Môn học

KIỂM THỬ PHẦN MỀM

Chương II

VAI TRÒ CỦA TESTING TRONG VÒNG ĐỜI PHÁT TRIỂN PM

Nội dung

1. Các mô hình phát triển phần mềm và vị trí của kiểm thử
2. Các mức kiểm thử
3. Các kiểu kiểm thử

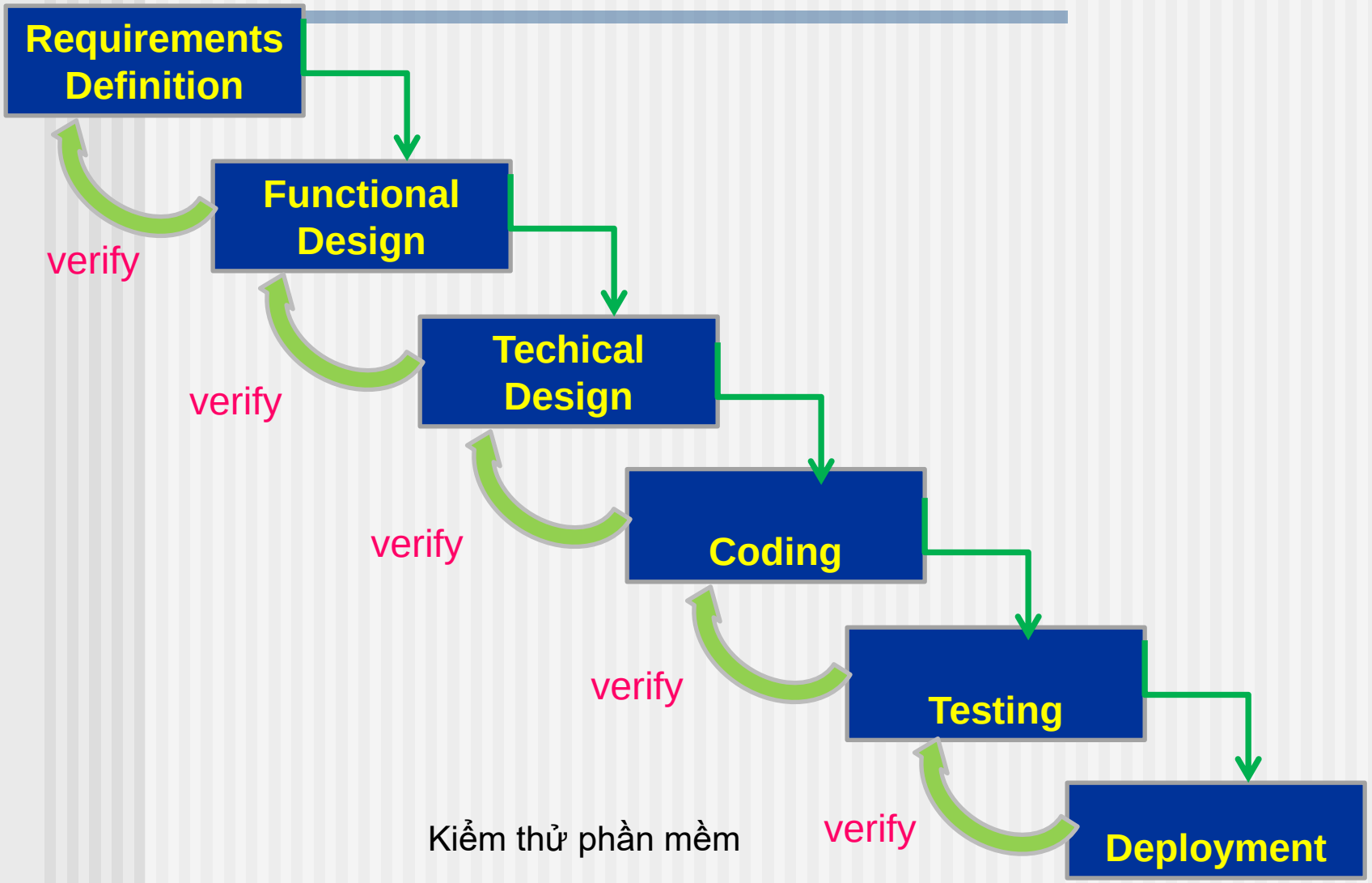
II.1 Các mô hình phát triển PM và vị trí của kiểm thử

- Mỗi mô hình phát triển PM
  - Phân bố hoạt động kiểm thử khác nhau → có thể ảnh hưởng đến việc tìm bugs
  - Một phần kiểm thử sẽ tập trung vào hoạt động verification (xác minh) và phần khác tập trung vào validation (thẩm định)

Verification và Validation

- Verification(xác minh): là quá trình kiểm tra PM có đúng đặc tả hay không.
- Validation(thẩm định): là quá trình kiểm tra PM có đáp ứng được yêu cầu người dùng không.

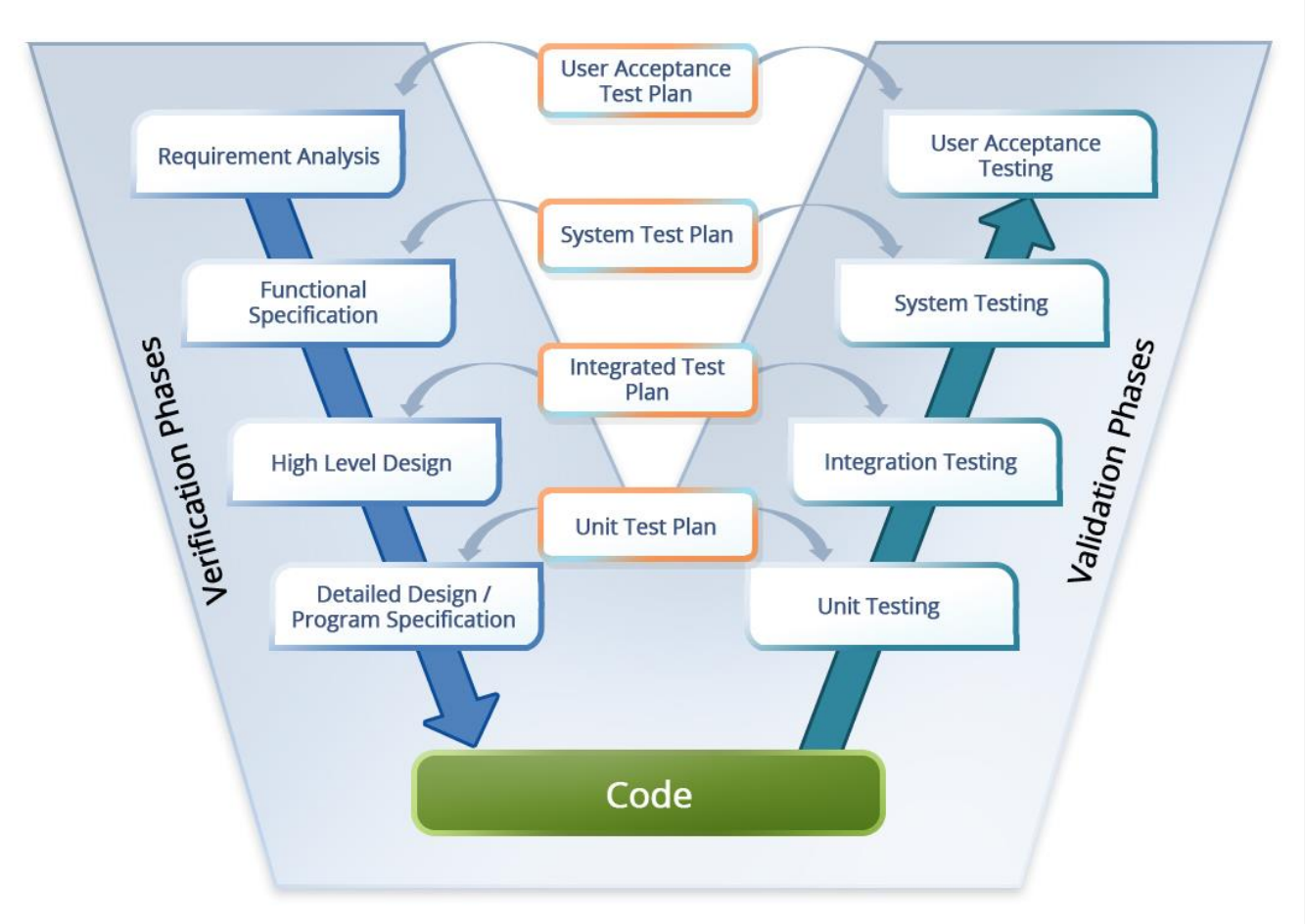
Waterfall Model



Testing trong Waterfall model

- Testing vốn có ở mỗi pha
- Testing thực hiện liên tục từ pha thiết kế, thực thi để thẩm định lại các pha trước đó
- Pha testing bắt đầu sau pha coding

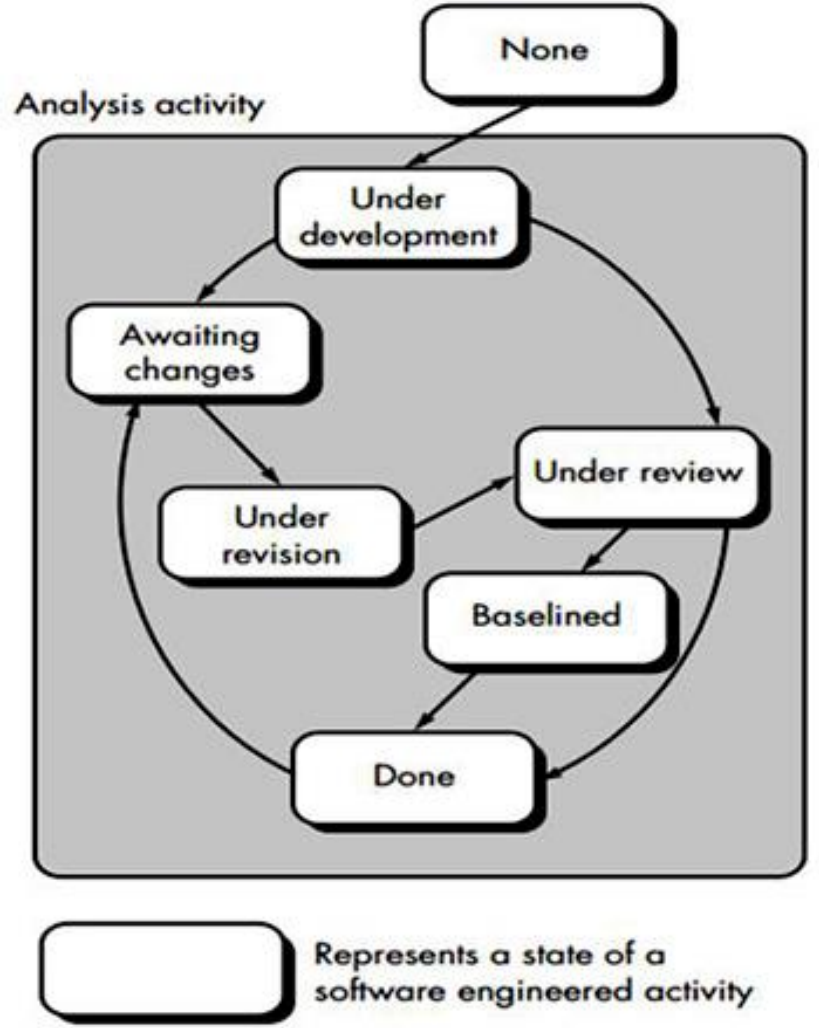
V-Model


User Acceptance Test Plan
Requirement Analysis
User Acceptance Testing
System Test Plan
Functional Specification
System Testing
Integrated Test Plan
High Level Design
Integration Testing
Unit Test Plan
Detailed Design / Program Specification
Unit Testing
Code
Verification Phases
Validation Phases

V-Model

- Các hoạt động kiểm thử được bắt đầu sớm và thực hiện song song với các hoạt động phát triển
- Sản phẩm ở mỗi pha là cơ sở cho kiểm thử ở các mức khác nhau

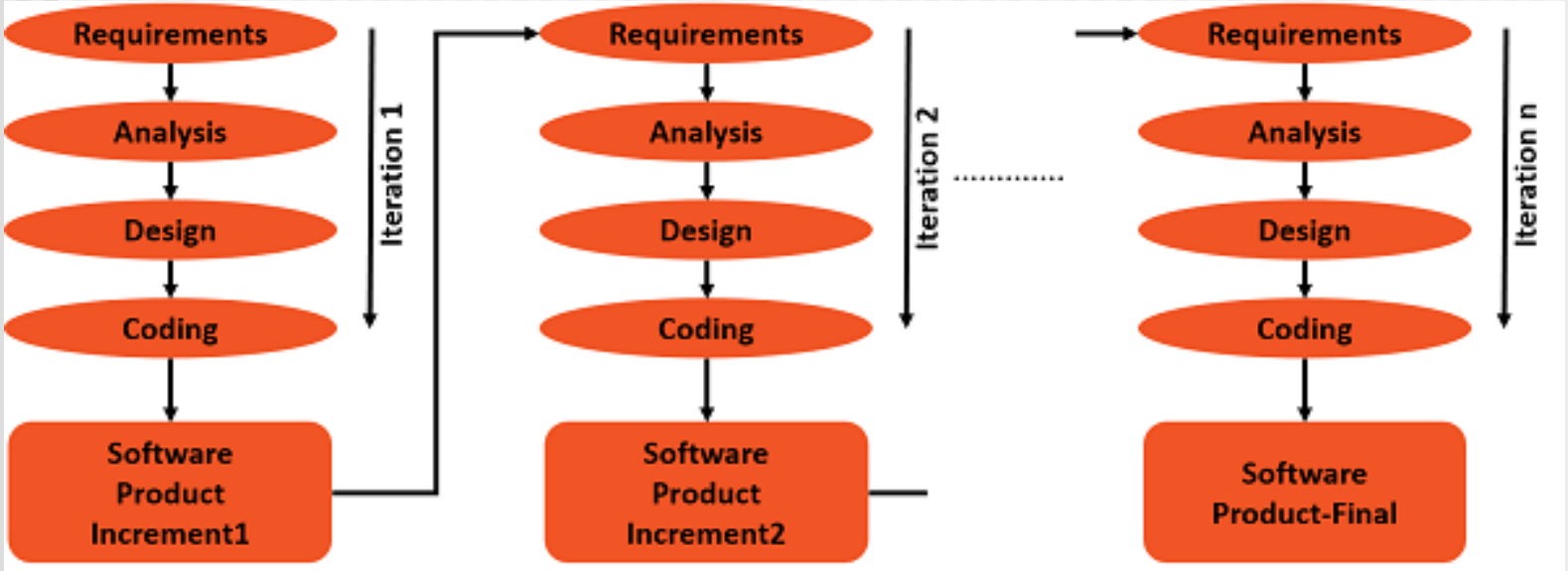
Concurrent model



Testing trong Concurrent Model

- Trong mô hình này, lập kế hoạch, thiết kế và phát triển PM xảy ra đồng thời cùng 1 lúc
- Toàn bộ dự án không được vạch kế hoạch rõ ràng → rất khó test
- Testing theo kiểu khám phá(ad-hoc)
- Bugs có thể sẽ bị bỏ sót trong quá trình test

Mô hình lặp (Iterative Model)



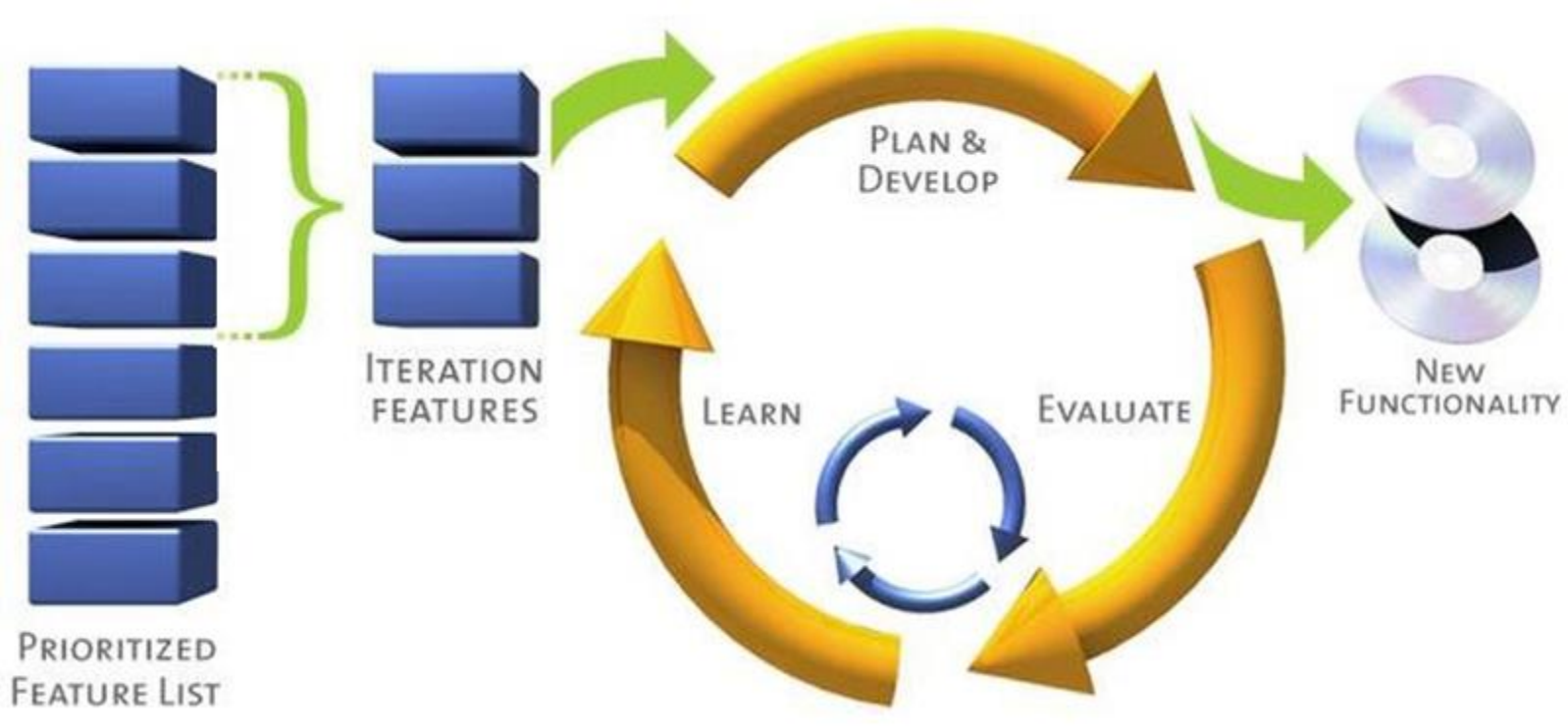
Mô hình lặp (Iterative Model)

- Mỗi bước lặp tiếp theo sẽ thực hiện
  - Test chức năng mới, đồng thời thực hiện regression testing
  - Kiểm thử tích hợp phần chức năng mới và cũ
- PM được release sớm, tức là validation được thực hiện sớm ở mỗi vòng lặp → nhận được feedback sớm từ người dùng

Mô hình lặp (Iterative Model)

- Các ví dụ về mô hình lặp:
  - Prototyping
  - Rapid Application Development (RAD)
  - Rational Unified Process (RUP)
  - Agile Model

Agile Model


PRIORITIZED
FEATURE LIST
ITERATION
FEATURES
PLAN &
DEVELOP
LEARN
EVALUATE
NEW
FUNCTIONALITY

Testing trong Agile Model

- Phát triển phần mềm theo kiểu tập trung
- Testing theo kiểu khám phá(ad-hoc), nhưng tập trung
- Dự án được phát triển linh động, nhiều thay đổi, nhưng tất cả các thay đổi đều được thảo luận và được ghi chú lại

II.2 Các mức kiểm thử

- Unit testing
- Integration testing
- System testing
- Acceptance testing

1/ Kiểm thử đơn vị (Unit Testing)

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

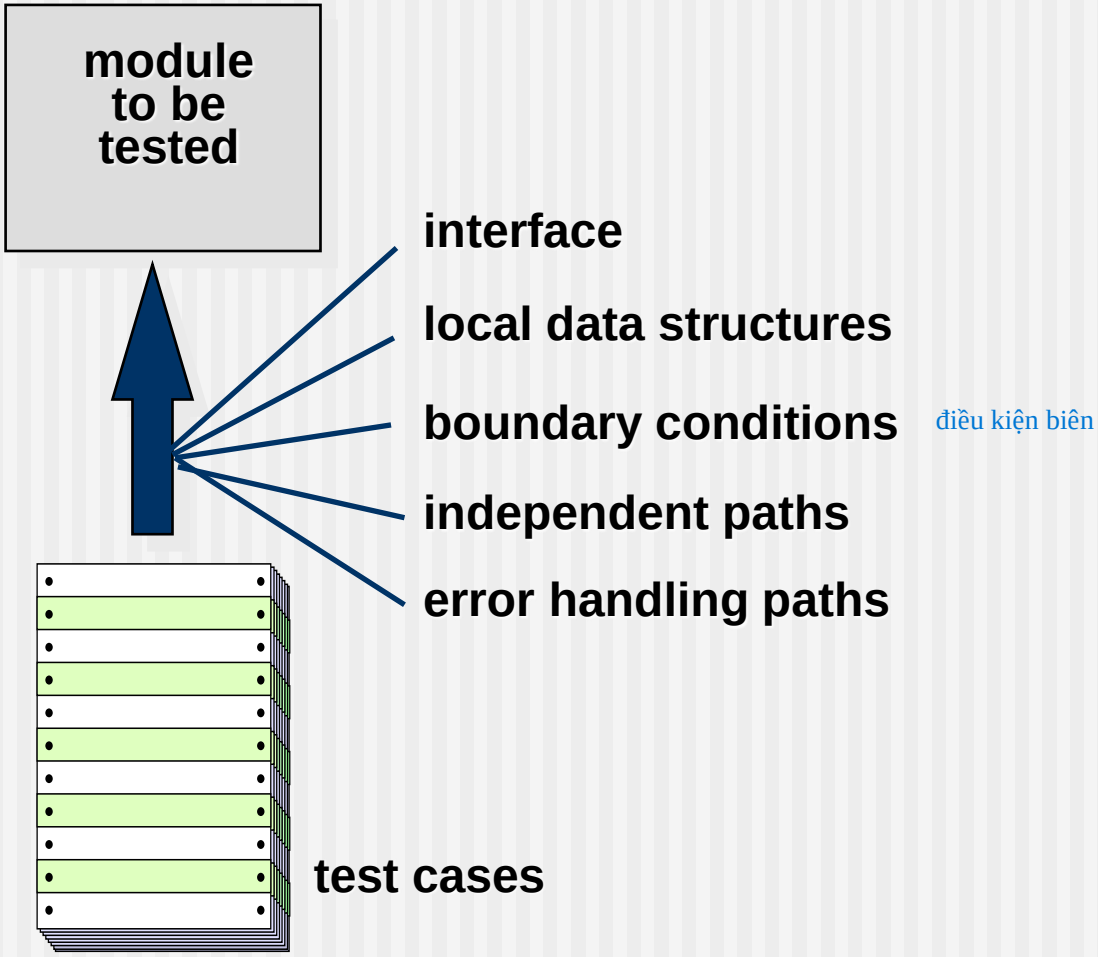
(cần nắm cấu trúc của code trong chương trình)

Chiến lược kiểm thử đơn vị

kiểm thử hộp trắng

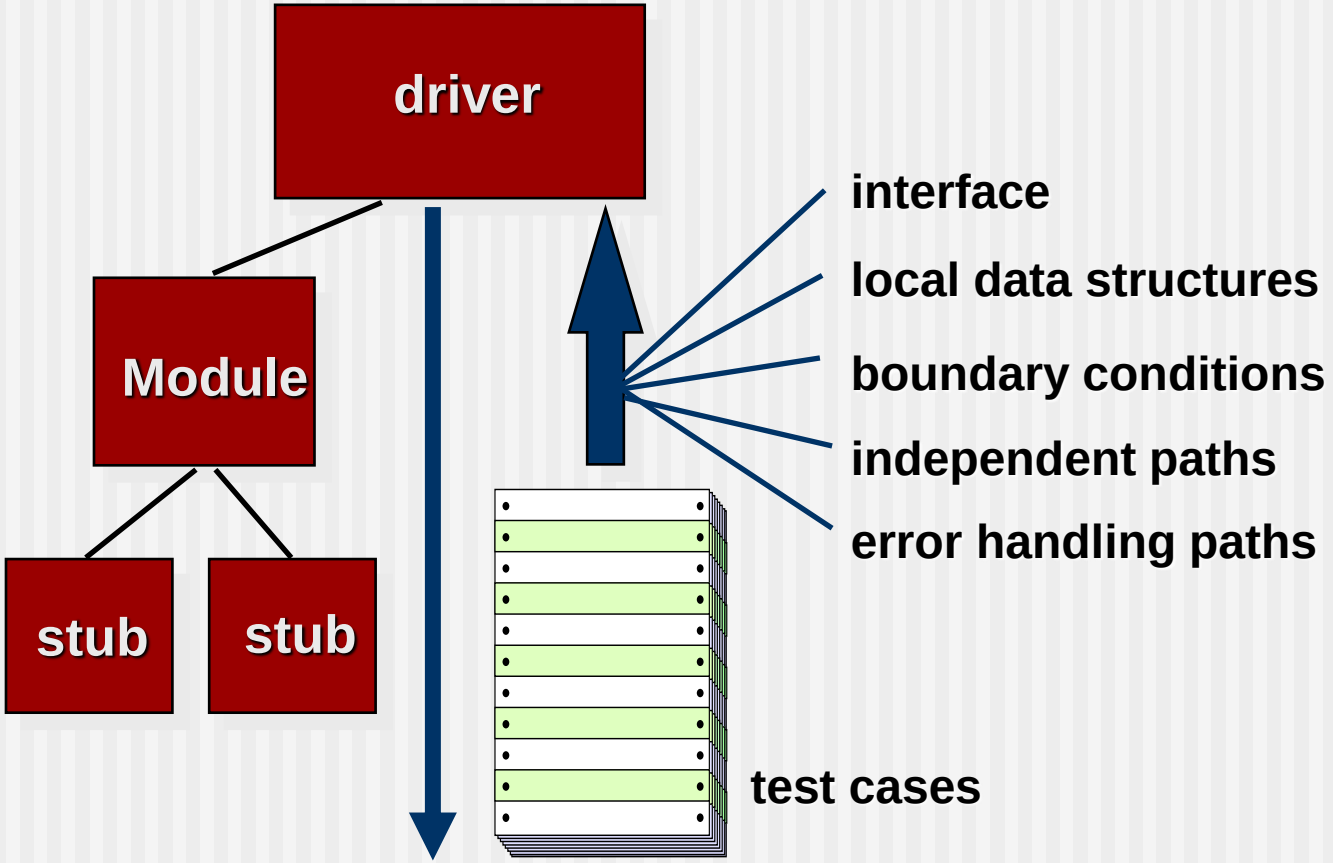
- Xác định các kĩ thuật kiểm thử(white-box)
- Đưa ra các tiêu chí để hoàn thành test
- Xác định mức độ độc lập khi thiết kế test
- Lập tài liệu về qui trình test và các hoạt động test(inputs và outputs)
- Những hoạt động lặp đi lặp lại sau mỗi lần fix lỗi hoặc thay đổi

Nội dung kiểm thử đơn vị



Môi trường kiểm thử đơn vị

module nằm trên (driver)



module nằm dưới (stub)

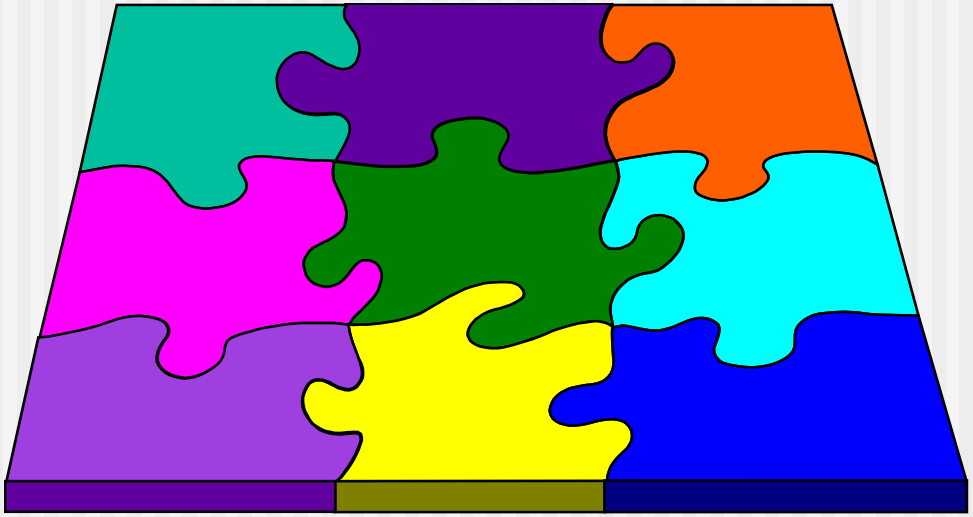
2/ Kiểm thử tích hợp (Integration Testing)

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

Chiến lược kiểm thử tích hợp

- Có 2 hướng tiếp cận cơ bản:
  - Big-bang có bao nhiêu module, func thì tích hợp hết & kiểm thử luôn 1 lần
  - Incremental (top-down, bottom-up)

tích hợp dần dần: tích hợp 2,3 module kiểm thử trước, sau đó bổ sung thêm



Chiến lược kiểm thử tích hợp

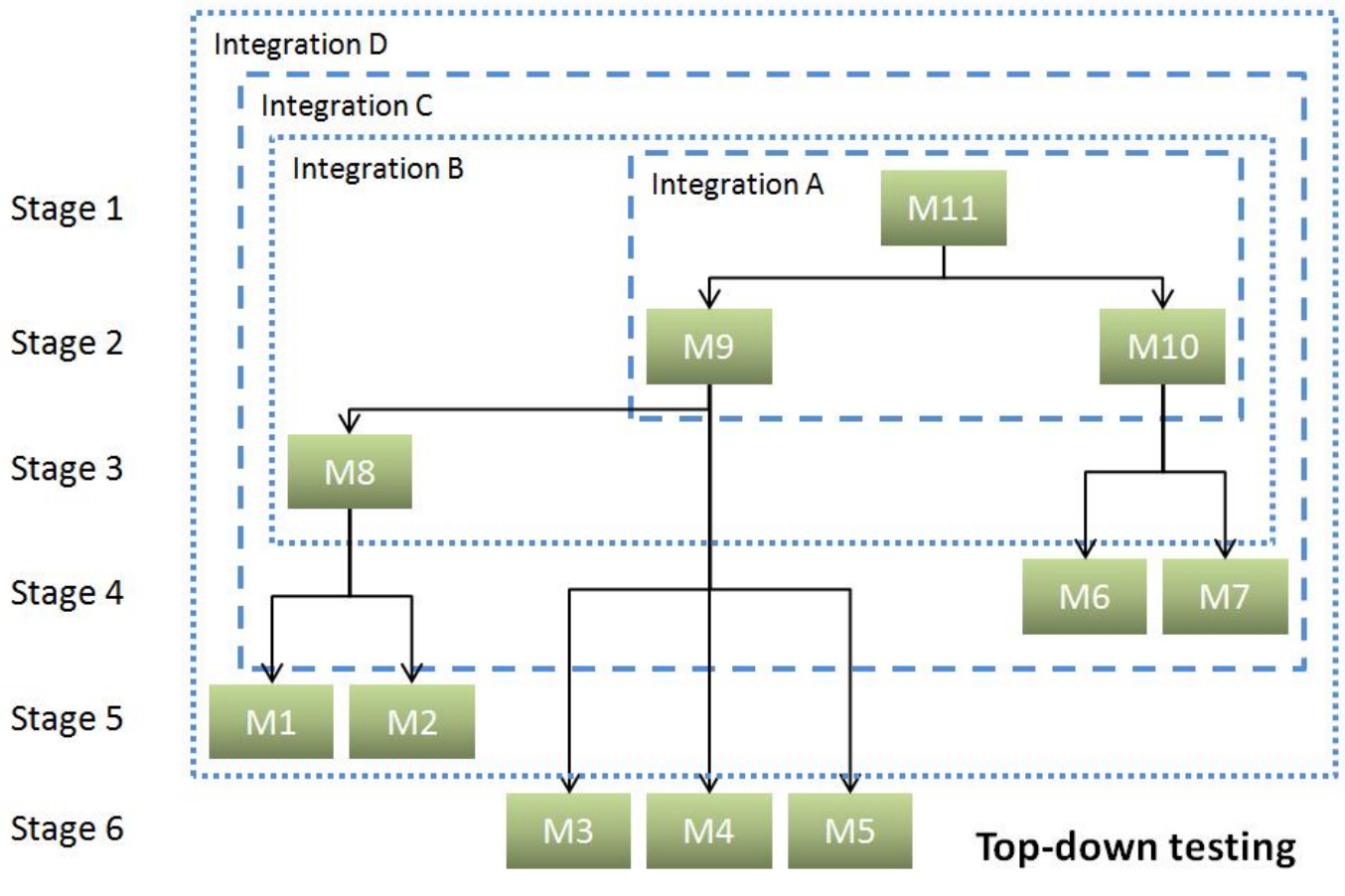
- Big-bang:
  - Tại sao chúng ta lại kết hợp tất cả các thành phần và test cùng 1 lúc? → tiết kiệm thời gian
  - Trong thực tế: độ phức tạp tăng lên rất nhiều
    - Mất nhiều thời gian để tìm lỗi và sửa lỗi
    - Test lại sau khi sửa lỗi sẽ phức tạp hơn nhiều

Chiến lược kiểm thử tích hợp

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

có 11 module, những module ở trên gọi những module ở dưới. ta sẽ tích hợp M11 với M9,M10 trước để kiểm thử=> test từ module lớn nhất trước
M11 gọi M9, M9 phải gọi M8,... mà những module con của M9 chưa kiểm thử, vậy làm sao để M9 chạy đúng?
giả sử có hàm main, và hàm sort thuộc hàm main, vậy ta test hàm main trước, và sort chưa test
=> xây dựng hàm thay thế cho hàm con => sử dụng stub: module mô phỏng (hàm thay thế cho hàm sort, viết hết sức đơn giản, và nó phải chạy đúng, 0 lỗi) cho các module con bên dưới
=> KL: xây dựng stub cho các M8, M3,M4,M5 để test được M9.

Tích hợp Top-down



Tích hợp top-down

Ưu điểm

module chính (lớn) => nhận input đầu vào sớm hơn các input con => dễ test

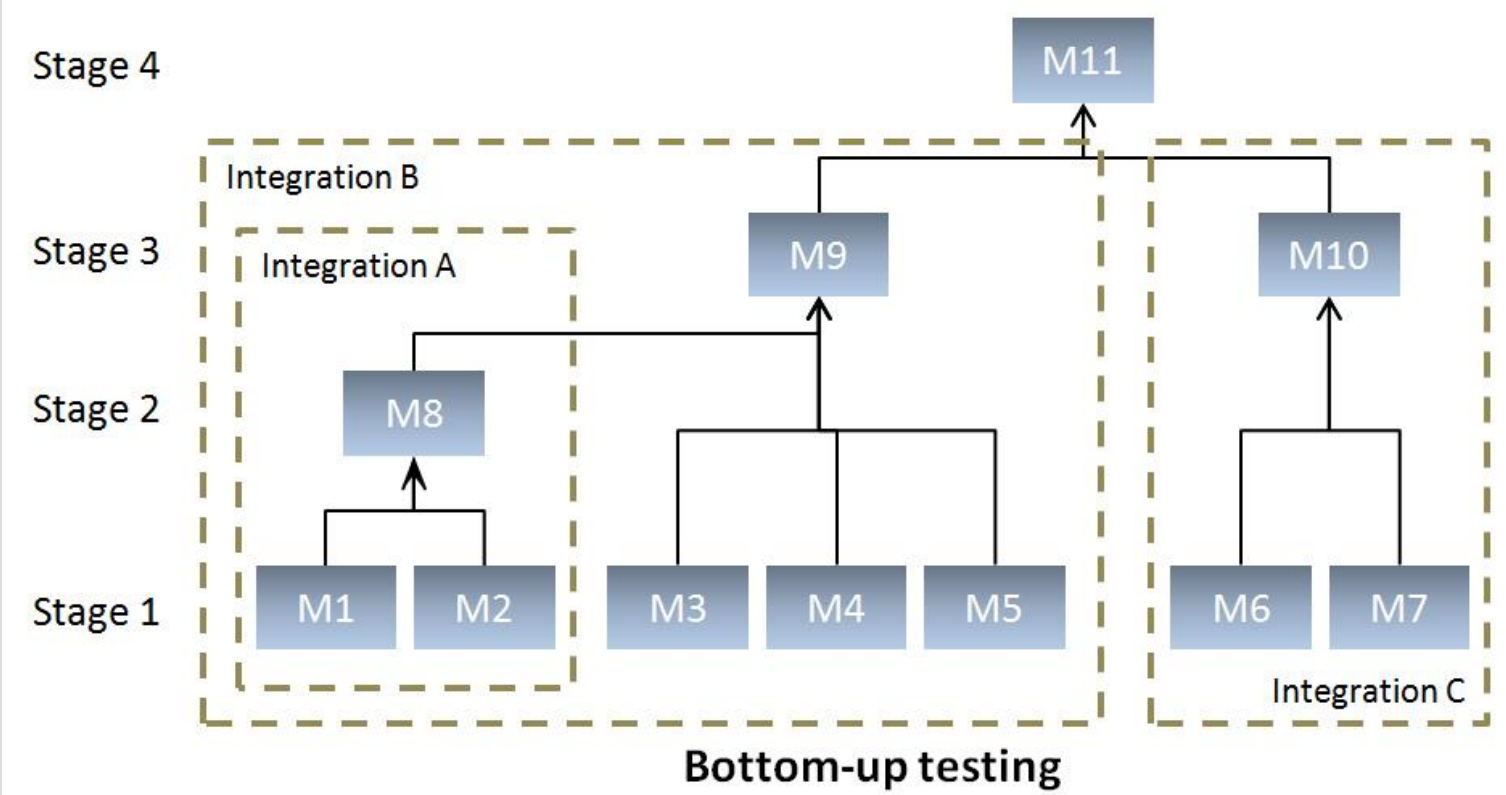
- Cấu trúc điều khiển quan trọng được test trước
- Hàm I/O được gọi sớm → viết test dễ
- Mô phỏng chức năng chính của PM sớm → nổi bật vấn đề liên quan đến yêu cầu

Khuyết điểm

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

Tích hợp những module nhỏ lại với nhau trước, sau đó tích hợp từ từ những module lớn phía trên.
giả sử vẫn làm sort , hàm main => test hàm sort trước, sau đó test hàm main
mỗi lần viết hàm con ta phải kiểm thử, chẳng hạn như sort, tuy nhiên chưa test/chưa viết hàm main (vì thường ta viết hàm con trước khi viết main)
=> xây dựng driver: module/hàm mô phỏng, đơn giản, 0 lỗi, để gọi module đang test, và ta phải truyền input (tham số)
viết driver để gọi hàm sort, và truyền dữ liệu các tham số để hàm sort có thể chạy được.

Tích hợp Bottom-up



Tích hợp Bottom-up

Ưu điểm

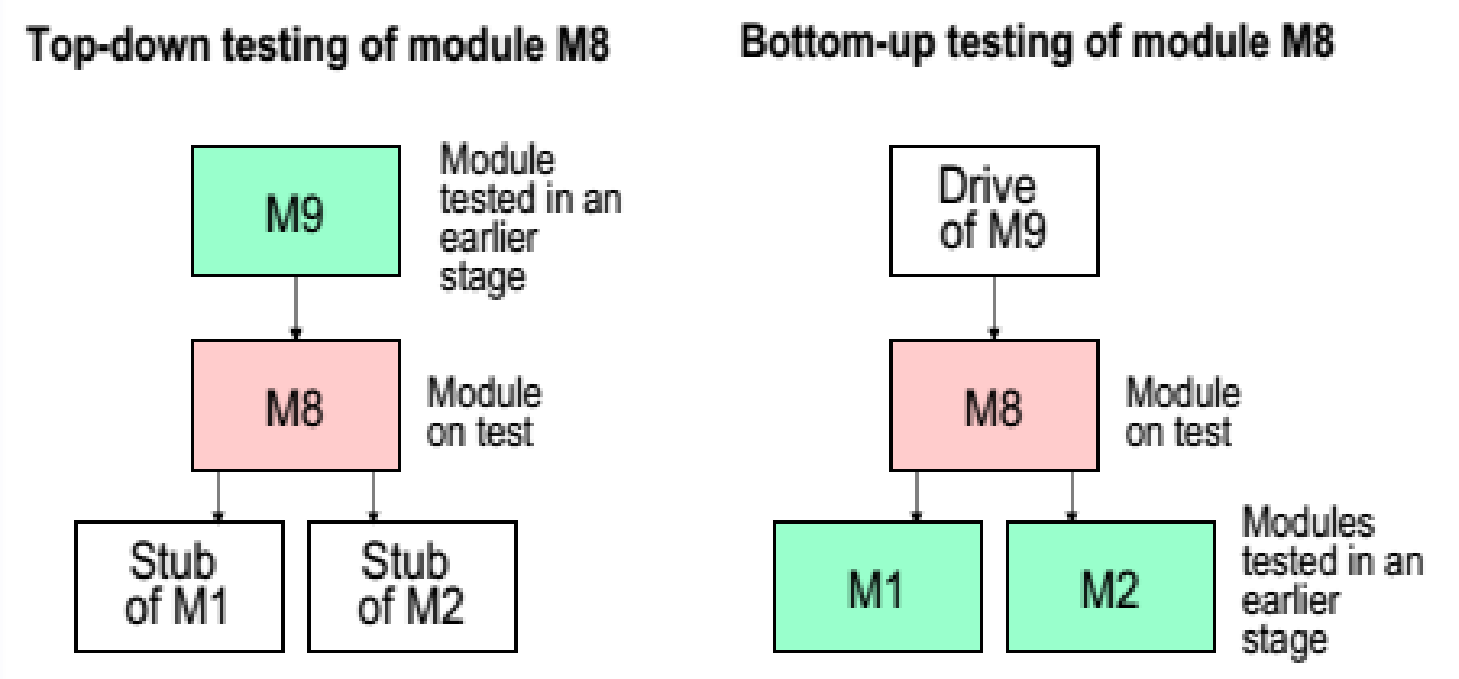
- Các mức thấp nhất được test đầu tiên
- Điều kiện test được tạo dễ dàng
- Dễ quan sát các kết quả test chi tiết

Khuyết điểm

phụ thuộc nhiều vào

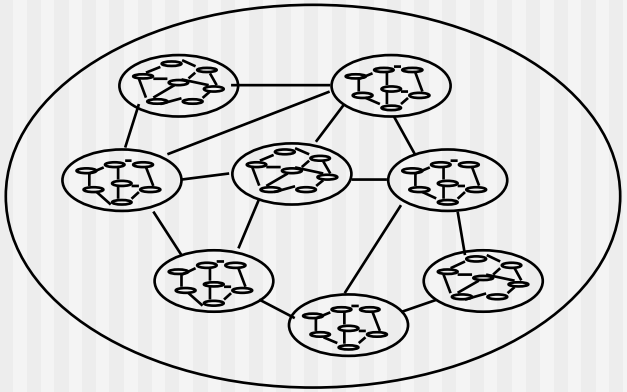
- Phải tạo các driver
- Chương trình tổng thể được quan sát trễ

Stubs and Drivers



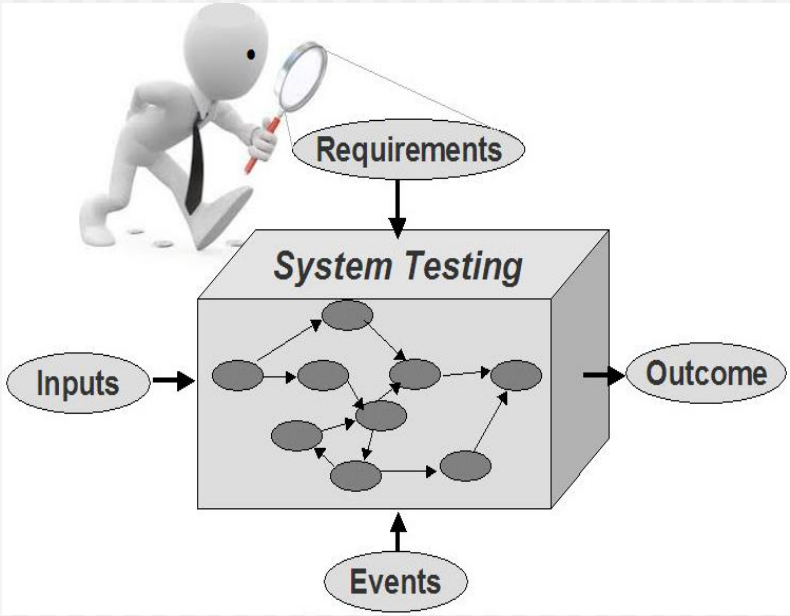
3/ System testing

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



Functional System Testing

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



phức tạp hơn so với kiểm thử yêu cầu chức năng (do phải kiểm thử nhiều thành phần hơn)

Non-functional System Testing

- Usability tính khả dụng
- Security
- Documentation
- Storage
- Volume kích thước dữ liệu
- Configuration / installation
- Reliability
- Recovery khả năng phục hồi
- Performance, load, stress hiệu năng

Usability Testing

tính khả dụng => dễ sử dụng

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

tính dễ sử dụng phụ thuộc vào mỗi người dùng
=> Đặt ra yêu cầu Để đo lường tính dễ sử dụng: chẳng hạn như phần mềm này sau khi hoàn thành, sẽ được training cho người dùng sau khoảng thời gian 1 ngày, sau khoảng thời gian này, mn đều sử dụng thành thục phần mềm này.
nếu tồn tại một số người vẫn khó dùng phần mềm => tính khả dụng không thỏa mãn

Security Testing

- Kiểm tra cơ chế bảo mật của hệ thống có hiệu quả không?
- Tester sẽ đóng vai trò là hacker

kiểm tra xem hệ thống này có đảm bảo an toàn không

bất kỳ hệ thống nào cũng có thể bị hack, chẳng qua có đủ thời gian, nguồn lực chi phí hay không? không ai dám cam đoan hệ thống mình an toàn.

- Bài toán thiết kế hệ thống an ninh là:

Các tiêu chí để hệ thống được coi là an ninh tốt.

Hacker bỏ nhiều công sức để hack. nhưng họ nhận lại những thứ không đáng với công sức họ bỏ ra.

  - Chi phí công cụ bảo vệ nhỏ hơn lợi ích bảo vệ khỏi đột nhập
  - Chi phí đột nhập lớn hơn lợi ích thu được từ đột nhập

Security Testing

- Ví dụ: cookie
  - Passwords, encryption
  - Mức độ truy cập thông tin, quyền
- Một số kĩ thuật test lỗ hổng của ứng dụng Web:
  - SQL Injection tester có thể sd test lỗi liên quan đến SQL, đưa vào 1 số câu lệnh SQL nếu có lỗi => trả về các tables (chẳng hạn như thông tin người dùng, admin...)
  - Cross-site Scripting(XSS) đưa vào các đoạn script vào website chẳng hạn có những chỗ cho người dùng nhập vào vd comment, nhưng ta lại đưa vào đoạn script vào, nếu có lỗi sẽ làm hỏng hệ thống.
  - …

Documentation Testing

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

tài liệu yc, đặc tả, bảo trì, hdan người dùng, kiểm thử...

các yêu cầu đã được KH sửa đổi, nhưng tài liệu hướng dẫn người dùng chưa sửa đổi => không khớp với thực tế

Performance Testing

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

trang courses cho tối đa
1k users truy cập đồng thời
=> nếu người 1001 vào thì không được
=>1000 chính là điểm thắt cổ chai.
(max users quantity)

phân biệt load test và stress test

Stress/Load Testing – Multi User

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

điểm thắt cổ chai là 1000 user truy cập cùng lúc
=> 10 user vào trước, sau đó cứ 10s tăng thêm 10user truy cập vào hệ thống việc tăng chỉ tăng đến mức 1000 người.

pm yêu cầu RAM 8GB mới chạy được => có thể kiểm thử ở vùng nhớ giảm 50% (4GB) lại để xem hệ thống hoạt động như thế nào.
pm nhập điểm cho GV => dùng VPN => ngắt VPN xem có thể truy cập vào hệ thống không? (có thể tắt được cửa sổ phần mềm không?)

Configuration/Installation Testing

- Configuration tests
  - Môi trường phần cứng, phần mềm khác nhau
  - Nâng cấp 1 phần của hệ thống có thể dẫn đến xung đột với phần khác
- Installation Tests thử cài đặt qua nhiều hình thức khác nhau
  - PM có thể cài đặt qua CD, networks,…
  - Thời gian cài đặt
  - Uninstall

yêu cầu chức năng: pm này phải có chức năng print, soạn thảo vb...
yêu cầu phi chức năng: tính khả dụng, bảo mật... => đo lường khó hơn so với yêu cầu chức năng: xác suất xảy ra lỗi là bao nhiêu, sau bao lâu thì ta phục hồi được hệ thống sau lỗi.

Reliability Testing

khác nhau ở những thời điểm khác nhau
(vd độ tin cậy cao ở trong khoảng thời gian nào)
XĐ độ tin cậy nằm trong khoảng thời gian nào

- Reliability (độ tin cậy)
  - Định nghĩa: là xác suất thao tác không thất bại của hệ thống trong khoảng thời gian xác định dưới điều kiện xác định.

thời gian tb giữa những lần thất bại kế tiếp

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

Recovery Testing

phải rơi vào trạng thái lỗi

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

kiểm thử chấp nhận người dùng

4/ User acceptance testing

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

User

customer có thể khác với user
user: những người sd trực tiếp phần mềm
customer: những người đi đặt hàng phần mềm, có thể sử dụng pm hoặc không

Tại sao người dùng nên tham gia test?

vì người dùng là người sử dụng đầu cuối, nên họ sẽ góp ý chi tiết nhất về giao diện, trải nghiệm...
đưa ra những vấn đề mà khi chuyển sang tự động hóa mới xảy ra (những dữ liệu mà người dùng nhập vào, dev &tester không nghĩ đến)
góc nhìn của người dùng sẽ khác so với góc nhìn của ng phát triển pm => phát hiện ra các ý kiến chủ quan.
ng dùng sẽ được trải nghiệm trước pm, sau này sử dụng dễ dàng hơn, và có thể góp ý bổ sung.

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

Kiểm thử Alpha và Beta

(users)

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

Kiểm thử Alpha

vẫn là bên user test, nhưng được mời bởi bên phát triển phần mềm về, mọi thứ hệ thống hạ tầng đều do ptrien pm chuẩn bị

- Do bên phát triển PM tiến hành
- Được tiến hành trong môi trường được điều khiển

do bên phát triển pm chuẩn bị trước

- Dữ liệu thường là mô phỏng

do bên phát triển pm chuẩn bị trước

Kiểm thử Beta

có thể tim ra lỗi mà khi Kiểm thử alpha không tìm ra được

- Do bên khách hàng tiến hành

thực hiện trên chính MT của người dùng, có cài đặt các phần mềm sẵn của họ

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

II.3 Các kiểu kiểm thử

- Mỗi kiểu kiểm thử tập trung vào một mục tiêu kiểm thử cụ thể tìm ra những loại lỗi nhất định.
- Các kiểu kiểm thử (cái nào tốt nhất/hiệu quả nhất? không có => áp dụng kết hợp 2,3 kiểu)
  - Kiểm thử chức năng
  - Kiểm thử phi chức năng
  - Kiểm thử cấu trúc PM
  - Kiểm thử liên quan đến thay đổi

Kiểm thử chức năng (Functional Testing)

- Là loại kiểm thử dựa trên:
  - Các chức năng được mô tả trong đặc tả yêu cầu
  - Quy trình nghiệp vụ của PM
- Các kỹ thuật được sử dụng để thực hiện loại kiểm thử này:
  - Specification-based (kiểm thử hộp đen)
  - Experienced-based (kinh nghiệm của kiểm thử viên)
- Có thể thực hiện ở tất cả các mức test

Kiểm thử phi chức năng (Non-Functional testing)

- Là loại kiểm thử các đặc trưng chất lượng của PM dựa trên các chuẩn
  - ISO/IEC 9126
  - McCall
- Bao gồm:
  - performance testing, load testing, stress testing, usability testing, maintainability testing, reliability testing,…

kiểm thử xem pm có khả năng bảo trì được không?
bảo trì = quá trình tìm lỗi, sửa lỗi, nâng cấp chức năng mới, mở rộng

- Có thể áp dụng ở tất cả các mức test

Kiểm thử cấu trúc PM

kiểm thử hộp trắng

- Hay còn được gọi là white-box hoặc glass-box, kiểm thử dựa trên cấu trúc bên trong của PM dựa vào sources code => phải do dev thực hiện, vì họ hiểu biết rõ nhất
- Kiểu kiểm thử này đo lường độ bao phủ các thành phần trong cấu trúc PM
- Chủ yếu được áp dụng ở mức kiểm thử đơn vị và tích hợp

Kiểm thử liên quan đến thay đổi

- Mục đích chính là kiểm thử sự thay đổi
- Gồm 2 loại:
  - Re-testing (Confirmation testing) kiểm thử lại/kiểm thử xác nhận
  - Regression testing kiểm thử hồi quy

Re-testing

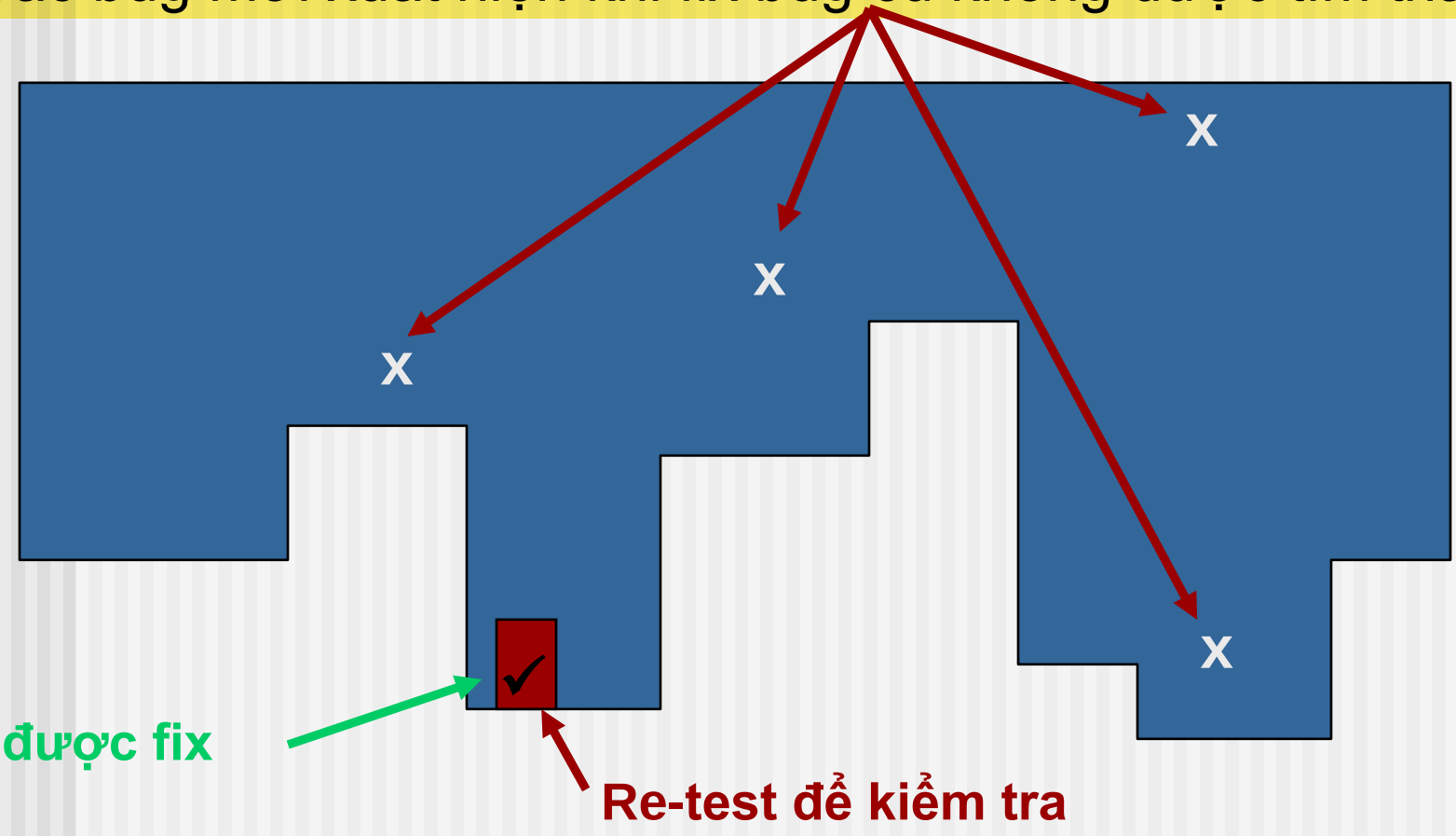
kiểm thử lại

test case mà làm lộ ra lỗi
=> test case failed
ngược lại => test case passed

- Chạy lại các test cases không thành công, kt lại xem bug ban đầu còn không?
- Phiên bản mới chỉ bao gồm các khiếm khuyết cũ được fix
- Chạy lại chính xác test

Re-testing

Các bug mới xuất hiện khi fix bug cũ không được tìm thấy



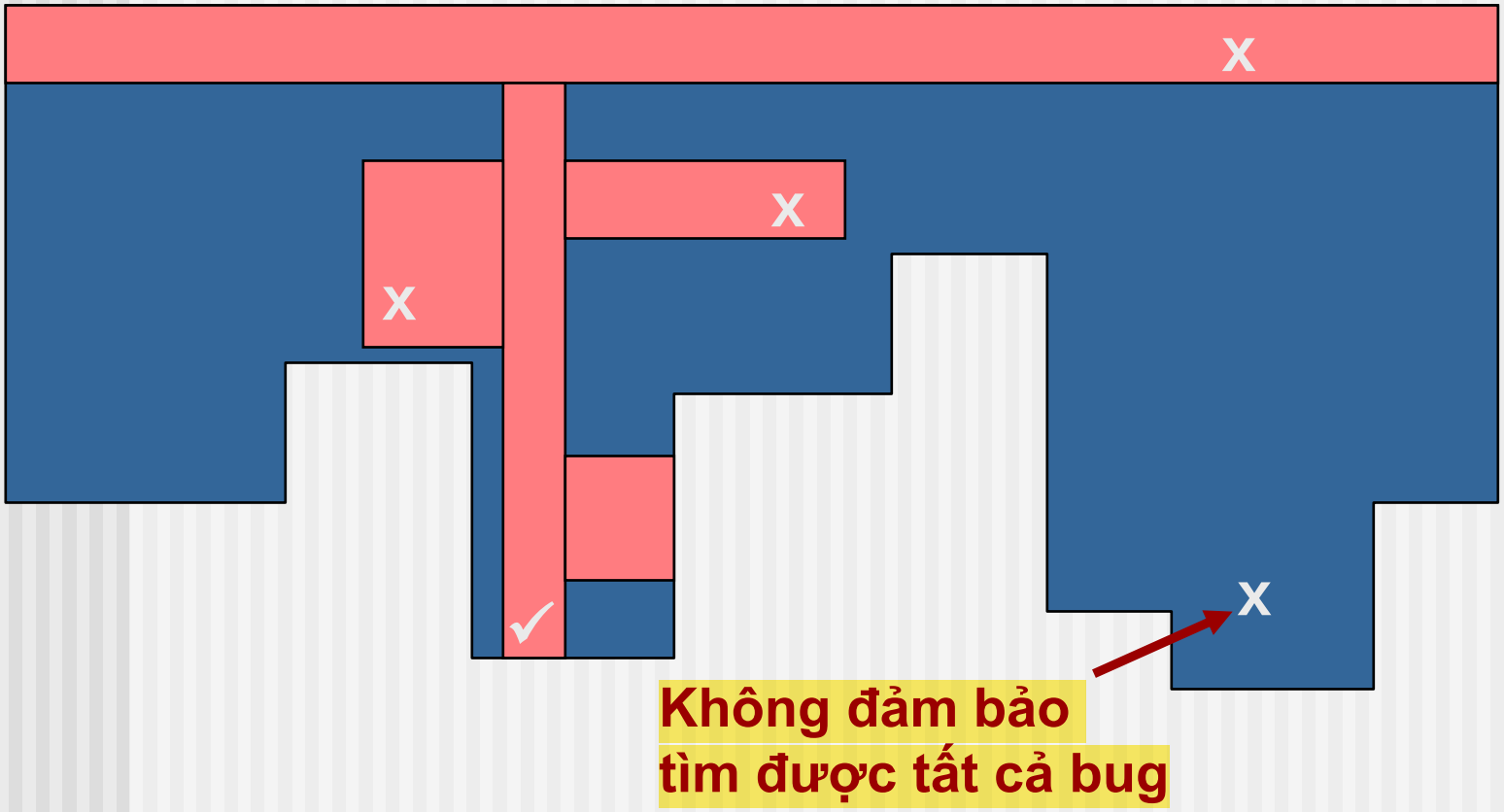
Regression testing

kiểm thử hồi quy=> mất tgian

- Chạy lại tất cả các test cases đã được thực thi trước đó kể cả TC fail, TC pass
  bug cũ đã fix chưa, và có phát sinh bug mới không?
- Phiên bản mới có thể bao gồm các khiếm khuyết cũ được fix, và các khiếm khuyết mới
- Có thể sử dụng các công cụ test tự động

Regression testing

không đảm bảo tìm ra được mọi lỗi



Regression testing

- Việc bảo trì bộ test hồi quy cần được thực hiện
  - Khi chức năng mới được thêm vào PM → cần bổ sung thêm các test hồi quy
  - Khi chức năng cũ bị thay đổi hoặc bỏ bớt → cần thay đổi hoặc bỏ bớt các test hồi quy

áp dụng cho các mô hình phát triển phần mềm => mất thời gian, cầu kỳ, tốn chi phí...

mô hình tốn chi phí kiểm thử regression nhiều nhất trong 3 mô hình: waterfall, V-model, lặp?
-> Mô hình lặp: chia ra làm nhiều vòng, mỗi vòng bổ sung thêm cn mới -> test lại mọi chức năng cũ và mới, để xem khi bổ sung chức năng mới có sinh ra bug nữa hay không -> chi phí kiểm thử sẽ tăng lên.