Kỹ thuật phần mềm

Testing

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

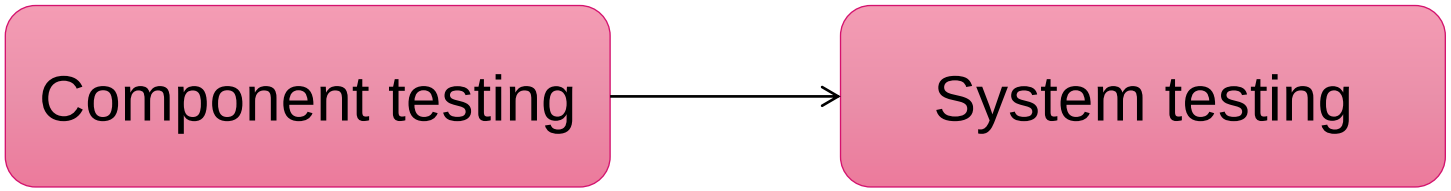
Các chủ đề

- System testing – kiểm thử hệ thống
- Component testing – kiểm thử thành phần
- Test case design – thiết kế test case
- Test automation – tự động hóa kiểm thử

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

Các pha kiểm thử



Nhóm phát triển phần mềm

Đội kiểm thử độc lập

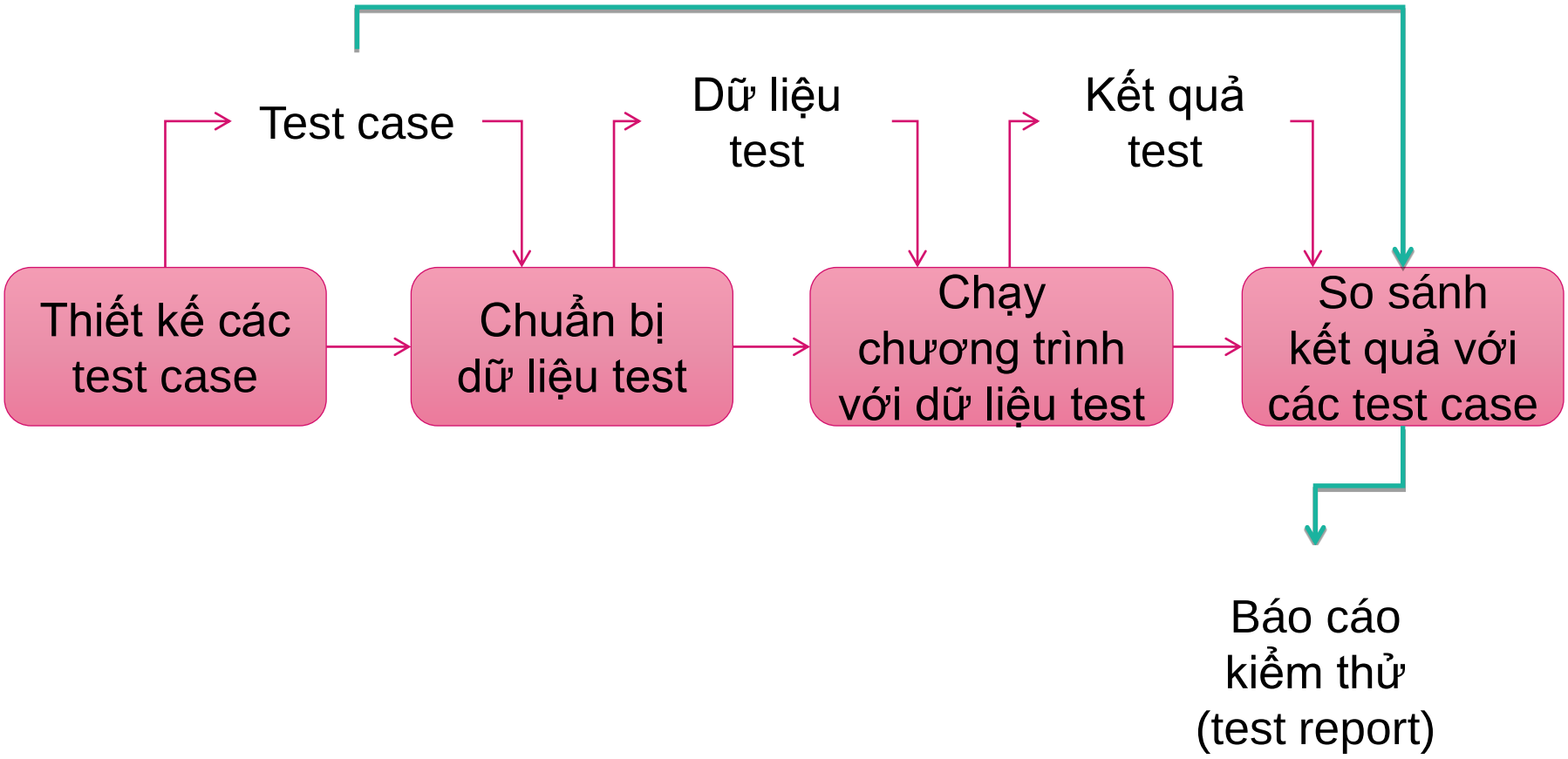
Defect testing

- 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

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

Quy trình kiểm thử phần mềm



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.

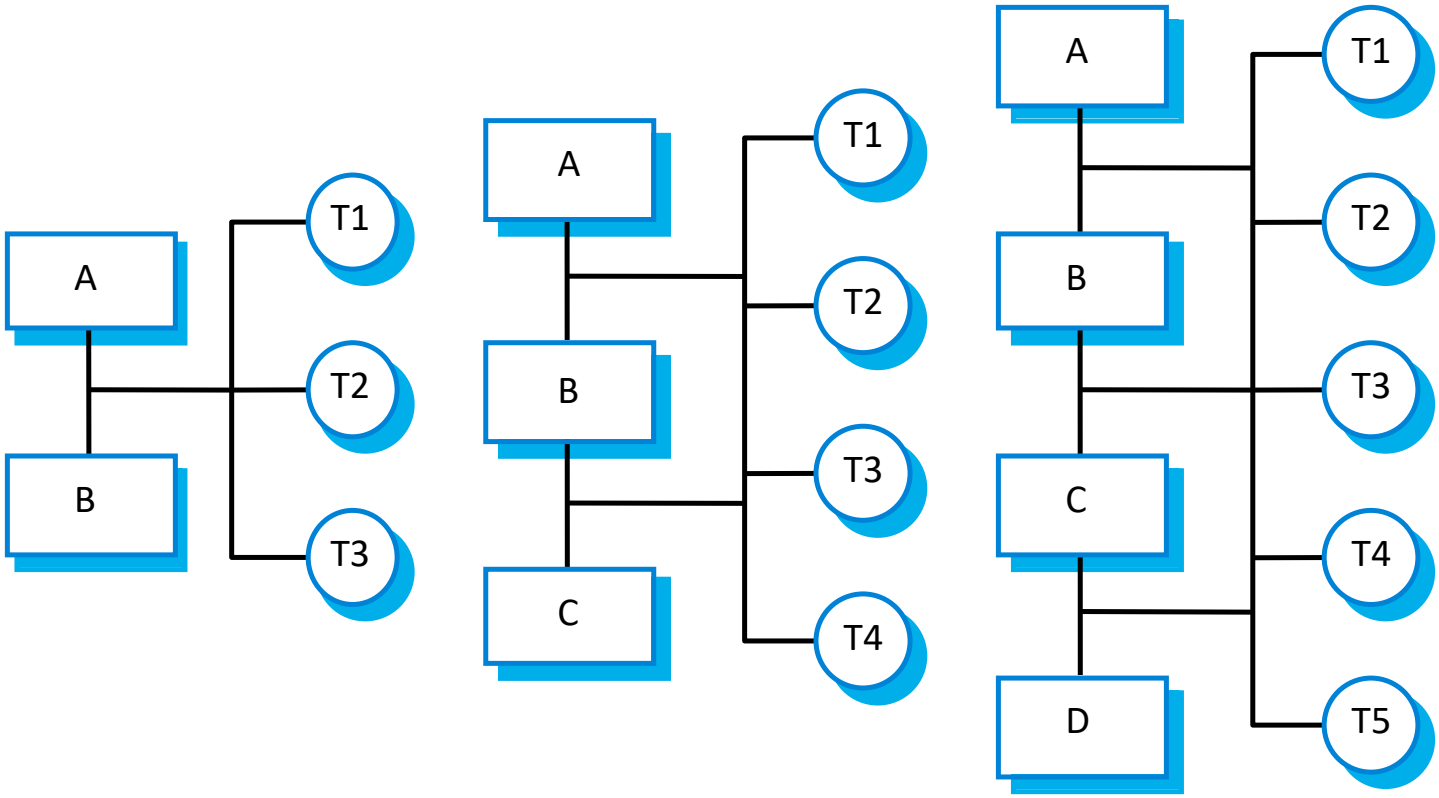
System testing – kiểm thử hệ thống

- 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

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

Incremental integration testing
kiểm thử tích hợp tăng dần



Test sequence 1

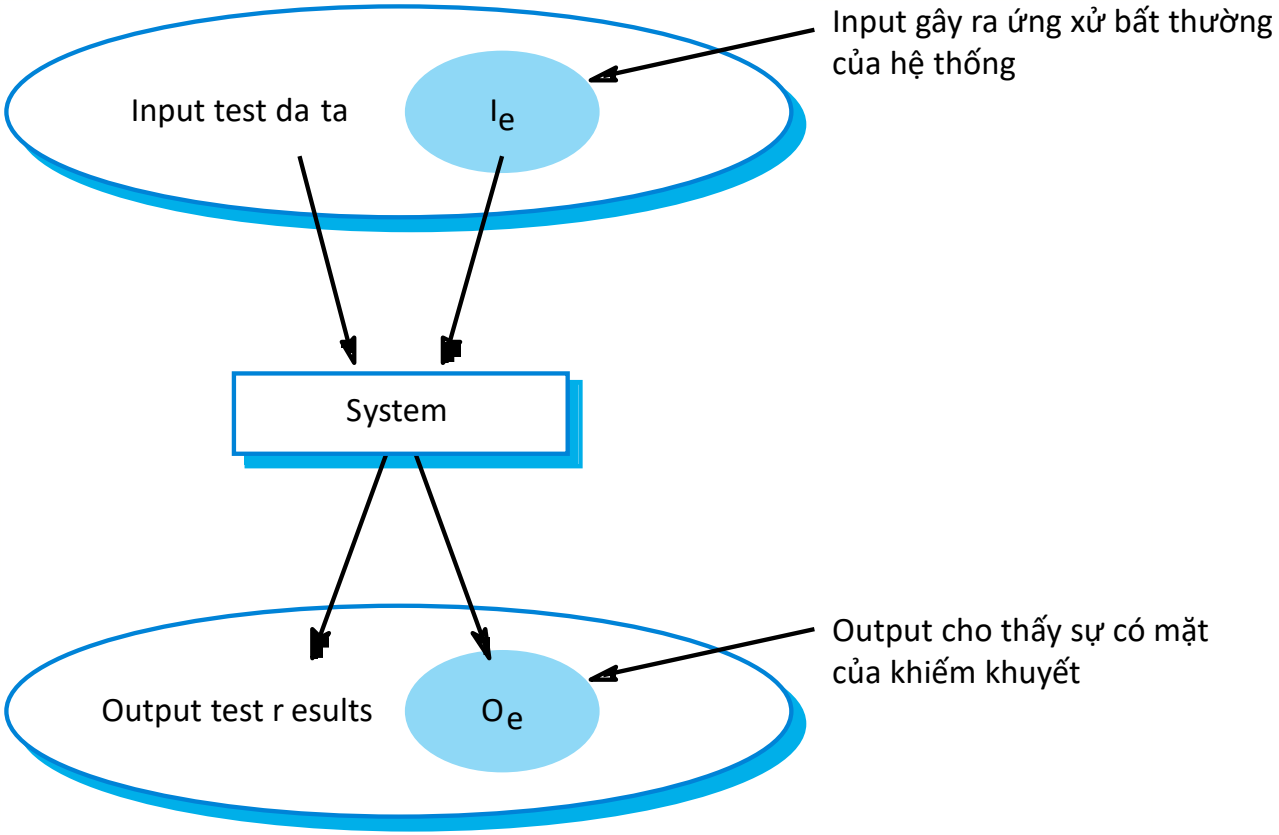
Test sequence 2

Test sequence 3

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 hăng của hệ thống
- Nếu có khách hàng tham gia kiểm thử thì còn gọi là acceptance test

Black-box testing
kiểm thử hộp đen



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

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.

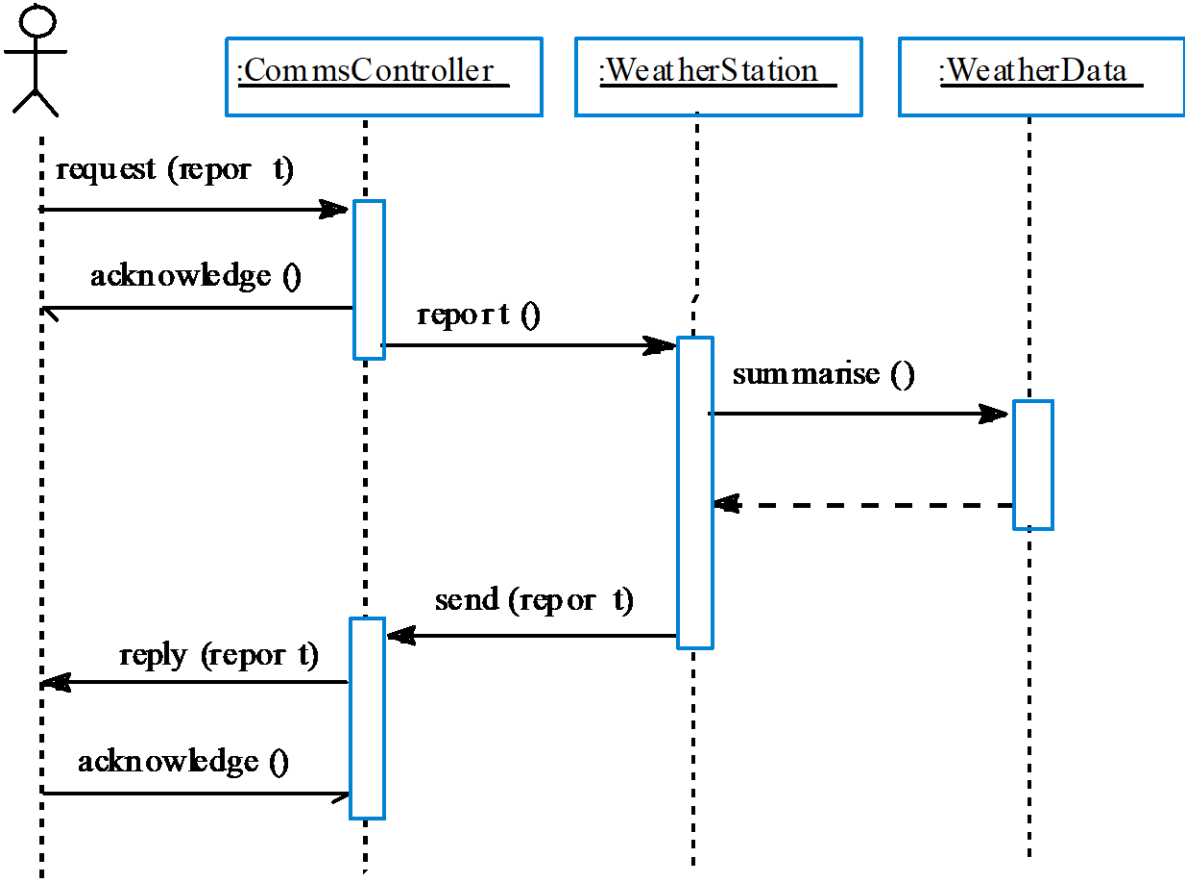
System tests – các test hệ thống

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.

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

Collect weather data sequence chart



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

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

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.

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

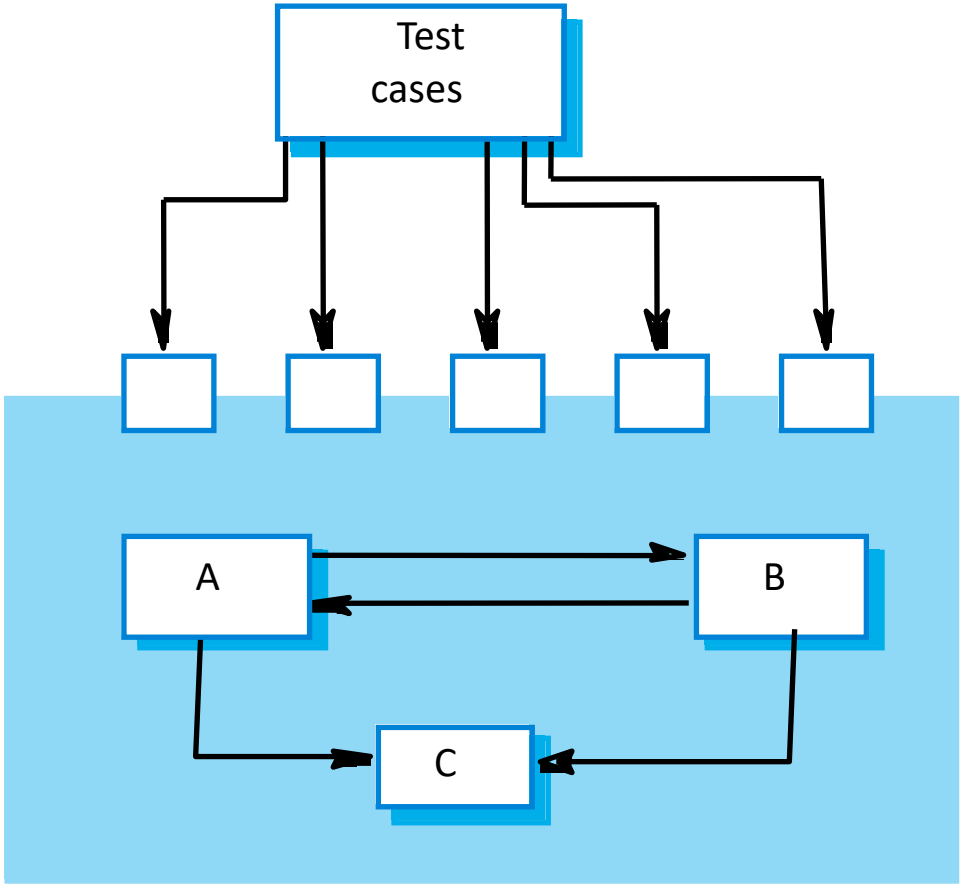
Object class testing

- 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.

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

Interface testing



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

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

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.

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.

Requirements based testing

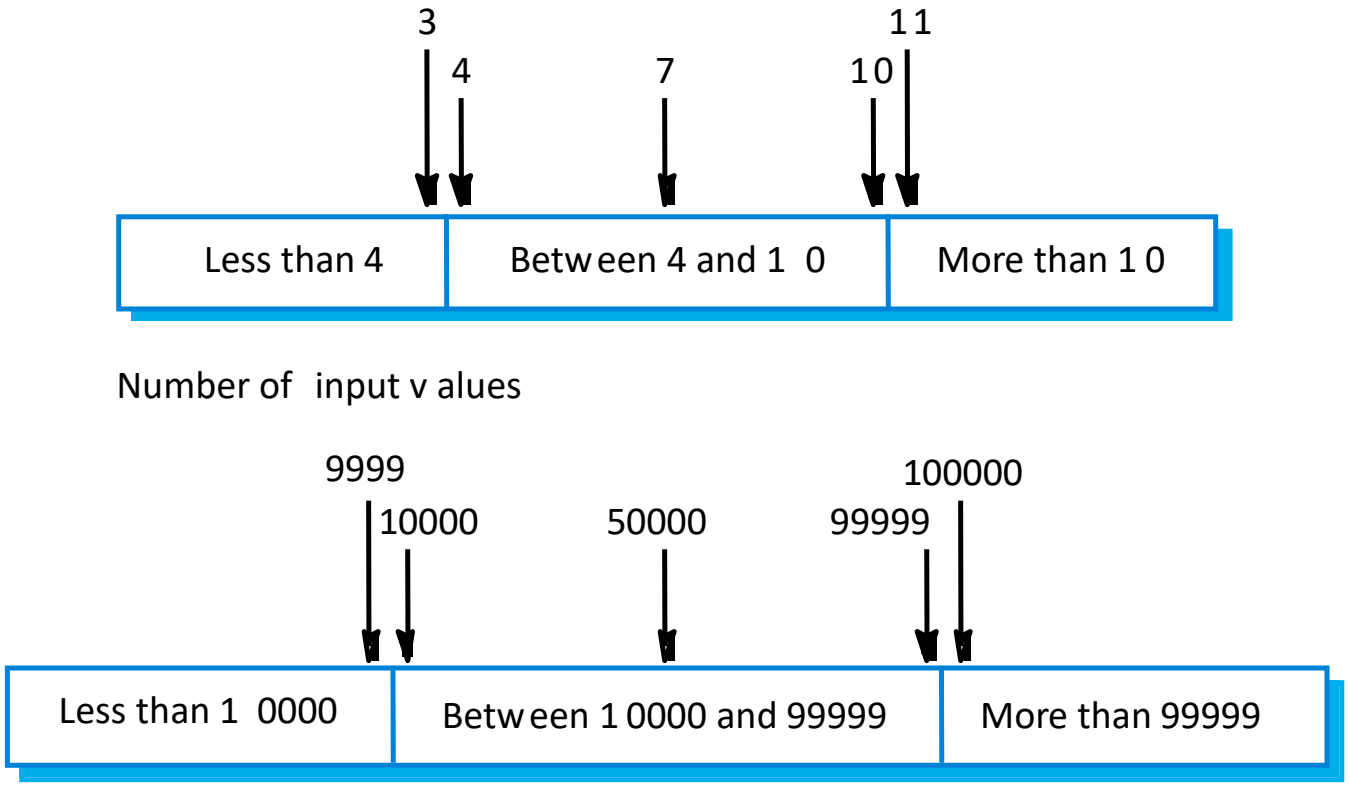
- Một nguyên tắc chung của kĩ nghệ yêu cầu là các yêu cầu đều phải test được
- Requirements-based testing là một kĩ thuật validation testing mà trong đó bạn xét từng yêu cầu và xác định một tập các test case nhằm chứng tỏ rằng hệ thống thỏa mãn yêu cầu đó.

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

Equivalence partitions



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 ))
```

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

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

Search routine - input partitions

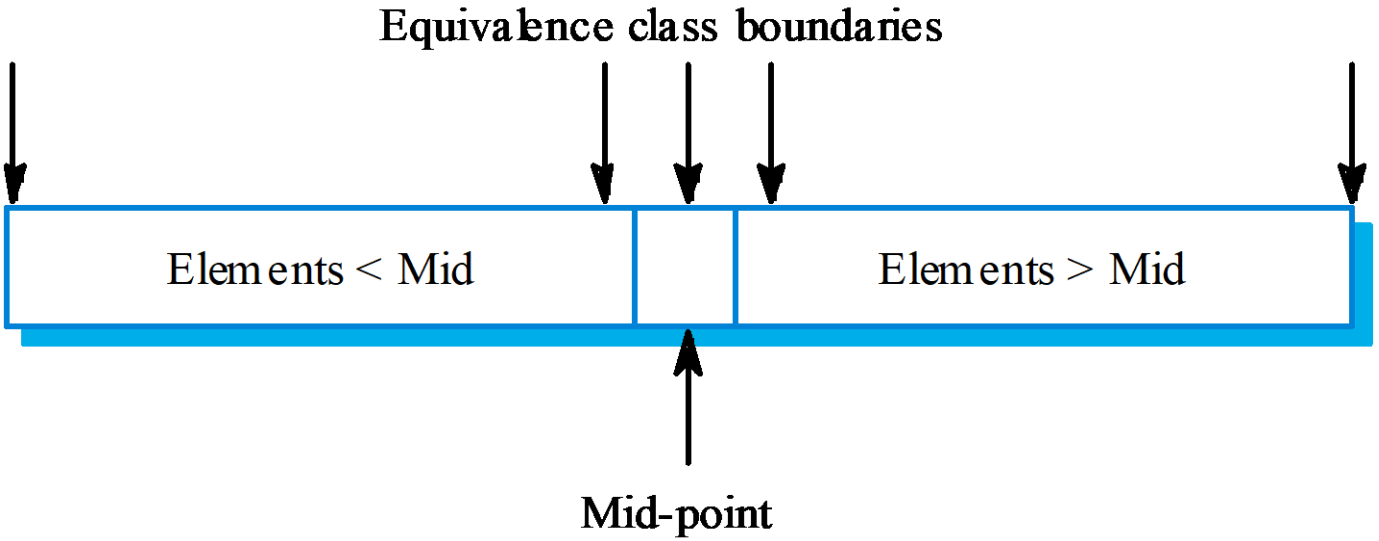
Sequence	**Element**
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

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
21, 23, 29, 33, 38	25	false, ??

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.

Binary search equiv. partitions



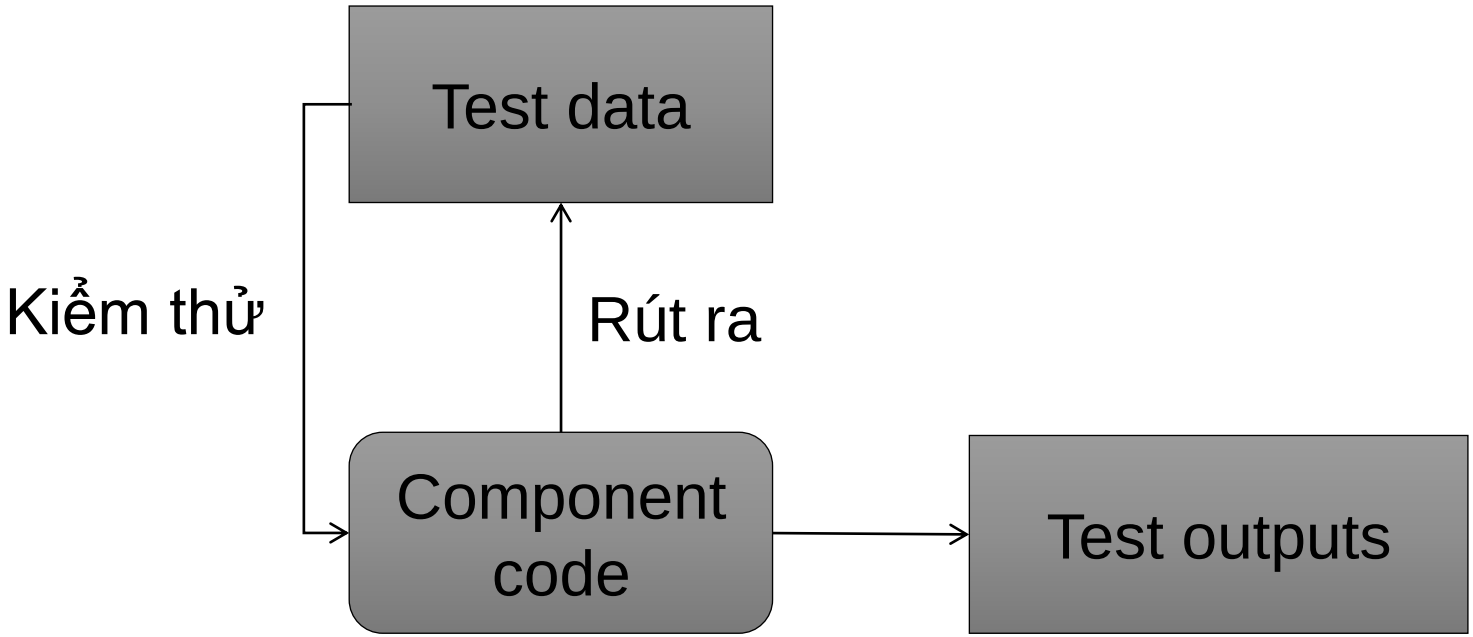
Binary search - test cases

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, ??

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)).

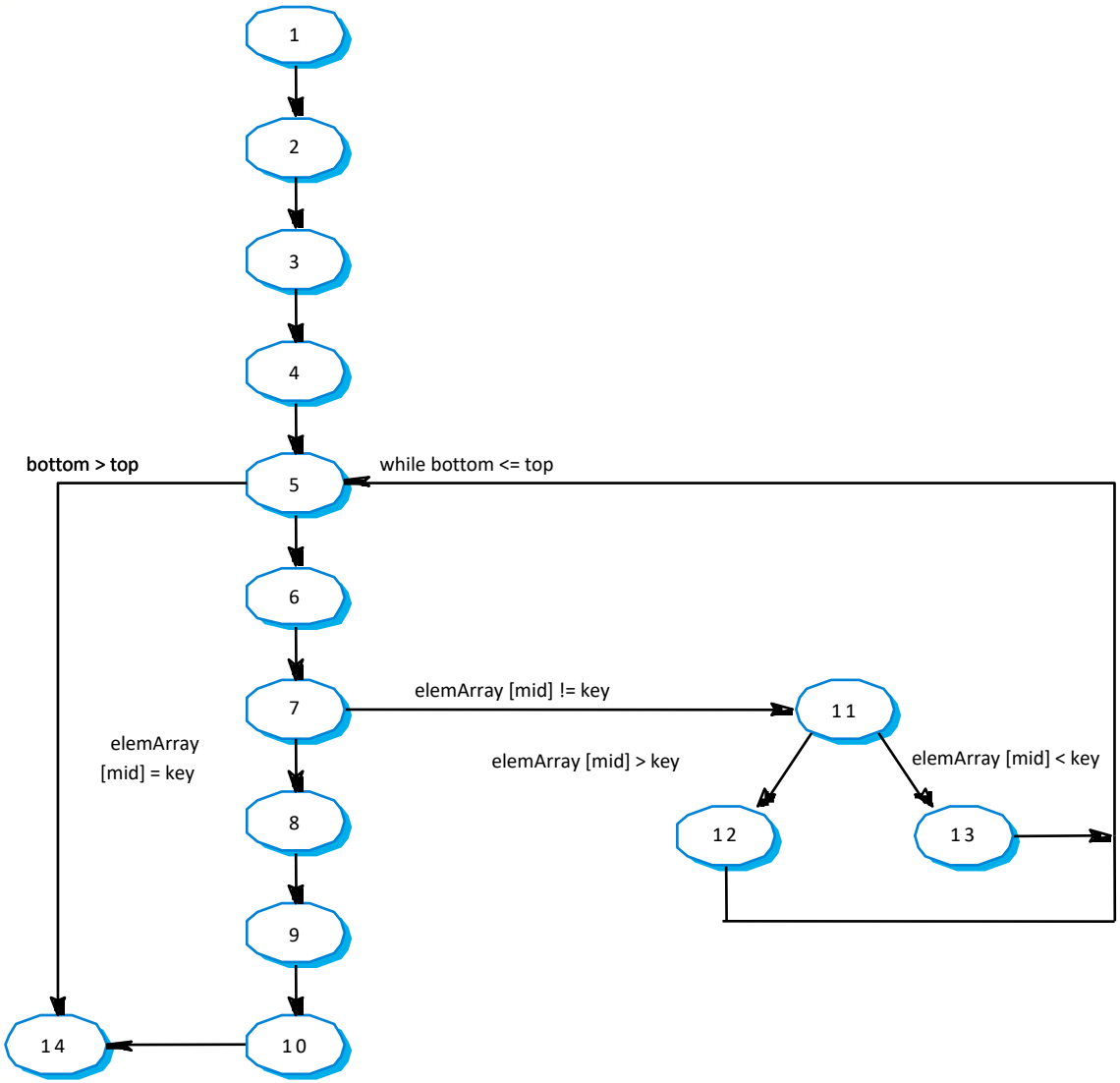
Structural testing



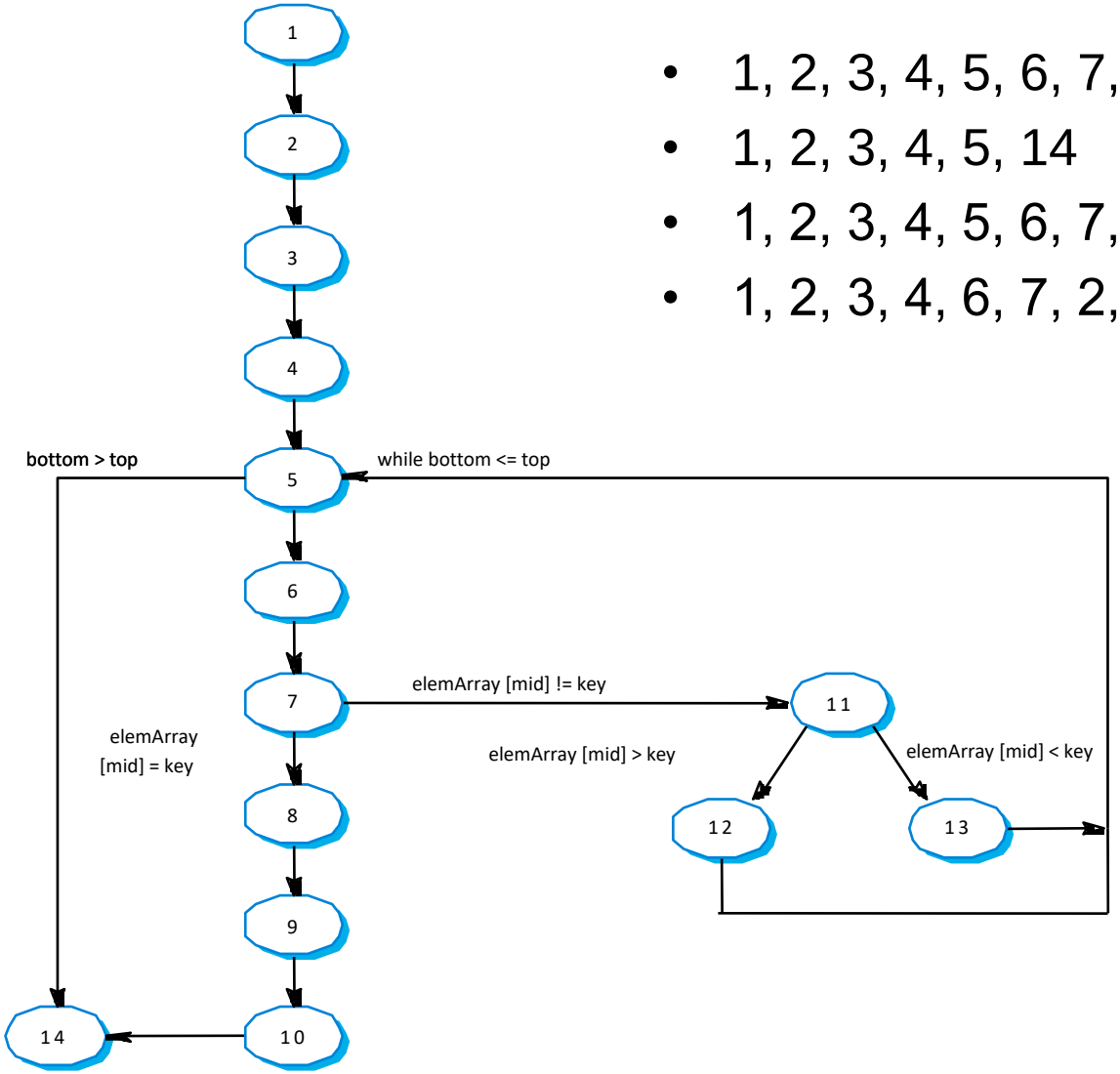
Path testing

- 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.

Binary search flow graph



Independent paths



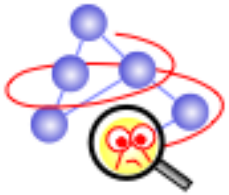
- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …

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

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.



Java PathFinder

Pex

Coding Duel for fun

Tóm tắt

- 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.

Tóm tắt

- 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.

Bài tập về nhà

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.