

Công nghệ phần mềm

Kiểm thử phần mềm

Nội dung

- Khái niệm kiểm thử PM
- Một số đặc điểm của kiểm thử PM
- Tại sao kiểm thử lại cần thiết?
- Quy trình kiểm thử
- Các mức độ test
- Kỹ thuật thiết kế test
- Vai trò của Tester
- Công việc Tester

Verification vs Validation

- Xác minh (Verification)
 - Kiểm tra xem PM làm ra có đúng đặc tả yêu cầu và thiết kế hay không?
- Thẩm định (Validation)
 - Kiểm tra xem PM có đáp ứng được yêu cầu người dùng không?
- Đây là 2 hoạt động chính yếu để đảm bảo chất lượng PM, diễn ra trong toàn bộ quá trình phát triển

Verification vs Validation

- Hai hoạt động chính của Thẩm định và Xác minh là rà soát và kiểm thử
- Có 2 hình thức:
 - Thẩm định và xác minh tĩnh
 - Thẩm định và xác minh động

Thẩm định, xác minh tinh

- Rà soát, xét duyệt các tài liệu PM: kế hoạch, yêu cầu, thiết kế, mã nguồn
- Phát hiện một số loại lỗi nhất định
- Khó đánh giá tính hiệu quả của sản phẩm

Thẩm định, xác minh động

- Thực hiện trên cơ sở cho vận hành sản phẩm PM
 - Làm mẫu yêu cầu
 - Vận hành chương trình (kiểm thử)
 - Mô phỏng hệ thống
- Người phát triển/người dùng trực tiếp kiểm tra đánh giá
- Phát hiện hầu hết lỗi và khiếm khuyết PM → hiệu quả cao

Rà soát phần mềm

- Rà soát là **xem xét, đánh giá sản phẩm được tiến hành mỗi giai đoạn** để phát hiện ra những hạn chế cần sửa trước khi sang giai đoạn sau
- Mục tiêu
 - Chỉ ra các khuyết điểm cần phải cải thiện
 - Khẳng định những sản phẩm đạt yêu cầu
 - Kiểm soát việc đạt chất lượng kỹ thuật tối thiểu của sản phẩm (có diện mạo không đổi, ổn định)
- Áp dụng tại các thời điểm khác nhau trong quá trình phát triển PM

Các hình thức rà soát

- Các loại hình rà soát
 - Thanh tra
 - Họp xét duyệt không chính thức
 - Họp chính thức với KH, nhà quản lý, nhân viên kỹ thuật (rà soát kỹ thuật chính thức – formal technical review (FTR))
- FTR chủ yếu do các kỹ sư PM thực hiện (là một phương tiện hiệu quả để cải thiện chất lượng)

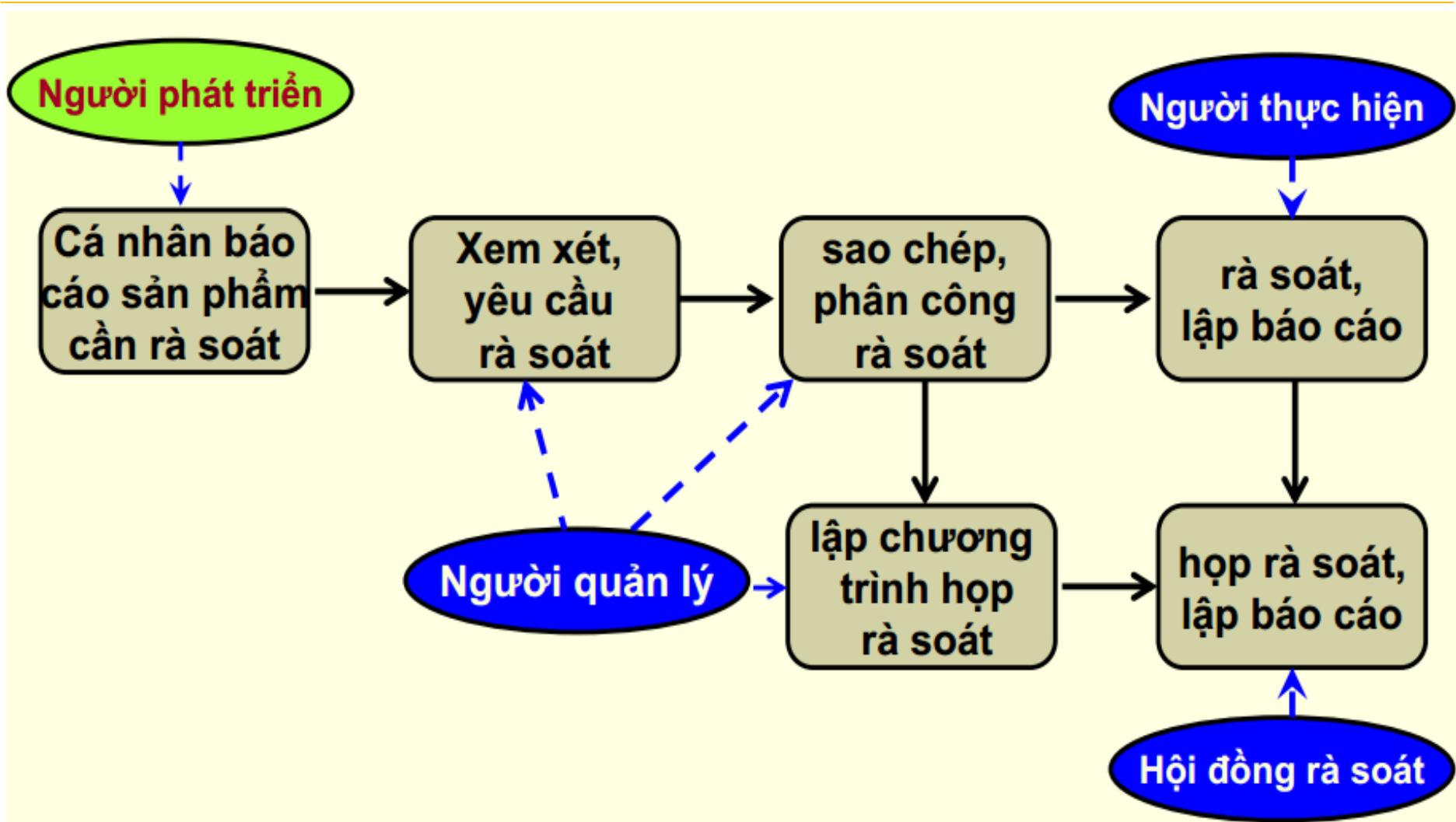
Rà soát kỹ thuật chính thức (FTR)

- FTR là hoạt động đảm bảo chất lượng PM do những người tham gia phát triển thực hiện
- Mục tiêu:
 - Phát hiện các lỗi trong chức năng, logic (chương trình và triển khai)
 - Kiểm thử sự phù hợp của PM với yêu cầu
 - Khẳng định phần đã đạt yêu cầu

Rà soát kỹ thuật chính thức (FTR)

- Bảo đảm PM phù hợp với các chuẩn đã định
- Đảm bảo PM được phát triển theo một cách thức nhất quán (uniform manner)
- Làm cho dự án dễ quản lý hơn
- Làm cơ sở huấn luyện các kỹ sư trẻ, và có ích ngay cả cho các kỹ sư đã có kinh nghiệm

Tiến trình hoạt động rà soát



Hợp rà soát

- Thành phần: lãnh đạo rà soát, các cá nhân rà soát, người tạo ra sản phẩm được rà soát, KH
- Kết luận: 1 trong 3 quyết định sau:
 - Chấp nhận sản phẩm không cần chỉnh sửa
 - Chấp nhận cho chỉnh sửa sản phẩm, sẽ rà soát lại
 - Từ chối sản phẩm vì các lỗi nghiêm trọng

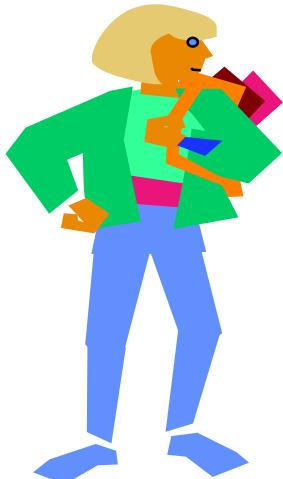
Hợp rà soát

- Sản phẩm của cuộc họp rà soát
 - 1 báo cáo các vấn đề nảy sinh do cá nhân rà soát nêu ra
 - 1 danh sách các vấn đề cần giải quyết
 - 1 bản tổng kết cuộc họp (phải có đủ chữ ký các thành viên)
- Bản tổng kết phải chỉ rõ:
 - Đã rà soát cái gì?
 - Ai rà soát?
 - Tìm thấy gì và kết luận ra sao?

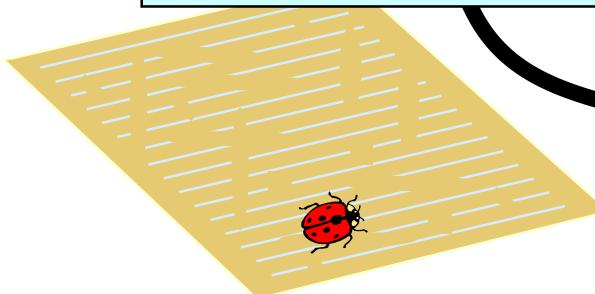
Khái niệm kiểm thử PM

- Kiểm thử là gì?

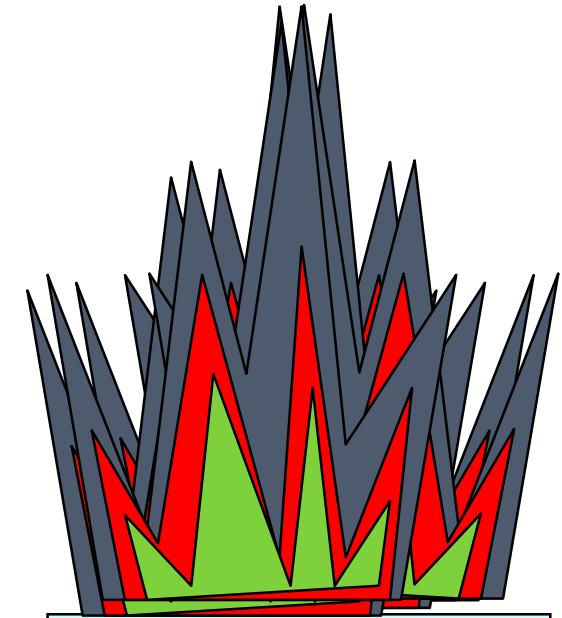
A person makes
an error ...



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



... that can
cause a failure
in operation



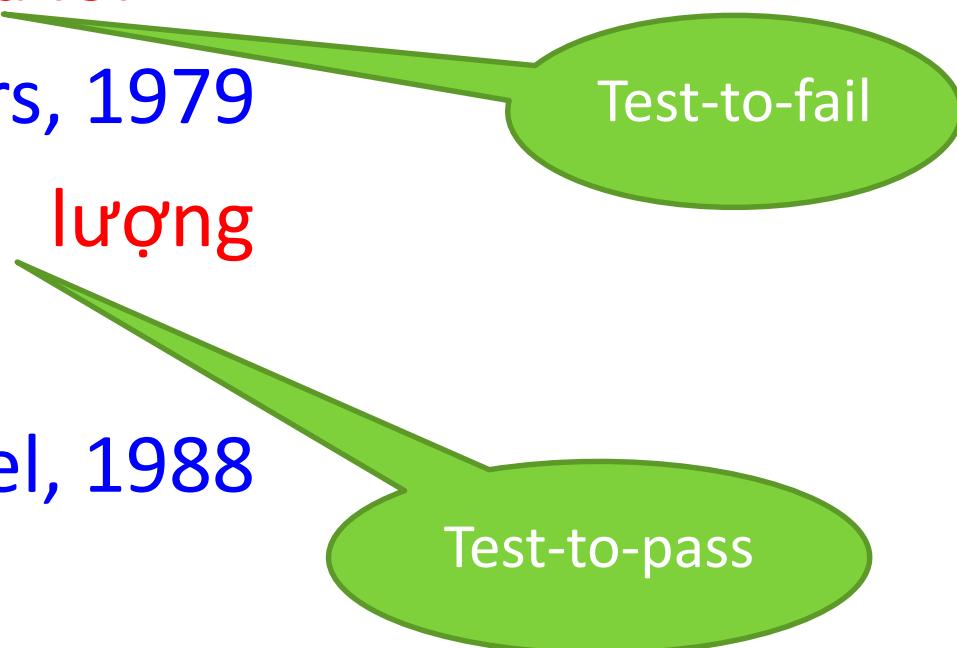
Khái niệm kiểm thử PM

- Kiểm thử PM là quá trình thực thi PM với mục tiêu **tìm ra lỗi**

Glen Myers, 1979

- Khẳng định được **chất lượng** của PM đang xây dựng

Hetzl, 1988



Test-to-fail

Test-to-pass

Một số đặc điểm kiểm thử PM

- Kiểm thử PM 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 PM hoặc phát hiện quá ít lỗi
 - PM có chất lượng quá tốt
 - Quy trình/Đội ngũ kiểm thử hoạt động không hiệu quả

Một số đặc điểm kiểm thử PM

- Vấn đề quan trọng cần cân nhắc:
 - Test được nhiều trường hợp càng tốt, nhưng chi phí chỉ có hạn
- Mục tiêu là tìm lỗi được càng nhanh càng tốt và càng rẻ càng tốt.
 - Lí tưởng: với mỗi một lỗi, ta chỉ cần thiết kế một test case để phát hiện lỗi đó rồi chạy nó
- Thực tế, ta phải chạy nhiều test case “thất bại” - chúng không phát hiện được lỗi

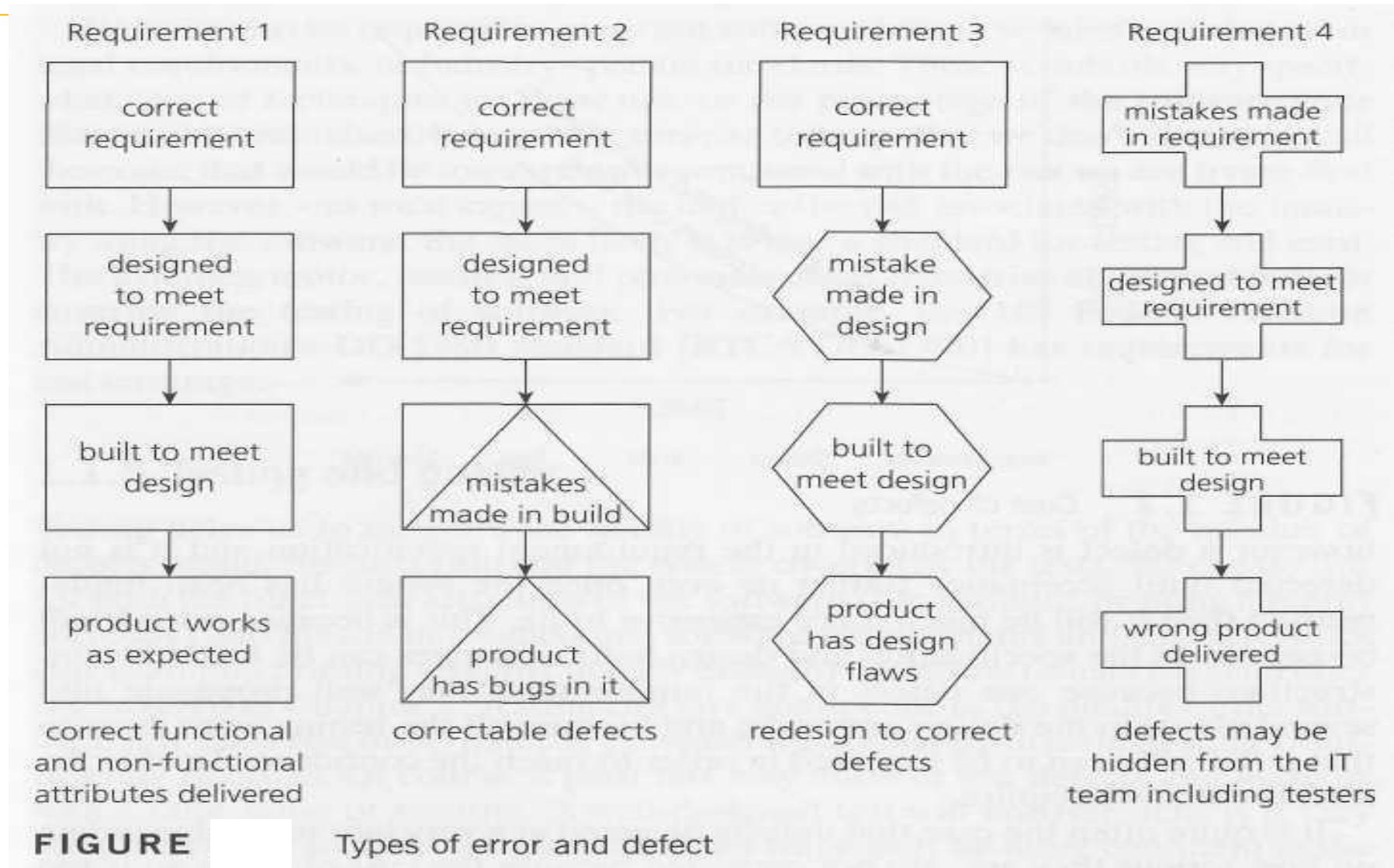
Tại sao kiểm thử lại cần thiết?

- Thông thường thì PM không hoạt động như mong muốn
→ lãng phí tiền bạc, thời gian, uy tín của doanh nghiệp, thậm chí có thể gây nên thương tích hay cái chết.
- Ví dụ:
 - Website công ty có nhiều lỗi chính tả trong câu chữ → KH có thể lảng tránh công ty với lý do công ty trông có vẻ không chuyên nghiệp.
 - Một PM 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,....

Tại sao kiểm thử lại cần thiết?

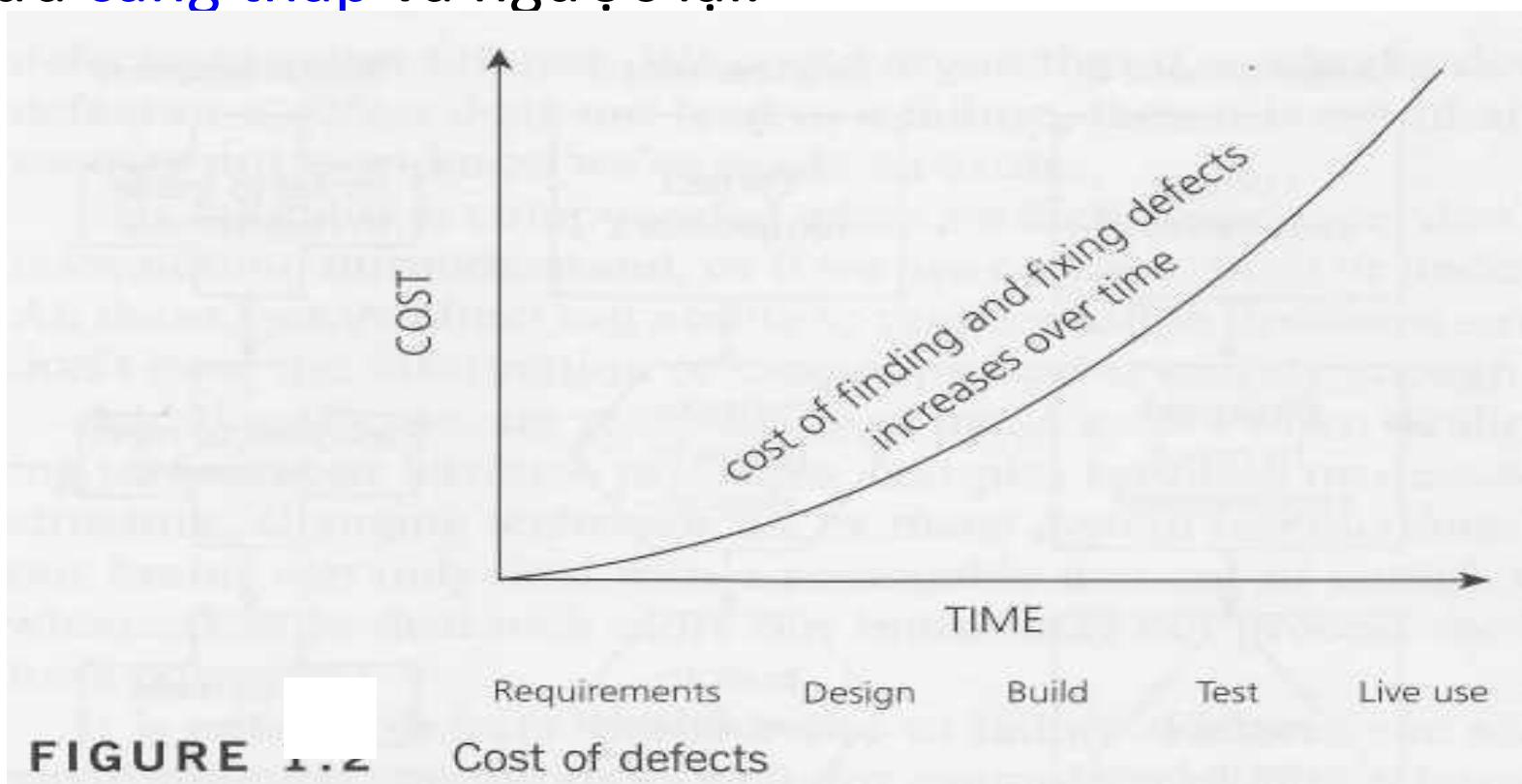
- Kiểm thử PM → chất lượng PM được nâng cao.
- Chúng ta có thể đánh giá chất lượng PM dựa vào số lượng lỗi tìm thấy và các đặc tính như: tính đúng đắn, tính dễ sử dụng, tính dễ bảo trì,...
- Kiểm thử có thể đem lại sự tin tưởng đối với chất lượng PM nếu có ít lỗi hoặc không có lỗi nào được tìm thấy. Nếu lỗi tìm thấy và được sửa thì chất lượng PM càng được tăng → Giảm chi phí trong quá trình phát triển, nâng cấp, bảo trì PM

Lỗi tăng lên khi nào?



Lỗi tăng lên khi nào?

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



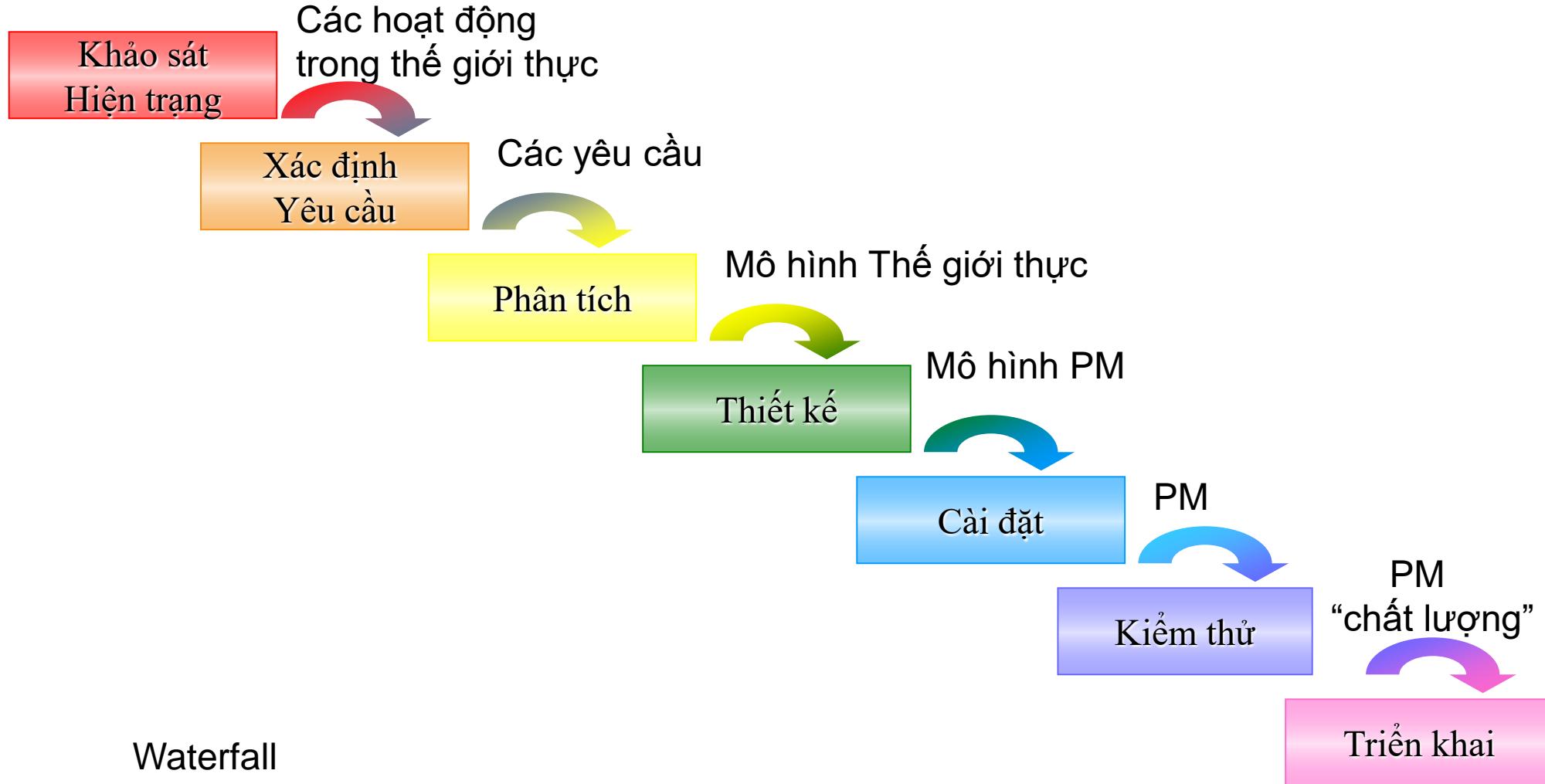
Các hoạt động của kiểm thử

- Các hoạt động của kiểm thử tồn tại cả trước và sau khi thực thi PM như:
 - Lập kế hoạch test (test plan)
 - Chọn các điều kiện test (test conditions)
 - Thiết kế các trường hợp test (test cases)
 - Kiểm tra kết quả, ước lượng khi nào thì dừng test.
 - Báo cáo kết quả test.

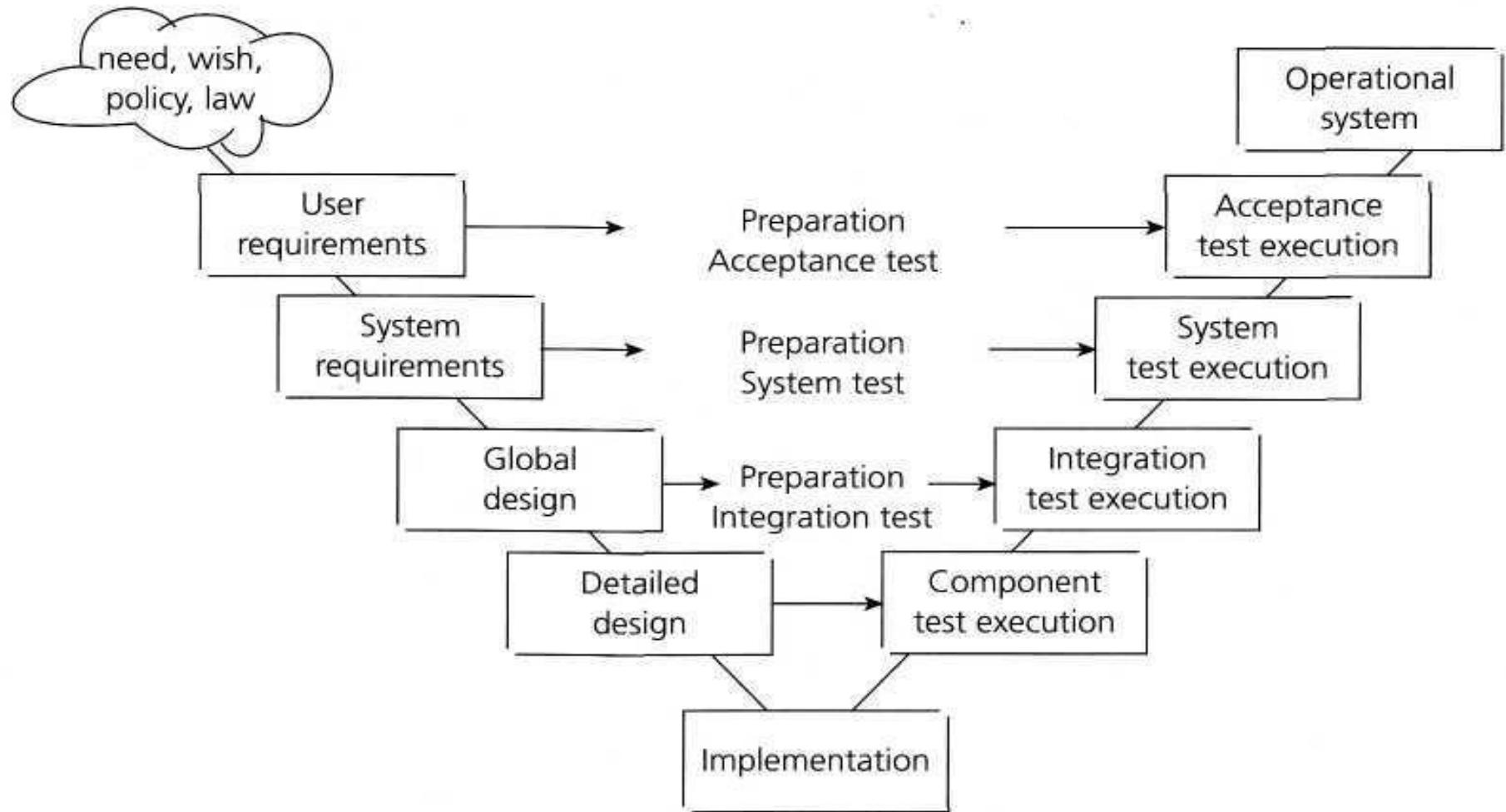
Vai trò kiểm thử

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

Mô hình thác nước



Mô hình chữ V



FIGURE

V-model

Mô hình phát triển lặp

- Chúng ta có thể chia nhỏ PM ra làm **nhiều giai đoạn** thay vì làm một lần từ đầu đến cuối. Mô hình này cần các hoạt động test như: **test chức năng mới**, **test lặp lại** cho những chức năng cũ, và **integration test** cho cả phần cũ và phần mới.

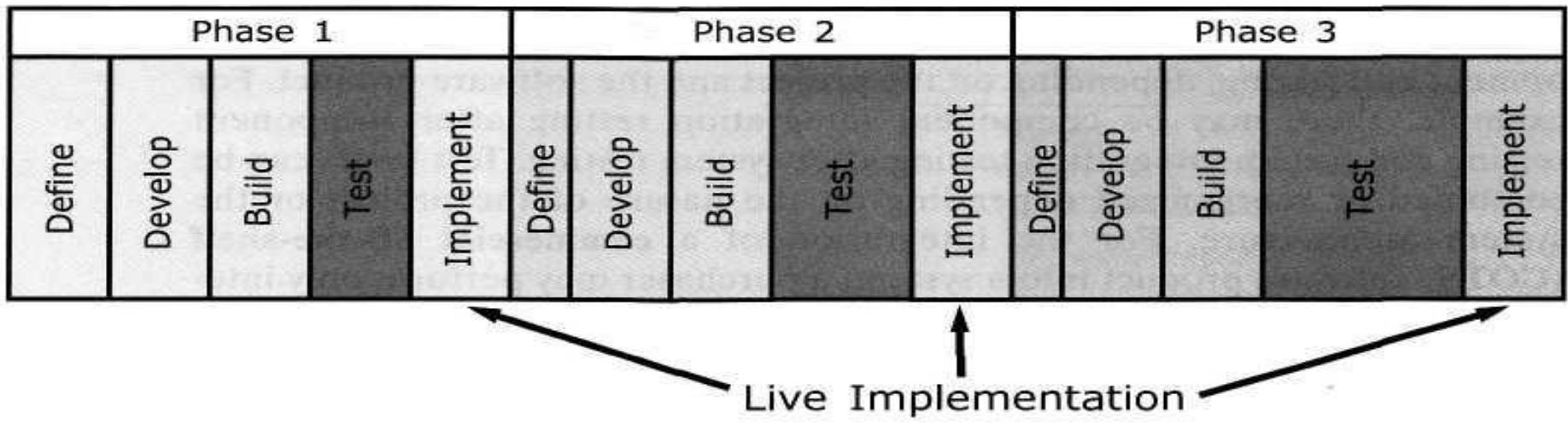
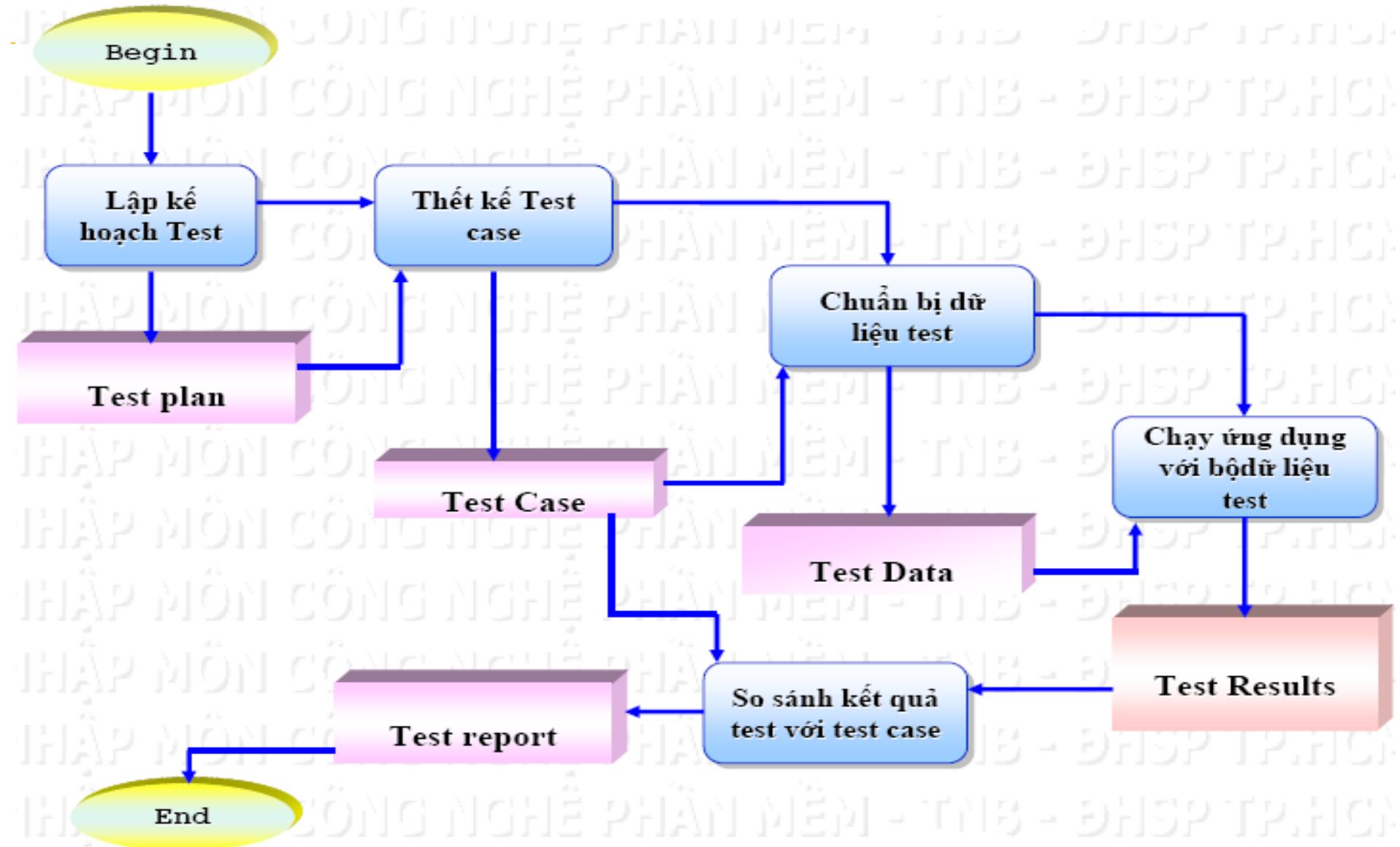
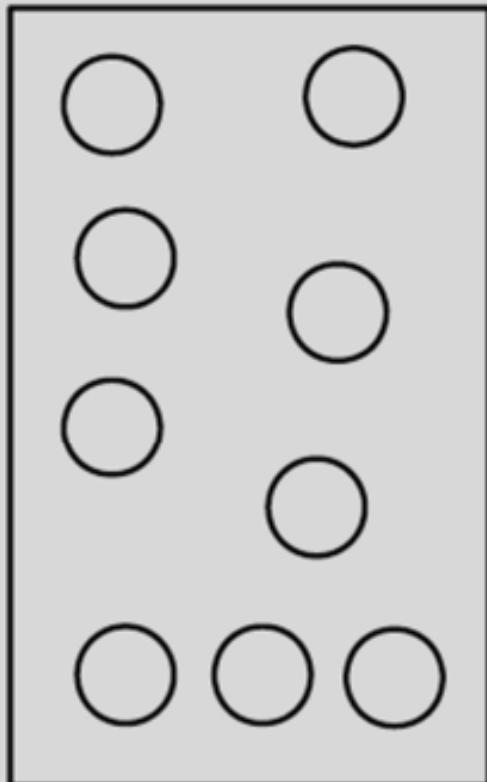


FIGURE 2.3 Iterative development model

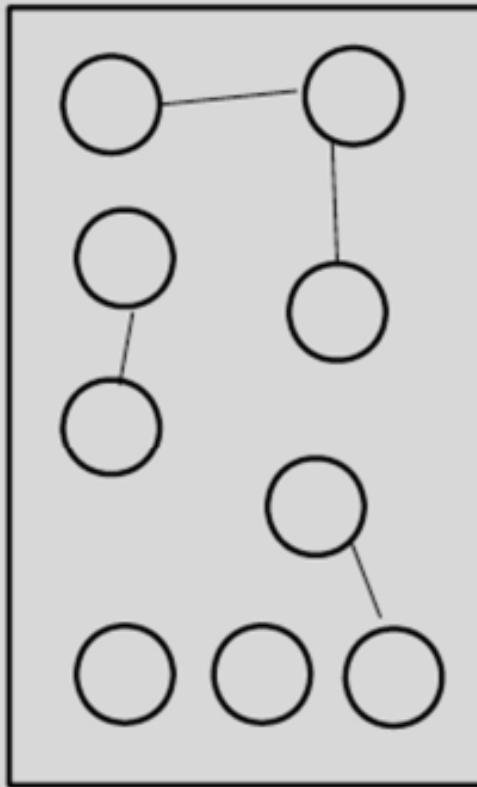
Qui trình kiểm thử PM



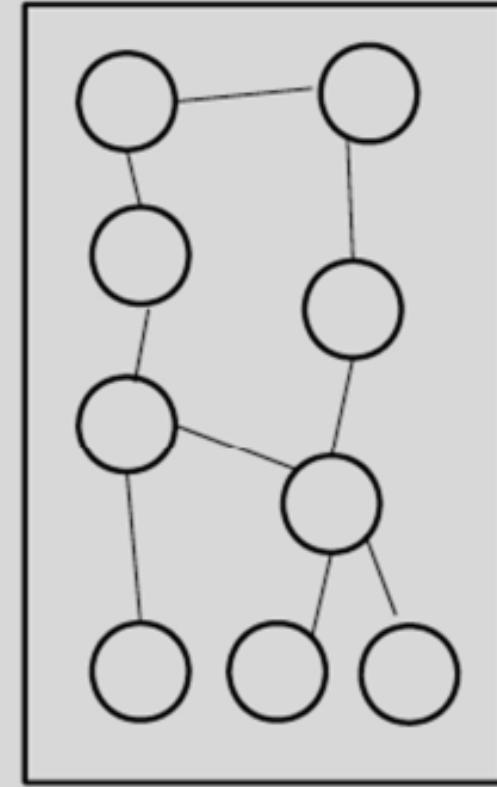
Các mức độ test (Test levels)



Unit testing



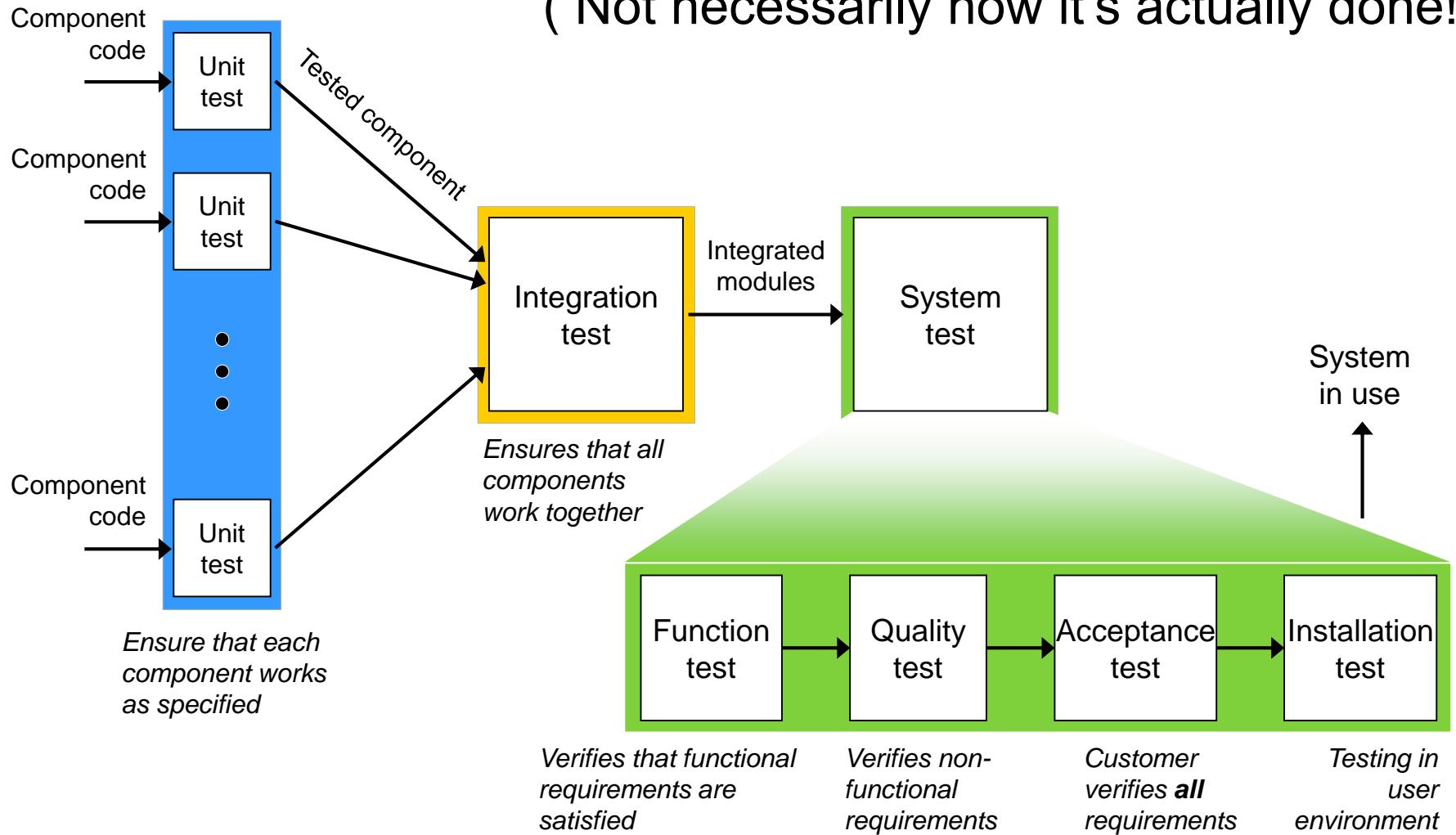
Integration testing



System testing

Quá trình kiểm thử

(Not necessarily how it's actually done!)

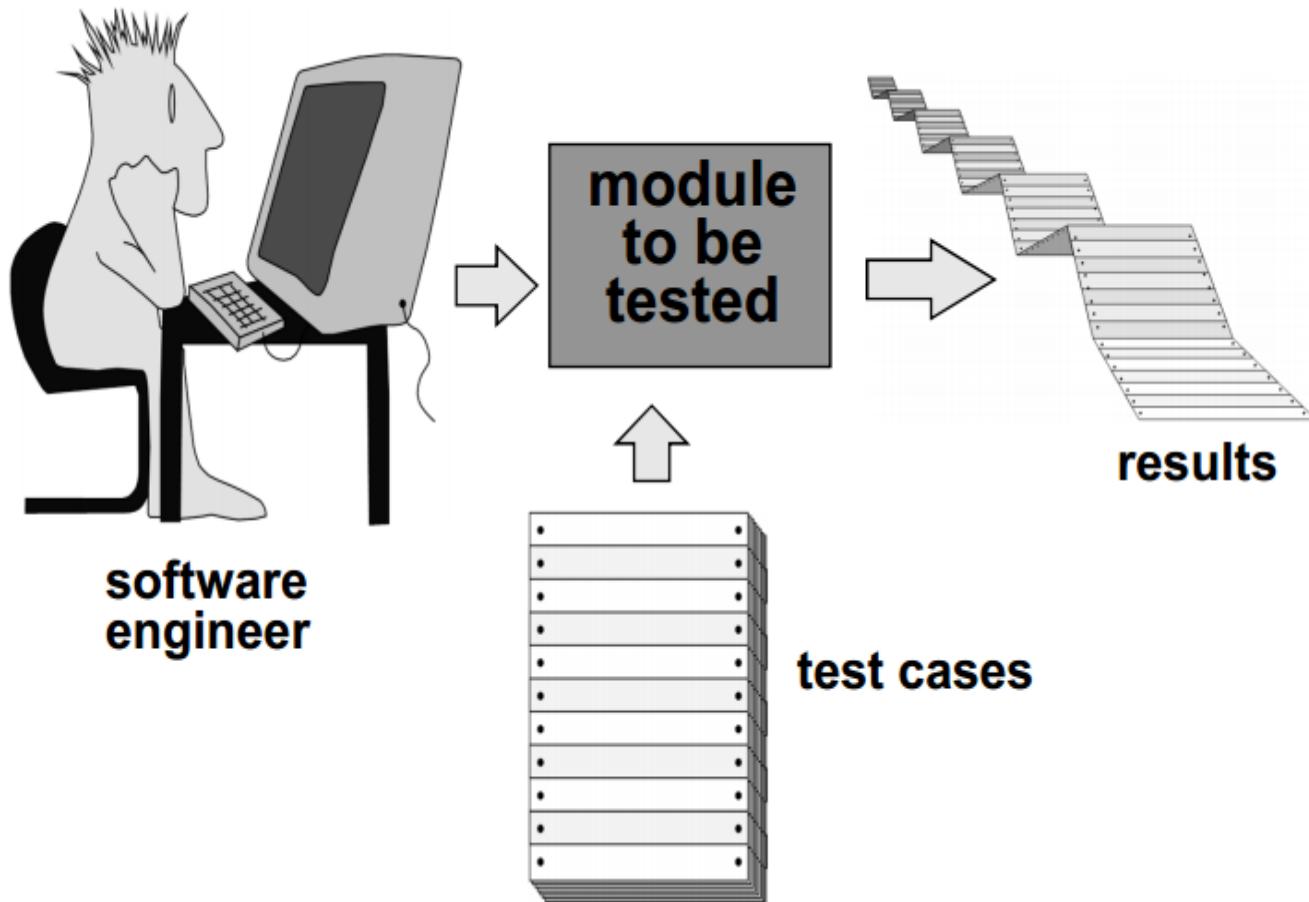


Các mức độ test (Test levels)

- Component testing (unit testing):

–Tìm lỗi trong các component của PM như: modules, programs, objects, classes,...

–Ai thực hiện?



Unit Testing

- Nội dung kiểm thử
 - Giao diện: DL qua giao diện, DL vào ra
 - CTDL sử dụng cục bộ
 - Đường điều khiển
 - Điều kiện logic
 - Phép toán xử lý
- Phương pháp sử dụng: white box

Các mức độ test (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,...
 - Nhằm nhận được 1 bộ phận chức năng/ 1 hệ con tốt
 - Có 2 cách tích hợp:
 - Tích hợp dần: từ trên xuống, dưới lên, kẹp
 - Tích hợp đồng thời 1 lúc: “bigbang”
 - Phương pháp: black box

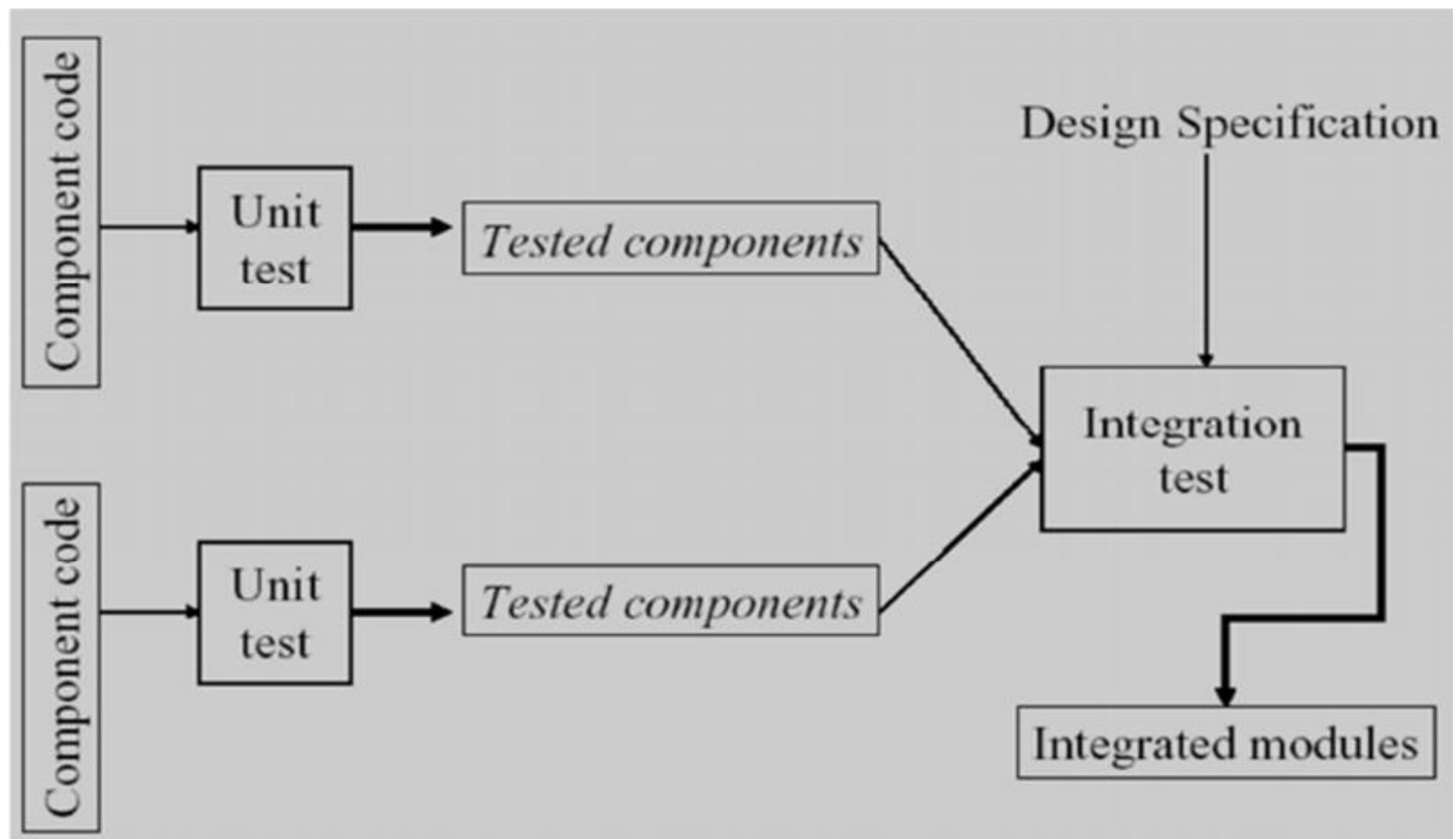
Các mức độ test (Test levels)

- **Integration testing:** Các vấn đề có thể gặp:
 - DL bị mất khi đi qua một giao diện
 - Hiệu ứng bất lợi 1 module vô tình gây ra đối với các module khác
 - Sự kết hợp các chức năng phụ có thể không sinh ra chức năng chính mong muốn
 - Sự phóng đại các sai sót riêng lẻ có thể bị đến mức không chấp nhận được
 - Vấn đề của CTDL toàn cục có thể để lộ ra

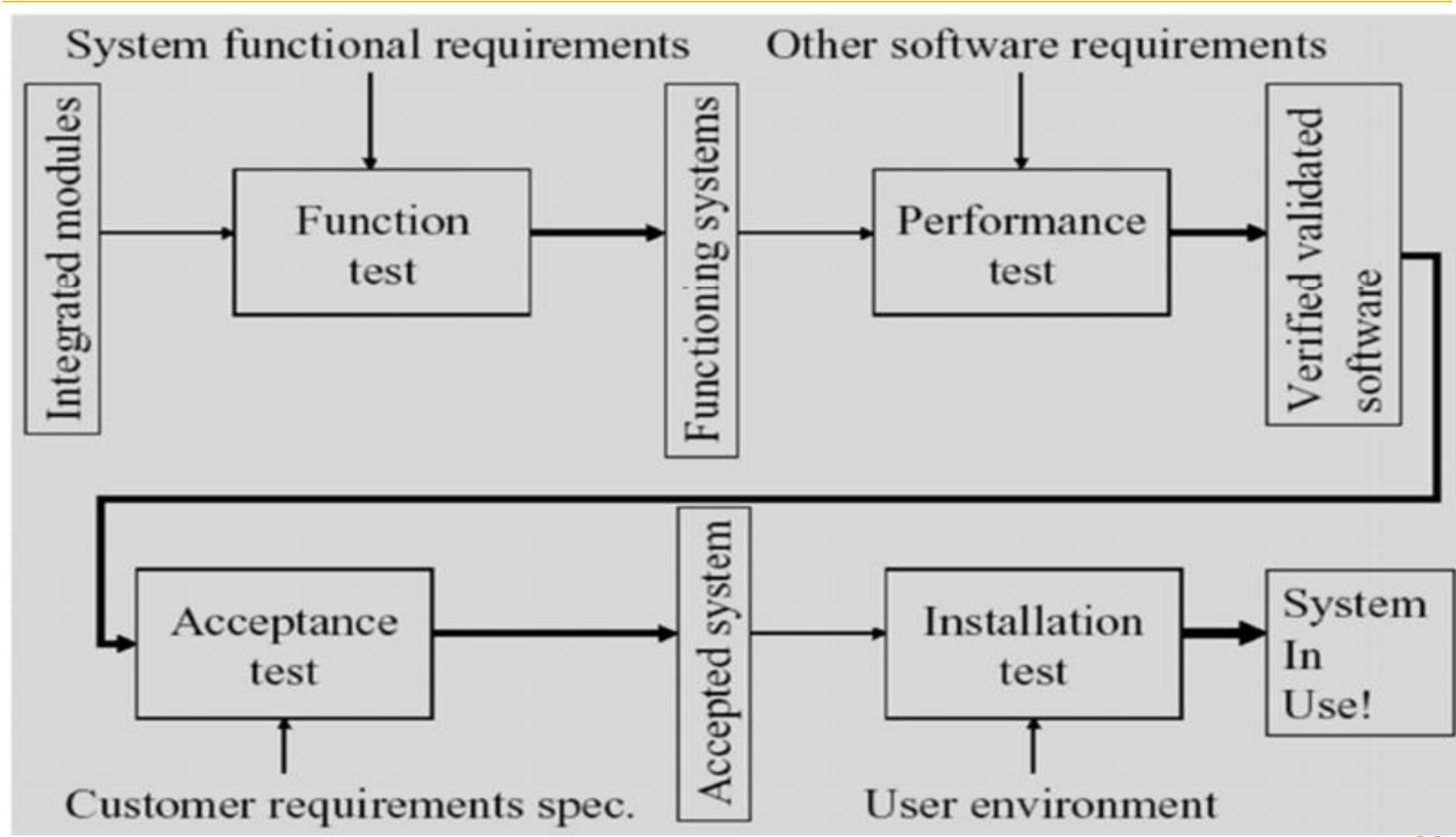
Các mức độ test (Test levels)

- System testing:
 - Hệ thống dựa trên máy tính (phần cứng & PM) do **nhiều bên xây dựng**, người phát triển PM chỉ là **1** => Chúng cần được kiểm tra tổng thể
 - 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.

System testing



System testing



Các mức độ test (Test levels)

- **System testing:** Các vấn đề cần kiểm tra:
 - Các DL qua giao diện của các thành phần được kiểm thử
 - Đường xử lý liên kết các thành phần
 - Sự tích hợp lỗi từ các thành phần khác nhau
 - Những hạn chế khác đến năng lực do ảnh hưởng từ các thành phần: chịu lỗi, an toàn, thực thi.

Các loại kiểm thử hệ thống

- Kiểm thử chức năng (mức hệ thống)
 - Các chức năng giao diện
 - Các chức năng mức người dùng
 - Đầu ra cuối cùng khỏi hệ thống
- Kiểm thử phục hồi (khả năng chịu lỗi)
 - Bắt PM phải thất bại để xem khả năng phục hồi của nó đến đâu. Có 2 mức phục hồi: tự động, nhân tạo
 - Độ tin cậy là một độ đo đánh giá khả năng phục hồi

Các loại kiểm thử hệ thống

- Kiểm thử an ninh (khả năng chịu tấn công)
 - Kiểm tra mọi cơ chế bảo vệ được xây dựng xem có đạt hiệu quả đề ra trước các đột nhập hay không?
 - Người kiểm thử đóng vai trò của kẻ đột nhập thực hiện mọi đột nhập có thể đánh giá
- Kiểm thử thi hành
 - Kiểm tra sự vận hành của PM khi hệ thống được tích hợp
 - Việc thi hành đúng bao gồm cả số lượng và chất lượng

Các loại kiểm thử hệ thống

- Kiểm thử chịu tải
 - Vận hành hệ thống khi sử dụng nguồn lực với số lượng, tần suất và cường độ **khác thường**
 - **VD:** Vận hành 1 CSDL với kích thước cực lớn, vận hành HĐH mạng với số máy nhiều dần

Các mức độ test (Test levels)

- Acceptance testing: đứng theo góc độ người dùng để xác định PM có được chấp nhận hay không.
- Thực hiện thông qua 1 loạt các kiểm thử black box
- Kế hoạch và thủ tục được thiết kế bảo đảm rằng:
 - Tất cả các yêu cầu được thỏa mãn
 - Các yêu cầu thi hành đã chính xác
 - Tài liệu đúng đắn
 - Các yêu cầu khác là thỏa đáng
- Có 2 loại kiểm thử alpha va beta

Kiểm thử Alpha và Beta

- Kiểm thử Alpha:
 - Được tiến hành ngay tại nơi sản xuất PM
 - Nhà phát triển PM sẽ quan sát người sử dụng sản phẩm và ghi nhận lại những lỗi phát sinh để sửa chữa
- Kiểm thử Beta:
 - Mở rộng của kiểm thử Alpha
 - PM được kiểm tra bên ngoài phạm vi của đơn vị sx
 - Khách hàng trực tiếp sử dụng và ghi nhận lỗi để báo cáo lại cho nhà phát triển sửa chữa.

Một số kỹ thuật test

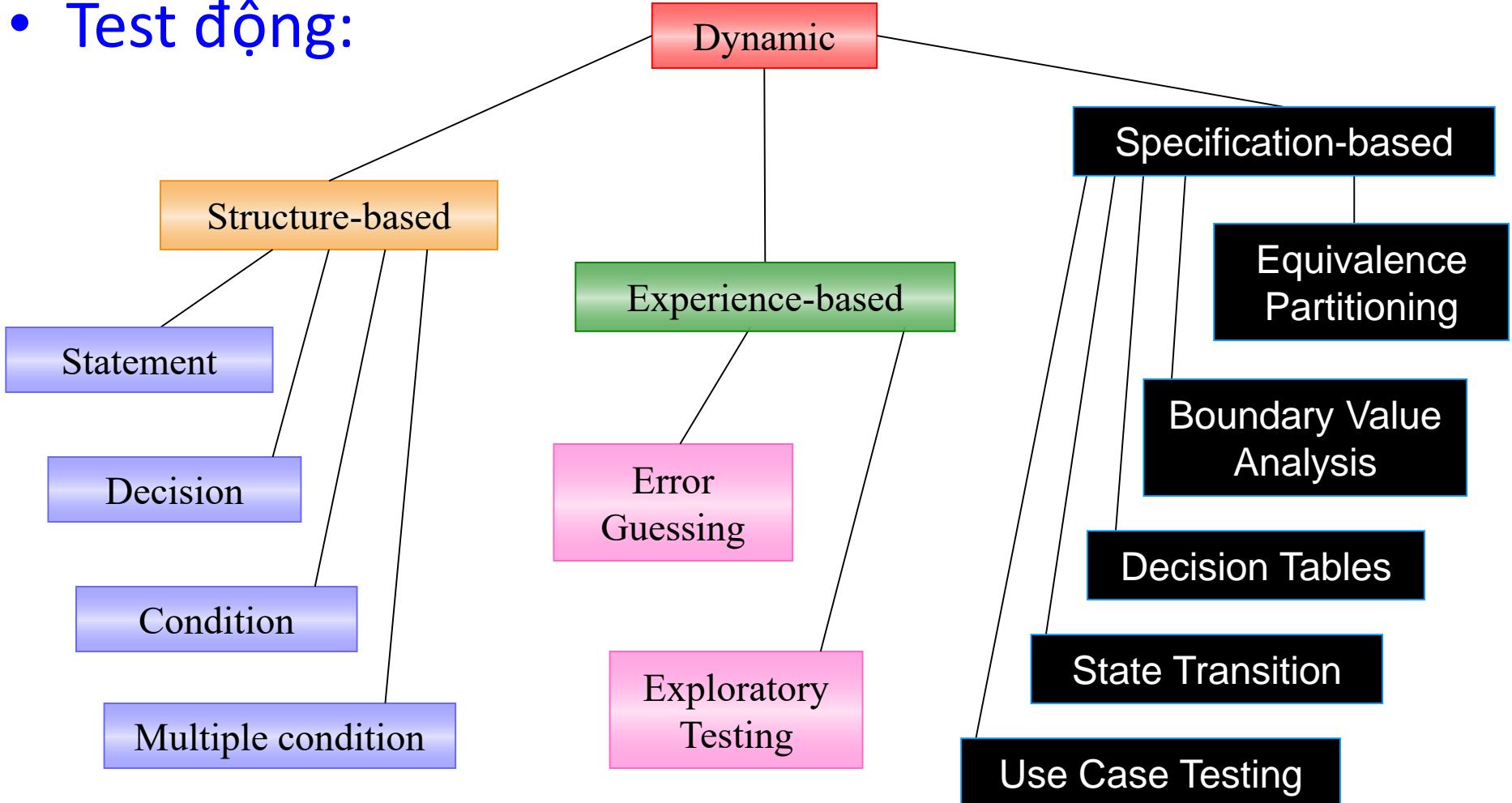
- Test tĩnh:
 - Dựa vào việc kiểm tra tài liệu, source code,... mà không cần phải thực thi PM.
 - Các lỗi được tìm thấy trong quá trình kiểm tra có thể dễ dàng được loại bỏ và chi phí rẻ hơn nhiều so với khi tìm thấy trong test động. Một số lợi ích khi thực hiện việc kiểm tra (reviews):
 - Lỗi sớm được tìm thấy và sửa chữa
 - Giảm thời gian lập trình
 - Giảm thời gian và chi phí test

Một số kỹ thuật test

- Test tĩnh (tt):
 - Các tài liệu được kiểm thử:
 - Tài liệu đặc tả yêu cầu
 - Tài liệu đặc tả thiết kế
 - Sơ đồ luồng dữ liệu
 - Mô hình ER
 - Source code
 - Test case
 - ...

Một số kỹ thuật test

- Test động:



Một số kỹ thuật test

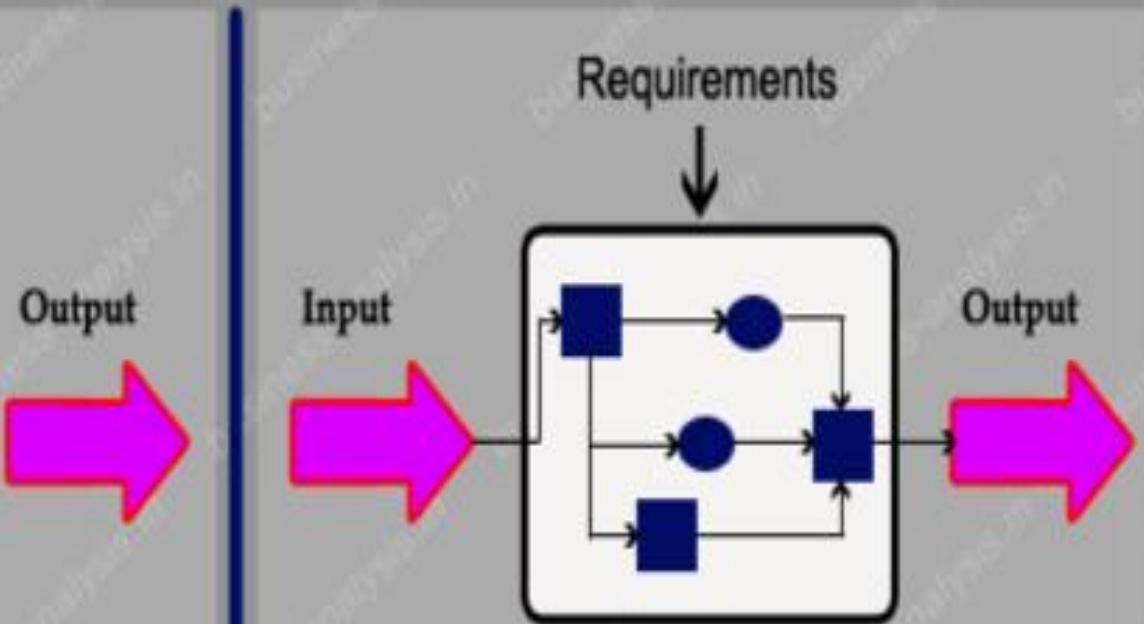
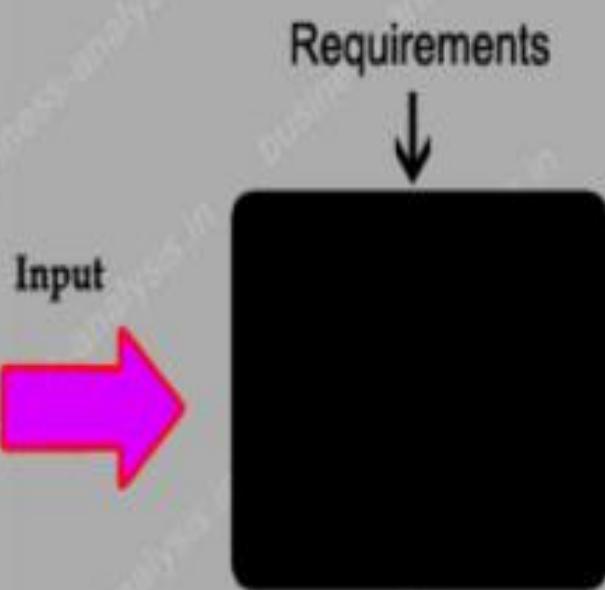
- Test động:
 - Test dựa trên mô tả (specification-based) hay còn gọi test chức năng (functional testing): Test những gì mà PM phải làm, không cần biết PM làm như thế nào (kỹ thuật black box)
 - Test dựa trên cấu trúc (structure-based) hay còn gọi test phi chức năng (non-functional testing): Test PM hoạt động như thế nào (kỹ thuật white box)
 - Test dựa trên kinh nghiệm (experience-based): đòi hỏi sự hiểu biết, kỹ năng và kinh nghiệm của người test

Khi nào thì dừng kiểm thử?

- Một số tiêu chí
 - Khi hết sai sót
 - Khi hết thời gian/ngân sách/tài nguyên
 - Khi kiểm thử không tìm thêm được sai sót mới
 - Khi các tiêu chuẩn kiểm thử đã tạo ra hết các ca kiểm thử
 - Khi tiêu chuẩn đặt ra đã đạt được
 - Khi tần suất sai phát hiện giảm xuống một ngưỡng

Một số kỹ thuật test

Black Box Testing Vs White Box Testing



While Box Testing

- Kiểm tra **tính logic** và **cấu trúc** của mã nguồn: gồm server code và client code
- Tester cần phải có kiến thức về ngôn ngữ lập trình (C/C++/C#/VB.Net/Java...), môi trường phát triển PM (IDE), cũng như các **hệ QTCSQL** (SQL server, Oracle, DB2, MySQL...)
- **Kiểm tra tất cả** các trường hợp có thể xảy ra trong mã nguồn (**cấu trúc điều khiển**, **cấu trúc lặp...**)

While Box Testing

- Ví dụ: Cho đoạn mã C/C++ như sau:

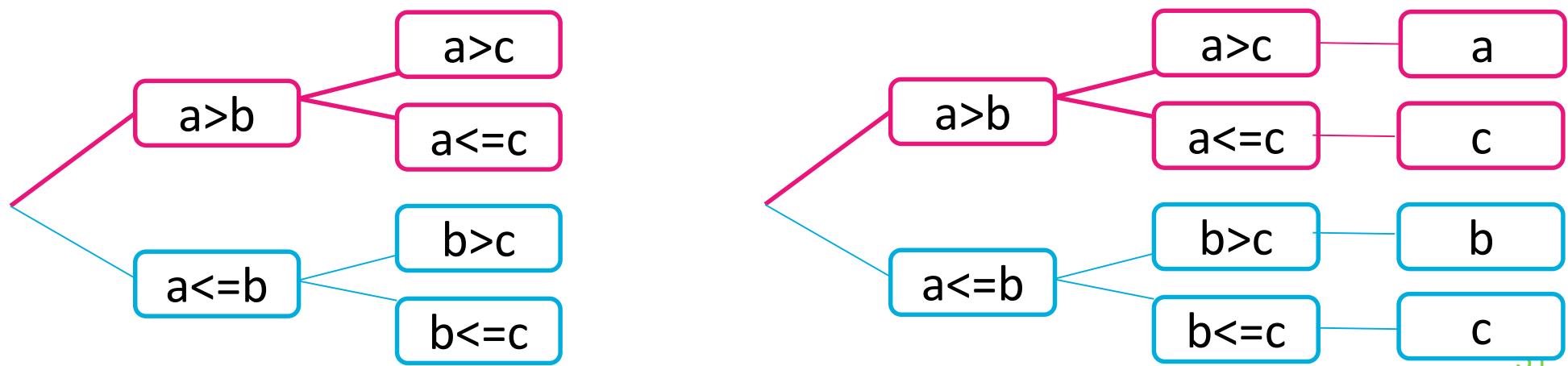
```
int Test(int a, int b, int c) {  
    if (a>b) {  
        if (a>c) return a;  
        else return c; } •  
    else {  
        if (b>c) return b;●  
        else return c; }  
}
```

Để kiểm tra tính đúng đắn của hàm Test trên, ta cần ít nhất bao nhiêu trường hợp?

While Box Testing

- Ví dụ: Cho đoạn mã C/C++ như sau:

```
int Test(int a, int b, int c) {  
    if (a>b) {  
        if (a>c) return a;  
        else return c; }  
    else {  
        if (b>c) return b;  
        else return c; }  
}
```



While Box Testing

- Ví dụ: Cho đoạn mã C/C++ như sau:

```
int Test(int a, int b, int c) {  
    if (a>b) {  
        if (a>c) return a;  
        else return c; }  
    else {  
        if (b>c) return b;  
        else return c; }  
}
```

Ta cần ít nhất 4 trường hợp (Test case), chẳng hạn như:

- **a = 6, b = 3, c = 4**
- **a = 6, b = 3, c = 7**
- **a = 3, b = 4, c = 2**
- **a = 3, b = 4, c = 6**

Black Box Testing

- Kiểm tra hệ thống dựa trên **bản đặc tả yêu cầu và chức năng**
- Tester **không cần phải** có kiến thức về ngôn ngữ lập trình (C/C++/C#/VB.Net/Java...), môi trường phát triển PM (IDE), cũng như các hệ QTCSQL (SQL server, Oracle, DB2, MySQL...)
- Tester thao tác các chức năng của hệ thống **như là một end-user**

Black Box Testing

- Dựa vào chức năng nhằm phát hiện lỗi:
 - Thiếu chức năng
 - Lỗi giao diện
 - Lỗi trong CTDL
 - Lỗi khi thực hiện

Black Box Testing

VD: Kiểm tra màn hình sau:

Max, Min (A, B, C)

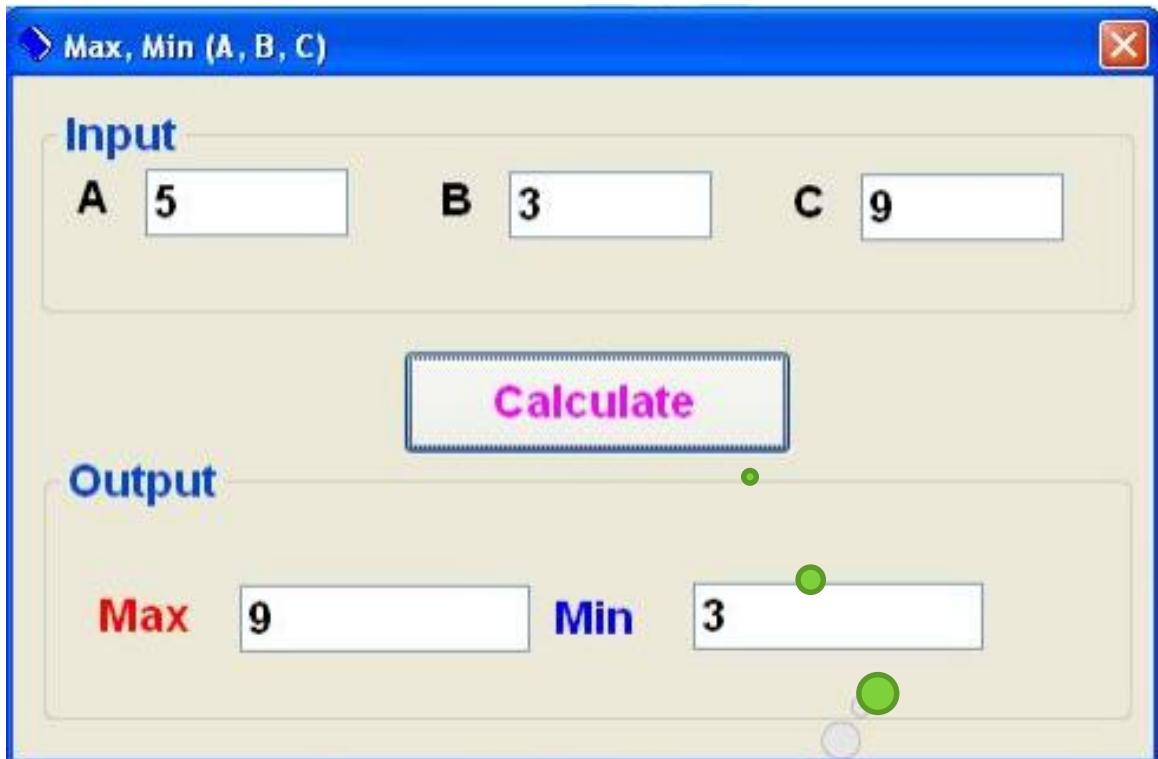
Input

A 5 B 3 C 9

Calculate

Output

Max 9 Min 3



Để kiểm tra tính đúng đắn của hàm Test trên,
ta cần ít nhất bao nhiêu trường hợp?

Black Box Testing

VD: Kiểm tra màn hình sau:

Max, Min (A, B, C)

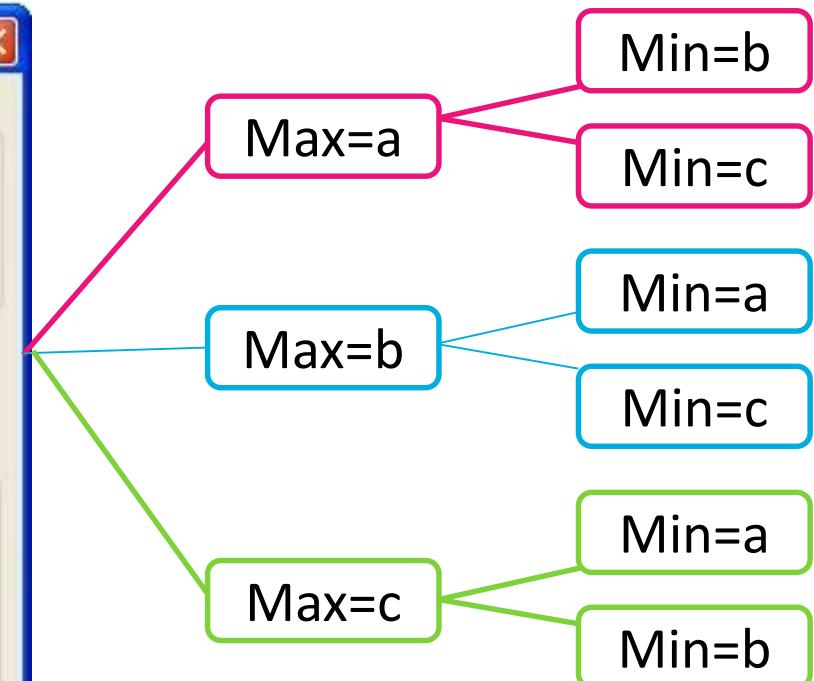
Input

A	5
B	3
C	9

Calculate

Output

Max	9
Min	3



Black Box Testing

VD: Kiểm tra màn hình sau:

Max, Min (A, B, C)

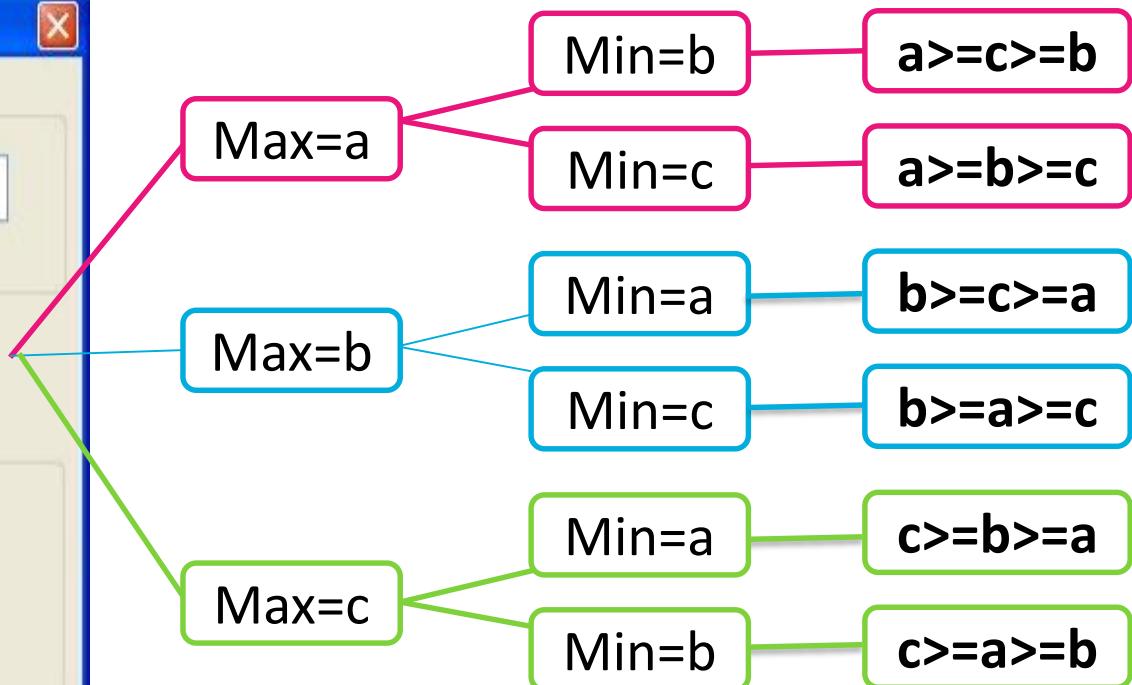
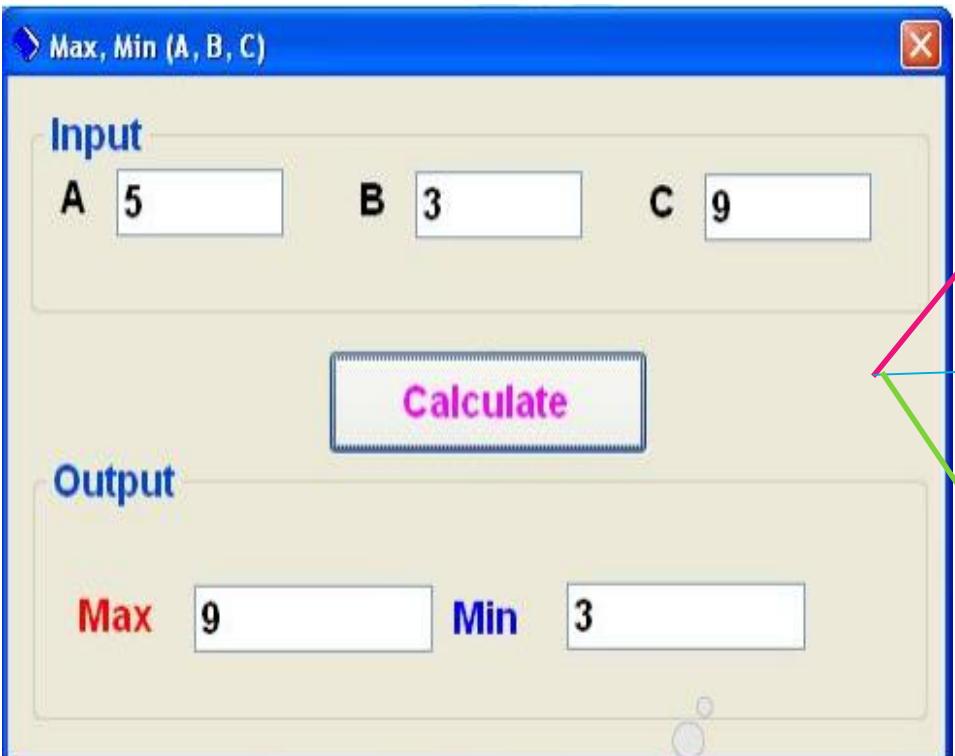
Input

A	5
B	3
C	9

Output

Max	9
Min	3

Calculate



Black Box Testing

VD: Kiểm tra màn hình sau:

Max, Min (A, B, C)

Input

A 5 B 3 C 9

Output

Max 9 Min 3

Calculate

Để kiểm tra màn hình trên ta cần ít nhất 6 trường hợp (test case), chẳng hạn như:

- a=5, b=4, c=2
- a=5, b=2, c=4
- b=5, a=4, c=2
- b=5, c=2, a=4
- c=5, a=4, b=2
- c=5, a=2, b=4

White Box vs Black Box Testing

- **Hộp trắng**
 - Số đường đi nhiều khi là vô hạn
 - Kiểm tra những gì đã làm, không phải những gì cần được làm
 - Không phát hiện được ca kiểm thử còn thiếu
 - thích hợp cho kiểm thử hệ thống và tích hợp
- **Hộp đen**
 - Dễ bùng nổ tổ hợp về số ca kiểm thử (dữ liệu đúng và dữ liệu sai)
 - Thường không chắc ca kiểm thử này có phát hiện được lỗi cụ thể kia hay không
 - Thích hợp cho kiểm thử đơn vị và tích hợp.

White Box vs Black Box Testing

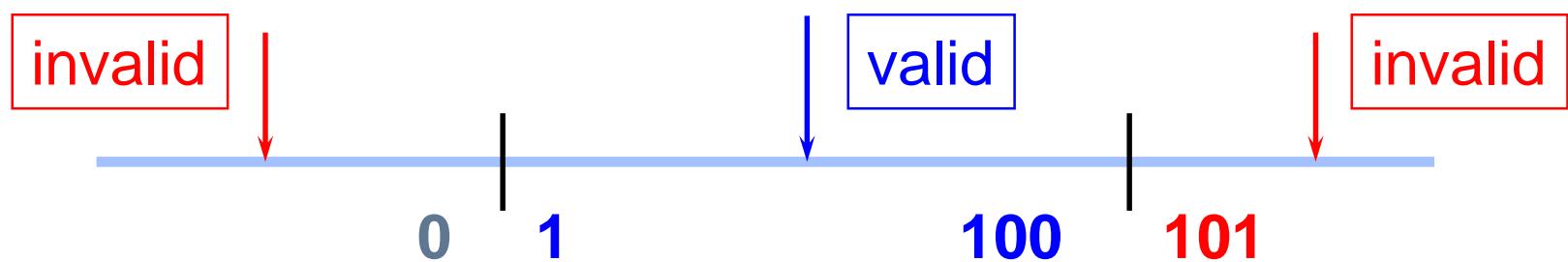
- Cần cả hai
- Kiểm thử hộp trắng và hộp đen là hai thái cực của kiểm thử
- Việc lựa chọn ca kiểm thử nằm giữa và phụ thuộc vào
 - Số đường đi logic có thể
 - Tính chất của dữ liệu đầu vào
 - Khối lượng tính toán
 - Độ phức tạp của cấu trúc dữ liệu và giải thuật
- Hai kỹ thuật là bổ sung cho nhau.

Kỹ thuật specification-based

- Kỹ thuật phân vùng tương đương – EP (Equivalence Partitioning)
 - 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). Nếu một giá trị trong nhóm hoạt động đúng thì tất cả các giá trị trong nhóm đó cũng hoạt động đúng và ngược lại.

Kỹ thuật specification-based

- Kỹ thuật phân vùng tương đương – EP (tt)
 - Trong ví dụ trên dùng kỹ thuật phân vùng tương đương, chia làm 3 phân vùng như sau:



- Như vậy chỉ cần chọn 3 test case để test trường hợp này: -5, 55, 102 hoặc 0, 10, 1000, ...

Kỹ thuật specification-based

- Kỹ thuật phân vùng tương đương – EP (tt)
 - Tuy nhiên nếu ta nhập vào số thập phân (55.5) hay một ký tự không phải là số (abc)?
 - Trong trường hợp trên có thể chia làm 5 phân vùng như sau:
 - Các số nguyên từ 1 đến 100
 - Các số nguyên nhỏ hơn 1
 - Các số nguyên lớn hơn 100
 - Không phải số
 - Số thập phân
 - Như vậy, việc phân vùng có đúng và đủ hay không là tùy thuộc vào kinh nghiệm của tester.

Kỹ thuật Boundary Value Analysis

- Kỹ thuật phân tích giá trị giới hạn - BVA (Boundary Value Analysis)

– Kỹ thuật BVA sẽ chọn các giá trị nằm tại các điểm giới hạn của phân vùng.



– Áp dụng kỹ thuật BVA cần 4 test case để test trường hợp này: 0,1,100,101

Kỹ thuật EP & BVA

- Xét ví dụ: Một ngân hàng trả lãi cho khách hàng dựa vào số tiền còn lại trong tài khoản. Nếu số tiền từ 0 đến 100\$ thì trả 3% lãi, từ lớn hơn 100 \$ đến nhỏ hơn 1000\$ trả 5% lãi, từ 1000\$ trở lên trả 7% lãi.

– Định nghĩa FP:

Invalid partition	Valid (for 3% interest)	Valid (for 5%)	Valid (for 7%)
-\$0.01	\$0.00	\$100.00	\$100.01

- Kỹ thuật EP: -0.44, 55.00, 777.50, 1200.00

– Kỹ thuật BVA: -0.01, 0.00, 100.00, 100.01, 999.99, 1000.00

Tại sao phải kết hợp BVA và EP

- Mỗi giá trị giới hạn đều nằm trong một phân vùng nào đó. Nếu chỉ sử dụng giá trị giới hạn thì ta cũng có thể test luôn phân vùng đó.
- Tuy nhiên vấn đề đặt ra là nếu như giá trị đó sai thì nghĩa là giá trị giới hạn bị sai hay là cả phân vùng bị sai. Hơn nữa, nếu chỉ sử dụng giá trị giới hạn thì không đem lại sự tin tưởng cho người dùng vì chúng ta chỉ sử dụng những giá trị đặc biệt thay vì sử dụng giá trị thông thường.
- Vì vậy, cần phải kết hợp cả BVA và EP

Ví dụ

Customer Name

2-64 chars.

Account number

6 digits, 1st
non-zero

Loan amount requested

£500 to £9000

Term of loan

Monthly repayment

1 to 30 years

Term:

Minimum £10

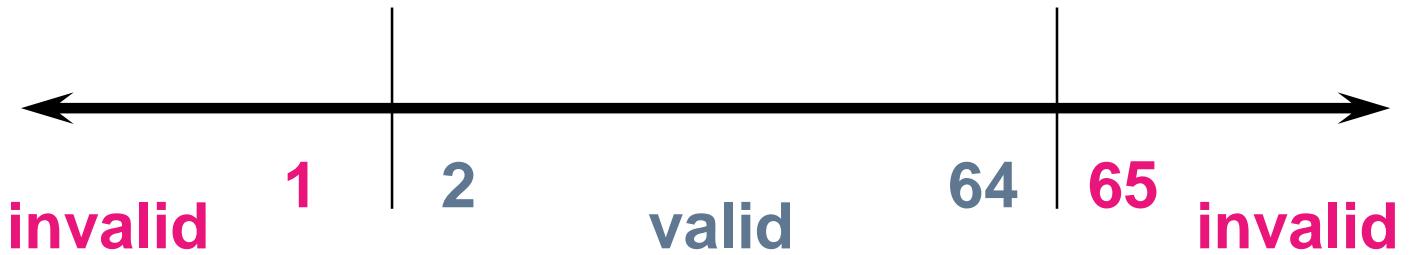
Repayment:

Interest rate:

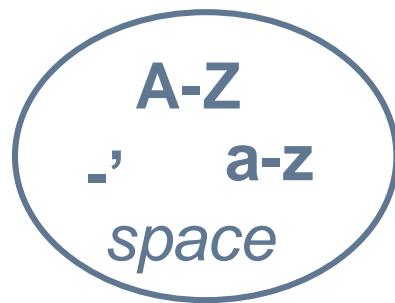
Total paid back:

Customer name

Number of characters:



Valid characters:



Any other

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

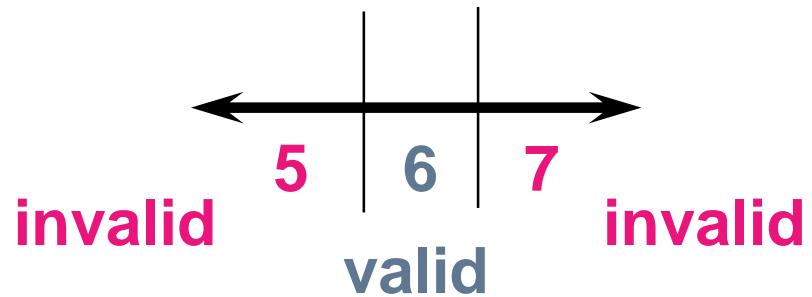
Account number

first character:

valid: non-zero

invalid: zero

number of digits:



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

Condition template

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

Design test cases

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

Phương pháp đoán lỗi (Error Guessing)

- Dựa vào trực giác và kinh nghiệm
- Nhược điểm: không phát hiện hết lỗi
- VD: Lỗi chia cho 0. Nếu module có thực hiện phép chia thì ta phải kiểm thử lỗi này

Test Automation

- Test tự động là gì?
 - Script, tool, framework
 - Thực hiện bằng máy, không phải bằng người
- Tại sao phải dùng test tự động?
 - Nhanh, ít công sức, có thể chạy ban đêm, có thể làm các công việc nặng nề
- Khi nào dùng test tự động?
 - Tính ổn định của phần mềm (regression test)
 - Nâng cấp khả năng phần mềm (load/stress/performance test)
- Làm thế nào để dùng?
 - Chọn công cụ thích hợp với tính chất của phần mềm
 - Chọn phạm vi test, phát triển các test case, cân bằng chi phí
 - Xây dựng và thực thi các test script.

Test Automation

- Kiểm thử tự động Dữ liệu
 - Bộ sinh DL thử
 - Bộ xác minh kết quả
- Kiểm thử tự động cài đặt
 - Bộ kiểm toán mã
 - Mô phỏng ứng xử của các modules phụ
 - Bộ so sánh đầu ra
- Mô phỏng môi trường
- Bộ phân tích dòng DL (quy mô và tần suất dòng DL)

Nhiều công cụ hỗ trợ các loại kiểm thử

- Kiểm thử đơn vị: Achoo, JUnit, Pex/Moles, PyUnit
- Tự động kiểm thử: TestComplete
- Kiểm thử hiệu năng và tải: JMeter
- Kiểm thử giao diện đồ họa (GUI): Abbot, Guitar
- Kiểm thử tổ hợp: AETG, FireEye
- Kiểm thử dựa trên mô hình: Spec Explorer
- Phân tích bao phủ: Corbertura
- Quản lý lỗi (defects): Bugzilla

Một số công cụ test tự động

- Junit: dùng cho code bằng Java
 - Bài tập:
 - Cài đặt lớp hình chữ nhật (Rectangle) với các phép toán tính chu vi, diện tích.
 - Kiểm thử lớp hình chữ nhật với Junit
- QuickTest Pro
 - Bài tập
 - Viết 3 trường hợp kiểm thử để thực hiện vài test đơn giản một trang web nào đó (ví dụ đăng nhập của website đk môn học, xem điểm,...)

Vai trò Tester

- Kiểm lỗi PM
- Kiểm lỗi bản đóng gói
- Kiểm lỗi tài liệu
 - User guide
 - Installation Guide
 - Release Notes
 - Troubleshooting

Công việc Tester

- Chuẩn bị môi trường test
 - Windows XP, 2000, 2003, 7, 8, 10
 - Linux
 - IE, Chrome, FireFox, Safari
 - Test Database, Test data
- Thiết kế Test case
- Thực hiện test các Test case trong từng môi trường khác nhau
- Mô tả Bug và chi tiết các bước để tạo ra bug
- Theo dõi quá trình Fix Bug
- Báo cáo kết quả test

Các web tham khảo

- Testing Tools
 - <http://www.aptest.com/resources.html>
- Testing Course
 - <http://www.aptest.com/courses.html>
 - <http://www.aptest.com/testtypes.html>
 - <http://www.appperfect.com/products/windowstester.html>
 - <http://www.openseminar.org/se/modules/7/index/screen.do>