

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



NHẬP MÔN

CÔNG NGHỆ PHẦN MỀM

Giảng viên: TS. Nguyễn Thị Xuân Hương

Email: huongntx@uit.edu.vn

NỘI DUNG MÔN HỌC

- Tổng quan về Công nghệ phần mềm
- Xác định và mô hình hóa yêu cầu phần mềm
- Thiết kế phần mềm
- Cài đặt phần mềm
- **Kiểm thử và bảo trì**
- Đồ án môn học

KIỂM THỬ VÀ BẢO TRÌ

1. Kiểm định và Thẩm định
2. Kiểm thử phần mềm
3. Bảo trì phần mềm

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

- Thẩm định và kiểm định phần mềm là gì?
- Quy trình kiểm tra chương trình và vai trò của nó trong thẩm định và kiểm định
- Kỹ thuật kiểm định phân tích tĩnh

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

■ Nội dung:

- ◆ Lập kế hoạch thẩm định và kiểm định
- ◆ Software inspections
- ◆ Phân tích tĩnh được tự động hóa

1. Thẩm định và kiểm định – V&V

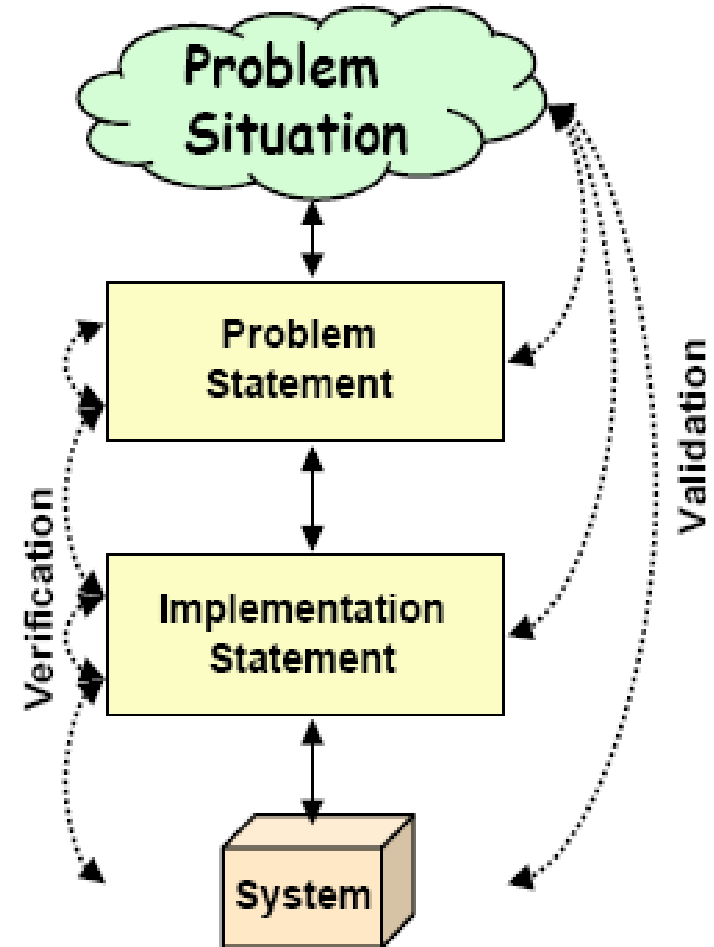
1.1. Thẩm định và kiểm định

■ **Thẩm định - Validation:** "Are we building the **right product**?"

- Phát biểu bài toán có phản ánh chính xác bài toán thực hay không?
- Ta đã xét đến nhu cầu của tất cả các stakeholder chưa?

■ **Kiểm định - Verification:** "Are we building the **product right**?"

- Thiết kế có tuân theo theo đặc tả không?
- Cài đặt có thỏa mãn đặc tả không?
- Hệ thống được giao cho khách hàng có thực hiện đúng những gì mà ta nói là nó sẽ làm?
- Các mô hình yêu cầu của ta có nhất quán với nhau không?



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

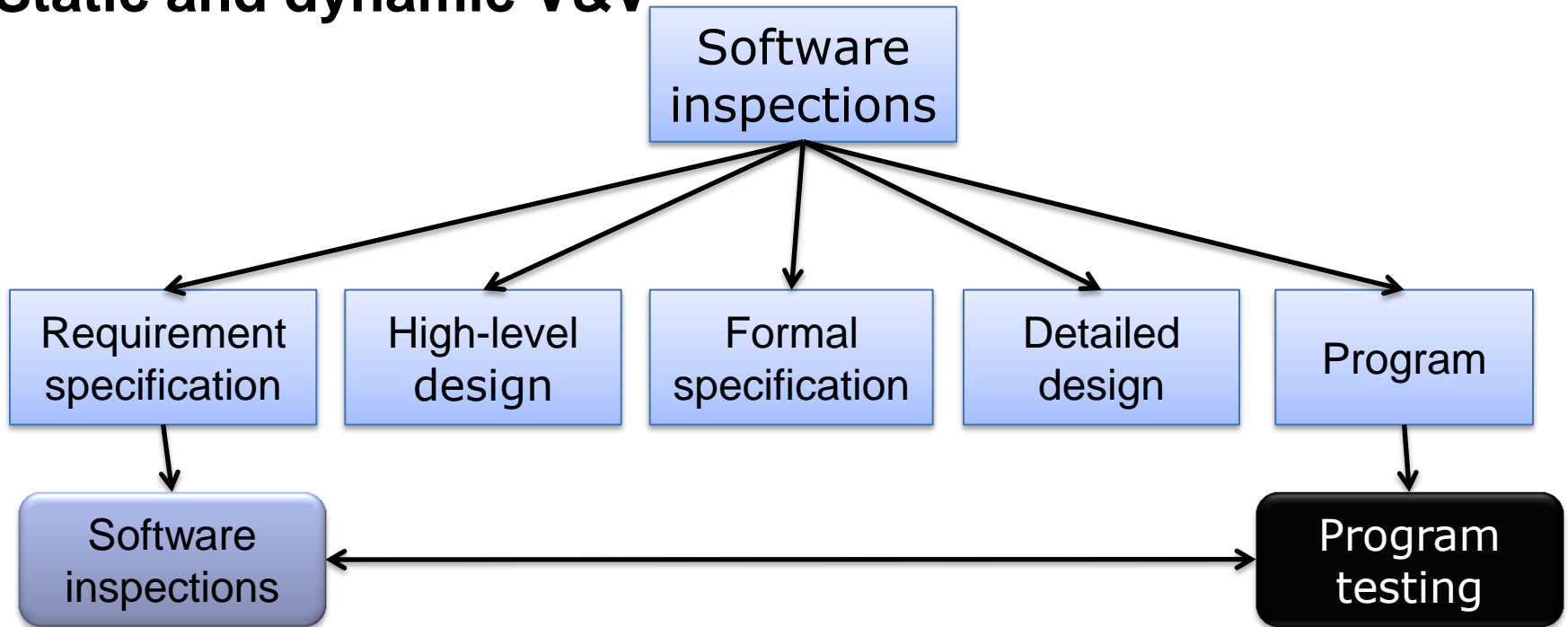
■ Quy trình V&V

- ◆ Quy trình kéo dài toàn bộ chu trình sống
 - V&V phải được áp dụng tại từng bước trong quy trình phần mềm
- ◆ Hai mục tiêu chính
 - Phát hiện các **khiếm khuyết** trong một hệ thống;
 - Đánh giá xem hệ thống có hữu ích và dùng được trong một tình huống vận hành hay không.

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

■ Static and dynamic V&V



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

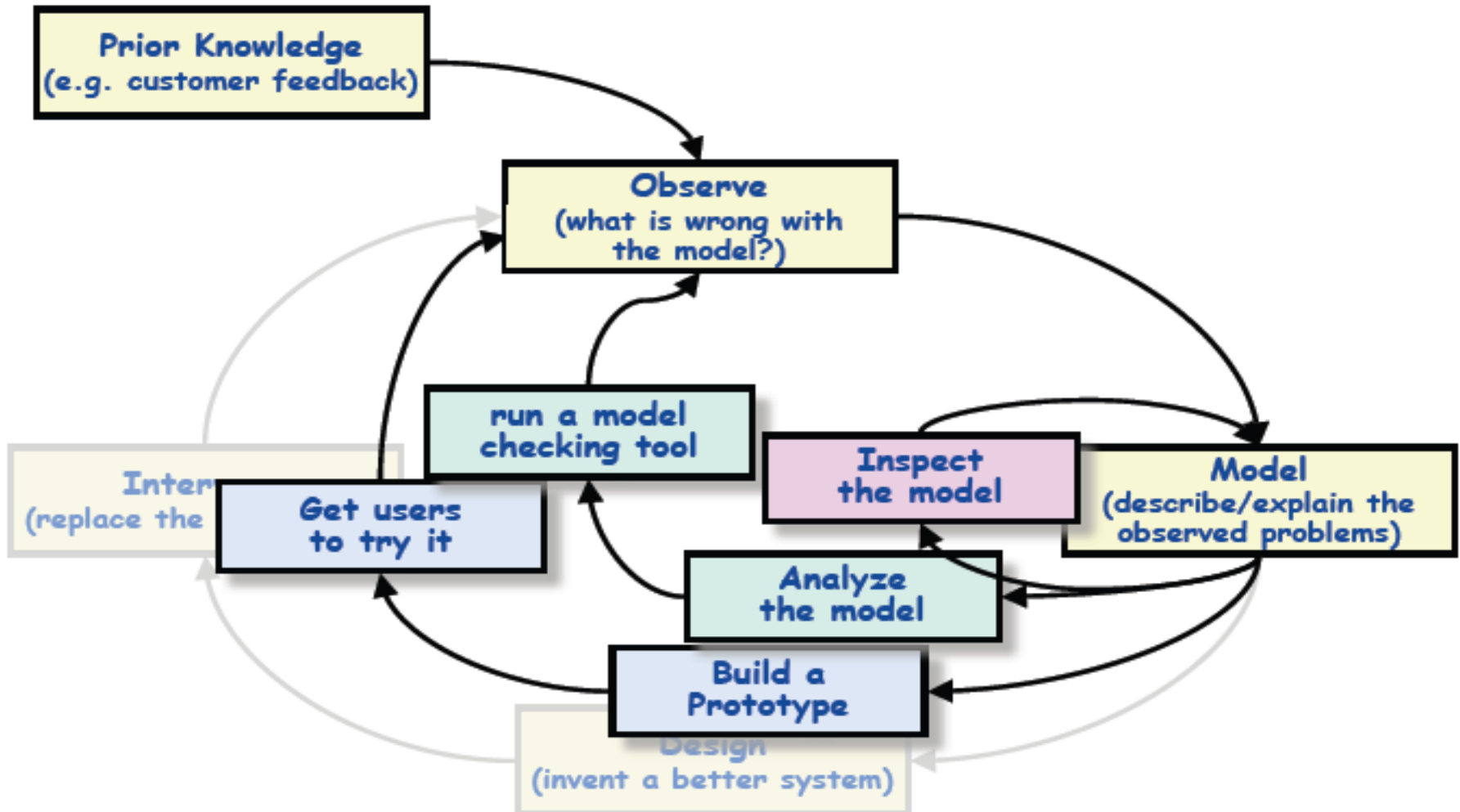
■ Kiểm định

- ◆ Kiểu truyền thống (code verification)
 - Kiểm thử chương trình – testing
 - Duyệt chương trình – inspection, reviews
- ◆ Dựa vào mô hình (model-based verification)
 - Các use case có thỏa mãn yêu cầu không? – goal analysis
 - Mô hình lớp đối tượng có thỏa mãn các use case không? – robustness analysis
 - Mã chương trình có nhất quán với mô hình hay không? – consistency checking

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

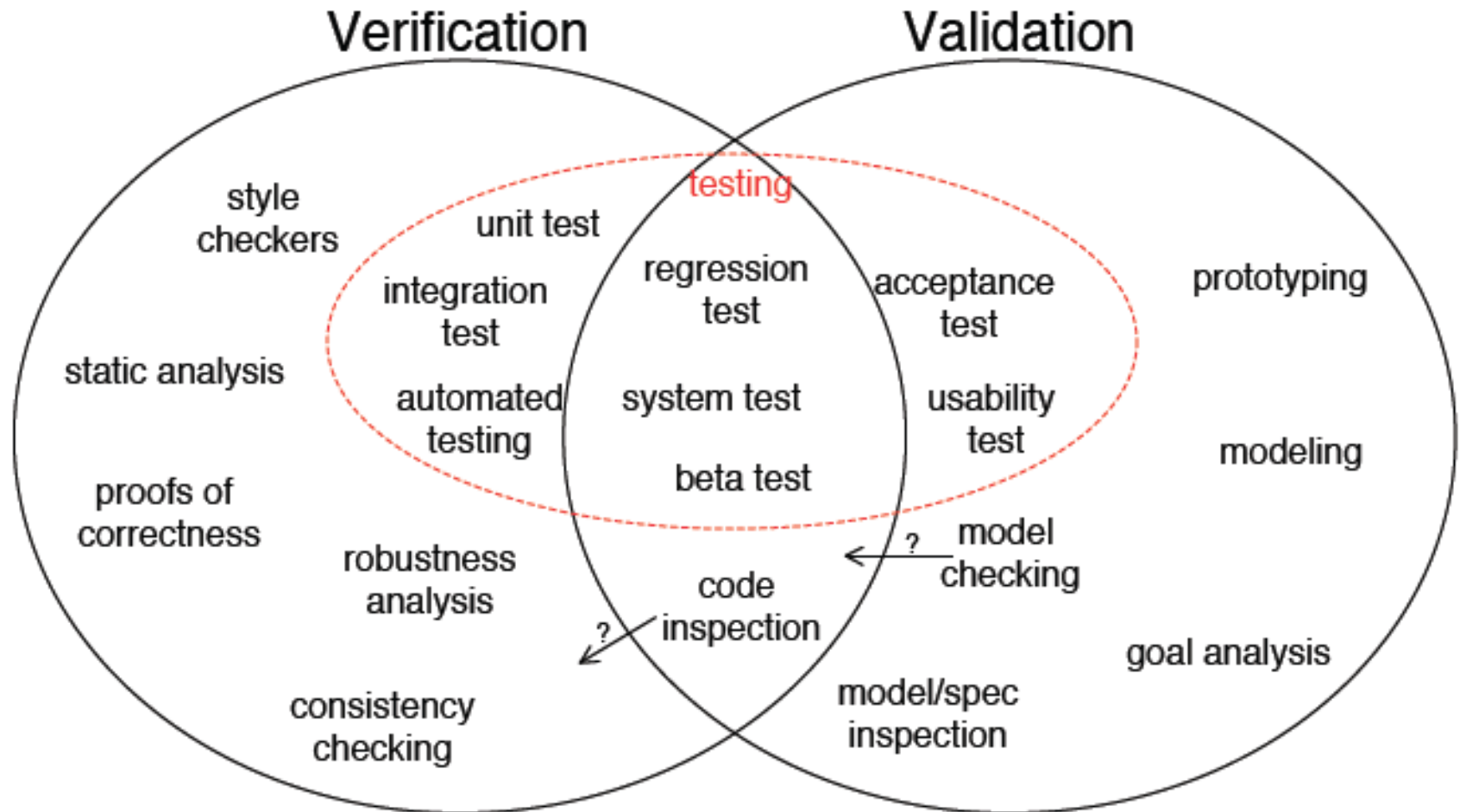
■ Các kĩ thuật thẩm định



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

■ Lựa chọn kĩ thuật



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

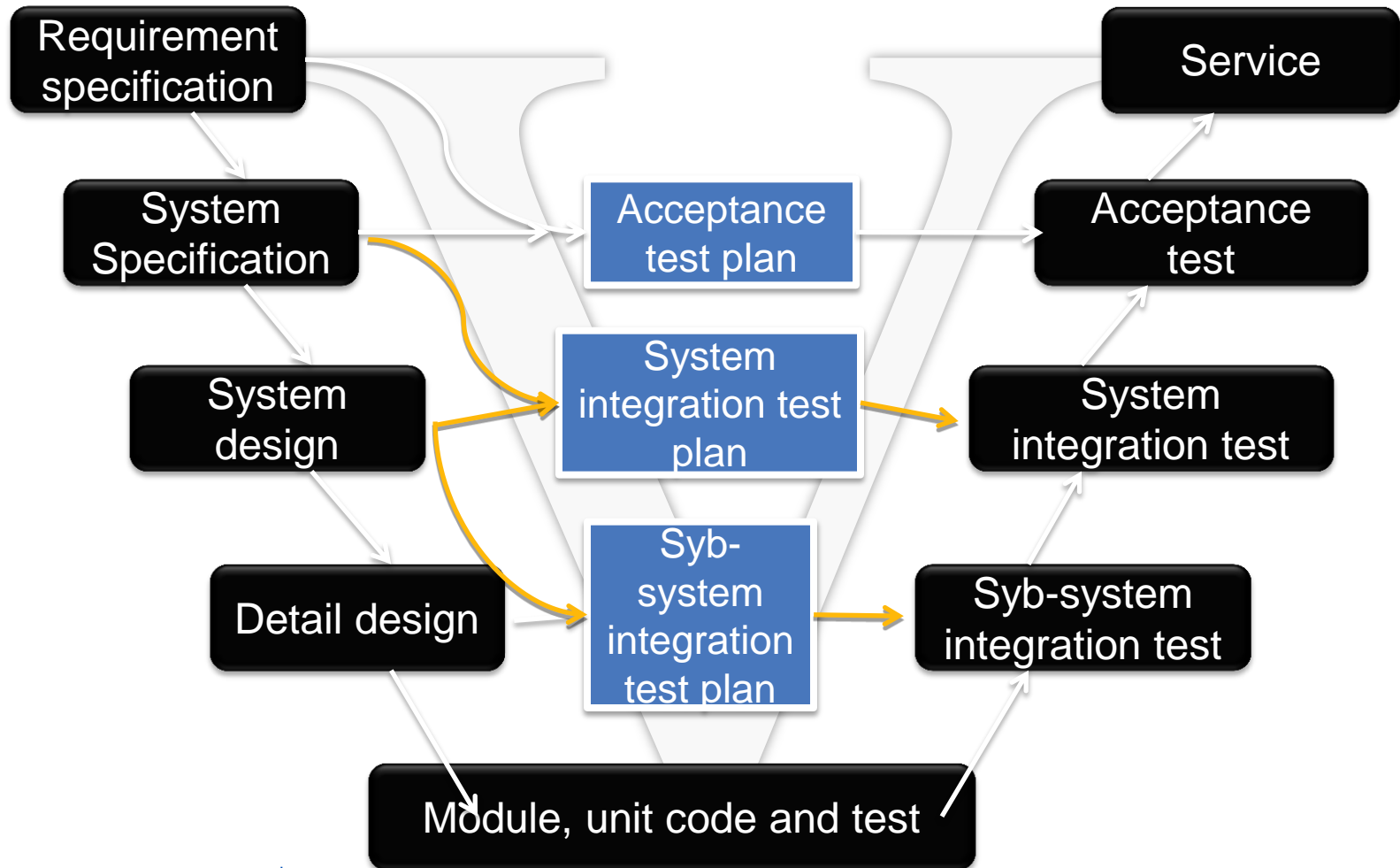
■ Lập kế hoạch V&V:

- ◆ V&V là quy trình rất tốn kém, có thể chiếm đến 50% tổng chi phí
- ◆ Lập kế hoạch tốt để thu được hiệu quả cao nhất của các quy trình kiểm thử (testing) và duyệt (inspection)
- ◆ Cần bắt đầu sớm trong quy trình phát triển.
- ◆ Kế hoạch cần xác định sự cân bằng giữa kiểm thử và duyệt

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.1. Thẩm định và kiểm định

■ The V-model of development



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

■ Cấu trúc của một test plan

- ◆ The testing process.
 - Mô tả các pha của quy trình test
- ◆ Requirements traceability.
 - Đảm bảo từng yêu cầu đều được test
- ◆ Tested items.
 - Liệt kê các sản phẩm quy trình cần được test
- ◆ Testing schedule.
 - Lịch kiểm thử và các tài nguyên cấp phát cho việc kiểm thử
- ◆ Test recording procedures.
 - Quy trình thủ tục để ghi lại quá trình test một cách có hệ thống
- ◆ Hardware and software **requirements**.
 - Các công cụ phần mềm cần dùng và ước lượng về nhu cầu phần cứng
- ◆ **Constraints**.
 - Các hạn chế ảnh hưởng đến việc kiểm thử, chẳng hạn thiếu nhân lực

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.2. Software inspections - Duyệt phần mềm

- Kiểm tra phần mềm để phát hiện bất thường, lỗi, thiếu....
- Áp dụng cho mọi loại biểu diễn của hệ thống
 - ◆ Tài liệu yêu cầu, thiết kế, dữ liệu cấu hình, dữ liệu test, mã nguồn,.....
- Đã được chứng tỏ là kỹ thuật hiệu quả cho việc tìm lỗi chương trình
- Bổ trợ và kết hợp với software testing trong quy trình V&V
 - ◆ Kiểm tra được xem đặc tả có được tuân theo hay không
 - ◆ Không kiểm tra được các tính chất phi chức năng (hiệu năng, độ tin cậy ...)

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

❖ Program inspections - Kiểm tra chương trình

- Cách tiếp cận đã được chuẩn hóa cho việc duyệt tài liệu
- Mục tiêu là để phát hiện khiếm khuyết (không phải để sửa)
- Các khiếm khuyết có thể là các lỗi lô-gic, các bất thường trong mã mà có thể là dấu hiệu của lỗi
 - ◆ Biến chưa khởi tạo
 - ◆ Không theo chuẩn
 - ◆ ...

```
int f() {  
    int x;  
    int y = x *  
2;  
    return y;  
}
```


1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

❖ Program inspections - Kiểm tra chương trình

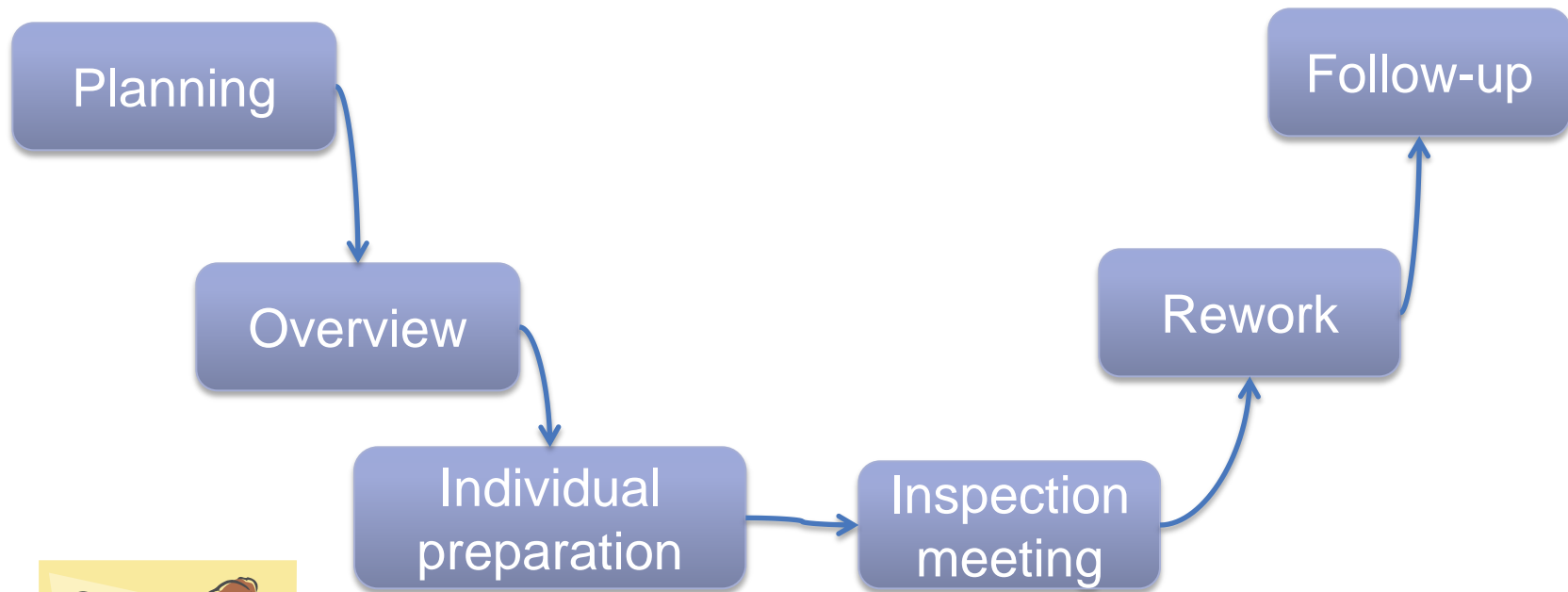
■ Chuẩn bị trước khi kiểm tra

- ◆ Phải có một đặc tả chính xác
- ◆ Đội kiểm tra phải quen với các chuẩn của tổ chức
- ◆ Mã chương trình hoặc tài liệu khác phải đúng cú pháp và đã đủ nội dung
- ◆ Cần chuẩn bị một **error checklist**
- ◆ Quản lý
 - Không dùng các cuộc kiểm tra để khen/chê nhân viên
 - tìm ra ai là thủ phạm

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.3. Program inspections - Kiểm tra chương trình

■ The inspection process



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

❖ Program inspections - Kiểm tra chương trình

■ Inspection procedure - Quy trình kiểm tra

1. Trình bày tổng quan hệ thống cho đội kiểm tra
2. Mã chương trình và các tài liệu có liên quan được giao trước cho đội kiểm tra
3. Thực hiện kiểm tra và ghi lại các lỗi phát hiện được
4. Sửa các lỗi đã phát hiện
5. Một cuộc tái kiểm tra có thể cần hoặc không cần đến

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

❖ Program inspections - Kiểm tra chương trình

Inspection roles

Author or owner	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

❖ Program inspections - Kiểm tra chương trình

■ Inspection checklists

- ◆ Checklist cho các lỗi thường gặp
- ◆ Error checklist phụ thuộc ngôn ngữ lập trình
 - vd. C++: rò rỉ bộ nhớ, con trỏ lạc...
- ◆ Nói chung, ngôn ngữ định kiểu (type checking) càng yếu thì checklist càng dài
 - Java: định kiểu mạnh hơn
 - PHP: định kiểu yếu hơn
- ◆ Ví dụ: khởi tạo, tên hằng, kết thúc vòng lặp, giới hạn mảng, v.v..

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

❖ Program inspections - Kiểm tra chương trình

■ Inspection rate

- ◆ Inspection là quy trình tốn kém
 - Trình bày tổng quan: 500 lệnh/giờ
 - Cá nhân kiểm tra: 125 lệnh/giờ
 - Kiểm tra trong buổi gặp: 90-125 lệnh/giờ
- ◆ Kiểm tra 500 dòng lệnh tốn tổng cộng khoảng 40 giờ làm việc (man/hour)



1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.3. Tự động hóa phân tích tĩnh

- **Static analyser** là các công cụ phần mềm để xử lý mã nguồn dạng text
- **Chúng phân tích text của chương trình để phát hiện các lỗi tiềm tàng**
 - ◆ Errors and warnings
- **Hiệu quả trong việc hỗ trợ inspection**
 - ◆ Hỗ trợ chứ không thay thế

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.3. Tự động hóa phân tích tĩnh

■ Static analysis checks

Các dạng lỗi	Static analysis check
Data faults (Lỗi dữ liệu)	Biến dùng trước khi khởi tạo Biến được khai báo nhưng không dùng Biến được gán liền hai lần mà không dùng đến Có thể vượt ra ngoài mảng Biến chưa khai báo
Control faults (Lỗi điều khiển)	Lệnh không bao giờ chạy đến Rẽ nhánh không điều kiện vào vòng lặp
Input/output faults (Lỗi vào ra dữ liệu)	Biến được output hai lần liên tiếp mà không được gán trị ở giữa
Interface faults (Lỗi giao diện)	Kiểu tham số không khớp Số tham số không khớp không dùng đến kết quả trả về của hàm Hàm và thủ tục không được gọi
Storage management faults (Lỗi quản lý lưu trữ)	Con trỏ không được gán Các phép tính con trỏ

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.3. Tự động hóa phân tích tĩnh

VD: LINT static analysis

```
1. #include <stdio.h>
2. printarray (int Anarray) {
3.     printf("%d",Anarray);
4. }
5. main () {
6.     int Anarray[5];
7.     int i; char c;
8.     printarray (Anarray, i, c);
9.     printarray (Anarray) ;
10. }
```

lint_ex.c(8): warning: c may be used before set

lint_ex.c(8): warning: i may be used before set

printarray: variable # of args. lint_ex.c(2) :: lint_ex.c(8)

printarray, arg. 1 used inconsistently lint_ex.c(2) :: lint_ex.c(8)

printarray, arg. 1 used inconsistently lint_ex.c(2) :: lint_ex.c(9)

printf returns value which is always ignored

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.3. Tự động hóa phân tích tĩnh

■ Sử dụng static analysis

- ◆ Đặc biệt giá trị cho các ngôn ngữ định kiểu yếu (weak typing)
 - Nhiều lỗi trình biên dịch không phát hiện được
 - C, C++
- ◆ Ít hiệu quả hơn cho các ngôn ngữ định kiểu mạnh
 - Trình biên dịch phát hiện được nhiều lỗi
 - Java, C#

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.4. Verification and formal methods

Kiểm định và các phương pháp hình thức

- Các phương pháp hình thức (PPHT) để phát triển phần mềm dựa vào biểu diễn toán học của phần mềm, thường ở dạng đặc tả hình thức (dùng **đặc tả toán học**)
- Các ppht quan tâm đến: phân tích toán học về đặc tả, biến đổi đặc tả thành một biểu diễn chi tiết hơn nhưng tương đương về ngữ nghĩa, chứng minh sự tương đương về ngữ nghĩa
 - ◆ chứng minh rằng một chương trình tuân theo đặc tả toán học của nó

1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.4. Verification and formal methods

Kiểm định và các phương pháp hình thức

$Proc ::=$	$STOP$	
	$SKIP$	
	$e \rightarrow Proc$	(<i>prefixing</i>)
	$Proc \square Proc$	(<i>external choice</i>)
	$Proc \sqcap Proc$	(<i>nondeterministic choice</i>)
	$Proc Proc$	(<i>interleaving</i>)
	$Proc \{X\} Proc$	(<i>interface parallel</i>)
	$Proc \setminus X$	(<i>hiding</i>)
	$Proc; Proc$	(<i>sequential composition</i>)
	if b then $Proc$ else $Proc$	(<i>boolean conditional</i>)
	$Proc \triangleright Proc$	(<i>timeout</i>)
	$Proc \triangle Proc$	(<i>interrupt</i>)

B
Method



$P ::=$	$x(y).P$
	$ \bar{x}\langle y \rangle.P$
	$ P P$
	$ (\nu x)P$
	$!P$
	$ 0$


1. KIỂM ĐỊNH VÀ THẨM ĐỊNH

1.4. Kiểm định và các phương pháp hình thức

■ Ưu/Nhược điểm của các phương pháp hình thức

- ◆ Phát hiện lỗi tại đặc tả yêu cầu
 - Việc tạo một đặc tả toán học đòi hỏi phân tích kĩ càng các yêu cầu
- ◆ Có thể phát hiện lỗi cài đặt trước khi test chương trình
 - Khi chương trình được phân tích cùng với đặc tả
- Khó
 - Cần đến các kĩ pháp chuyên sâu mà các chuyên gia trong miền ứng dụng không hiểu được
- Chi phí cao
 - Chi phí về thời gian và công sức cho việc viết đặc tả
 - Việc chứng minh một chương trình thỏa mãn đặc tả đó còn đắt đỏ hơn nữa





Q & A

- **Thẩm định (validation) và kiểm định (verification) không giống nhau**
 - ◆ Thẩm định chứng tỏ rằng chương trình thỏa mãn nhu cầu khách hàng
 - ◆ Kiểm định chứng tỏ rằng chương trình tuân theo đặc tả
- **Cần thiết kế các kế hoạch test để định hướng cho quy trình kiểm thử**
- **Các kĩ thuật kiểm thử tĩnh đòi hỏi kiểm tra và phân tích chương trình để phát hiện lỗi (không chạy chương trình)**
- **Program inspection rất hiệu quả cho việc phát hiện lỗi**
- **Các công cụ phân tích tĩnh giúp phát hiện các bất thường mà có thể là dấu hiệu của lỗi trong mã nguồn**

Bài tập về nhà

- 1. Phân biệt giữa thẩm định và kiểm định. Giải thích vì sao thẩm định lại là một quy trình đặc biệt khó?**
- 2. Tại sao cần lập kế hoạch test đủ chi tiết để khi tiến hành test có thể thực hiện một cách máy móc và có hệ thống?**
- 3. Lập một checklist gồm những dạng lỗi có thể xảy ra trong các script PHP trong ứng dụng Web+CSDL.**
- 4. Giải thích tại sao tuy các phương pháp hình thức đòi hỏi chi phí cao nhưng khi áp dụng cho những hệ thống đòi hỏi độ an toàn cao thì vẫn được cho là giải pháp có hiệu quả về tài chính.**

2. KIỂM THỬ PHẦN MỀM

■ Mục tiêu:

- ◆ Phân biệt giữa **validation testing** và **defect testing**
- ◆ **Các nguyên lý** của kiểm thử hệ thống và kiểm thử thành phần
- ◆ Các chiến lược **sinh** test case hệ thống

2. KIỂM THỬ PHẦN MỀM

■ Nội dung

1. System testing – kiểm thử hệ thống
2. Component testing – kiểm thử thành phần
3. Test case design – thiết kế test case
4. Test automation – tự động hóa kiểm thử

2. KIỂM THỬ PHẦN MỀM

2.1. Quy trình kiểm thử

■ Component testing – kiểm thử thành phần

- ◆ Kiểm thử từng thành phần của chương trình
- ◆ Trách nhiệm của người phát triển thành phần
- ◆ Các test được rút ra từ kinh nghiệm của người phát triển

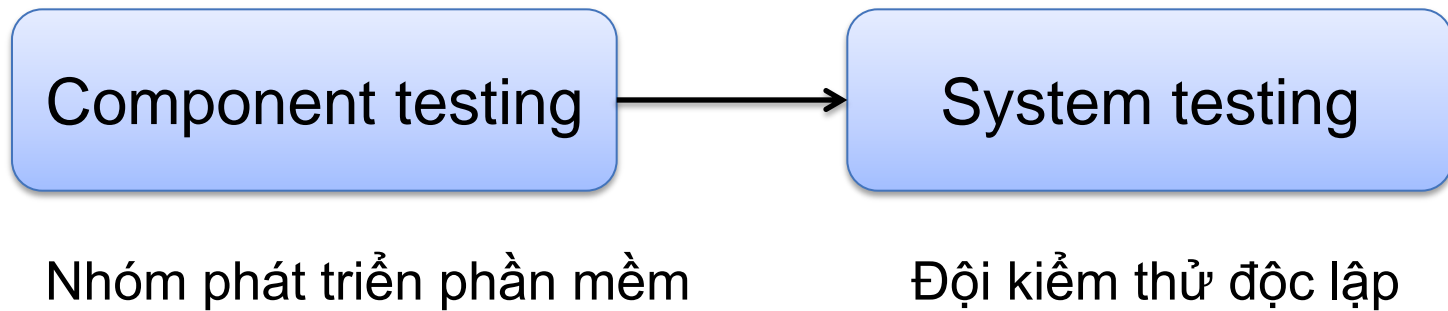
■ System testing – kiểm thử hệ thống

- ◆ Kiểm thử các nhóm thành phần được tích hợp để tạo nên một hệ thống hoặc một hệ thống con
- ◆ Trách nhiệm của một đội kiểm thử độc lập
- ◆ Các test dựa trên đặc tả hệ thống

2. KIỂM THỬ PHẦN MỀM

2.1. Quy trình kiểm thử

■ Các pha kiểm thử



2. KIỂM THỬ PHẦN MỀM

2.1. Quy trình kiểm thử

■ Mục tiêu các quy trình kiểm thử

■ Validation testing (test thẩm định)

- ◆ Để chứng tỏ rằng phần mềm thỏa mãn các yêu cầu
- ◆ Một test thành công là test cho thấy hệ thống hoạt động như trông đợi

■ Defect testing (test lỗi)

- ◆ Để phát hiện lỗi hoặc khiếm khuyết của phần mềm khi nó hoạt động sai
 - Không tuân theo đặc tả
- ◆ Một test thành công là test cho thấy hệ thống hoạt động không đúng
 - Làm lộ ra một khiếm khuyết của hệ thống

2. KIỂM THỬ PHẦN MỀM

2.1. Quy trình kiểm thử

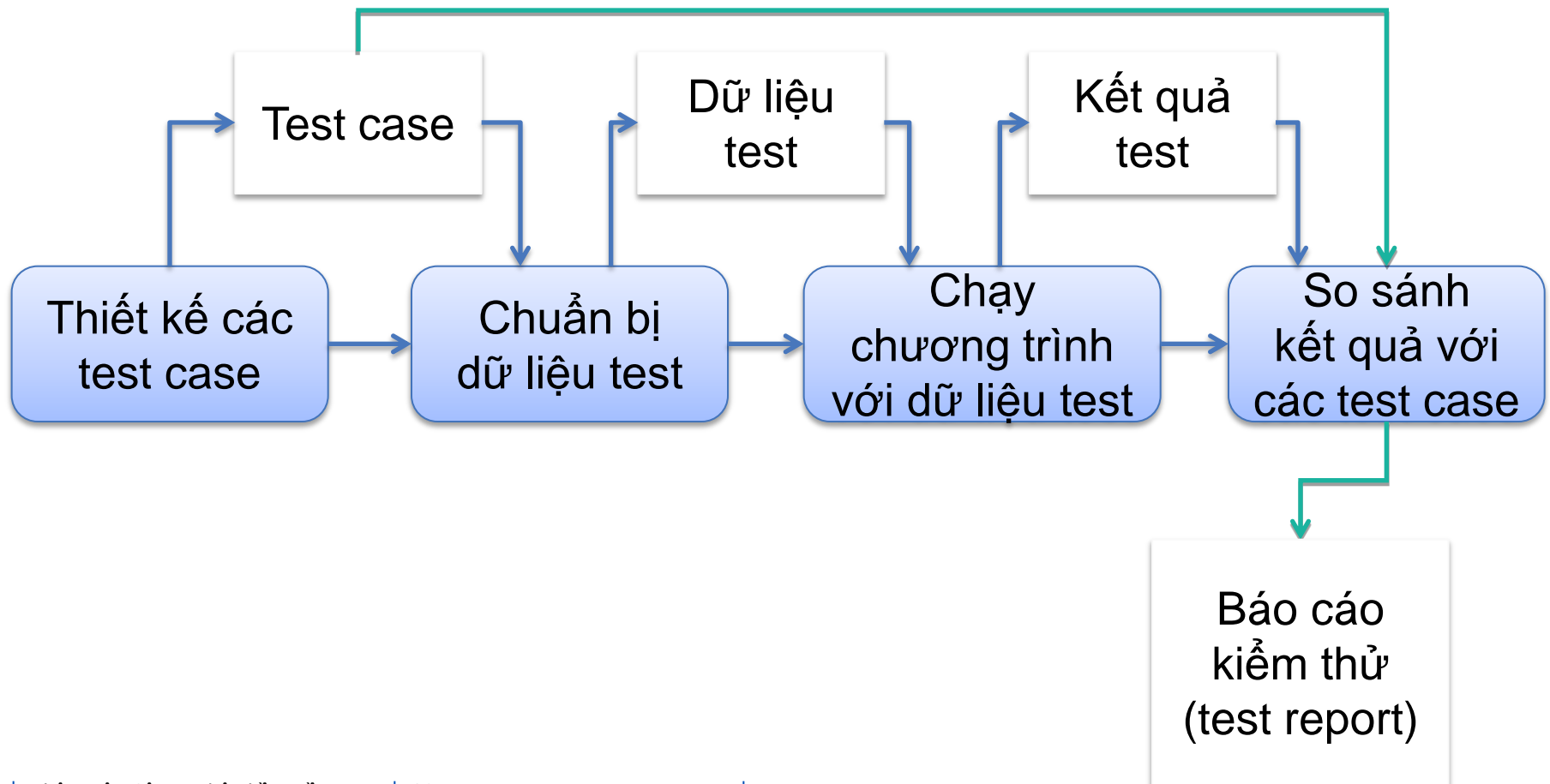
■ Defect testing

- ◆ Mục tiêu của defect testing là tìm các khiếm khuyết (defect) trong chương trình
- ◆ Một defect test thành công là test làm cho chương trình cư xử một cách bất thường
- ◆ Các test nhằm chứng tỏ sự có mặt của các khiếm khuyết chứ không thể chứng minh rằng không có khiếm khuyết

2. KIỂM THỬ PHẦN MỀM

2.1. Quy trình kiểm thử

■ Quy trình



2. KIỂM THỬ PHẦN MỀM

2.1. Quy trình kiểm thử

■ Testing policies - Chính sách kiểm thử

- ◆ Chỉ có kiểm thử toàn diện vét cạn tất cả các trường hợp thực thi (exhaustive testing) mới có thể chứng tỏ rằng một chương trình không có lỗi
 - Kiểm thử toàn diện là **bất khả thi**
- ◆ Thực tế, việc kiểm thử chỉ thực hiện với một tập con của tập tất cả các trường hợp có thể xảy ra.
- ◆ Chính sách kiểm thử xác định phương pháp chọn các test hệ thống. Ví dụ
 - Test tất cả các chức năng truy nhập được từ các menu;
 - Test các tổ hợp chức năng truy nhập được từ cùng một menu;
 - Ở đầu cần đến input của người dùng, test tất cả các chức năng với cả input sai và input đúng.

2. KIỂM THỬ PHẦN MỀM

2.2. System testing – kiểm thử hệ thống

- Tích hợp các thành phần để tạo một hệ thống hoặc hệ thống con rồi test hệ thống đó
- Trong quy trình phát triển lặp, test hệ thống là test từng bản increment để giao cho khách hàng. Trong quy trình thác nước, kiểm thử hệ thống là test toàn bộ hệ thống.
- Hai pha của kiểm thử hệ thống:
 - ◆ **Integration testing – kiểm thử tích hợp.** Đội kiểm thử có thể truy nhập mã nguồn. Còn hệ thống được test khi bổ sung các thành phần. Đội tích hợp xác định lỗi thuộc thành phần nào để debug.
 - ◆ **Release testing – kiểm thử bản release.** Đội kiểm thử test hệ thống chuẩn bị giao cho khách hàng, test để kiểm tra xem có thỏa mãn yêu cầu không, thực hiện theo kiểu black-box

2. KIỂM THỬ PHẦN MỀM

2.2. System testing – kiểm thử hệ thống

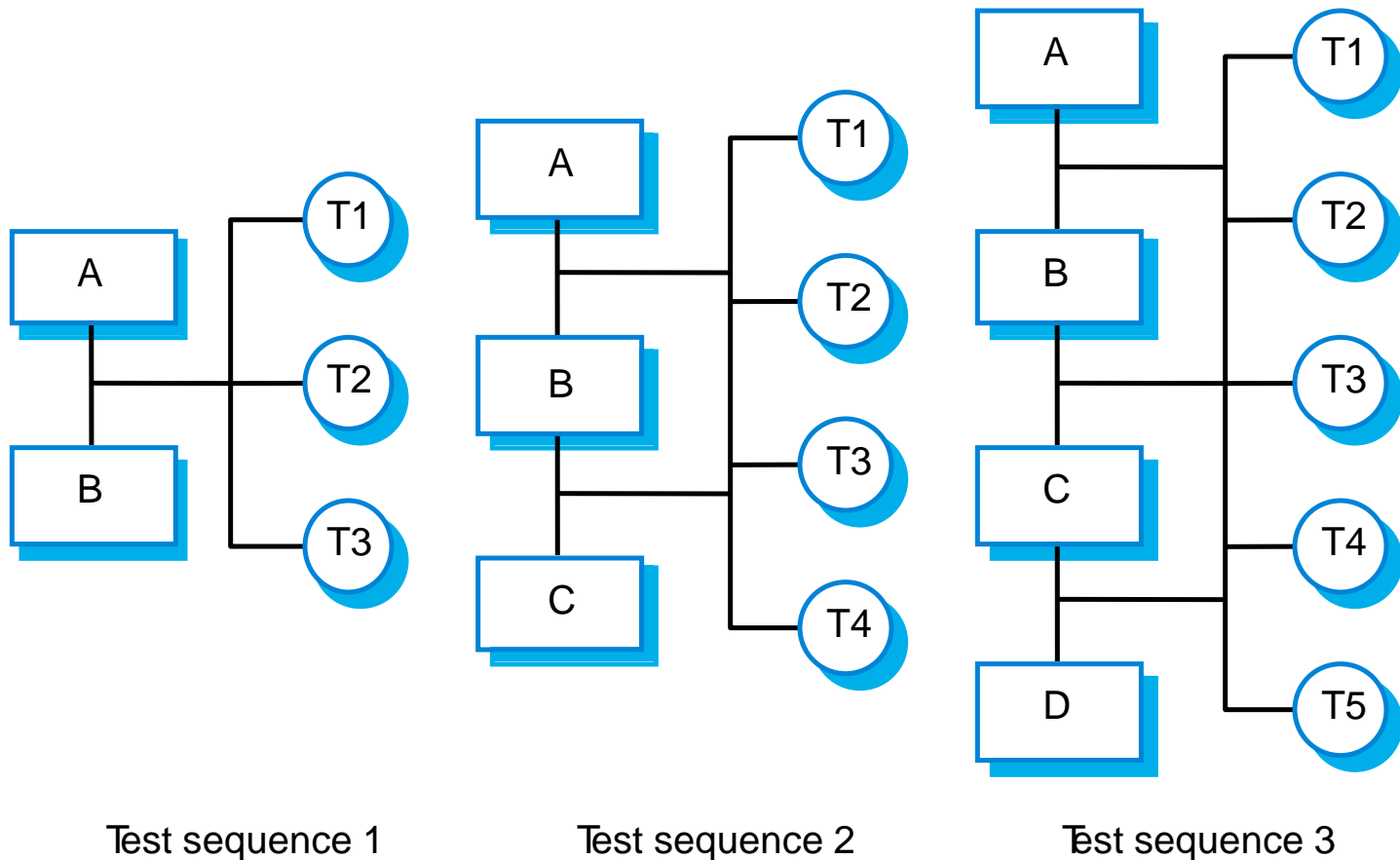
■ Integration testing

- ◆ Xây dựng một hệ thống từ các thành phần của nó và test xem có vấn đề nào nảy sinh khi các thành phần tương tác với nhau hay không
- ◆ Top-down integration
 - Phát triển khung hệ thống rồi lắp các thành phần vào
- ◆ Bottom-up integration
 - Tích hợp các thành phần cơ sở hạ tầng (mạng, cơ sở dữ liệu...) rồi thêm các thành phần chức năng
- ◆ Để đơn giản hóa việc định vị lỗi, cần tích hợp hệ thống một cách tăng dần

2. KIỂM THỬ PHẦN MỀM

2.2. System testing – kiểm thử hệ thống

■ Incremental integration testing - kiểm thử tích hợp tăng dần



2. KIỂM THỬ PHẦN MỀM

2.2. System testing – kiểm thử hệ thống

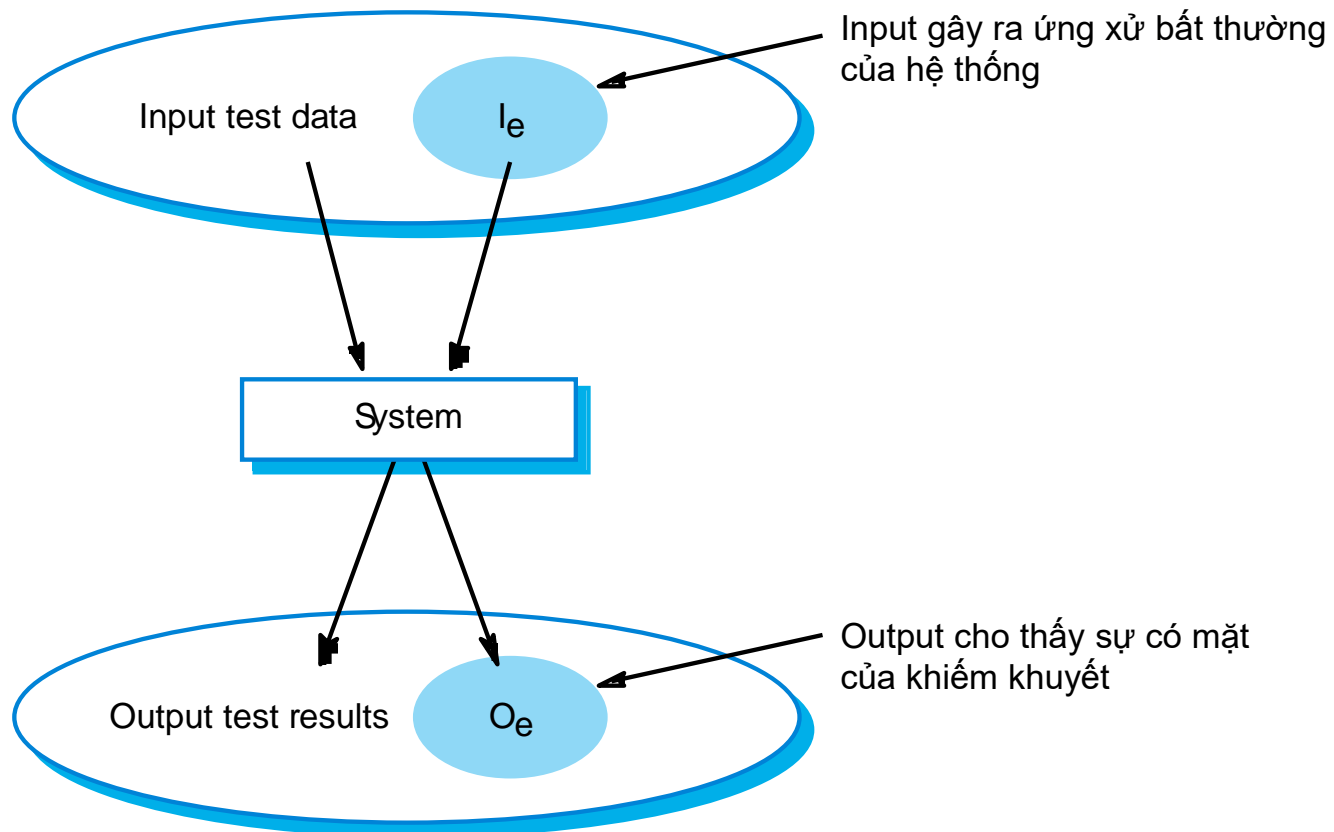
■ Release testing

- ◆ Kiểm thử một bản release của một hệ thống mà sẽ được giao cho khách hàng
- ◆ Mục đích là để chắc chắn hơn rằng hệ thống thỏa mãn các yêu cầu
- ◆ Release testing thường là kiểm thử hộp đen
 - Chỉ dựa vào đặc tả hệ thống
 - Người test không biết gì về cài đặt hệ thống
- ◆ Còn gọi là **kiểm thử chức năng** vì chỉ liên quan đến các chức năng của hệ thống
- ◆ Nếu có khách hàng tham gia kiểm thử thì còn gọi là **acceptance test**

2. KIỂM THỬ PHẦN MỀM

2.2. System testing – kiểm thử hệ thống

■ Black-box testing - kiểm thử hộp đen



2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Làm cách nào để các khiếm khuyết lộ ra?

- ◆ Chọn các input nhằm buộc hệ thống phải sinh tất cả các thông báo lỗi
- ◆ Thiết kế các input gây tràn bộ đệm dành cho input
- ◆ Lặp một input hoặc một chuỗi input nhiều lần
- ◆ Buộc hệ thống sinh output không hợp lệ
- ◆ Buộc kết quả tính toán trở thành quá lớn hoặc quá nhỏ

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Testing scenario – kịch bản test

Một sinh viên ở Scotland đang học lịch sử Mỹ và cần phải viết một báo cáo về ‘Tâm lý tiền phương ở vùng viễn Tây Mỹ trong giai đoạn 1840-1880’. Để làm việc này, cô cần tìm tài liệu ở một loạt các thư viện. Cô log vào hệ thống LIBSYS và dùng tiện ích tìm kiếm để xem mình có thể truy cập các tài liệu nguyên gốc từ thời đó hay không. Cô tìm thấy nguồn ở nhiều thư viện đại học Mỹ và download bản sao của một số tài liệu. Tuy nhiên, có một tài liệu mà cô cần phải xin chứng nhận của trường đại học cô đang học rằng cô là sinh viên và sẽ không sử dụng cho mục đích thương mại. Cô sinh viên dùng tiện ích của LIBSYS để yêu cầu chứng nhận đó và đăng kí yêu cầu của mình. Nếu được chấp nhận, tài liệu sẽ được download về server của thư viện đã đăng ký rồi in tại đó cho cô. Cô nhận được một thông điệp từ LIBSYS nói rằng cô sẽ nhận được một email khi cô có thể đến nhận bản in tài liệu đó.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ System tests – các test hệ thống

1. Test cơ chế đăng nhập bằng các lần đăng nhập đúng và sai để kiểm tra việc người dùng hợp lệ được chấp nhận và người dùng không hợp lệ bị từ chối.
2. Test tiện ích tìm kiếm bằng các truy vấn khác nhau về các nguồn đã biết để kiểm tra xem cơ chế tìm kiếm có thực sự tìm thấy tài liệu không.
3. Test chức năng trình bày của hệ thống để kiểm tra xem thông tin về các tài liệu có được hiển thị đúng đắn không.
4. Test cơ chế xin phép tải xuống.
5. Test phản ứng bằng e-mail khi thông báo rằng tài liệu được download đã sẵn sàng để dùng.

2. KIỂM THỬ PHẦN MỀM

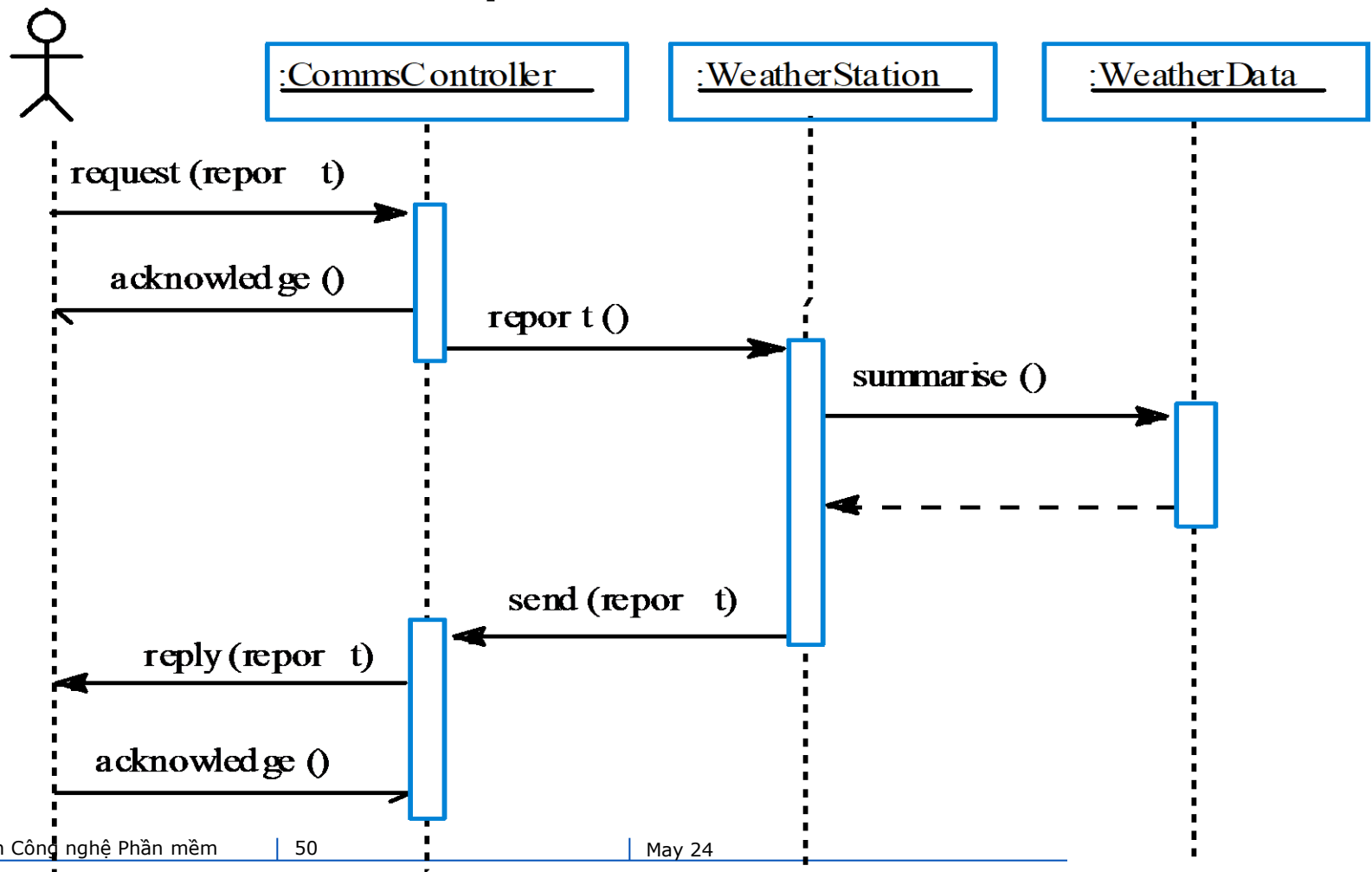
2.3. Các hướng dẫn kiểm thử

■ Use cases

- ◆ Có thể dùng các use case làm cơ sở sinh các test cho một hệ thống.
 - Giúp xác định các thao tác cần test
 - Giúp thiết kế các test case cần thiết
- ◆ Từ một biểu đồ tuần tự liên quan, có thể xác định các input và output cần tạo

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử Collect weather data sequence chart



2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Performance testing - kiểm thử hiệu năng

- ◆ Trong release testing có thể cần test cả các tính chất phát sinh của hệ thống
 - Hiệu năng, độ tin cậy...
- ◆ Performance test thường gồm một chuỗi các test mà trong đó tải hệ thống được tăng dần cho đến khi hiệu năng hệ thống trở nên không thể chấp nhận được
 - Stress testing

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Stress testing

- ◆ Chạy hệ thống vượt quá tải tối đa theo thiết kế.
 - Làm lộ diện các khiếm khuyết mà ở điều kiện bình thường có thể không phát hiện ra được
- ◆ Đặc biệt liên quan đến các hệ phân tán
 - Hiệu năng giảm trầm trọng khi mạng trở nên quá tải

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Component testing

- ◆ Component testing hoặc unit testing là quy trình kiểm thử từng thành phần một cách độc lập
- ◆ Là một quy trình defect testing
- ◆ Các thành phần có thể là:
 - Từng hàm hoặc phương thức bên trong một đối tượng;
 - Các lớp đối tượng với một vài thuộc tính và phương thức;
 - Composite component (thành phần phức hợp) với các interface được định nghĩa sẵn dùng để truy nhập các chức năng của chúng.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Function/method testing

- ◆ Dạng component đơn giản nhất
- ◆ Bộ test là một tập các lời gọi tới các hàm này với các tham số khác nhau
- ◆ Có thể dùng cách thiết kế test case sẽ được nói đến trong mục sau

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Object class testing

- ◆ Một phủ test hoàn chỉnh cho một lớp đối tượng bao gồm
 - Test độc lập tất cả các thao tác của một đối tượng;
 - Gán và truy vấn giá trị tất cả các thuộc tính đối tượng;
 - Chạy thử đối tượng trong tất cả các trạng thái có thể. Nghĩa là giả lập tất cả các sự kiện có thể làm đối tượng chuyển trạng thái
- ◆ Thừa kế gây khó khăn cho việc thiết kế test cho lớp đối tượng. Khi test một lớp con, phải test cả những thao tác được thừa kế từ lớp cha.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

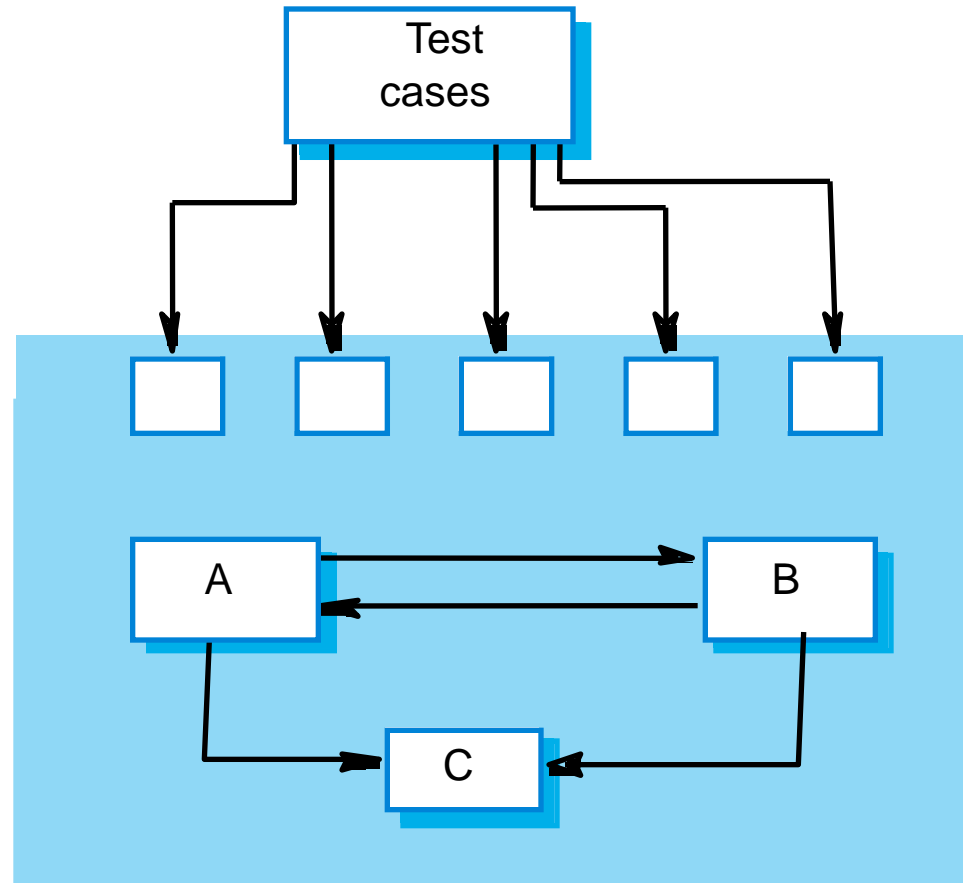
■ Interface testing

- ◆ Kiểm thử các composite component là để kiểm tra xem giao diện component có hoạt động theo đặc tả hay không.
- ◆ Nhằm phát hiện lỗi do sai giao diện hoặc giả thiết không hợp lệ về giao diện
- ◆ Đặc biệt quan trọng cho việc phát triển hướng đối tượng khi các đối tượng được định nghĩa bởi các giao diện của chúng

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Interface testing



2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Các dạng interface

- ◆ Giao diện tham số - Parameter interfaces
 - Dữ liệu được truyền qua lại giữa các component khác nhau
- ◆ Giao diện bộ nhớ dùng chung - Shared memory interfaces
 - Các component dùng chung một khối bộ nhớ
- ◆ Giao diện thủ tục - Procedural interfaces
 - Một component đóng gói một loạt các thủ tục mà các component khác có thể gọi. Các đối tượng và các component tái sử dụng có dạng interface này
- ◆ Giao diện truyền thông điệp - Message passing interfaces
 - Một component yêu cầu dịch vụ từ một component khác bằng cách gửi thông điệp, thông điệp trả về sẽ chứa kết quả thực hiện dịch vụ. Một số hệ thống hướng đối tượng cũng như client-server có dạng interface này

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Các lỗi interface

◆ Dùng sai

- Ví dụ. Sai kiểu tham số, thứ tự tham số

◆ Hiểu sai

- Một thành phần gọi hiểu sai đặc tả giao diện của thành phần được gọi, dẫn đến cư xử của thành phần được gọi không như trông đợi của thành phần gọi. Ví dụ gọi tìm kiếm nhị phân với mảng chưa sắp xếp

◆ Lỗi thời gian

- Thường gặp ở hệ thống thời gian thực, khi các component dùng message-passing hoặc shared-memory interface. Lỗi khi data producer và data consumer không đồng bộ gây ra chuyện consumer sử dụng thông tin lỗi thời chưa kịp update

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Interface testing guidelines

- ◆ Tham số tại các cực điểm của khoảng xác định
- ◆ Dùng con trỏ null để test các tham số con trỏ
- ◆ Với procedural interface, thiết kế các test nhằm làm cho component bị thất bại
- ◆ Dùng stress testing trong các hệ thống truyền thông điệp -> có thể làm lộ các vấn đề về thời gian
- ◆ Trong các hệ thống dùng chung bộ nhớ, thay đổi thứ tự mà các component bị kích hoạt -> có thể làm lộ ngộ nhận của lập trình viên về thứ tự tạo và dùng dữ liệu dùng chung.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Test case design

- ◆ Thiết kế các test case (gồm input và expected output) dùng để test hệ thống
- ◆ Mục tiêu là tạo các bộ test có hiệu lực trong validation testing và defect testing
- ◆ Các cách tiếp cận:
 - Requirements-based testing – test yêu cầu;
 - Partition testing – test phân hoạch;
 - Structural testing – test cấu trúc.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Requirements based testing

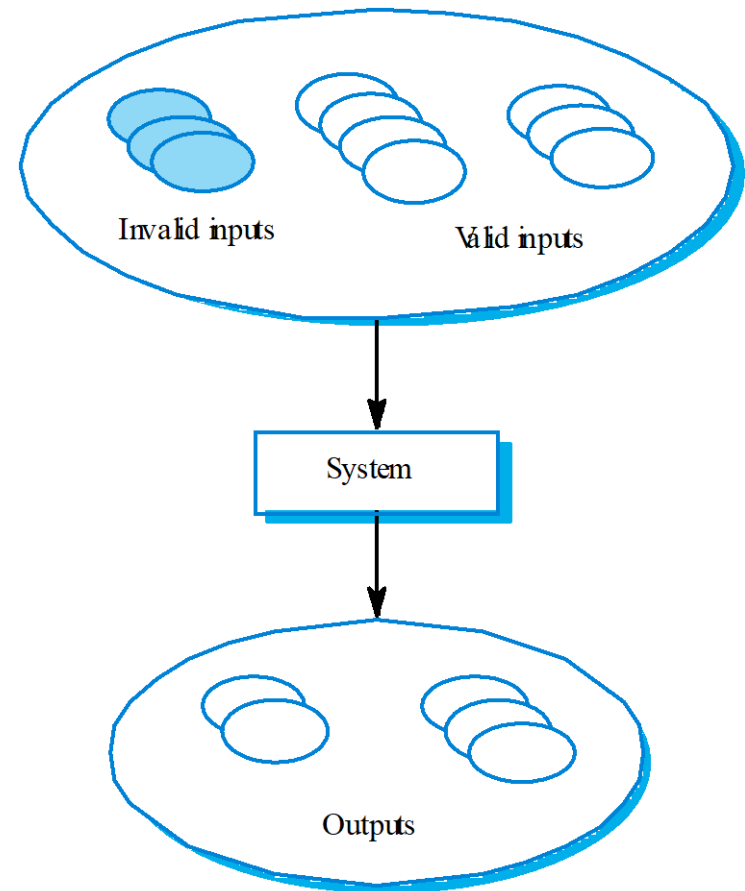
- ◆ Một nguyên tắc chung của kỹ nghệ yêu cầu là các yêu cầu đều phải test được
- ◆ Requirements-based testing là một kỹ thuật validation testing mà trong đó bạn xét từng yêu cầu và xác định một tập các test case nhằm chứng tỏ rằng hệ thống thỏa mãn yêu cầu đó.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Partition testing

- ◆ Input và output thường tập trung thành các nhóm, mỗi nhóm chứa các thành viên có tính chất giống nhau.
- ◆ Mỗi nhóm đó là một **phân hoạch tương đương (equivalence partition)** hoặc miền (domain) mà với mỗi điểm trong đó chương trình hoạt động gần như nhau
- ◆ Cần chọn test case cho từng phân hoạch

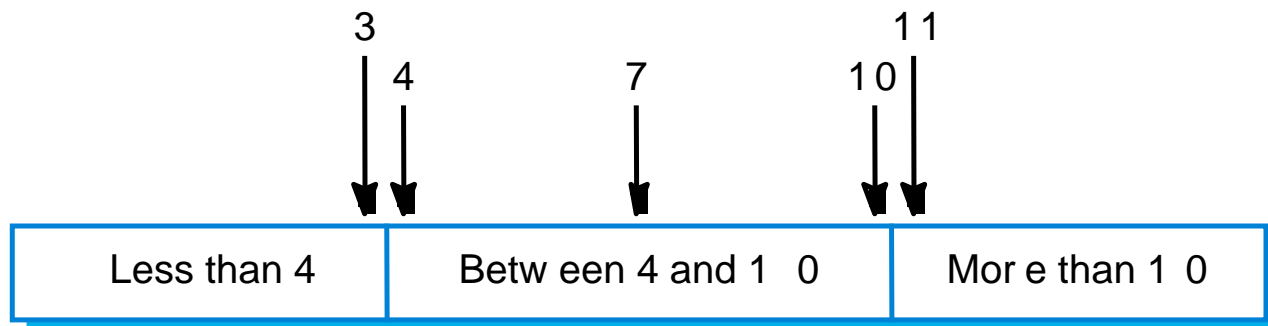


equivalence partition

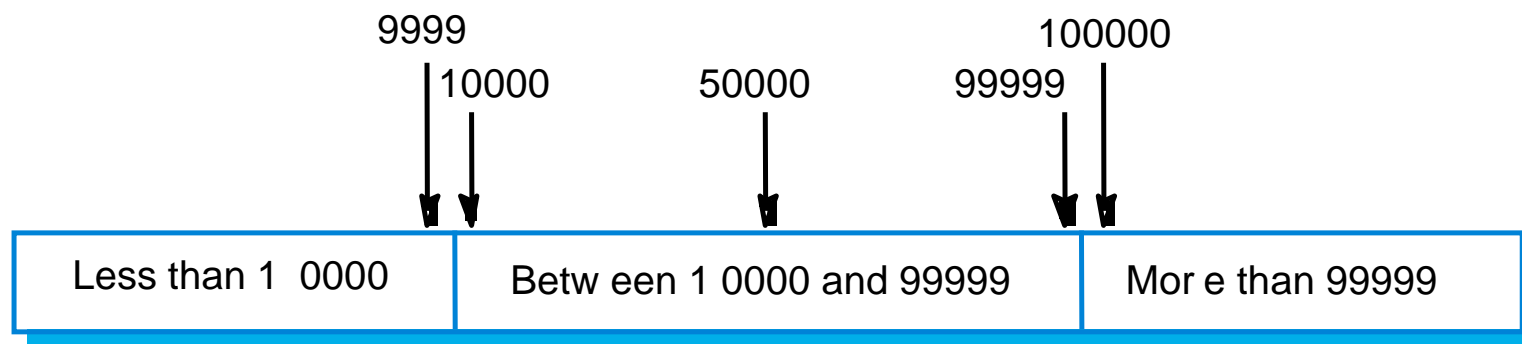
2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Equivalence partitions



Number of input values



Input values

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Search routine specification

```
procedure Search (Key : ELEM ; T: SEQ of ELEM;  
    Found : in out BOOLEAN; L: in out ELEM_INDEX) ;
```

Pre-condition

```
-- the sequence has at least one element  
T'FIRST <= T'LAST
```

Post-condition

```
-- the element is found and is referenced by L  
( Found and T (L) = Key)
```

or

```
-- the element is not in the array  
( not Found and  
not (exists i, T'FIRST <= i <= T'LAST, T (i) = Key ))
```

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Search routine - input partitions

- ◆ Các input thỏa mãn pre-condition
- ◆ Các input không thỏa mãn pre-condition
- ◆ Các input có key là phần tử của mảng
- ◆ Các input có key không phải là phần tử của mảng

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Hướng dẫn kiểm thử (chuỗi)

- ◆ Test phần mềm với các chuỗi chỉ có đúng 1 phần tử
- ◆ Dùng các chuỗi có độ dài khác nhau
- ◆ Tạo các test sao cho các phần tử đầu tiên, cuối dùng, và ở giữa chuỗi được dùng đến
- ◆ Test với các chuỗi rỗng

2. KIỂM THỬ PHẦN MỀM

■ Search routine - input partitions

Sequence	Element
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Sequence	Element	Sequence
Input sequence (T)	Key (Key)	Output (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

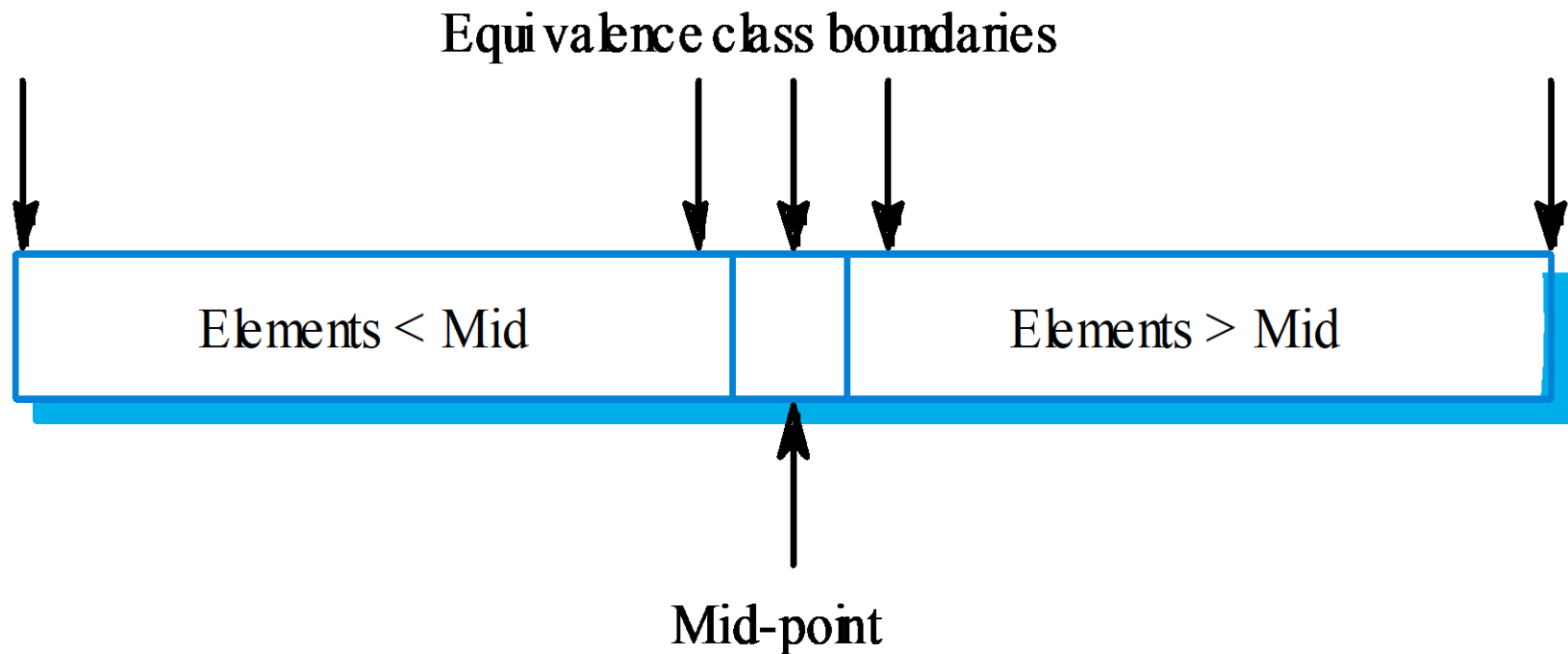
■ Binary search - equiv. partitions

- ◆ Pre-conditions satisfied, key element in array.
- ◆ Pre-conditions satisfied, key element not in array.
- ◆ Pre-conditions unsatisfied, key element in array.
- ◆ Pre-conditions unsatisfied, key element not in array.
- ◆ Input array has a single value.
- ◆ Input array has an even number of values.
- ◆ Input array has an odd number of values.

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Binary search equiv. partitions



2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Binary search - test cases

Input array (T)	Key (Key)	Output (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

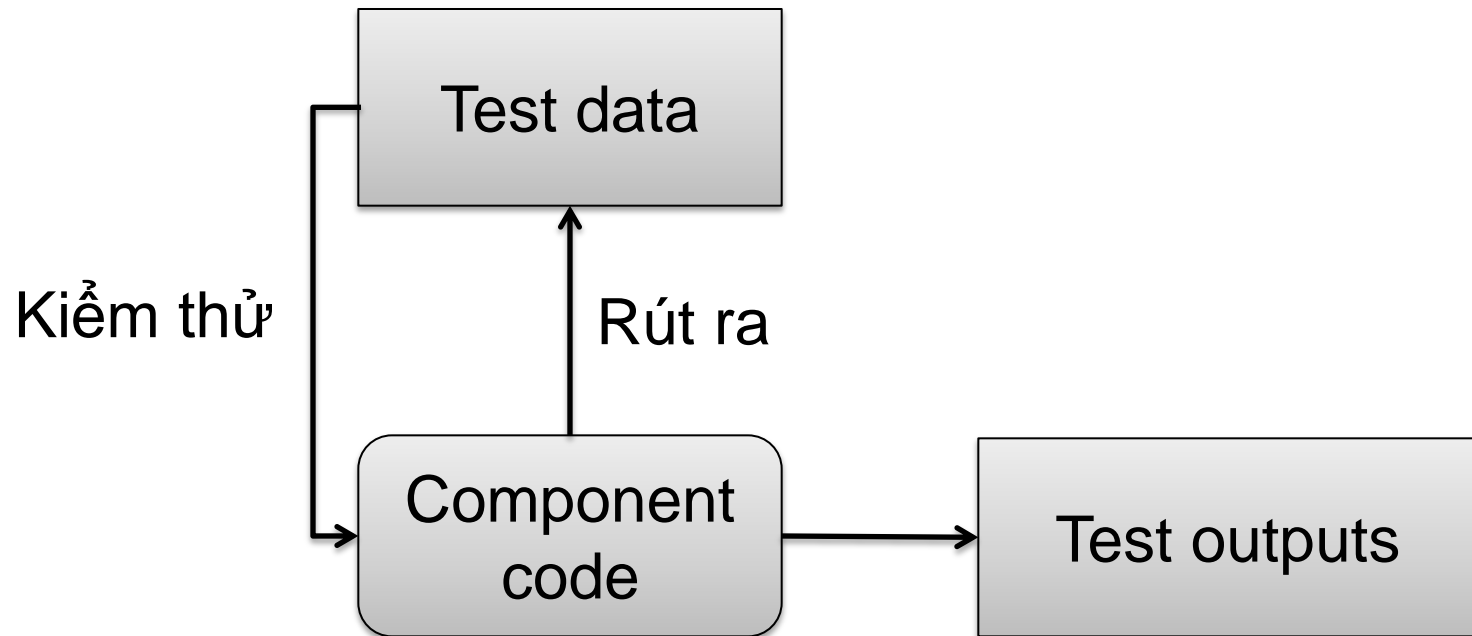
■ Structural testing

- ◆ Đôi khi gọi là test hộp trắng (white-box testing)
- ◆ Tạo các test case theo cấu trúc chương trình
- ◆ Mục tiêu là để chạy tất cả các lệnh trong chương trình (không phải tất cả các tổ hợp đường chạy (path)).

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Structural testing



2. KIỂM THỬ PHẦN MỀM

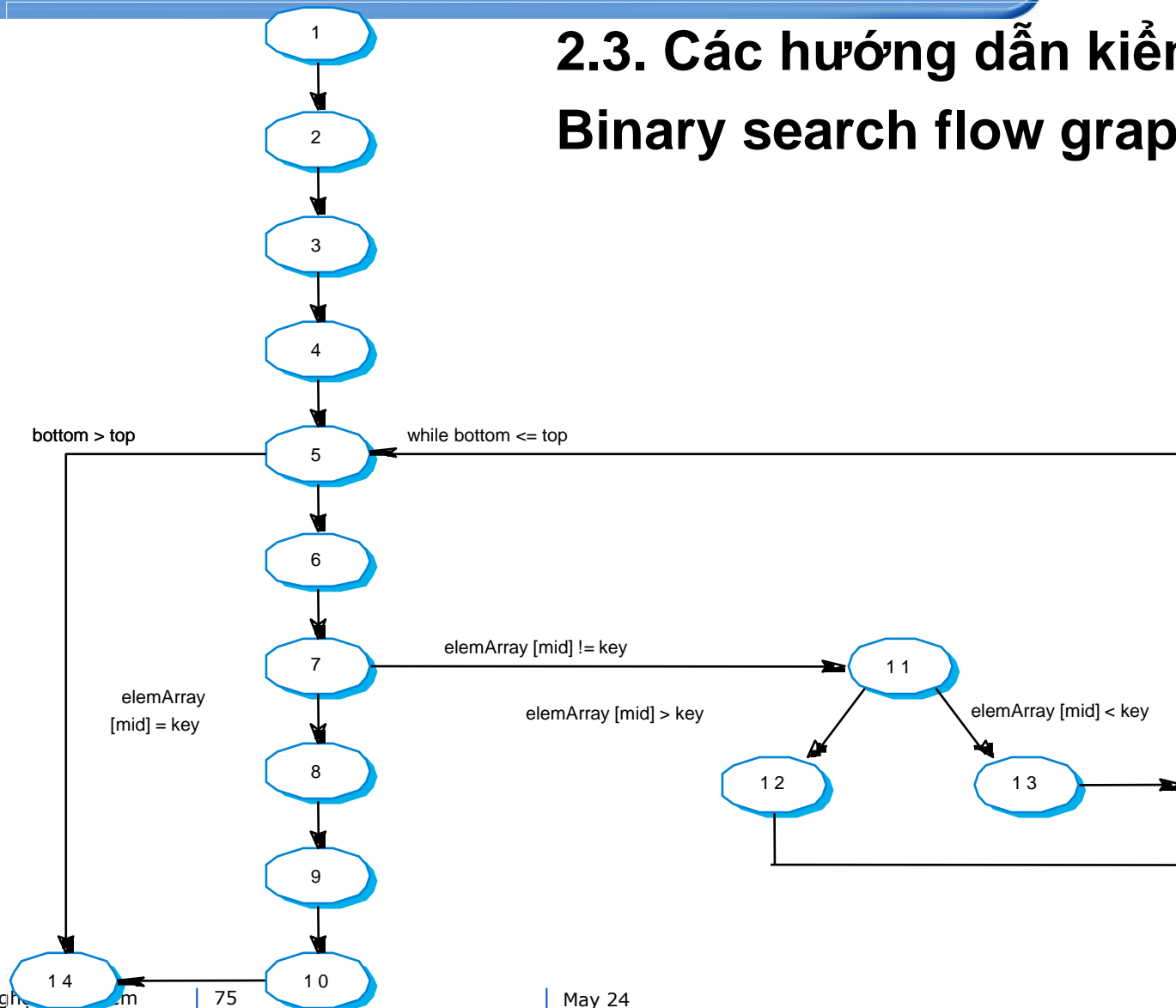
2.3. Các hướng dẫn kiểm thử

■ Path testing

- ◆ Mục tiêu của path testing là đảm bảo rằng tập các test case đủ để mỗi đường chạy (path) của chương trình đều được thực hiện ít nhất một lần.
- ◆ Xuất phát từ sơ đồ luồng điều khiển (flow graph) với các nút biểu diễn quyết định, các cung biểu diễn luồng điều khiển.
- ◆ Các câu lệnh điều kiện trở thành các nút trong sơ đồ luồng điều khiển.

2. KIỂM THỬ PHẦN MỀM

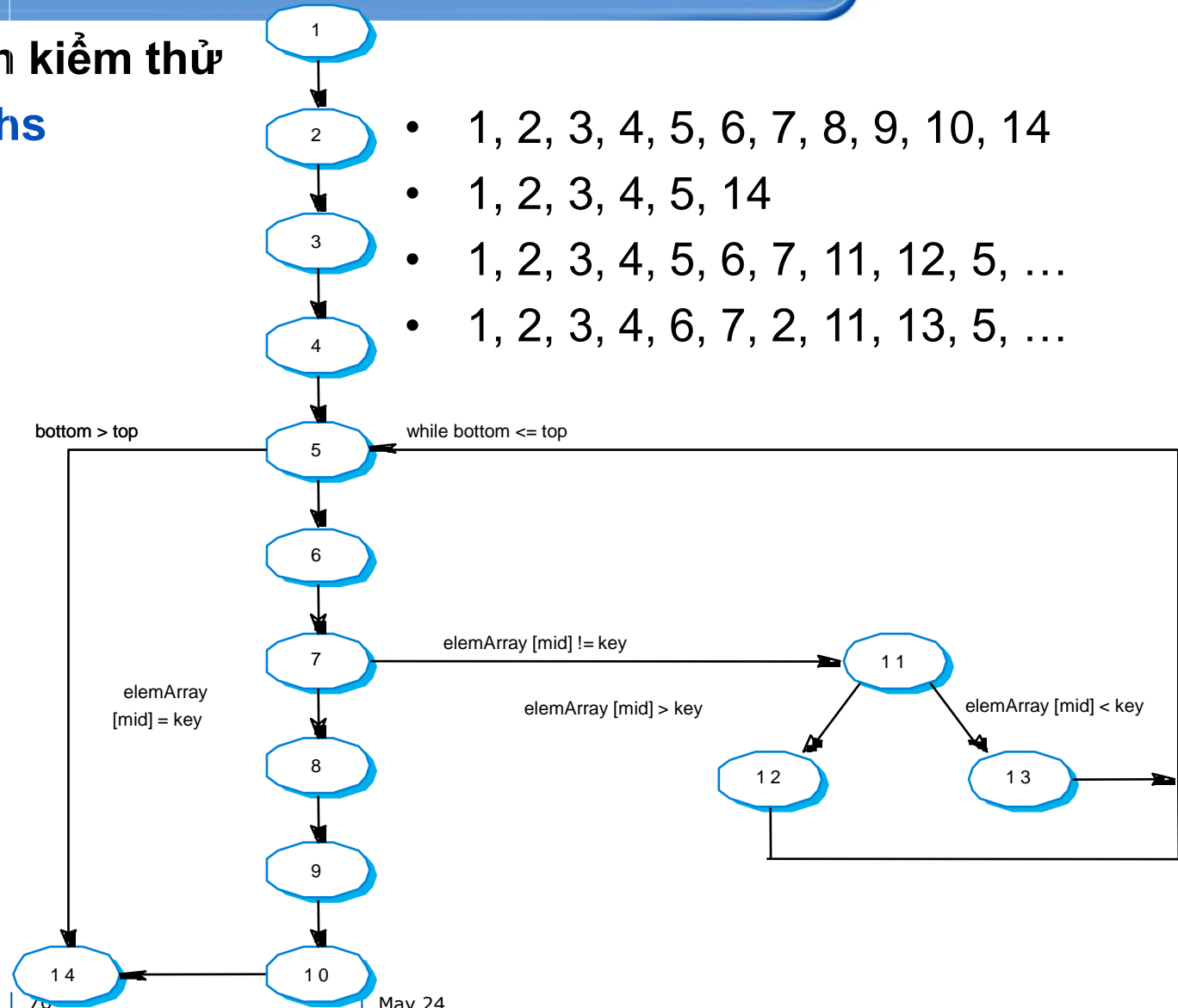
2.3. Các hướng dẫn kiểm thử Binary search flow graph



2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Independent paths



2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Independent paths

- ◆ Cần tạo các test case để chạy tất cả các independent path
- ◆ Có thể dùng phần mềm phân tích động chương trình để kiểm tra xem các path đó đã được chạy qua chưa

2. KIỂM THỬ PHẦN MỀM

2.3. Các hướng dẫn kiểm thử

■ Tự động hóa test

- ◆ Kiểm thử là một bước quy trình tốn kém.
- ◆ Các Testing workbench cung cấp nhiều công cụ để giảm thời gian và chi phí kiểm thử.
- ◆ Các hệ thống như Junit hỗ trợ tự động hóa việc chạy test.
- ◆ Nhiều testing workbench là các hệ thống mở vì các nhu cầu kiểm thử có tính đặc thù đối với từng tổ chức.

3. BẢO TRÌ PHẦN MỀM

Bảo trì phần mềm được chia thành 4 loại:

- **Sửa lại cho đúng (corrective):** là việc sửa các lỗi phát sinh trong quá trình sử dụng.
- **Thích ứng (adaptative):** là việc chỉnh sửa hệ thống cho phù hợp với môi trường đã thay đổi.
- **Hoàn thiện (perfective):** là việc chỉnh sửa để đáp ứng các yêu cầu mới **hoặc các yêu cầu đã thay đổi** của người sử dụng.
- **Bảo vệ (preventive):** làm cho hệ thống dễ dàng bảo trì hơn trong những lần tiếp theo.

Q & A

Ôn tập

- Việc kiểm thử (testing) có thể cho thấy lỗi trong hệ thống; nó không thể chứng minh hệ thống không còn lỗi nào.
- Những người phát triển component có trách nhiệm thực hiện component testing; system testing là trách nhiệm của một đội khác.
- Integration testing là kiểm thử các bản tăng dần (increment) của hệ thống; release testing kiểm thử hệ thống trước khi trao cho khách hàng.
- Dùng kinh nghiệm và hướng dẫn để thiết kế các test case cho defect testing.

- Interface testing được thiết kế để phát hiện lỗi trong các interface của các composite component.
- Phân hoạch tương đương (equivalence partitioning) là cách tìm ra các test case – tất cả các case trong một phân hoạch cần vận hành như nhau.
- Structural testing dựa vào việc phân tích chương trình và rút ra các test từ cấu trúc chương trình.
- Tự động hóa test giúp làm giảm chi phí.

Ôn tập

1. Chọn 1 yêu cầu trong dự án bài tập lớn, thiết kế các test case đủ để kiểm tra xem một phần mềm có thỏa mãn yêu cầu đó không.
2. Sử dụng cách tiếp cận object testing, thiết kế các test case để kiểm thử các trạng thái của lò vi sóng theo như state diagram trong slide bài giảng tuần 6-7 (Modelling)