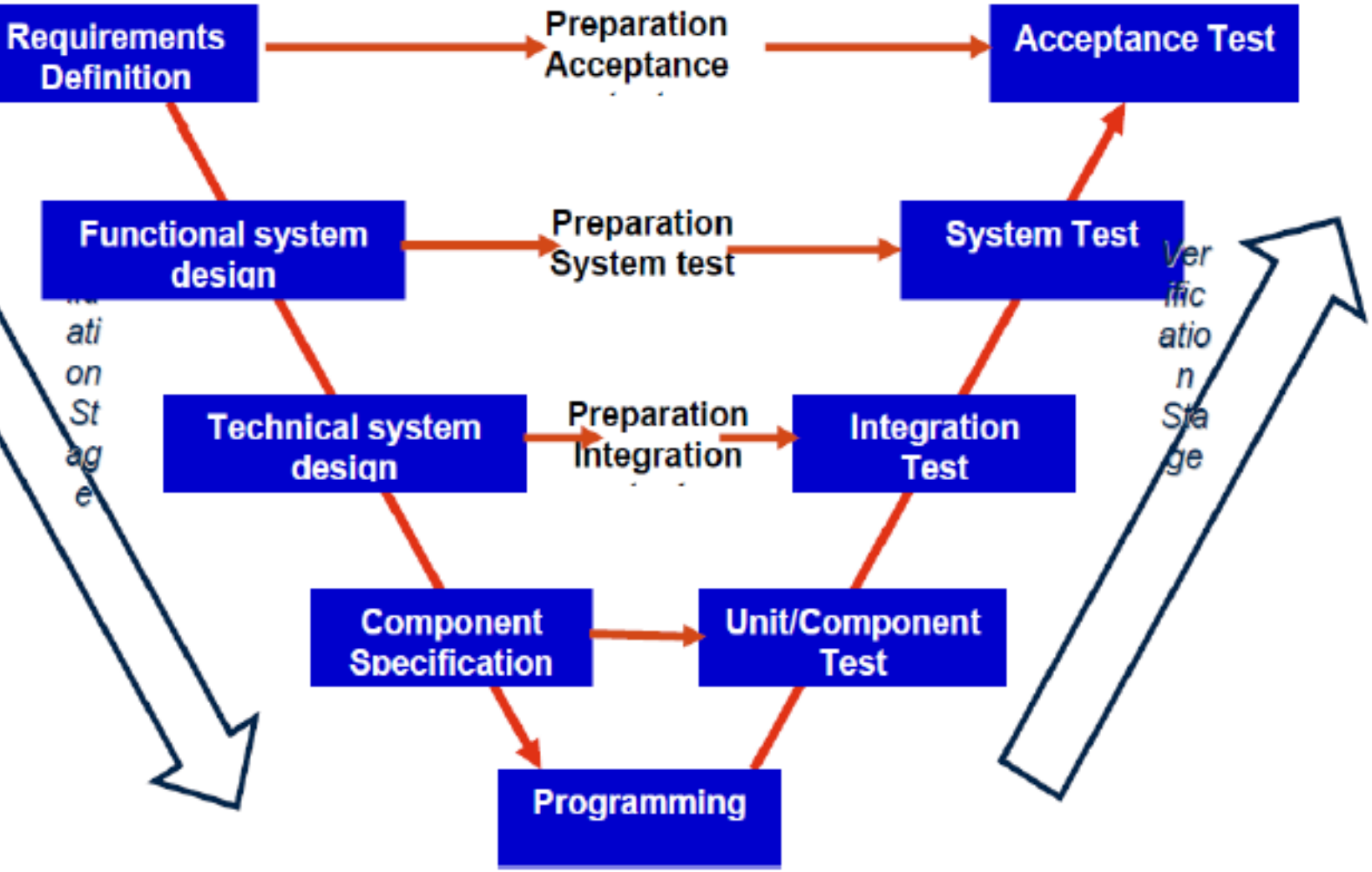
KIỂM THỬ PHẦN MỀM

(Software Testing)

GV: ThS. Nguyễn Thị Thanh Trúc
Khoa: Công nghệ Phần mềm
Email: trucntt@uit.edu.vn

BÀI 3: Các cấp độ kiểm thử



Một chiến thuật kiểm thử phổ biến

- Bắt đầu tại từng module rồi tích hợp lớn dần đến toàn bộ hệ thống.
- Các kỹ thuật khác nhau được sử dụng thích hợp tại các giai đoạn khác nhau.
- Kiểm thử có thể được tiến hành bởi người phát triển phần mềm, nhưng đối với các dự án lớn thì việc kiểm thử phải được tiến hành bởi một nhóm độc lập.
- Kiểm thử và sửa lỗi là các hoạt động độc lập nhưng việc sửa lỗi phải phù hợp với các chiến thuật kiểm thử.

Kiểm thử từng module

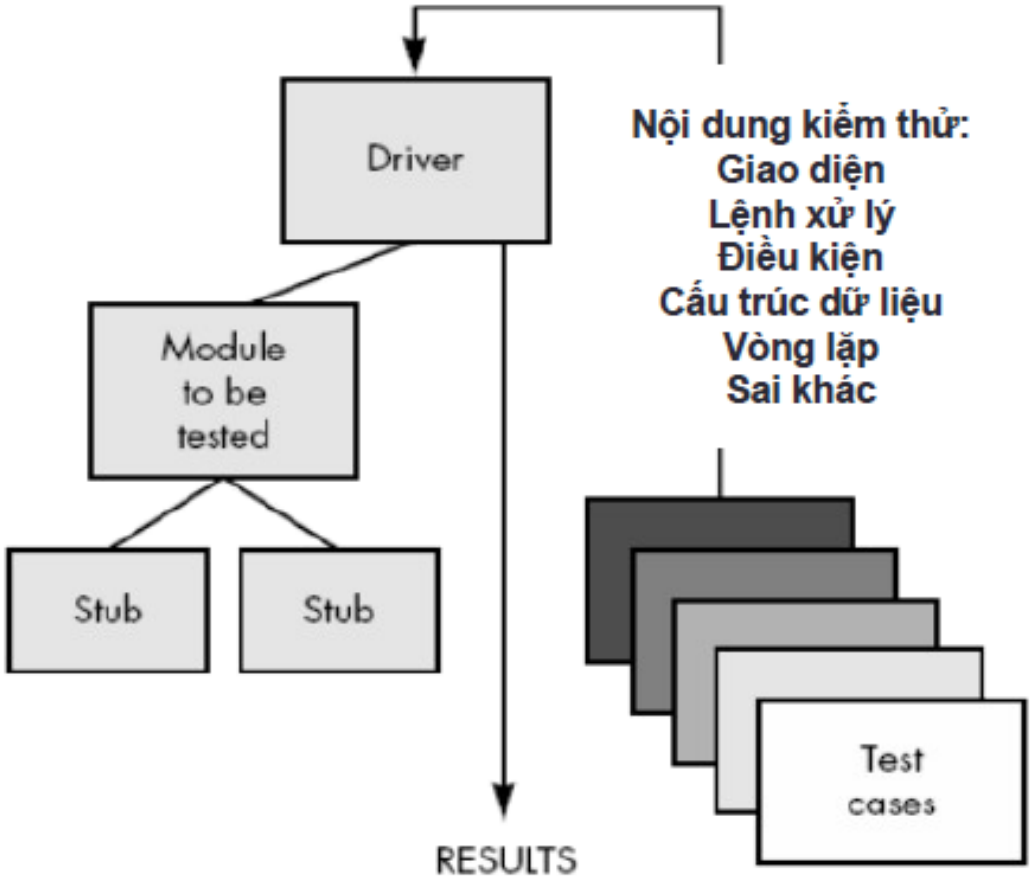
- Tiến hành kiểm thử trên từng đơn vị nhỏ nhất của phần mềm, đó là module mã nguồn, sau khi đã thiết kế, mã hoá và biên dịch thành công
- Thường dùng kỹ thuật kiểm thử white-box
- Có thể tiến hành kiểm thử cùng lúc nhiều module.
- Một số vấn đề trong việc xây dựng các test case
  - Test case nào?
  - Dữ liệu đầu vào và đầu ra có từ đâu?
  - Tính độc lập/phụ thuộc hoạt động của các module

3.1 Kiểm thử đơn vị

- Kiểm thử đơn vị nhằm kiểm tra đơn vị thiết kế nhỏ nhất một module phần mềm. Một module hoạt động thường có trao đổi thông tin với module mức dưới và mức trên nó, do đó phạm vi phát hiện lỗi liên quan chặt chẽ tới module này
- **Người tiến hành kiểm thử đơn vị**: lập trình viên cùng nhóm của mình.
- **Kỹ thuật kiểm thử đơn vị**: chủ yếu là hộp trắng, trong các trường hợp cần thiết có thể sử dụng thêm kỹ thuật kiểm thử hộp đen

3.1.1 Mô hình kiểm thử đơn vị

- Driver, stub



Kiểm thử module

- Mỗi module mã nguồn không phải là một chương trình hoàn chỉnh và đôi khi phải gọi các module chưa được kiểm thử khác → có thể phải thiết lập **driver** và/hoặc **stub**: phí tổn khá lớn (70%)
- **Driver** là một chương trình chính có nhiệm vụ nhận dữ liệu kiểm thử, chuyển dữ liệu đó xuống cho module để kiểm tra và in ra các kết quả kiểm tra tương ứng.
- **Stub** thay thế các module được gọi bởi module đang kiểm tra.

Làm thế nào để giảm các chi phí tạo driver hay stub

3.1.2 Nội dung kiểm thử đơn vị

- a) Kiểm thử giao diện(các tham số vào/ra qua giao diện)
- b) Kiểm thử vào/ra (các file, bộ đệm và các lệnh đóng mở)
- c) Kiểm thử cấu trúc dữ liệu cục bộ (khai báo và sử dụng biến)
- d) Kiểm thử xử lý (các phép toán và tính đúng đắn của kết quả)
- e) Kiểm thử điều kiện logic
- f) Kiểm thử sai tiềm ẩn (về ngoại lệ, mô tả)
- g) Kiểm thử các giá trị biên

a. Kiểm thử dữ liệu qua giao diện

- Kiểm thử dòng dữ liệu qua giao diện của module liên quan đến định lượng và định dạng của các biến và các module sử dụng trên giao diện
- Đặc trưng cụ thể:
  - Số lượng?
  - Định dạng?

a. Kiểm thử dữ liệu qua giao diện

- Các đặc trưng qua giao diện là:
- Số tham số= số đối số?
- Tính chất của tham số= tính chất của đối số
- Đơn vị của tham số= đơn vị của đối số
- Số đối số được truyền gọi module= số các tham số đầu vào của module?
- Thứ tự truyền tham số ko chính xác

…

• Ví dụ:

```
String calc_day(date d) {…}
Void calc_day_test()
{
        date d;
        string s;
        …
        d= calc_day(s);// truyền tham số ko chính xác
…
}
```

b. Kiểm thử vào/ra

- Kiểm thử các file, bộ đệm, các lệnh đóng, mở
- Khi thực hiện kiểm thử vào/ ra cần xem xét:
  - Tính chất của các file có đúng đắn ko?
  - Các câu lệnh OPEN/CLOSE có đúng đắn ko?
  - Đặc tả hình thức có đúng đắn ko?
  - Các file có mở trước khi sử dụng ko?
  - Các điều kiện end of file có được xử lý không?
  - Có sai văn bản nào trong thông tin ra?

c. Kiểm thử cấu trúc dữ liệu cục bộ

- Kiểm thử khai báo và sử dụng biến
- Cấu trúc dữ liệu cục bộ cho module có thể sai. Vì thế thiết kế các kiểm thử cần làm lộ ra các loại lỗi sau:
  - Đánh máy ko đúng hoặc ko nhất quán?
  - Giá trị ngầm định hoặc giá trị khởi tạo sai
  - Tên các biến ko đúng (sai chữ hoặc mất chữ)
  - Kiểu dữ liệu không nhất quán

Vd:

```
int i;
d= i*10; // lỗi chưa khai báo biến d
```

d. Kiểm thử về các xử lý

- Kiểm thử các phép toán và tính đúng đắn của kết quả
- Cần lưu ý các sai về trình tự, độ chính xác:
  - Thứ tự ưu tien các phép tính số học
  - Sự nhất quán của các phép toán trộn module
  - Khởi tạo/kết thúc không đúng
  - Độ chính xác của kết quả trả về
- Ví dụ:
  - Thực hiện phép toán trên toán hạng ko phải là số:
  - String s1, s2;
  - Int ketqua=s1/s2;

e. Kiểm thử các điều kiện logic

- Các sai kiểu, toán tử, ngữ nghĩa:
  - So sánh các kiểu dữ liệu khác nhau
  - Ưu tiên hoặc toán tử logic không đúng đắn
  - Dự đoán một biểu thức so sánh, trong khi sai số làm cho đẳng thức không chắc có thực
  - Các giá trị so sánh không đúng đắn
- Ví dụ:

```
int ival;
char sval[20];
If(ival==sval) {…} //So sánh 2 dữ liệu ko tương thích
```

f. Kiểm thử sai tiềm ẩn

- Các sai tiềm ẩn cần được xem xét là:
  - Mô tả sai(khó hiểu)
  - Dữ liệu ghi không tương ứng với sai đã gặp
  - Điều kiện sai có trước khi xử lý sai
  - Xử lý điều kiện ngoại lệ là không đúng đắn
  - Mô tả sai không cung cấp đủ thông tin để trợ giúp định vị nguyên nhân của sai
- Ví dụ:

```
If (a>0) then {…}
If (a=0) then {…} // thiếu trường hợp xét a<0
```

g. Kiểm thử các giá trị biên

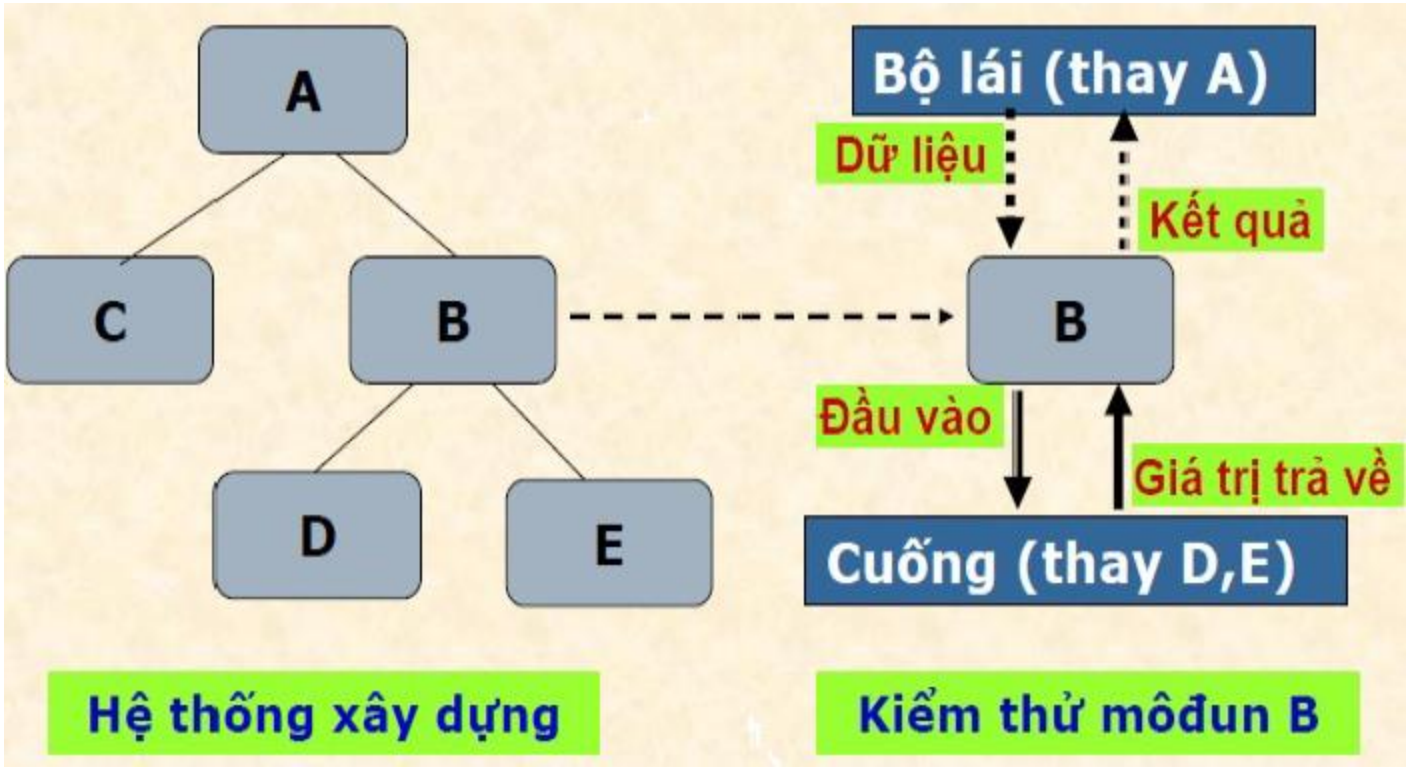
- Các sai biến, số vòng lặp:
  - Vòng lặp không kết thúc hoặc kết thúc không chính xác
  - Lặp vô hạn
  - Biến lặp bị thay đổi không chính xác
- Sai ở các biên:
  - Kiểm thử ở biên là nhiệm vụ cuối cùng của kiểm thử đơn vị. Các giá trị ở biên thường hay gây ra lỗi.
- VD:

```
Int i,j;
For (i=1;i<=10,i++)
For (j=i+1, j<=10, j++) {…}
```

3.1.3 Kỹ thuật kiểm thử đơn vị

- Module không phải là một chương trình độc lập, nên cần phát triển thêm các Driver và Stub để tiến hành kiểm thử đơn vị.
- **Bộ lái (driver):** là một hàm main điều khiển việc đưa dữ liệu vào và nhận kết quả của module đang cần kiểm thử
- **Cuống (stub):** là một chương trình máy tính dùng để thay thế cho một module phần mềm sẽ được xác định sau (IEEE)
- Stub (dummy program): Là một đoạn mã dùng để mô phỏng hoạt động của thành phần còn thiếu.

3.1.3 Kỹ thuật kiểm thử đơn vị



Code example -Stub

```
void function WeTest(params…) {
….
int p= price(param1);
….
} // chương trình chính
void price(int param) {
return 10; //không cần quan tâm giá là gì, được tính thế nào chỉ cần giá trị trả về để test module function WeTest
} // đây là stub
```

Code example- Driver

```
void function ThatCallPrice(params…) {// đây là driver
        int p= price(param1);
        printf(“Price is:%d”,p);
}
void price(int param){
….
// chương trình chính để tính được giá trị thật
…
}
```

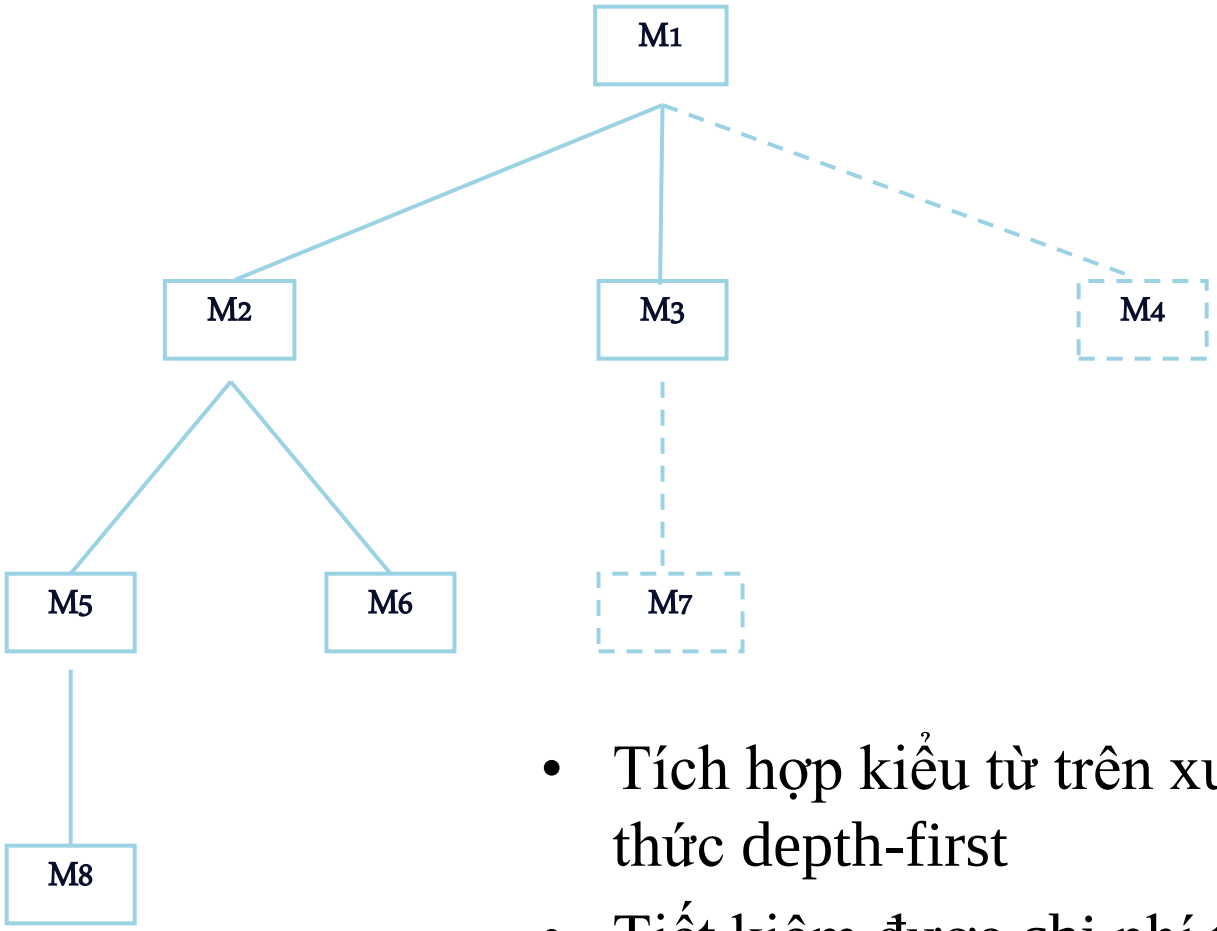
3.2. Kiểm thử tích hợp

- Kiểm thử tích hợp nhằm nhận được một bộ phận chức năng hay một hệ con tốt
- Là một kỹ thuật có tính hệ thống để xây dựng cấu trúc của chương trình
- Từ các module đã qua kiểm thử đơn vị, xây dựng cấu trúc chương trình đảm bảo tuân theo thiết kế
- Có hai cách tích hợp:
- **Tích hợp từng bước**. Theo cách này có 3 chiến lược:
  - Tích hợp từ dưới lên (bottom-up testing)
  - Tích hợp từ trên xuống (top-down testing)
  - Kết hợp 2 chiến lược trên (sandwich testing)
- **Tích hợp đồng thời**: kiểm thử vụ nổ lớn (big bang testing)

Tích hợp từ trên xuống

- Module chính được dùng như là driver, và stub được thay thế bởi các module con trực tiếp của của module chính này.
- Tuỳ thuộc vào cách tích hợp theo chiều sâu (depth-first) hoặc chiều ngang(breath-first), mỗi stub con được thay thế một lần bởi module tương ứng đã kiểm thử.
- Tiến hành kiểm thử khi có sự thay thế mới
- Tiến hành kiểm thử hồi quy để phát hiện các lỗi khác trong từng module

Tích hợp từ trên xuống

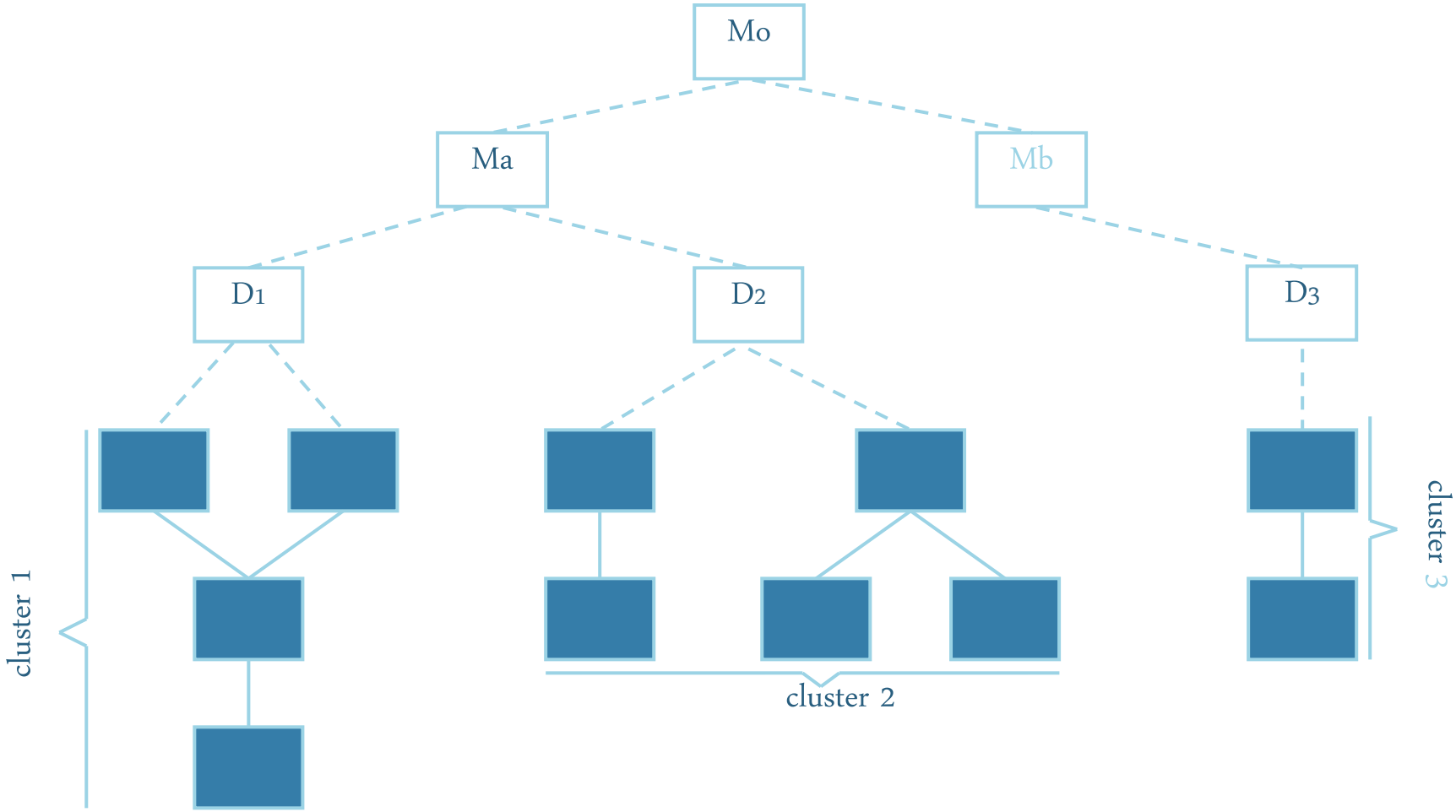


- Tích hợp kiểu từ trên xuống theo hình thức depth-first
- Tiết kiệm được chi phí tạo các driver

Tích hợp từ dưới lên

- Các module mức thấp nhất được kết hợp thành các nhóm thể hiện một chức năng con đặc biệt của phần mềm.
- Một driver được tạo ra để thao tác các test-case
- Các module được kiểm thử theo từng nhóm (Cluster): là nhóm các module mà module phía trên cần đến khi kiểm thử
- Driver được bỏ đi và các nhóm module được kết hợp dần lên phía trên trong sơ đồ phân cấp của chương trình.
- Tiết kiệm được chi phí tạo các stub

Tích hợp từ dưới lên



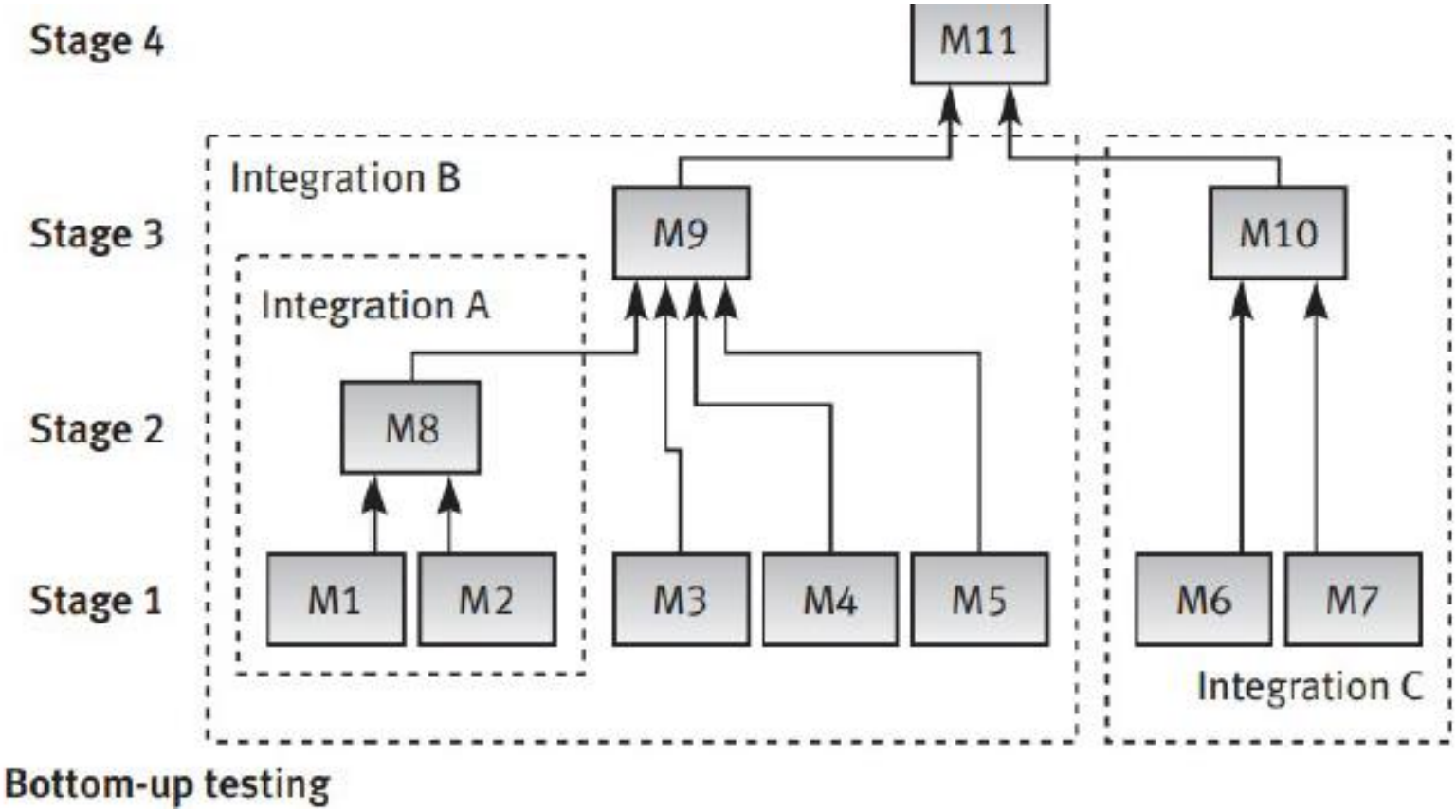
3.2.1. Các lỗi thường gặp khi tích hợp

- Dữ liệu bị mất khi đi qua một giao diện
- Hiệu ứng 1 module vô tình gây ra ảnh hưởng tới các module khác
- Sự kết hợp các chức năng phụ có thể không tạo ra được chức năng chính mong muốn
- Các sai sót nhỏ có thể trở thành thảm họa
- Có thể gặp vấn đề với các cấu trúc dữ liệu toàn cục

3.2.2. Kiểm thử từ dưới lên

- Là quá trình tích hợp và kiểm thử bắt đầu với các module mức thấp.
- Để kiểm thử các module cấp dưới, lúc đầu ta dùng các bộ lái, sau đó thay thế dần các bộ lái bằng các module thượng cấp đã được phát triển.

3.2.2. Kiểm thử từ dưới lên



Bottom-up testing

3.2.2. Kiểm thử từ dưới lên

- **Ưu điểm:**
  - Tránh phải tạo các cuống phức tạp hay tạo các kết quả nhân tạo: do tích hợp từ dưới lên nên chỉ cần tạo ra các bộ lái, các module mức dưới đã được kiểm thử
  - Thuận tiện cho phát triển các module cấp dưới: nhờ phát triển từ dưới lên, người thiết kế có thể thiết kế các module dịch vụ dùng chung cho nhiều chức năng của hệ thống

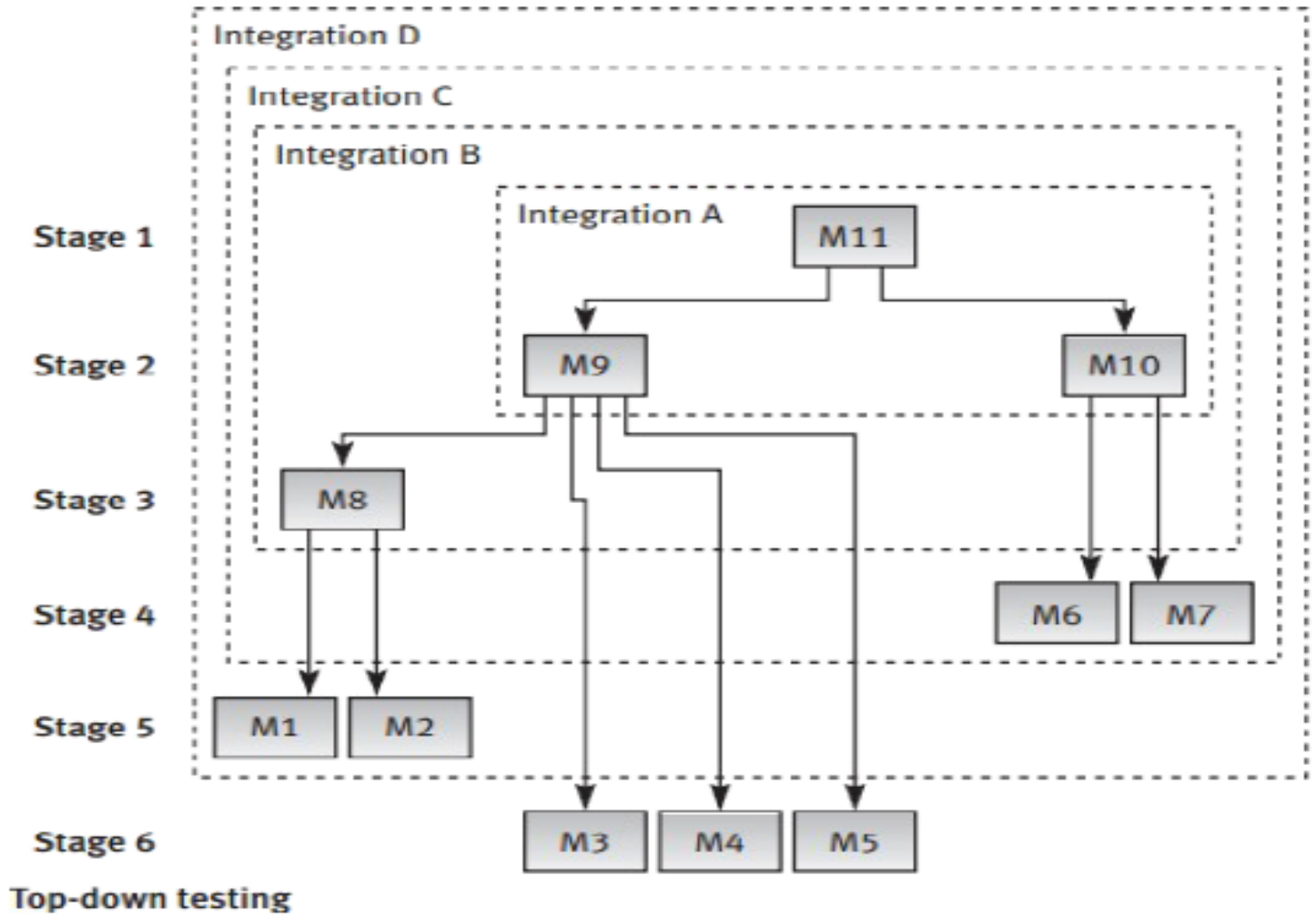
3.2.2. Kiểm thử từ dưới lên

- **Nhược điểm:**
  - Chậm phát hiện các lỗi thiết kế: các lỗi tổng thể như phát triển sai chức năng hay hệ thống kém kiệu quả thường bị phát hiện muộn. Do phát hiện lỗi muộn nên chi phí và thời gian sửa lỗi tăng cao
  - Chậm có phiên bản của hệ thống làm việc được.

3.2.3. Kiểm thử từ trên xuống

- Kiểm thử từ trên xuống tiến hành kiểm thử các module bắt đầu từ mức cao, các module mức thấp tạm sử dụng các chức năng hạn chế, có giao diện như đặc tả.
- Module mức thấp có thể chỉ đơn giản là các cuống trả lại kết quả với một vài đầu vào được xác định trước. Sau đó các cuống được thay thế dần bằng các module thực đã được phát triển.
- Kiểm thử từ trên xuống có thể thực hiện theo chiều sâu hoặc theo chiều rộng

3.2.3. Kiểm thử từ trên xuống



Top-down testing

3.2.3. Kiểm thử từ trên xuống

- **Ưu điểm:**
  - Phát hiện sớm các lỗi thiết kế: dễ dàng phát hiện các lỗi như phát triển nhầm, thiếu chức năng so với đặc tả, do đó làm giảm chi phí cho việc thiết kế và cài đặt lại
  - Có phiên bản hoạt động sớm: kiểm thử từ trên xuống luôn đảm bảo có phiên bản hoạt động sớm, do đó có thể thẩm định tính dùng được của sản phẩm và dùng nó để huấn luyện người dùng
- **Nhược điểm:**
  - Tạo ra các cuống rất phức tạp, có thể dẫn tới việc không kiểm thử được đầy đủ chức năng các module

3.3. Kiểm thử hệ thống

- Kiểm thử hệ thống:
  - Tìm kiếm các lỗi, nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống
  - Mức kiểm thử này đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với phần mềm hoặc phần cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ
  - Đòi hỏi nhiều thời gian, công sức, thiết bị…
- **Mục đích**: kiểm thử thiết kế và toàn bộ hệ thống (sau khi tích hợp) có thỏa mãn yêu cầu đặt ra hay không tìm.
- **Phương pháp**: kiểm thử hộp đen
- **Người thực hiện**: kiểm thử viên

3.3. Kiểm thử hệ thống

Khi nào có thể thực hiện kiểm thử hệ thống:

- Hệ thống cần kiểm thử đã hoàn thiện
- Kiểm thử tích hợp và đơn vị đã hoàn thành
- Sản phẩm được tích hợp đúng thiết kế
- Các tài liệu đặc tả đã là bản cuối cùng
- Các tài liệu hỗ trợ kiểm thử như test plan, test case đã hoàn thành.

3.4. Kiểm thử chấp nhận

- Kiểm thử chấp nhận (aceptance testing) : vận hành hệ thống trong môi trường của người sử dụng
- Kiểm thử alpha (alpha testing)
  - Người dùng thực hiện với số liệu giả lập
  - Trong môi trường phát triển
- Kiểm thử beta (beta testing)
  - Người dùng thực hiện với số liệu thực
  - Trong môi trường ứng dụng thực

Kiểm thử tính năng

- **Kiểm thử tính năng** hiểu theo cách đơn giản nhất là: kiểm tra các chức năng của phần mềm đáp ứng được nhu cầu của khách hàng đã được xác định trong văn bản đặc tả yêu cầu của phần mềm

- Áp dụng kỹ thuật black-box

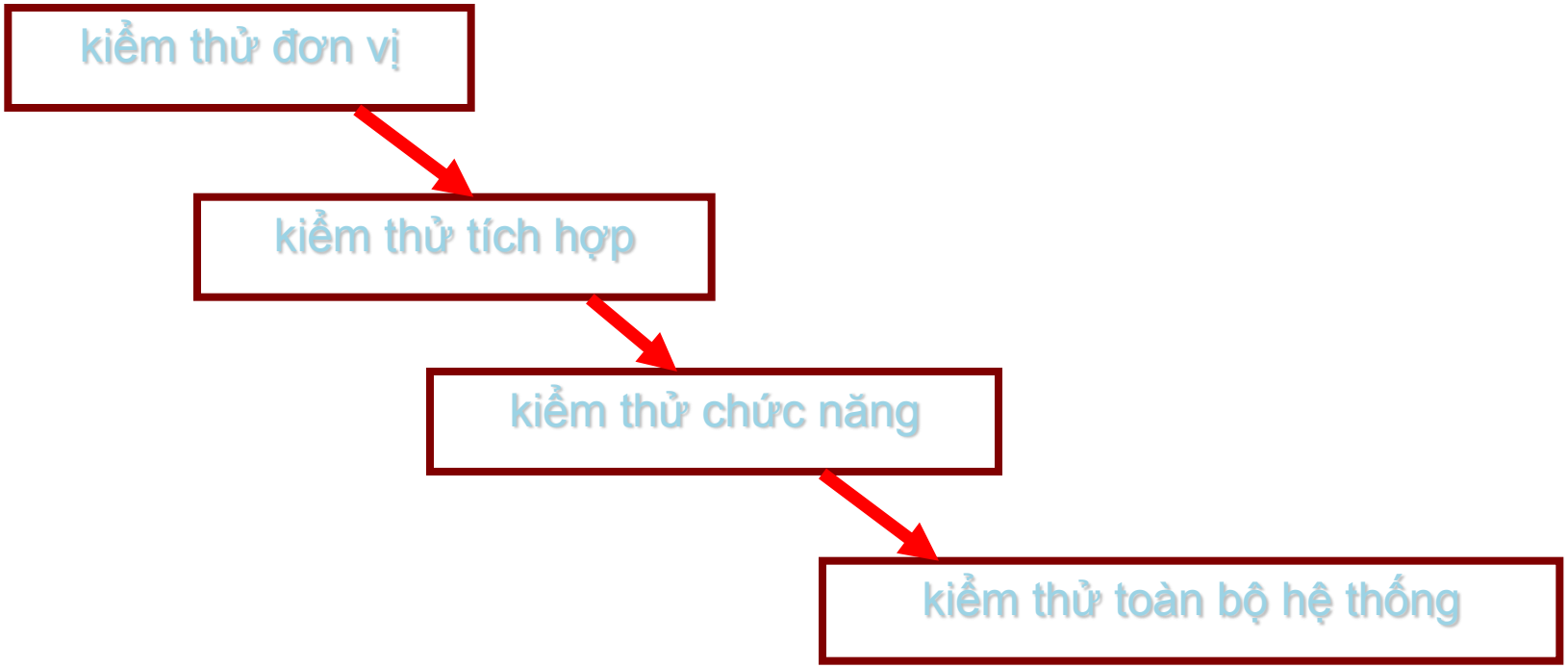
- Kiểm thử tính năng bao gồm
  - Xem xét lại cấu hình phần mềm theo lược đồ triển khai
  - kiểm thử alpha
  - kiểm thử beta

Kiểm thử tính năng

- Kiểm thử alpha
  - Được tiến hành ngay tại nơi sản xuất phần mềm.
  - Nhà phát triển phần mềm sẽ quan sát người sử dụng dùng sản phẩm và ghi nhận lại những lỗi phát sinh để sửa chữa.
- Kiểm thử beta
  - Phần mềm được kiểm tra bên ngoài phạm vi của đơn vị sản xuất.
  - Khách hàng trực tiếp sử dụng và ghi nhận lỗi để báo lại cho nhà phát triển sửa chữa.

Kiểm thử hướng đối tượng

- Về cơ bản chiến thuật kiểm thử hướng đối tượng cũng theo thứ tự giống như kiểm thử cổ điển:



Kiểm thử hướng đối tượng

- Không thể tách rời từng tác vụ của đối tượng/lớp để kiểm thử
  - Tác vụ được đóng bao trong lớp
  - Các lớp con có thể override một tác vụ nào đó
- Kiểm thử đơn vị hướng đối tượng tập trung vào các lớp → kiểm thử hành vi của lớp

Kiểm thử tích hợp hướng đối tượng

- Khái niệm sơ đồ phân cấp không còn nhiều ý nghĩa trong chương trình hướng đối tượng → kiểm thử tích hợp theo cách khác
- Hai hình thức kiểm thử tích hợp hướng đối tượng
  - kiểm thử trên cơ sở thread: tích hợp các lớp tạo thành một thread để phục vụ cho một input nào đó của chương trình
  - kiểm thử trên cơ sở sử dụng: các lớp client sẽ được tích hợp để sử dụng dịch vụ nào đó cung cấp bởi các lớp server

Kiểm thử theo kịch bản

- Dựa vào các use-case để soạn ra các kịch bản
- Ví dụ: một kịch bản cho hệ thống đăng ký môn học qua WEB

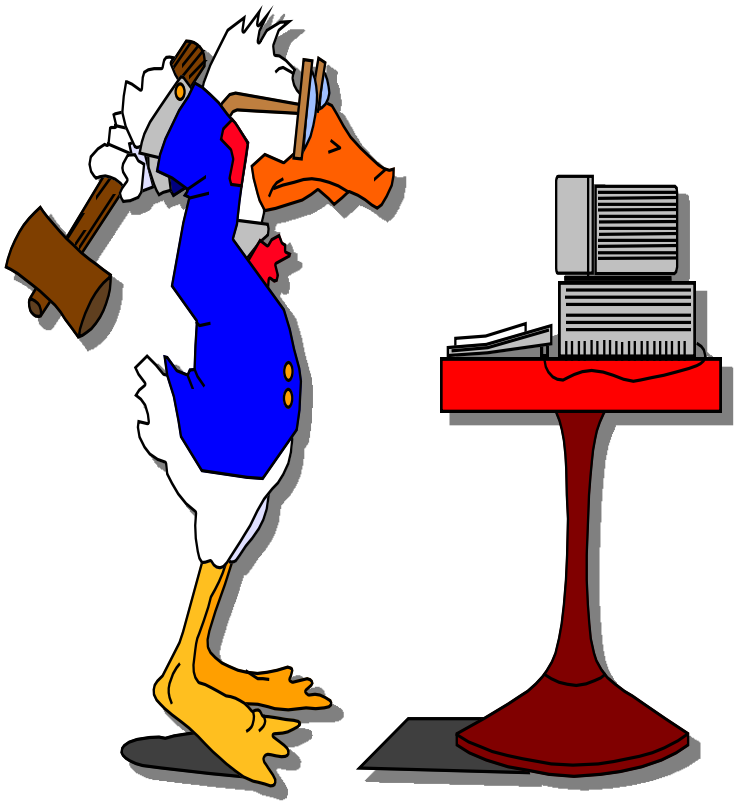
1. Login với username = “e59306547”, password = “6547”
2. Chọn chức năng đăng ký môn học
3. Chọn 5 nhóm môn học của 5 môn: CNPM, AI, XLTHS, PTTK, XLSS trong đó có 2 nhóm trùng thời khoá biểu
4. Nhấn nút Submit

Chương trình phải báo lỗi và liệt kê 2 nhóm bị trùng thời khoá biểu

Nghệ thuật gỡ rối - DEBUG

- Gỡ rối là một quá trình nhằm loại bỏ các lỗi được phát hiện trong quá trình kiểm tra.
- Gỡ rối được thực hiện như là một kết quả của việc kiểm tra: lỗi phát hiện được → tìm kiếm nguyên nhân → sửa lỗi
- Có 3 hình thức gỡ rối: brute force, loại trừ nguyên nhân và theo vết. Nên dùng kết hợp cả 3 hình thức này.

Nghệ thuật gỡ rối



- Gỡ rối là công việc khó khăn và dễ gây tâm lý chán nản bởi nguyên nhân gây ra lỗi nhiều khi lại mơ hồ: do timeout, do độ chính xác, do chủ quan lập trình...
- Khả năng gỡ rối gần như là bẩm sinh của mỗi người

Brute Force

- Là phương pháp phổ biến nhất nhưng lại ít hiệu quả nhất cho việc phát hiện nguyên nhân gây lỗi phần mềm.
- Triết lý của phương pháp này là: “Hãy để máy tính tìm ra lỗi”.
- Có 3 cách thực hiện:
  - Lấy dữ liệu trong bộ nhớ để xem xét.
  - Dùng run-time trace để tìm lỗi.
  - Dùng lệnh WRITE để xuất dữ liệu cần kiểm tra ra màn hình.
- Áp dụng phương pháp này khi tất cả các phương pháp khác đều thất bại.

Loại trừ nguyên nhân

- Phương pháp này dựa trên nguyên tắc phân chia nhị phân.
- Cách thực hiện:
  - Khi một lỗi được phát hiện, cố gắng đưa ra một danh sách các nguyên nhân có thể gây ra lỗi.
  - Danh sách này được nghiệm lại để loại bỏ dần các nguyên nhân không đúng cho đến khi tìm thấy một nguyên nhân khả nghi nhất.
  - Khi đó dữ liệu kiểm thử sẽ được tinh chế lại để tiếp tục tìm lỗi.

Theo vết

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