# Kiểm thử phần mềm

Nguyễn Thị Minh Tuyền

Nội dung của slide này dựa vào các slide của Ian Sommerville

# Nội dung

1. Kiểm thử trong khi xây dựng
2. Phát triển theo hướng kiểm thử
3. Kiểm thử bản release
4. Kiểm thử người dùng

# Kiểm thử chương trình

- Mục tiêu của kiểm thử là để chỉ ra rằng một chương trình thực hiện đúng như mong đợi và tìm ra được lỗi của chương trình trước khi đưa vào sử dụng.
- Khi kiểm thử phần mềm, ta chạy phần mềm đó với dữ liệu nhân tạo.
- Kiểm tra kết quả của việc kiểm thử để tìm ra lỗi, những bất thường hoặc thông tin về các thuộc tính phi chức năng của chương trình.
- Có thể chỉ ra sự có mặt của lỗi, không chỉ ra được chương trình không có lỗi.
- Kiểm thử là một phần của quy trình thẩm định và kiểm định phần mềm (verification and validation – V&V), gồm các kỹ thuật thẩm định tĩnh.

# Mục tiêu của kiểm thử chương trình

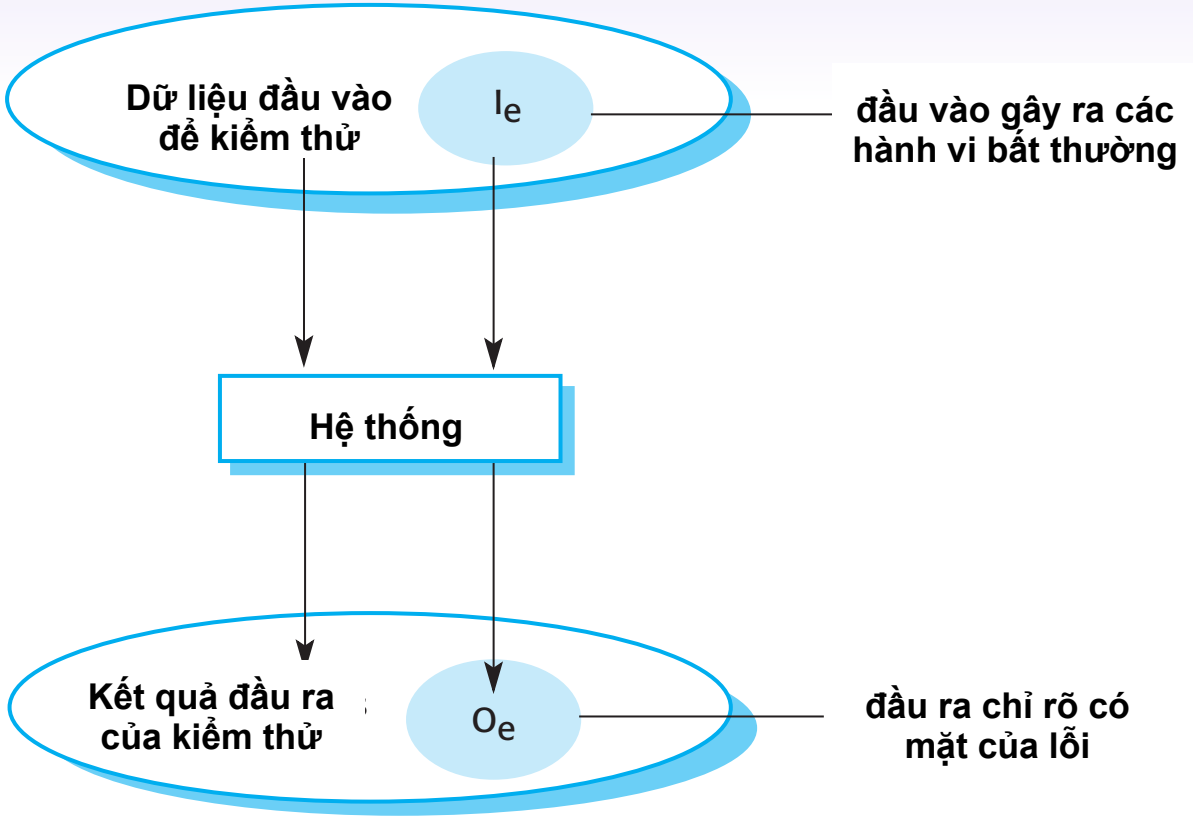
Validation testing

**Để chỉ ra cho người phát triển và khách hàng rằng phần mềm thỏa mãn các yêu cầu đưa ra.**

Defect testing

**Để chỉ ra các tình huống trong đó các hành vi của phần mềm không đúng, không như mong đợi hoặc không tương thích với đặc tả.**

# Mô hình input-output của kiểm thử chương trình



# Kiểm định và thẩm định

- **Kiểm định (verification):**
  **"Are we building the product right".**
  - Phần mềm phải tương thích với đặc tả.
- **Thẩm định(validation):**
  **"Are we building the right product".**
  - Phần mềm phải thỏa mãn được những gì người dùng thật sự yêu cầu.

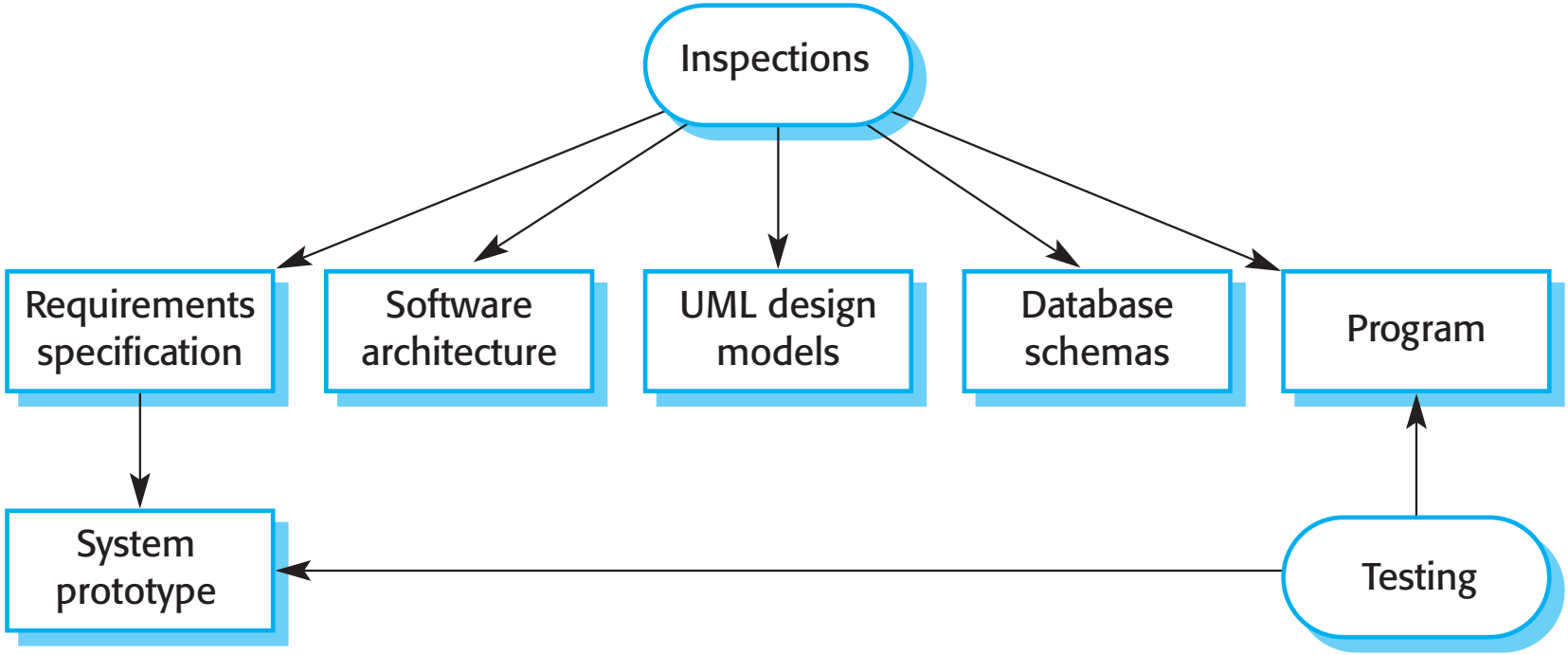
# Mục tiêu của V & V

- **Mục tiêu của V & V là để thiết lập độ tin cậy rằng hệ thống thỏa mãn mục tiêu đặt ra.**
- **Phụ thuộc vào:**
  - Mục đích phần mềm
    - Độ tin cậy  của phần mềm phụ thuộc vào tầm quan trọng của phần mềm đối với một tổ chức.
  - Mong đợi của người dùng
    - Người dùng có thể có mong đợi thấp về một số loại sản phẩm nào đó.
  - Môi trường thương mại
    - Việc thương mại hóa một sản phẩm sớm có thể quan trọng hơn việc tìm lỗi trong chương trình.

# Thanh tra và kiểm thử

- **Thanh tra phần mềm (Software inspection)**
  - Liên quan đến việc phân tích các biểu diễn tĩnh của hệ thống để tìm ra lỗi *(*static verification).
- **Kiểm thử phần mềm (Software testing)**
  - Liên quan đến việc thực hiện và quan sát hành vi của sản phẩm (dynamic verification).
  - Hệ thống được thực thi với dữ liệu kiểm thử và quan sát hành vi hoạt động của hệ thống.

# Thanh tra và kiểm thử



# Thanh tra phần mềm

- Có sự tham gia của con người, kiểm tra biểu diễn nguồn với mục đích tìm ra những bất thường và lỗi.
- Không yêu cầu chạy chương trình, có thể được áp dụng cho các hoạt động trước khi cài đặt.
- Có thể áp dụng cho bất cứ biểu diễn nào của hệ thống (yêu cầu, thiết kế, cấu hình dữ liệu, dữ liệu kiểm thử,... ).
- Đã được chứng minh là một kỹ thuật hiệu quả trong việc tìm ra lỗi chương trình.

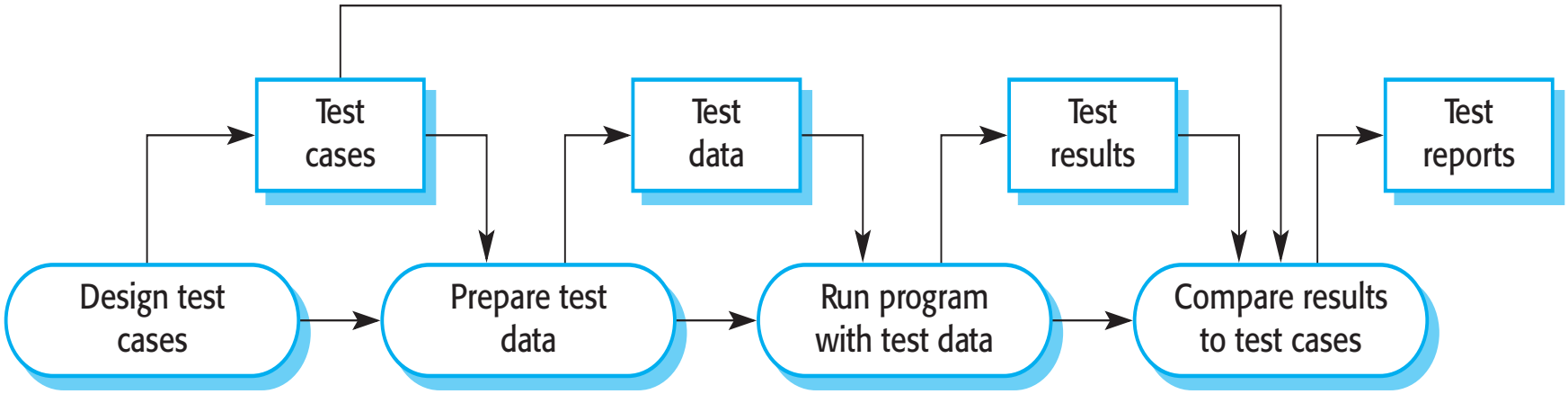
# Ưu điểm của thanh tra phần mềm

- **Trong suốt quá trình kiểm thử, lỗi có thể bị che giấu bởi các lỗi khác. Vì thanh tra là một quy trình tĩnh, ta không cần quan tâm đến tương tác giữa các lỗi.**
- **Có thể sử dụng phương pháp này với các phiên bản chưa hoàn thành mà không thêm chi phí.**
- **Cũng như tìm kiếm lỗi chương trình, thanh tra cũng có thể xem xét các thuộc tính về chất lượng của một chương trình.**
  - Ta có thể tìm ra những điểm không hiệu quả, không hợp lý trong thuật toán, ...

# Thanh tra và kiểm thử

- Cả hai kỹ thuật hỗ trợ cho nhau và không trái ngược nhau.
- Nên sử dụng cả hai trong quy trình V & V.
- Thanh tra có thể kiểm tra một cài đặt thỏa mãn đặc tả nhưng không thỏa mãn yêu cầu thực của khách hàng.
- Thanh tra không thể kiểm tra các đặc điểm phi chức năng như hiệu năng, tính sử dụng,...

# Một mô hình của quy trình kiểm thử phần mềm



# Các giai đoạn của kiểm thử

- **Kiểm thử trong khi xây dựng (Development testing)**
  - Hệ thống được kiểm thử trong quá trình phát triển để tìm ra lỗi.
- **Kiểm thử bản release (Release testing)**
  - Một đội ngũ kiểm thử động lập sẽ kiểm thử một phiên bản hoàn chỉnh của hệ thống trước khi nó được bàn giao cho người dùng.
- **Kiểm thử người dùng(User testing)**
  - Người dùng hoặc người dùng tiềm năng của hệ thống kiểm thử hệ thống trên môi trường của họ.

# Nội dung

1. **Kiểm thử trong khi xây dựng**
2. Phát triển theo hướng kiểm thử
3. Kiểm thử bản release
4. Kiểm thử người dùng

# Kiểm thử trong khi xây dựng

- **Gồm tất cả các hoạt động kiểm thử được tiến hành bởi nhóm phát triển hệ thống.**
  - Kiểm thử đơn vị (unit testing)
    - Kiểm thử các đơn vị chương trình riêng lẻ hay các lớp đối tượng.
    - Nên tập trung vào việc kiểm thử chức năng của các đối tượng hay các phương thức.
  - Kiểm thử component (component testing)
    - Vài đơn vị riêng lẻ được tích hợp thành các component.
    - Nên tập trung vào kiểm thử giao diện component.
  - Kiểm thử hệ thống (system testing)
    - Một số hoặc tất cả các component trong hệ thống được tích hợp và hệ thống được kiểm thử toàn bộ.
    - Nên tập trung vào kiểm thử tương tác giữa các component.

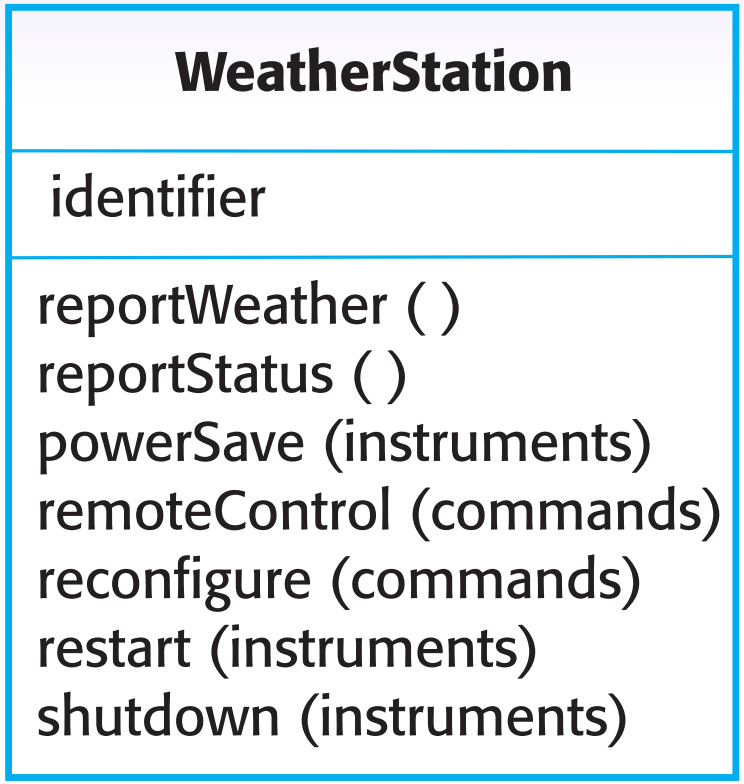
# Kiểm thử đơn vị

- **Là quy trình kiểm thử từng component riêng lẻ.**
- **Đây là quy trình kiểm thử tìm lỗi.**
- **Các đơn vị có thể là:**
  - Các hàm hay phương thức đơn lẻ trong một đối tượng.
  - Các lớp đối tượng chứa vài thuộc tính và phương thức.
  - Các component với các giao diện được định nghĩa sẵn để truy cập vào các tính năng của chúng.

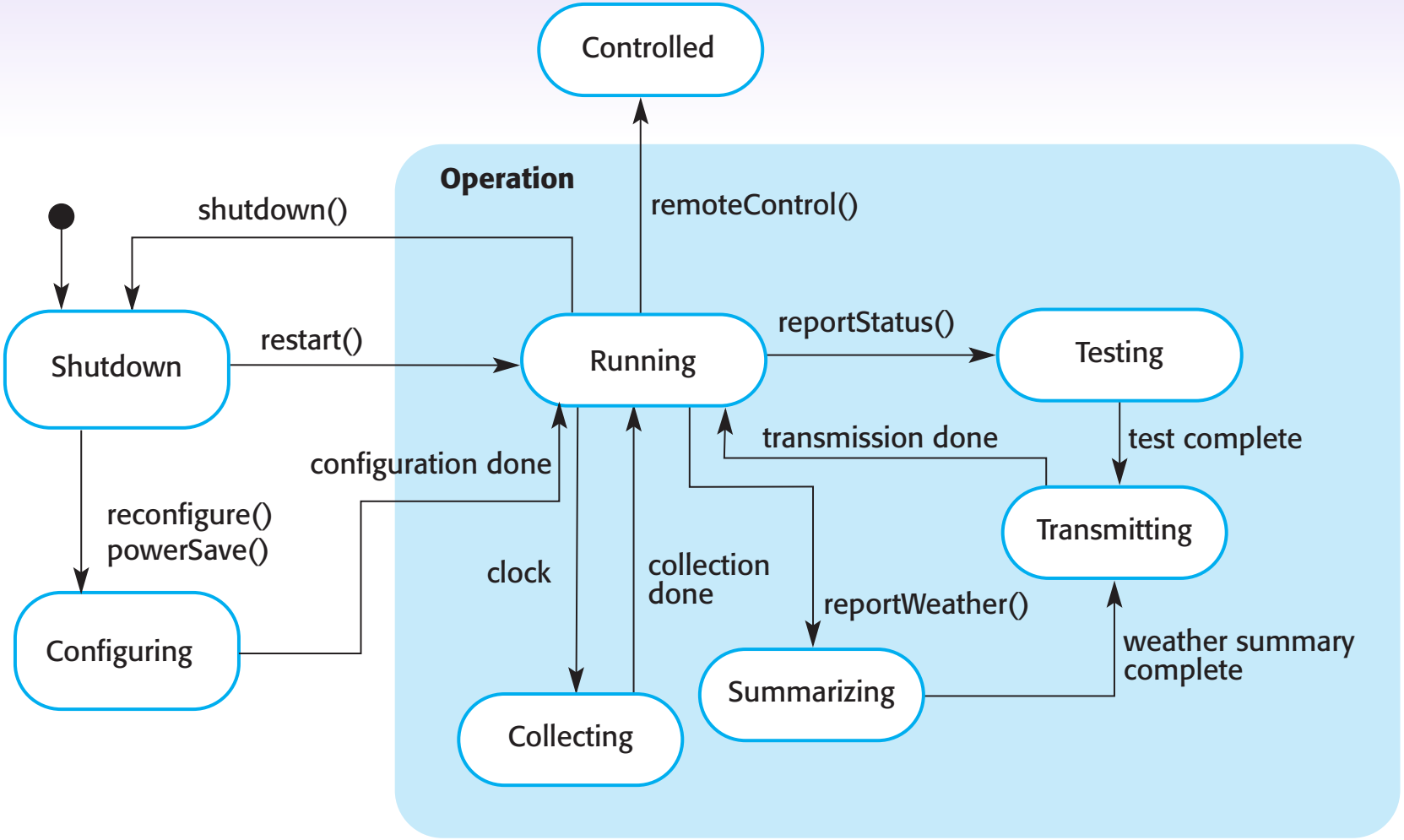
# Kiểm thử lớp đối tượng

- **Việc kiểm thử bao phủ một lớp đối tượng gồm**
  - Kiểm thử tất cả các thuộc tính liên quan đến một đối tượng.
  - Thiết lập và kiểm thử giá trị của tất cả các thuộc tính.
  - Thực thi đối tượng với tất cả các trạng thái có thể.
- **Tính kế thừa làm cho việc thiết kế các kiểm thử lớp đối tượng trở nên khó khăn vì thông tin cần kiểm thử không được định vị.**

# Giao diện đối tượng weather station



# Nhắc lại: biểu đồ trạng thái của Weather station



# Kiểm thử Weather station

- **Cần định nghĩa các test case cho reportWeather, calibrate, test, startup và shutdown.**
- **Sử dụng mô hình trạng thái, nhận diện chuỗi các chuyển dịch trạng thái để kiểm thử và chuỗi các tác động gây nên các chuyển dịch trạng thái đó.**
- **Ví dụ:**
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

# Kiểm thử tự động

- Nếu có thể, nên tự động hóa việc kiểm thử đơn vị để các test được chạy và kiểm tra mà không cần sự can thiệp của con người.
- Trong kiểm thử đơn vị tự động, sử dụng framework hỗ trợ kiểm thử tự động (JUnit chẳng hạn) để viết và chạy chương trình test.
- Các framework hỗ trợ kiểm thử đơn vị cung cấp các lớp kiểm thử tổng quát cho phép ta tạo ra các test case cụ thể. Các framework này có thể chạy tất cả các test mà ta đã cài đặt, thường là thông qua một số GUI, để xem các test thành công hay thất bại.

# Các thành phần của kiểm thử tự động

- **Phần thiết lập**
  - Khởi tạo hệ thống với test case, gọi là đầu vào và đầu ra mong muốn.
- **Phần gọi**
  - Gọi đối tượng hoặc phương thức được kiểm tra.
- **Phần assertion**
  - So sánh kết quả của việc gọi với kết quả mong đợi. Nếu assertion được chứng minh là đúng, thì test thành công, nếu sai thì thì test thất bại.

# Tính hiệu quả của kiểm thử đơn vị

- **Các test case nên chỉ ra rằng component mà ta đang kiểm thử phải thực hiện được những gì nó được giả định sẽ thực hiện.**
- **Nếu có lỗi trong component đang kiểm thử, thì những lỗi này nên được tìm ra bởi test case.**
- **Điều này dẫn đến hai loại test case đơn vị:**
  - Loại thứ nhất: phản ánh hoạt động bình thường của chương trình và chỉ ra rằng component thực hiện các thao tác như mong đợi.
  - Loại thứ hai: dựa vào kinh nghiệm kiểm thử để chỉ ra những lỗi thông thường. Sử dụng các đầu vào bất thường để kiểm tra rằng những đầu vào này được xử lý và không bị lỗi chương trình.

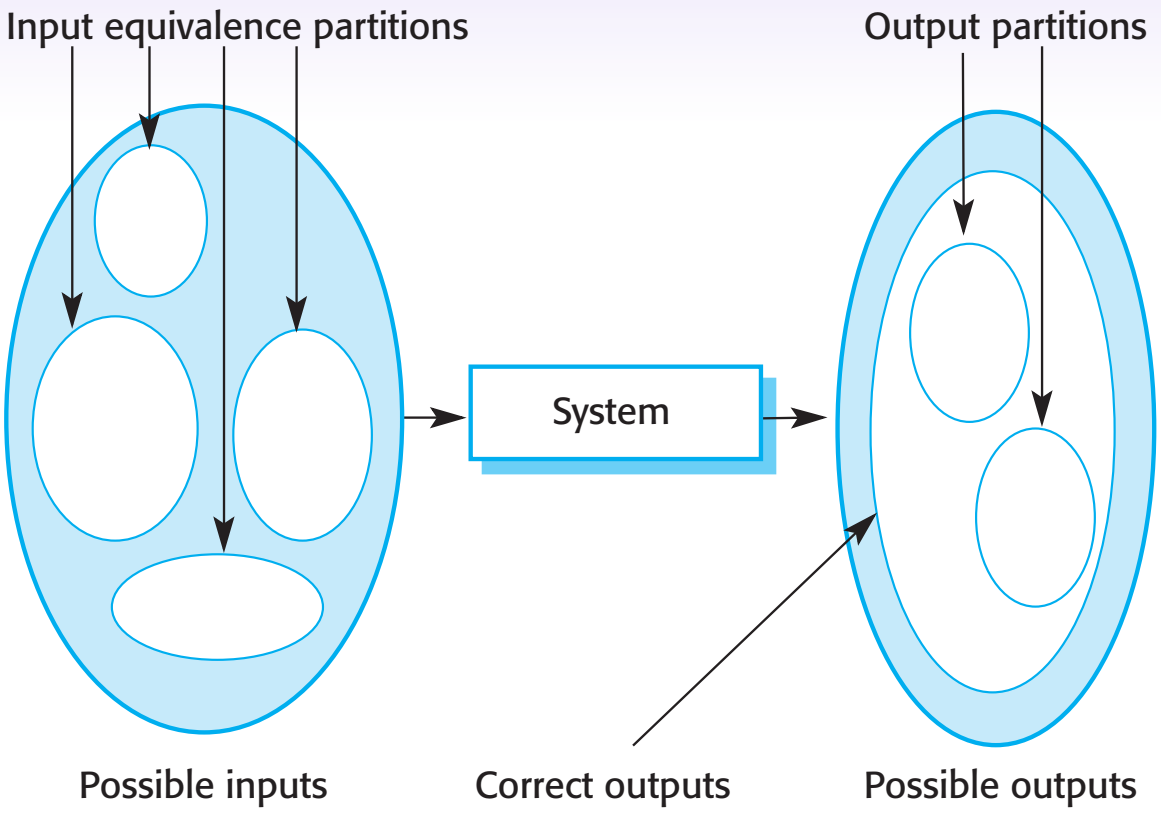
# Chiến thuật kiểm thử

- **Partition testing**
  - Nhận diện các nhóm đầu vào có cùng đặc điểm và được xử lý cùng cách.
  - Nên chọn các test từ mỗi nhóm này.
- **Guideline-based testing**
  - Sử dụng các chỉ dẫn về kiểm thử để chọn các test case.
  - Các chỉ dẫn này phản ánh kinh nghiệm trước đó về một số loại lỗi mà người lập trình thường mắc phải khi phát triển các component.

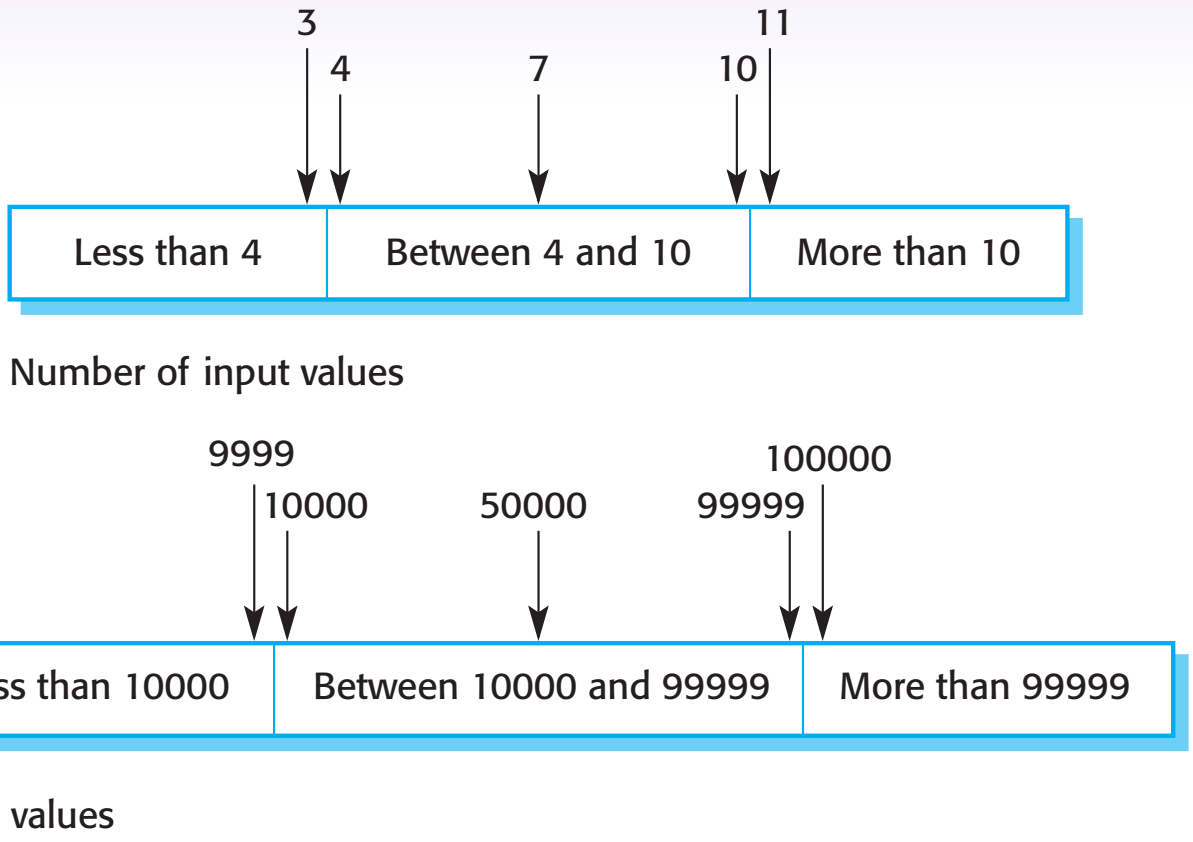
# Partition testing

- Dữ liệu đầu vào và kết quả đầu ra thường rơi vào các lớp khác nhau mà trong đó các phần tử của một lớp có đặc điểm chung.
- Mỗi lớp này là một equivalence partition trong đó chương trình xử lý theo cùng một cách cho mỗi phần tử của lớp.
- Các test case nên được chọn từ mỗi partition.

# Equivalence partitioning



# Equivalence partitions



# Testing guidelines (đối với xử lý chuỗi)

- Sử dụng một giá trị để test các chuỗi.
- Sử dụng chuỗi có kích thước khác nhau trong các test khác nhau.
- Chọn các test sao cho những phần tử đầu tiên, ở chính giữa và cuối cùng của chuỗi được truy cập.
- Kiểm thử với chuỗi có kích thước bằng 0.

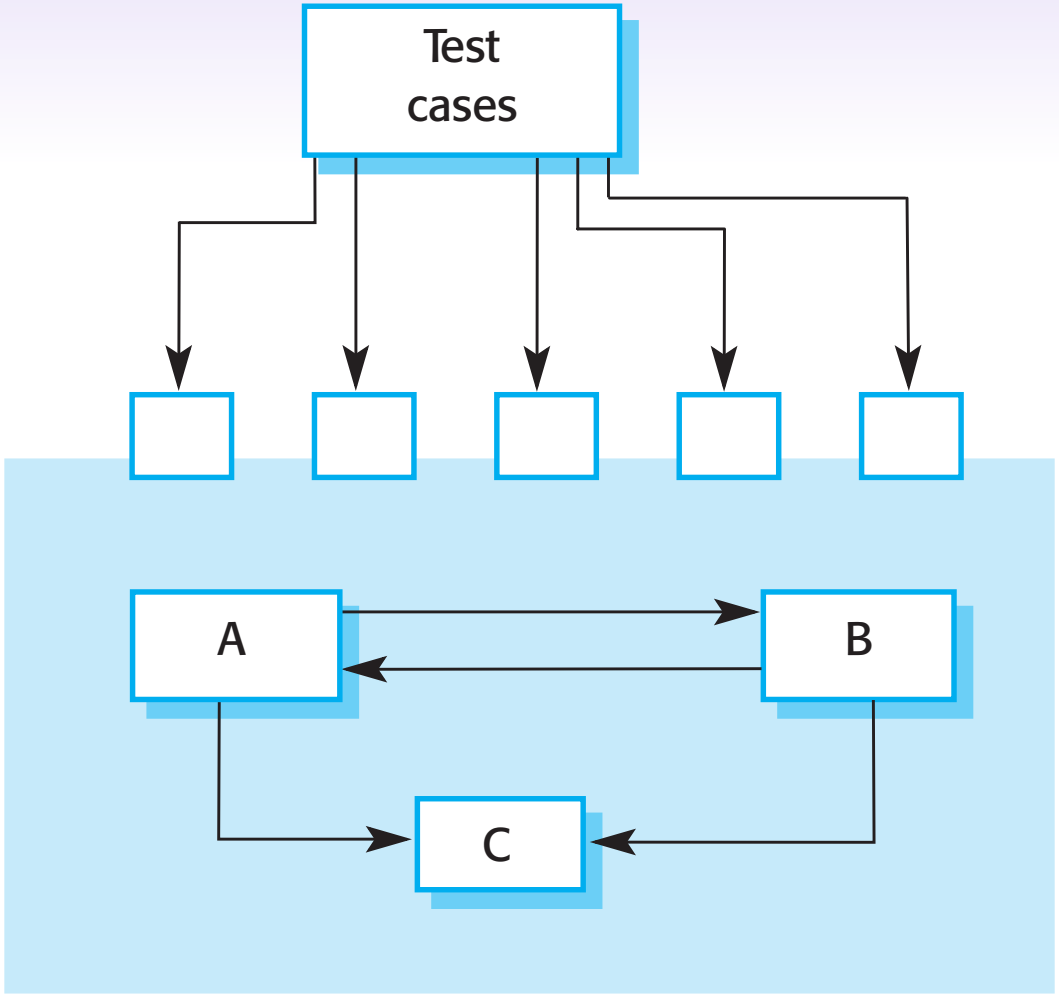
# Chỉ dẫn tổng quát về kiểm thử

- Chọn các đầu vào để bắt buộc chương trình phát sinh ra tất cả các thông báo lỗi.
- Thiết kế các đầu vào mà nó gây nên lỗi tràn buffer.
- Lặp lại cùng đầu vào hoặc một chuỗi các đầu vào nhiều lần.
- Buộc chương trình phải phát sinh ra các đầu ra không hợp lệ.
- Buộc sinh ra các kết quả tính toán quá lớn hoặc quá nhỏ.

# Kiểm thử component

- **Các component thường được tạo ra bởi vài đối tượng tương tác với nhau.**
- **Truy cập vào những đối tượng này thông qua giao diện component được định nghĩa sẵn.**
- **Nên tập trung vào việc chỉ ra rằng giao diện component thỏa mãn đặc tả của nó.**
  - Giả sử rằng kiểm thử đơn vị trên các đối tượng đơn lẻ đã hoàn thành.

# Kiểm thử giao diện



# Kiểm thử giao diện

- **Mục tiêu là tìm ra lỗi gây ra bởi các lỗi giao diện hoặc giả định sai về các giao diện.**
- **Các loại giao diện**
  - **Giao diện có tham số** Dữ liệu được chuyển từ phương thức hay thủ tục này sang phương thức hay thủ tục khác.
  - **Giao diện sử dụng bộ nhớ chia sẻ** Các vùng nhớ được chia sẻ giữa các thủ tục hay các hàm.
  - **Giao diện chứa các thủ tục** Hệ thống con chứa một tập các thủ tục được gọi bởi các hệ thống con khác.
  - **Giao diện truyền thông điệp** Các hệ thống con yêu cầu các dịch vụ từ các hệ thống con khác.

# Lỗi giao diện

- **Sử dụng sai**
  - Một component gọi một component khác và gây ra lỗi trong việc sử dụng giao diện. Ví dụ như thứ tự các tham số bị sai.
- **Hiểu sai về giao diện**
  - Một component gọi đưa ra giả định sai về hành vi của component được gọi.
- **Lỗi định thời gian**
  - Component gọi và được gọi thực hiện ở tốc độ khác nhau do đó thông tin cũ được truy cập.

# Các chỉ dẫn về kiểm thử giao diện

- Thiết kế các test sao cho các tham số truyền đến thủ tục được gọi ở điểm cận của khoảng giá trị.
- Luôn luôn kiểm thử các tham số con trỏ với giá trị null.
- Thiết kế test sao cho nó làm cho component sinh lỗi.
- Sử dụng stress testing trong hệ thống truyền thông điệp.
- Trong hệ thống chia sẻ bộ nhớ, thay đổi thứ tự các component được kích hoạt.

# Kiểm thử hệ thống

- Gồm việc tích hợp các component để tạo ra một phiên bản của hệ thống và sau đó kiểm thử hệ thống được tích hợp.
- Tập trung vào việc kiểm thử tương tác giữa các component.
- Kiểm tra rằng các component tương thích với nhau, tương tác đúng và chuyển đúng dữ liệu, đúng thời điểm thông qua giao diện của chúng.

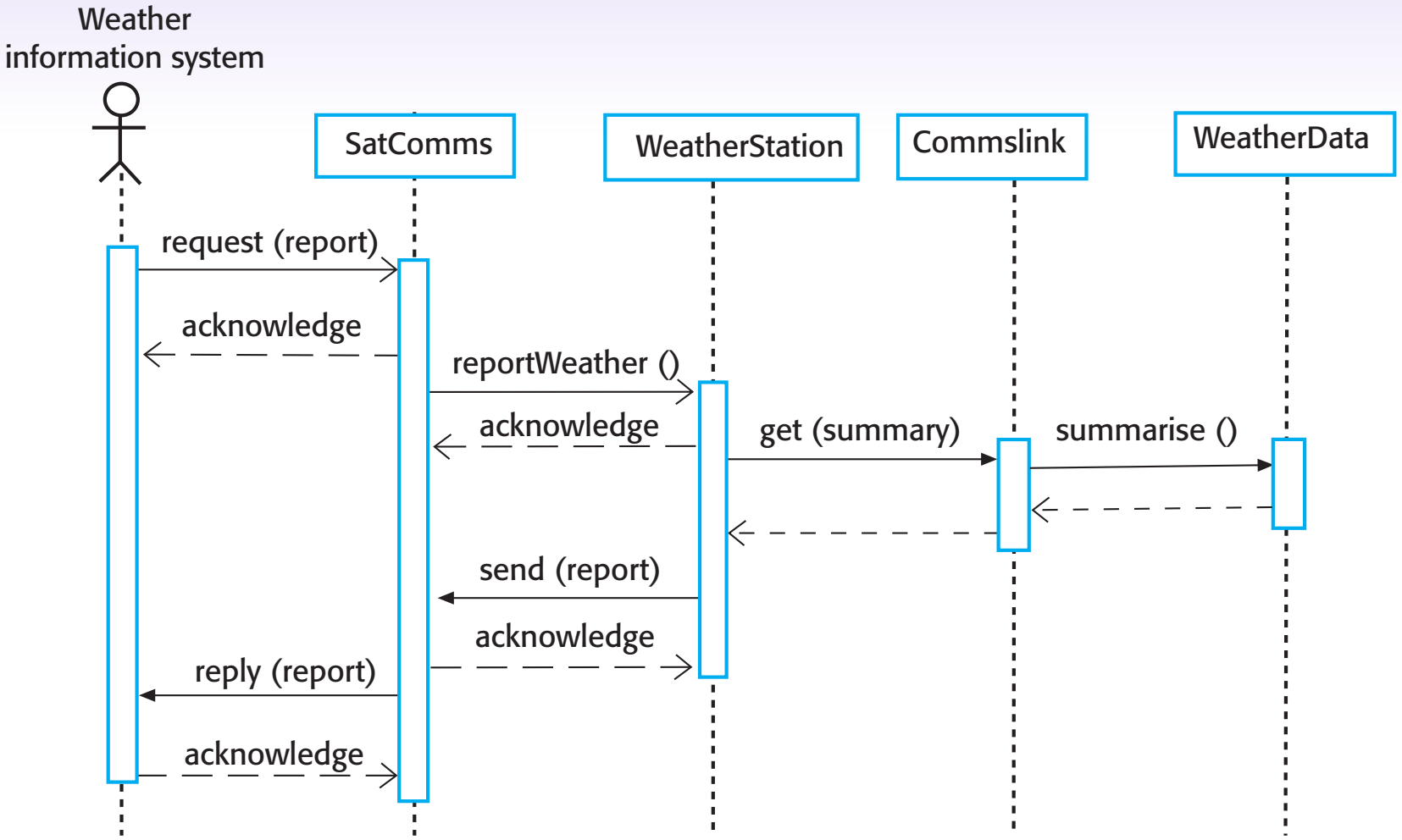
# Kiểm thử hệ thống và kiểm thử component

- **Trong suốt quá trình kiểm thử hệ thống, các component sử dụng lại đã được phát triển độc lập và các hệ thống có sẵn được tích hợp với các component đang phát triển. Hệ thống hoàn chỉnh được kiểm thử.**
- **Các component được phát triển bởi các thành viên khác nhau của nhóm phát triển được tích hợp lại trong giai đoạn này.**
  - Trong một số công ty, kiểm thử hệ thống có thể được thực hiện bởi một nhóm độc lập không tham gia vào việc thiết kế và cài đặt.

# Kiểm thử dựa vào use-case

- Các use-case được phát triển để nhận diện các tương tác hệ thống có thể được sử dụng làm cơ sở để kiểm thử hệ thống.
- Mỗi use case thường gồm một số component hệ thống vì vậy việc kiểm thử dựa vào use-case buộc các tương tác này phải  xảy ra.
- Biểu đồ tuần tự cũng có thể được sử dụng để kiểm thử.

# Biểu đồ tuần tự về thu thập dữ liệu thời tiết



# Các chính sách kiểm thử

- **Việc kiểm thử đầy đủ cả hệ thống là điều không thể**
  - Vì vậy cần phát triển các chính sách kiểm thử để định nghĩa độ bao phủ khi kiểm thử hệ thống.
- **Ví dụ về các chính sách kiểm thử:**
  - Tất cả các hàm của hệ thống được truy cập thông qua menu nên được kiểm thử.
  - Việc kết hợp các hàm được truy cập qua cùng menu phải được kiểm thử.
  - Khi đầu vào được cung cấp, tất cả các hàm phải được kiểm tra với cả hai trường hợp giá trị đầu vào đúng và sai.

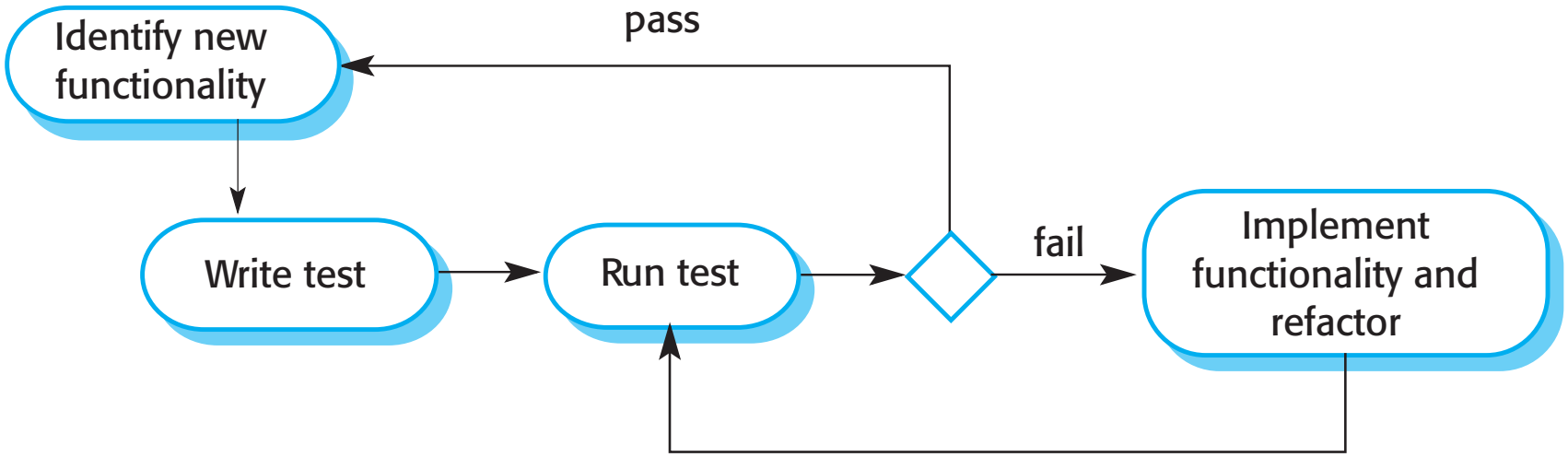
# Nội dung

1. Kiểm thử trong khi xây dựng
2. **Phát triển theo hướng kiểm thử**
3. Kiểm thử bản release
4. Kiểm thử người dùng

# Phát triển theo hướng kiểm thử (Test-driven development – TDD)

- Là một phương pháp phát triển chương trình trong đó việc phát triển mã nguồn và kiểm thử đan xen nhau.
- Các test được viết trước khi lập trình và phải "pass" các test là yếu tố quan trọng của việc phát triển.
- Phát triển mã nguồn theo kiểu tăng dần, song song với việc kiểm thử cho từng phần đó. Ta không thể chuyển sang cài đặt phần tiếp theo cho đến khi mã nguồn đang phát triển "pass" tất cả các test của nó.
- TDD được xem là một phần của các phương pháp linh hoạt như Extreme Programming. Tuy nhiên, nó cũng có thể được dùng trong các quy trình phát triển hoạch định sẵn.

# Phát triển theo hướng kiểm thử



# Lợi ích của TDD

- **Bao phủ mã nguồn**
  - Mỗi phần mã nguồn ta viết ra đều ít nhất liên quan đến một test case. Vì vậy tất cả các mã nguồn được viết ra có ít nhất một test case.
- **Kiểm thử hồi quy (Regression testing)**
  - Một bộ kiểm hồi quy được phát triển dần dần như một chương trình được phát triển.
- **Đơn giản hóa việc sửa lỗi**
  - Khi một test thất bại, ta thấy được rõ ràng vấn đề nằm ở đâu. Mã nguồn mới được viết ra cần được kiểm tra và bổ sung.
- **Tài liệu hệ thống**
  - Bản thân các test là một dạng tài liệu mà nó mô tả mã nguồn làm gì.

# Kiểm thử hồi quy

- Là việc kiểm thử hệ thống để kiểm tra rằng sự thay đổi không phá vỡ việc cài đặt mã nguồn trước đó.
- Trong quy trình kiểm thử bằng tay, kiểm thử hồi quy rất tốn kém. Tuy nhiên, với kiểm thử tự động, kiểm thử hồi quy lại đơn giản và trực tiếp. Tất cả các test đều được thực thi lại mỗi khi có sự thay đổi trong chương trình.
- Các test phải được thực thi thành công trước khi chấp nhận một thay đổi.

# Nội dung

1. Kiểm thử trong khi xây dựng
2. Phát triển theo hướng kiểm thử
3. **Kiểm thử bản release**
4. Kiểm thử người dùng

# Kiểm thử bản release

- **Là quy trình kiểm thử một bản release của hệ thống, bản này sẽ sử dụng bên ngoài đội ngũ phát triển hệ thống.**
- **Mục tiêu chính là để thuyết phục khách hàng rằng hệ thống đủ tốt để đưa vào sử dụng.**
  - Phải chỉ ra được rằng hệ thống hỗ trợ các tính năng đã đặc tả, đảm bảo hiệu năng và độ tin cậy, và không có lỗi khi sử dụng.
- **Là quy trình kiểm thử hộp đen trong đó các test chỉ bắt nguồn từ đặc tả hệ thống.**

# Kiểm thử bản release và kiểm thử hệ thống

- **Kiểm thử bản release là một hình thức của kiểm thử hệ thống.**
- **Điểm khác nhau quan trọng:**
  - Một nhóm tách biệt không tham gia vào việc phát triển sẽ chịu trách nhiệm  về kiểm thử bản release.
  - Kiểm thử hệ thống bởi nhóm phát triển nên tập trung vào việc tìm lỗi trong hệ thống (defect testing).
  - Mục tiêu của kiểm thử bản release là để chứng tỏ rằng hệ thống đáp ứng yêu cầu và đủ tốt để đưa ra sử dụng bên ngoài (validation testing).

# Kiểm thử dựa vào yêu cầu

- **Gồm việc kiểm tra mỗi yêu cầu và phát triển test cho yêu cầu đó.**
- **Ví dụ: các yêu cầu của hệ thống MHC-PMS:**
  - Nếu một bệnh nhân được biết là dị ứng với một loại thuốc nào đó, khi kê đơn loại thuốc đó hệ thống sẽ phải đưa ra cảnh báo đến người dùng hệ thống.
  - Nếu người kê đơn chọn thuốc mà bỏ qua cảnh báo về dị ứng, họ sẽ phải đưa ra lý do tại sao lại bỏ qua cảnh báo.

# Các test dựa vào yêu cầu

- Thiết lập một hồ sơ bệnh nhân với thông tin không bị dự ứng loại thuốc nào. Kê đơn thuốc liên quan đến các dị ứng. Kiểm tra rằng thông điệp cảnh báo không xuất hiện.
- Thiết lập một hồ sơ bệnh nhân với thông tin bị dị ứng với một loại thuốc. Kê đơn thuốc có loại thuốc mà bệnh nhân bị dị ứng, và kiểm tra rằng cảnh báo được đưa ra bởi hệ thống.
- Thiết lập một hồ sơ bệnh nhân trong đó có thông tin dị ứng với hai hoặc nhiều hơn hai loại thuốc. Kê đơn cả hai loại này tách biệt nhau và kiểm tra rằng cảnh báo đúng cho từng loại thuốc được đưa ra.
- Kê đơn hai loại thuốc mà bệnh nhân bị dị ứng. Kiểm ra rằng hai cảnh báo đúng được đưa ra.
- Kê đơn một loại thuốc mà cảnh báo xuất hiện và bỏ qua cảnh báo đó. Kiểm tra rằng hệ thống yêu cầu người dùng cung cấp lý do tại sao bỏ qua cảnh báo.

# Một kịch bản cho hệ thống MHC-PMS

Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side -effects.

On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side -effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

# Chức năng được kiểm định dựa vào kịch bản

- Phân quyền bằng cách đăng nhập vào hệ thống.
- Tải và upload hồ sơ bệnh nhân từ máy tính.
- Lập lịch thăm bệnh nhân tại nhà.
- Mã hóa và giải mã hồ sơ bệnh nhân trên thiết bị di động.
- Tìm kiếm và bổ sung hồ sơ.
- Liên kết tới CSDL thuốc có chứa thông tin về hiệu ứng phụ.
- Hệ thống hỗ trợ việc nhắc nhở lịch hẹn.

# Performance testing

- Là một phần của kiểm thử bản release, có thể bao gồm việc kiểm thử các thuộc tính của hệ thống chẳng hạn như hiệu năng hay độ tin cậy.
- Các test nên phản ánh tính sử dụng của hệ thống.
- Gồm việc lên kế hoạch cho một chuỗi các test mà tại đó tải tăng ổn định cho đến khi hiệu năng của hệ thống trở nên không chấp nhận được.
- Stress testing là một hình thức của performance testing ở đó hệ thống cố tình bị quá tải để kiểm tra hành vi lỗi của nó.

# Nội dung

1. Kiểm thử trong khi xây dựng
2. Phát triển theo hướng kiểm thử
3. Kiểm thử bản release
4. **Kiểm thử người dùng**

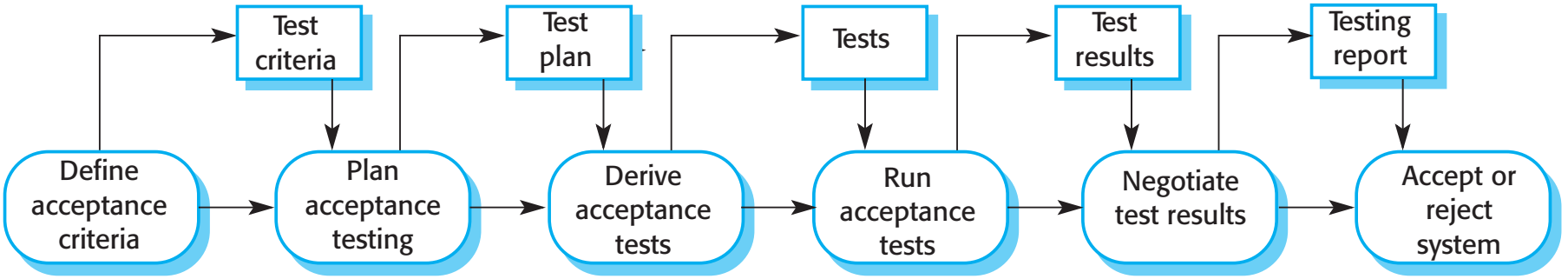
# Kiểm thử người dùng(user testing)

- **Là một giai đoạn trong quy trình kiểm thử trong đó người dùng cung cấp đầu vào và đưa ra lời khuyên cho việc kiểm thử hệ thống.**
- **Kiểm thử người dùng là cần thiết, thậm chí khi một hệ thống đã rõ ràng và kiểm thử bản release đã được tiến hành.**
  - Lý do cho điều này là ảnh hưởng từ môi trường làm việc của người sử dụng có một ảnh hưởng quan trọng lên độ tin cậy, hiệu năng, tính sử dụng và khả năng chịu lỗi của một hệ thống. Những điều này không thể mô phỏng trong môi trường kiểm thử.

# Các loại kiểm thử người dùng

- **Alpha testing**
  - Người dùng phần mềm làm việc với nhóm phát triển để kiểm thử phần mềm tại nơi phát triển phần mềm.
- **Beta testing**
  - Một bản release có sẵn cho phép người dùng sử dụng chúng lấy kinh nghiệm và tìm ra lỗi với người phát triển hệ thống.
- **Acceptance testing**
  - Khách hàng kiểm thử hệ thống để quyết định xem hệ thống này có được chấp nhận để triển khai đến môi trường làm việc của khách hàng hay không.

# Quy trình acceptance testing



# Phương pháp linh hoạt và acceptance testing

- Trong phương pháp linh hoạt, người dùng/khách hàng là một phần của nhóm phát triển và chịu trách nhiệm đưa ra các quyết định về việc chấp nhận hệ thống.
- Các test được định nghĩa bởi người dùng/khách hàng và được tích hợp vào các test khác trong đó chúng được kiểm tra tự động khi có thay đổi xảy ra.
- Không có một quy trình acceptance testing tách biệt.
- Vấn đề chính ở đây là liệu người dùng tham gia trực tiếp là người đại diện cho tất cả các mối quan tâm của toàn bộ stakeholder hệ thống hay không.


# Thank You !