

## CÔNG NGHỆ PHẦN MỀM

### Chương 8

# Kiểm thử phần mềm

# Nội dung

1. Chiến lược kiểm thử (Testing Strategy)
2. Kỹ thuật kiểm thử phần mềm (Software Testing Techniques)

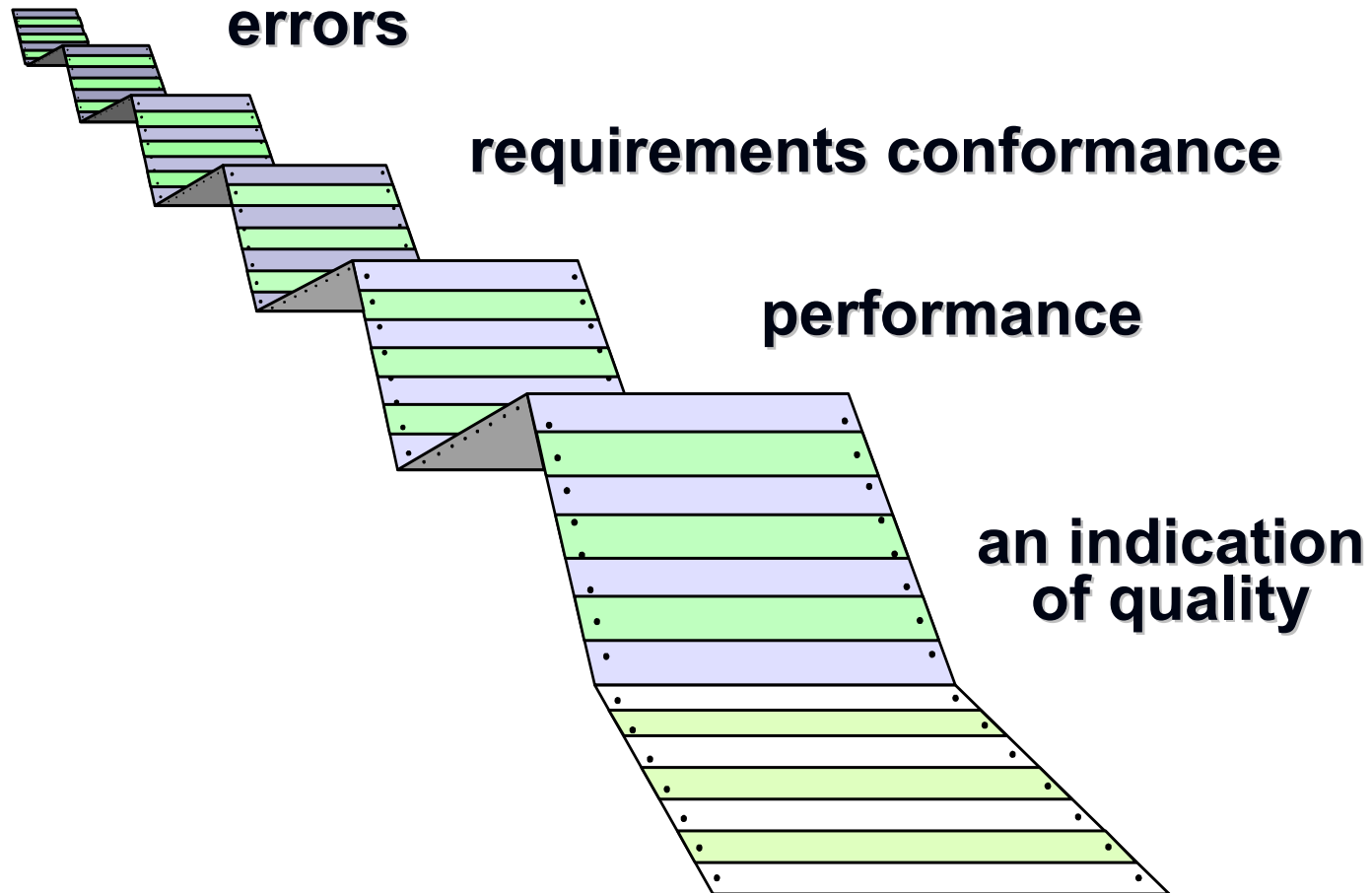
# Kiểm chứng và thẩm định (V&V)

- Kiểm chứng và thẩm định bao gồm kiểm thử phần mềm
- Kiểm chứng (Verification): “Chúng ta đang xây dựng sản phẩm theo đúng cách”
  - Phần mềm phải phù hợp với đặc tả của nó
- Thẩm định (Validation): “Chúng ta đang xây dựng sản phẩm đúng”
  - Phần mềm phải thực hiện những gì người dùng thật sự cần

# Kiểm thử phần mềm

Testing is the process of exercising a program with the specific intent of finding errors prior to (trước khi) delivery to the end user.

# What Testing Shows

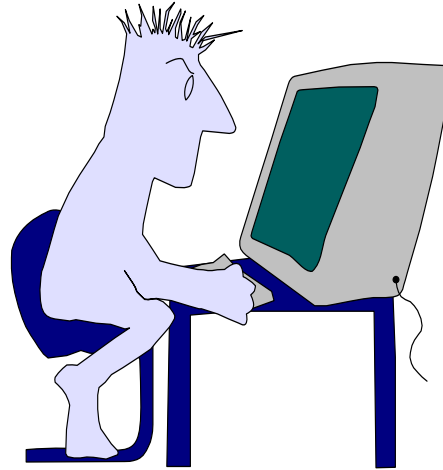


# Who Tests the Software?



***developer***

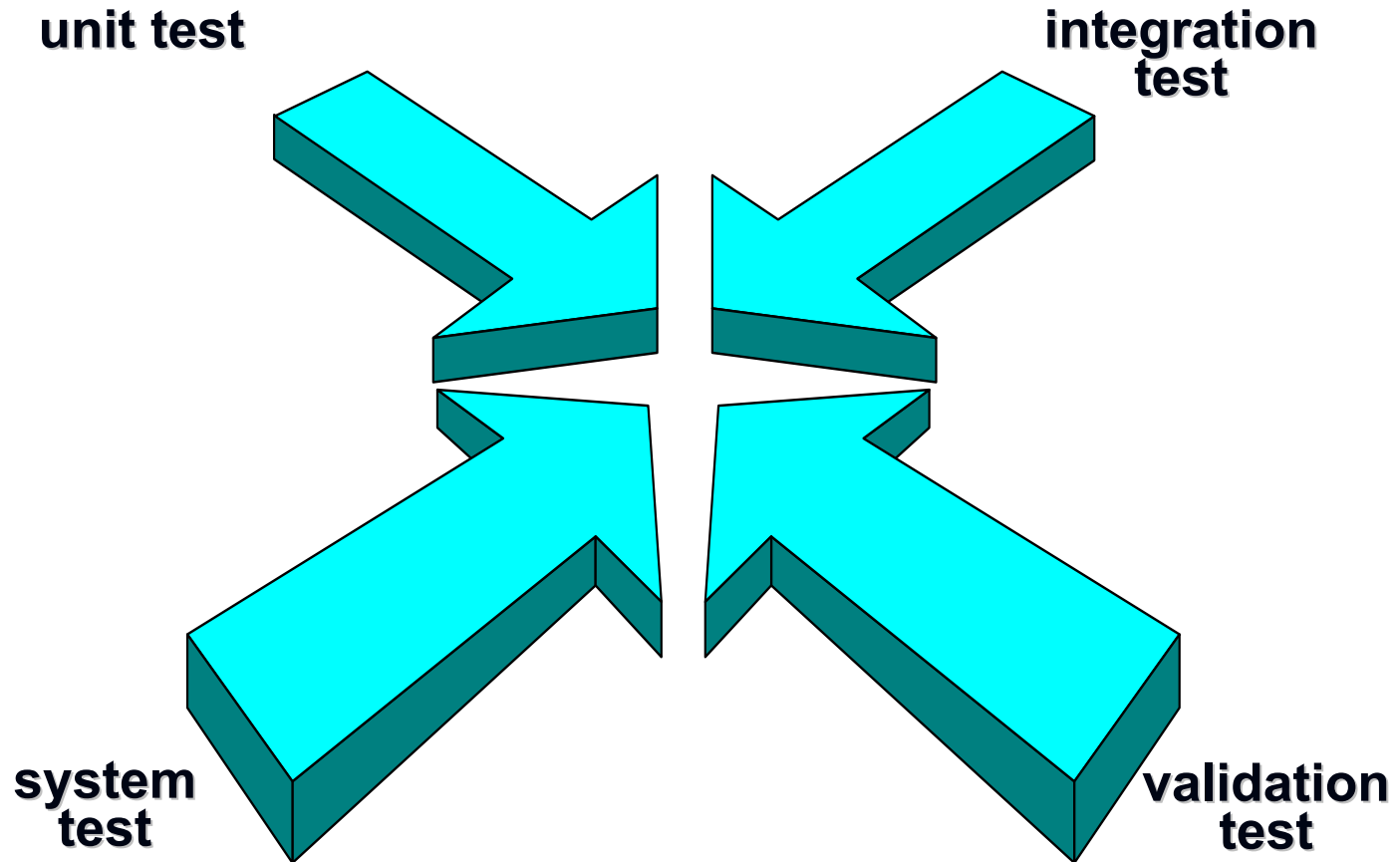
**Understands the system  
but, will test "gently"  
and, is driven by "delivery"**



***independent tester***

**Must learn about the system,  
but, will attempt to break it  
and, is driven by quality**

# 1. Chiến lược kiểm thử



# Chiến lược kiểm thử

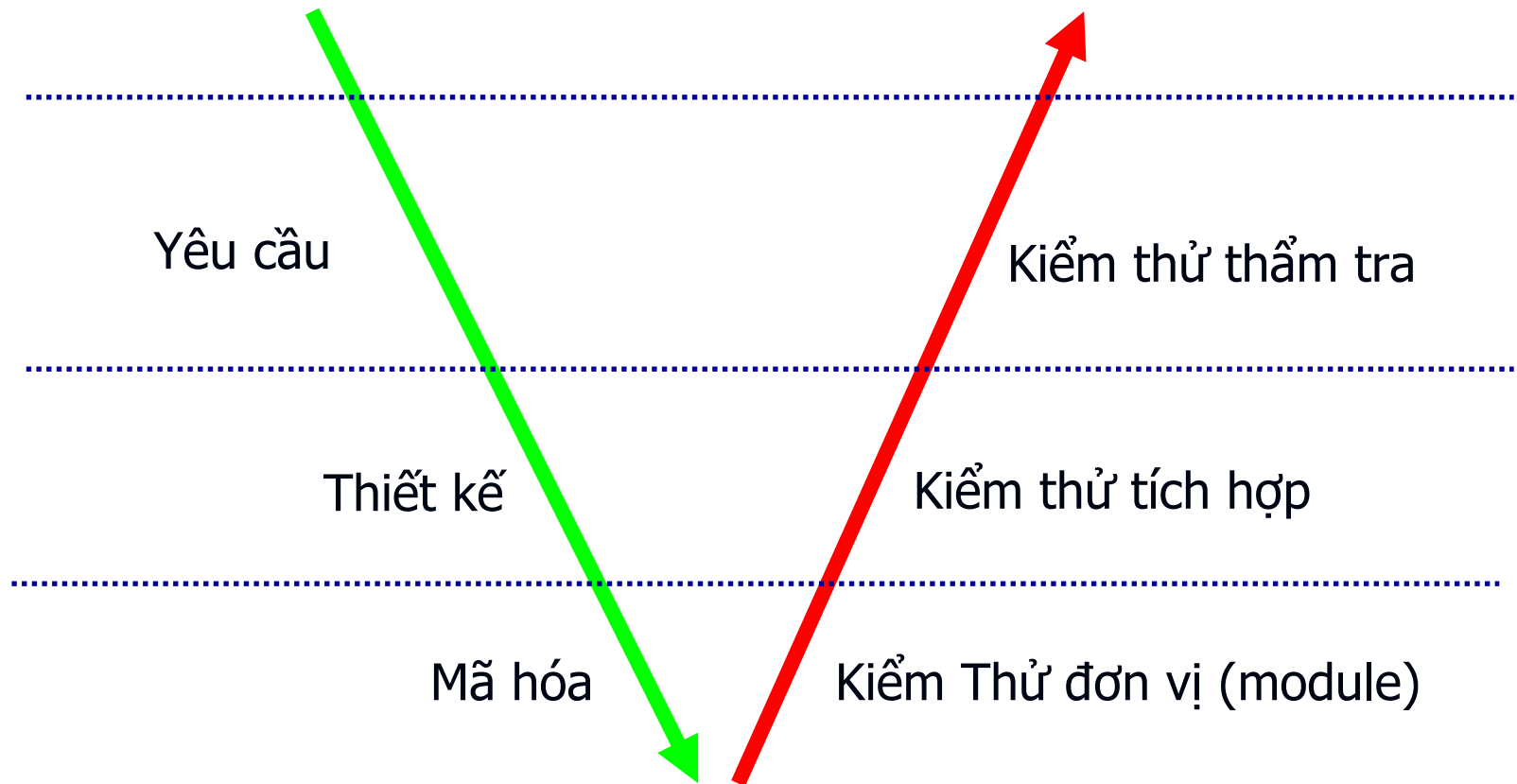
- Bắt đầu với ‘testing-in-the-small’ rồi tiến tới ‘testing-in-the-large’
- Với phần mềm truyền thống
  - Kiểm thử module (component)
  - Kiểm thử tích hợp module
- Với phần mềm hướng đối tượng
  - Khi bắt đầu “testing in the small” thì tập trung vào lớp (classes) mà chứa thuộc tính và phương thức, liên quan đến truyền thông và cộng tác



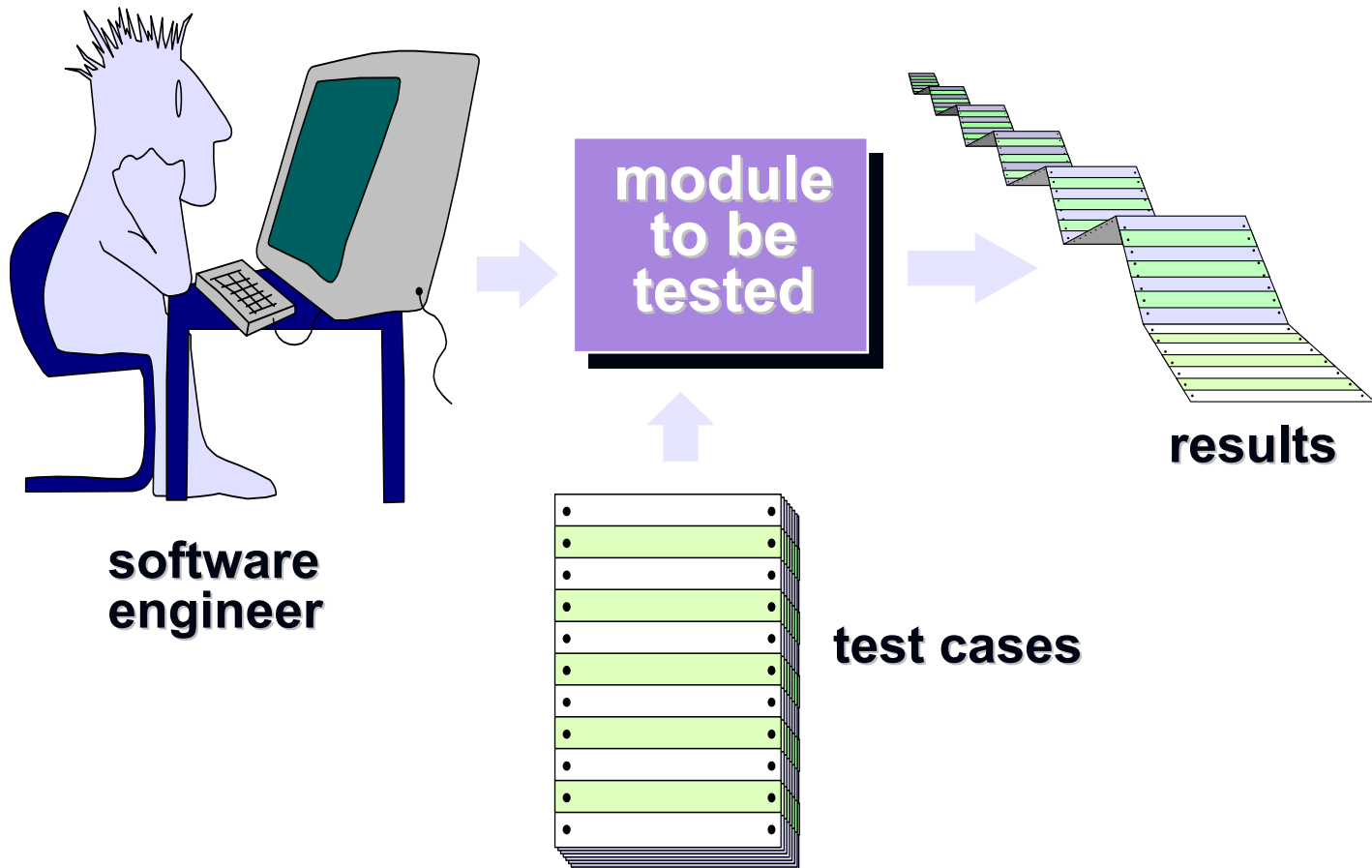
# Các công việc cần thiết trong Chiến lược

- Xác định rõ ràng các đối tượng kiểm thử
- Hiểu biết về người dùng phần mềm và tạo ra một tiền sử (profile) cho mỗi loại người dùng
- Xây dựng một kế hoạch kiểm thử mà nhấn mạnh tới “rapid cycle testing”
- Xây dựng phần mềm có tính kháng lỗi cao dùng cho kiểm thử
- Dùng kiểm tra kỹ thuật hình thức như là một bộ lọc trước khi kiểm thử
- Đề ra những kiểm tra kỹ thuật hình thức để đánh giá chiến lược kiểm thử và các test case
- Phát triển một hướng cải tiến liên tục cho qui trình kiểm thử

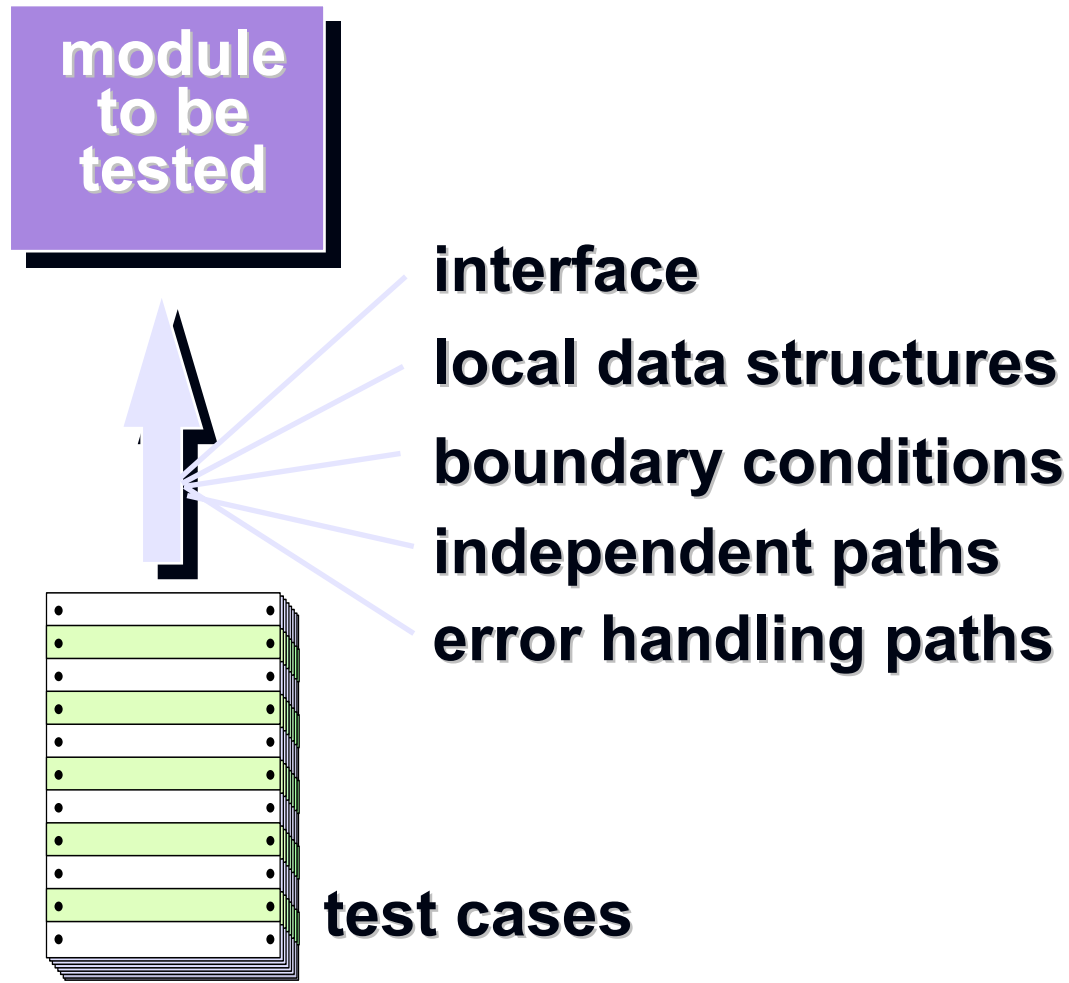
# Một chiến thuật kiểm nghiệm phổ biến (V)



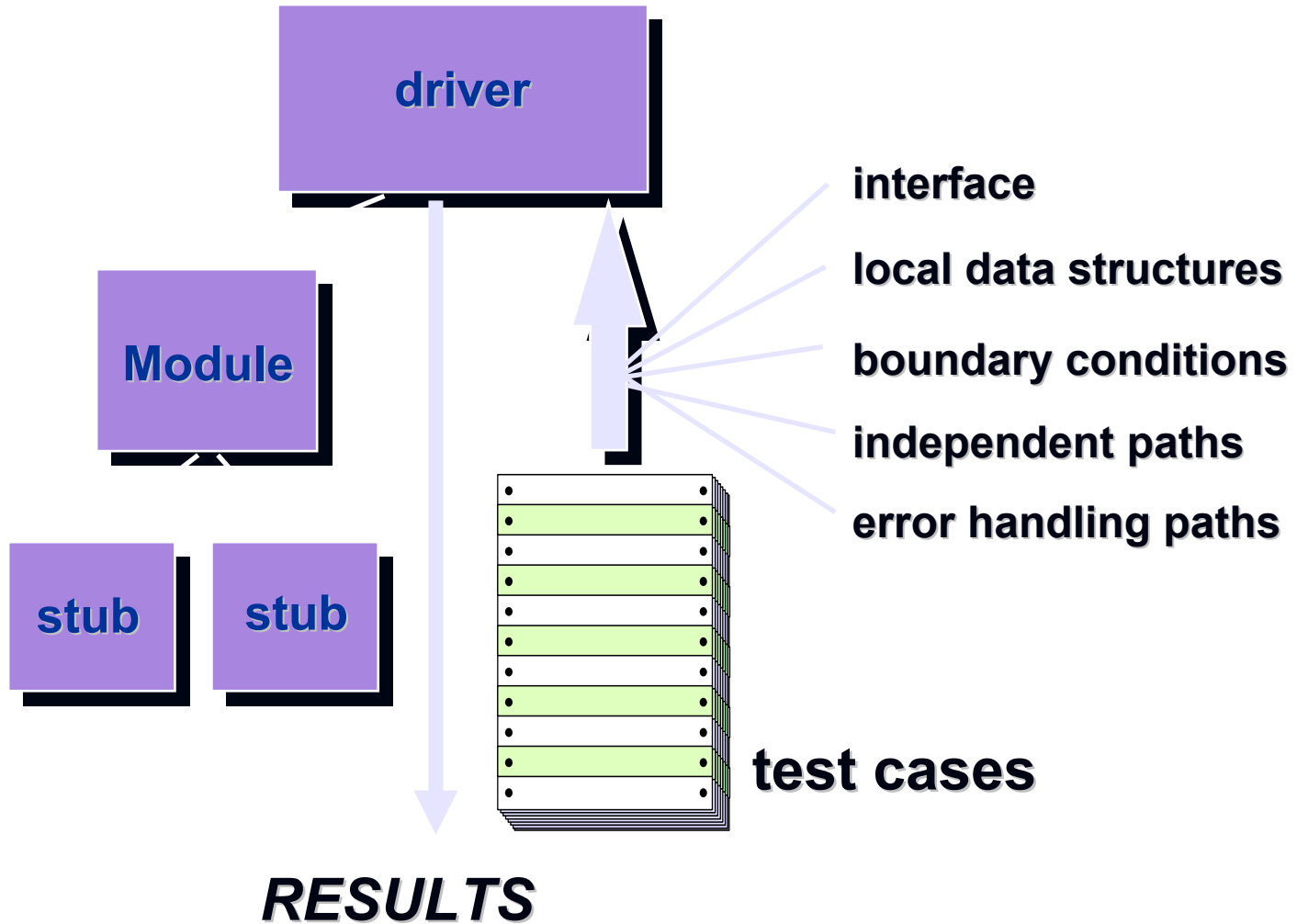
# Kiểm thử đơn vị



# Kiểm thử đơn vị



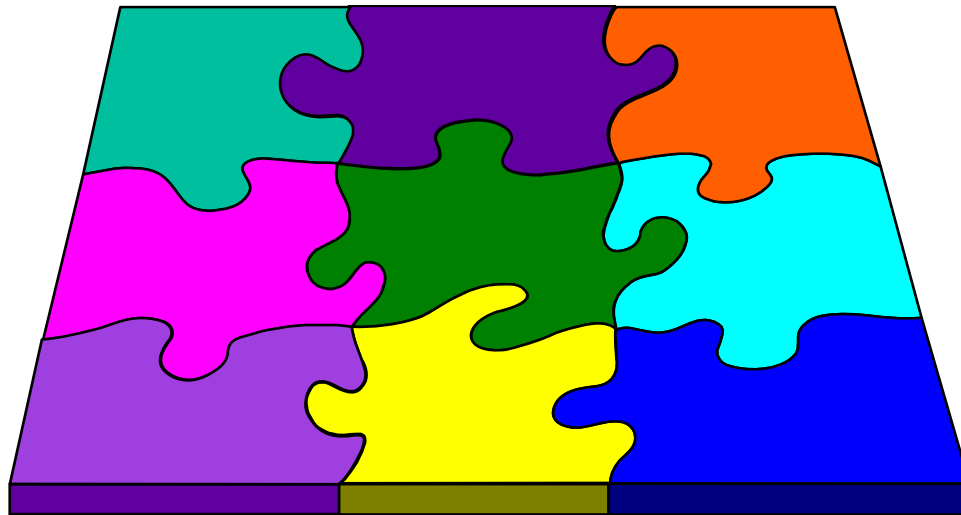
# Môi trường kiểm thử đơn vị



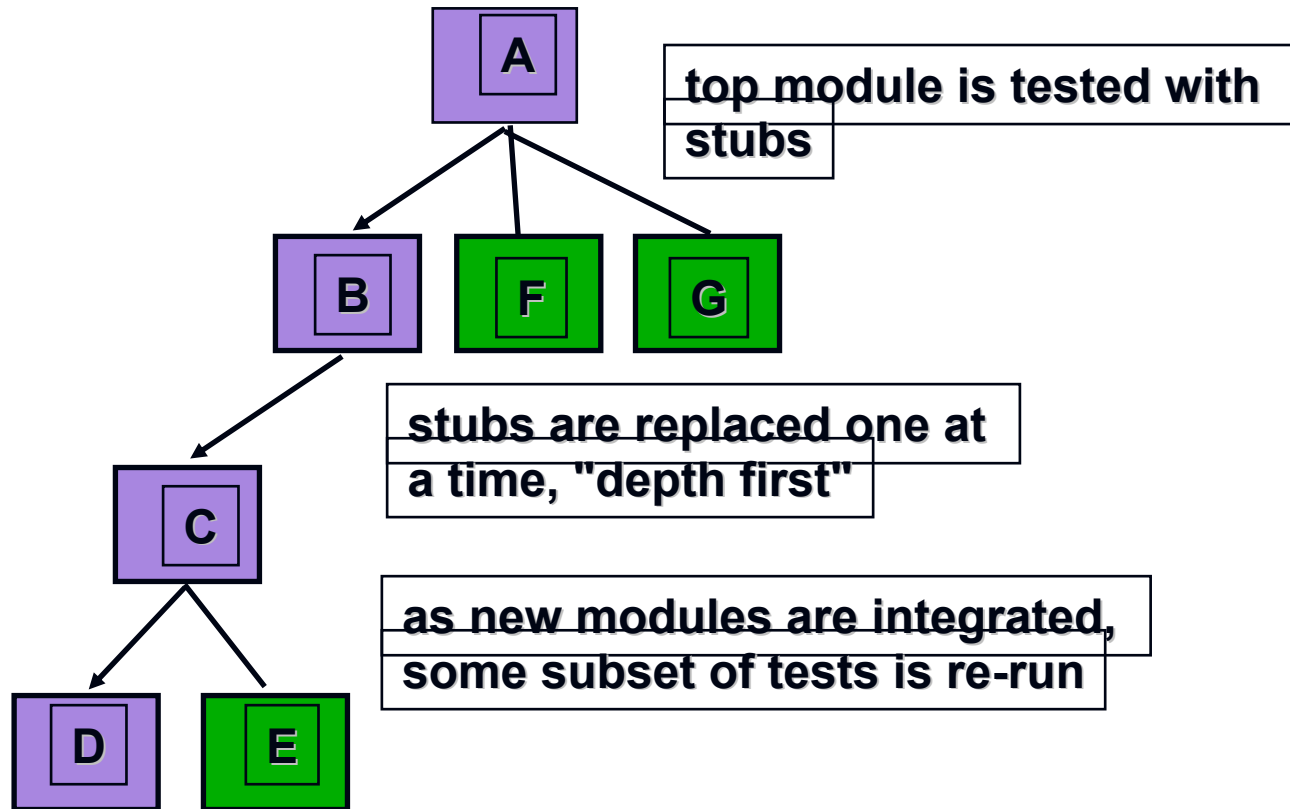
# Chiến lược kiểm thử tích hợp

**Chọn lựa:**

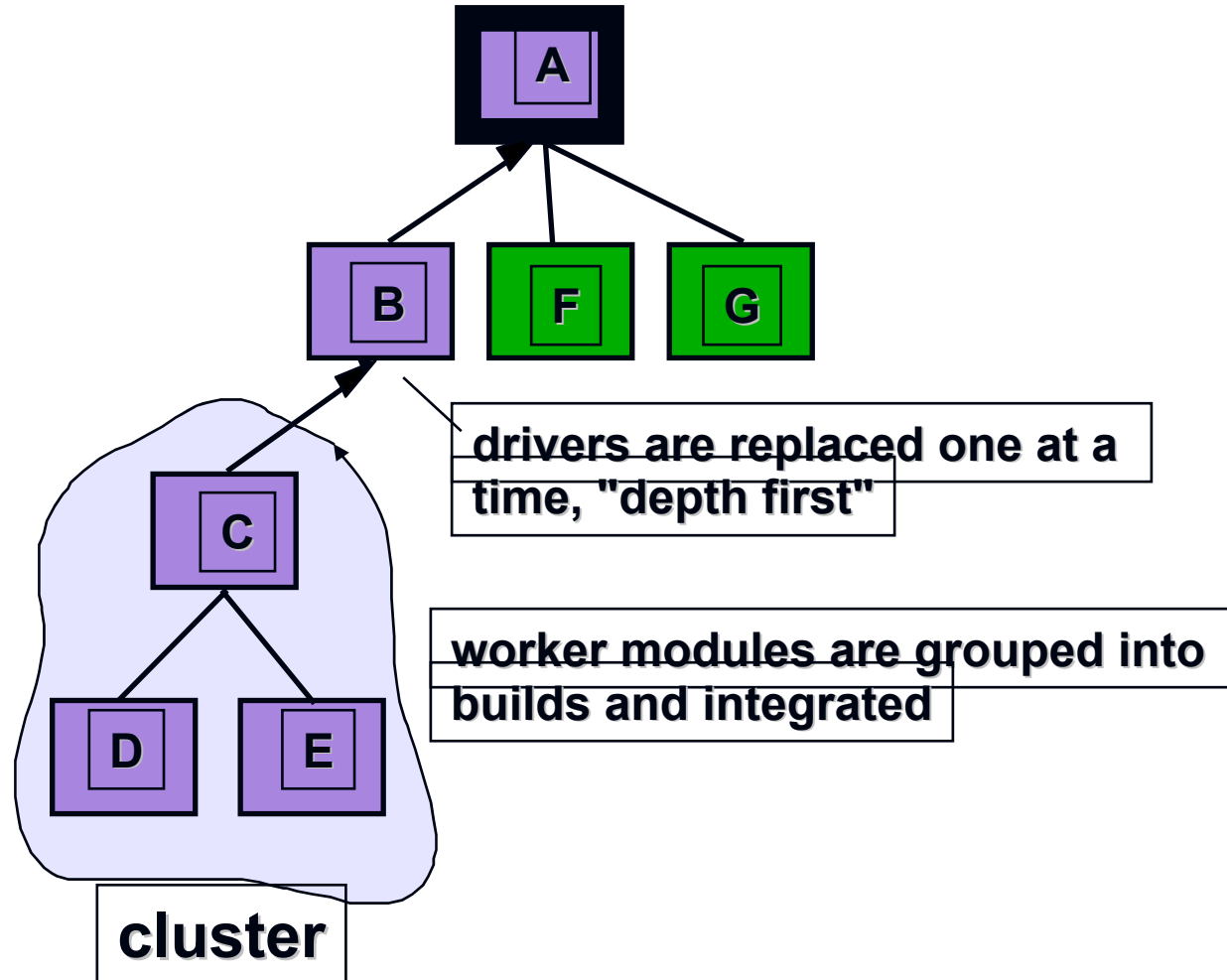
- **Hướng tiếp cận “big bang”**
- **Chiến lược xây dựng gia tăng**



# Tích hợp Top Down

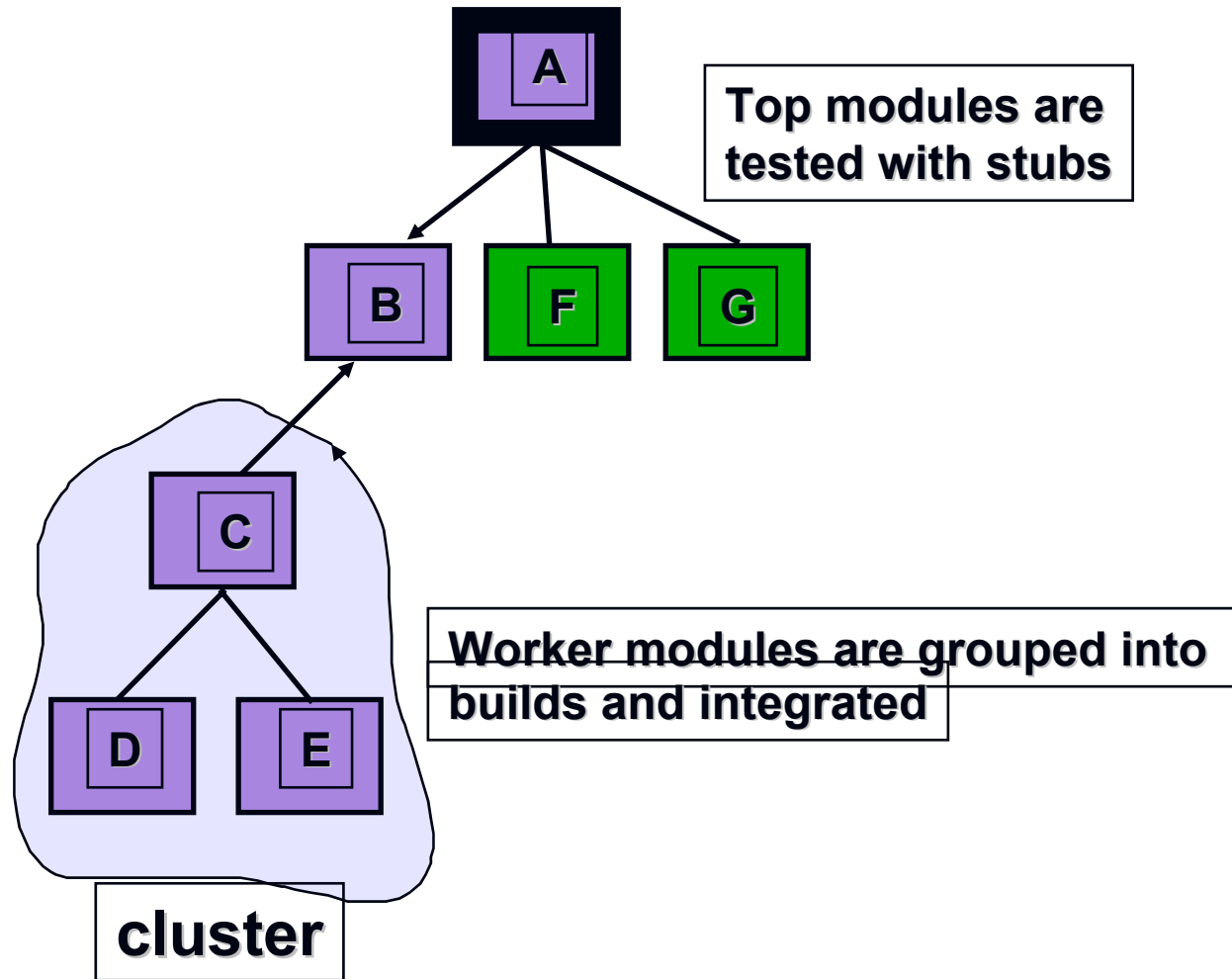


# Tích hợp Bottom-Up





# Kiểm thử Sandwich



# KIỂM THỬ HỒI QUY (regression)

1. Việc kết hợp các module lại với nhau có thể ảnh hưởng đến vòng lặp điều khiển, cấu trúc dữ liệu hay I/O chia sẻ trong một số module
2. Điều đó làm lộ ra một số lỗi không thể phát hiện được khi tiến hành kiểm nghiệm theo đơn vị
3. Kiểm nghiệm hồi quy có thể được tiến hành thủ công bằng cách thực hiện lại các *test-case* đã tạo ra. Hoặc có thể dùng một công cụ *capture-playback* để thực hiện tự động

# Kiểm thử hướng đối tượng

- Bắt đầu bằng cách đánh giá sự đúng đắn và toàn vẹn của mô hình OOA va OOD
- Thay đổi chiến lược kiểm thử so với trước đây
  - Khái niệm 'unit'
  - Việc tích hợp tập trung vào lớp và thực thi qua một tiến trình (thread) hay ngữ cảnh của một kịch bản được dùng
  - Việc thẩm định sử dụng phương pháp blackbox
- Thiết kế test case theo phương pháp cũ nhưng phải bao gồm thêm những đặc trưng mới
- Kiểm thử mô hình CRC

# Chiến lược trong OOT

- Kiểm thử lớp (unit testing)
  - Kiểm thử tác vụ (operations)
  - Kiểm tra hành vi, trạng thái của lớp
- Kiểm thử tích hợp
  - **thread-based testing** — integrates the set of classes required to respond to one input or event
  - **use-based testing** — integrates the set of classes required to respond to one use case
  - **cluster testing** (kiểm thử cụm) — integrates the set of classes required to demonstrate one collaboration

# Kiểm thử Smoke

- Một hướng thông dụng cho việc tạo “daily builds” cho sản phẩm phần mềm
- Các bước kiểm thử Smoke:
  - Những thành phần phần mềm được thể hiện dưới dạng mã được tích hợp thành một ‘build’ (kiểu kiến trúc)
    - Một build bao gồm tất cả các file dữ liệu, thư viện, những module sử dụng lại và những thành phần kỹ nghệ mà được yêu cầu thực thi một hay nhiều chức năng của sản phẩm
  - Một số test được thiết kế để khám phá những lỗi khi build thực hiện những chức năng của nó
    - Nhằm khám phá những lỗi ảnh hưởng tới lịch biểu
  - Những build được tích hợp với những built khác và sản phẩm toàn bộ (theo thời gian) là smoke được kiểm thử hằng ngày.
    - Hướng tích hợp có thể là top down hay bottom up

# Kiểm thử HO (High Order)

- Kiểm thử thẩm tra (Validation testing)
  - Alpha/Beta testing
- Kiểm thử hệ thống (System testing)
  - Recovery testing
  - Security testing
  - Stress testing
  - Performance Testing

# Kiểm thử thẩm tra

- Kiểm thử thẩm tra (Validation testing) hiểu theo cách đơn giản nhất là kiểm tra các chức năng của phần mềm đáp ứng được nhu cầu của khách hàng đã được xác định trong văn bản đặc tả yêu cầu của phần mềm
- Áp dụng kỹ thuật black-box

# Kiểm thử thẩm tra

## ■ Kiểm nghiệm alpha

- Được tiến hành ngay tại nơi sản xuất phần mềm.
- Nhà phát triển phần mềm sẽ quan sát người sử dụng dùng sản phẩm và ghi nhận lại những lỗi phát sinh để sửa chữa.

## ■ Kiểm nghiệm beta

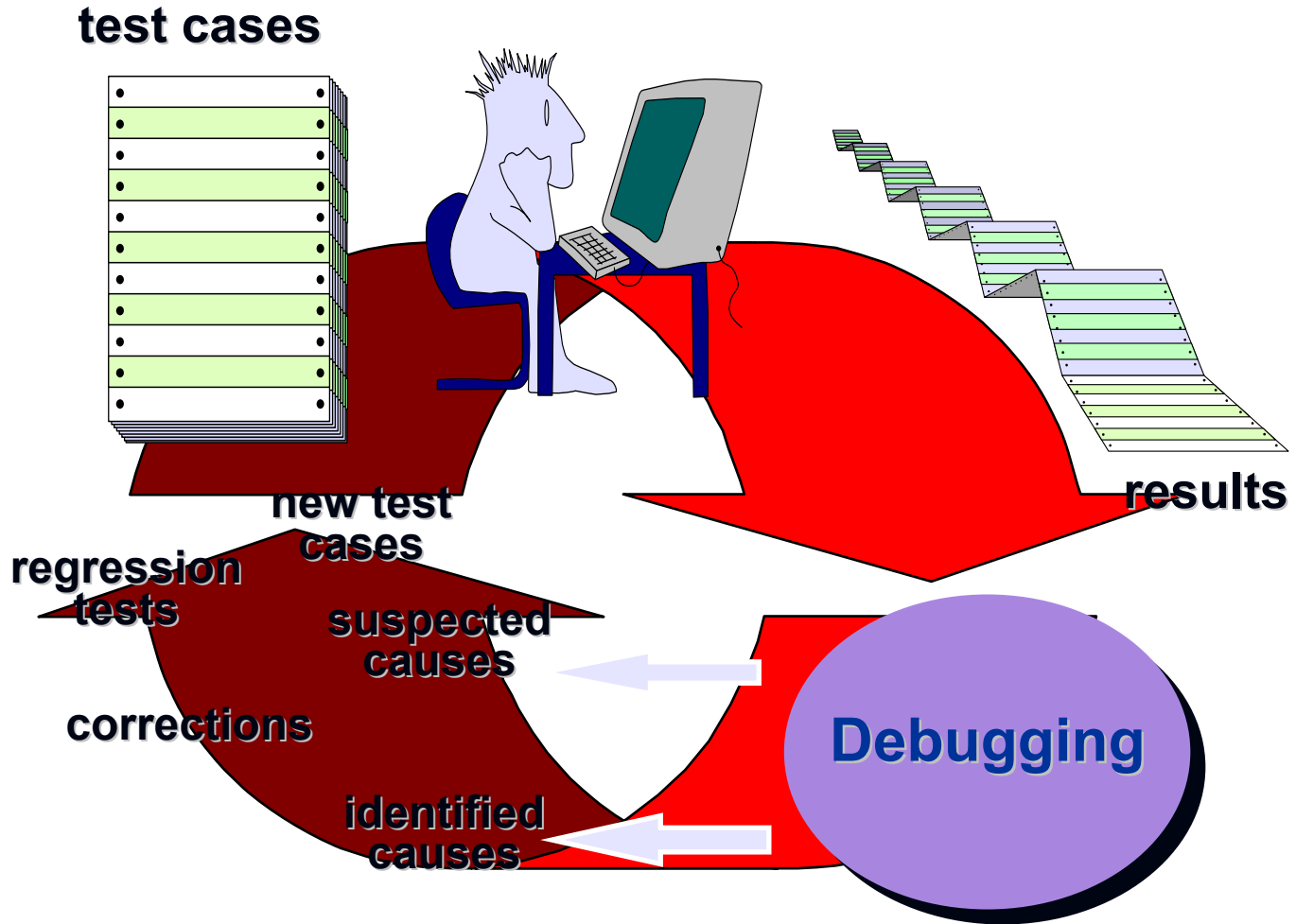
- Phần mềm được kiểm tra bên ngoài phạm vi của đơn vị sản xuất.
- Khách hàng trực tiếp sử dụng và ghi nhận lỗi để báo lại cho nhà phát triển sửa chữa.



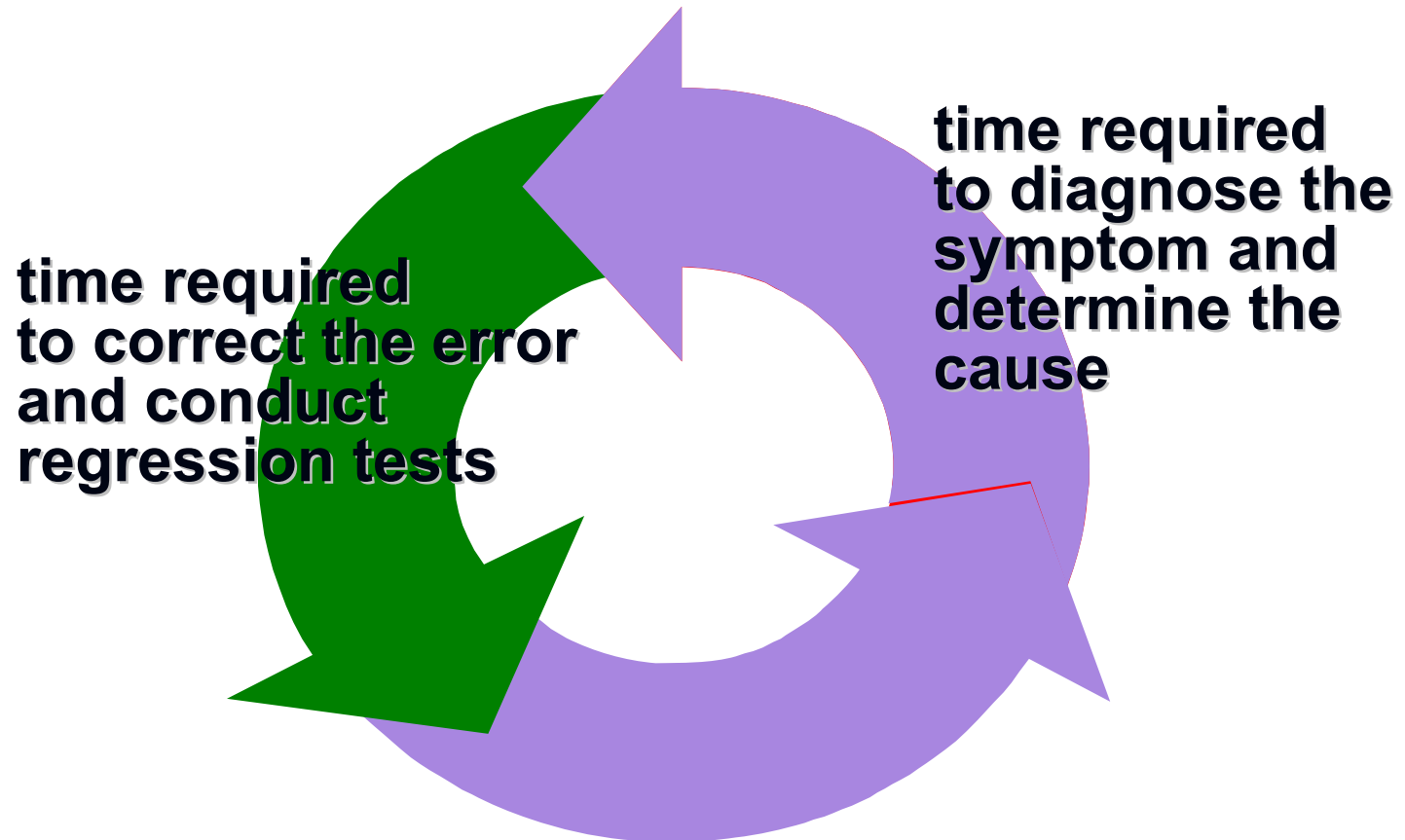
# Debugging (gỡ lỗi): Một quá trình phân tích



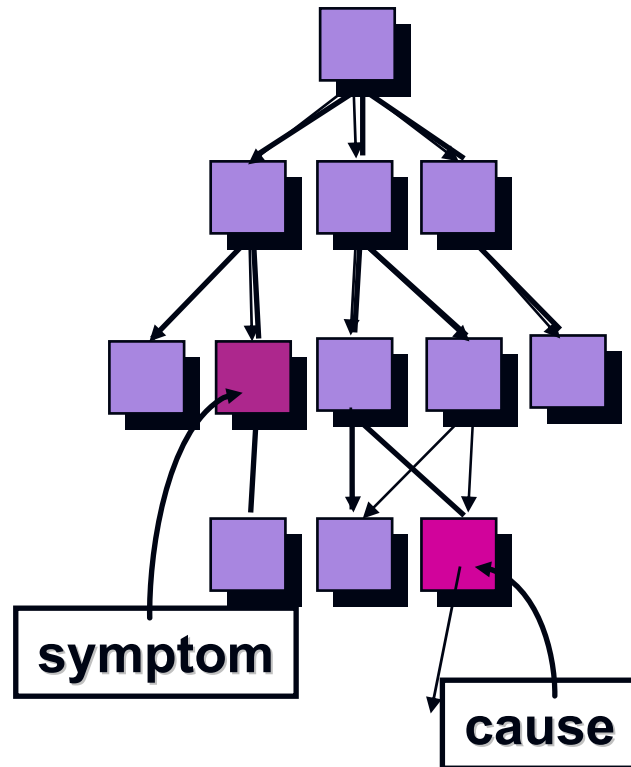
# Qui trình gỡ lỗi



# Nỗ lực gỡ lỗi

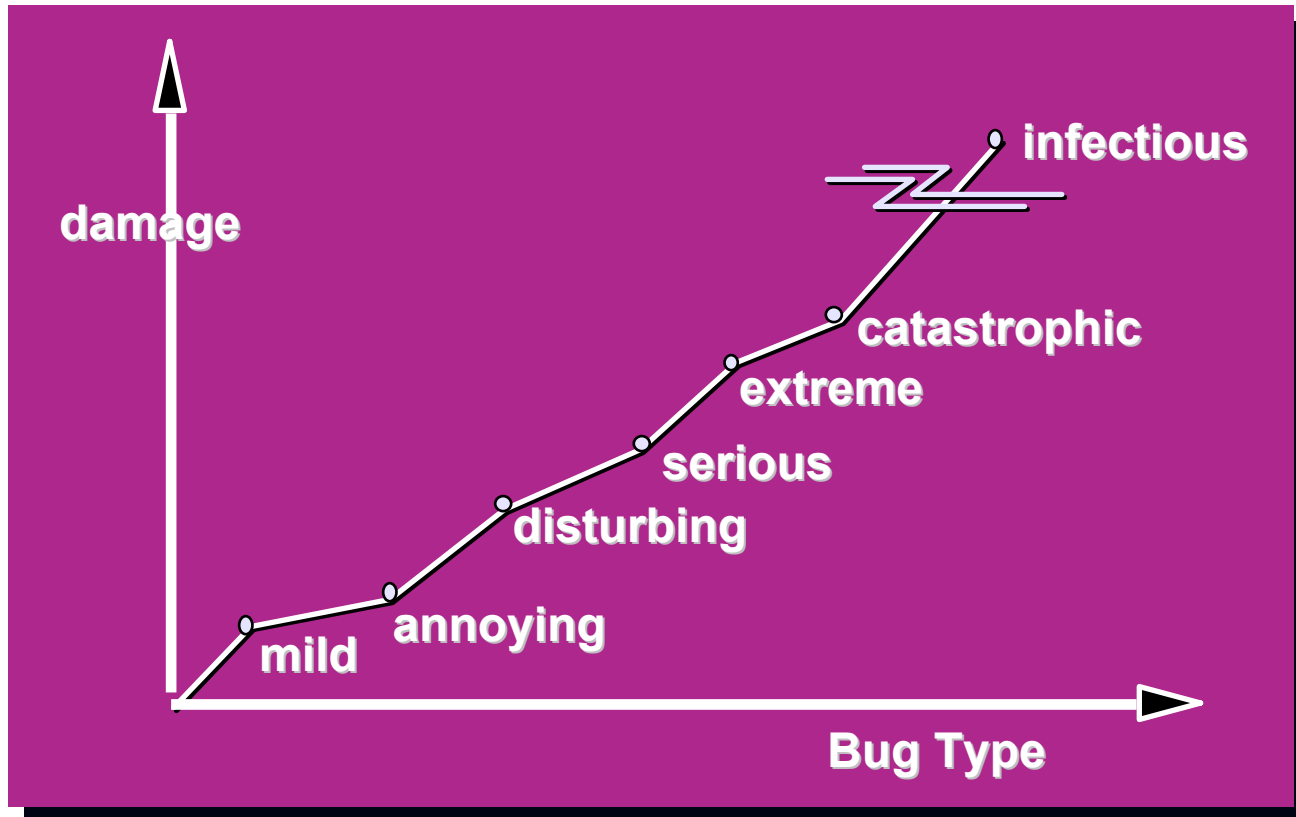


# Dấu hiệu và nguyên nhân



- ☐ Dấu hiệu và nguyên nhân có thể khác biệt về nơi
- ☐ Dấu hiệu có thể biến mất khi một vấn đề khác đã được sửa
- ☐ Nguyên nhân có thể do sự kết hợp của yếu tố không thực sự là lỗi
- ☐ Nguyên nhân có thể là do lỗi của hệ thống hay của bộ biên dịch
- ☐ Nguyên nhân có thể là những giả định mà mọi người tin tưởng
- ☐ Dấu hiệu có thể lúc có lúc không

# Hậu quả của lỗi



**Bug Categories:** function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.

# Kỹ thuật gỡ lỗi

- Brute force
- Backtracking
- Loại trừ nguyên nhân  
(cause elimination)  
(Induction (qui nạp),  
Deduction (suy diễn))

# BRUTE FORCE

- Là phương pháp phổ biến nhất nhưng lại ít hiệu quả nhất cho việc phát hiện nguyên nhân gây lỗi phần mềm.
- Triết lý của phương pháp này là: “Hãy để máy tính tìm ra lỗi”.
- Cách thực hiện:
  - ◆ Lấy dữ liệu trong bộ nhớ để xem xét.
  - ◆ Dùng *run-time trace* để tìm lỗi.
  - ◆ Dùng lệnh **WRITE** để xuất dữ liệu cần kiểm tra ra màn hình....
- Áp dụng phương pháp này khi tất cả các phương pháp khác đều thất bại.

# LÀN VẾT NGƯỢC (Backtracking)

- Là một phương pháp gỡ lỗi khá phổ biến có thể dùng thành công trong các chương trình nhỏ nhưng khó áp dụng cho đối với các chương trình rất lớn.
- Cách thực hiện: bắt đầu tại dòng mã nguồn có triệu chứng lỗi thực hiện lần ngược trở lại từng dòng mã nguồn cho đến khi tìm thấy dòng gây ra lỗi.



# LOẠI TRỪ NGUYÊN NHÂN

- Cách thực hiện:
  - Khi một lỗi được phát hiện, cố gắng đưa ra một danh sách các nguyên nhân có thể gây ra lỗi (các giả thiết)
  - Danh sách này được xem xét lại để loại bỏ dần các nguyên nhân không đúng cho đến khi tìm thấy một nguyên nhân khả nghi nhất (dùng dữ liệu liên quan)
  - Khi đó dữ liệu kiểm thử sẽ được tinh chế lại để tiếp tục tìm lỗi.

# Các lưu ý

1. Đừng vội vã hãy suy xét đến những dấu hiệu mà bạn thấy
2. Dùng tool (dynamic debugger...) để có thể nhìn sâu hơn vào bên trong
3. Khi bế tắc nên nhờ người khác trợ giúp
4. Khi gỡ lỗi cần phải thực hiện kiểm thử hồi qui (regression tests)

## 2. Kỹ thuật kiểm thử phần mềm

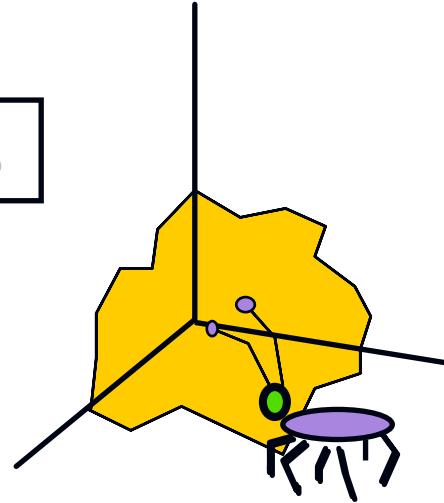
### Một test “tốt”?

- Có khả năng tìm lỗi
- Không dư thừa
- “best of breed”
- Không quá đơn giản và quá phức tạp

# Thiết kế Test Case

"Bugs lurk in corners  
and congregate at  
boundaries ..."

*Boris Beizer*



**OBJECTIVE**

to uncover errors

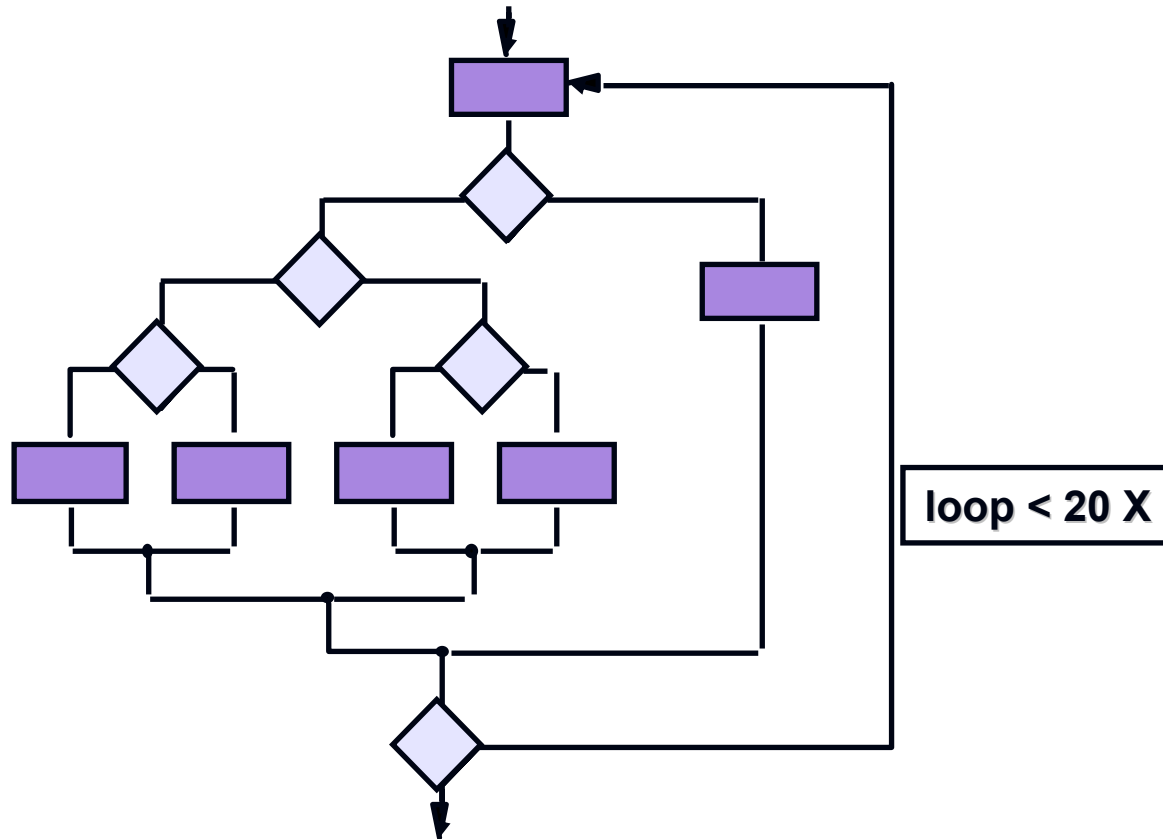
**CRITERIA**

in a complete (toàn diện) manner

**CONSTRAINT**

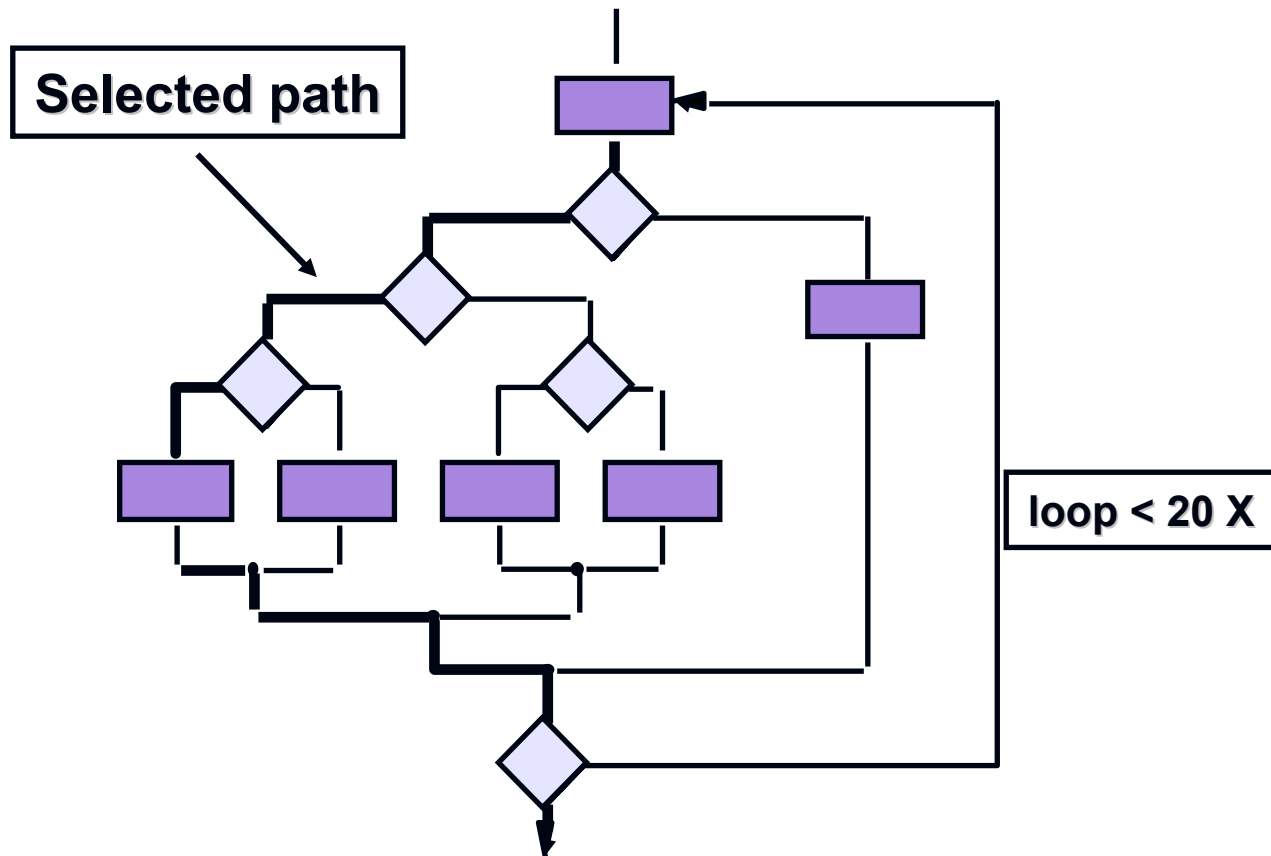
with a minimum of effort and time

# Kiểm thử vét cạn (Exhaustive)

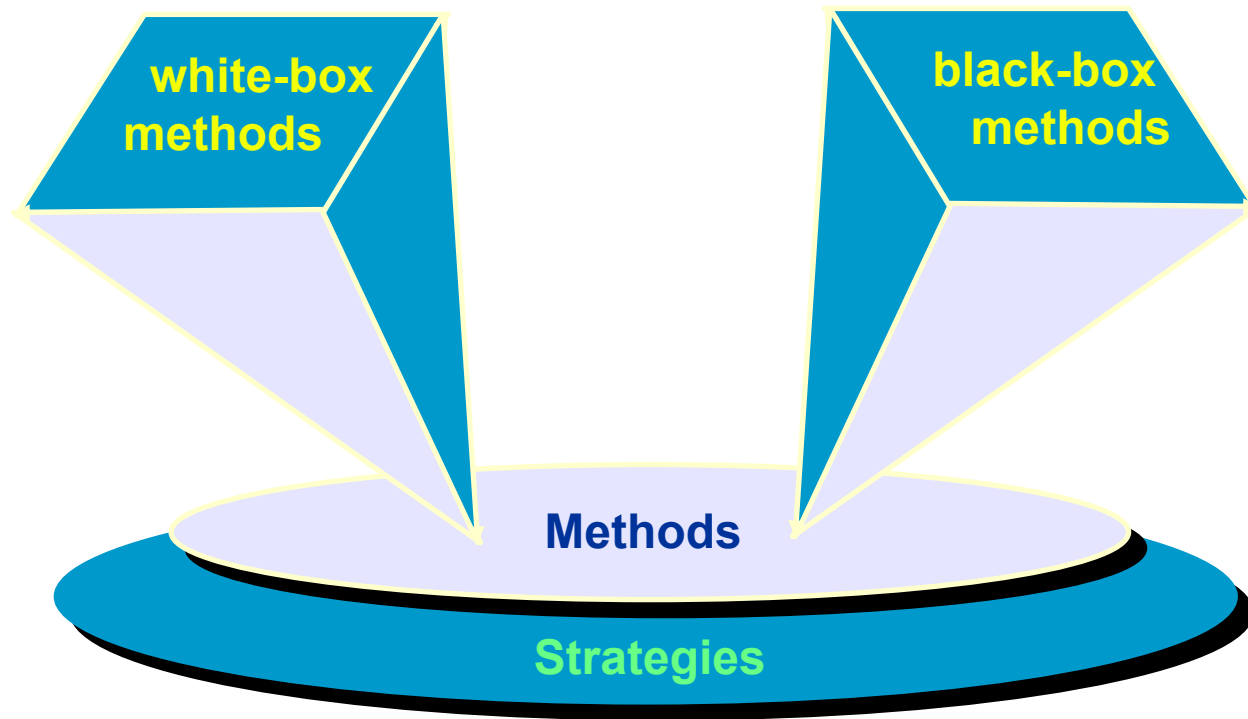


There are <sup>14</sup>~~10~~ possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

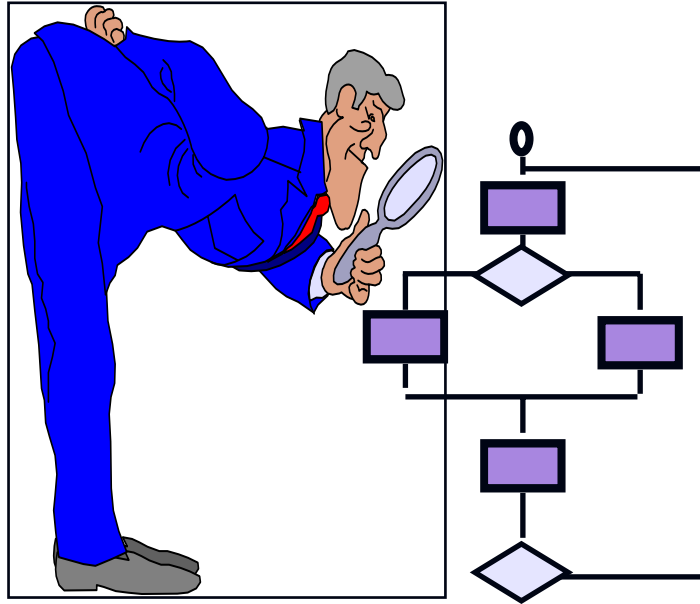
# Kiểm thử chọn lựa (Selective)



# Phương pháp kiểm thử



# Kiểm thử White-Box



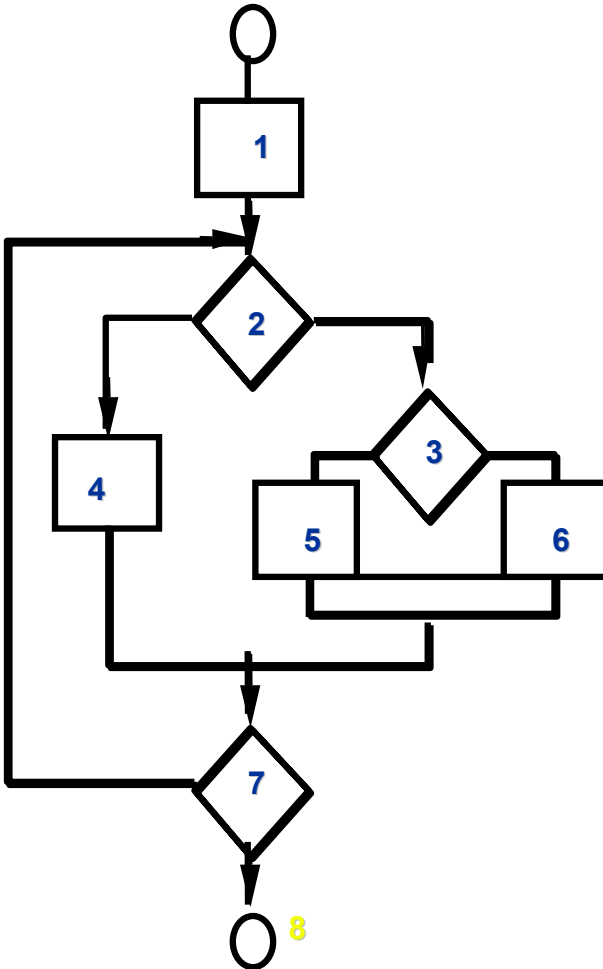
**... our goal is to ensure that all statements and conditions have been executed at least once ...**



# Khó phát hiện lỗi ?

- ☐ Chúng ta thường tin rằng một path có vẻ như không được thực hiện, nhưng thực tế thường ngược với trực giác
- ☐ Lỗi về chữ (typographical) là ngẫu nhiên, những path mà không kiểm thử thường chứa vài lỗi này
- ☐ Lỗi logic và những giả định không đúng thì tỷ lệ nghịch với khả năng thực thi của đường

# Đường cơ bản



**Path 1: 1,2,3,6,7,8**

**Path 2: 1,2,3,5,7,8**

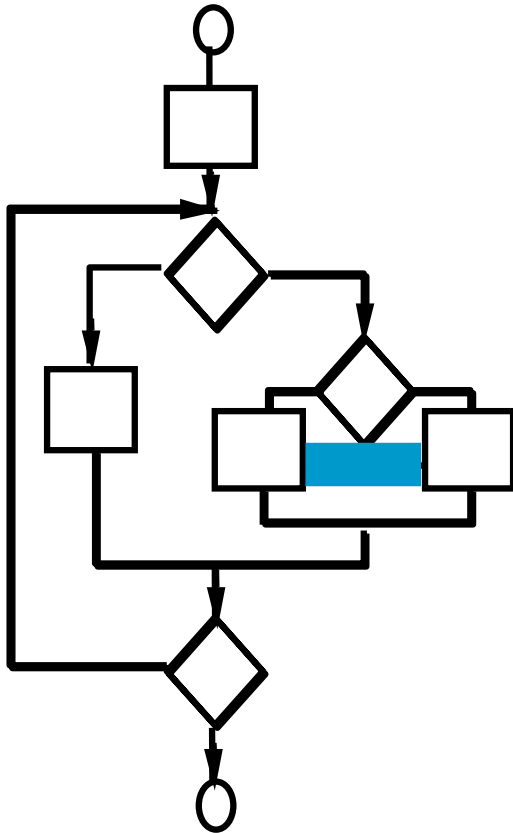
**Path 3: 1,2,4,7,8**

**Path 4: 1,2,4,7,2,4,...7,8**

# Kiểm nghiệm các đường cơ bản

- Kiểm nghiệm white-box dựa vào cấu trúc điều khiển của thiết kế thủ tục để sinh các test-case với tiêu chí
  - Kiểm nghiệm các đường cơ bản là một trong những phương cách kiểm nghiệm white-box
  - Bảo đảm số phép thử là ít nhất đủ để phát hiện các lỗi
  - Tất cả các đường cơ bản được thử qua ít nhất một lần
  - Thử các điều kiện rẽ nhánh ở cả 2 nhánh true và false
  - Thử qua vòng lặp tại biên cũng như bên trong
  - Thử qua cấu trúc dữ liệu để đảm bảo tính toàn vẹn của nó

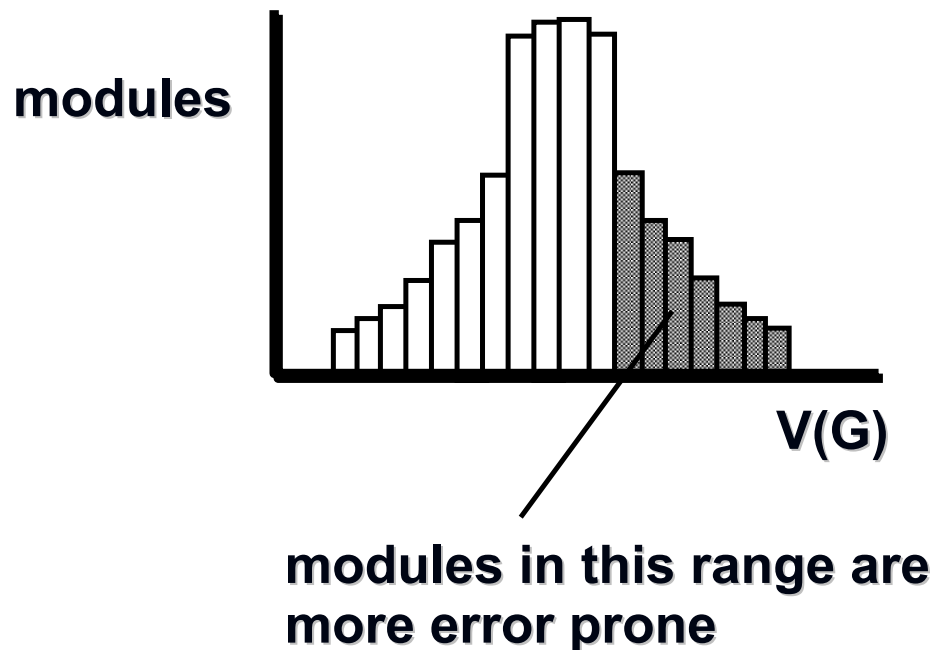
# Độ phức tạp lộ trình Cyclomatic Complexity $V(G)$



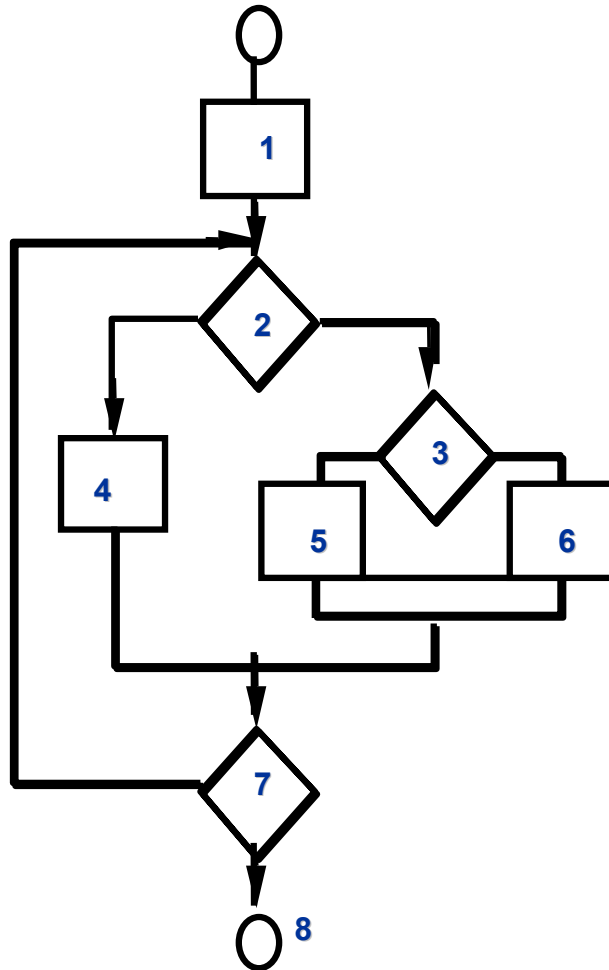
$$V(G) = 4$$

# Độ phức tạp lộ trình và lỗi

A number of industry studies have indicated that the higher  $V(G)$ , the higher the probability or errors.



# Đưa ra đường cơ bản



Next, we derive the independent paths:

Since  $V(G) = 4$ , there are four paths

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

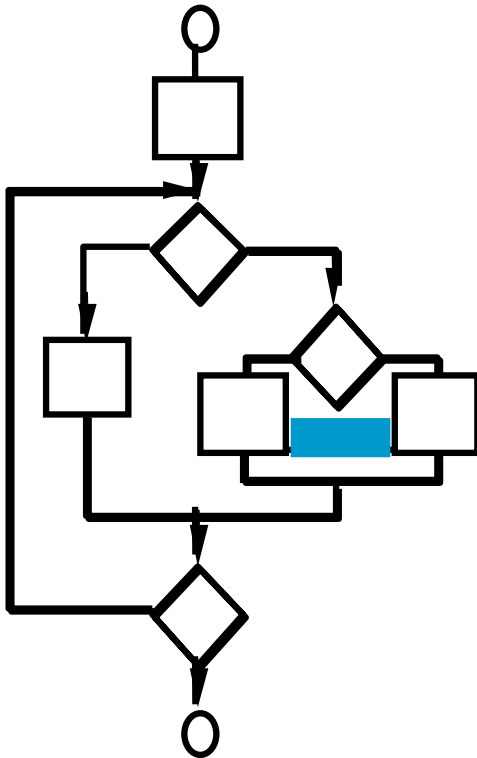
Path 2: 1,2,3,5,7,8

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

Path 4: 1,2,4,7,2,4,...7,8

Finally, we derive test cases to exercise these paths.

# Lưu ý trong kiểm thử đường cơ bản

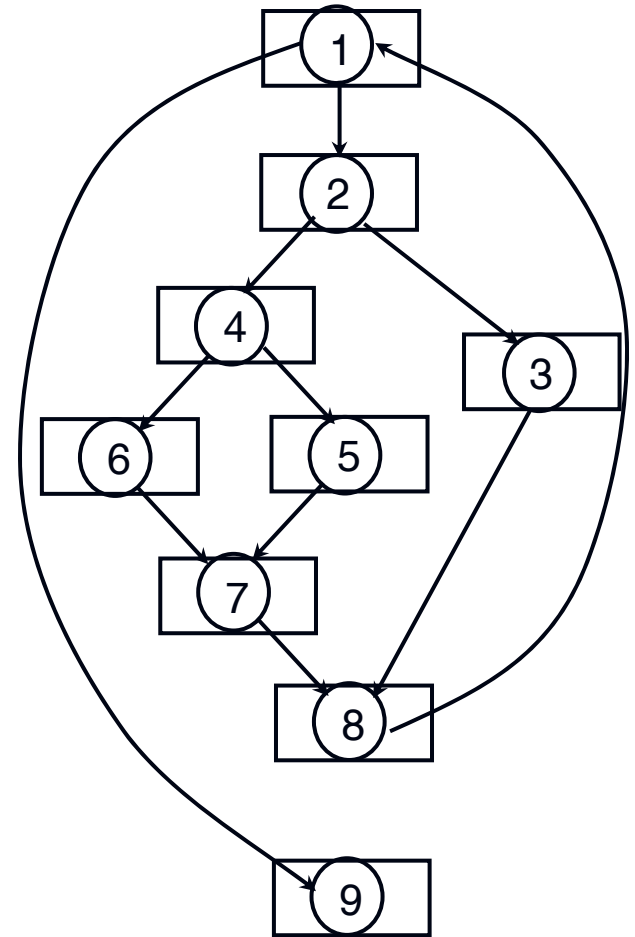


- ☐ Bạn không cần một flow chart, nhưng hình ảnh thì dễ vạch ra những path
- ☐ Tính mỗi test logic đơn giản, test phức hợp được tính là 2 hay nhiều hơn
- ☐ Kiểm thử đường cơ bản phải áp dụng cho những module có tính nghiêm ngặt (critical)

# Các đường độc lập cơ bản

- Đối với chương trình con DoSomething
- Tổng số đường :  
 $V = 3 + 1 = 4$
- Đường 1: 1-9
- Đường 2: 1-2-3-8-1...
- Đường 3: 1-2-4-5-7-8-1...
- Đường 4: 1-2-4-6-7-8-1...

Chú ý: dấu 3 chấm (...) mang ý nghĩa “không quan tâm”, từ đó có thể đi theo bất kỳ cạnh nào bởi vì các cạnh sau đó đã được duyệt qua rồi





# Ví dụ với AnalyzeTriangle

- Đối với chương trình con **AnalyzeTriangle**
- Tổng số đường:  
 $V = 6 + 1 = 7$
- Đường 1: 1-3-12
- Đường 2: 1-2-3-12
- Đường 3: 1-2-4-5-12
- Đường 4: 1-2-4-6-7-12
- Đường 5: 1-2-4-6-8-7-12
- Đường 6: 1-2-4-6-8-9-10-12
- Đường 7: 1-2-4-6-8-9-11-12

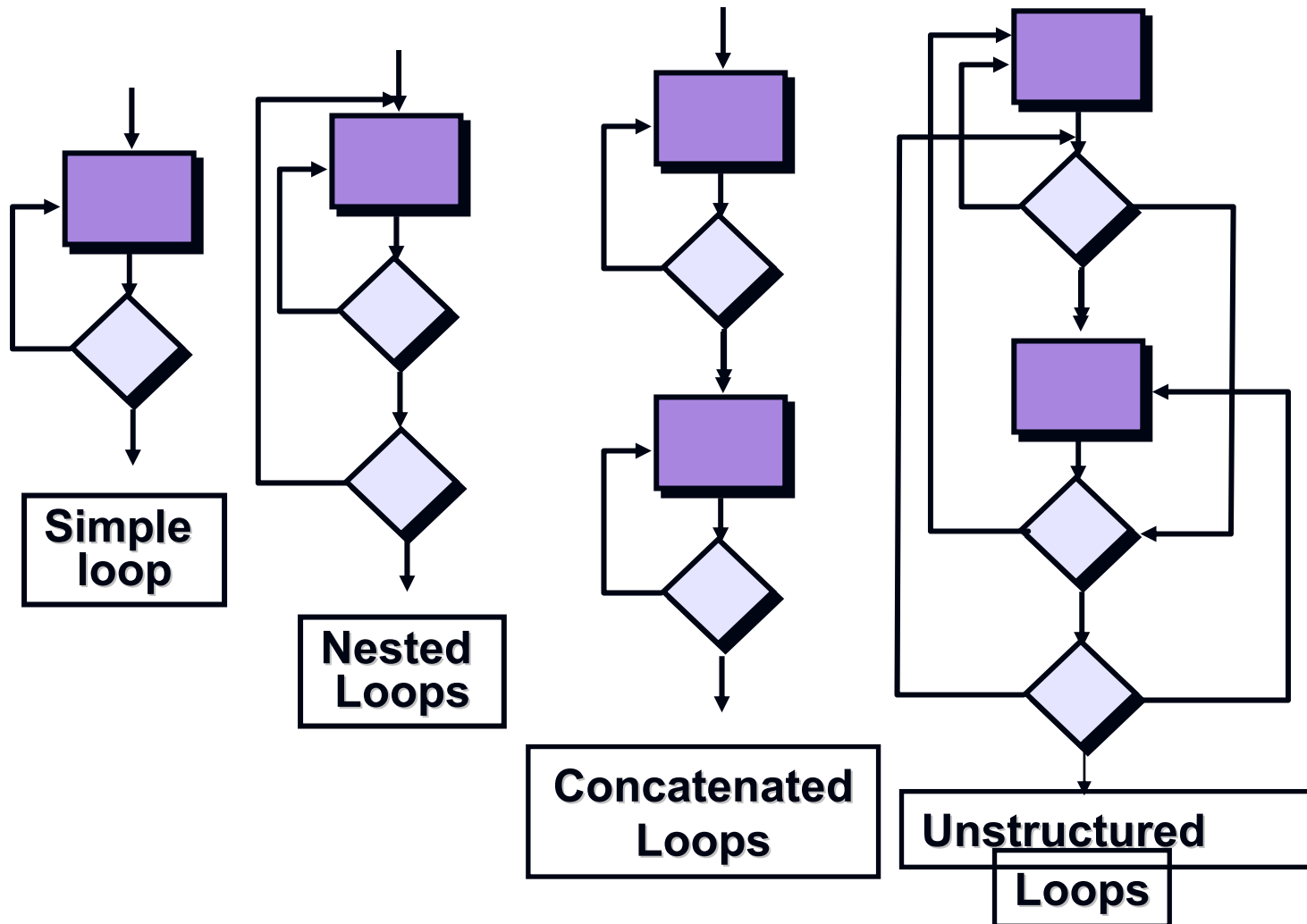
# Test case

Conditions	1	2	3	4	5	6	7	8
$a = b$ or $a = c$ or $b = c$	T	T	T	T	T	F	F	F
$a = b$ and $b = c$	T	T	F	F	F	F	F	F
$a < b + c$ or $b < a + c$ or $c < a + b$	T	F	T	T	F	T	T	F
$a > 0$ or $b > 0$ or $c > 0$	T	F	T	F	F	T	F	F
Sample test case	0,0,0	3,3,3	0,4,0	3,8,3	5,8,5	0,5,6	3,4,8	3,4,5
Expected output	Bad inputs	Equilateral	Bad inputs	Not triangle	Isosceles	Bad inputs	Not triangle	Scalene

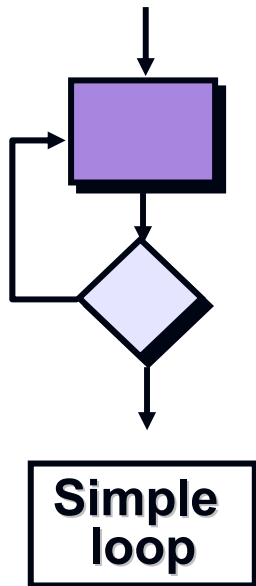
# Kiểm thử cấu trúc điều khiển

- **Kiểm thử điều kiện (Condition testing):** a test case design method that exercises the logical conditions contained in a program module
- **Kiểm thử luồng dữ liệu (data flow testing):** dựa vào vị trí định nghĩa và dùng của biến trong chương trình
- **Kiểm thử vòng lặp (Loop)**

# Kiểm thử vòng lặp (Loop)



# Vòng lặp đơn

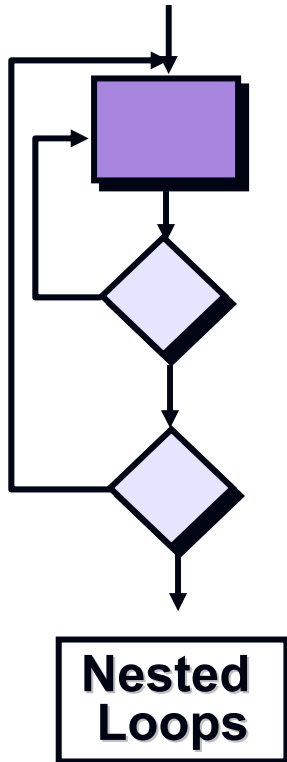


## Minimum conditions—Simple Loops

1. skip the loop entirely
2. only one pass through the loop
3. two passes through the loop
4.  $m$  passes through the loop  $m < n$
5.  $(n-1)$ ,  $n$ , and  $(n+1)$  passes through the loop

where  $n$  is the maximum number of allowable passes

# Vòng lặp lồng nhau



**Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.**

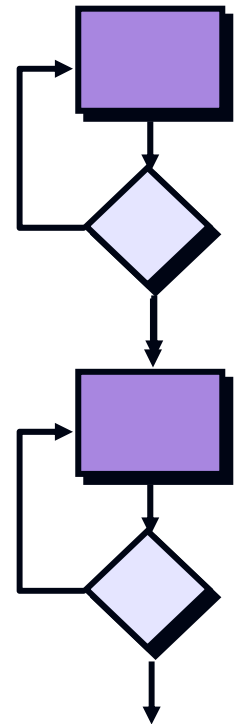
**Test the min, typical, max for the innermost loop, while holding the outer loops at their minimum values.**

**Move out one loop and set it up as in step 2, holding all inner loops at typical values. Continue this step until the outermost loop has been tested.**

# Vòng lặp nối tiếp

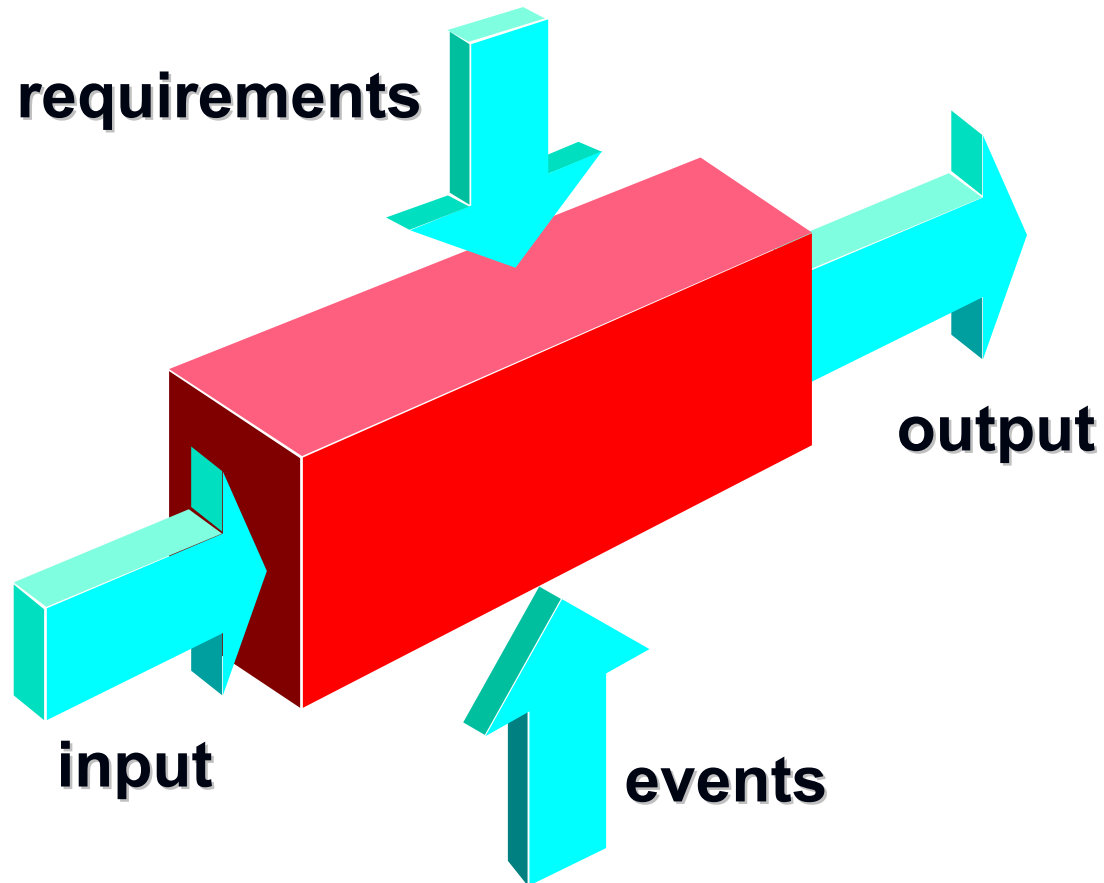
If the loops are independent of one another  
then treat each as a simple loop  
else\* treat as nested loops  
endif\*

*for example, the final loop counter value of loop 1 is  
used to initialize loop 2.*



Concatenated  
Loops

# Kiểm thử Black-Box

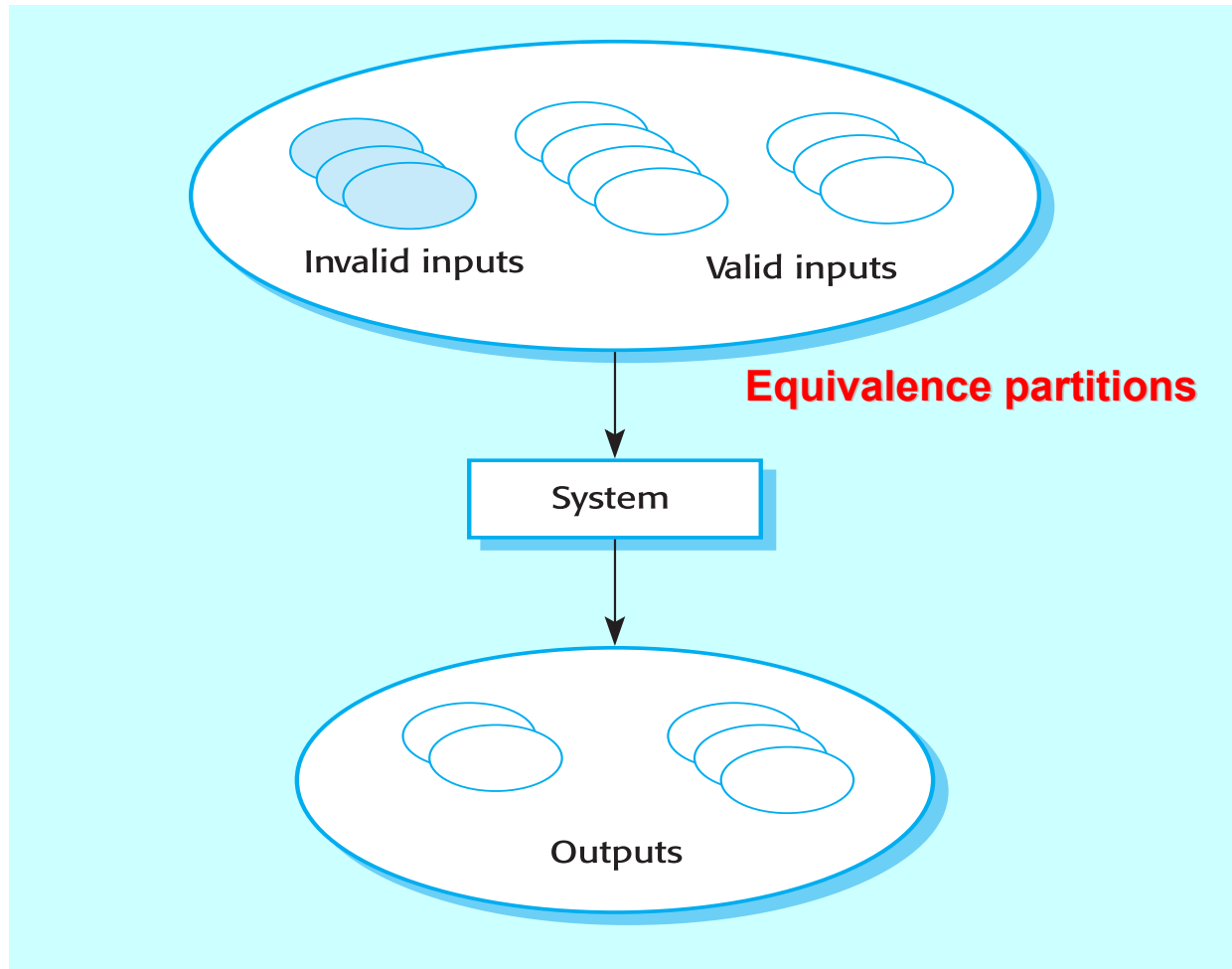




# Kiểm thử Black-Box

- Giá trị của những chức năng được kiểm thử bằng cách nào?
- Việc thực thi và hành vi của hệ thống được kiểm như thế nào?
- Những lớp input nào sẽ tạo ra những test case tốt?
- Hệ thống thường nhạy cảm với những giá trị input xác định nào?
- Biên của những lớp dữ liệu được cô lập như thế nào?
- Tỷ lệ và độ lớn của dữ liệu mà hệ thống có thể chịu đựng?
- Sự kết hợp dữ liệu đặc trưng sẽ có hiệu ứng gì trong hoạt động của hệ thống?

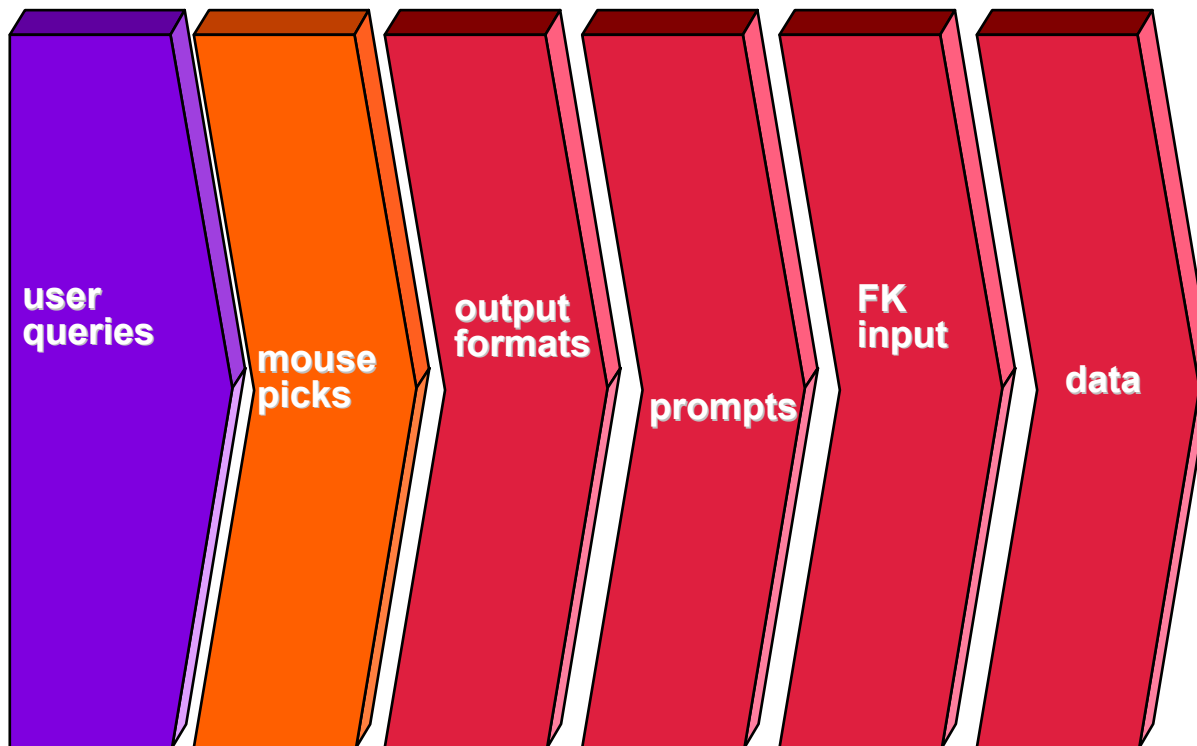
# Phân hoạch tương đương (Equivalence partitions)



# Hướng dẫn phân chia lớp

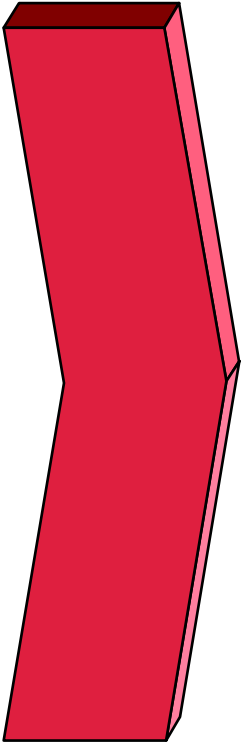
- Nếu input là một dãy, chia thành 1 lớp valid và 2 lớp invalid
- Nếu input là một giá trị đặc biệt, chia thành 1 lớp valid và 2 lớp invalid
- Nếu input là một thành viên của tập hợp, chia thành 1 lớp valid và 1 lớp invalid
- Nếu input là một giá trị boolean, chia thành 1 lớp valid và 1 lớp invalid

# Phân hoạch tương đương



# Mẫu lớp tương đương

## Valid data



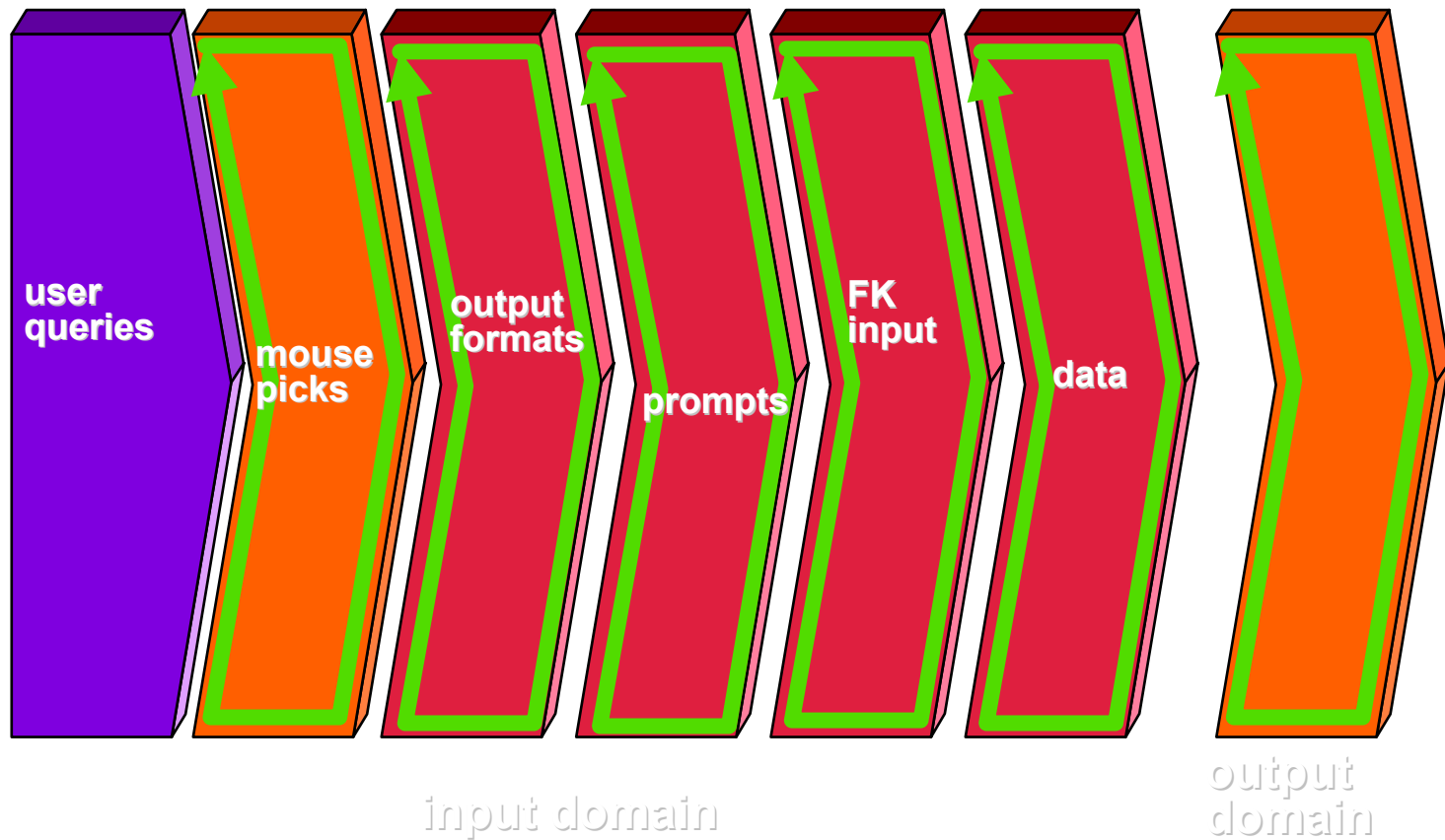
- user supplied commands
- responses to system prompts
- file names
- computational data
  - physical parameters
  - bounding values
  - initiation values
- output data formatting
- responses to error messages
- graphical data (e.g., mouse picks)

## Invalid data

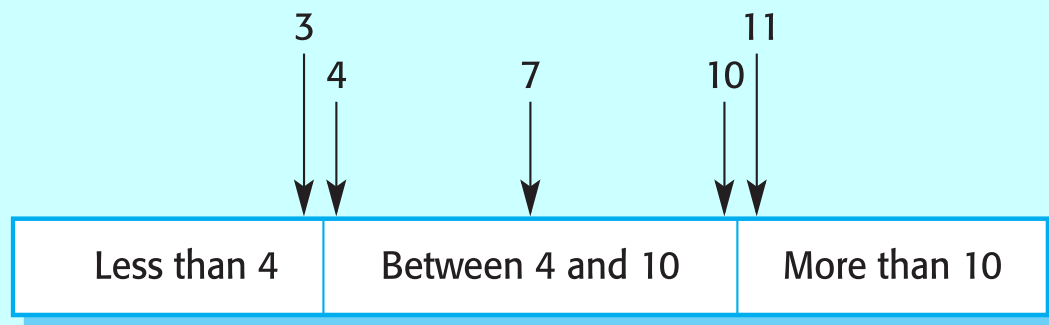
- data outside bounds of the program
- physically impossible data
- proper value supplied in wrong place

# Phân tích giá trị biên

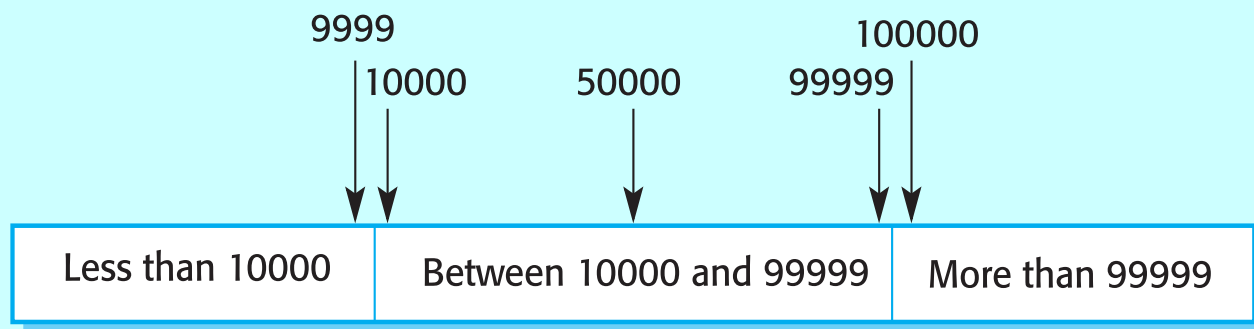
## BVA (Boundary Value Analysis)



# Phân hoạch tương đương

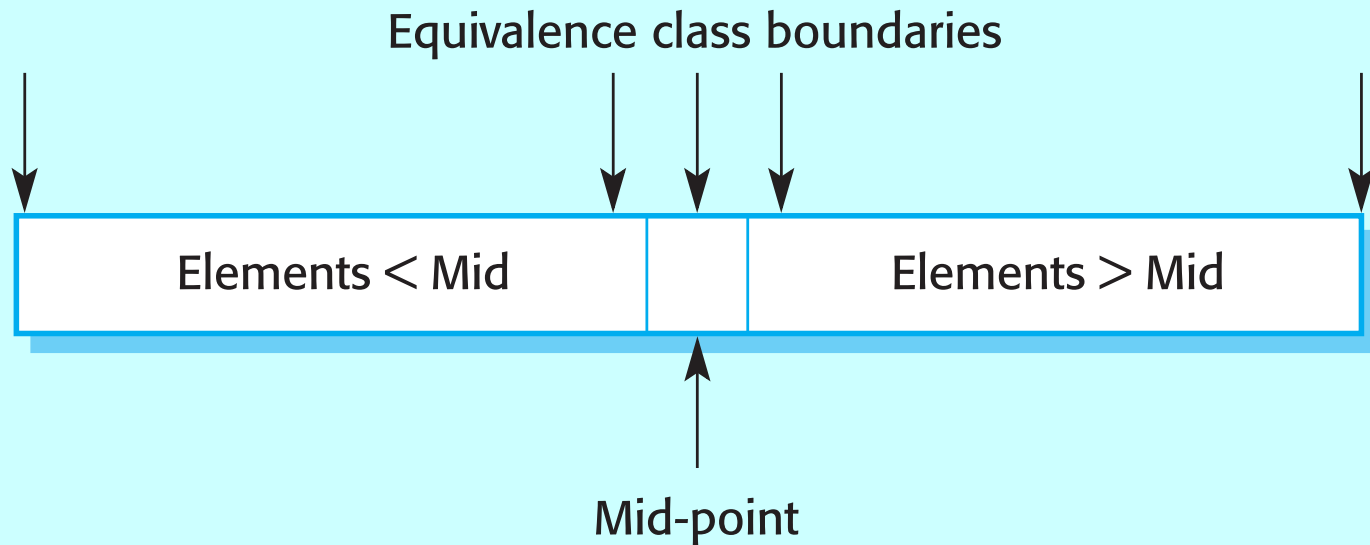


Number of input values



Input values

# Lớp tương đương cho tìm kiếm nhị phân





# Một testcase cho tìm kiếm nhị phân

Input array (T)	Key (Key)	Output (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

# Kiểm thử so sánh

- Được dùng với những phần mềm cần có tính tin cậy rất cao
  - Phân chia những nhóm kỹ sư phần mềm phát triển những version độc lập của một ứng dụng dùng cùng một đặc tả
  - Mỗi version được kiểm thử với cùng dữ liệu kiểm thử và bảo đảm rằng tất cả có output giống nhau
  - Tất cả các version thực thi song song với thời gian thực và so sánh kết quả

# Kiểm thử hướng đối tượng OOT

**Berard [BER93] đề nghị cho thiết kế test case:**

1. Mỗi test case phải có định danh duy nhất và phải kết hợp rõ ràng với class được test
2. Chỉ rõ mục đích của test
3. Các bước cho mỗi test [BER94]:
  - a. Một danh sách những trạng thái cho đối tượng được kiểm thử
  - b. Một danh sách những messages và tác vụ sẽ được thực hiện
  - c. Danh sách những loại trừ (exceptions) có thể xuất hiện
  - d. Danh sách những đối tượng ngoài (i.e., changes in the environment external )
  - e. Các thông tin hỗ trợ

# Phương pháp kiểm thử OOT ...

- **Kiểm thử hướng lỗi (Fault-based testing)**
  - **Thiết kế test case dựa vào dự đoán những lỗi có khả năng xảy ra**
- **Kiểm thử lớp và phân cấp của lớp (Class Testing and the Class Hierarchy)**
- **Thiết kế test dựa vào kịch bản (Scenario - Based Test Design)**
  - **Dựa vào những gì người dùng làm (use-case)**

# ...Phương pháp kiểm thử OOT ...

- **Kiểm thử ngẫu nhiên (Random testing)**
  - Xác định những tác vụ có thể áp dụng cho một lớp
  - Xác định những ràng buộc trong việc dùng chúng
  - Xác định một trình tự kiểm thử nhỏ nhất
    - an operation sequence that defines the **minimum life history of the class (object)**
  - Tạo ra một trình tự kiểm thử ngẫu nhiên (nhưng có giá trị)
    - exercise other **(more complex) class instance life histories**

# ...Phương pháp kiểm thử OOT ...

## ■ Kiểm thử Partition

- Làm giảm số test case được yêu cầu để kiểm thử một lớp
- Phân chia dựa vào trạng thái
  - categorize and test operations based on their ability to change the state of a class
- Phân chia dựa vào thuộc tính
  - categorize and test operations based on the attributes that they use
- Phân chia dựa vào loại (category)
  - categorize and test operations based on the generic function each performs

# ...Phương pháp kiểm thử OOT ...

## ■ Kiểm thử tương tác (Inter-class)

- Với mỗi lớp client sử dụng một **danh sách những tác vụ** của lớp để tạo ra **một chuỗi những trình tự test ngẫu nhiên**. Những tác vụ này sẽ gửi thông điệp tới những lớp của server
- Với mỗi thông điệp được tạo xác định lớp cộng tác và tác vụ đáp ứng trong đối tượng server
- Với mỗi tác vụ trên đối tượng server được gọi, **xác định thông điệp được truyền**
- Với mỗi thông điệp được truyền này **xác định mức kế tiếp của những tác vụ** mà được khản nài và kết hợp chúng với trình tự test

# ...Phương pháp kiểm thử OOT

## Kiểm thử Behavior

The tests to be designed should achieve all state coverage [KIR94]. That is, the operation sequences should cause the Account class to make transition through all allowable states

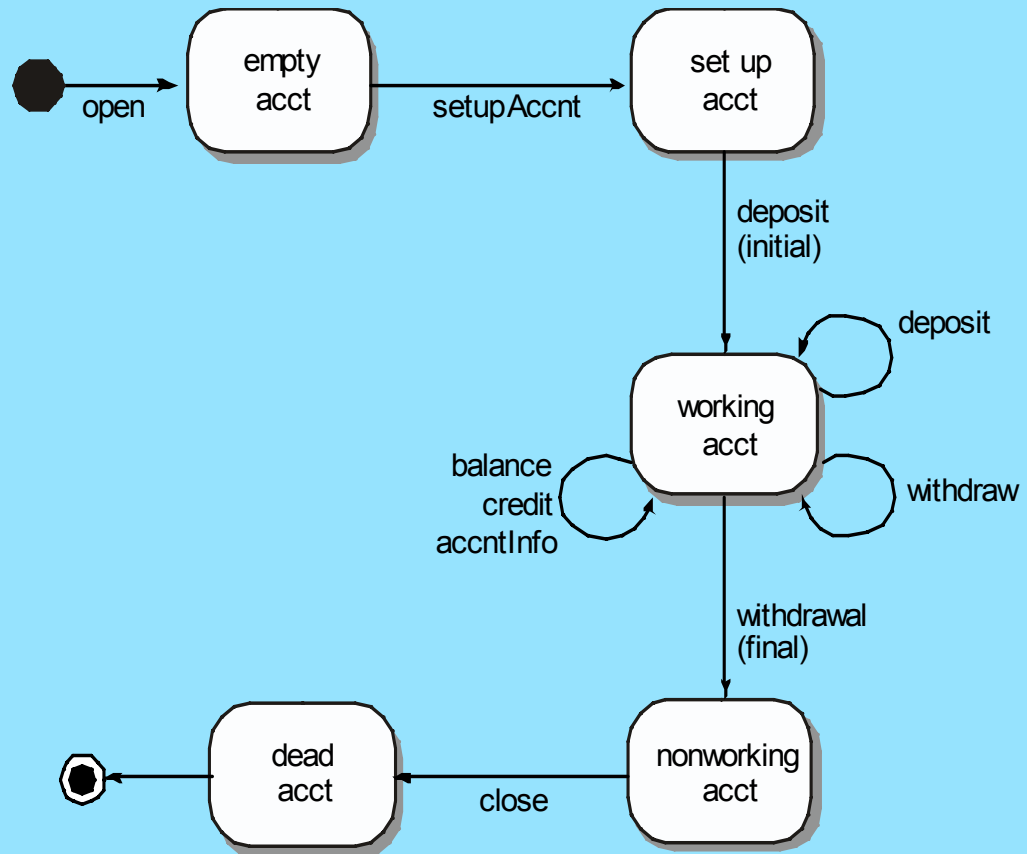


Figure 14.3 State diagram for Account class (adapted from [ KIR94])



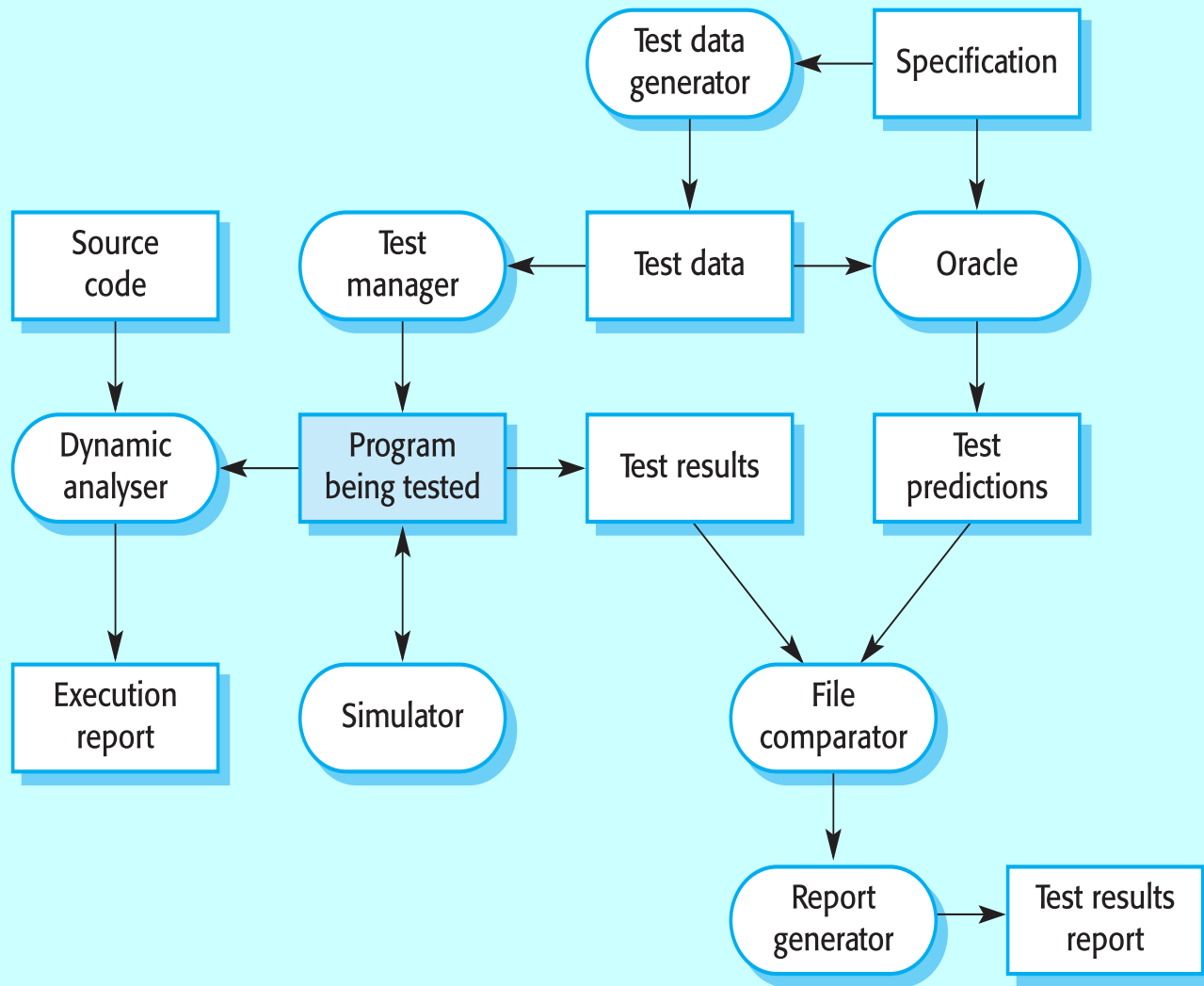
# Các mẫu kiểm thử (Testing Patterns)

- **Kiểm thử cặp (pair testing):** Hai người kiểm thử cùng làm việc với nhau trong thiết kế và thực thi các Test
- **Giao diện test riêng biệt (Separate test interface):** dùng để test những lớp nội (internal classes) là những lớp phô bày giao diện ra bên ngoài

# Tự động kiểm thử

- Kiểm thử là công việc tốn nhiều công sức. Những hệ thống kiểm thử cung cấp những công cụ cho phép giảm thời gian và chi phí
- Phần lớn hệ thống kiểm thử là những hệ thống mở
  - Có thể dùng kịch bản để tạo dữ liệu test
  - Output có thể dùng để so sánh một cách thủ công
  - Có thể phát triển những hệ thống so sánh file

# Một hệ thống kiểm thử



# Kiểm thử kiến trúc, môi trường và ứng dụng đặc trưng

- Kiểm thử GUI: Dùng đồ thị mô hình trạng thái xác định, công cụ kiểm thử tự động
- Kiểm thử client/server:
  - Kiểm thử chức năng ứng dụng
  - Kiểm thử server
  - Kiểm thử cơ sở dữ liệu
  - Kiểm thử truyền thông mạng
- Kiểm thử tư liệu và trợ giúp
- Kiểm thử hệ thống thời gian thực
  - Kiểm thử công việc (task)
  - Kiểm thử hành vi
  - Kiểm thử liên tác vụ (inter task)
  - Kiểm thử hệ thống: interrupt với độ ưu tiên, xử lý interrupt, số lượng interrupt xuất hiện trong một khoảng thời gian...