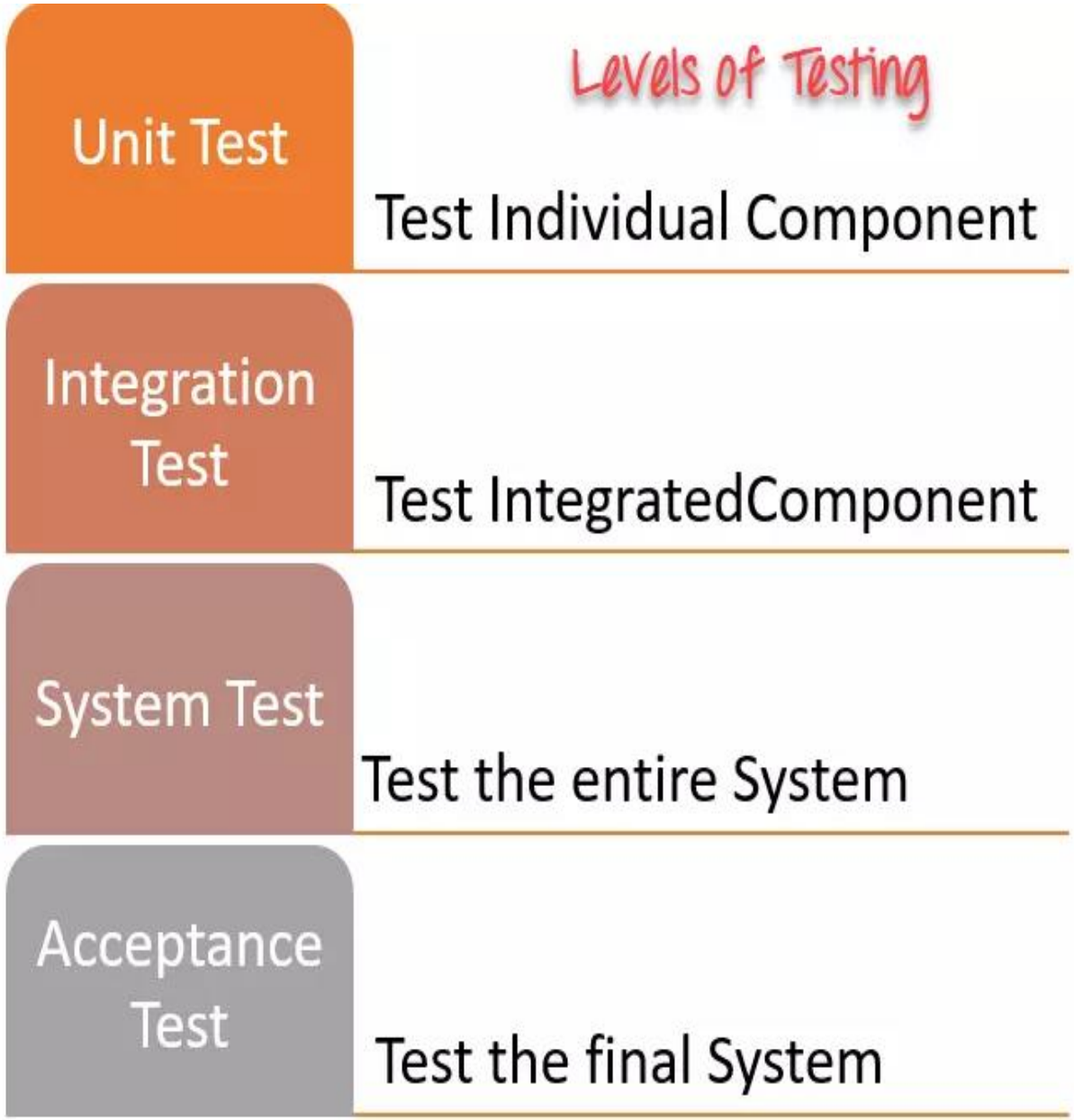
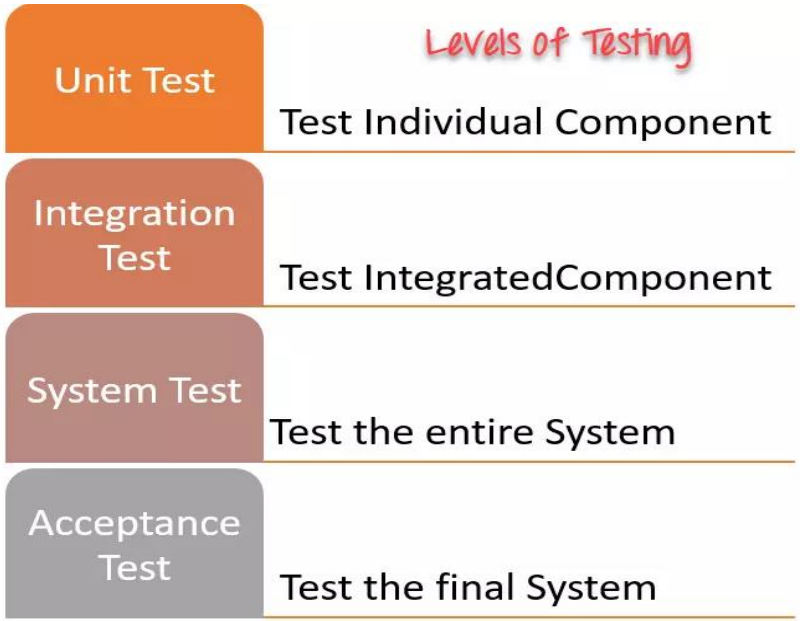
Bài 2. Các mức độ trong kiểm thử phần mềm

NỘI DUNG

- Các mức độ kiểm thử
  - Component Testing - Kiểm thử thành phần
  - Integration Testing - Kiểm thử tích hợp
  - System Testing - Kiểm thử hệ thống
  - Acceptance Testing - Kiểm thử chấp nhận
- Một số loại kiểm thử quan trọng
  - Smoke Testing - Kiểm thử khói
  - Sanity Testing - Kiểm thử độ tỉnh táo
  - Regression Testing - Kiểm thử hồi quy

Các mức độ kiểm thử





KIỂM THỬ THÀNH PHẦN

COMPONENT TESTING

- Component Testing - Kiểm thử thành phần
  - Hay còn gọi là Unit Testing là một mức kiểm thử đầu tiên trong kiểm thử phần mềm với mục đích để xác nhận từng thành phần của phần mềm được phát triển đúng như được thiết kế.
  - Unit testing là mức test nhỏ nhất trong bất kỳ phần mềm nào. Các hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method) đều có thể được xem là Unit.

COMPONENT TESTING

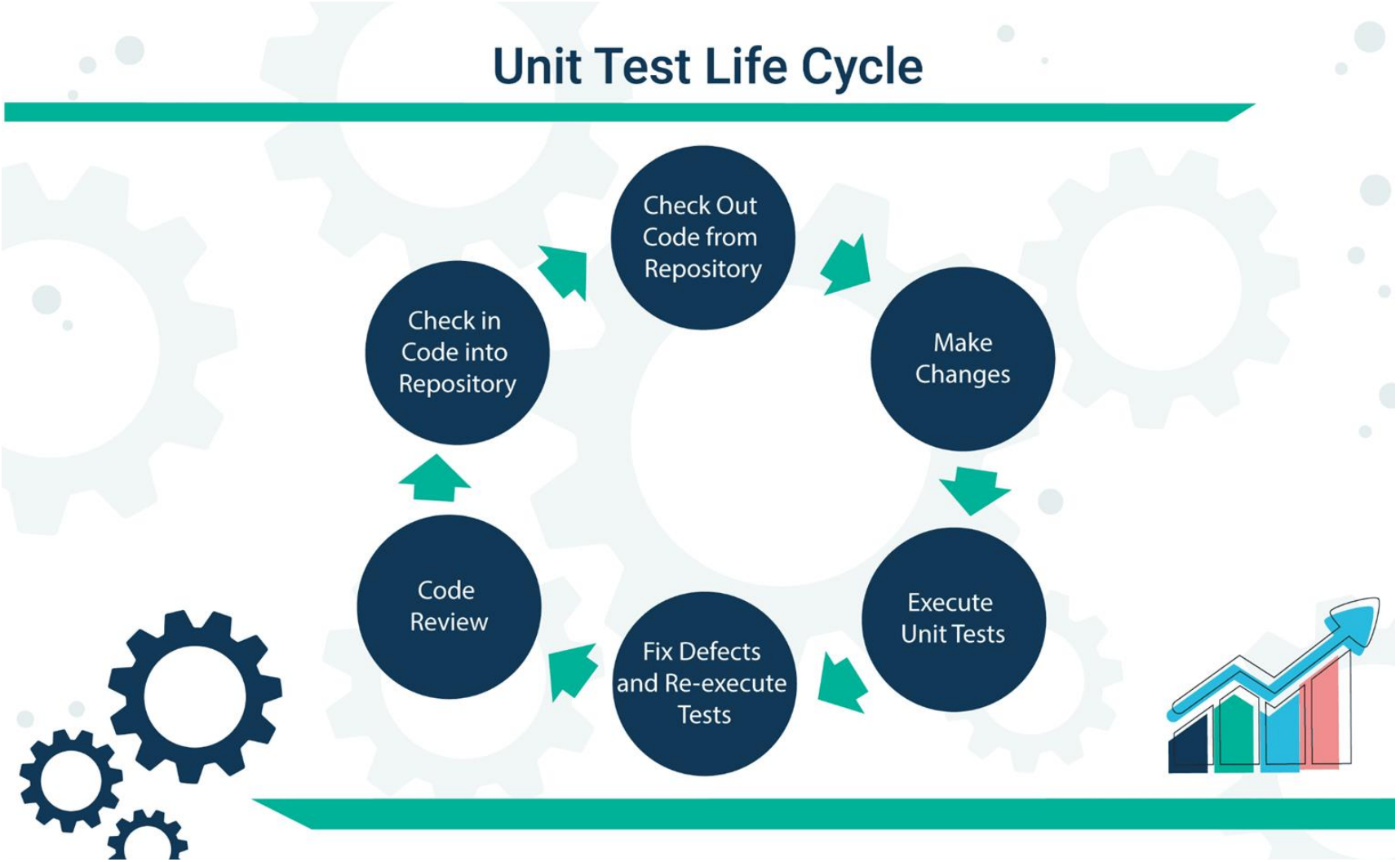
- Unit Testing do lập trình viên thực hiện.
- Unit Testing nên được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt quá trình phát triển phần mềm.

COMPONENT TESTING

- Unit testing bao gồm kiểm thử chức năng (như kiểm tra việc tính toán chính xác của thuật toán) hoặc các đặc điểm phi chức năng cụ thể liên quan đến tài nguyên (như rò rỉ bộ nhớ), kiểm thử hiệu suất (như tính toán hoàn thành đủ nhanh).

COMPONENT TESTING

- Vòng đời của kiểm thử thành phần



COMPONENT TESTING

Ví dụ Component Testing

Tạo một class MathUtil ở `..\src\main\java`

```
public class MathUtil {
    public int sum(int a, int b){
        return a+b;
    }
}
```

COMPONENT TESTING

Chúng ta viết vài test case nhẹ nhàng thông qua Annotation @Test và hàm assertEquals().

Tạo một class MathUtilTest tại ..\src\test\java


```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MathUtilTest {
    MathUtil util = new MathUtil();

    @Test
    void test1(){
        assertEquals(util.sum(1,1),2);
    }
    @Test
    void test2(){
        assertEquals(util.sum(1,1),3);
    }
    @Test
    void test3(){
        assertEquals(util.sum(1,2),3);
    }
}
```

COMPONENT TESTING

Giới thiệu về Stub và Driver

- Vì sản phẩm chưa hoàn thiện nên khi kiểm thử thành phần dựa trên 2 cơ sở: Stub và Driver
  - ✔ Stub là giả lập giao tiếp giữa các module ở cấp độ thấp hơn, chưa sẵn sàng để sử dụng hoặc chưa được phát triển. Thường được sử dụng trong phương pháp kiểm thử Top-down
  - ✔ Drivers mô phỏng giao tiếp giữa các module (interface) ở cấp độ cao nhất chưa sẵn sàng để sử dụng hoặc chưa được phát triển. Thường được sử dụng trong phương pháp kiểm thử Bottom-up

COMPONENT TESTING

Ví dụ Stub

Có 3 module: Đăng nhập, Trang chủ và Người dùng. Module Đăng nhập đã sẵn sàng và cần phải kiểm thử, nhưng khi chúng ta gọi các chức năng từ Trang chủ và Người dùng (các module này chưa code xong). Để kiểm thử module Đăng nhập, chúng ta cần phải viết một số code để giả lập 2 module trang chủ và người dùng, các module này sẽ trả giá trị về cho module Đăng nhập, code giả lập này được gọi là Stub.

COMPONENT TESTING

Ví dụ Driver

Nếu chúng ta đã code xong các module Trang chủ và Người dùng, trong khi đó module Đăng nhập chưa xong, và chúng ta cần phải test các module Trang chủ và Người dùng, các module này sẽ nhận giá trị từ module Đăng nhập. Vì vậy, để lấy được các giá trị từ module Đăng nhập chúng ta phải viết code để giả lập module Đăng nhập, module này sẽ trả giá trị về cho module trang chủ và người dùng. Các đoạn mã giả này được gọi là Driver.

COMPONENT TESTING

- Cơ sở đầu vào để thực hiện kiểm thử thành phần
  - Thiết kế chi tiết
  - Mã nguồn
  - Mô hình dữ liệu
  - Đặc tả thành phần (nếu có)
- Đối tượng kiểm thử thành phần
  - Thành phần (đơn vị hoặc mô-đun)
  - Lớp(Class), Phương thức(Method)...
  - Mô hình cơ sở dữ liệu

COMPONENT TESTING

- Ưu điểm của kiểm thử thành phần
  - Phát hiện sớm các lỗi (bug), giảm rủi ro, cải thiện chất lượng code.
  - Thẩm tra hành vi chức năng và phi chức năng của thành phần có đúng như được thiết kế và chỉ định hay không.
  - Xây dựng niềm tin vào chất lượng của thành phần: như đo lường độ phủ cấu trúc của kiểm thử, tạo sự tin tưởng rằng thành phần đã được kiểm thử kỹ càng theo kế hoạch.
  - Phát hiện lỗi trong từng thành phần của source code.
  - Ngăn ngừa tình trạng lọt lỗi xuống lần thử nghiệm sau.

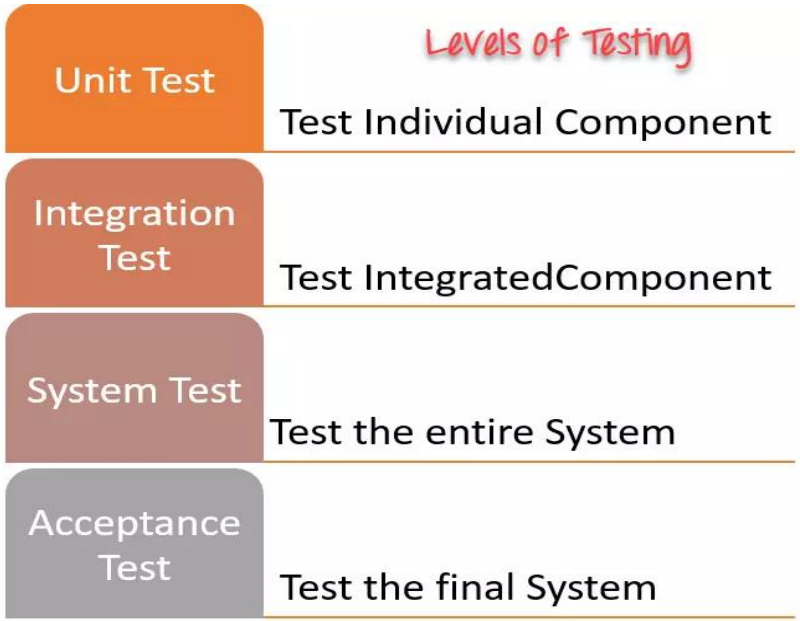
COMPONENT TESTING

Nhược điểm của kiểm thử thành phần

- Unit Test cũng là một chương trình, vì thế nó cần được tạo nên bởi các mã code.
- Không phải lập trình viên nào cũng xây dựng được hệ thống unit case chuẩn và hiệu quả. Đôi khi việc lập trình Unit Test thậm chí còn khó hơn cả xây dựng phần mềm. Chính vì thế, bạn phải là một lập trình viên dày dặn kinh nghiệm thì mới có thể tạo nên những Unit Test hiệu quả được.
- Việc tạo ra Unit Test cũng mất khá nhiều thời gian. Nhiều hệ thống Unit Case vô cùng đồ sộ và phức tạp đòi hỏi phải có nguyên một team vận hành. Nhiệm vụ của Unit Test là phát hiện lỗi nhưng đôi khi chính nó cũng mắc lỗi do người lập trình ra nó phạm sai lầm.

COMPONENT TESTING

- Những lỗi và hỏng hóc điển hình trong kiểm thử thành phần
  - Thành phần không được mô tả trong đặc tả thiết kế.
  - Vấn đề luồng dữ liệu.
  - Mã nguồn hoặc logic không chính xác.



KIỂM THỬ TÍCH HỢP

INTEGRATION TESTING

Integration Testing - Kiểm thử tích hợp

- Kiểm thử tích hợp là mức thứ 2 trong các mức kiểm thử phần mềm. Nó được thực hiện sau Unit Testing và trước System testing.
- Kiểm thử tích hợp là một mức của kiểm thử phần mềm kiểm tra một nhóm các module nhỏ liên quan đến nhau xem chúng có hoạt động đúng chức năng như trong thiết kế hay không.
- Kiểm thử tích hợp có thể được thực hiện bởi developer, một test team chuyên biệt hay một nhóm chuyên developer/kiểm thử viên tích hợp bao gồm cả kiểm thử phi chức năng.

INTEGRATION TESTING

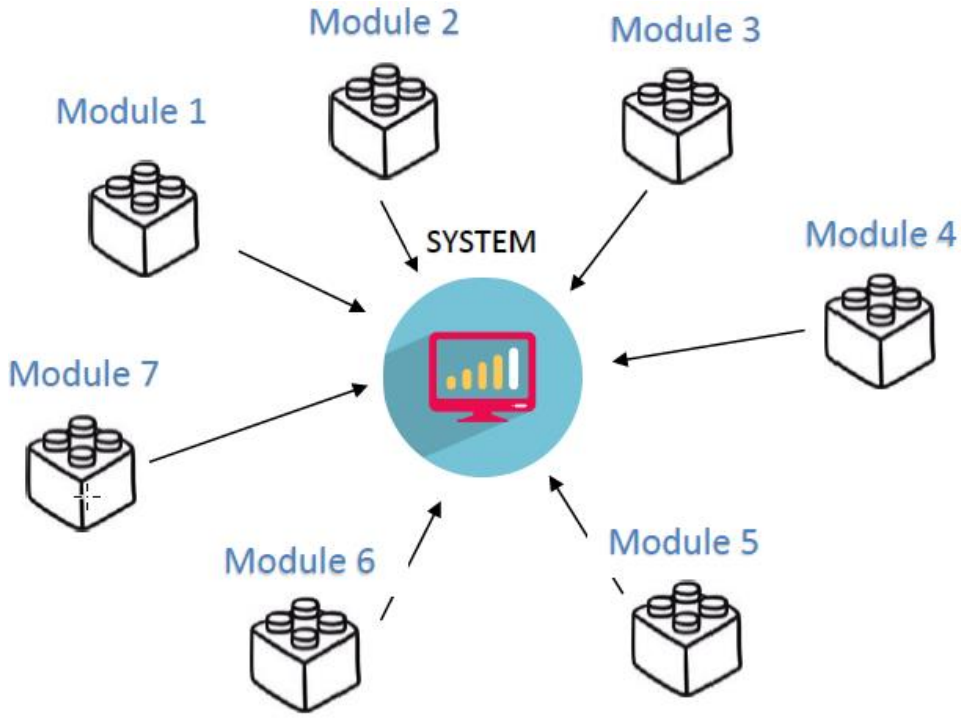
- Có 2 mức độ kiểm thử tích hợp :
  - ✔ Kiểm thử tích hợp thành phần: kiểm tra sự tương tác giữa các thành phần với điều kiện các thành phần đã pass ở phần kiểm thử thành phần trước đó
  - ✔ Kiểm thử tích hợp hệ thống: kiểm tra sự tương tác giữa các hệ thống con khác nhau và các hệ thống này đã pass ở lần kiểm thử trước đó

INTEGRATION TESTING

- Ví dụ kiểm thử tích hợp

INTEGRATION TESTING

- Giới thiệu về phương pháp kiểm thử Bigbang
  - Big Bang là phương pháp kiểm thử tích hợp thường được sử dụng cho những dự án nhỏ. Trong kiểm tra tích hợp Big Bang, tất cả những module sẽ được tích hợp và kiểm tra cùng một thời điểm.



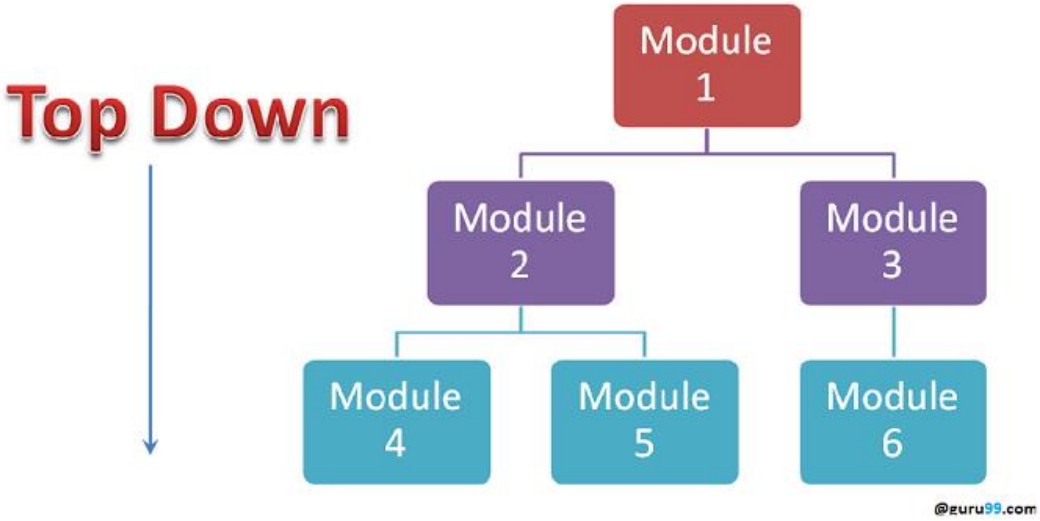
INTEGRATION TESTING

- Nhược điểm phương pháp kiểm thử Bigbang
  - Khó khăn trong việc phát hiện bug.
  - Có thể bỏ qua các bug giao diện nhỏ trong quá trình tìm bug.
  - Vì các module được kiểm thử cùng 1 lúc nên các module có nguy cơ bị cô lập trong quá trình kiểm thử.

INTEGRATION TESTING

Giới thiệu về phương pháp kiểm thử Top Down

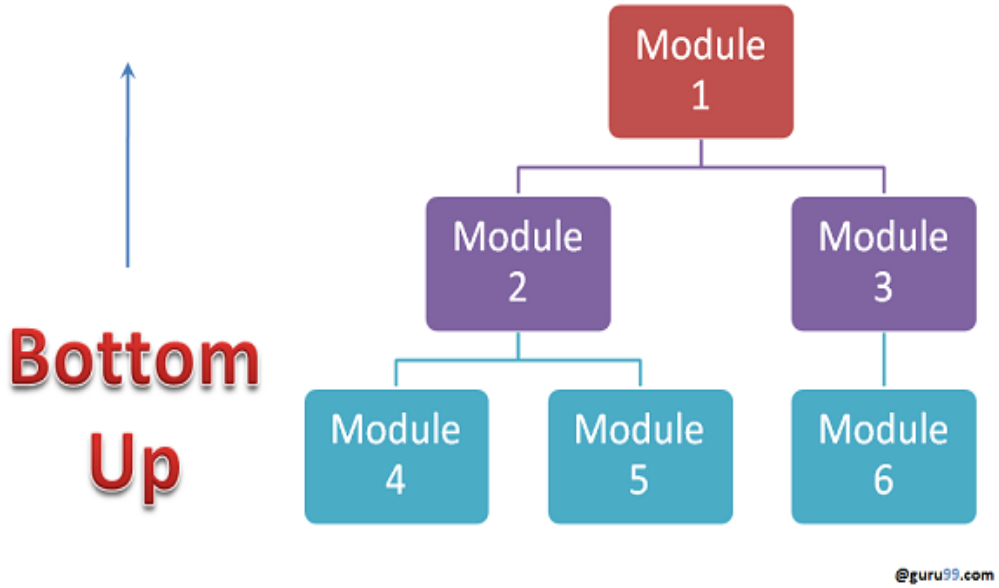
- Top Down là phương pháp kiểm thử tích hợp từ trên xuống dưới theo dòng điều khiển của hệ thống phần mềm. Sử dụng phương pháp Top Down sẽ giúp việc tìm kiếm bug trong từng module dễ dàng hơn rất nhiều và có thể tìm được lỗi lớn trong các module được ưu tiên. Tuy nhiên, để thực hiện phương pháp Top Down, ta cần rất nhiều Stubs.



INTEGRATION TESTING

Giới thiệu về phương pháp kiểm thử Bottom Up

- Bottom Up là phương pháp kiểm thử tích hợp ngược lại so với phương pháp Top Down. Nhờ việc tiếp cận các module từ thấp lên cao, lập trình viên có thể dễ dàng phát hiện những lỗi cơ bản nhất của phần mềm, đồng thời tìm kiếm và khắc phục lỗi ngay cả khi không cần đợi các module tích hợp lại với nhau. Tuy nhiên, phương pháp Bottom Up có sẽ không giữ được nguyên mẫu đầu tiên của hệ thống.



INTEGRATION TESTING

- Cơ sở đầu vào để kiểm thử tích hợp
  - Thiết kế phần mềm và hệ thống.
  - Sơ đồ phân rã chức năng.
  - Giao diện và đặc tả giao thức
  - Kiến trúc ở mức thành phần và hệ thống
  - Định nghĩa giao diện bên ngoài.

INTEGRATION TESTING

- Đối tượng kiểm thử tích hợp
  - Hệ thống con, cơ sở dữ liệu, cơ sở hạ tầng .
  - Giao diện lập trình ứng dụng API.

INTEGRATION TESTING

Ưu điểm kiểm thử tích hợp

- Giảm rủi ro, bằng cách kiểm thử các tích hợp rủi ro cao trước.
- Xác định xem hành vi chức năng và phi chức năng của giao diện có đúng như được thiết kế và chỉ định không.
- Xây dựng niềm tin vào chất lượng của giao diện.
- Phát hiện lỗi trong chính giao diện hoặc trong thành phần hoặc hệ thống đang được kiểm thử cùng nhau.
- Ngăn ngừa tình trạng lọt lỗi xuống lần thử nghiệm sau.

COMPONENT TESTING

- Những lỗi và hỏng hóc điển hình trong kiểm thử tích hợp
  - Trong kiểm thử tích hợp thành phần :
    - ✓ Dữ liệu không chính xác, thiếu dữ liệu.
    - ✓ Trình tự hoặc thời gian gọi giao diện không chính xác.

COMPONENT TESTING

- ✓ Giao diện không khớp, ví dụ một bên gửi tham số có giá trị > 1000 nhưng bên kia chỉ mong giá trị <= 1000.
- ✓ Lỗi giao tiếp giữa các thành phần.
- ✓ Lỗi giao tiếp giữa các thành phần không được xử lý hoặc xử lý không đúng cách.
- ✓ Giả định không chính xác về ý nghĩa, đơn vị hoặc ranh giới của dữ liệu được truyền giữa các thành phần.

COMPONENT TESTING

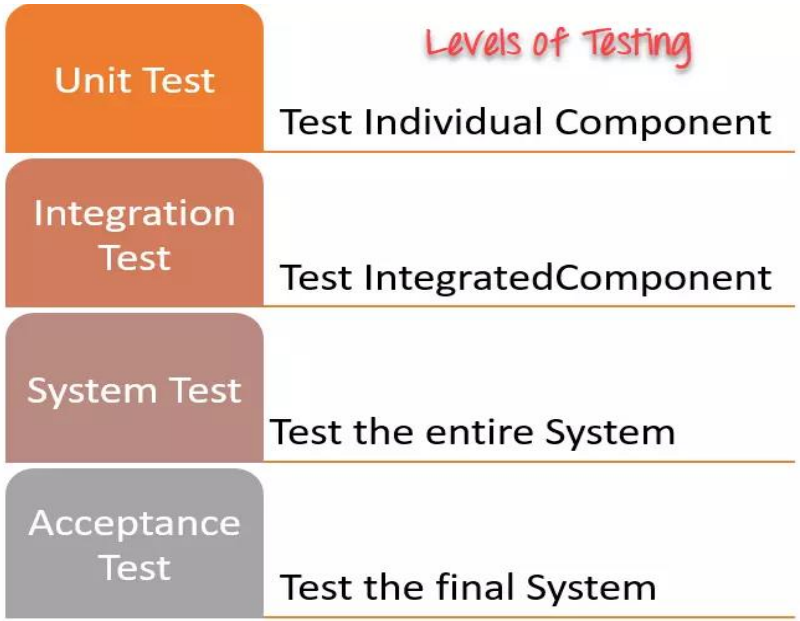
- Những lỗi và hỏng hóc điển hình trong kiểm thử tích hợp
  - Trong kiểm thử tích hợp hệ thống :
    - ✓ Cấu trúc thông điệp không nhất quán giữa các hệ thống.
    - ✓ Dữ liệu không chính xác, thiếu dữ liệu hoặc mã hóa dữ liệu không chính xác.
    - ✓ Lỗi giao tiếp giữa các hệ thống.

COMPONENT TESTING

- ✓ Lỗi giao tiếp giữa các hệ thống không được xử lý hoặc xử lý không đúng cách.
- ✓ Giả định không chính xác về ý nghĩa, đơn vị hoặc ranh giới của dữ liệu được truyền giữa các hệ thống.
- ✓ Không tuân thủ các quy định bảo mật bắt buộc.

COMPONENT TESTING

- Một số lưu ý trước khi thực hiện kiểm thử tích hợp
  - Đảm bảo rằng có tài liệu thiết kế chi tiết phù hợp trong đó các tương tác giữa mỗi thành phần được xác định rõ ràng.
  - Đảm bảo rằng có hệ thống quản lý cấu hình phần mềm mạnh mẽ tại chỗ. Hoặc nếu không, bạn sẽ có một thời gian khó khăn theo dõi phiên bản phù hợp của từng đơn vị, đặc biệt là nếu số lượng đơn vị được tích hợp là rất lớn.
  - Đảm bảo rằng mỗi thành phần được kiểm thử trước khi bạn bắt đầu Kiểm thử tích hợp.



KIỂM THỬ HỆ THỐNG

SYSTEM TESTING

System Testing - Kiểm thử hệ thống

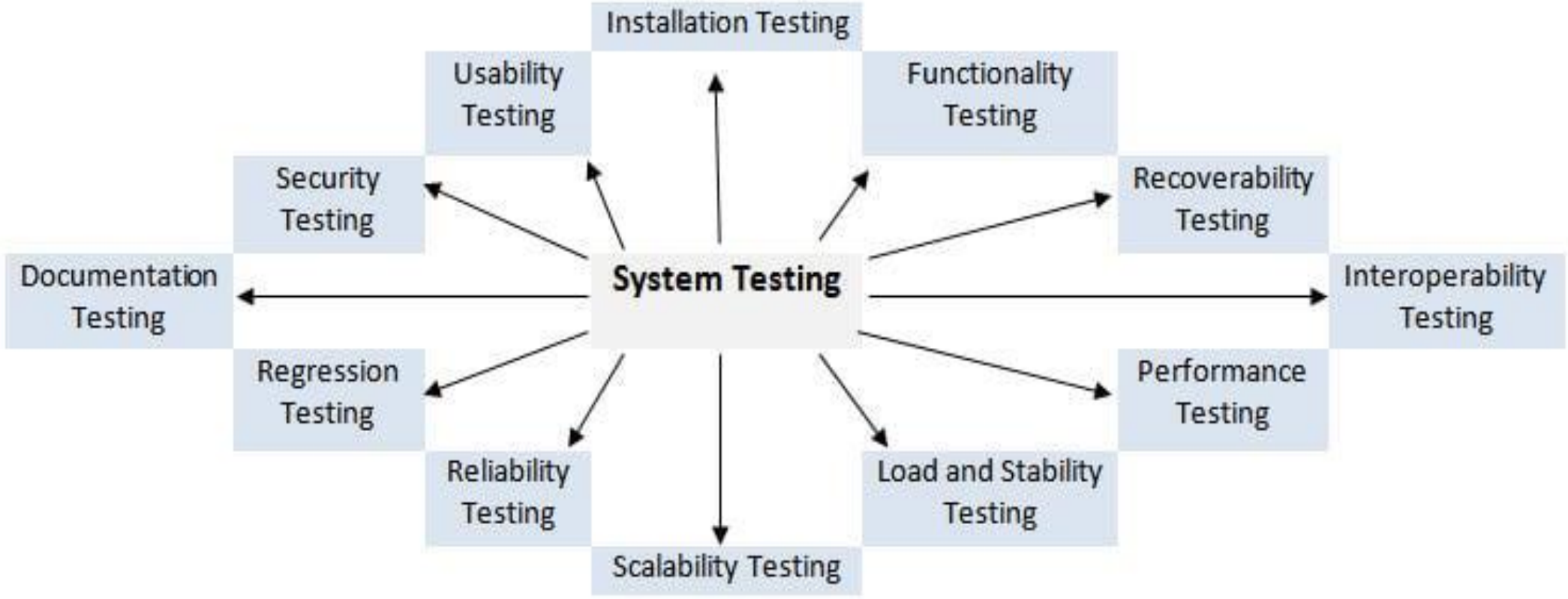
- Kiểm thử hệ thống là mức kiểm thử thứ 3 trong các mức kiểm thử phần mềm được thực hiện sau Integration Testing và trước Acceptance Testing.
- Kiểm thử hệ thống là kiểm tra lại toàn bộ hệ thống sau khi tích hợp. Nó cho phép kiểm tra sự tuân thủ của hệ thống theo yêu cầu. Loại kiểm thử này kiểm tra sự tương tác tổng thể của các thành phần. Nó liên quan đến tải, hiệu suất, độ tin cậy và kiểm tra bảo mật.

SYSTEM TESTING

- Kiểm thử hệ thống có thể dựa vào báo cáo phân tích rủi ro, đặc tả yêu cầu phần mềm, chức năng, quy trình nghiệp vụ, trường hợp sử dụng, hoặc đặc tả tổng quan về hành vi hệ thống, tương tác với hệ điều hành và các tài nguyên hệ thống.

SYSTEM TESTING

- Ví dụ kiểm thử hệ thống



SYSTEM TESTING

- Cơ sở đầu vào để kiểm thử hệ thống
  - Đặc tả yêu cầu hệ thống và phần mềm (chức năng và phi chức năng).
  - Báo cáo phân tích rủi ro trường hợp sử dụng.
  - Yêu cầu của người dùng.
  - Mô hình hành vi hệ thống.
  - Sơ đồ trạng thái hệ thống.
  - Hướng dẫn sử dụng.

SYSTEM TESTING

- Đối tượng kiểm thử hệ thống
  - Phần cứng/phần mềm hệ thống.
  - Hệ điều hành.
  - Cấu hình hệ thống đang kiểm thử.
  - Cấu hình hệ thống và dữ liệu cấu hình.

SYSTEM TESTING

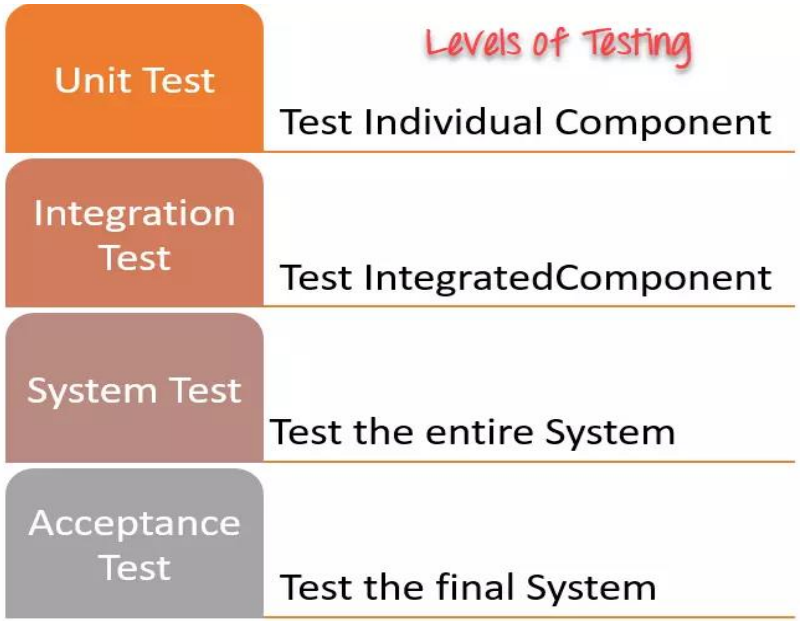
- Mục đích kiểm thử hệ thống
  - Giảm thiểu rủi ro.
  - Xác định các hành vi chức năng và phi chức năng của hệ thống có như được chỉ định hay không.
  - Xác minh rằng hệ thống đã hoàn tất và sẽ hoạt động đúng như mong đợi.
  - Xây dựng niềm tin vào chất lượng của toàn bộ hệ thống phát hiện lỗi.
  - Ngăn ngừa tình trạng lọt lỗi xuống lần thử nghiệm sau hoặc xuống khâu sản xuất.

SYSTEM TESTING

- Những lỗi và hỏng hóc điển hình trong kiểm thử hệ thống
  - Tính toán không chính xác.
  - Hành vi chức năng hoặc phi chức năng không chính xác hoặc không như mong đợi của hệ thống.
  - Sự kiểm soát và/hoặc luồng công việc không chính xác trong hệ thống.

SYSTEM TESTING

- Không thực hiện đúng và đủ các tác vụ có chức năng đầu-cuối.
- Hệ thống không hoạt động một cách đúng đắn trong môi trường thực tế.
- Hệ thống không hoạt động đúng như được mô tả trong hướng dẫn sử dụng.



KIỂM THỬ CHẤP NHẬN

ACCEPTANCE TESTING

Acceptance Testing - Kiểm thử chấp nhận

- Kiểm thử chấp nhận là mức thứ 4 được thực hiện sau khi hoàn thành kiểm thử hệ thống và trước khi đưa sản phẩm vào sử dụng chính thức.
- Kiểm thử chấp nhận chính thức liên quan đến yêu cầu và quy trình kinh doanh để xác định liệu hệ thống có đáp ứng tiêu chí chấp nhận hay không và cho phép người dùng, khách hàng hoặc tổ chức được ủy quyền khác xác định có chấp nhận hệ thống hay không.

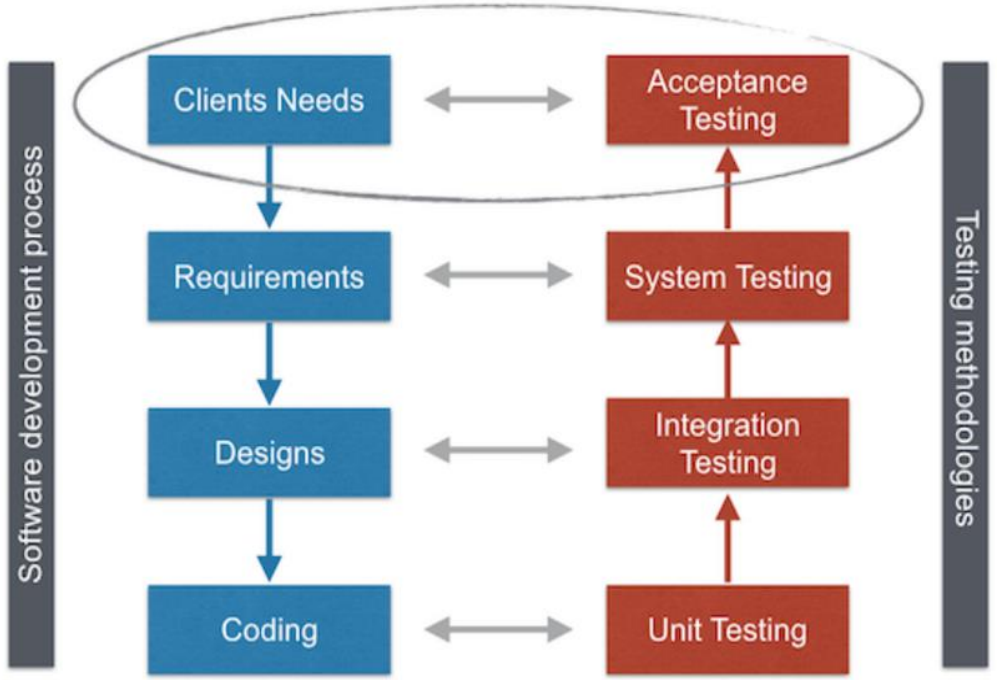
ACCEPTANCE TESTING

- Kiểm thử chấp nhận được chia thành 2 mức khác nhau
  - ✓ Kiểm thử Alpha: được thực hiện bởi những người trong tổ chức nhưng không tham gia phát triển phần mềm.
  - ✓ Kiểm thử Beta: được thực hiện bởi khách hàng hoặc người dùng cuối tại địa điểm của người dùng cuối.

ACCEPTANCE TESTING

Ví dụ kiểm thử chấp nhận

Phần mềm xuất sắc vượt qua các phép kiểm thử về chức năng thực hiện bởi nhóm thực hiện dự án, nhưng khách hàng khi kiểm thử sau cùng vẫn thất vọng vì bố cục màn hình nghèo nàn, thao tác không tự nhiên, không theo tập quán sử dụng của khách hàng…



ACCEPTANCE TESTING

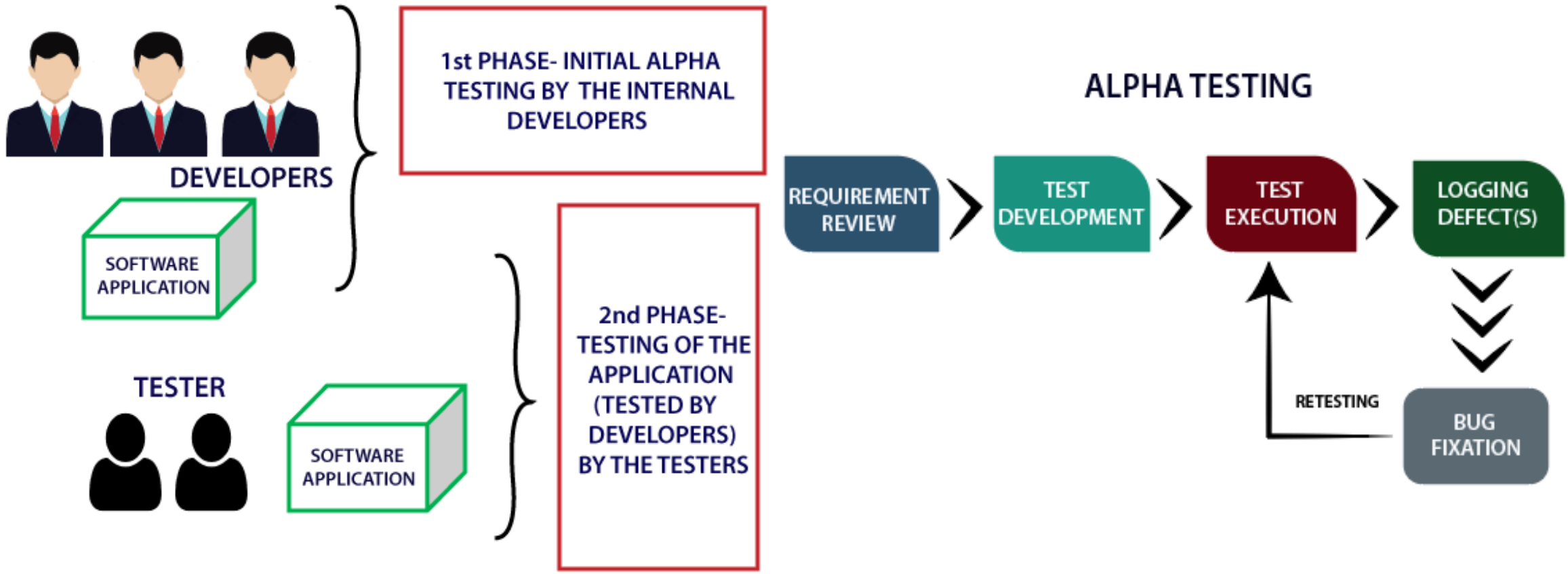
- Giới thiệu về phương pháp kiểm thử Alpha
  - Alpha testing là một dạng của Acceptance testing.
  - Được gọi là Alpha vì nó được thực hiện sớm, gần cuối của sự phát triển của phần mềm, và trước khi thử nghiệm Beta.
  - Alpha testing thực hiện để xác định tất cả các vấn đề/ lỗi có thể xảy ra trước khi phát hành sản phẩm đến tay người dùng.

ACCEPTANCE TESTING

- Trọng tâm của việc kiểm thử Alpha là để người dùng thực tế trải nghiệm sản phẩm.
- Alpha testing được thực hiện trong môi trường lab và thường các tester là nhân viên nội bộ của tổ chức, công ty.

ACCEPTANCE TESTING

- Đối tượng tham gia kiểm thử Alpha



ACCEPTANCE TESTING

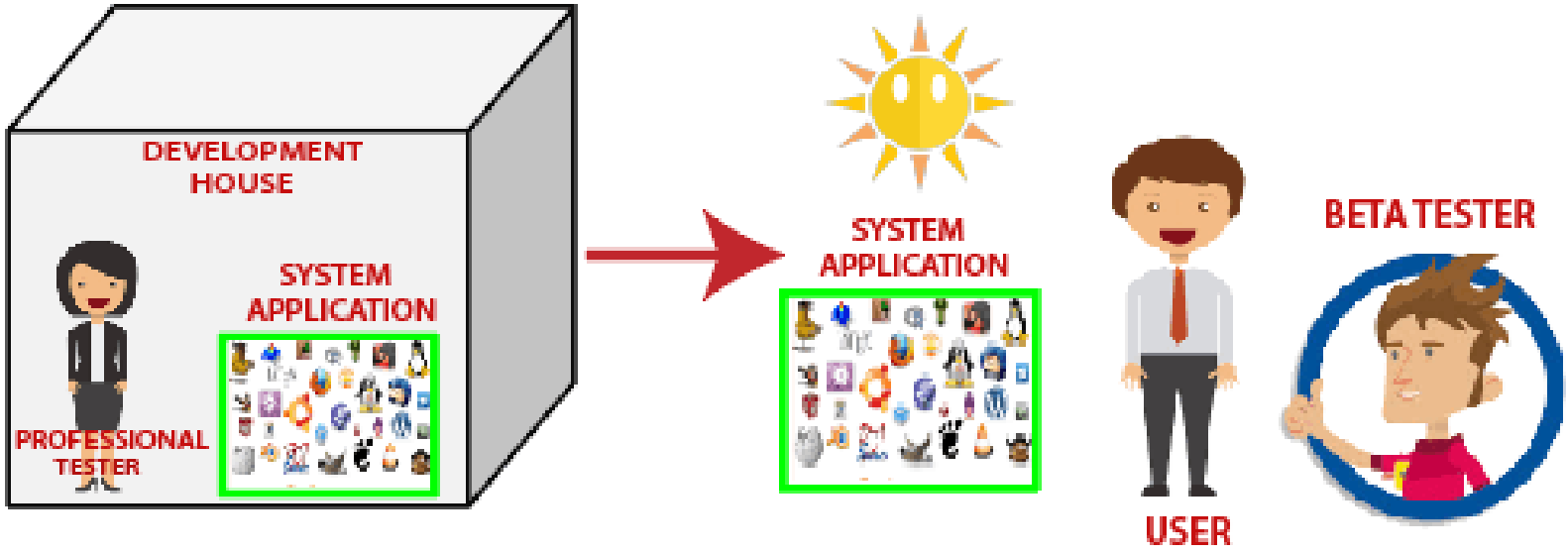
- Giới thiệu về phương pháp kiểm thử Beta
  - Beta testing là một dạng của Acceptance testing và thực hiện bởi người dùng ngoài đội dự án phát triển.
  - Phiên bản beta của phần mềm chỉ được phát hành/công bố cho một số lượng hạn chế người dùng cuối để lấy thông tin phản hồi về chất lượng sản phẩm.
  - Beta testing làm giảm nguy cơ thất bại của sản phẩm và tăng độ tin tưởng vào chất lượng của nó thông qua các ý kiến nhận xét, đánh giá từ khách hàng.

ACCEPTANCE TESTING

- Beta testing là bước kiểm tra cuối cùng trước khi chuyển một phần mềm đến tay khách hàng.
- Lợi thế lớn nhất của beta test là phản hồi trực tiếp từ phía người dùng cuối, nó giúp kiểm tra phần mềm trong môi trường thực tế.

ACCEPTANCE TESTING

- Đối tượng tham gia kiểm thử Beta



ACCEPTANCE TESTING

- Cơ sở đầu vào để kiểm thử chấp nhận
  - Quy trình nghiệp vụ
  - Yêu cầu của người dùng hoặc yêu cầu nghiệp vụ quy định, hợp đồng pháp lý và tiêu chuẩn
  - Trường hợp sử dụng (use case)
  - Yêu cầu hệ thống
  - Tài liệu người dùng hoặc hệ thống
  - Thủ tục cài đặt
  - Báo cáo phân tích rủi ro

ACCEPTANCE TESTING

- Đối tượng kiểm thử chấp nhận
  - Hệ thống đang kiểm thử
  - Cấu hình hệ thống và dữ liệu cấu hình
  - Quy trình nghiệp vụ cho một hệ thống tích hợp đầy đủ
  - Hệ thống khôi phục và hot site (để kiểm thử tính liên tục của nghiệp vụ và phục hồi sau thảm họa)
  - Quy trình vận hành và bảo trì
  - Báo cáo

ACCEPTANCE TESTING

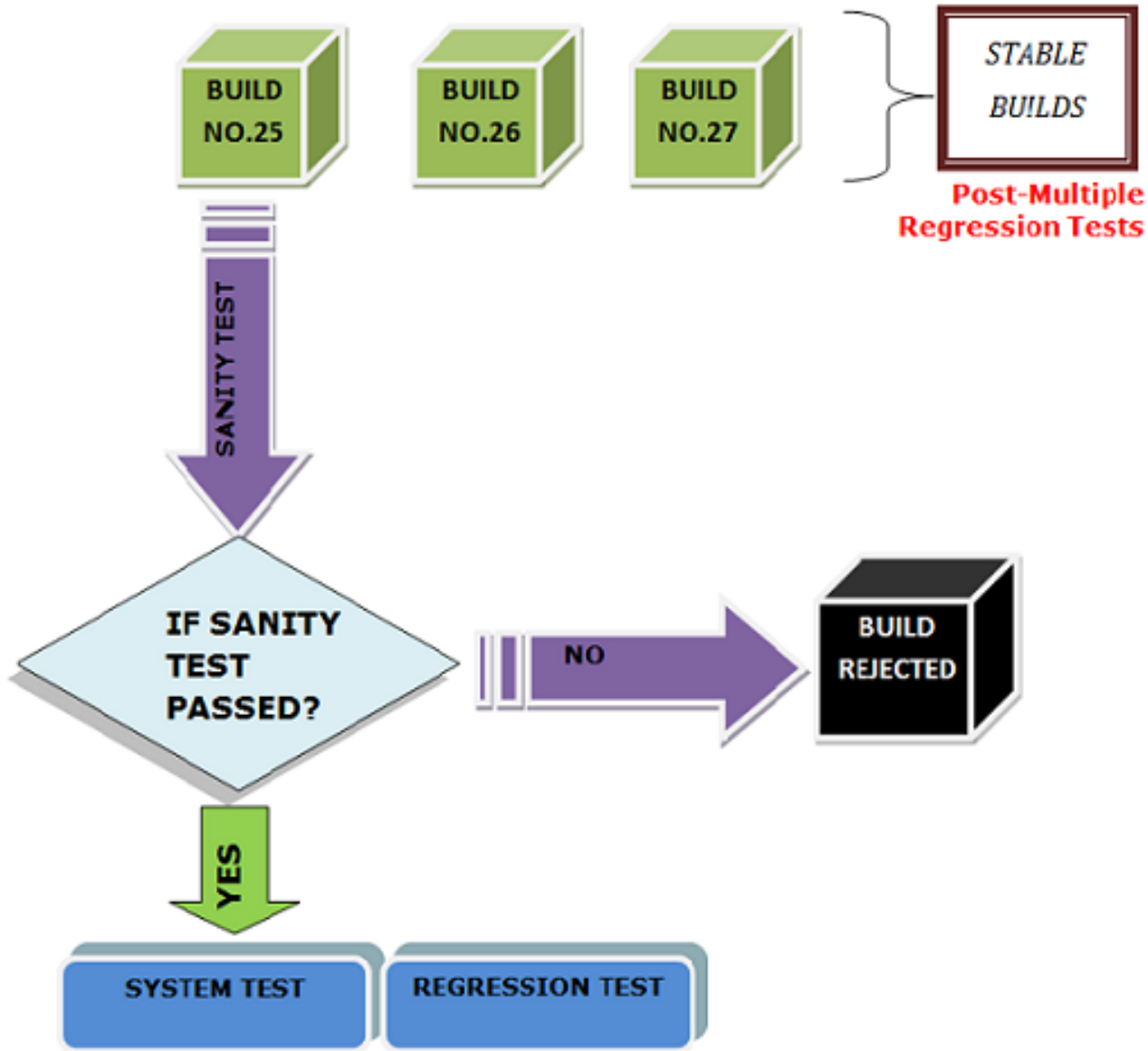
- Lỗi và hỏng hóc điển hình trong kiểm thử chấp nhận
  - Luồng công việc của hệ thống không đáp ứng yêu cầu nghiệp vụ và người dùng
  - Quy tắc nghiệp vụ không được thực hiện đúng
  - Hệ thống không đáp ứng yêu cầu về hợp đồng hoặc pháp lý
  - Lỗi phi chức năng: như lỗ hổng bảo mật, hiệu suất không đủ để đáp ứng tải cao, hoặc hoạt động không đúng trên nền tảng được hỗ trợ

🔔 Chú ý:

- Các mức kiểm thử thành phần, kiểm thử tích hợp và kiểm thử hệ thống là thực hiện việc kiểm tra Xác minh(**Verification**)
- Mức kiểm thử chấp nhận là thực hiện việc kiểm tra Thẩm định (**Validation**)

MỘT SỐ LOẠI KIỂM THỬ QUAN TRỌNG

KIỂM THỬ ĐỘ TỈNH TÁO


BUILD NO.25
BUILD NO.26
BUILD NO.27
STABLE BUILDS
Post-Multiple Regression Tests
SANITY TEST
IF SANITY TEST PASSED?
NO
BUILD REJECTED
YES
SYSTEM TEST
REGRESSION TEST

SANITY TESTING

Sanity Testing - Kiểm thử độ tỉnh táo

- Sanity testing là một loại Kiểm thử phần mềm được thực hiện sau khi nhận được một bản build phần mềm, với những thay đổi nhỏ về mã, hoặc chức năng, để xác định rằng các lỗi đã được sửa và không có vấn đề gì khác xảy ra do những thay đổi này.

SANITY TESTING

Ví dụ kiểm thử độ tỉnh táo

Hệ thống phần mềm bán dừa qua internet phiên bản 1.0 với 20 chức năng khác nhau. Nhưng khi tạo một hóa đơn cho người mua dừa có lỗi không hiển thị địa chỉ của người này. Người chủ muốn qua phiên bản: 2.0 sẽ sửa được lỗi trên và có thể hiển thị thêm Coupon khi tạo hóa đơn.

=> Là tester ta áp dụng kỹ thuật Sanity Test

Khi áp dụng: ta sẽ thực hiện việc testing trên hai phần:

Một là kiểm tra lỗi không hiển thị địa chỉ có còn hay không, hai là kiểm tra chức năng mới thêm Coupon có được thực thi vào hay không?

SANITY TESTING

- Cơ sở đầu vào để kiểm thử độ tỉnh táo
  - Các issues đã được tạo và có trạng thái cần testing.

- Đối tượng kiểm thử độ tỉnh táo
  - Bug hoặc chức năng thay đổi.

SANITY TESTING

- Mục đích kiểm thử độ tỉnh táo
  - Xác định rằng chức năng được đề xuất hoạt động gần như mong đợi. Nếu Sanity testing không thành công, bản build bị từ chối để tiết kiệm thời gian và chi phí liên quan đến một thử nghiệm nghiêm ngặt hơn.
  - Tiết kiệm nhiều thời gian và công sức bởi vì Sanity testing tập trung vào một hoặc một vài vùng ảnh hưởng của chức năng.

SANITY TESTING

- Không mất công sức để chuẩn bị tài liệu của Sanity Testing vì nó thường không có kịch bản
- Nó giúp xác định các đối tượng thiếu phụ thuộc
- Nó được sử dụng để verify rằng một chức năng nhỏ của ứng dụng vẫn hoạt động đúng sau thay đổi nhỏ.

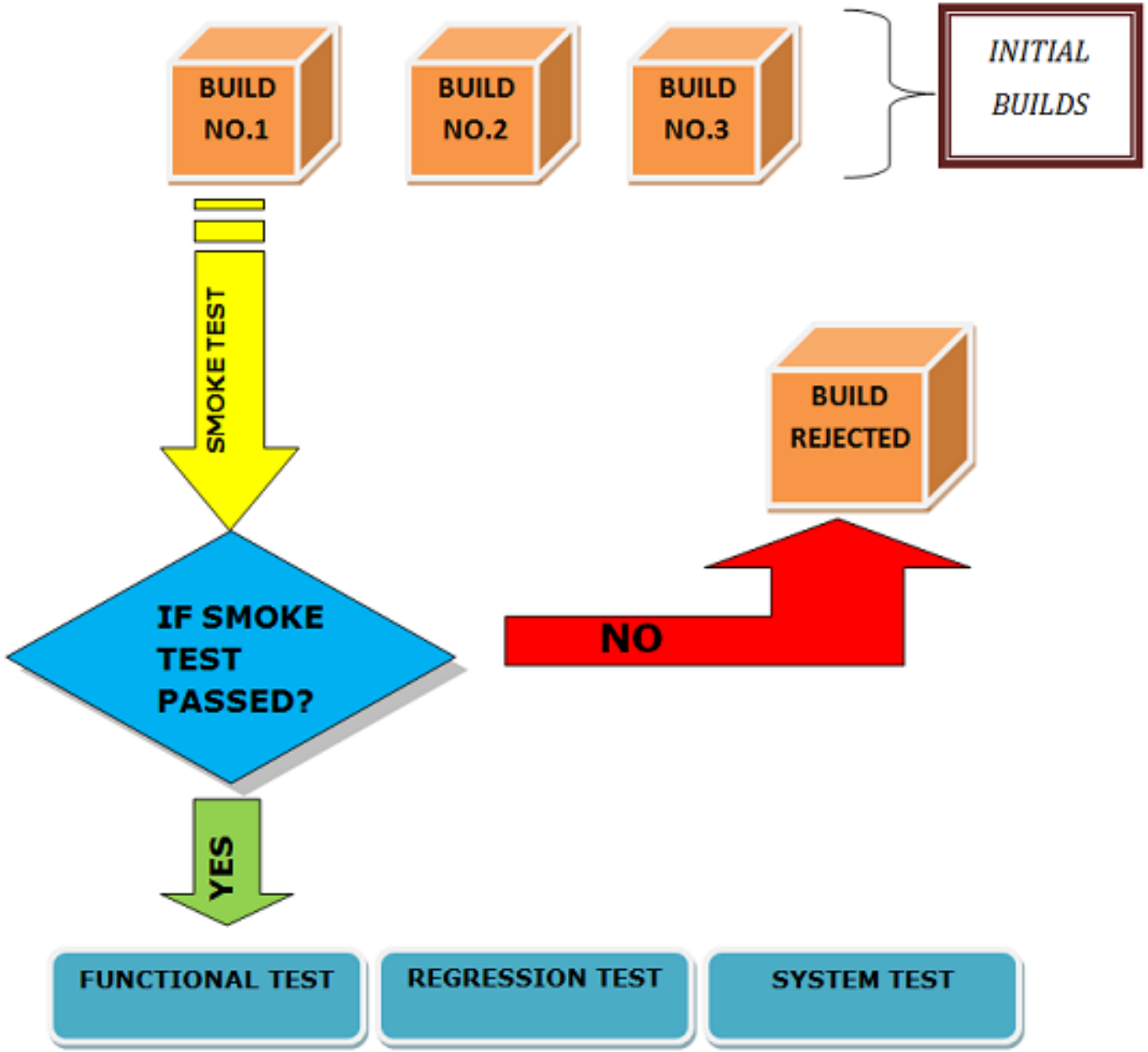
SANITY TESTING

- Những lỗi và hỏng hóc điển hình trong kiểm thử độ tỉnh táo
  - Dễ bị sót bug khi Sanity testing chỉ focus vào các câu lệnh và các function của phần mềm.
  - Nó không đi đến mức cấu trúc thiết kế vì vậy rất khó để developers hiểu cách fix những issue được tìm thấy trong sanity testing.

SANITY TESTING

- Trong Sanity testing, việc test chỉ được thực hiện cho một vài chức năng hạn chế, vì vậy nếu có vấn đề xảy ra với những chức năng khác thì sẽ khó để phát hiện
- Sanity testing thường không có kịch bản vì vậy việc tham khảo cho tương lai là không có sẵn.

KIỂM THỬ KHÓI


BUILD NO.1
BUILD NO.2
BUILD NO.3
INITIAL BUILDS
SMOKE TEST
IF SMOKE TEST PASSED?
NO
BUILD REJECTED
YES
FUNCTIONAL TEST
REGRESSION TEST
SYSTEM TEST

SMOKE TESTING

Smoke Testing - Kiểm thử khói

- Khi có 1 bản build mới, Test team xác định chức năng chính trong ứng dụng để thực hiện Smoke Testing xem liệu phần mềm có bị vấn đề gì ở những chức năng chính hay không.
- Smoke testing được thực hiện trước giai đoạn kiểm thử hồi quy
- Smoke Testing giúp xác định xem build có thiếu sót gì không để tránh lãng phí thời gian và tài nguyên.

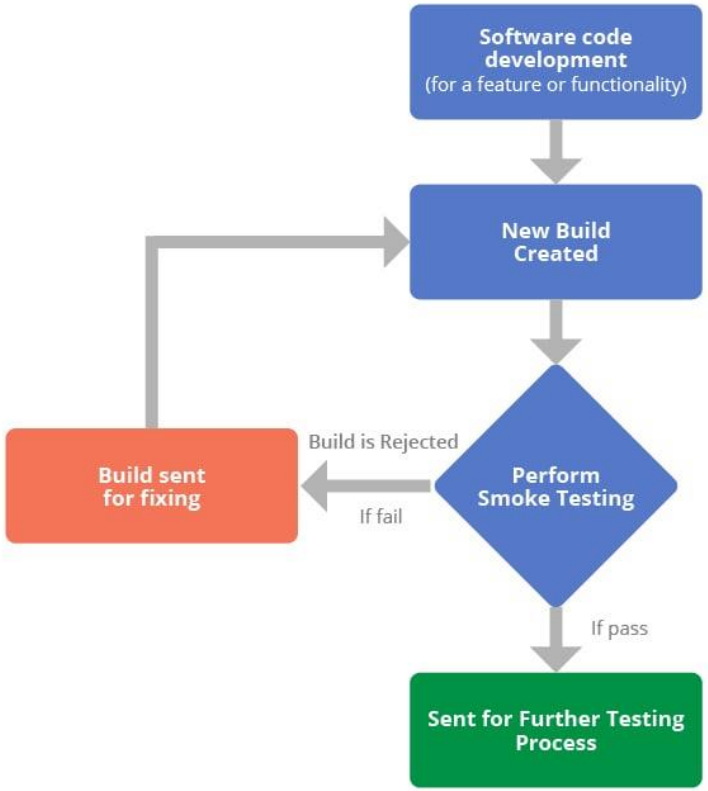
SMOKE TESTING

- Smoke Testing được thực hiện bất cứ khi nào các chức năng mới của phần mềm được phát triển và tích hợp với bản build hiện hành mà được triển khai trong môi trường kiểm thử. Nó đảm bảo rằng tất cả các chức năng quan trọng đang hoạt động chính xác hay không.

SMOKE TESTING

Ví dụ kiểm thử khói

Nút đăng ký mới được bổ sung vào màn hình đăng nhập và bản build được triển khai với code mới. Chúng ta thực hiện Smoke Testing trên 1 bản build mới.



SMOKE TESTING

- Cơ sở đầu vào để kiểm thử khói
  - Nội dung thay đổi của chức năng được định nghĩa bằng Task.

- Đối tượng kiểm thử khói
  - Các tính năng mới được triển khai và tích hợp vào hệ thống.

SMOKE TESTING

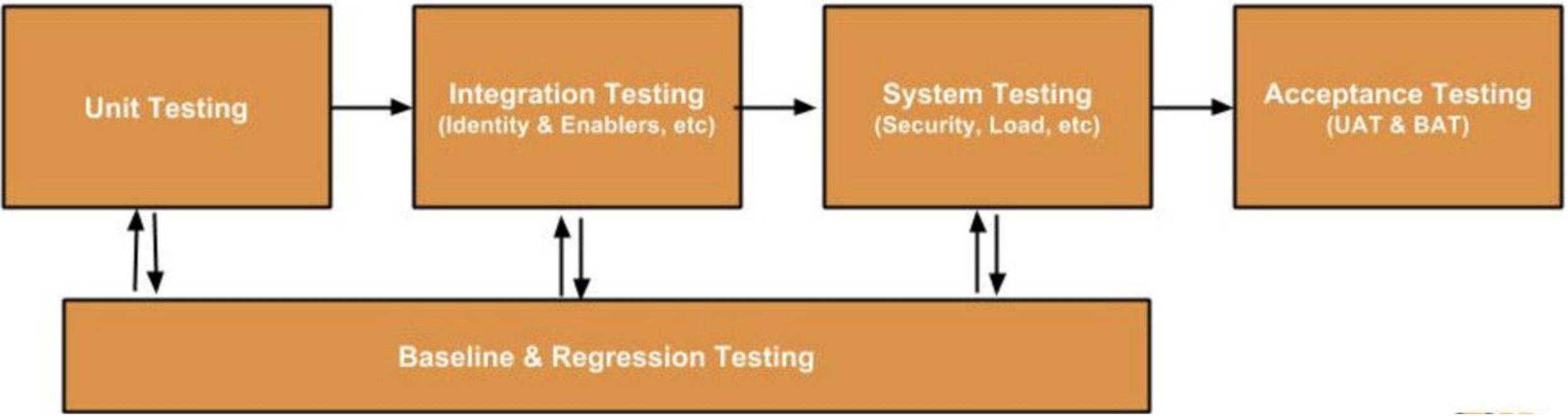
- Mục đích kiểm thử khói
  - Dễ dàng thực hiện việc test
  - Các lỗi sẽ được nhận diện trong giai đoạn đầu
  - Cải thiện chất lượng của hệ thống
  - Giảm thiểu rủi ro
  - Tiến trình dễ dàng truy cập
  - Tối ưu hiệu quả và thời gian của việc test
  - Dễ dàng phát hiện các lỗi quan trọng và sửa chữa các lỗi
  - Giảm thiểu các rủi ro phát sinh

SMOKE TESTING

- Những lỗi và hỏng hóc điển hình trong kiểm thử khói
  - Tính năng mới tích hợp vào hệ thống được bàn giao không đúng đặc tả yêu cầu.
  - Tính năng đã bàn giao tuy nhiên hệ thống chưa sẵn sàng tích hợp.
  - Khi tích hợp vào hệ thống thì tính năng bàn giao hoạt động đúng đặc tả nhưng lại gây lỗi tới những tính năng khác.

KIỂM THỬ HỒI QUY

Testing Levels

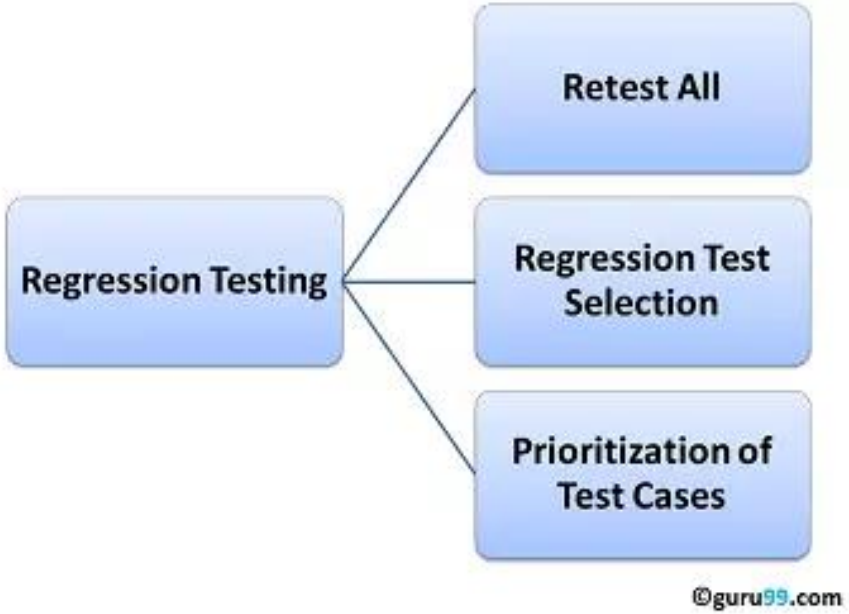


REGRESSION TESTING

- Regression Testing - Kiểm thử hồi quy
  - Kiểm thử hồi quy xác nhận rằng một tính năng mới được thêm không ảnh hưởng xấu đến các tính năng hiện có.
  - Kiểm thử hồi quy là kiểm thử lại các trường hợp đã được thực hiện để đảm bảo các chức năng hiện có hoạt động tốt.
  - Kiểm thử hồi quy đảm bảo rằng những thay đổi source code sẽ không ảnh hưởng tới các chức năng hiện có, đảm bảo rằng code cũ vẫn hoạt động sau khi thực hiện thay đổi source code.

REGRESSION TESTING

- Phương pháp Kiểm thử hồi quy
  - Retest All: Kiểm tra lại tất cả
  - Regression Test Selection: Lựa chọn kiểm tra hồi quy
  - Prioritization of Test Cases: Ưu tiên cho trường hợp thử nghiệm.



🔔 Chú ý:

- Khi tester kiểm tra chức năng A với 5 ca test case có một ca bị fail chuyển về dev để sửa code.
- Khi dev chuyển lại cho tester. Nếu tester thực hiện cả 5 ca test case không quan tâm là trước đó nó là pass hay fail thì gọi là kiểm thử hồi quy. Còn nếu tester chỉ thực hiện lại ca test case bị fail thì đó gọi là re-test.

REGRESSION TESTING

- Ví dụ kiểm thử hồi quy

  Ví dụ 1: Khi chức năng phần mềm được thay đổi

  Giả sử có một ứng dụng quản lý sinh viên, phần mềm này có chức năng thêm, sửa, xóa sinh viên và tất cả các tính năng này đang hoạt động bình thường. Tuy nhiên khách hàng muốn thay đổi nút xóa thành toggle để enable hoặc disable sinh viên. Việc thay đổi tính năng này phải được kiểm thử để đảm bảo rằng tính năng được thay đổi hoạt động bình thường và tất cả các tính năng cũ cũng được hoạt động bình thường. Quá trình này được gọi là kiểm thử hồi quy

REGRESSION TESTING

- Ví dụ kiểm thử hồi quy

  Ví dụ 2: Khi thêm 1 tính năng mới vào ứng dụng

  - Phần mềm ứng dụng luôn luôn được cải tiến thêm các tính năng mới để đáp ứng nhu cầu của người dùng. Chính vì thế khi một chức năng mới được thêm vào phần mềm đã được phát triển chúng ta cũng cần thực hiện kiểm thử hồi quy trên toàn bộ ứng dụng để đảm bảo rằng việc thêm chức năng mới đó không làm ảnh hưởng đến các chức năng cũ đã được phát triển.

REGRESSION TESTING

Ví dụ kiểm thử hồi quy

Ví dụ 3: Khi developer fix bug

- Trong quá trình fix bug có thể developer fix được bug của chức năng A nhưng chức năng B có liên quan đến chức năng A lại bị lỗi nào đó do quá trình sửa code gây ra. Chính vì vậy khi developer thực hiện fix bug chúng ta sẽ **Retest** lại chức năng A và thực hiện **Regression test** chức năng B để đảm bảo các chức năng của hệ thống vẫn hoạt động tốt.

REGRESSION TESTING

- Cơ sở đầu vào để kiểm thử hồi quy
  - Tài liệu đặc tả
  - Testcase
- Đối tượng kiểm thử hồi quy
  - Thay đổi trong yêu cầu và source code được sửa đổi theo yêu cầu
  - Tính năng mới được thêm vào phần mềm
  - Sửa lỗi
  - Khắc phục sự cố hiệu suất

REGRESSION TESTING

- Mục đích kiểm thử hồi quy
  - Dễ dàng thực hiện việc test
  - Các lỗi sẽ được nhận diện trong giai đoạn đầu
  - Cải thiện chất lượng của hệ thống
  - Giảm thiểu rủi ro
  - Tiến trình dễ dàng truy cập
  - Tối ưu hiệu quả và thời gian của việc test
  - Dễ dàng phát hiện các lỗi quan trọng và sửa chữa các lỗi
  - Giảm thiểu các rủi ro phát sinh

REGRESSION TESTING

- Những lỗi và hỏng hóc điển hình trong kiểm thử hồi quy
  - Kiểm thử có thể vẫn còn lỗi do khi kiểm thử hồi quy liên tục được thực hiện, các bộ test cases trở nên khá lớn. Do hạn chế về thời gian và ngân sách, toàn bộ test cases kiểm thử hồi quy không thể được thực thi.
  - Tối thiểu hóa bộ test cases trong khi vẫn đạt được phạm vi kiểm thử tối đa vẫn là một thách thức
  - Mỗi lần sửa đổi hoặc mỗi lần cập nhật bản build hoặc sau một loạt các lỗi được sửa là một thách thức.

SO SÁNH KIỂM THỬ HỒI QUY VỚI KIỂM THỬ KHÓI & KIỂM THỬ ĐỘ TỈNH TÁO

	Kiểm thử khói (Smoke Testing)	Kiểm thử độ tỉnh táo (Sanity Testing)	Kiểm thử hồi quy (Regression Testing)
Mục đích	Kiểm tra xem hệ thống có hoạt động sơ bộ sau khi build không.	Kiểm tra nhanh bản sửa lỗi hoặc thay đổi nhỏ.	Kiểm tra xem thay đổi mới có làm ảnh hưởng đến các chức năng cũ không.
Phạm vi	Rộng, kiểm tra các chức năng chính.	Hẹp, chỉ kiểm tra phần bị ảnh hưởng.	Rộng, kiểm tra cả tính năng cũ và mới.
Thời điểm thực hiện	Sau mỗi lần build mới.	Sau khi sửa lỗi hoặc có thay đổi nhỏ.	Sau mỗi lần cập nhật, sửa lỗi hoặc thêm tính năng mới.
Cách thực hiện	Test case đơn giản, kiểm tra nhanh.	Kiểm tra nhanh mà không cần đi sâu.	Chạy lại các test case cũ, có thể tự động hóa.

SỬ DỤNG CÁC LOẠI KIỂM THỬ

Loại Kiểm Thử	Mức Độ Kiểm Thử	Mục Đích Chính
Kiểm thử khói (Smoke Testing)	Kiểm thử tích hợp, Kiểm thử hệ thống	Kiểm tra xem phần mềm có thể chạy cơ bản hay không sau khi build mới.
Kiểm thử độ tỉnh táo (Sanity Testing)	Kiểm thử tích hợp, Kiểm thử hệ thống	Xác minh nhanh các thay đổi hoặc bản sửa lỗi có hoạt động đúng không mà không kiểm tra toàn bộ hệ thống.
Kiểm thử hồi quy (Regression Testing)	Kiểm thử tích hợp, Kiểm thử hệ thống, Kiểm thử chấp nhận	Đảm bảo rằng các thay đổi mới không ảnh hưởng đến các chức năng cũ của phần mềm.