Software Quality Assurance

Đỗ Thị Bích Ngọc

PTIT/FIT/SE

ngocdtb@ptit.edu.vn

Software Testing

Đặc tả và lỗi phần mềm

Đặc tả

"if you can't say it, you can't do it"

- Ta cần biết sản phẩm như thế nào trước khi ta có thể nói nó có lỗi.
- Đặc tả định nghĩa sản phẩm và:
  - Yêu cầu chức năng mô tả các tính năng của sản phẩm. Ví dụ, calculator:
    - Save, +, -, *, /, …
  - Yêu cầu phi chức năng là các ràng buộc về sản phẩm. Ví dụ ,
    - thân thiện với người dùng, hiệu năng, …

Đặc tả và lỗi phần mềm

Lỗi phần mềm

- Phần mềm KHÔNG làm nhiệm vụ được đưa ra ở đặc tả (ví dụ, thiếu phép trừ)
- Phần mềm làm công việc mà đặc tả KHÔNG cho phép
- Phần mềm làm công việc mà đặc tả không đề cập tới (ví dụ, tính căn bậc hai của số nguyên)
- Phần mềm không làm công việc mà đặc tả không đề cập tới nhưng nên làm (ví dụ, kiểm tra divided by 0)
- Phần mềm khó hiểu, khó dùng, chậm …

Chi phí sửa lỗi

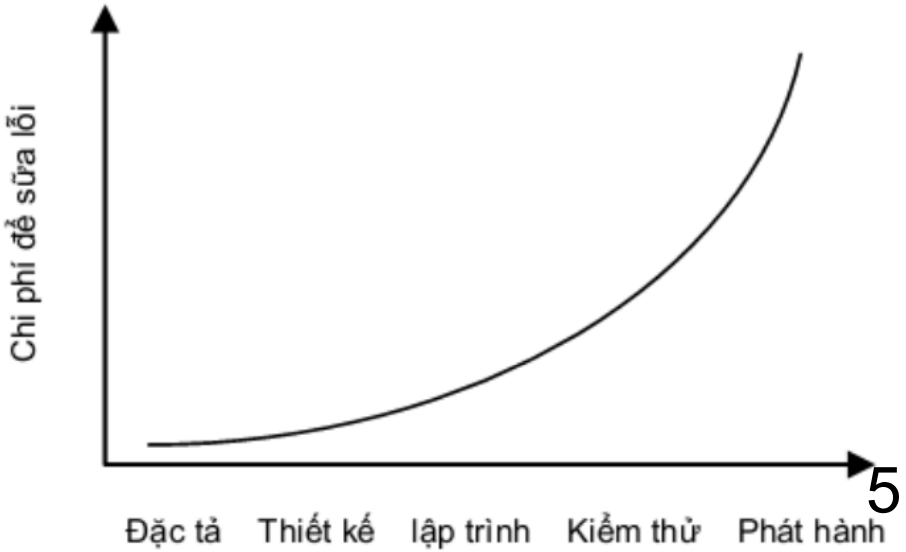
Chi phí sửa lỗi tăng theo cấp số nhân (10x) theo thời gian

Ví dụ, một lỗi phát hiện trong pha đặc tả tốn $1 để sửa.

… nếu phát hiện trong pha thiết kế, tốn $10

… nếu phát hiện trong pha cài đặt, tốn $100

… nếu phát hiện sau khi phát hành, tốn $1000



Kiểm thử - Testing

Kiểm thử phần mềm - KTPM

- KTPM là quá trình khảo sát một hệ thống hay thành phần dưới những điều kiện xác định, quan sát và ghi lại các kết quả, và đánh giá một khía cạnh (IEEE Standard Glossary of Software Engineering Terminology).
- KTPM là quá trình thực thi một chương trình với mục đích tìm lỗi. (“The Art of Software Testing”).
- Một cách dễ hiểu: KTPM là quá trình thực thi một hệ thống phần mềm để xác định xem phần mềm có đúng với đặc tả không và môi trường hoạt động có đúng yêu cầu không.

Software testing objectives

- Các mục tiêu trực tiếp
  - Xác định và phát hiện nhiều lỗi nhất có thể trong phần mềm được kiểm thử
  - Sau khi sửa chữa các lỗi đã xác định và kiểm tra lại, làm cho phần mềm đã được kiểm thử đến một mức độ chấp nhận được về chất lượng.
  - Thực hiện các yêu cầu kiểm thử cần thiết một cách hiệu quả và có hiệu quả, trong phạm vi ngân sách và thời gian cho phép.
- Các mục tiêu gián tiếp
  - Biên dịch một bản ghi về các lỗi phần mềm để sử dụng trong công tác phòng chống lỗi (bằng các hành động khắc phục và ngăn ngừa).

Ca kiểm thử (test case)

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

Nội dung của test case

Tên module/chức năng muốn kiểm thử dữ liệu vào

- dữ liệu thông thường: số, xâu kí tự, file,...
- môi trường thử nghiệm: phần cứng, OS,...
- thứ tự thao tác (khi kiểm thử giao diện)

Kết quả mong muốn

- thông thường: số, xâu kí tự, file,...
- màn hình, thời gian phản hồi

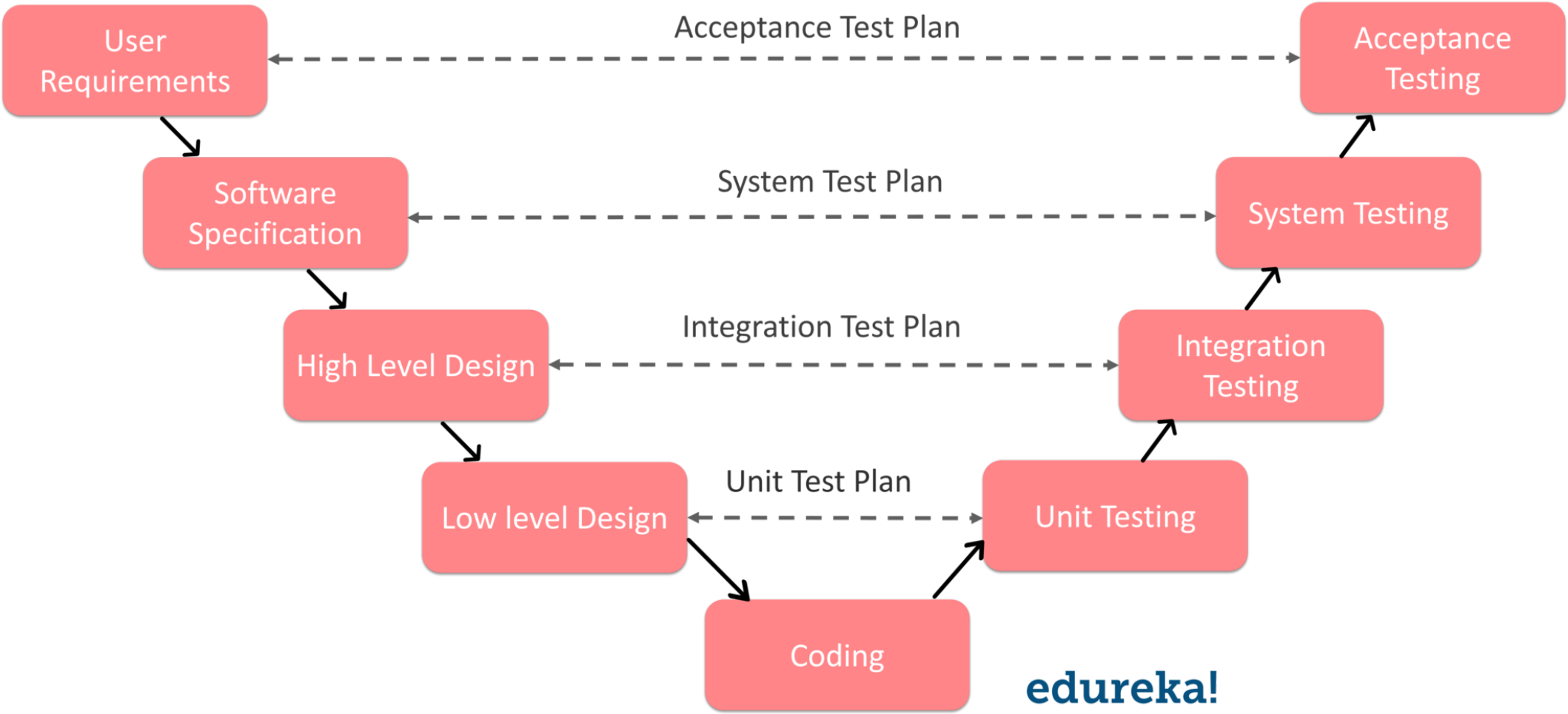
Kết quả thực tế

Khó khăn của kiểm thử

"Testing shows the presence, not the absence of bugs." —Edsger W. Dijkstra

- A key tradeoff of testing:
  - Kiểm thử càng nhiều trường hợp tiềm năng càng tốt nhưng phải trong phạm vi tài chính cho phép
- Mục tiêu là tìm bugs càng rẻ & nhanh càng tốt.
  - Tư tưởng: thiết kế "right" test case mà phát hiện ra fault và chạy nó
- Trên thực tế, ta phải thực hiện nhiều "unsuccessful" test cases mà không phát hiện ra bugs nào.

Các mức(cấp độ) kiểm thử
Testing level


User Requirements
Acceptance Test Plan
Acceptance Testing
Software Specification
System Test Plan
System Testing
High Level Design
Integration Test Plan
Integration Testing
Low level Design
Unit Test Plan
Unit Testing
Coding
edureka!

Unit Testing

- Kiểm thử đơn vị là hoạt động kiểm thử nhỏ nhất Kiểm thử thực hiện trên các hàm hay thành phần riêng lẻ.
- Cần hiểu biết về thiết kế chương trình và code.
- Thực hiện bởi Lập trình viên (không phải kiểm thử viên)
- **Đơn vị:** Là thành phần nhỏ nhất của phần mềm có thể kiểm thử được. Ví dụ: Các hàm, lớp, thủ tục, phương thức.

Integration Testing

- Kiểm thử tích hợp nhằm phát hiện lỗi giao tiếp xảy ra giữa các thành phần cũng như lỗi của bản thân từng thành phần (nếu có).

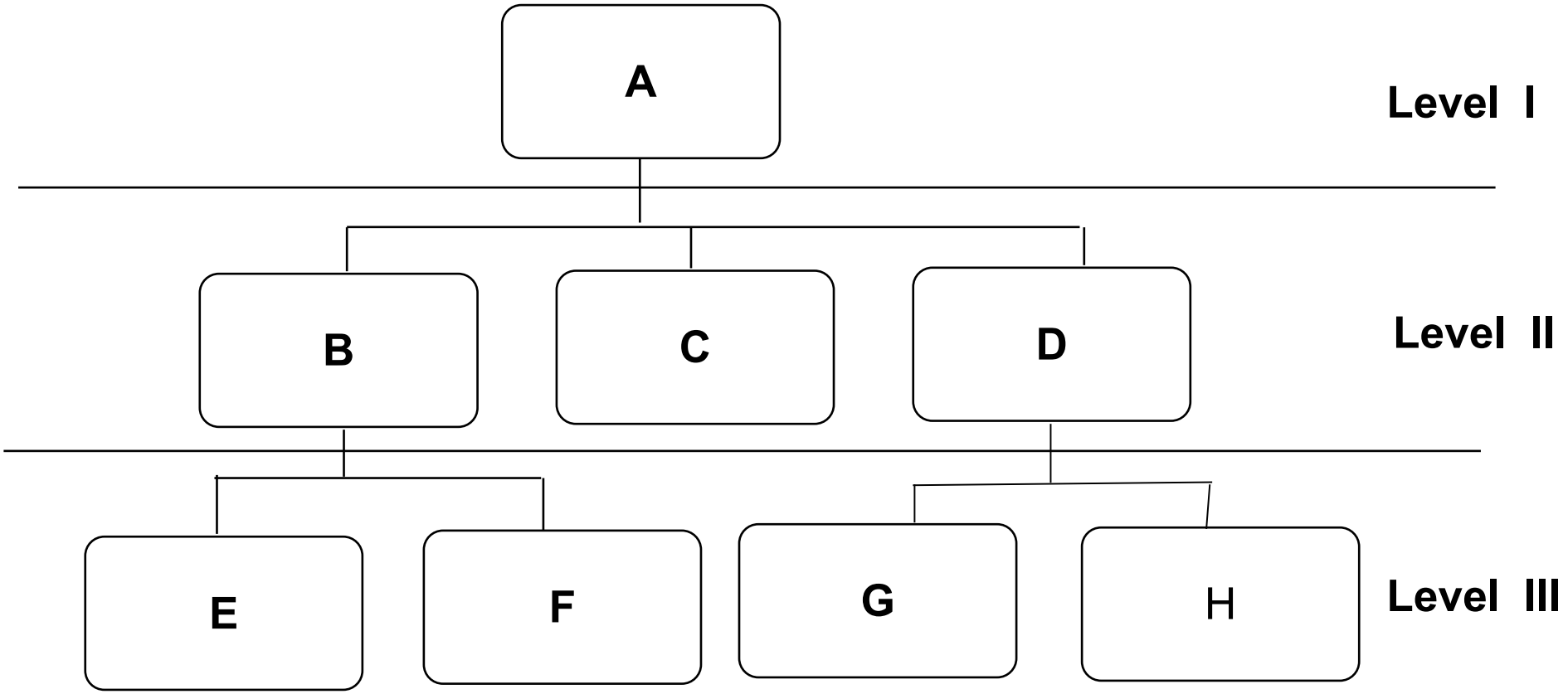
- *Thành phần* có thể là
  - các module
  - các ứng dụng riêng lẻ
  - các ứng dụng client/server trên một mạng

Integration Testing Strategy

- Một system được tạo bởi 1 tập subsystems (sets of classes) xác định trong quá trình thiết kế hệ thống

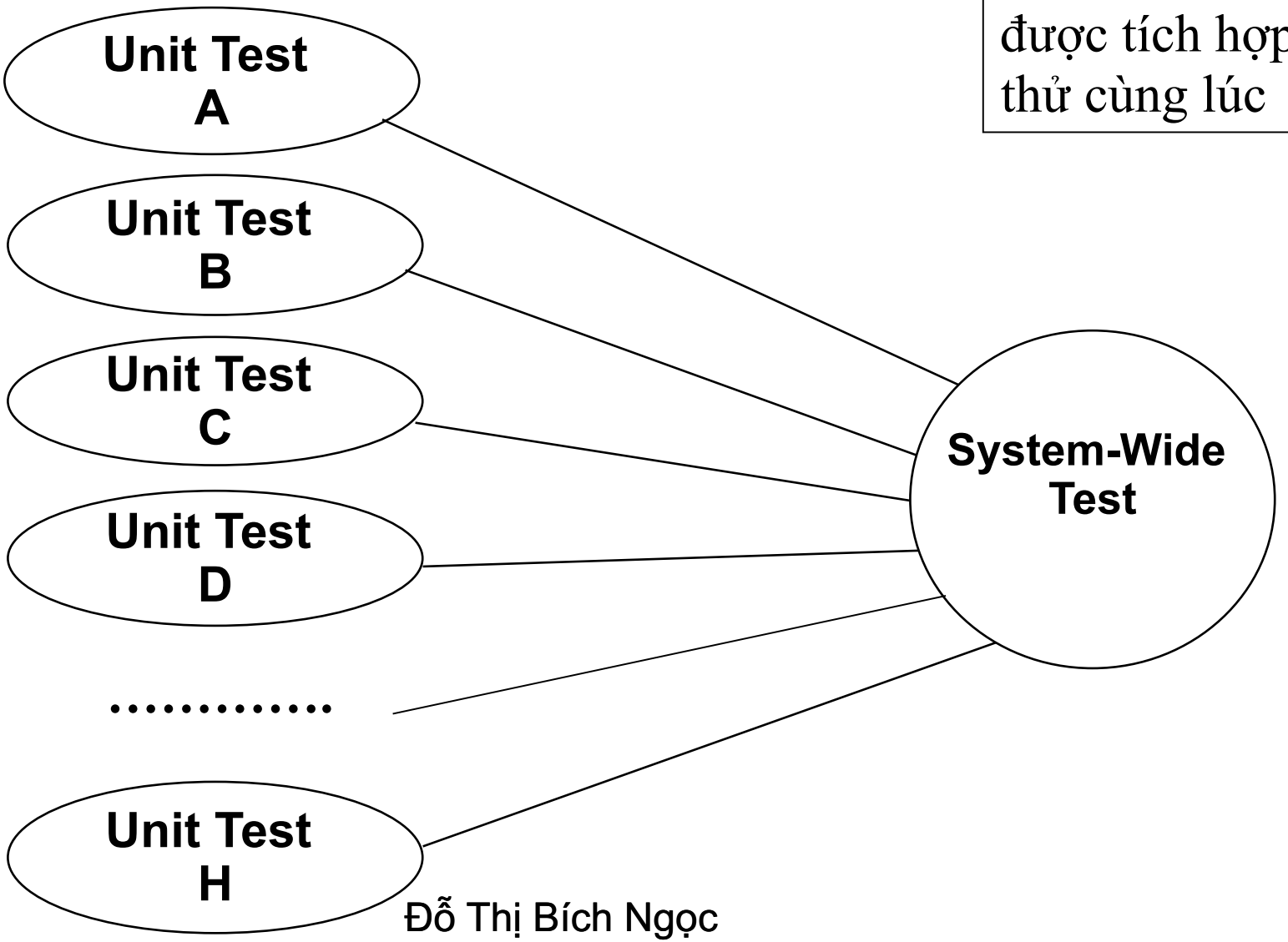
- Thứ tự các subsystems được lựa chọn để kiểm thử và tích hợp xác định loại kiểm thử
  - Big bang integration (Nonincremental)
  - Bottom up integration
  - Top down integration
  - Sandwich testing
  - Biến thể của các loại trên

Ví dụ: hệ thống 3 mức gọi



Big-Bang Integration Testing

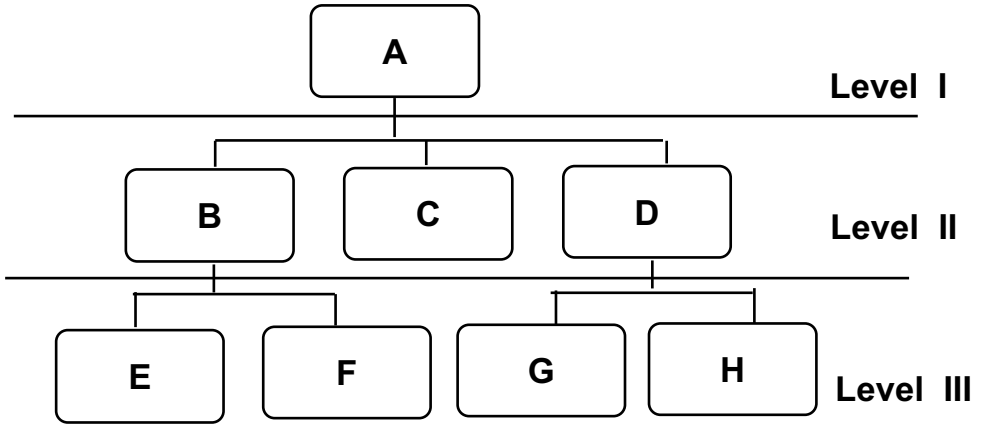
Tất cả các thành phần được tích hợp và kiểm thử cùng lúc

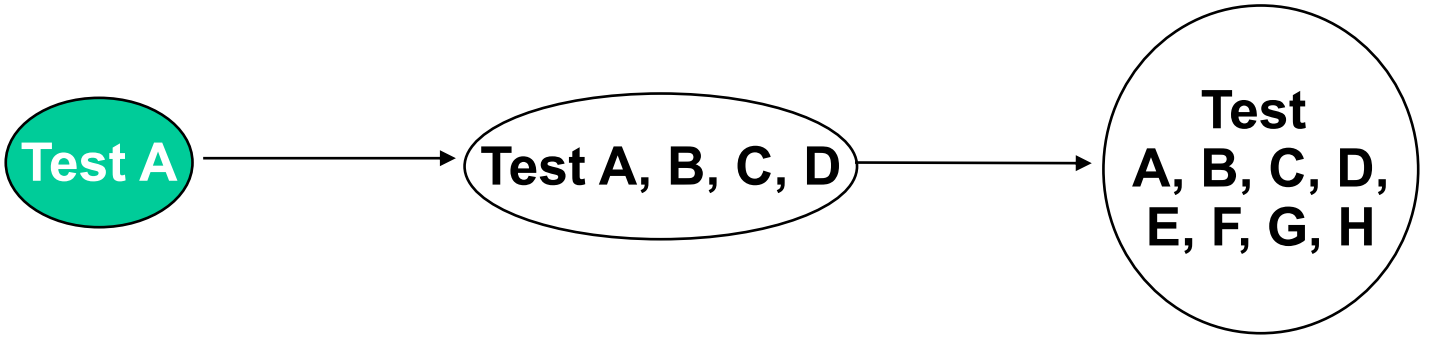


Chiến lược kiểm thử tích hợp Top-Down

- Chiến lược kiểm thử tích hợp Top-down thực hiện kiểm thử top layer đầu tiên (hàm *main*, hoặc gốc của call tree)
- Thông thường, ta sẽ thêm dần subsystems mà được referenced/required bởi các hệ thống con đã test. Lặp lại cho tới khi tất cả referenced/required được kiểm thử hết
- Cần có một chương trình đặc biệt để thực hiện kiểm thử, gọi là *Test stub:*
  - Một program hay method mà mô phỏng input-output của subsystem còn thiếu bằng cách trả lời các lời gọi subsystem đó và trả về kết quả mô phỏng, còn gọi là "canned" data.

Top-down Integration Testing Strategy





Level I Unit(s)

Level I and II Units

All Units – all levels

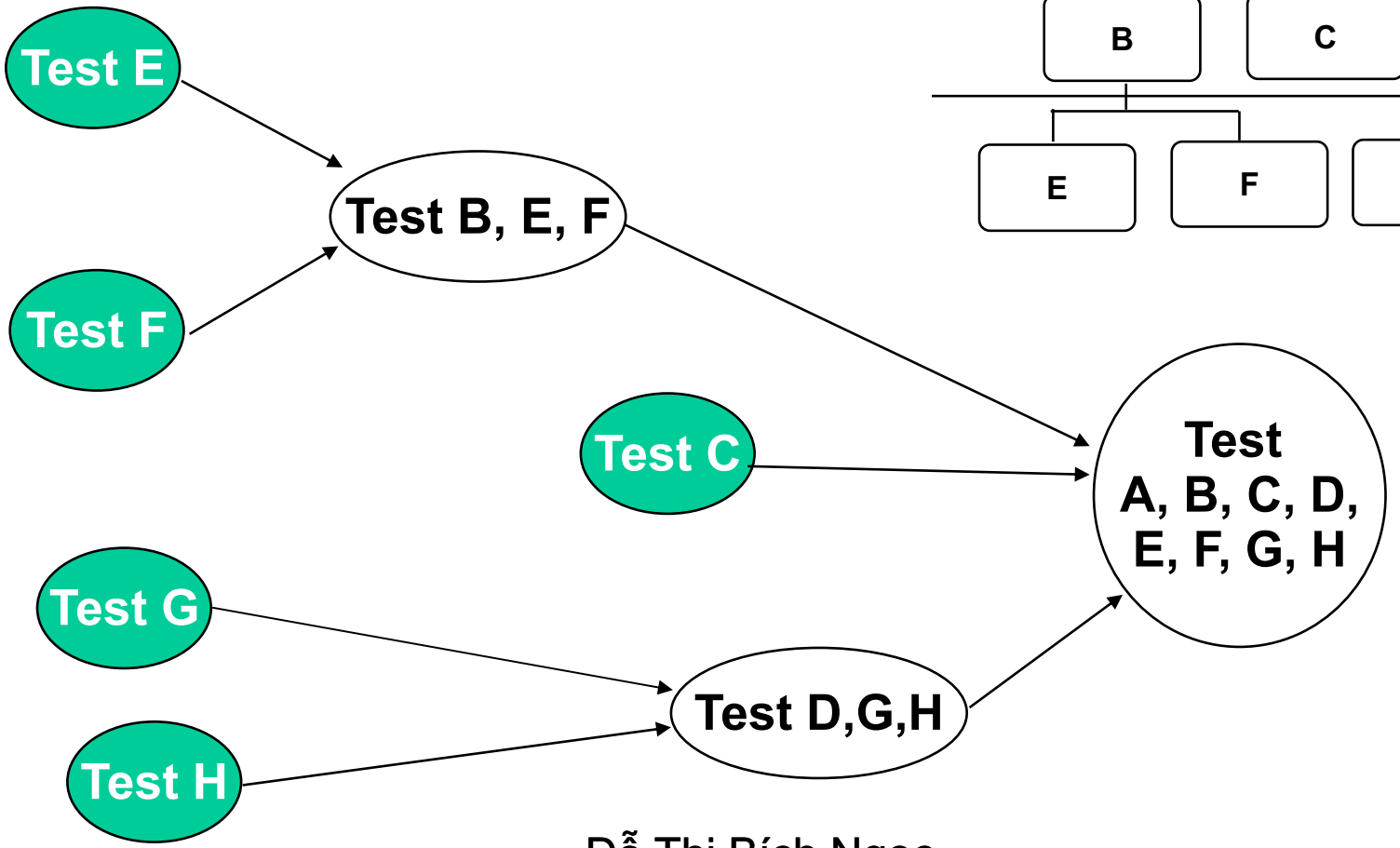
Ưu nhược của Top-Down Integration Testing

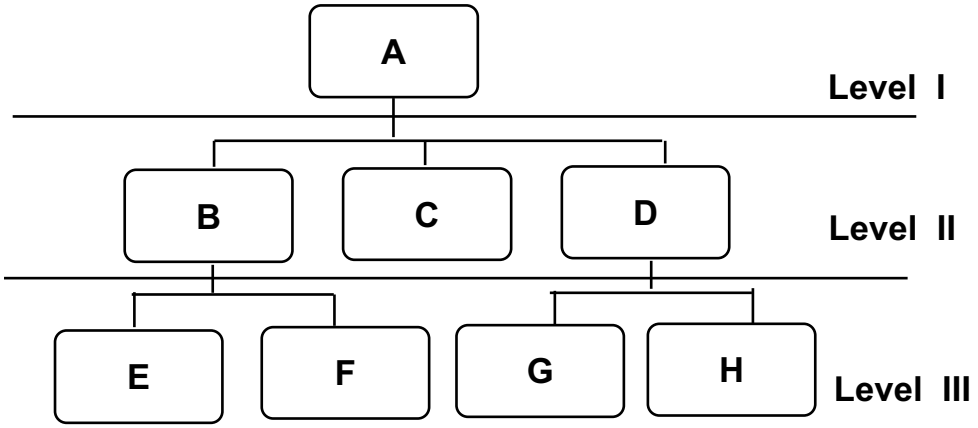
- Test cases được định nghĩa dựa theo chức năng của hệ thống (functional requirements).
- Viết stubs có thể khó, nhất là khi tham số truyền vào phức tạp. Stubs phải cho phép kiểm thử tất cả các điều kiện có thể xảy ra được
- Có thể cần một lượng lớn stubs, nhất là khi mức thấp nhất của hệ thống có nhiều lời gọi hàm
- Một giải pháp để hạn chế stubs: *Modified top-down testing strategy (Bruege)*
  - kiểm thử mỗi layer của hệ thống phân rã độc lập rồi kết hợp các layers sau
  - nhược điểm: cần cả stubs và drivers

Bottom-Up Integration Testing Strategy

- Thực hiện kiểm thử units ở level thấp nhất (units ở mức lá của cây phân rã decomposition tree)
- Thêm dần các subsystems mà reference/require các subsystems đã test
- Lặp lại cho tới khi tất cả các subsystems đã được thêm vào
- Cần một chương trình đặc biệt gọi là *Test Driver* ,
  - Test Driver làl một “fake” routine mà gọi tới/tham chiếu một subsystem và truyền một 1 test case cho nó

Example Bottom-Up Strategy





Ưu nhược của Bottom-Up Integration Testing

- Chưa tối ưu hoá việc phân cấp chức năng:
  - Tests subsytem quan trọng nhất sau cùng (UI)
- Hiệu quả cho tích hợp các hệ thống sau
  - Object-oriented systems
  - Real-time systems
  - Systems với yêu cầu hiệu năng cao

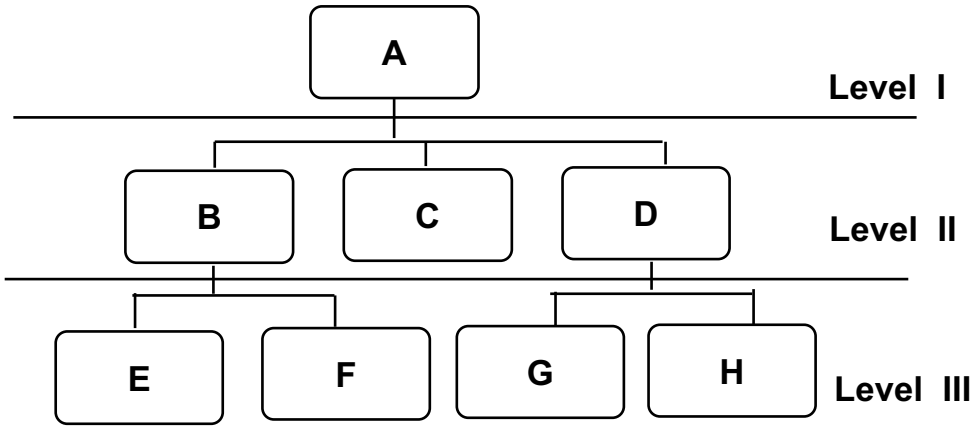
Sandwich Testing Strategy

- Kết hợp 2 chiến lược top-down và bottom-up

- *Hệ thống được xem như có 3 layers*
  - Layer cần test là ở giữa
  - Một layer ngay trên layer cần test
  - Một layer ngay dưới layer cần test
  - Kiểm thử tập trung vào layer cần test

- Làm thế nào để xác định layer cần test nếu có nhiều hơn 3 layers?
  - Heuristic: giảm thiểu số stubs và drivers

Sandwich Testing Strategy



Bottom Level Tests

Test E

Test F

Test G

Test H

Test B, E, F

Test D,G,H

Test A,B,C, D

Test A

Top Level Tests

Test A, B, C, D, E, F, G, H

System test

•Kiểm thử hệ thống là một mức của tiến trình kiểm thử phần mềm khi các module và tích hợp các module đã được test.

•Mục tiêu của kiểm thử hệ thống là để đánh giá phần mềm có tuân thủ theo các yêu cầu đã đưa ra không.

Taxonomy of System Tests

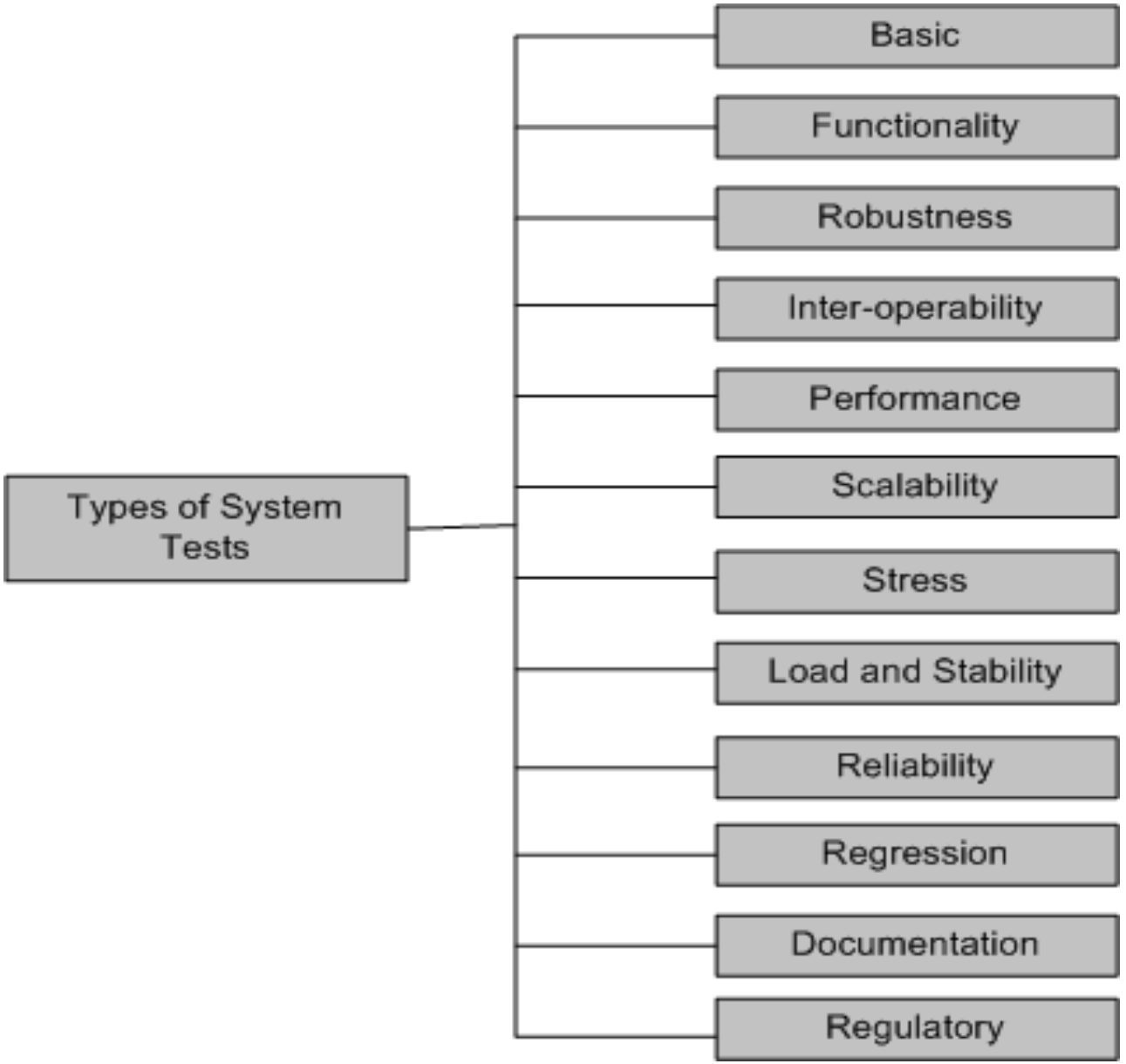


Figure 8.1: Types of system tests

29

Phân loại System Tests

Basic tests để chứng tỏ hệ thống có thể cài đặt được, cấu hình được và hoạt động được

Functionality tests cung cấp kiểm tra toàn bộ yêu cầu (requirements) trên cả hệ thống

Robustness tests xác định xem khả năng phục hồi của hệ thống từ input errors và các tình huống failure khác

Inter-operability tests xác định xem hệ thống có thể tương thích (inter-operate) với các sản phẩm của bên thứ 3

Performance tests đánh giá hiệu năng của hệ thống, e.g., băng thông, phản hồi dưới các điều kiện khác nhau

Phân loại System Tests

Scalability tests xác định giới hạn quy mô của hệ thống, như quy mô người dùng, quy mô địa lý, quy mô nguồn lực

Stress tests để hệ thống ở tình trạng áp lực (stress) để xác định giới hạn của hệ thống và, khi nó fail, xác định cách thức để gây ra failure

Load and Stability kiểm tra khả năng ổn định của hệ thống trong thời gian dài với toàn tải (full load)

Reliability tests đánh giá khả năng hệ thống giữ hoạt động trong thời gian dài mà không gây ra failures

Regression tests kiểm tra hệ thống vẫn ổn định khi tích hợp thêm các hệ thống con khác và khi bảo trì

Documentation tests đảm bảo system's user guides là chính xác và khả dụng

Acceptance Testing

Gồm hai loại kiểm thử gọi là

- Alpha Test, người dùng kiểm thử phần mềm ngay tại nơi phát triển phần mềm, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa.
- Beta Test, phần mềm sẽ được gửi tới cho người dùng để kiểm thử ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Acceptance Testing

- Kiểm thử chấp nhận là một cấp độ trong tiến trình kiểm thử phần mềm nhằm kiểm thử hệ thống về khả năng chấp nhận được.
- Mục tiêu của kiểm thử này là để đánh giá sự tuân thủ của hệ thống với các yêu cầu nghiệp vụ và thẩm định xem đã có thể chấp nhận để bàn giao chưa.
- Kiểm thử chấp nhận được khách hàng thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện).

Acceptance Testing

Gồm hai loại kiểm thử gọi là

- Alpha Test, người dùng kiểm thử phần mềm ngay tại nơi phát triển phần mềm, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa.
- Beta Test, phần mềm sẽ được gửi tới cho người dùng để kiểm thử ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.