseif ( record field 2 = 0 ) then
5.     reset record;
6.   else
     process record;
     store in file;
7a. endif;
7b. enddo;
8. end;
```

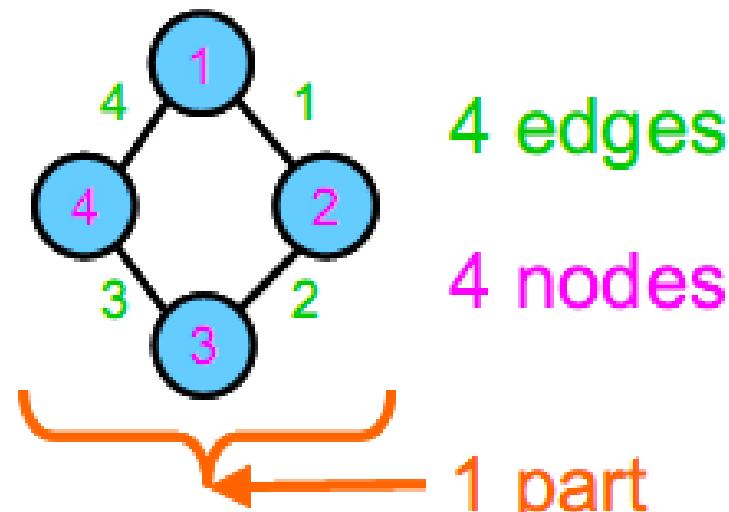
Tính toán độ phức tạp Cyclomatic

❖ Cách 1: Dựa trên công thức của McCabe

- $V(G) = \text{edges} - \text{nodes} + 2p$
- $p = \text{number of unconnected parts of the graph}$

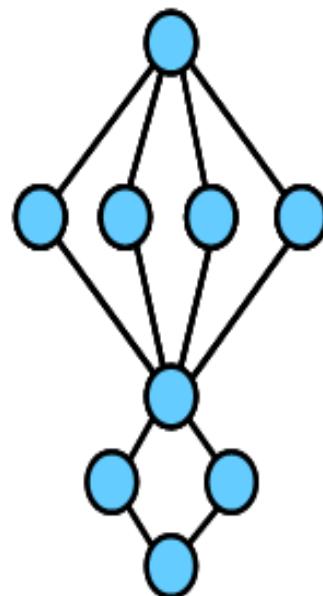


$$V(G) = 1 - 2 + 2 \times 1 = 1$$



$$V(G) = 4 - 4 + 2 \times 1 = 2$$

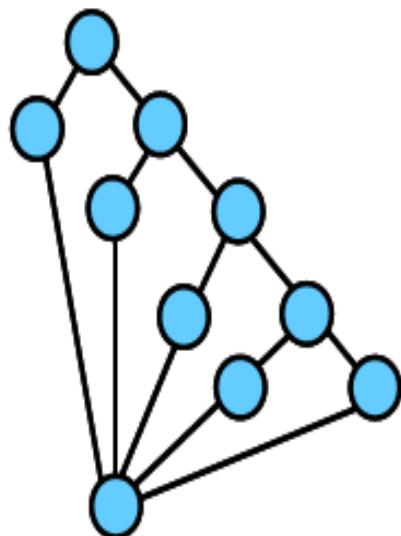
Tính toán độ phức tạp Cyclomatic



12 edges

9 nodes

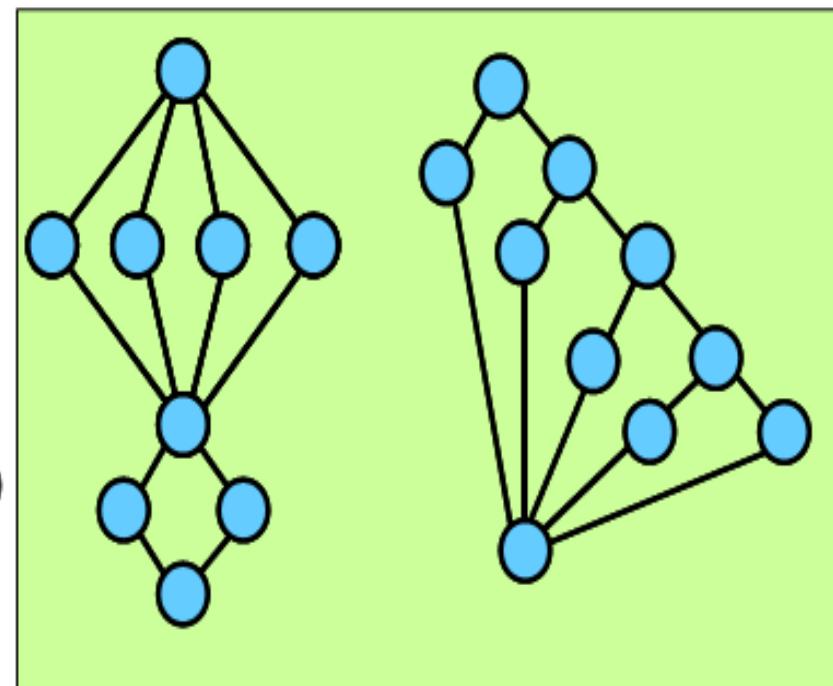
$$12 - 9 + 2(1) = 5$$



13 edges

10 nodes

$$13 - 10 + 2(1) = 5$$



25 edges

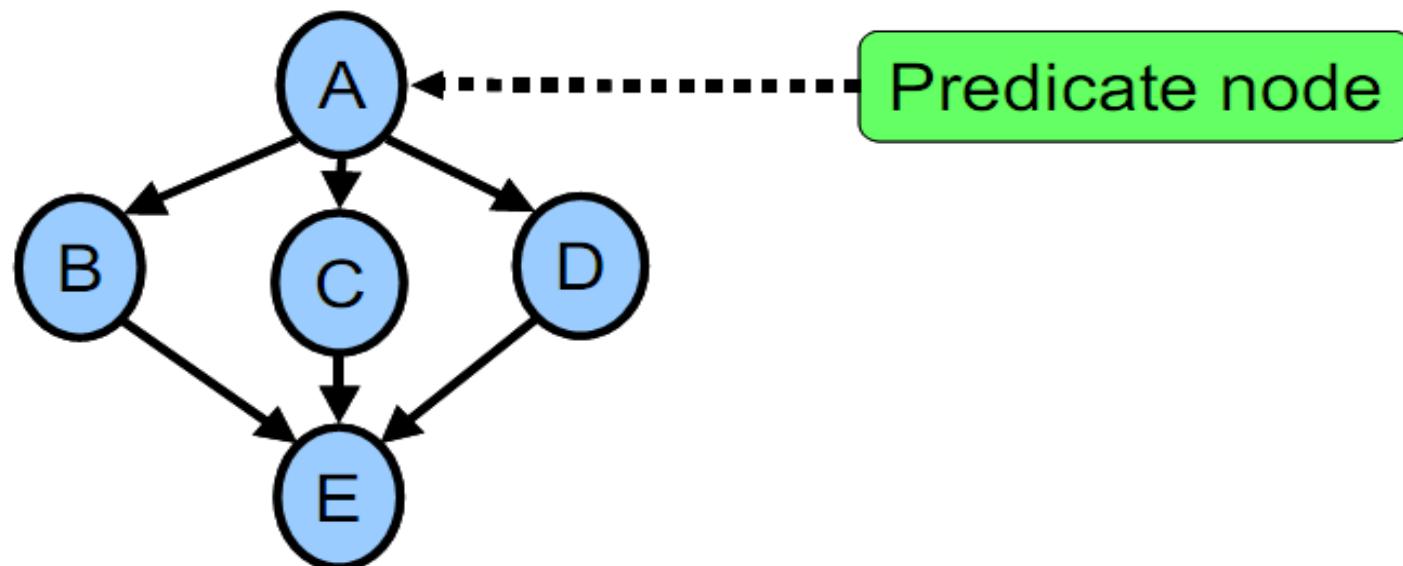
19 nodes

$$25 - 19 + 2(2) = 10$$

Tính toán độ phức tạp Cyclomatic

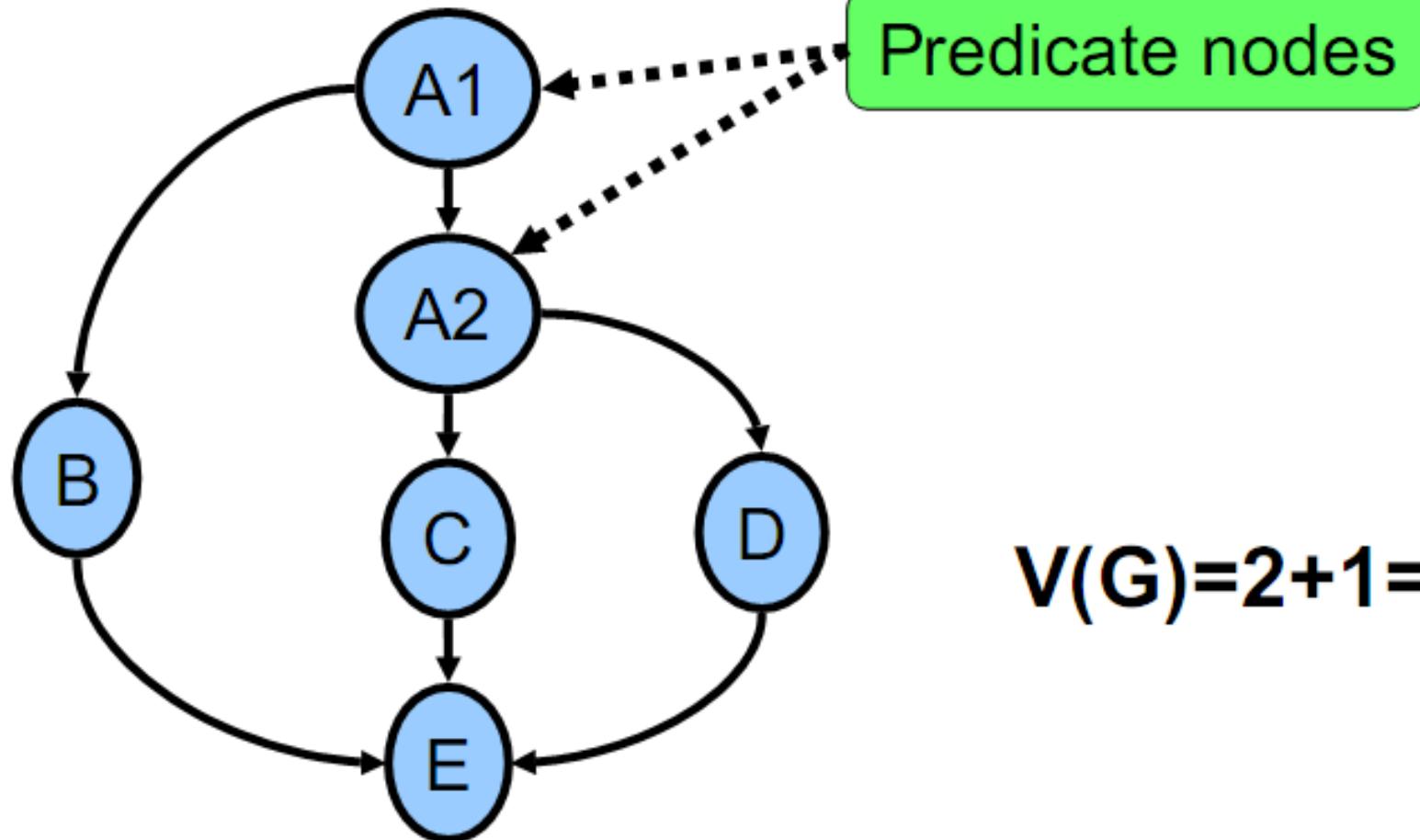
❖ Cách 2: Dựa vào số lượng Predicate Node

- $V(G) = \text{Number of Predicate Nodes} + 1$

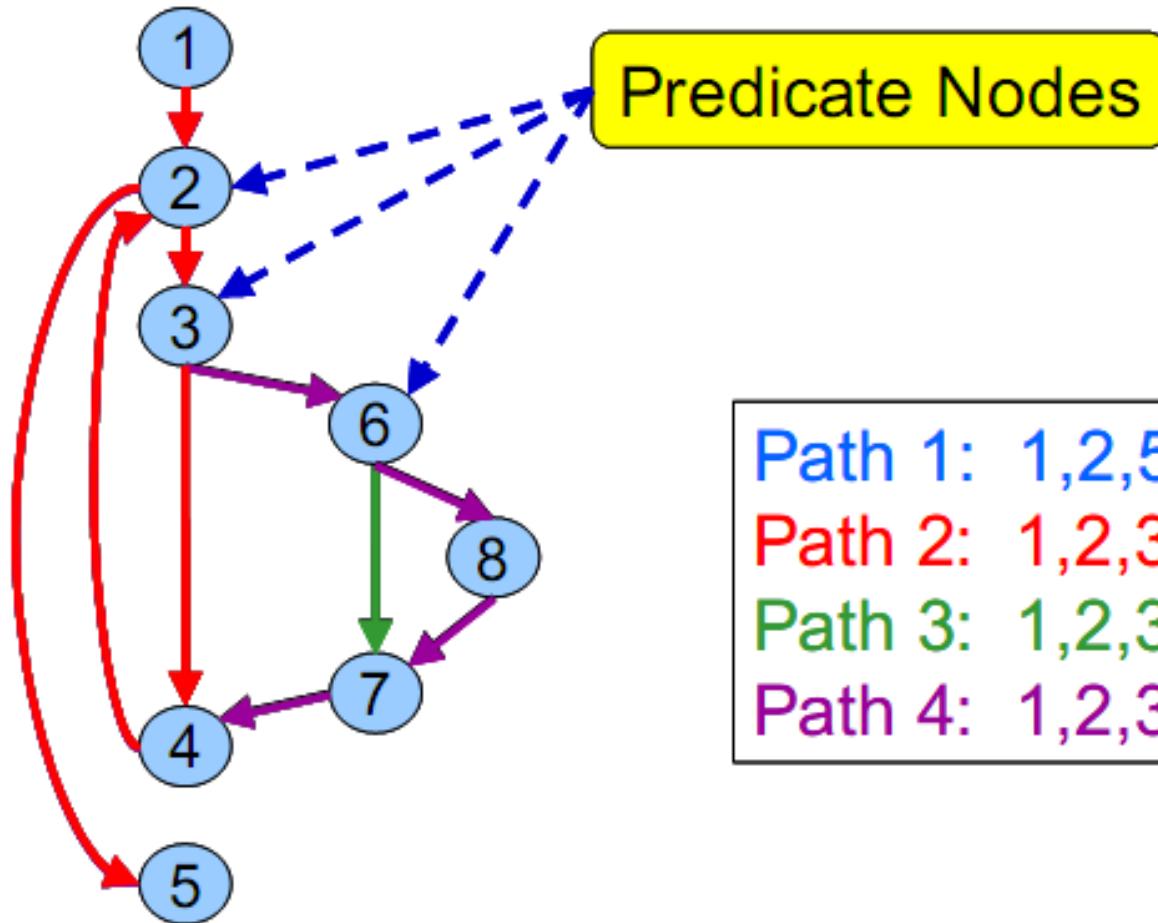


McCabe: $V(G) = 6 - 5 + 2(1) = 3$

Tính toán độ phức tạp Cyclomatic



Chọn ra tập path cơ sở cần test



Path 1: 1,2,5

Path 2: 1,2,3,4,2,5

Path 3: 1,2,3,6,7,4,2,5

Path 4: 1,2,3,6,8,7,4,2,5

Phát sinh test case

Test case 1	Path 1: 1,2,5
Test case 2	Path 2: 1,2,3,4,2,5
Test case 3	Path 3: 1,2,3,6,7,4,2,5
Test case 4	Path 4: 1,2,3,6,8,7,4,2,5



Control-flow/Coverage Testing

- ❖ Là kỹ thuật thiết kế test case đảm bảo “cover” được tất cả các câu lệnh, biểu thức điều kiện trong code module cần test
- ❖ Có bốn tiêu chí đánh giá độ bao phủ
 - Method Coverage (**phương thức**)
 - Statement Coverage (**câu lệnh**)
 - Decision/Branch Coverage (**biểu thức điều kiện**)
 - Condition Coverage (**biểu thức điều kiện đơn**)



Method Coverage

- ❖ Tỷ lệ phần trăm các **phương thức** trong chương trình được gọi thực hiện bởi các test case
- ❖ Test case cần phải đạt được 100% method coverage



Ví dụ - Method Coverage

❖ Xét đoạn chương trình

- ❖ Test case 1: foo (0,0,0,0,0)
- ❖ 100% method coverage

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if(a==0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a))) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```

Statement Coverage

- ❖ Tỷ lệ phần trăm các **câu lệnh** trong chương trình được gọi thực hiện bởi các test case
- ❖ Test case 1 thực hiện các lệnh từ 1→5 trong 12 câu lệnh đạt 42% Statement Coverage
- ❖ Để đạt 100% Statement Coverage
→Test case 2: **foo (1,1,1,1,1)**

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if (a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a))) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```



Decision/Branch Coverage

- ❖ Tỷ lệ phần trăm các **biểu thức điều kiện** trong chương trình được ước lượng giá trị trả về (true, false) khi thực thi các test case
- ❖ Một biểu thức điều kiện (cho dù là single hay complex) phải được kiểm tra trong cả hai trường hợp giá trị của biểu thức là **true** hay **false**
- ❖ Đối với các hệ thống lớn, thường chỉ đạt từ 75% → 85% **độ bao phủ**

Decision/Branch Coverage

Line #	Predicate	True	False
3	(a == 0)	Test Case 1 foo(0, 0, 0, 0, 0) return 0	Test Case 2 foo(1, 1, 1, 1, 1) return 1
7	((a==b) OR ((c == d) AND bug(a)))	Test Case 2 foo(1, 1, 1, 1, 1) return 1	

→ Đạt 75% coverage

→ Test case 3: **foo (1,2,1,2,1)** → 100% coverage

Condition Coverage

- ❖ Tỷ lệ phần trăm các biểu thức điều kiện đơn trong biểu thức điều kiện phức của chương trình được ước lượng giá trị trả về (true, false) khi thực thi các test case
- ❖ Ví dụ: 50% coverage

Predicate	True	False
(a==b)	Test Case 2 foo(1, 1, x, x, 1) return value 0	Test Case 3 foo(1, 2, 1, 2, 1) division by zero!
(c==d)		Test Case 3 foo(1, 2, 1, 2, 1) division by zero!

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if(a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a) )) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```

Condition Coverage

- ❖ Thiết kế thêm Test case 4, 5 để đạt 100% coverage

Predicate	True	False
(a==b)	Test Case 2 foo(1, 1, x, x, 1) return value 0	Test Case 3 foo(1, 2, 1, 2, 1) division by zero!
(c==d)	Test Case 4 foo(1, 2, 1, 1, 1) return value 1	Test Case 3 foo(1, 2, 1, 2, 1) division by zero!
bug(a)	Test Case 4 foo(1, 2, 1, 1, 1) return value 1	Test Case 5 foo(3, 2, 1, 1, 1) division by zero!

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if (a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a) )) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```



Data-flow Testing

- ❖ Là kỹ thuật thiết kế **test case** dựa vào việc khảo sát **sự thay đổi trạng thái** trong chu kỳ **sống** của các biến trong chương trình
- ❖ **Ví dụ:** Một số pattern lỗi thường gặp
 - Sử dụng biến mà chưa khai báo
 - Sử dụng biến đã hủy trước đó
 - ...



Hệ thống ký hiệu trạng thái dữ liệu

Hệ thống ký hiệu

d	defined, created, initialized
k	killed, terminated, undefined
u	used c – used in a computation (sử dụng trong biểu thức tính toán) p – used in a predicate (sử dụng trong các biểu thức điều kiện)
$\sim x$	Cho biết trước khi tất cả hành động liên quan đến x
$x\sim$	Cho biết tất cả hành động không có thông báo liên quan đến x

Một số ví dụ

- ❖ **v = expression**
 - c – use của các biến trong biểu thức
 - definition của v
- ❖ **read (v1, v2, ..., vn)**
 - definitions của v1, ..., vn
- ❖ **write (v1, v2, ..., vn)**
 - c - uses của v1, ..., vn
- ❖ **method call: P (c1, ..., cn)**
 - definition của mỗi tham số
- ❖ **While B do S**
 - p – use của mỗi biến trong biểu thức điều kiện

Ví dụ

```
1. read (x, y);  
2. z = x + 2;  
3. if (z < y)  
4.     w = x + 1;  
    else  
5.     y = y + 1;  
6. print (x, y, w, z);
```

<i>Def</i>	<i>C-use</i>	<i>P-use</i>
x, y		
z	x	
w	x	z, y
y	y	
	x, y, w, z	



Các chiến lược thiết kế test case

- ❖ All-du paths (ADUP)
- ❖ All-Uses (AU)
- ❖ All-p-uses (APU)
- ❖ All-c-uses (ACU)
- ❖ All-p-uses / Some-c-uses (APU+C)
- ❖ All-c-uses / Some-p-uses (ACU+P)
- ❖ All-definition (AD)

Ví dụ

```
public static double calculateBill (int usage)
{
    double bill = 0;

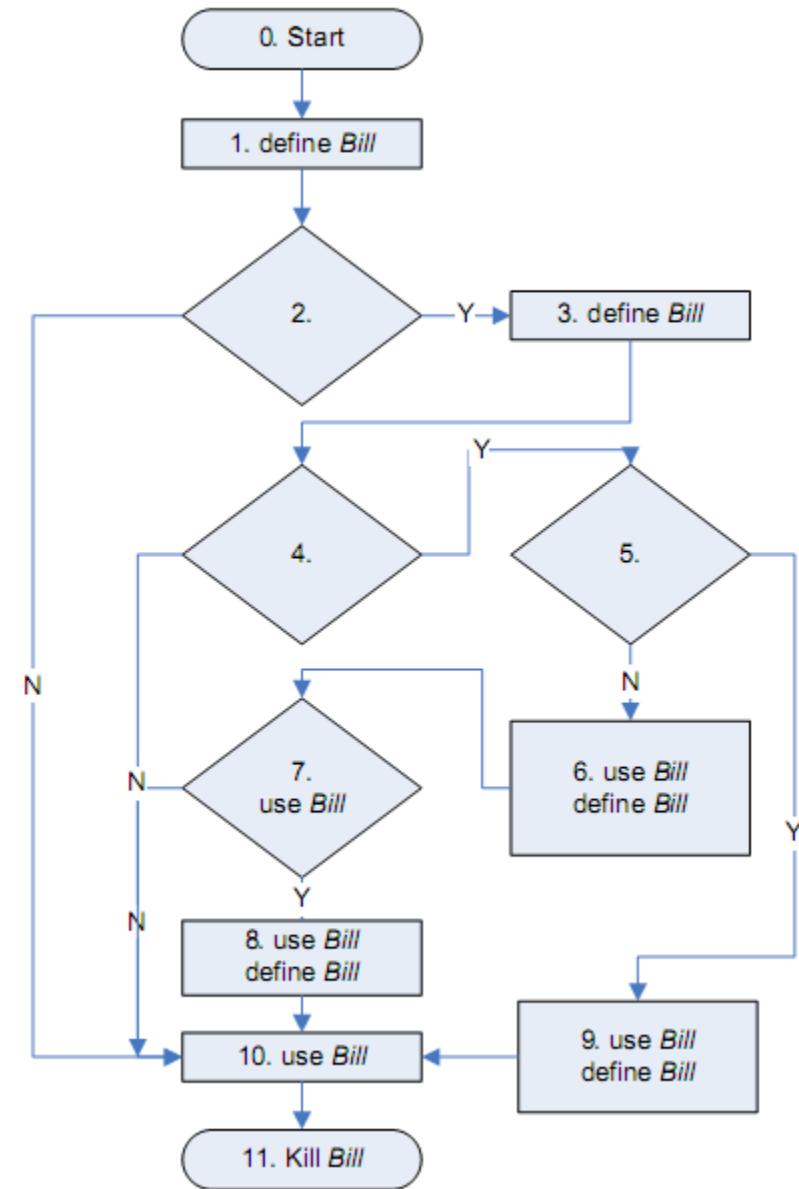
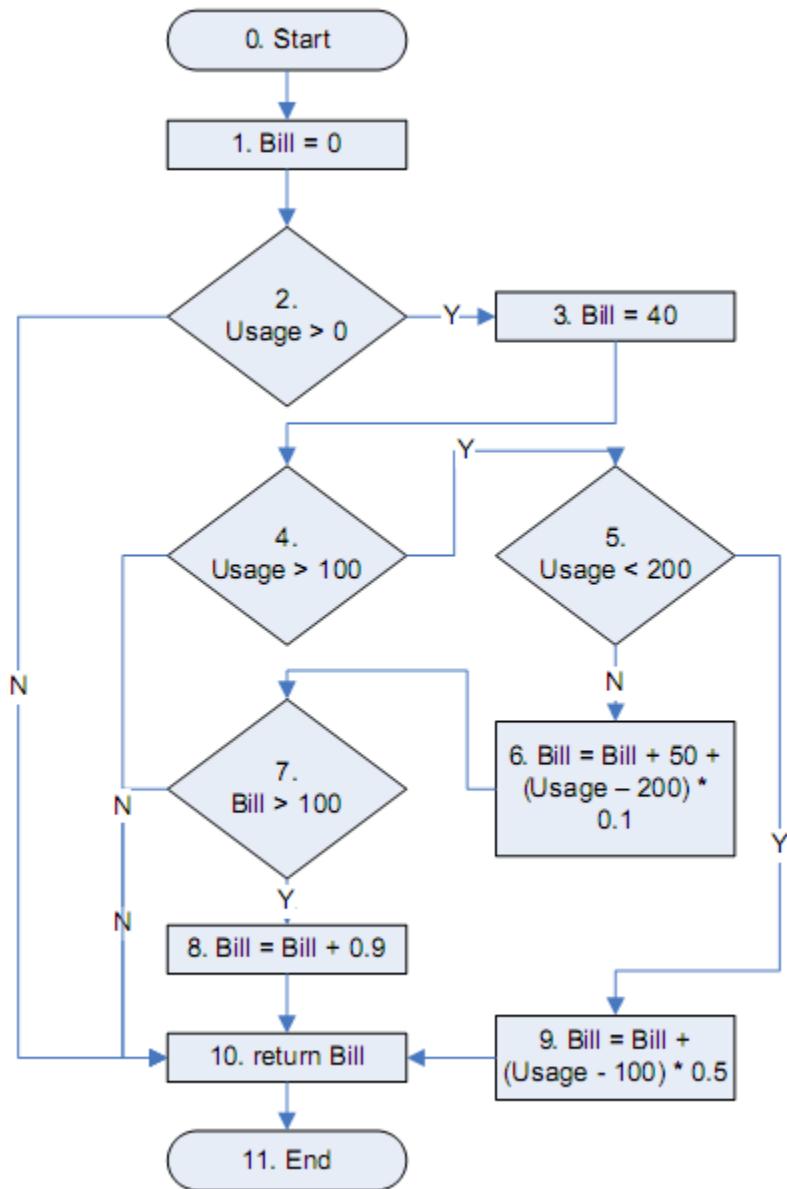
    if(usage > 0)
    {
        bill = 40;
    }

    if(usage > 100)
    {
        if(usage <= 200)
        {
            Bill = bill + (usage - 100) * 0.5;
        }
        else
        {
            Bill = bill + 50 + (usage - 200) * 0.1;

            if(bill >= 100)
            {
                bill = bill * 0.9;
            }
        }
    }

    return bill;
}
```

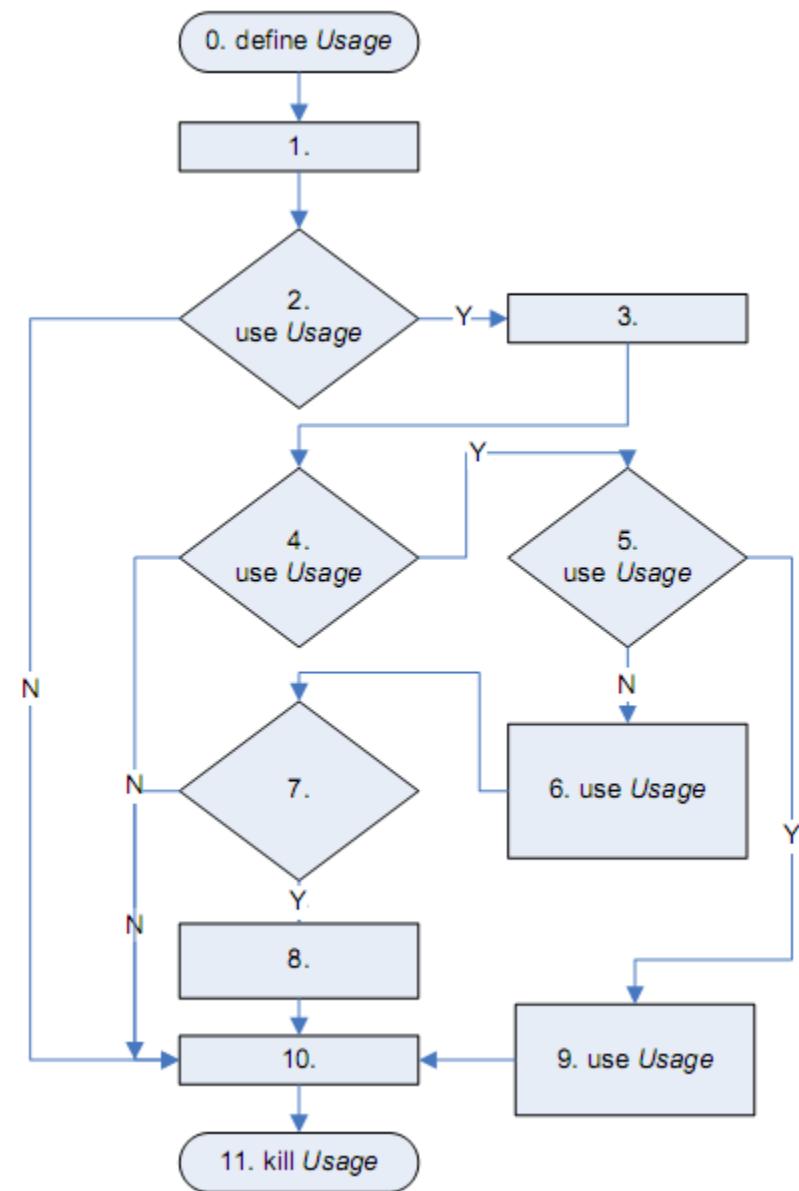
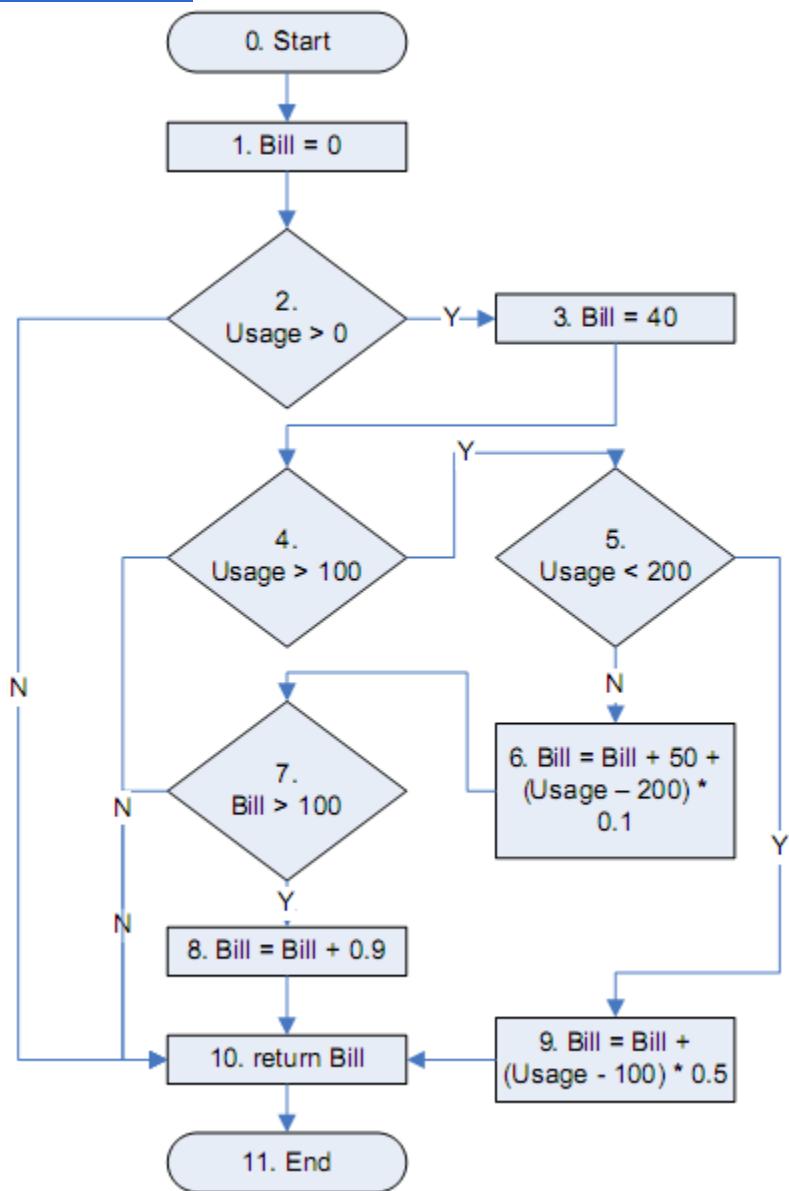
Xét biến “Bill”



Bảng mô tả biến “Bill”

Anomaly	Explanation
~d	0-1 Allowed. Normal case
dd	0-1-2-3 Potential bug. Double definition.
du	3-4-5-6 Allowed. Normal case.
ud	6 Allowed. Data is used and then redefined.
uk	10-11 Allowed.
dd	1-2-3 Potential bug. Double definition.
uu	7-8 Allowed. Normal case.
k~	11 Allowed. Normal case.

Xét biến “Usage”



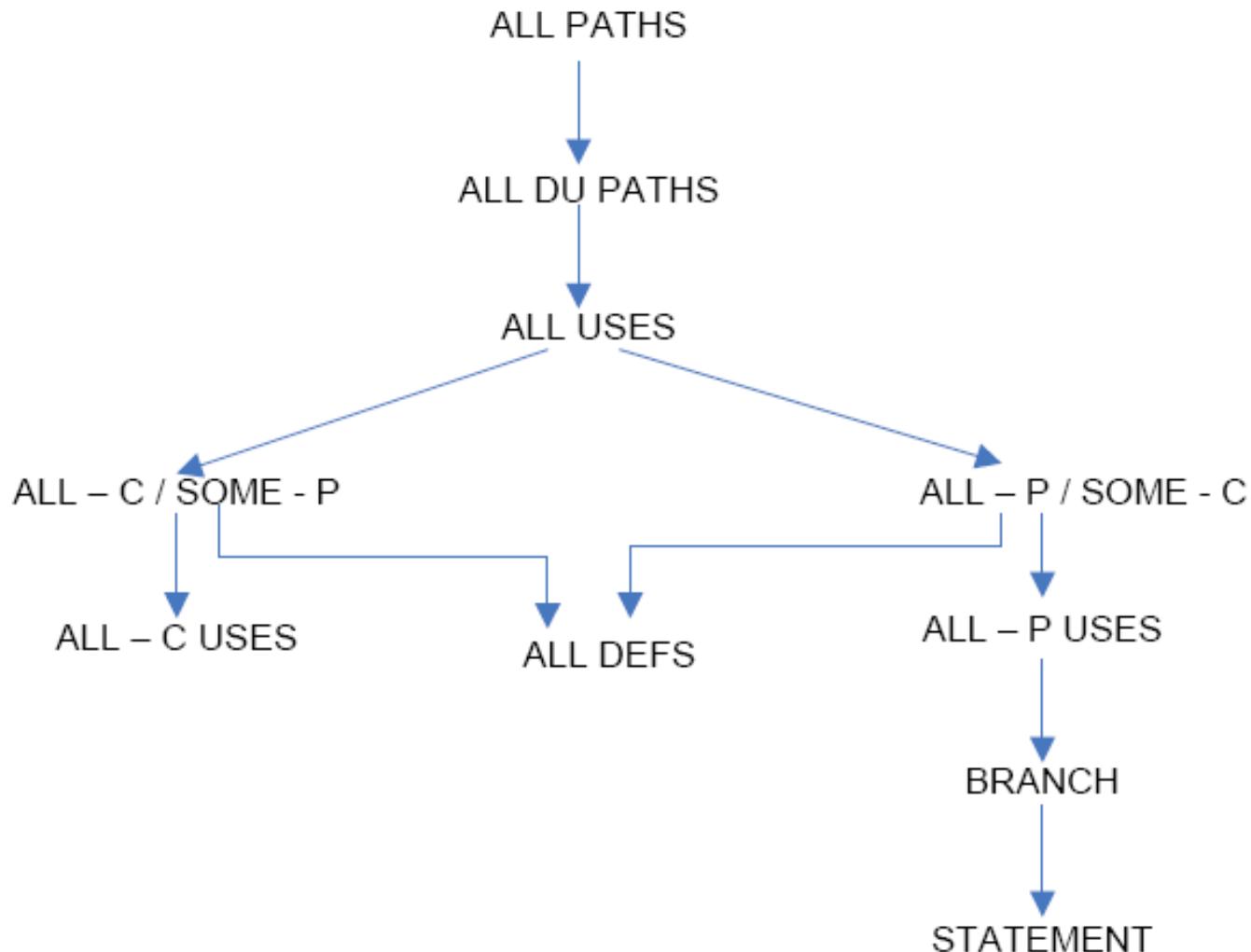
Bảng mô tả biến “Usage”

Anomaly	Explanation
~d	0
du	0-1-2
uk	9-10-11
uu	5-6
k~	11

Data-flow testing paths for each variable

Strategy	Bill	Usage	Strategy	Bill	Usage
All uses (AU)	3-4-5-6 6-7 6-7-8 8-10 3-4-5-9	0-1-2 0-1-2-3-4 0-1-2-3-4-5 0-1-2-3-4-5-6 0-1-2-3-4-5-9	All p – use/ some c (APU + C)	1-2-3-4-5-6-7 3-4-5-6-7 6-7 8-10 9-10	0-1-2 0-1-2-3-4 0-1-2-3-4-5
All p – uses (APU)	1-2-3-4-5-6-7 3-4-5-6-7 6-7	0-1-2 0-1-2-3-4 0-1-2-3-4-5	All c – use/ some p (ACU + P)	1-2-10 3-4-5-6 3-4-5-9 6-7-8 8-10 9-10	0-1-2-3-4-5-6 0-1-2-3-4-5-9
All c – uses (ACU)	1-2-10 3-4-5-6 3-4-5-9 3-4-10 6-7-8 6-7-10 8-10 9-10	0-1-2-3-4-5-6 0-1-2-3-4-5-9	All du (ADUP)	(ACU+P) + (APU+C)	(ACU+P) + (APU+C)
			All definition (AD)	1-2-10 3-4-5-6 6-7 8-10 9-10	0-1-2

Mối quan hệ giữa các chiến lược data-flow test





Các công cụ hỗ trợ kiểm thử

- ❖ Các công cụ hỗ trợ quản lý quá trình kiểm thử
- ❖ Các công cụ hỗ trợ thực hiện các kỹ thuật kiểm thử
 - Công cụ kiểm thử hiệu năng (Performance)
 - Công cụ kiểm thử chức năng (Functional)
 - Công cụ kiểm thử bảo mật (Security)
 - Công cụ kiểm thử đơn vị (UnitTesting)
 - ...



Các công cụ hỗ trợ quản lý quy trình kiểm thử phần mềm (1)

❖ Các đối tượng cần quản lý của 1 công cụ kiểm thử PM

- Project
- User
- User Role
- Requirement
- Release: Phiên bản của project.
- Test Plan: Kế hoạch test.
- Test types: Các loại test.
- Test cases: Các trường hợp test
- Teststep: Các bước thực hiện cho mỗi test case
- Result: Kết quả thực thi test.
- Bug: Lỗi
- Reports: Các thông báo về tình trạng của tiến trình: Tình trạng lỗi, tiến triển của công việc: ...
- Các tài liệu hướng dẫn sử dụng chương trình (Help)



Các công cụ hỗ trợ quản lý quy trình kiểm thử phần mềm (2)

❖ Các chức năng cần phải có

- Quản lý project.
- Quản lý User.
- Phân quyền User.
- Quản lý requirement theo phiên bản.
- Quản lý release.
- Quản lý các thành phần của release: build, component,..
- Quản lý testplan.
- Quản lý testcase.
- Cập nhật kết quả cho test case.
- Cập nhật tình trạng lỗi.
- Thống kê lỗi cho mỗi release hoặc mỗi thành phần của release.
- Tự động cập nhật kết quả kiểm thử



Các công cụ hỗ trợ quản lý quy trình kiểm thử phần mềm (3)

No	Name	Desc	REQ	Download
1	TestLink		Apache, MySQL, PHP	48797
2	Fitnessse		Mac, Windows, POSIX	24475
3	QATraq		Windows, BSD, Linux, SunOS/Solaris	21992
4	Bugzilla Test Runner		Bugzilla 2.16.3 or above	17291
5	rth		All 32-bit MS Windows (95/98/NT/2000/XP), All POSIX (Linux/BSD/UNIX-like OSes), IBM AIX	9563
6	TestMaster		Linux, Apache, PostgreSQL	6728
7	TCW		Any (PHP/SQL/Apache)	4488
8	Tesly		OS Independent	3327
9	qaProjectManager		Platform Independent	3133
10	Testitool		Apache, PHP, MySQL	701



Công cụ kiểm thử hiệu năng

- ❖ Là một dạng kiểm tra tự động nhằm tìm ra những điểm “thắt cổ chai” của phần mềm, giúp cho người phát triển có những thay đổi thích hợp để tăng khả năng thực thi, tốc độ xử lý của phần mềm
- ❖ Giúp người kiểm tra xác định được những thông số ngưỡng của phần mềm, đề ra tiêu chuẩn cho những lần kiểm tra sau
- ❖ Thường được áp dụng đối với các PM được triển khai trên môi trường nhiều người dùng (ví dụ: ứng dụng web)
- ❖ Kết quả mong đợi của việc kiểm thử hiệu năng phải được định nghĩa một cách rõ ràng
- ❖ Ví dụ:
 - Số kết nối (session) đồng thời mà server có thể phục vụ
 - Thời gian (bao nhiêu phút/giây) mà trình duyệt nhận được kết quả từ server
 -

Công cụ kiểm thử hiệu năng

No	Name	Requirements	Download
1	OpenSTA	Windows 2000, NT4 and XP	251965
2	Grinder	OS Independent	156458
3	TPTEST	MacOS/Carbon and Win32	108036
4	Database Opensource Test Suite	Linux, POSIX	103484
5	Sipp	Linux/Unix/Win32-Cygwin	102111
6	WebLOAD	32-bit MS Windows (NT/2000/XP), Linux, Windows Server 2003	39401
7	OpenWebLoad	Linux, DOS	31204
8	Hammerhead 2 - Web Testing Tool	Hammerhead has been used with Linux, Solaris and FreeBSD.	24814
9	Dieseltest	Windows	14618
10	DBMonster	OS Independent	13710



Các công cụ hỗ trợ kiểm thử đơn vị

- ❖ Có rất nhiều công cụ kiểm thử đơn vị được viết bằng nhiều ngôn ngữ khác nhau
 - ADA
 - C++
 - HTML
 - Java
 - .NET
 - Pert
 - PHP
 - SQL
 - XML
 - Ruby
 - ...

Các công cụ hỗ trợ kiểm thử đơn vị

No	Name	Requirements	Download
1	JUnit	OS Independent	2151874
2	Findbugs	JRE (or JDK) 1.4.0 or later	379779
3	PMD	JDK 1.3 or higher	344688
4	Checkstyle	OS Independent	216780
5	EclEmma	Eclipse	209153
6	Dbunit	JUnit	129300
7	StrutsTestCase for JUnit v1.9.5	OS Independent	106860
8	Emma	Java	59435
9	MockObjects	OS independent	55457
10	JUnitEE	JUnit	54618

Các công cụ hỗ trợ kiểm thử đơn vị

No	Name	Requirements	Download
1	NUnit	Windows NT/2000	1061875
2	NUnitAsp	Windows NT/2000	72724
3	NUnit Addin for Visual Studio.NET	Windows	58588
4	NUnitForms	Windows NT/2000	46880
5	csUnit	csUnit has been tested using the Microsoft .NET framework 1.0 Service Pack 2 runtime on an Intel-compatible platform.	31483
6	NCover	All 32-bit MS Windows (95/98/NT/2000/XP)	14264
7	VSNUnit	All 32-bit MS Windows (95/98/NT/2000/XP)	8763
8	dotUnit	All 32-bit MS Windows (95/98/NT/2000/XP)	6230
9	.NETUnit	OS Independent (Written in an interpreted language)	5558
10	ASPUnit	Microsoft Internet Information Server 5.0 or 5.1	5197



Một số công cụ hỗ trợ kiểm thử chức năng

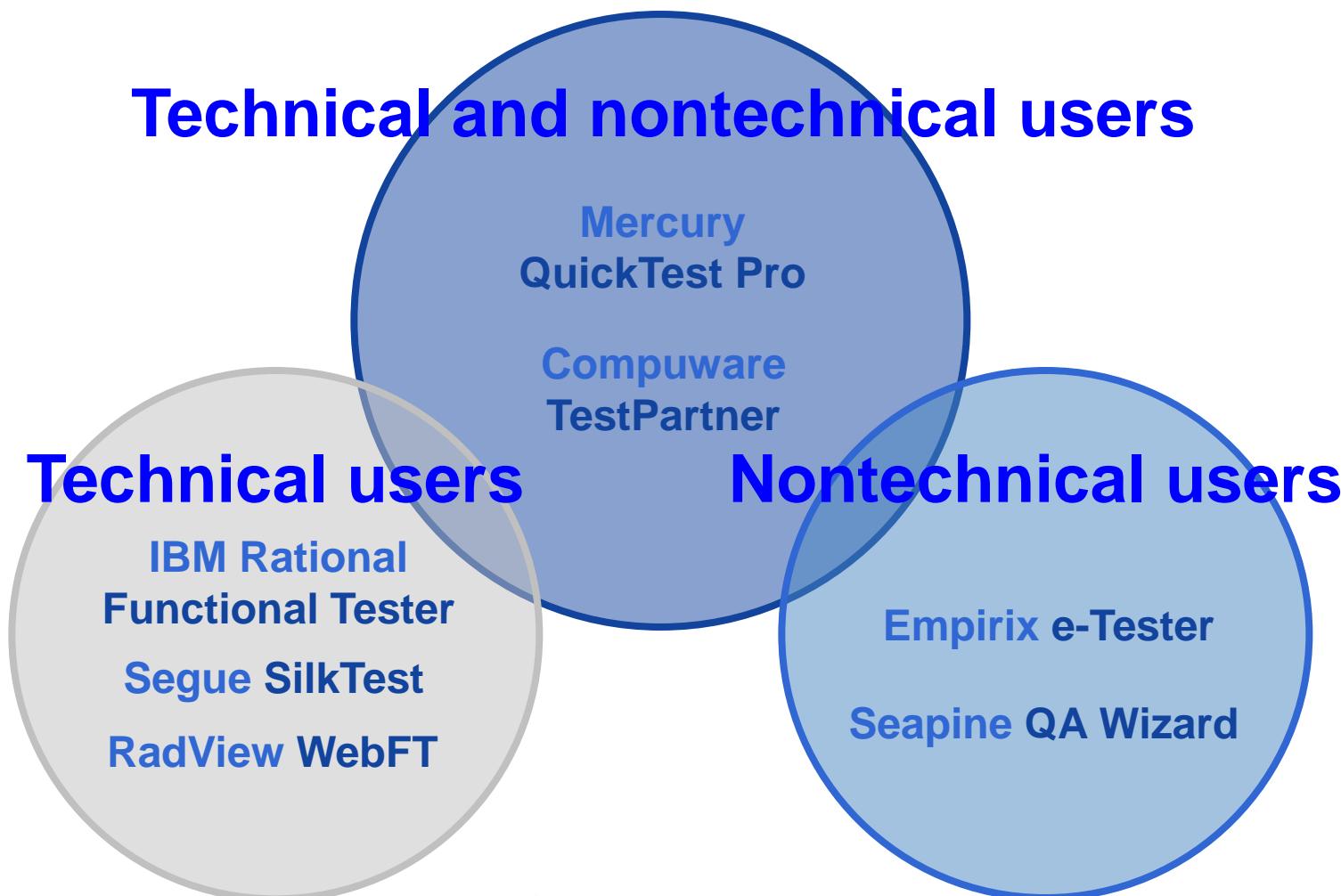
No	Name	Desc	Req	Download
1	<u>Software Testing Automation Framework (STAF)</u>		Windows, Linux, Solaris, AS/400, AIX, HP-UX, Irix	212018
2	<u>soapui</u>		Java 1.5	178985
3	<u>Linux Test Project</u>		Linux	103484
4	<u>jWebUnit</u>		OS Independent	56526
5	<u>Abbot Java GUI Test Framework</u>		TBC	56118
6	<u>Software Automation Framework Support</u>		All 32-bit MS Windows (95/98/NT/2000/XP)	43735
7	<u>Jameleon</u>		OS Independent, JDK 1.4 or higher	43507
8	<u>WebInject</u>		Windows, OS Independent, Linux	40891
9	<u>Marathon</u>		Java 1.3 or later	30328
10	<u>Solex</u>		Eclipse 2.1 or above	29591



Các công cụ kiểm thử thương mại

Vendor	Tool	Name of testing suite or companion tools
Compuware	TestPartner	QACenter Enterprise Edition+
Empirix	e-Tester	e-TEST suite
IBM Rational	Functional Tester	Test Manager, Manual Tester, Performance Tester
Mercury	QuickTest Professional	Quality Center
RadView	WebFT	TestView Suite
Seapine	QA Wizard	TestTrack Pro
Segue	SilkTest	SilkCentral, SilkPerformer

Các công cụ kiểm thử thương mại





Tài liệu tham khảo

- ❖ Software Testing, A Craftsman's Approach, Paul C.Jorgensen
- ❖ Practical Software Testing, EleneBurnstein
- ❖ Slides: Software Testing ISEB Foundation Certificate Course
- ❖ Slides: Software Testing, Dr. Balla Katalin
- ❖ Slide: Equivalence Class Testing, Prof. Schlingloff & Dr. M Roggenbach
- ❖ Slide: Decision Table Based Testing, Neelam Gupta, The University of Arizona Tucson, Arizona, USA
- ❖ Object Oriented Testing, Ali Kamandi, Sharif University of Technology

