HIENLTH

Verification, Validation and Testing

Chủ đề 6: Kiểm thử Phần mềm

Tài liệu – Textbook

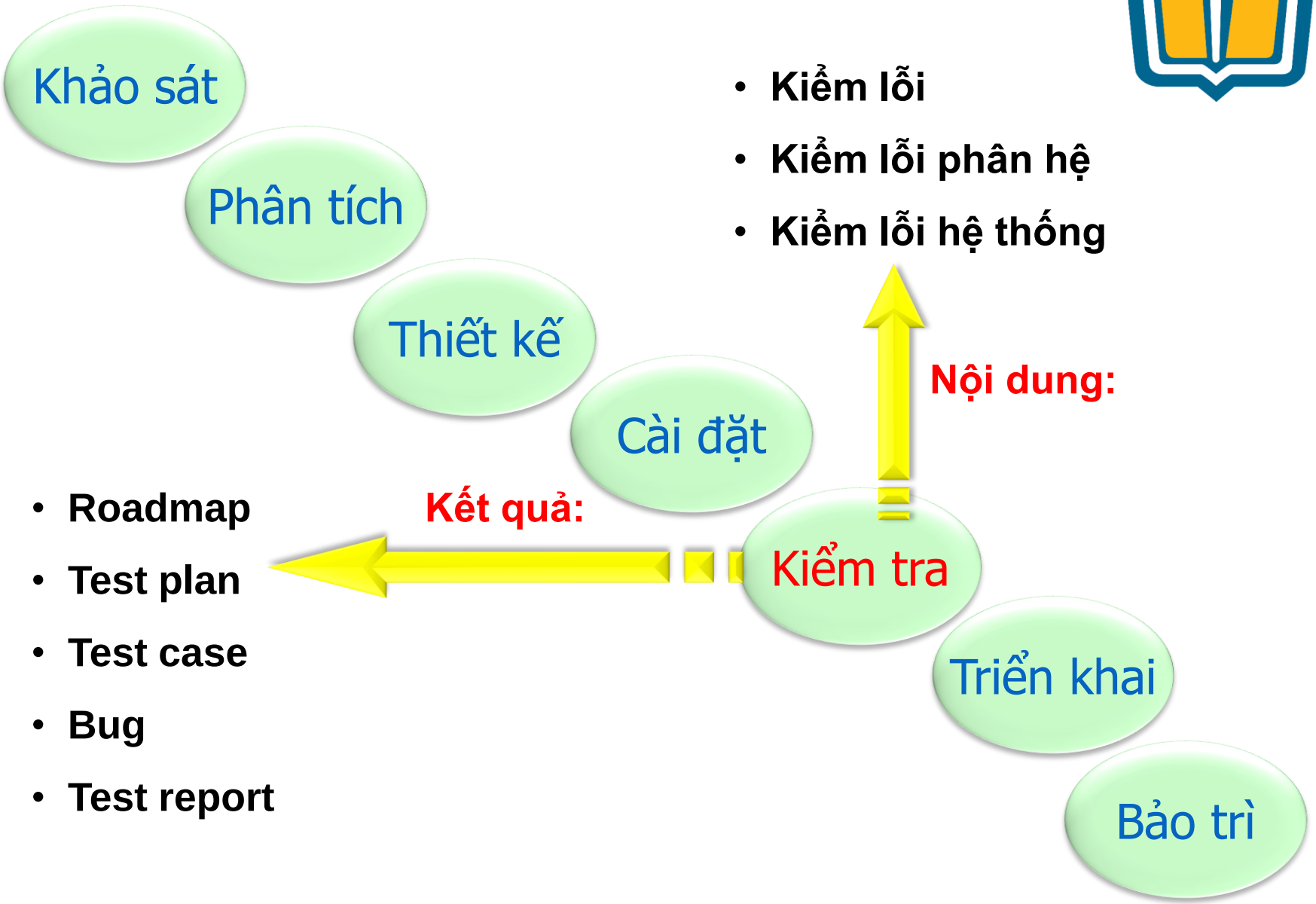
- Pressman, Kỹ nghệ phần mềm, chương 18~19.
- Sommerville: Software Engineering, chương 22~23.

References

Bài giảng này tham khảo từ các nguồn sau:

- Slide bài giảng CNPM, ***Trần Ngọc Bảo***, ĐH Sư phạm TpHCM.
- Slide bài giảng CNPM, ***Trần Anh Dũng***, ĐH CNTT, ĐHQG TpHCM.
- Slide bài giảng Kỹ nghệ Phần mềm, ***Nguyễn Văn Vy***, ĐH Công nghệ, ĐHQG Hà Nội.

Giai đoạn kiểm tra



- Kiểm lỗi
- Kiểm lỗi phân hệ
- Kiểm lỗi hệ thống

- Roadmap
- Test plan
- Test case
- Bug
- Test report

Mục tiêu

- Biết được quy trình kiểm thử phần mềm
- Biết được các khái niệm liên quan đến kiểm thử (testing)
- Biết được các bước kiểm thử
- Biết sử dụng một số công cụ hỗ trợ testing
- Biết viết sưu liệu kiểm thử

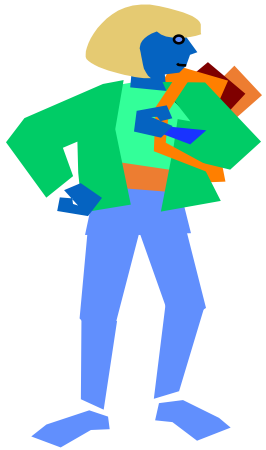
Nội dung

- Khái niệm kiểm thử phần mềm
- Một số đặc điểm của kiểm thử phần mềm
- Tại sao kiểm thử lại cần thiết?
- Qui trình kiểm thử
- Tổ chức và vai trò của các thành viên trong nhóm test
- Công cụ hỗ trợ test:
  - Công cụ theo dõi quá trình test
  - Công cụ hỗ trợ test tự động
- Sưu liệu kiểm thử: Test plan, test case, test log, test report,…

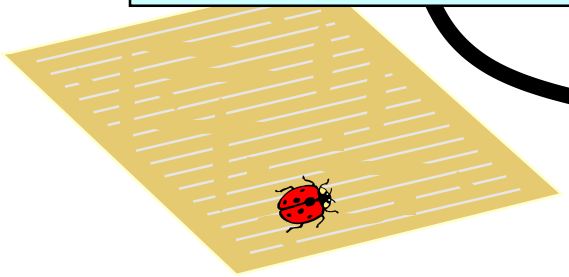
Khái niệm kiểm thử phần mềm

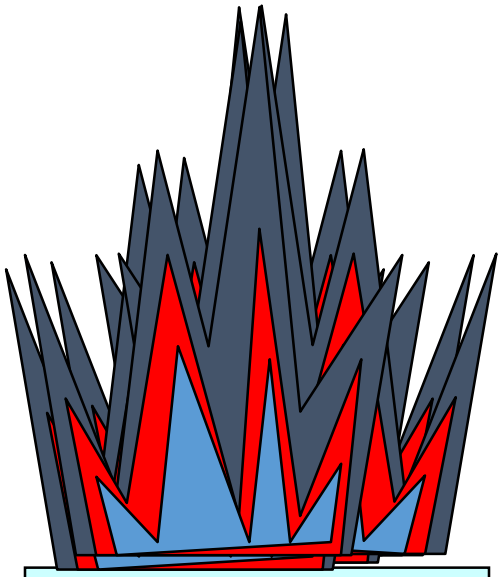
- Kiểm thử là gì?

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 phần mềm

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

Glen Myers, 1979

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

Hetzel, 1988

Một số đặc điểm kiểm thử PM

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

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

  Beizer
- Điều gì xảy ra nếu việc kiểm thử không tìm được lỗi trong phần mềm hoặc phát hiện quá ít lỗi
  - Phần mềm có chất lượng quá tốt
  - Quy trình/Đội ngũ kiểm thử hoạt động không hiệu quả

Tại sao kiểm thử lại cần thiết?

- Thông thường thì phần mềm 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 →Khách hàng có thể lãng tránh công ty với lý do công ty trông có vẻ không chuyên nghiệp.
  - Một phần mềm tính toán lượng thuốc trừ sâu dùng cho cây trồng, vì lý do tính sai số lượng lên gấp 10 lần →Nông dân phải bỏ nhiều tiền mua, cây trồng hư hại, môi trường sống, nguồn nước bị ảnh hưởng,….

Tại sao kiểm thử lại cần thiết?

- Kiểm thử phần mềm → chất lượng phần mềm được nâng cao.
- Chúng ta có thể đánh giá chất lượng phần mềm 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 phần mềm nếu có ít lỗi hoặc không có lỗi nào được tìm thấy. Nếu lỗi tìm thấy và được sửa thì chất lượng phần mềm càng được tăng ➔ Giảm chi phí trong quá trình phát triển, nâng cấp, bảo trì phần mềm.

Lỗi tăng lên khi nào?

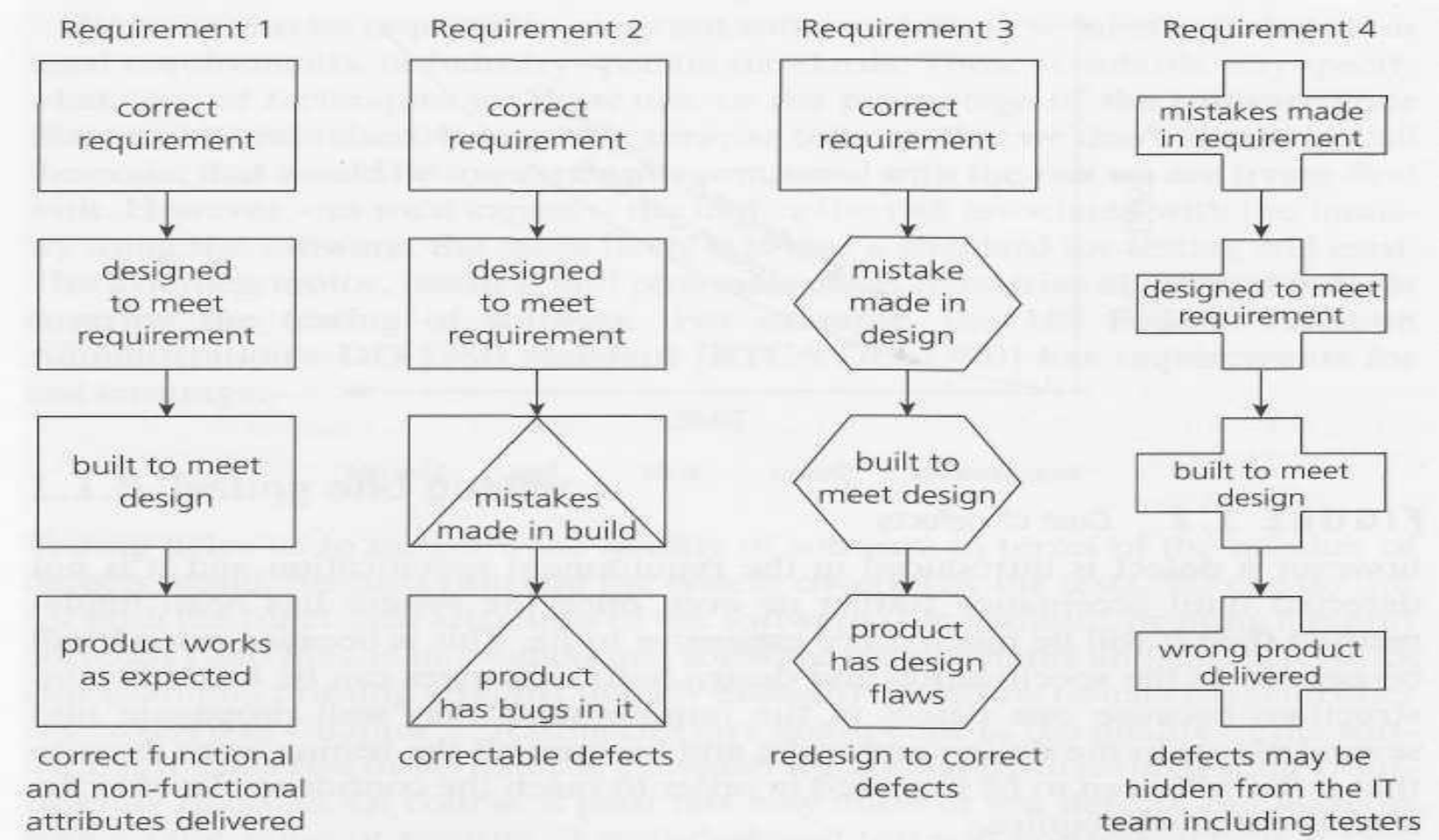
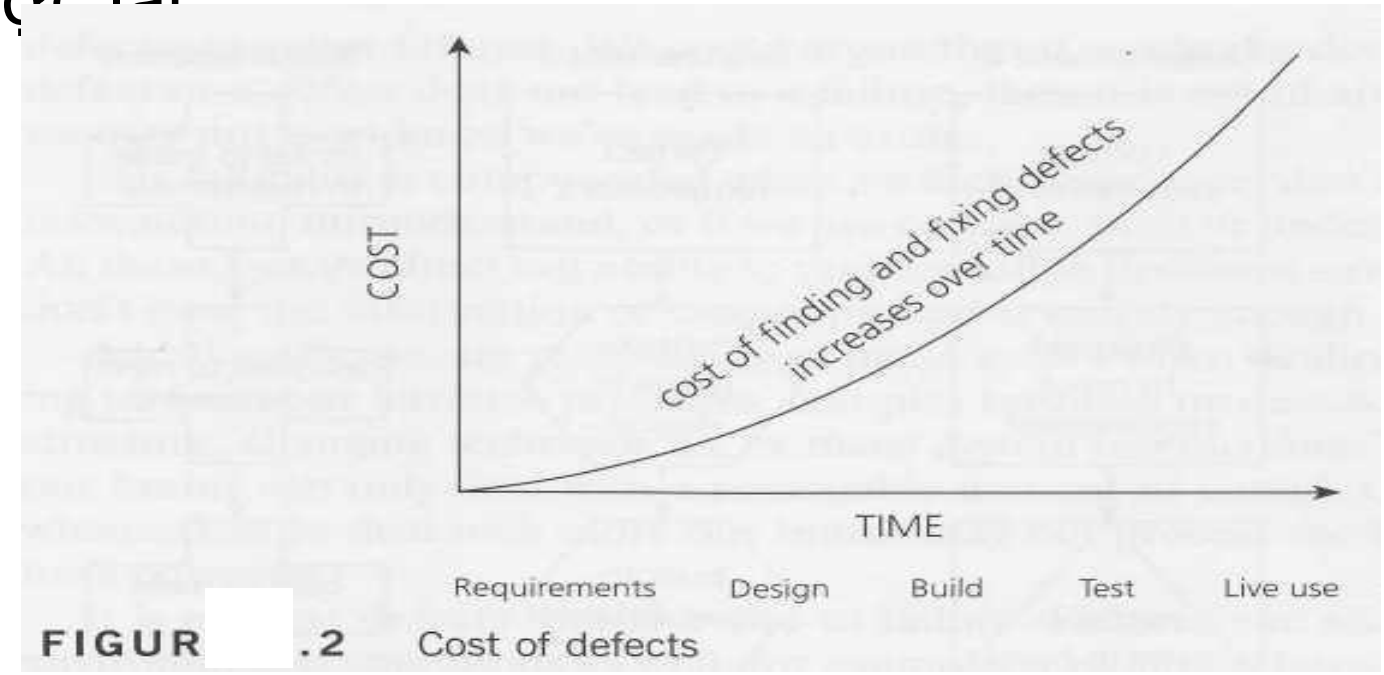


FIGURE Types of error and defect

Lỗi tăng lên khi nào?

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



FIGUR .2 Cost of defects

Thời điểm tiến hành kiểm thử

Tiến hành ở mọi công đoạn phát triển phần mềm

- **phân tích**
  - **- xét duyệt đặc tả yêu cầu**
- **thiết kế**
  - **- xét duyệt đặc tả thiết kế**
- **mã hóa**
  - **- kiểm thử chương trình**

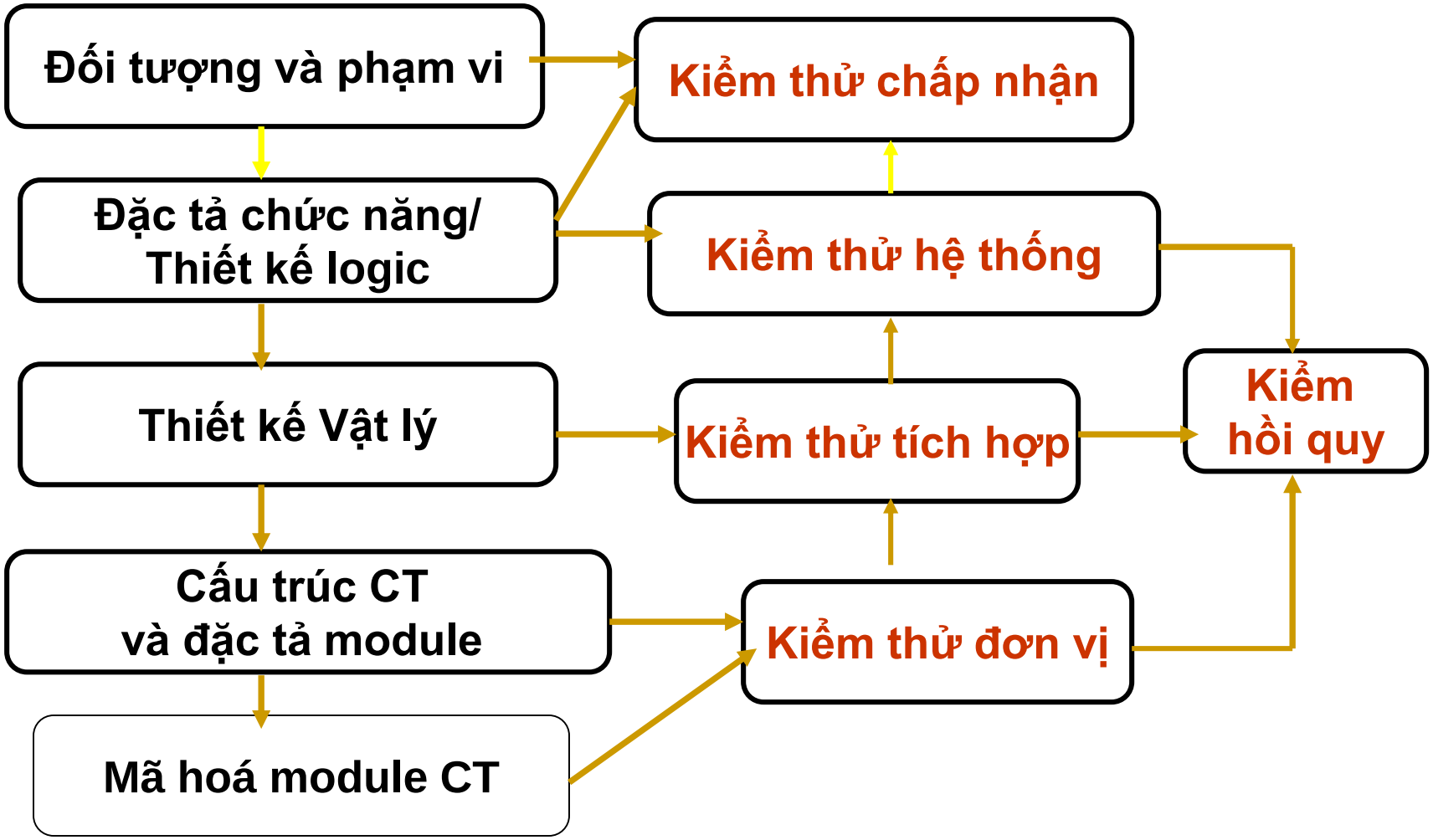
Mục đích của kiểm thử

- Mục tiêu: tìm lỗi
- Các loại lỗi:
  - Lỗi yêu cầu.
  - Lỗi thiết kế.
  - Lỗi cài đặt.

Yêu cầu đối với kiểm thử

- **Tính lặp lại**
  - kiểm thử phải lặp lại được (kiểm tra xem lỗi đã được sửa hay chưa)
  - dữ liệu/trạng thái phải mô tả được
- **Tính hệ thống**
  - đảm bảo kiểm tra hết các trường hợp
- **Được lập tài liệu**
  - kiểm soát tiến trình/kết quả

Vòng đời dự án và Kiểm thử



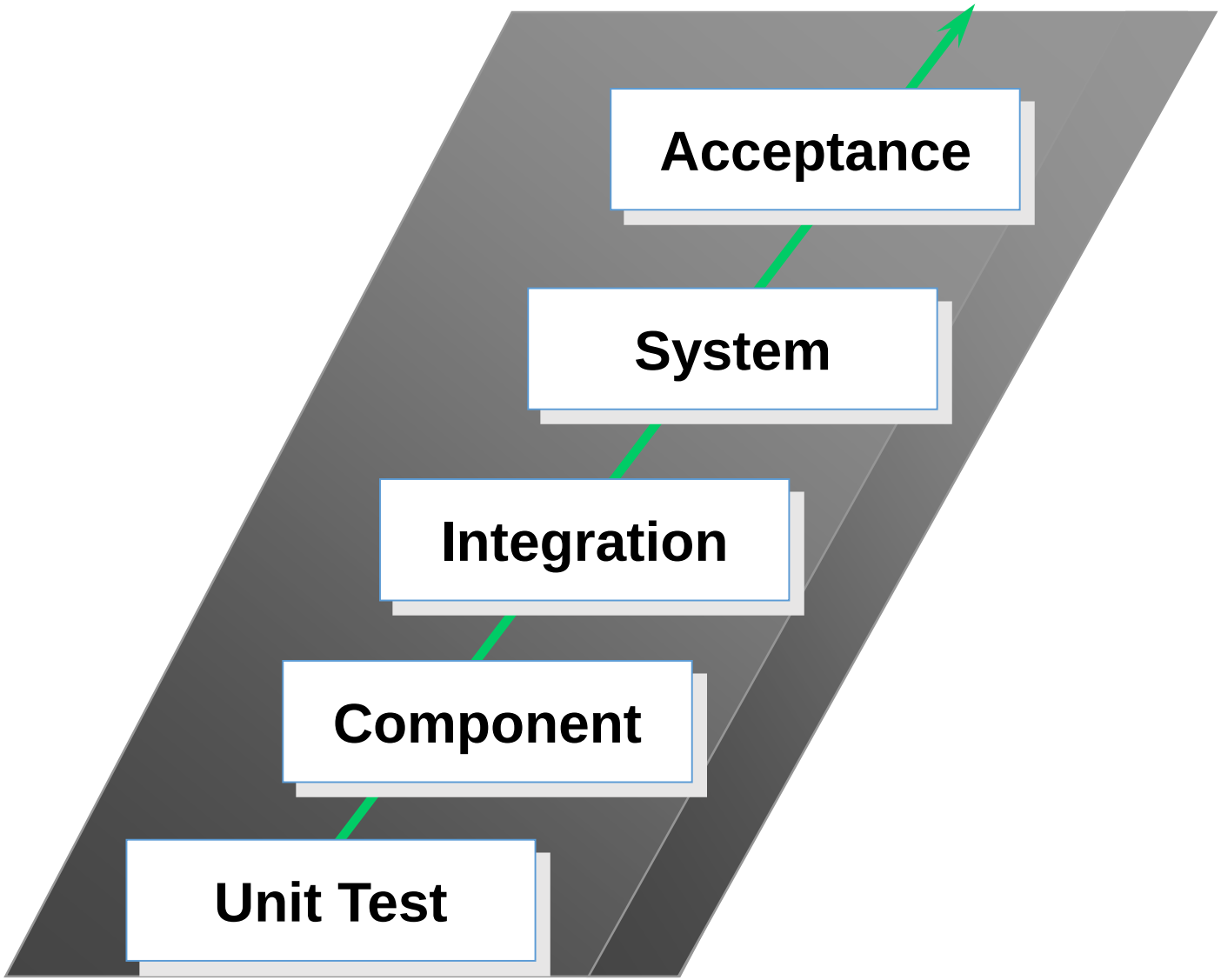
Các loại kiểm thử

1. **Developer test** – kiểm thử trong quá trình phát triển
   a) **Unit test** – kiểm thử đơn vị: test lớp, phương thức
   b) **Component test** – kiểm thử thành phần: nhóm các lớp
   c) **System test** – kiểm thử hệ thống: tích hợp các thành phần
2. **Release test** – kiểm thử để chuẩn bị phát hành

   Validation test + Defect testing
   a) **Scenario based testing** – kiểm thử theo kịch bản (use case)
   b) **Performance testing** – kiểm thử hiệu năng
3. **User test**

   • **Acceptance test** – kiểm thử chấp nhận
      • Thẩm định xem có đúng yêu cầu
      • Tìm khiếm khuyết

Các mức độ kiểm thử (Test levels)



Các mức độ kiểm thử (Test levels)

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

Các mức độ kiểm thử (Test levels)

- Integration testing:
  - Test sự kết hợp của các component, sự tác động của các phần khác nhau trong một hệ thống, sự kết hợp của các hệ thống với nhau,…

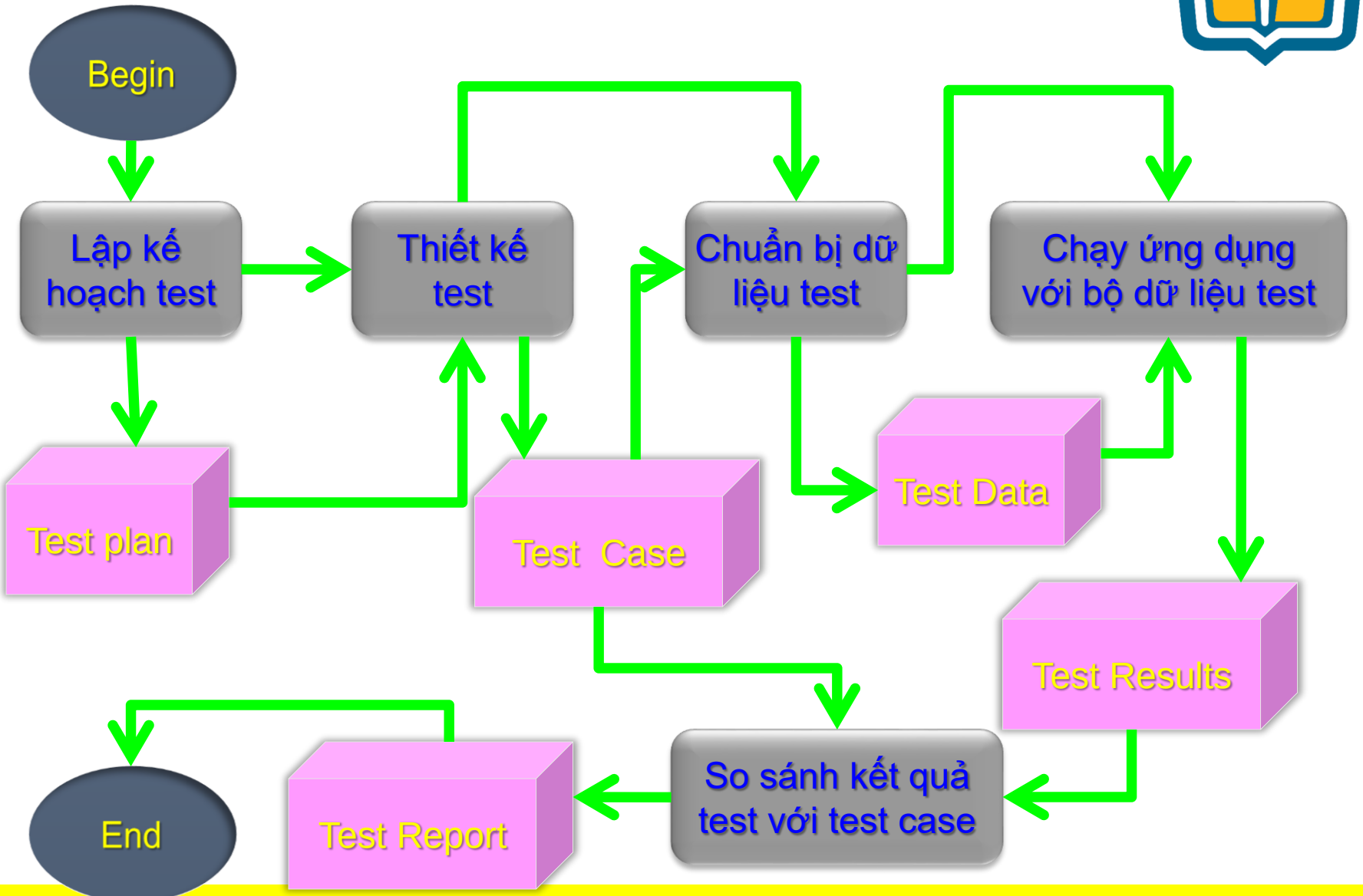
Các mức độ kiểm thử (Test levels)

- System testing:
  - Đảm bảo rằng hệ thống (sau khi tích hợp) thỏa mãn tất cả các yêu cầu của người sử dụng
  - Tập trung vào việc phát hiện các lỗi xảy ra trên toàn hệ thống
- Acceptance testing:
  - Test phần mềm đứng dưới góc độ người dùng để xác định phần mềm có được chấp nhận hay không.

Qui trình kiểm thử

- Kiểm thử thành phần
  - Kiểm thử của các từng thành phần chương trình;
  - Thường là trách nhiệm của lập trình viên tạo ra thành phần đó;
  - Các test được tạo ra từ kinh nghiệm của lập trình viên.
- Kiểm thử hệ thống
  - Kiểm thử một nhóm các thành phần được kết hợp lại để tại ra hệ thống hay hệ thống con;
  - Trách nhiệm của một đội test độc lập;
  - Các test được tạo ra dựa trên bản đặc tả hệ thống.

Qui trình kiểm thử phần mềm

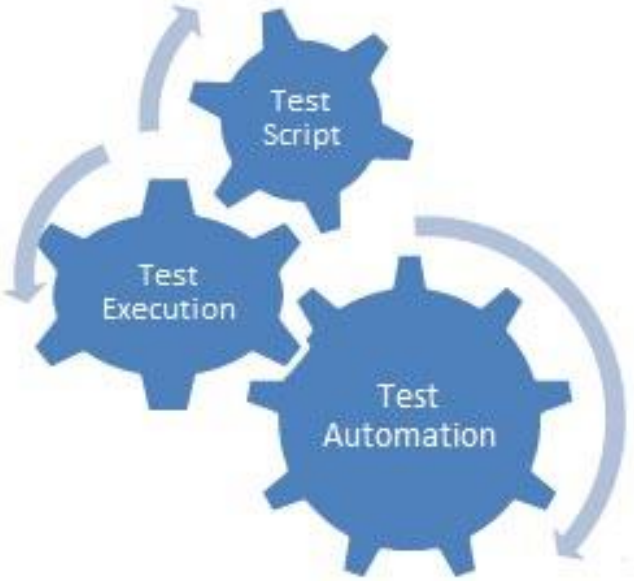


Kiểm thử thủ công (Manual Testing)

- Tester làm mọi việc hoàn toàn bằng tay, từ viết test case đến thực hiện test, mọi thao tác như nhập điều kiện đầu vào, thực hiện một số sự kiện khác như click nút và quan sát kết quả thực tế, sau đó so sánh kết quả thực tế với kết quả mong muốn trong test case, điền kết quả test.
- Kiểm thử thủ công là chủ yếu.

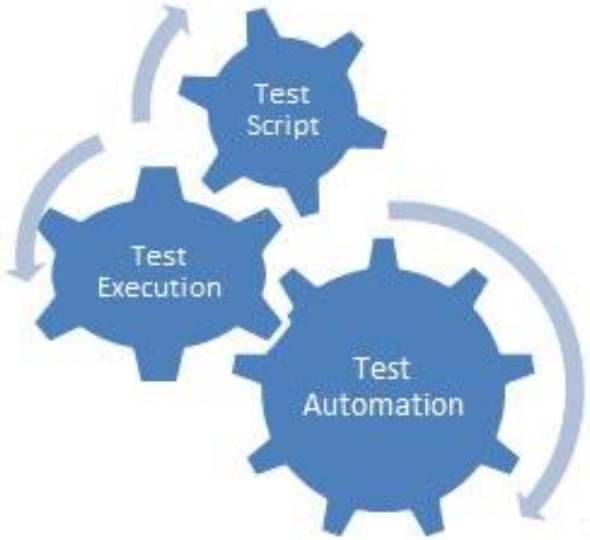
Kiểm thử tự động (Automation Testing)

- Sử dụng phần mềm chuyên biệt tiến hành kiểm thử (QTP, Selenium, …).
- Hạn chế sự tương tác của người sử dụng, giúp cho (tester) không phải lặp đi lặp lại các bước nhàm chán.



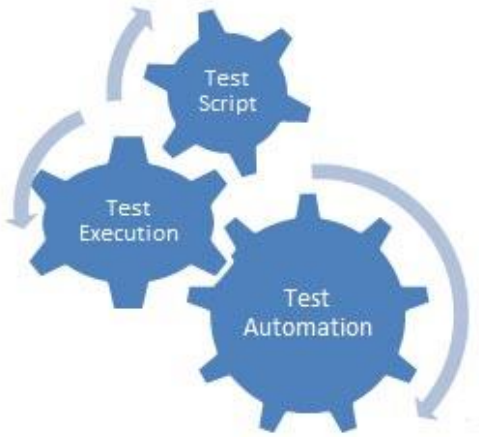
Kiểm thử tự động (Automation Testing)

- Công cụ kiểm thử tự động có thể lấy dữ liệu từ file bên ngoài (excel, csv…) nhập vào ứng dụng, so sánh kết quả mong đợi (từ file excel, csv…) với kết quả thực tế và xuất ra báo cáo kết quả kiểm thử.



Kiểm thử tự động (Automation Testing)

- Sử dụng khi nào?
  - Khi thực thi một số lượng test case quá lớn trong một thời gian ngắn.
  - Khi số lượng đầu vào cho một test case quá nhiều.
  - Khi muốn thực thi performance test hoặc load test.



So sánh

	Manual Testing	Automation Testing
Điểm mạnh	• Cho phép tester thực hiện việc kiểm thử khám phá. • Thích hợp kiểm tra sản phẩm lần đầu tiên. • Thích hợp kiểm thử trong trường hợp các test case chỉ phải thực hiện một số ít lần. • Giảm được chi phí ngắn hạn	• Thích hợp với test nhiều lần cho một case, có tính ổn định và tin cậy cao hơn so với kiểm thử thủ công. • Có thể thực hiện các thao tác lặp đi lặp lại (nhập dữ liệu, click, check kết quả...) giúp tester không phải làm những việc gây nhàm chán và dễ nhầm lẫn. • Giảm chi phí đầu tư dài hạn
Điểm yếu	• Tốn thời gian. Đối với mỗi lần release, người kiểm thử vẫn phải thực hiện lại một tập hợp các test case đã chạy dẫn đến sự mệt mỏi và lãng phí effort	• Tốn kém hơn kiểm thử thủ công, chi phí đầu tư ban đầu lớn. • Kiểm thử thủ công là không thể thay thế vì người ta không thể tự động hóa mọi thứ

Nguyên tắc kiểm thử

- Chọn các input làm cho hệ thống tạo ra tất cả các thông báo lỗi;
- Thiết kế input làm tràn bộ đệm;
- Lặp lại cùng một input hay một dãy các input một vài lần;
- Ép các output không hợp lệ phải xuất hiện;
- Ép các kết quả tính toán phải hoặc là quá lớn hoặc là quá nhỏ.

Các nguyên lý trong kiểm thử PM

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

Chính sách kiểm thử (Testing Policy)

- Kiểm tra tất cả các chức năng trong hệ thống menu.
- Kiểm tra tất cả các mục khác có cùng chức năng trong hệ thống menu (Toolbar, Listbar, Dialog bar, Context Menu,…)
- Kiểm tra cùng một chức năng với nhiều vai trò khác nhau (đối với hệ thống có nhiều người dùng)
- Kiểm tra tất cả các dữ liệu bắt buộc nhập trong các màn hình (hợp lệ/không hợp lệ)

Một số khái niệm cơ bản

- Test plan
- Test case
- Bug
- Test report
- Test Manager
- Test Designer
- Tester

Test plan

- Cấu trúc chung của một test plan
  - Tên project
  - Danh sách các Module cần test
  - Ngày bắt đầu, ngày kết thúc
  - Danh sách các Test case
  - Nhân sự tham gia
  - Tài nguyên sử dụng (Servers, Workstations, Printers,…)
  - Kế hoạch thực hiện (sử dụng Ms Project lập kế hoạch)
  - …

Giai đoạn kiểm thử

- Roadmap
- Test plan
- Test case
- Bug
- Test Report

Test case

- Ca kiểm thử: dữ liệu để kiểm tra hoạt động của chương trình
- Ca kiểm thử tốt:
  - được thiết kế để phát hiện một lỗi của chương trình
- Kiểm thử thành công: phát hiện ra lỗi
- Mục đích:
  - Chứng minh được sự tồn tại của lỗi
  - Không chứng minh được sự không có lỗi

Test case

- Cấu trúc chung của một test case:
  - Tên project, module
  - Màn hình, chức năng
  - Mã số
  - Tài liệu tham khảo (SRS)
  - Mục đích
  - Dữ liệu test
  - Mô tả các bước (Test step)
  - Trạng thái
  - Ngày tạo
  - …

Test case

- Ví dụ: kiểm tra màn hình đăng nhập

Test case

- Ví dụ: kiểm tra màn hình đăng nhập
  - Project: Web testing application
  - Module: Testing
  - Màn hình: Đăng nhập hệ thống
  - Chức năng: đăng nhập
  - Mã số: TC001
  - Dữ liệu test
    - Username = “hienlth”, pass = “123456”
    - Username = “admin”, pass = “admin”
  - Các bước thực hiện kiểm tra

Test case – Test step

Step no	Steps	Data	Expected result	Actual results
1	Nhập **Username** và ấn nút **OK**	Username = "hienlth"	Hiển thị thông báo "Vui lòng nhập username và password"	
2	Nhập **Password** và ấn nút **OK**	Password = "123456"	Hiển thị thông báo "Vui lòng nhập username và password"	
3	Nhập **Username , password** và ấn nút **OK**	Username = "hienlth" Password = "abc"	Hiển thị thông báo "Username và password không hợp lệ"	
4	Nhập **Username , password** và ấn nút **OK**	Username = "abc" Password = "hienlth"	Hiển thị thông báo "Username và password không hợp lệ"	
5	Nhập **Username, password** và ấn nút **OK**	Username = "abc" Password = "abc"	Hiển thị thông báo "Username và password không hợp lệ"	
6	Nhập **Username, password** và ấn nút **OK**	Username = "" Password = ""	Hiển thị thông báo "Username và password không hợp lệ"	
7	Nhập **Username, password** và ấn nút **OK**	Username = "hienlth" Password = "123456"	Hiển thị trang chính của user "**hienlth**"	
8	Nhập **Username, password** và ấn nút **OK**	Username = "admin" Password = "admin"	Hiển thị trang chính của user "**admin**"	
...				

Giai đoạn kiểm thử

- Roadmap
- Test plan
- Test case
- Bug
- Test Report

Bug

- Cấu trúc chung của Bug:
  - Tên bug
  - Mã số, mức độ
  - Test case tương ứng (nếu có)
  - Màn hình, chức năng
  - Dữ liệu
  - Mô tả các bước thực hiện
  - Hình chụp màn hình/quay phim các thao tác.
  - Trạng thái
  - Ngày tạo
  - …

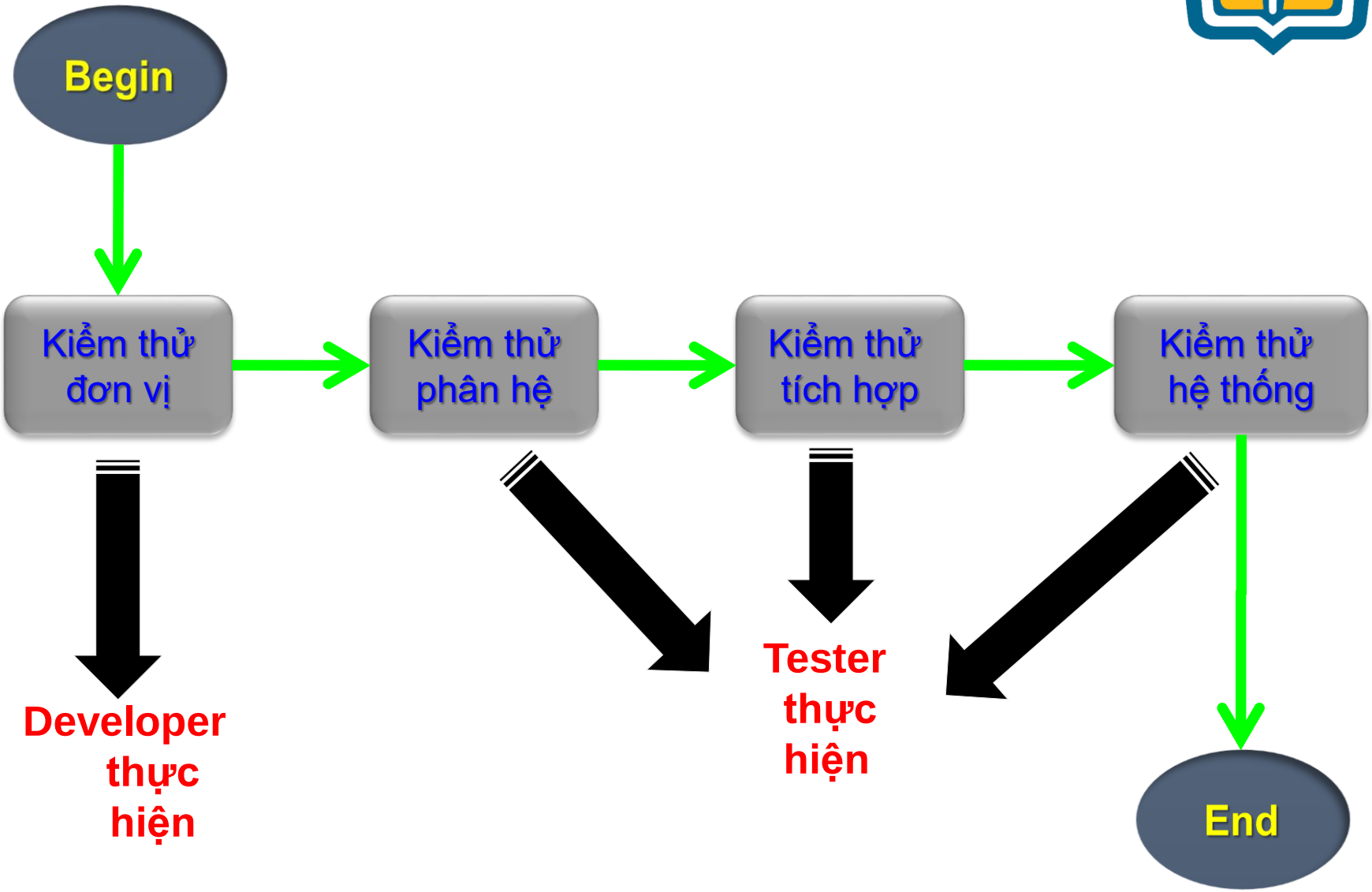
Giai đoạn kiểm thử

- Roadmap
- Test plan
- Test case
- Bug
- Test Report

Test report

- Cấu trúc chung của Test report:
  - Test plan ?
  - Tên người thực hiện
  - Ngày thực hiện
  - Môi trường test
  - Bảng mô tả module/chức năng/test case và kết quả tương ứng
  - Kết luận, đề xuất (nếu có)
  - ….

Chiến lược kiểm thử

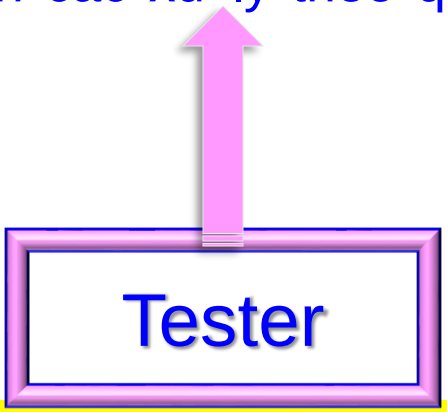


Phân loại kiểm thử (Testing type)

- White-box testing (Strategy)
  - Component or Unit Testing
  - Object class testing
- Black-box testing (Strategy)
  - Functional testing
  - Interface testing
  - Ad hoc testing
  - Performance testing
  - Stress testing
  - Alpha testing
  - Beta testing
  - Release testing, ….

White-Box testing

- Phần mềm là một hệ thống gồm 3 thành phần cơ bản: thành phần lưu trữ, thành phần giao tiếp, thành phần xử lý cần phải thực hiện theo yêu cầu của người dùng.
- Thành phần giao tiếp: giao diện chương trình
- Thành phần lưu trữ: cho phép lưu trữ và truy xuất dữ liệu.
- Thành phần xử lý: thực hiện các xử lý theo qui trình nghiệp vụ của người dùng



White-box testing

- Kiểm tra tính logic và cấu trúc của mã nguồn (source code): bao gồm server code và client code
- Tester cần phải có kiến thức về ngôn ngữ lập trình (C, C++, VB.NET, Java,…), môi trường phát triển phần mềm (IDE), cũng như các hệ quản trị cơ sở dữ liệu (SQL Server, Oracle, DB2,…), …
- Kiểm tra tất cả các trường hợp có thể xảy ra trong mã nguồn (cấu trúc điều khiển, cấu trúc lặp,…)

White-box testing

- Ví du: cho đoan mã C/C++ như sau:

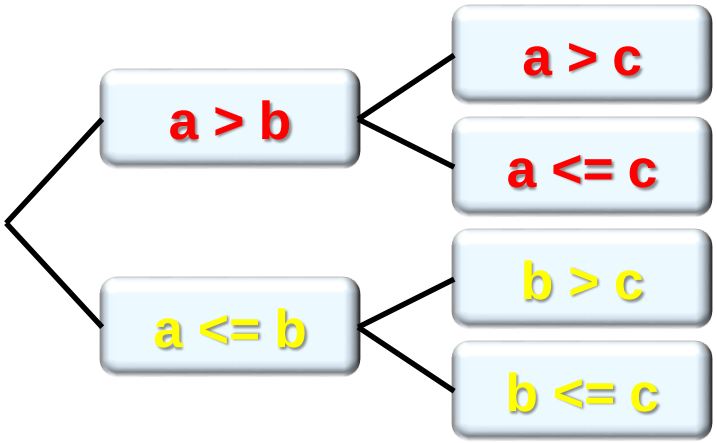
```
int  Test(int a, int b, int c) {
  if (a>b) {
     if (a>c) return a;
     else return c; }
  else  {
     if (b>c) return b;
     else return c; }
}
```

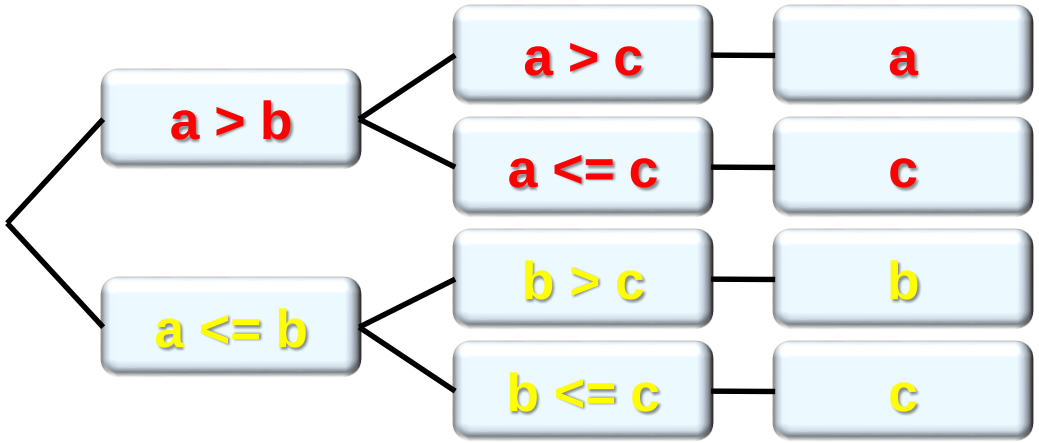
Để kiểm tra tính đúng đắn của đoạn code trên chúng ta cần ít nhất bao nhiêu trường hợp ?

White-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; }
}
```





White-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 đoạn code trên chúng ta cần ít nhất 4 trường hợp (Test case), ví dụ:

- **a = 4, b = 2, c = 3**
- **a = 4, b = 2, c = 5**
- **a = 3, b = 4, c = 2**
- **a = 3, b = 4, c = 6**

Component testing

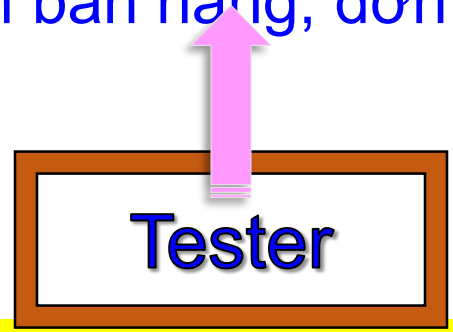
- Kiểm thử thành phần hay kiểm thử đơn vị là quá trình kiểm thử các thành phần một cách đơn lẻ và cô lập.
- Là một loại kiểm thử thiếu sót.
- Thành phần có thể là:
  - Các hàm hay phương thức đơn lẻ trong một đối tượng;
  - Các lớp đối tượng;
  - Các thành phần kết hợp với giao diện được định trước để truy xuất đến các chức năng của chúng.

Kiểm thử lớp đối tượng

- Một test hoàn chỉnh cho một lớp bao gồm
  - Kiểm thử tất cả các phương thức liên kết với một đối tượng;
  - Thiết lập và interrogating tất cả thuộc tính của đối tượng;
  - Thực hành đối tượng tại tất cả trạng thái có thể.
- Tính kế thừa làm cho việc kiểm thử lớp đối tượng khó hơn bởi vì thông tin được kiểm thử không có tính cục bộ.

Black-Box testing

- Hệ thống phần mềm là một công cụ hỗ trợ để thực hiện các công việc chuyên môn của người sử dụng trên máy tính..
- Phầm mềm quản lý giáo vụ trường phổ thông hỗ trợ các nghiệp vụ: quản lý hồ sơ học sinh, kết quả học tập, tính điểm trung bình, …
- Phần mềm quản lý bán hàng hỗ trợ các nghiệp vụ: lập chứng từ hóa đơn bán hàng, đơn đặt hàng, tính doanh thu, in báo cáo..



Black-Box testing

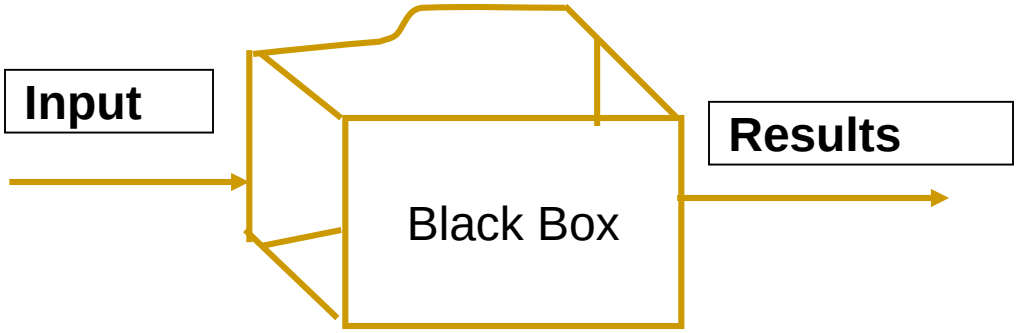
- Kiểm tra hệ thống dựa trên bản đặc tả yêu cầu và chức năng
- Tester không cần phải có kiến thức về ngôn ngữ lập trình, môi trường phát triển phần mềm (IDE), cũng như các hệ quản trị cơ sở dữ liệu (SQL Server, Oracle, DB2,…), …
- Trong trường hợp này, tester thao tác các chức năng của hệ thống như là một người sử dụng hệ thống (end-user).

Black-Box testing

- Dựa vào chức năng nhằm phát hiện lỗi:

  1) Thiếu chức năng 2) Lỗi giao diện

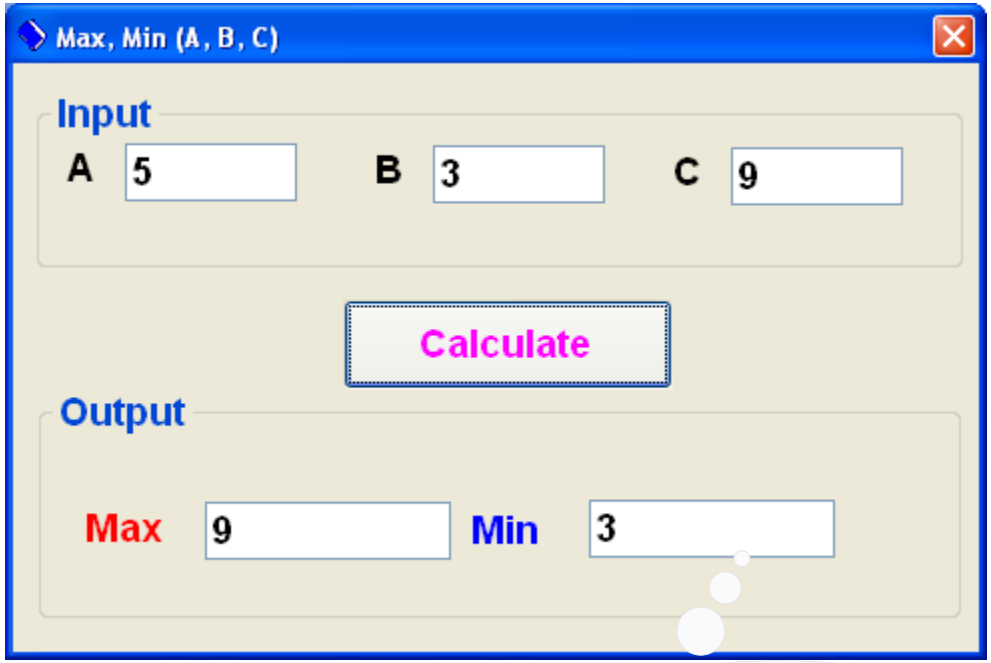
  3) Lỗi trong CTDL 4) Lỗi khi thực hiện



Black box Data Testing Strategy

Black-Box testing

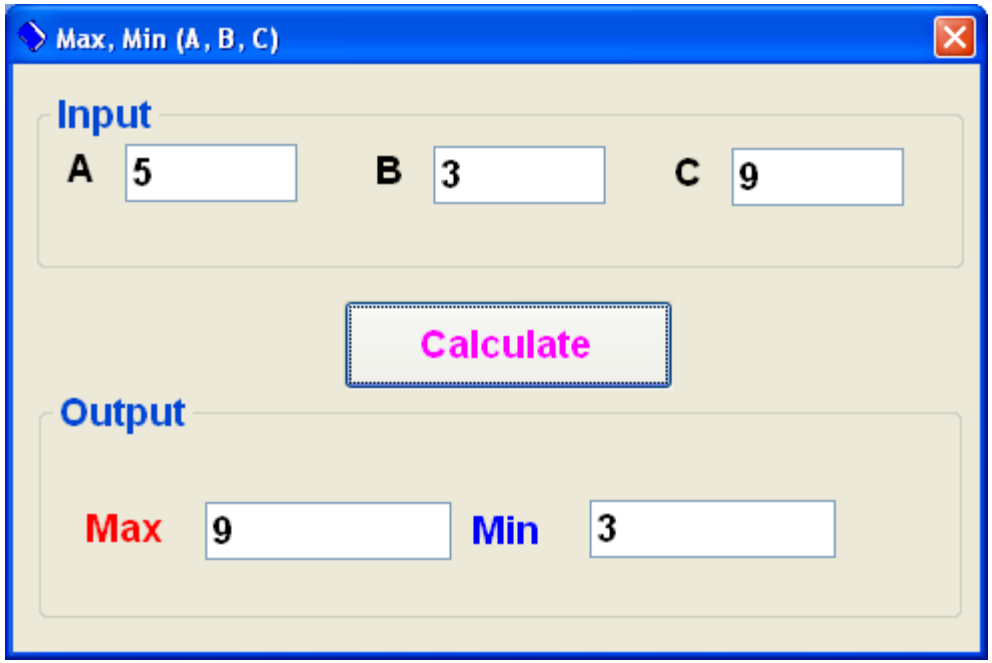
- Ví dụ: Kiểm tra màn hình sau

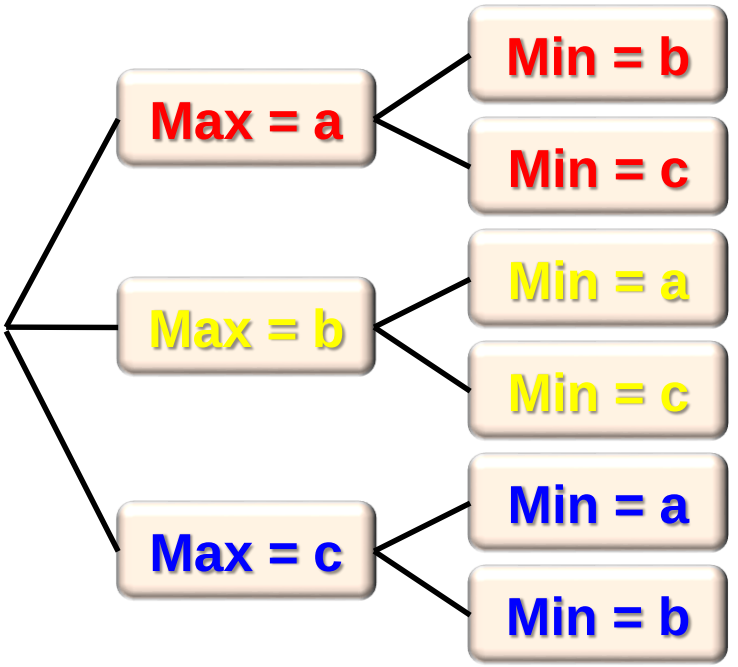


Để kiểm tra tính đúng đắn của đoạn code trên chúng ta cần ít nhất bao nhiêu trường hợp ?

Black-Box testing

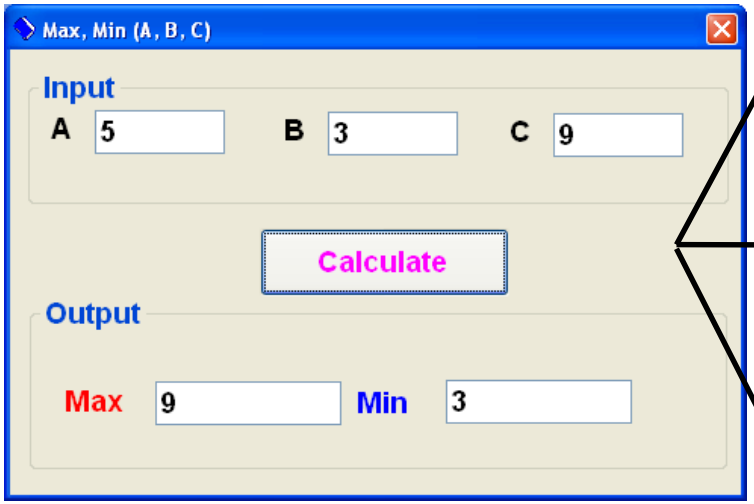
- Ví dụ: Kiểm tra màn hình sau

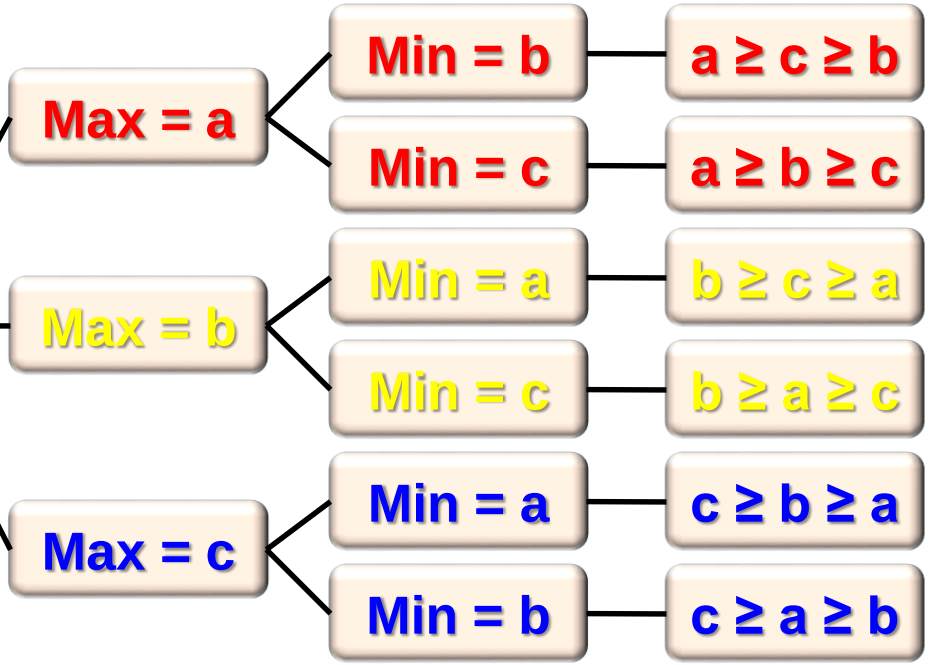




Black-Box testing

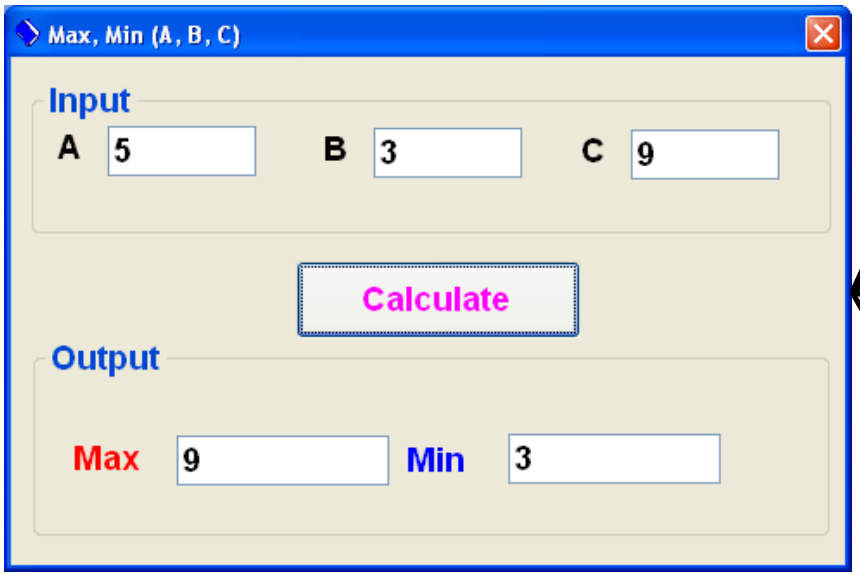
- Ví dụ: Kiểm tra màn hình sau

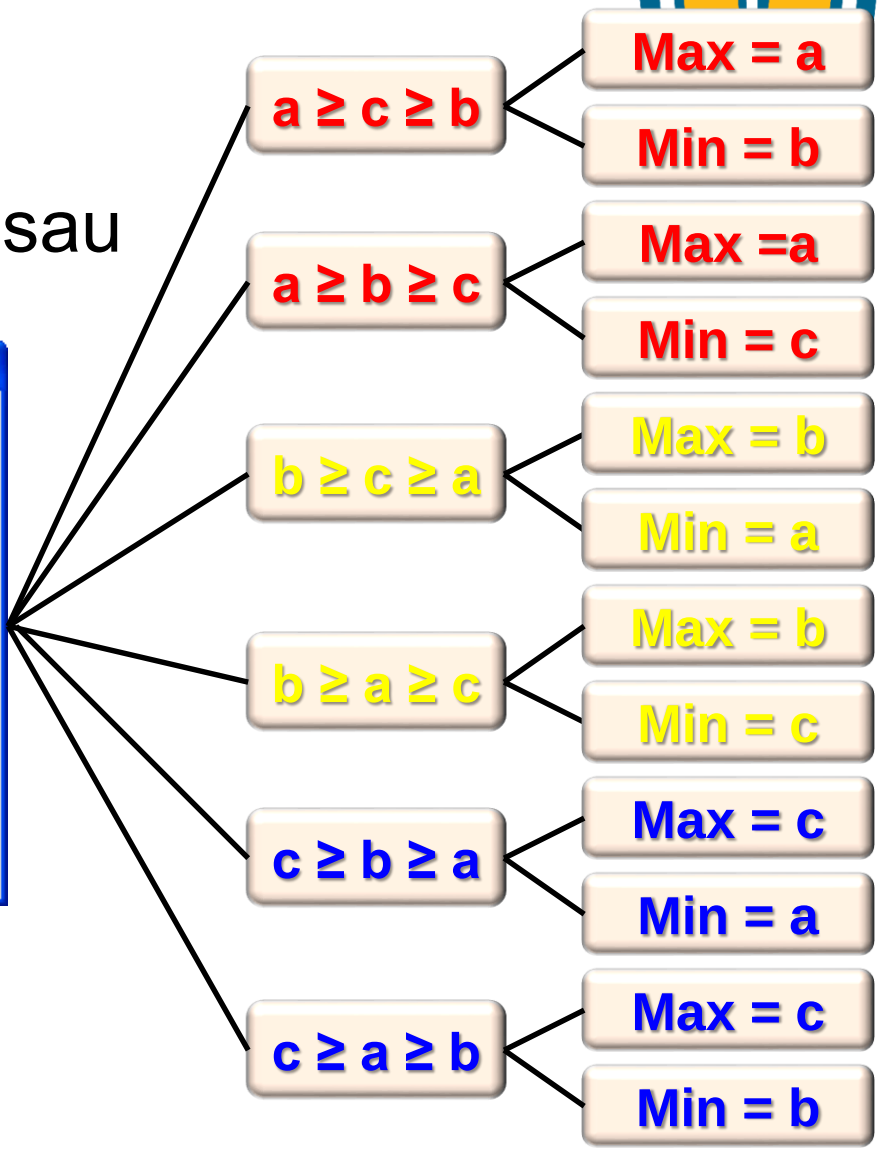




Black-Box testing

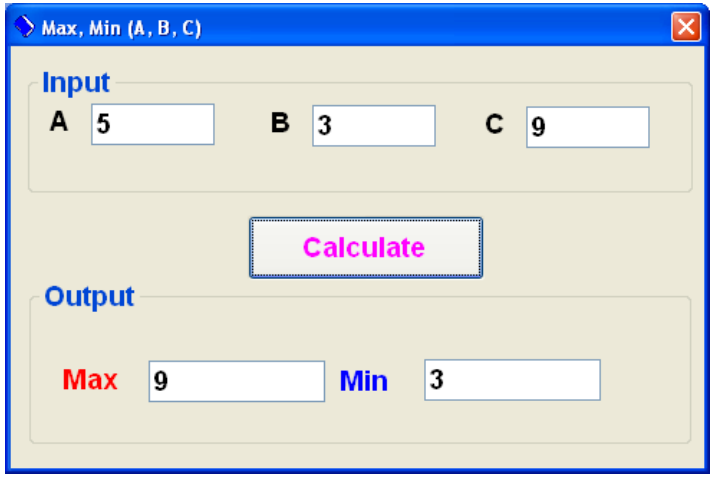
- Ví dụ: Kiểm tra màn hình sau





Black-Box testing

- Ví dụ: Kiểm tra màn hình sau



Để kiểm tra màn hình trên chúng ta cần ít nhất 6 trường hợp (Test case), ví dụ:

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

Interface testing

- Mục đích là phát hiện ra lỗi do lỗi giao diện hay những giả sử không hợp lý về giao diện.
- Đặc biệt quan trọng cho phát triển hướng đối tượng khi các đối tượng được định nghĩa bởi các giao diện.

Các loại giao diện

- Giao diện tham số
  - Dữ liệu chuyển từ một thủ tục sang một thủ tục khác.
- Giao diện chia sẻ bộ nhớ
  - Vùng nhớ được chia sẻ giữa các thủ tục hay hàm.
- Giao diện thủ tục
  - Hệ thống con đóng gói một tập các thủ tục được gọi bởi các hệ thống con khác.
- Giao diện truyền thông điệp
  - Các hệ thống con yêu cầu dịch vụ từ các hệ thống con khác

Lỗi giao diện

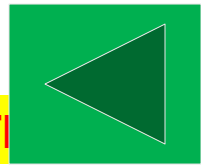
- Sử dụng nhầm giao diện
  - Một thành phần gọi một thành phần khác và tạo ra một lỗi trong quá trình sử dụng giao diện của nó, ví dụ tham số không đúng thứ tự.
- Hiểu nhầm giao diện
  - Một thành phần ngầm định về hành vi của một thành phần được gọi khác nhưng ngầm định đó không đúng.
- Lỗi về thời gian
  - Thành phần gọi và được gọi hoạt động với tốc độ khác nhau và dẫn đến truy cập đến thông tin không đúng (chậm cập nhật).

Nguyên tắc kiểm thử giao diện

- Thiết kế test sao cho tham số ở những giới hạn cuối của phạm vi của nó.
- Luôn kiểm thử tham số con trỏ với con trỏ rỗng (null).
- Thiết kế test làm cho thành phần thất bại.
- Dùng stress testing trong hệ truyền thông điệp.
- Trong hệ thống chia sẻ vùng nhớ, làm đa dạng thưc tứ các thành phần hoạt động.

Stress testing

- Cho hệ thống hoạt động trong môi trường vượt quá khả năng tải tối đa của nó. Thường sẽ bộc lộ các thiếu sót của hệ thống.
- Nhằm kiểm thử các hành vi thất bại. Hệ thống không nên rơi vào một ngữ cảnh thất bại “thảm họa”.
- Thích hợp cho các hệ phân tán.



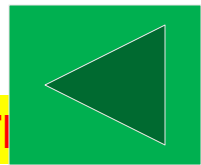
Kiểm thử Alpha, Beta

❑ *Kiểm thử alpha*

Là kiểm thử chấp nhận được tiến hành ở môi trường khách hàng.

❑ *Kiểm thử beta*

- **Mở rộng của alpha testing**
- **Được tiến hành với một lượng lớn users**
- **User tiến hành kiểm thử không có sự hướng dẫn của người phát triển**
- **Thông báo lại kết quả cho người phát triển**



Release testing

- Quá trình kiểm thử một release của một hệ thống sẽ được phân phối đến cho khách hàng.
- Mục đích chính là tăng niềm tin của nhà cung cấp trong việc hệ thống đáp ứng được các yêu cầu của nó.
- Release testing thường là black-box hay là kiểm thử chức năng
  - Chỉ dựa trên đặc tả hệ thống;
  - Chuyên viên kiểm thử không cần phải có kiến thức về cài đặt hệ thống.

Một số kỹ thuật test

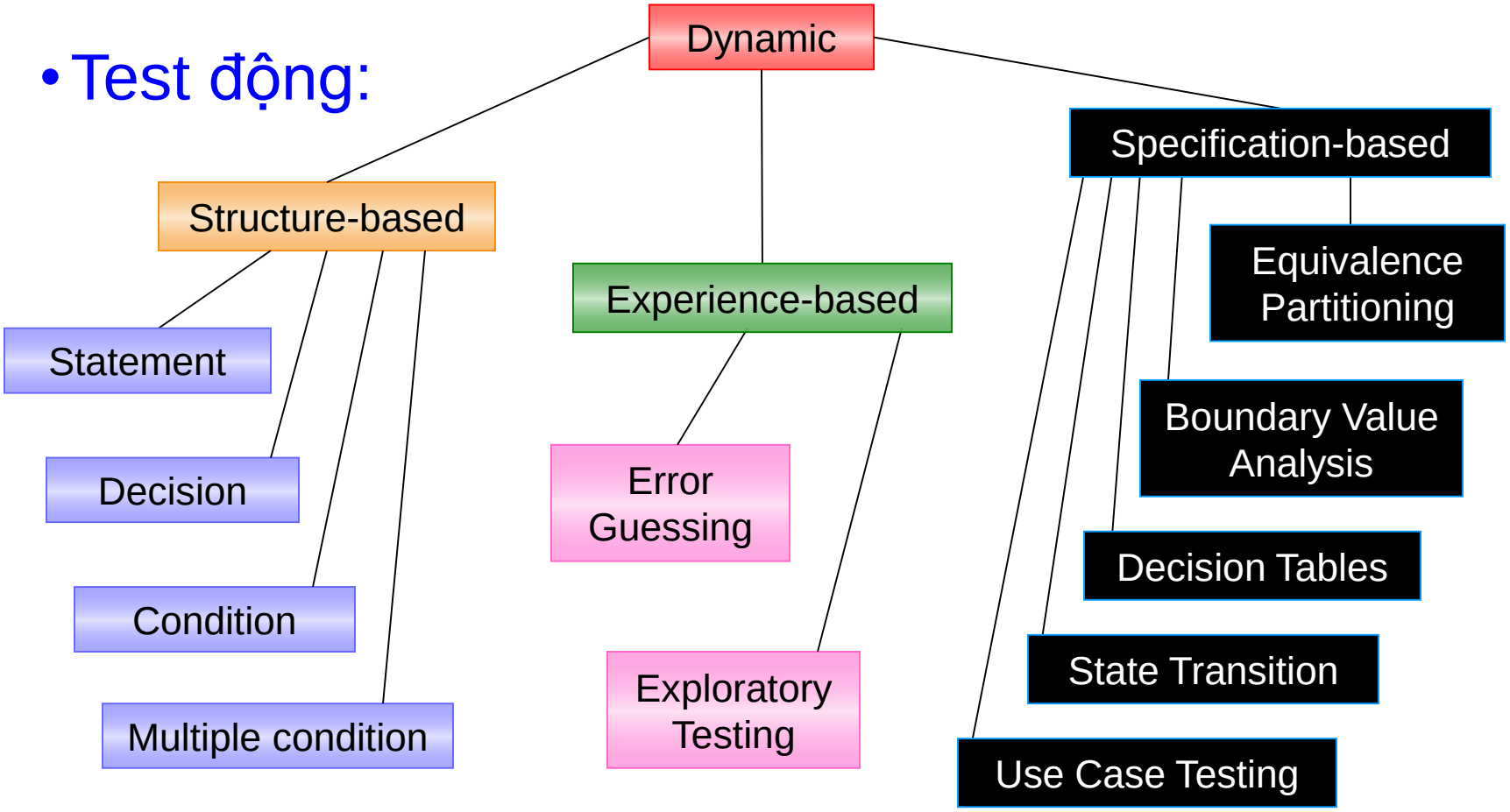
- Test tĩnh **(Static Verification)**
  - Dựa vào việc kiểm tra tài liệu, source code,… mà không cần phải thực thi phần mềm.
  - Các lỗi được tìm thấy trong quá trình kiểm tra có thể dễ dàng được loại bỏ và chi phí rẻ hơn nhiều so với khi tìm thấy trong test động. Một số lợi ích khi thực hiện việc kiểm 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:



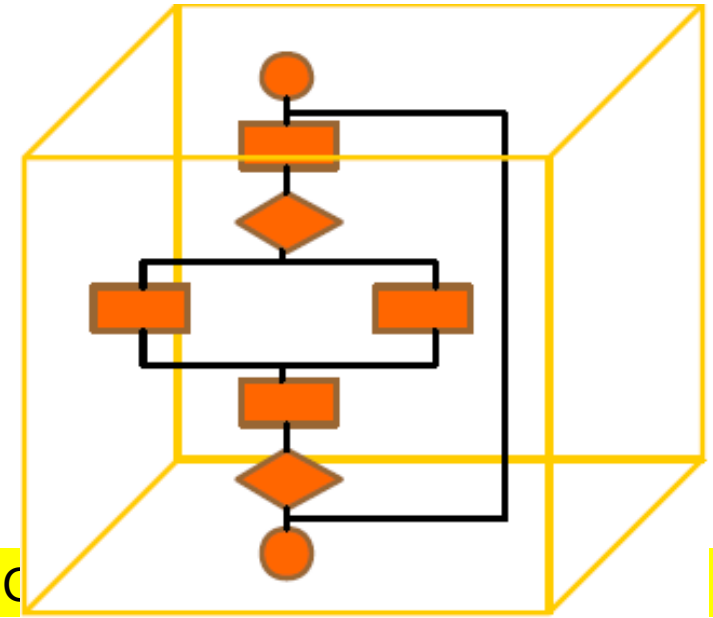
Một số kỹ thuật test

- Test động:
  - Test dựa trên mô tả (specification-based) hay còn gọi test chức năng (functional testing): Test những gì mà phần mềm phải làm, không cần biết phần mềm 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 phần mềm 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

Kỹ thuật Thiết kế Test case White Box Testing

Design White Box Testing (2)

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



Design White Box Testing (1)

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

Các kỹ thuật kiểm thử hộp trắng

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

Design White Box Testing (3)

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

Kỹ thuật Thiết kế Test case Black Box Testing

Các kỹ thuật kiểm thử hộp đen

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

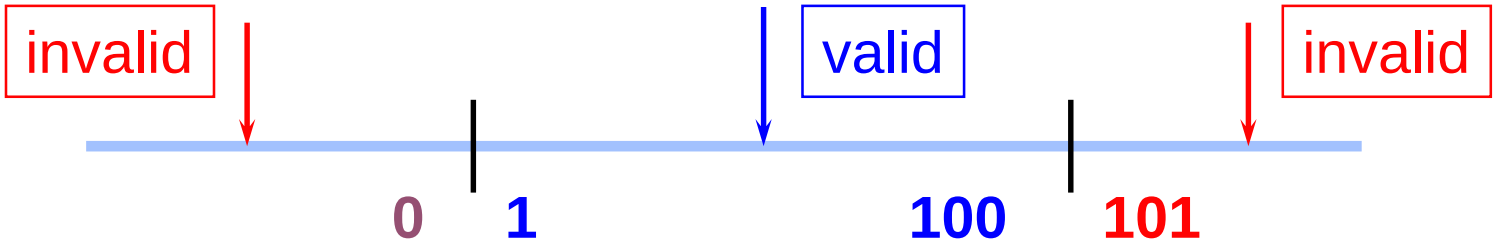
Kỹ thuật Phân vùng tương đương

- 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.
  - Áp dụng: Màn hình, Menu hay mức quá trình

Phân vùng tương đương – EP

- Trong ví dụ trên dùng kỹ thuật phân vùng tương đương, chia làm 3 phân vùng như sau:



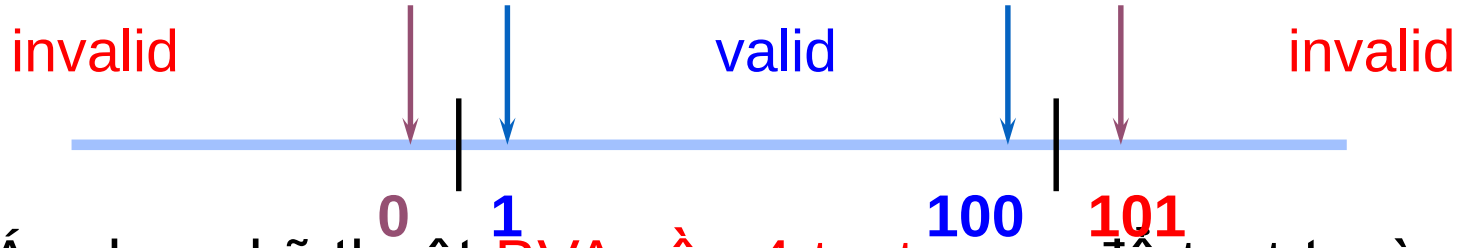
- Như vậy chỉ cần chọn 3 test case để test trường hợp này: -5, 55, 102 hoặc 0, 10, 1000, …

Phân vùng tương đương – EP

- 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 biên - BVA (Boundary Value Analysis)
  - Là 1 trường hợp riêng của EP.
  - Kỹ thuật BVA sẽ chọn các giá trị nằm tại các điểm giới hạn của phân vùng.
  - Thường dùng trong kiểm thử module.



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

Phân vùng tương đương – Ví dụ

- Hàm tính trị tuyệt đối: |x|
  - Miền dữ liệu: ≥ 0
  - Miền dữ liệu: < 0
  - Các giá trị cần test:

Input	Expected Result
100	100
-20	20
0	**0**

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.
  - Dùng kỹ thuật EP:
    - 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

Invalid partition	Valid (for 3% interest)		Valid (for 5%)		Valid (for 7%)
–$0.01	$0.00	$100.00	$100.01	$999.99	$1000.00

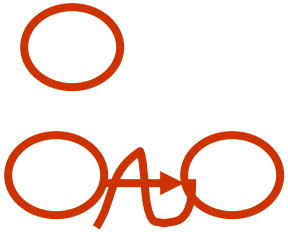
Tại sao phải kết hợp BVA và EP

- 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

Phương pháp đoán lỗi (Error Guessing)

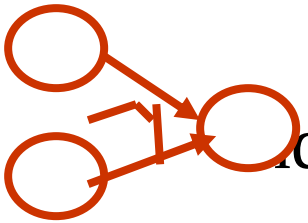
- Dựa vào trực giác và kinh nghiệm
- Thí dụ lỗi chia cho 0. Nếu môđun có phép chia thì phải kiểm thử lỗi này
- Nhược điểm: không phát hiện hết lỗi

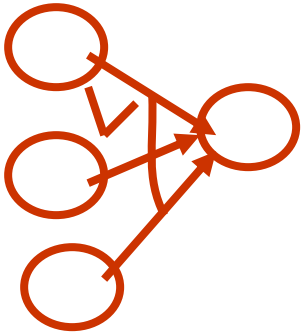
Phương pháp đồ thị nguyên nhân - kết quả (Cause-effect Graphing)



Mã tuần tự

Phủ định





Or

Do Until

Bài tập

- Tạo test case (dựa trên phân hoạch tương đương) cho hàm tìm kiếm sau:
- input:
  - mảng số nguyên a[] đã sắp xếp
  - khóa tìm kiếm k (số nguyên)
- output:
  - vị trí của k trong mảng a[] nếu có,
  - -1 nếu không có

Testcase cho tìm kiếm nhị phân

- Số phần tử của mảng:
  - 0, 1
  - lớn hơn 1
- Khóa tìm kiếm:
  - không có trong mảng
    - nhỏ hơn, lớn hơn
    - xen kẽ
  - có trong mảng
    - phần tử đầu tiên, cuối cùng
    - phần tử ở vị trí bất k

Testcase cho tìm kiếm nhị phân

Số phần tử	Mảng	Khóa	Kết quả
0		7	-1
1	10	20	-1
1	10	3	-1
1	10	10	0
4	3, 7, 10, 20	1	-1
4	3, 7, 10, 20	30	-1
4	3, 7, 10, 20	8	-1
4	3, 7, 10, 20	3	0
4	3, 7, 10, 20	20	3
4	3, 7, 10, 20	7	1

Tester

- Vai trò
  - Kiểm lỗi phần mềm
  - Kiểm lỗi bản đóng gói
  - Kiểm lỗi tài liệu
    - User guide
    - Installation Guide
    - Release Notes
    - Troubleshooting

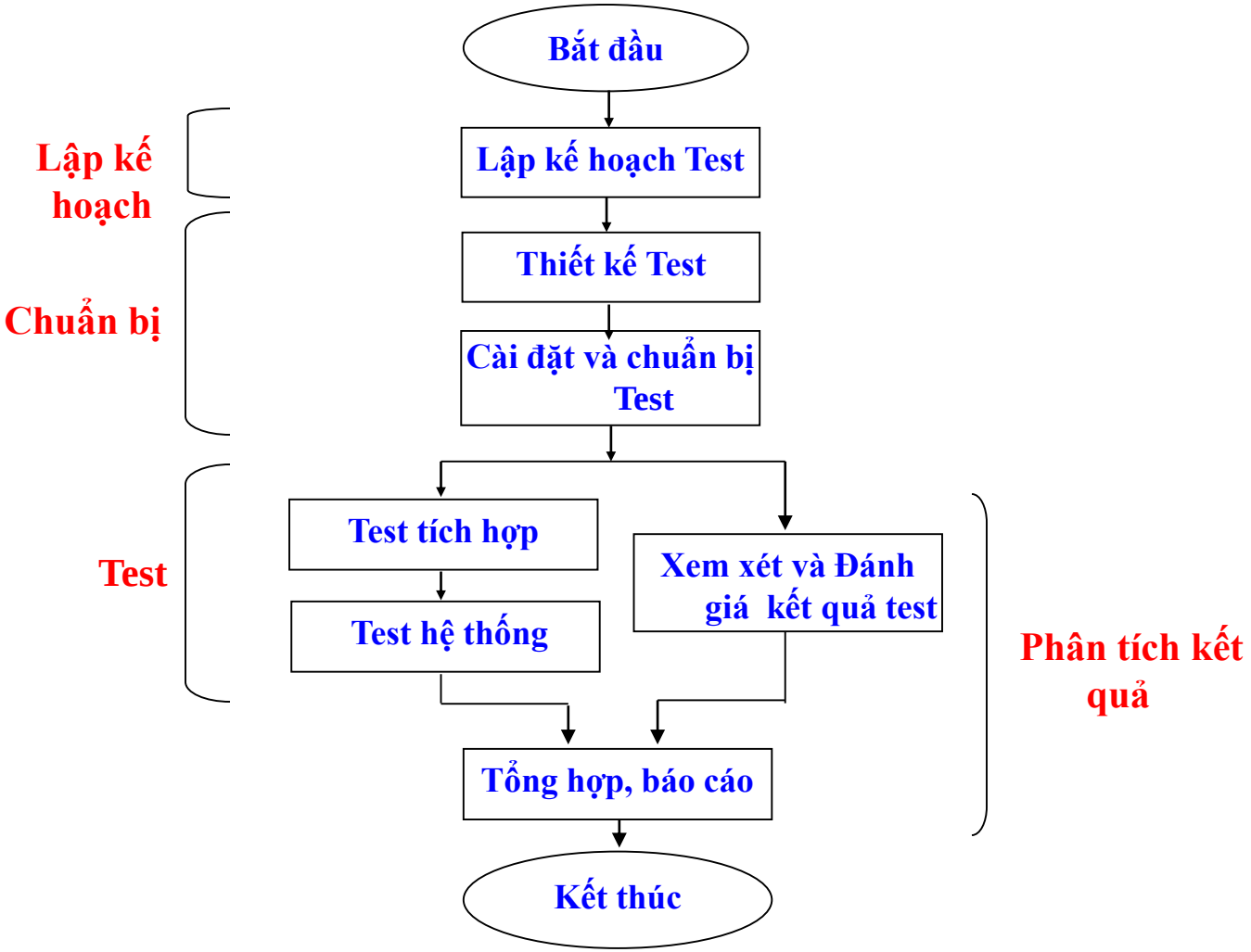
Tester

- Công việc
  - Chuẩn bị môi trường test
    - Windows XP, 2000, 2003
    - Linux
    - IE, FireFox, Netscape, Mozilla
    - Test Database, Test data
  - Viết 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

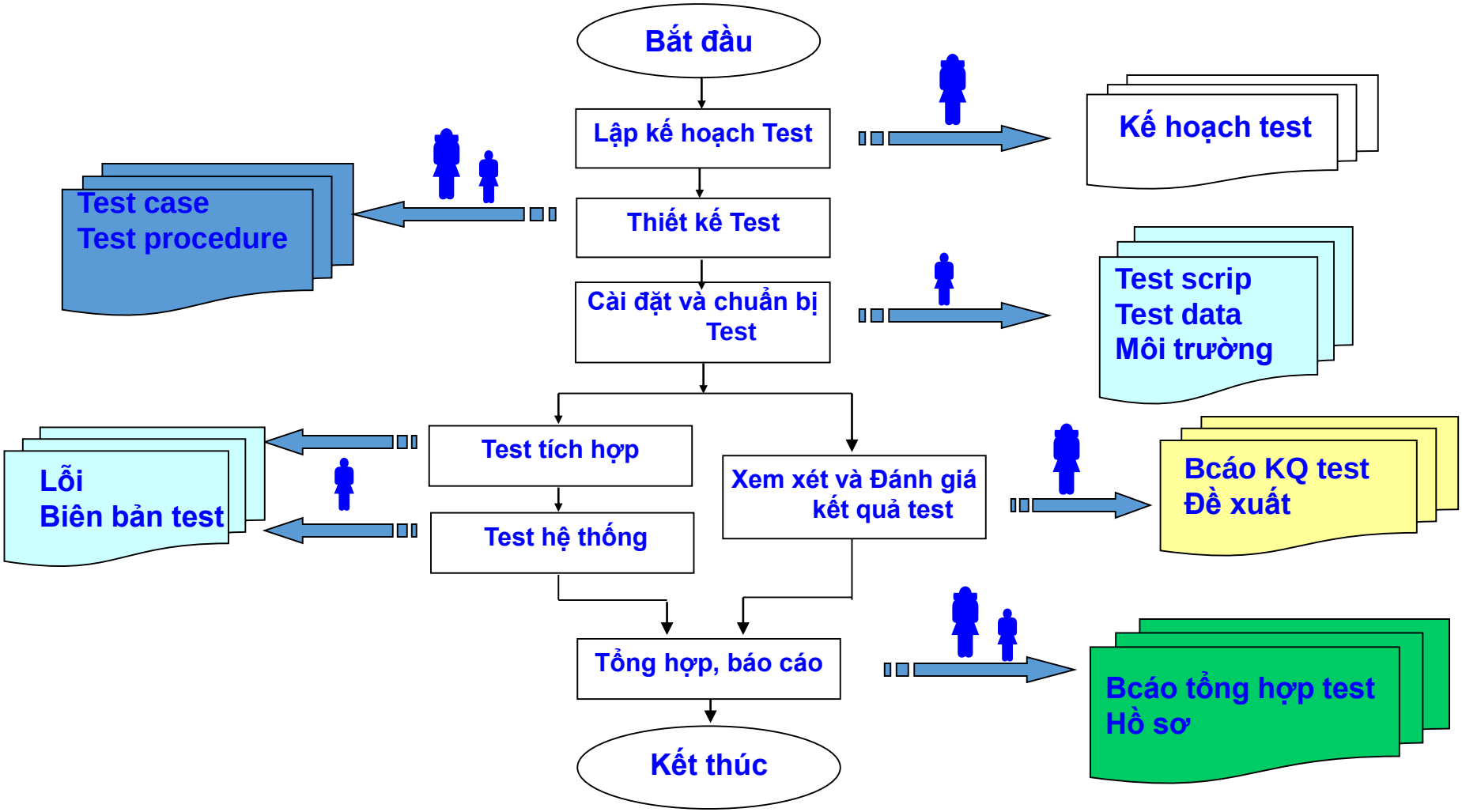
Tester

- Phần mềm sử dụng
  - Web testing
    - Test Manager Role
    - Tester Role
  - Manual Test (Rational Manual Test, Test Complete…)
  - Automation Test (Rational Functional Test, Test Complete,…)
  - Load testing
  - Code Analysis
  - Project Management Tool
    - Tester Role
  - Workflow
    - Tester Role

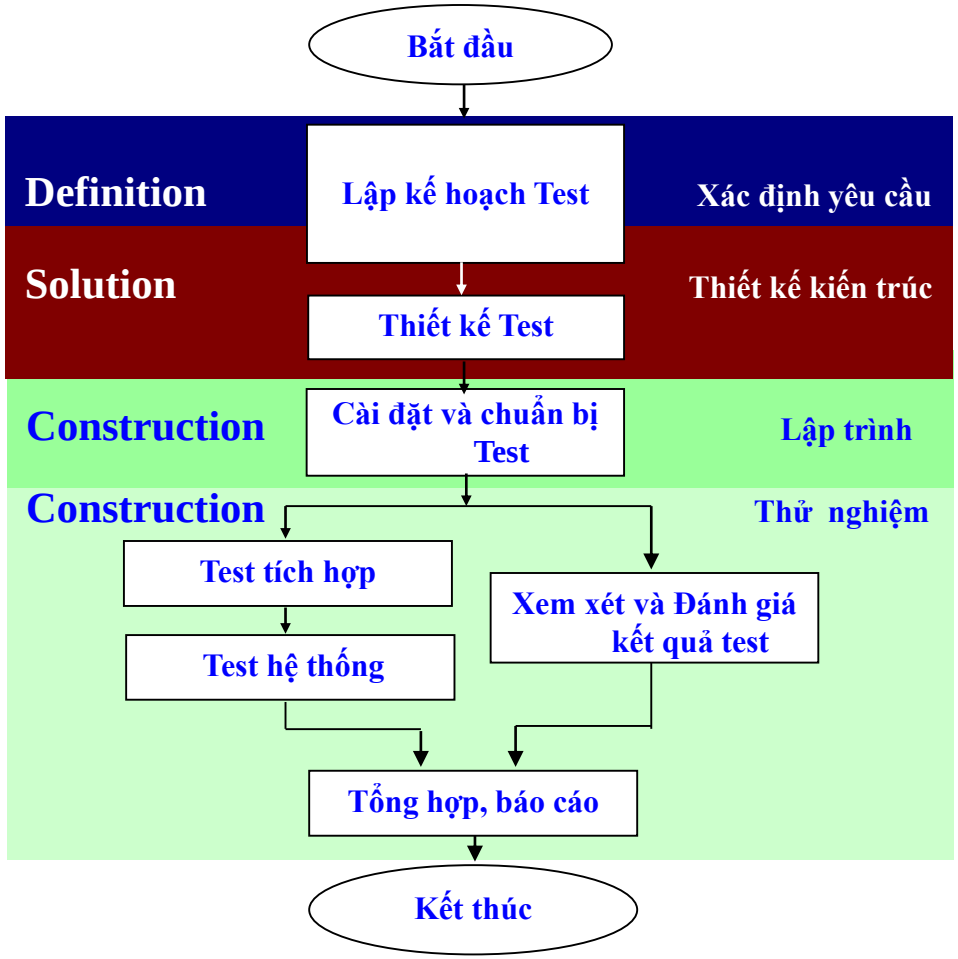
CÁC HOẠT ĐỘNG KIỂM THỬ

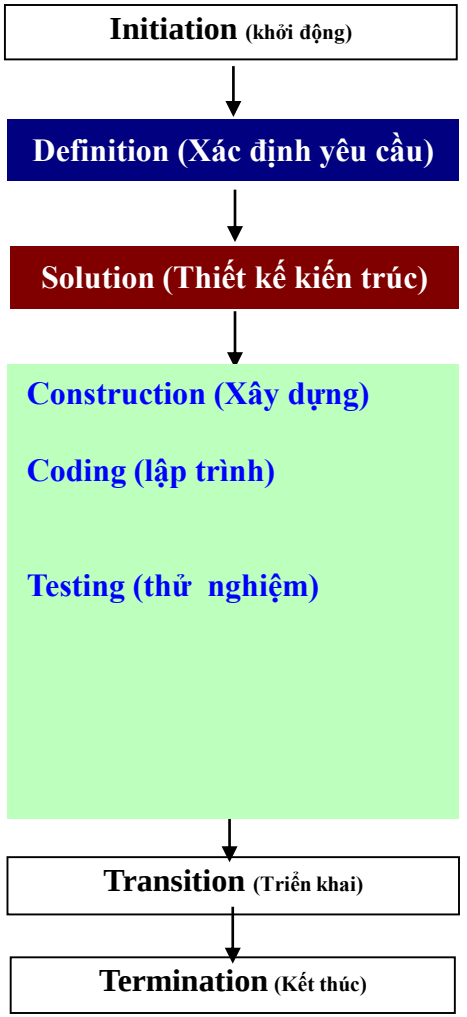


CÁC HOẠT ĐỘNG KIỂM THỬ



CÁC HOẠT ĐỘNG KIỂM THỬ





CÁC HOẠT ĐỘNG - Lập kế hoạch test

1. Xác định yêu cầu cho test	Các yêu c	
2. Đánh giá rủi ro và lập mức ưu tiên cho các yêu cầu	Các rủi ro	
3. Xác lập chiến lược test		TN Test
4. Xác định nguồn lực và môi trường	C	
5. Lập lịch trình test		TN Test
6. Tổng hợp thông tin, lập KH test		TN Test
7. Xem xét và thông qua KH test	t	QTDA

- Dựa trên các sản phẩm phân tích
- Xác định: test cái gì?
- Xác định: giới hạn công việc, thời gian, nguồn lực

- Phương thức thực hiện
- Tiêu chuẩn hoàn thành việc test, đánh giá chất lượng sản phẩm
- Các yếu tố cần chú ý

- Số người, kỹ năng
- Môi trường test: phần mềm, cứng…
- Công cụ test
- Dữ liệu test

- Xác định nguồn lực
- Lịch trình test
- Các mốc chính

CÁC HOẠT ĐỘNG - thiết kế test

- Dựa trên các sản phẩm của:
  - Phân tích
  - bước lập KH test
- Xác định các test case
- Mô tả test case: vào, ra, điều kiện

1. Lập danh sách các loại test, đảm bảo cho việc xác lập tính đúng đắn & thỏa mãn yêu cầu của sản phẩm	Danh sá	
2. Xây dựng tình huống test (test case)	Thiết kế test, các mẫu mã sử dụng, yêu cầu về dữ liệu test	TN Test, CBT
3. Xây dựng và tổ chức thủ tục test (test procedure)	Các thủ tục test	TN Test,
4. Xem xét tình huống test và thủ tục test, đánh giá tỷ lệ yêu cầu của khách hàng (hoặc tình huống sử dụng) sẽ được test dựa trên thiết kế test đã lập	Biên ba	
5. Thông qua thiết kế Test	Thiết kế test được thông qua	QTDA

Tình huống kiểm thử

Kiểm tra 1 chức năng của chương trình

Các giá trị đầu vào — Các điều kiện thực hiện → Kết quả dự kiến

- Dựa vào: các test case
- Xác định các test procedure
- Xác lập mối quan hệ và thứ tự giữa
  - Các test procedure với nhau
  - Các test procedure - Test cases

Thủ tục kiểm thử

- Các chỉ dẫn để thiết lập, thực hiện và đánh giá kết quả test
- Cho 1 hoặc nhiều tình huống test (test case)

CÁC HOẠT ĐỘNG - Cài đặt và chuẩn bị test

Test script

- **Các lệnh dùng để tự động hoá các thủ tục test**
- **Lập trình, sử dụng công cụ test tự động**

- **Lập trình hoặc sinh tự động: = tool**
- **Kiểm tra thử nghiệm**

1.Lập các test script để thực hiện các tình huống test/thủ tục test (nếu cần)	Test scipts	TN Test/
2. Chuẩn bị dữ liệu test	Dữ l	
3. Chuẩn bị môi trường	Môi t việc thực hiện test	
4. Kiểm tra các công cụ test	Biên bản kiểm tra công cụ	CBT
5. Xem xét môi trường, điều kiện và dữ liệu test	Mô ng và dữ liệu test được k	TN Test/ CBT

- **Tạo mới dữ liệu**
- **Tái sử dụng các dữ liệu cũ**

- **Phần mềm, phần cứng**
- **Các điều kiện khác**

CÁC HOẠT ĐỘNG - test tích hợp

- **Tài liệu**
- **Gói phần mềm**
- **…**

1. Nhận bàn giao với đội lập trình	Các sản phẩm cầ tiếp nhận	
2. Cài đặt	Hệ thống để test sẵn sàng	CBT
3. Thực hiện test và ghi nhận lỗi	Biên bản test	CBT
4. Xử lý lỗi	Danh sách lỗi phát hiện	TN Test, CBT
5. Xem xét các kết quả test việc thực hiện khắc phục lỗ		TN Test, QTDA, CBT, CBCL (nếu cần)

- **Dựa trên thiết kế test**
- **Ghi nhận lỗi vào DMS**

- **Phối hợp với đội lập trình**
- **Test lại -> Ghi nhận lỗi mới**

CÁC HOẠT ĐỘNG - test hệ thống

- **Kiểm tra và cập nhật:**
  - **Test case**
  - **Test procedure**
  - **Test script**
  - **Test data**

1. Nhận bàn giao với đội lập t…	…iếp	CBT
2. Chỉnh sửa thiết kế test	…ệu	TN Test
3. Cài đặt	Chương trình được cài đặt	CBT
4. Thực hiện test và ghi nhận lỗi	Biên bản test	CBT
5. Xử lý lỗi	Danh sách lỗi phát hiện	TN Test, CBT
6. Xem xét các kết quả test và việc thực hiện khắc phục lỗi	Kết quả test được xem xét	TN Test,QTDA, CBT, CBCL (nếu cần)
7. Kết quả test được xem xét	Xác nhận sản phẩm đủ tiêu chuẩn	QTDA,

- **Khi hệ thống thoả mãn các tiêu chuẩn đặt ra**

CÁC HOẠT ĐỘNG - xem xét và đánh giá kết quả test

1. Phân tích lỗi và đưa ra đề xuất	Báo cáo kết quả test Đề xuất	TN Test
2. Đánh giá tỷ lệ test, đánh giá mức độ đạt được các tiêu chí để hoàn thành test	áo cáo kết quả test	TN Test
3. Xem xé áo cáo kết quả test	Báo c ả test được xem xét	QTDA, CBCL

- **Dựa trên tỷ lệ YC được test/tổng số YC cần test**

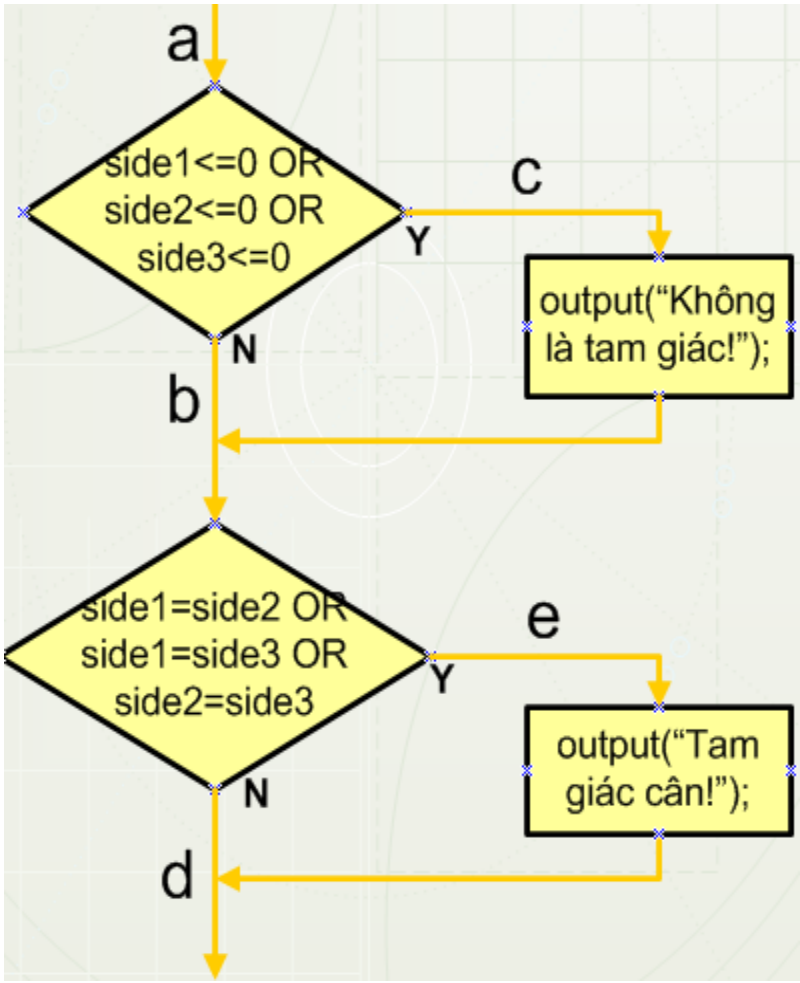
- **Phân tích lỗi dựa:**
  - **mức độ lặp lại**
  - **độ nghiêm trọng**
  - **Thời gian sửa lỗi…**

CÁC HOẠT ĐỘNG - tổng hợp, báo cáo

1. Tập hợp các dữ liệu, kết quả test	Dữ liệu, kết quả test	CBT
2. Lập báo cáo tổng hợp test	Báo cáo tổng hợp test	TN Test
3. Tổ chức lưu trữ tài liệu, hồ sơ	Hồ sơ, files	CBT

VD kiểm tra 3 cạnh tam giác

- Ví dụ bên có 5 nhánh a,b,c,d,e.
- Có 4 đường đi ace, abd, abe, acd.
- Có 6 điều kiện: side1<=0, side2<=0, side3<=0, side1=side2, side2=side3, side1=side3.

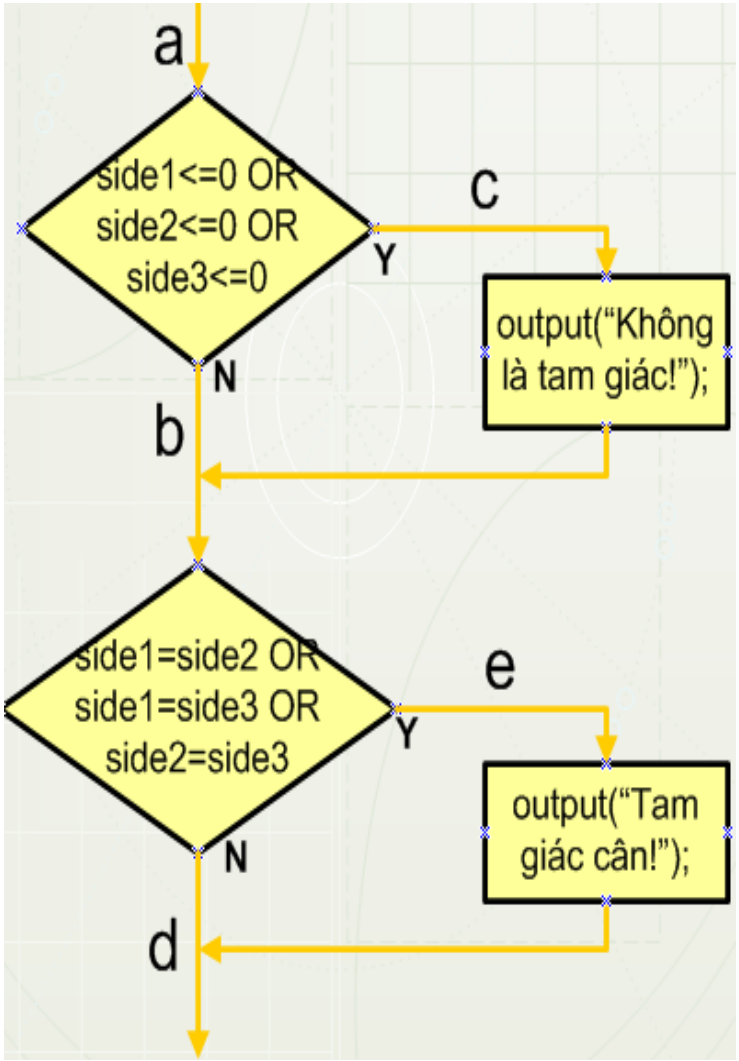


Số testcase ít nhất để thỏa:

- Phủ nhánh:
  1. {1, 1, 2} (abe)
  2. {-1, -2, -3} (acd)

- Phủ đường đi:
  1. {1, 1, 1} (abe)
  2. {-1, -2, -3} (acd)
  3. {-1, -1, -1} (ace)
  4. {1, 2, 3} (abd)

- Phủ điều kiện:
  1. {1, 1, 2} (abe)
  2. {-1, -2, -1} (ace)
  3. {-1, -2, -2} (ace)



- Phủ điều kiện/ nhánh:
  1. {1, 1, 1} (abe)
  2. {-1, -2, -3} (acd)

- Phủ kết hợp nhiều điều kiện:

  số testcase ít nhất là 2^6=64

Tài liệu tham khảo

- Testing Tools
  - http://www.aptest.com/resources.html
  - http://www-01.ibm.com/software/awdtools/tester/functional/features/index.html?S_TACT=105AGX15&S_CMP=LP
  - http://www-01.ibm.com/software/awdtools/test/manager/
- 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

Câu hỏi và thảo luận

Thank you!!!