KHOA CÔNG NGHỆ THÔNG TIN
Bộ môn Hệ thống thông tin & Toán ứng dụng

BÀI GIẢNG CÔNG NGHỆ PHẦN MỀM

(COURSE OF SOFTWARE ENGINEERING)

Th.s Huỳnh Xuân Hiệp
Điện thoại: 84.71.831301
E-mail: hxhiep@ctu.edu.vn

TÀI LIỆU THAM KHẢO

1. [Schach1999] Stephen R. Schach, *Classical and object-oriented software engineering*, McGRAW-HILL Inc, 1999,1996.
2. [Quang2000] Trương Minh Nhật Quang, *Bài giảng Công nghệ phần mềm*, Đại Học Cần Thơ, 2000.
3. [Tuyền2000] Trương Thị Thanh Tuyền, *Bài giảng Công nghệ phần mềm*, Đại Học Cần Thơ, 2000.
4. [Pressman1997] Roger S.Pressman, *Software engineering*, McGRAW-HILL Inc, 1997. (Sách dịch – Ngô Trung Việt)

NỘI DUNG MÔN HỌC

Phần 1 Giới thiệu về chu trình sống của phần mềm

1 Phạm vi của công nghệ phần mềm
2 Tiến trình phần mềm
3 Các mô hình chu trình sống của phần mềm
4 Nhóm làm việc và các công cụ nghề nghiệp
5 Kiểm thử
6 Giới thiệu về đối tượng
7 Một số vấn đề : sử dụng lại, dễ di chuyển và vận hành tương tác
8 Hoạch định và ước lượng

Phần 2 Các giai đoạn trong chu trình sống của phần mềm

9 Phân tích yêu cầu
10 Đặc tả
11 Thiết kế
12 Phân tích hướng đối tượng
13 Cài đặt
14 Cài đặt và tích hợp
15 Bảo trì

Phần

1

giới thiệu về chu trình sống của phần mềm

(INTRODUCTION TO THE SOFTWARE LIFE CYCLE)

1 phạm vi của công nghệ phần mềm

(SCOPE OF SOFTWARE ENGINEERING)

Nội dung:

- Lịch sử
- Kinh tế
- Bảo trì
- Đặc tả và thiết kế
- Đội ngũ lập trình
- Mô hình hướng đối tượng
- Thuật ngữ

1.1 Lịch sử
(historical aspects)

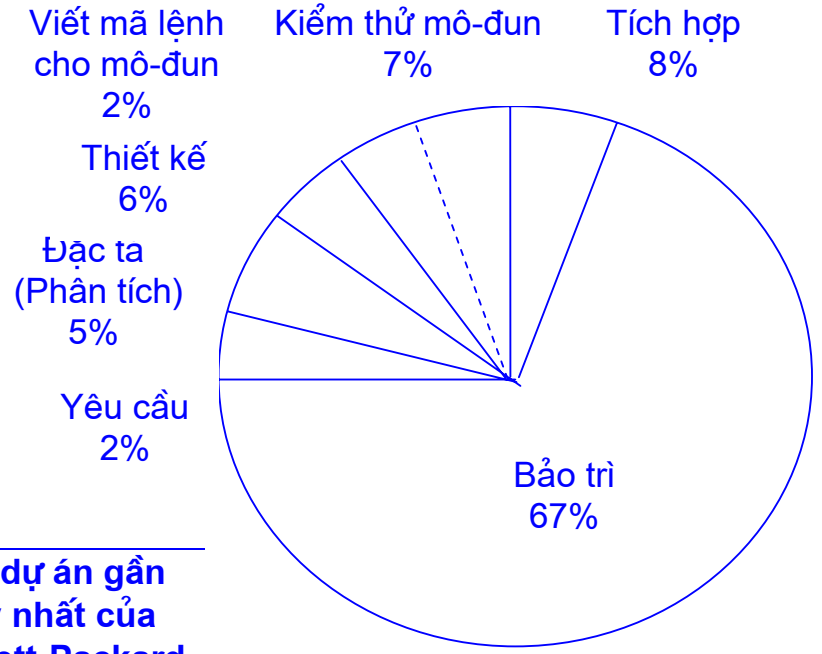
- Thuật ngữ công nghệ phần mềm (software engineering-SE) được đề xuất bởi một nhóm nghiên cứu của NATO vào năm 1967
- Hội nghị về SE được tổ chức tại Garmisch-Đức năm 1968 nhằm giải quyết vấn đề khủng hoảng phần mềm
- Cuộc khủng hoảng vẫn còn kéo dài đến nay vì hai lý do:
  - tiến trình sản xuất phần mềm có những thuộc tính và vấn đề riêng
  - sự trì trệ của phần mềm (software depression) với thời gian kéo dài và những dự đoán nghèo nàn
- Sự phát triển của phần cứng và hệ điều hành
  - hệ điều hành đa nhiệm (1960s)
  - bộ nhớ ảo (1970s)
  - đa xử lý (multiprocessor)
  - hệ điều hành phân tán (mạng),...
- Vấn đề bảo trì phần mềm

1.2 Kinh tế
(economic aspects)

- Sự lựa chọn kỹ thuật thực hiện nhanh hơn để giảm giá thành
- Sự ảnh hưởng của kỹ thuật mới lên công ty phần mềm
  - khó bảo trì
  - thời gian huấn luyện
  - kinh nghiệm làm việc trên kỹ thuật mới chưa nhiều
- Phụ thuộc vào sự lựa chọn của khách hàng
- Luật bản quyền

1.3 Bảo trì
(maintenance aspects)

- Chu trình sống của phần mềm
  - yêu cầu
  - đặc tả (phân tích)
  - thiết kế
  - cài đặt
  - tích hợp
  - bảo trì
  - kết thúc hoạt động



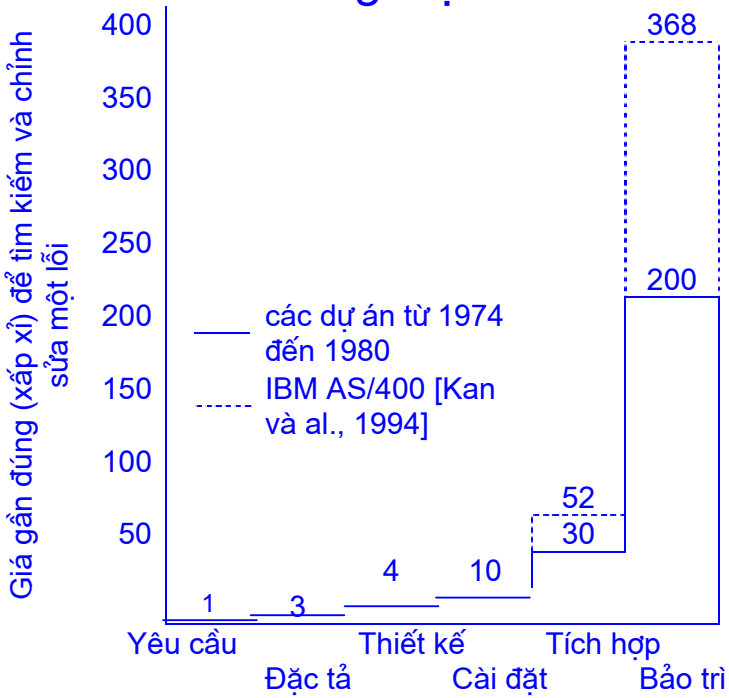
Hình 1.1 Giá thành của các giai đoạn trong chu trình sống của phần mềm

	Các dự án khác nhau từ 1976 đến 1981	132 dự án gần đây nhất của Hewlett-Packard
Giai đoạn yêu cầu và đặc tả	21%	18%
Giai đoạn thiết kế	18	19
Giai đoạn cài đặt	36	34
Giai đoạn tích hợp	24	29

Hình 1.2 Một số so sánh trên các dự án

1.4 Đặc tả và thiết kế
(specification and design aspects)

- Sai sót tại các giai đoạn trước sẽ ảnh hưởng đến các giai đoạn sau, tạo ra các lỗi. Công việc sửa chữa các lỗi càng sớm càng tốt là rất quan trọng



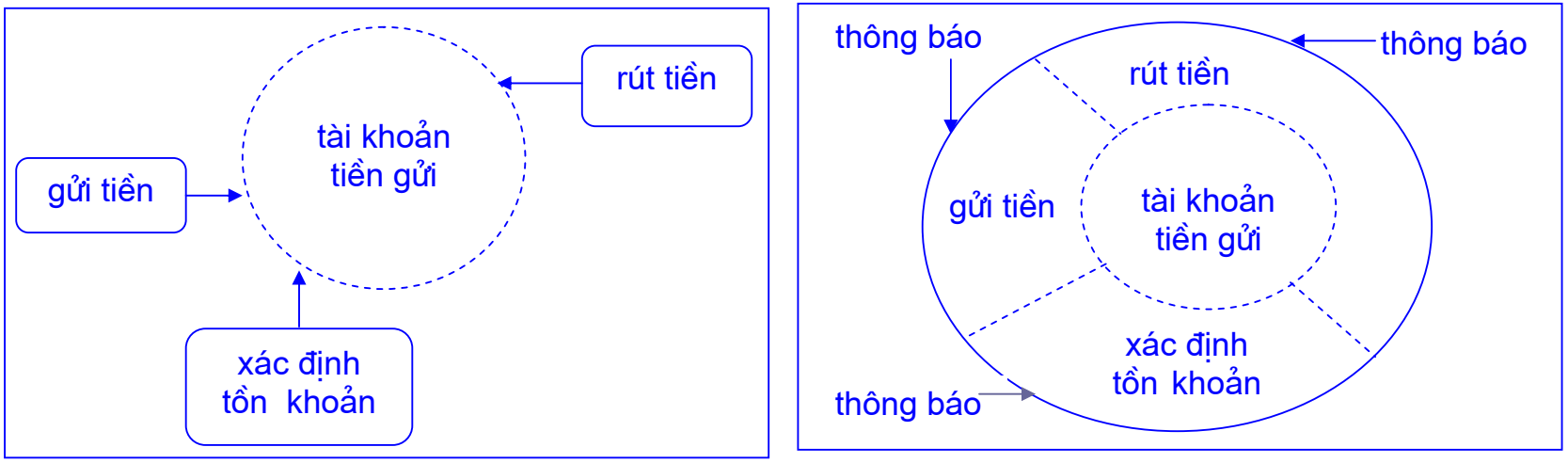
Hình 1.4 Giá phải trả để điều chỉnh lỗi

1.5Đội ngũ lập trình
(team programming aspects)

- Hình thành từng nhóm làm việc chuyên biệt trong từng lĩnh vực. Một số vấn đề nảy sinh như:
  - cách chia xẻ các phần công việc
  - mối quan hệ, sự giao tiếp giữa các thành viên với nhau
- Kỹ thuật tổ chức và quản lý đội ngũ phát triển phần mềm
  - lập trình viên
  - đặc tả viên
  - thiết kế viên,...
- Sự ràng buộc lẫn nhau giữa các thành viên cùng nhóm, khác nhóm,...
- Cách đánh giá thời gian làm việc
- Cách đánh giá hiệu quả công việc
- Cách đánh giá về kinh nghiệm thực hiện công việc

1.6 Mô hình hướng đối tượng
(the object-oriented paradigm)

- Dữ liệu và tác động có vai trò quan trọng như nhau. Một số tên gọi khác:
  - thiết kế hướng trách nhiệm (responsibility-driven design)[Wirfs-Brock, Wilkerson và Wiener, 1990]
  - thiết kế theo hợp đồng [Meyer, 1992a]



(a) (b)

Hình 1.4 So sánh hai phương pháp cài đặt *(a) cấu trúc* và *(b) hướng đối tượng*

Hướng cấu trúc	Hướng đối tượng
1. Yêu cầu	1. Yêu cầu
2. Đặc tả (phân tích)	2. (*) Đặc tả (phân tích) hướng đối tượng
3. Thiết kế	3. (*)Thiết kế hướng đối tượng
4. Cài đặt	4. (*)Lập trình hướng đối tượng
5. Tích hợp	5. Tích hợp
6. Bảo trì	6. Bảo trì
7. Kết thúc hoạt động	7. Kết thúc hoạt động

Hình 1.5 So sánh chu trình sống giữa hướng cấu trúc và hướng đối tượng

Hướng cấu trúc	Hướng đối tượng
2. Đặc tả (phân tích) • xác định sản phẩm phải làm gì	2. (*) Đặc tả (phân tích) hướng đối tượng • xác định sản phẩm phải làm gì • tạo các đối tượng
3. Thiết kế • thiết kế kiến trúc (tạo các mô-đun) • thiết kế chi tiết	3. (*)Thiết kế hướng đối tượng • thiết kế chi tiết
4. Cài đặt • cài đặt trên ngôn ngữ lập trình thích hợp	4. (*)Lập trình hướng đối tượng • cài đặt trên ngôn ngữ lập trình hướng đối tượng thích hợp

Hình 1.6 Sự khác nhau giữa hướng cấu trúc và hướng đối tượng

1.7 Thuật ngữ
(terminology)

- Phần mềm (software)
  - mã lệnh dưới dạng máy có thể đọc được
  - các dạng tài liệu đặc tả, thiết kế, luật và sổ sách về chi phí
  - kế hoạch quản lý dự án phần mềm và các tài liệu quản lý khác
  - các dạng tài liệu hướng dẫn sử dụng
- Chương trình (program), là một đoạn mã lệnh có thể tự thực thi được
- Hệ thống (system), là tập hợp các chương trình liên quan với nhau
- Sản phẩm (product)
  - một mẩu bình thường của phần mềm
  - kết quả đạt được sau một tiến trình (process) phát triển phần mềm
- Sản xuất phần mềm (software production) bao gồm hai giai đoạn:
  - phát triển phần mềm (software development)
  - bảo trì (maintenance)

- Tập hợp các kỹ thuật (methodology, paradigm)
- Lỗi (error) hay có lỗi (bug)
- Thuộc tính (attribute), là thành phần dữ liệu của một đối tượng. Còn gọi là:
  - trạng thái biến (state variable) trong ngữ cảnh hướng đối tượng
  - thể hiện biến (instance variable,field) trong Java
  - trường (field,member) trong C++
- Phương thức (method). Còn gọi là:
  - hàm thành viên (member function) trong C++
  - trường (field) trong Java
- Khi một phương thức bên trong một đối tượng được kích hoạt, ta gọi là gửi một thông báo (sending a message) đến đối tượng

2 TIẾN TRÌNH PHẦN MỀM

(THE SOFTWARE PROCESS)

Nội dung:

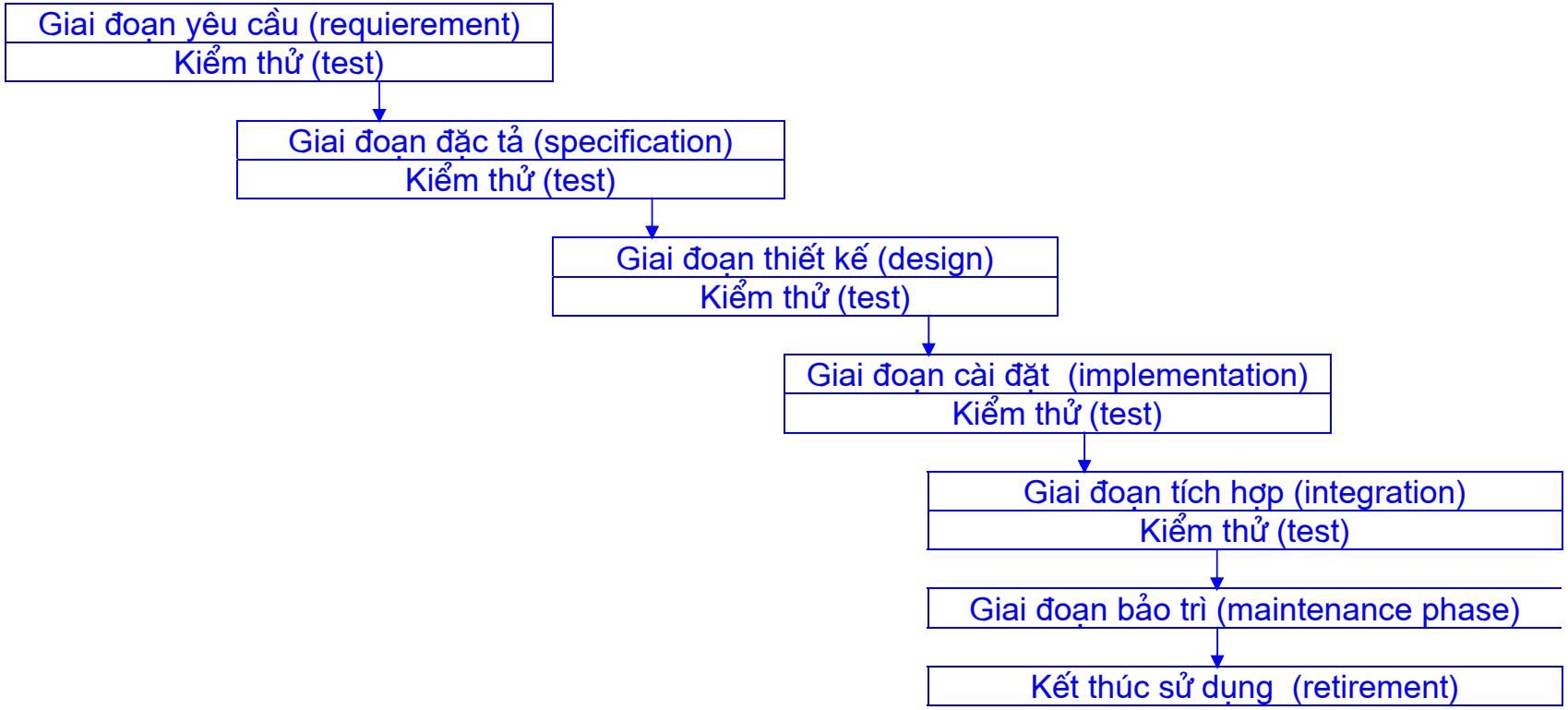
- Khái niệm về khách hàng, nhà phát triển và người sử dụng
- Các giai đoạn trong chu trình sống của phần mềm
- Một số khía cạnh trong sản xuất phần mềm
- Cải tiến tiến trình phần mềm : mô hình khả trưởng
- Các tiêu chuẩn quốc tế

2.1 Khái quát chung
(overview)

- Tiến trình phần mềm là cách thức tạo ra phần mềm
- Mỗi công ty có tiến trình phần mềm riêng
- Khách hàng (client): cá nhân hay công ty đặt hàng sản phẩm
- Nhà phát triển (developer): các thành viên của công ty có trách nhiệm phát triển phần mềm đã được đặt hàng
  - có thể quán xuyến toàn bộ các công việc của sản phẩm
  - có trách nhiệm một phần như thiết kế, cai đặt,...
- Các dạng quan hệ giữa khách hàng và nhà phát triển
  - cùng cơ quan, phần mềm nội bộ (internal software)
  - khác cơ quan, phần mềm hợp đồng (contract software)
- Người sử dụng (user): một hay nhiều cá nhân thay mặt khách hàng để sử dụng sản phẩm

- Phát triển phần mềm (software development): bao gồm tất cả các công việc tạo ra sản phẩm trước khi nó được chuyển sang giai đoạn bảo trì

2.2 Các giai đoạn (the phases)



Hình 2.1 Các giai đoạn trong chu trình sống của phần mềm

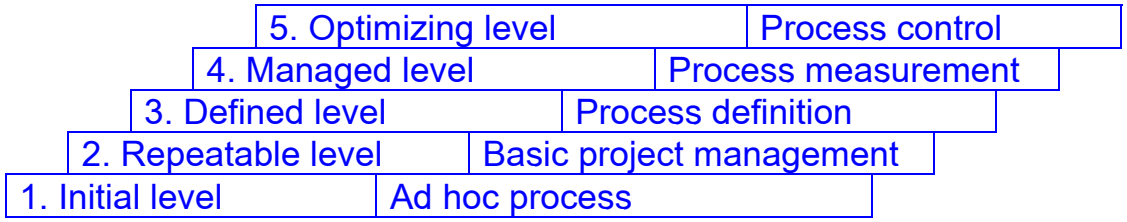
2.3 Một số khía cạnh trong sản xuất phần mềm
(the aspects of software production)

- Độ phức tạp (complexity)
  - là một thuộc tính của phần mềm
  - tác động trên tiến trình phần mềm và cả công tác quản lý tiến trình
  - có thể biểu diễn bằng toán học và vật lý
  - ảnh hưởng đến công tác bảo trì
- Sự thích ứng (conformity)
  - phần mềm phải thích ứng được với các thiết bị sẵn có (không phải các thiết bị đáp ứng phần mềm)
  - thích ứng tốt với phần cứng phục vụ phần mềm
- Dễ chuyển đổi (changeability)
  - phần mềm phải thay đổi theo thực tiễn
  - mở rộng các chức năng ban đầu
  - thay đổi phần mềm dễ hơn thay đổi về phần cứng
  - phần cứng thay đổi theo sự phát triển của phần mềm hoặc công nghệ

- Tính vô hình (invisibility)
  - giấu các công đoạn phức tạp khi thực hiện phần mềm
  - dễ dàng thuyết minh, thuyết phục khách hàng
  - dễ dàng giao tiếp giữa các bộ phận thực hiện phần mềm
  - sử dụng các phương pháp, công cụ trực quan sinh động
- Nhanh chóng tạo phần mềm mới từ các bộ phận hay công cụ có sẵn (silver bullet)
  - nhanh chóng chuyển đổi chức năng của sản phẩm
  - giảm thời gian cũng như chi phí thực hiện phần mềm
  - sử dụng mô hình chuyển đổi nhanh
- Hình thành thuật ngữ : lỗi trên 1000 dòng lệnh tương đương assembler (faults per million equivalent assembler source - MEASL)

2.4 Cải tiến tiến trình phần mềm: mô hình khả trưởng
(improving the software process: capability maturity models)

- CMMs là nhóm các chiến lược liên quan với nhau nhằm cải tiến tiến trình phần mềm. Được đề xuất tại Viện công nghệ phần mềm (software engineering institute - SEI) [Humphrey, 1989]
  - cho phần mềm SW-CMM (software)
  - cho quản lý nguồn nhân lực P-CMM (people)
  - cho công nghệ hệ thống SE-CMM (system engineering)
  - cho phát triển sản phẩm tích hợp IPD-CMM (integrated product development)
  - cho đạt được sản phẩm SA-CMM (software aquisition)



Hình 2.2 Năm mức của CMM

2.5 Các tiêu chuẩn quốc tế (international standards)

- CMMs là tiêu chuẩn khởi điểm cho các tiêu chuẩn về sau
- ISO 9000
  - International Standards Organization (ISO) 9000-series standards
  - gồm 5 chuẩn áp dụng rộng rãi cho các hoạt động công nghiệp: thiết kế (design), phát triển(development), sản xuất(production), cài đặt (installation) và bảo dưỡng (servicing)
  - ISO 9001 dành cho chất lượng sản phẩm [ISO 9001, 1987]
  - áp dụng ISO 9001 cho phần mềm : ISO 9000-3 [ISO 9000-3, 1991]
  - trên 60 nước chấp thuận: Mỹ, Nhật, Canađa, EU, Việt Nam,...
- SPICE
  - Software Process Improvement Capability dEtermination
  - do Bộ quốc phòng Anh đề xuất vào năm 1995
  - tương tự như SW-CMM và ISO 9000
  - thống nhất 2 chuẩn từ 06/1997: ISO/IEC 15504 hay 15504
  - trên 40 nước chấp thuận

2.6 Giá thành và lợi ích của việc cải tiến tiến trình phần mềm
(costs and benefits of 'software process improvement-SPI')

- Công ty Software Engineering Division of Hughes Aircaft ở Fullerton, California đã trả $500000 để thực hiện chuyển đổi từ mức 2 sang mức 3-4-5 và tiết kiệm được hàng năm $2000000 !

Thể loại	Khoảng giới hạn	Trung bình	Số điểm dữ liệu (data points)
Số năm tham gia SPI	1-9	3.5	24
Trị giá hàng năm của SPI cho mỗi SE	$490	$2004	5
Hiệu suất tăng hàng năm	9%-67%	35%	4
Tỷ lệ lỗi phát hiện sớm hàng năm	6%-25%	22%	3
Thời gian rút ngắn hàng năm để tham gia thị trường	15%-23%	19%	2
Thời gian rút ngắn hàng năm việc gửi trễ hạn các báo cáo lỗi	10%-94%	39%	5
Giá trị kinh doanh (tiết kiệm/chi phí của SPI)	4.0-8.8:1	5.0:1	5

Hình 2.3 Dữ liệu cải tiến phần mềm SW-CMM [Herbsleb và al., 1994]

Mức CMM	Số lượng dự án	Thời gian giảm tương đối	Lỗi trên MEASL phát hiện được trong phát triển	Hiệu suất tương đối
1	3	1.0	-	-
2	9	3.2	890	1.0
3	5	2.7	411	0.8
4	8	5.0	205	2.3
5	9	7.8	126	2.8

Hình 2.4 Kết quả của 34 dự án GED của Motorola (Government Electronics Division-GED)

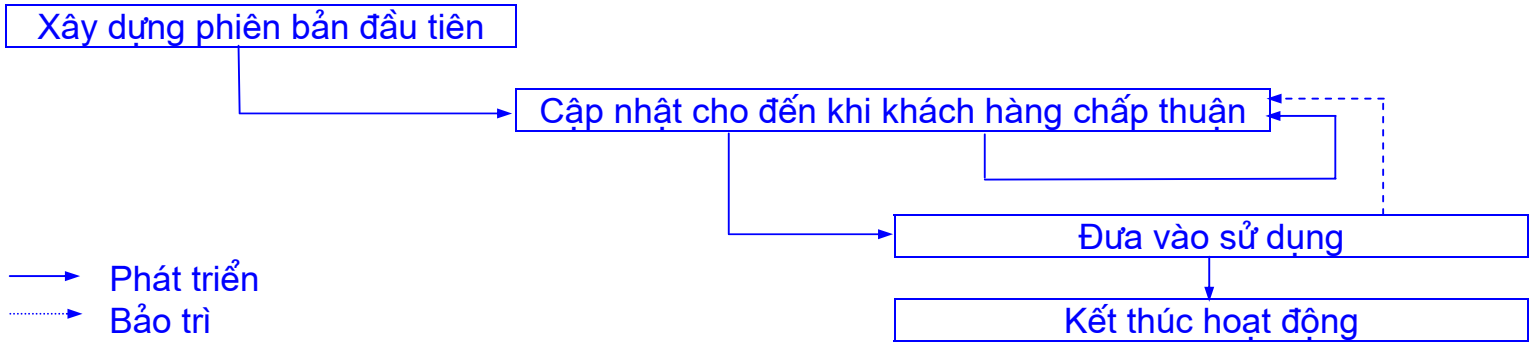
3 CÁC MÔ HÌNH CHU TRÌNH SỐNG CỦA PHẦN MỀM (SOFTWARE LIFE-CYCLE MODELS)

Nội dung:

- Mô hình xây dựng và hiệu chỉnh
- Mô hình thác nước
- Mô hình định khung nhanh
- Mô hình tăng trưởng
- Mô hình đồng bộ và ổn định
- Mô hình xoắn ốc
- Các mô hình hướng đối tượng
- So sánh các mô hình

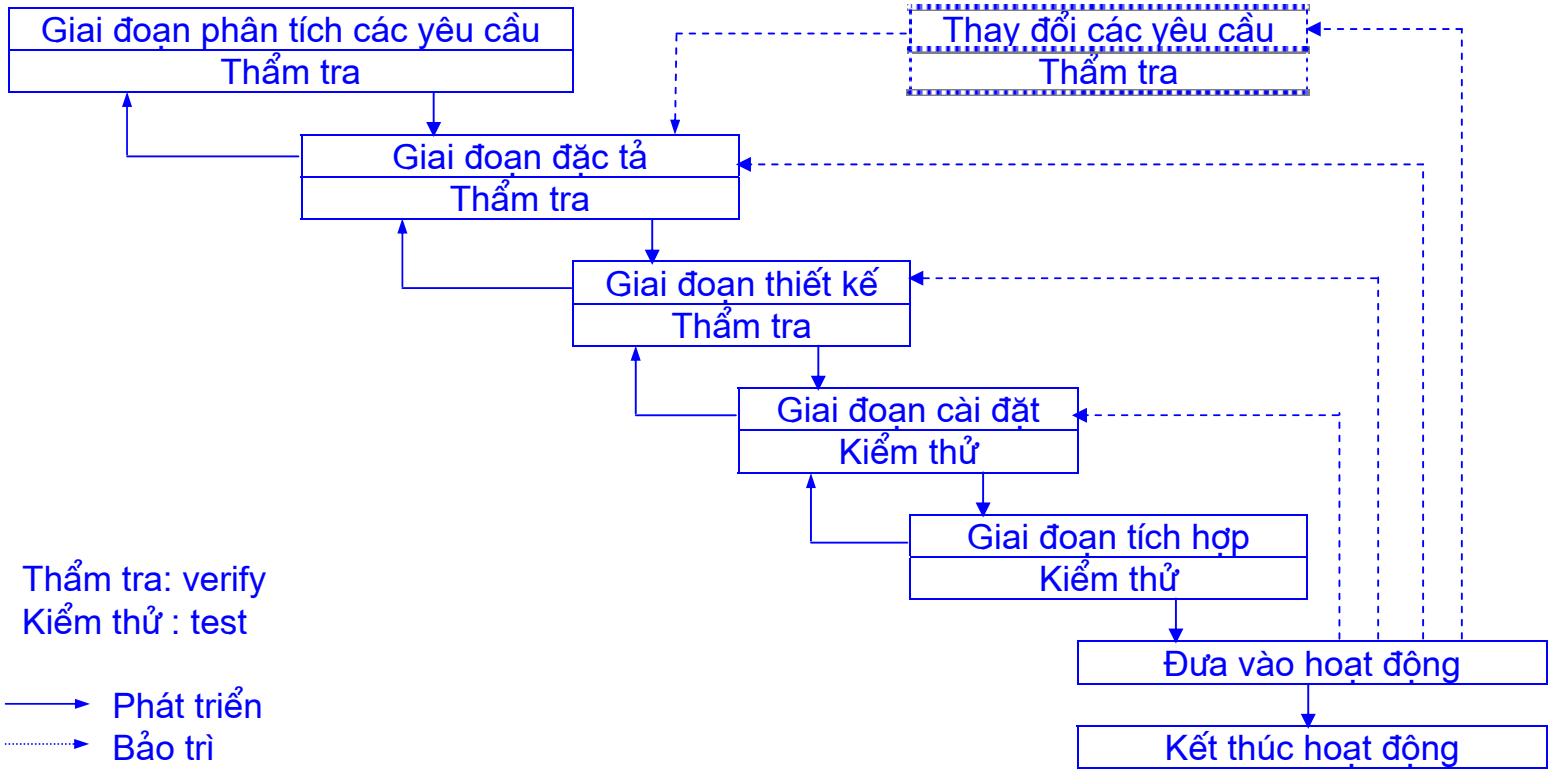
3.1 Mô hình xây dựng và hiệu chỉnh (build-and-fix model)

- Không có đặc tả hay thiết kế
- Chỉ đơn giản là làm đi làm lại cho đến khi nào đáp ứng được yêu cầu của khách hàng
- Thường sử dụng trong các bài tập lập trình từ 100 đến 200 dòng mã lệnh



Hình 3.1 Mô hình xây dựng và hiệu chỉnh

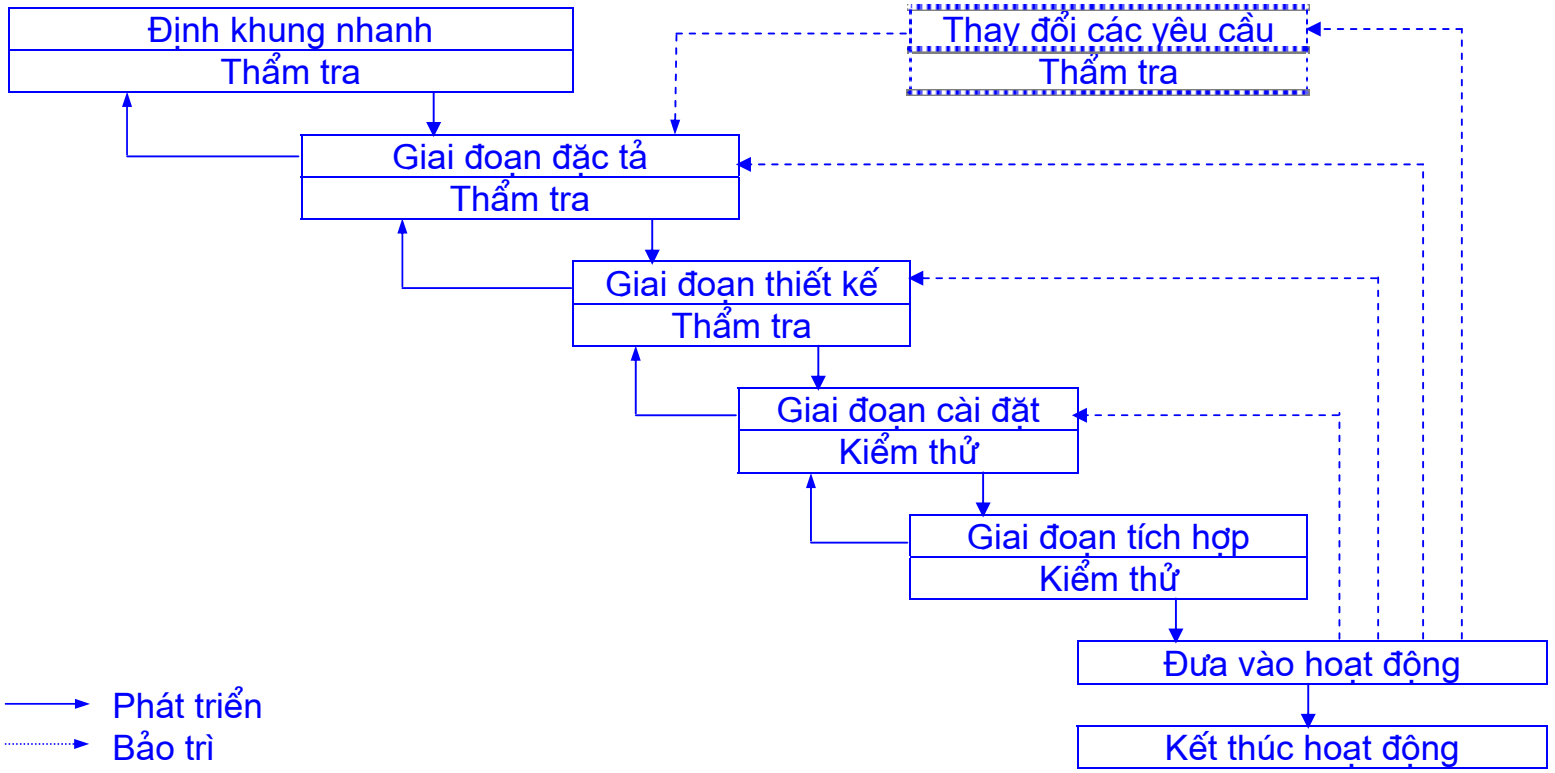
3.2 Mô hình thác nước
(waterfall model)



Hình 3.2 Mô hình thác nước

- Do Royce đề xuất [Royce, 1970]
- Các lỗi ở một số giai đoạn trước được phản hồi bởi các giai đoạn sau
- Mỗi giai đoạn chỉ được xem là hoàn thành sau khi đã có đầy đủ tài liệu cho giai đoạn đó và được nhóm SQA chấp thuận
- Các bước tiến hành chính:
  - các yêu cầu được xác định và kiểm chứng bởi khách hàng và nhóm SQA
  - các đặc tả được kiểm chứng bởi nhóm SQA và gửi cho khách hàng
  - lập SPMP và bảng thời gian làm việc chi tiết
  - giai đoạn thiết kế bắt đầu sau khi khách hàng đồng ý về giá thành và thời gian thực hiện; thực hiện cài đặt và tích hợp
  - khách hàng cho hoạt động thử; chấp nhận sản phẩm
  - chuyển sang giai đoạn bảo trì
- Ưu điểm: kỷ luật cao; quy định tốt về tài liệu cho mỗi giai đoạn; kiểm chứng cẩn thận bởi nhóm SQA; được ứng dụng rộng rãi
- Khuyết điểm:
  - quá nhiều kiểm thử, thẩm tra và tài liệu
  - hướng tài liệu: khó hình dung và khó hiểu đối với khách hàng

3.3 Mô hình định khung nhanh
(rapid prototyping model)



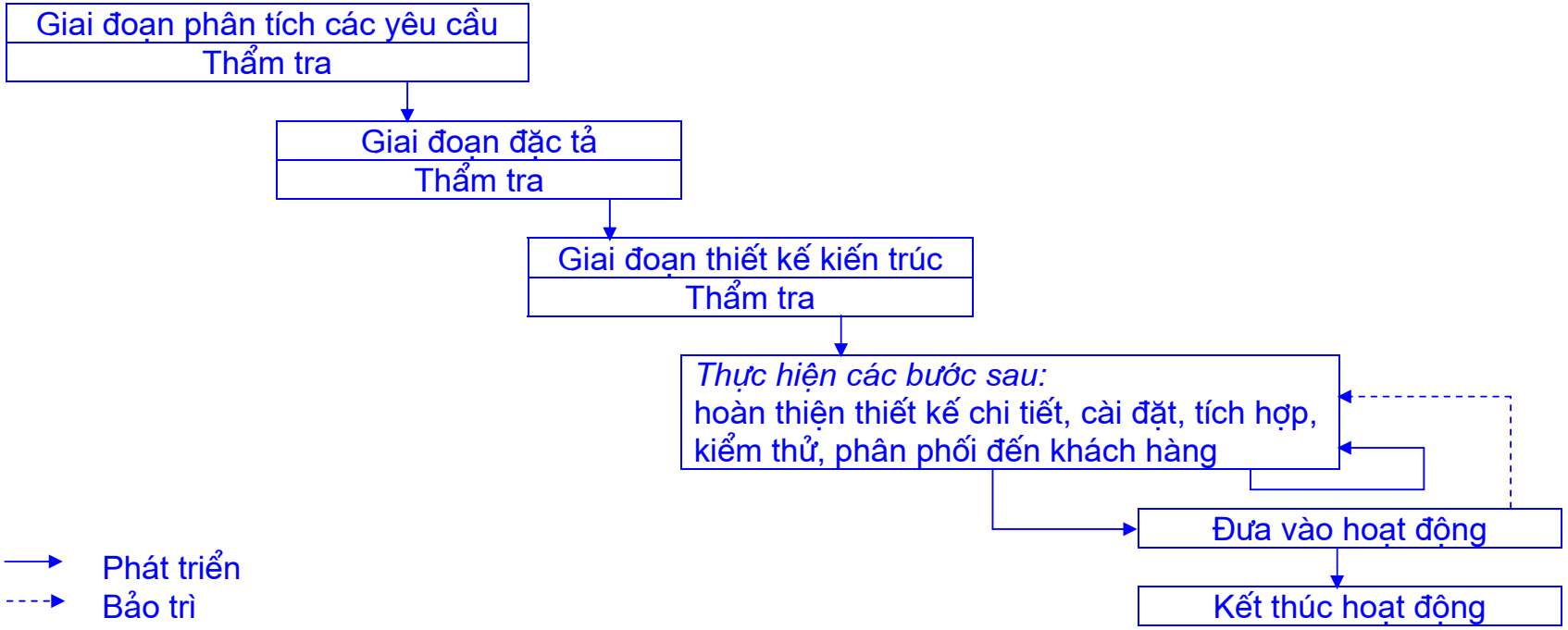
Hình 3.3 Mô hình định khung nhanh

- Là mô hình hoạt động có chức năng tương đương với một tập hợp con (subset) của sản phẩm
  VD: Nếu chức năng sản phẩm đích là trả tiền tài khoản, nhận tiền từ tài khoản và xếp hàng vào kho thì việc định khung nhanh có thể bao gồm các công việc của sản phẩm như: màn hình nhập liệu, in các báo cáo nhưng không có các công việc như cập nhật tập tin hay bắt các lỗi xuất hiện.
- Các bước thực hiện chính:
  - bước đầu tiên là định khung nhanh mô hình,tạo điều kiện cho khách hàng và người sử dụng tương lai tương tác với mô hình và thử nghiệm
  - chuyển sang giai đoạn đặc tả sau khi khách hàng đã chấp thuận rằng các yêu cầu cần thiết đã có trong quá trình định khung nhanh
- Yêu cầu của mô hình là thực hiện càng nhanh càng tốt để tăng tốc độ của tiến trình phát triển phần mềm

❖**Tích hợp hai mô hình thác nước và định khung nhanh**

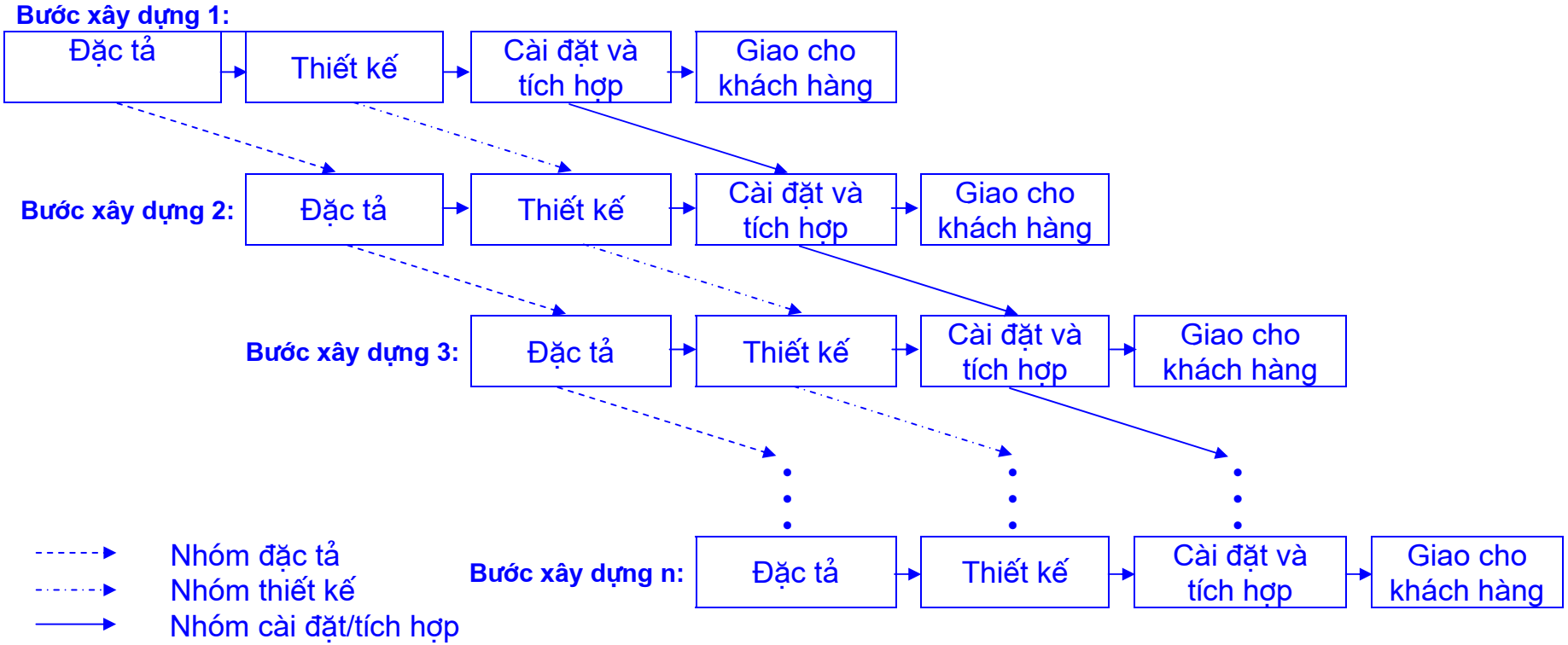
- xem việc định khung nhanh là đầu vào của mô hình thác nước
- có thể xảy ra một số hiệu ứng lề và có thể có rủi ro (risk) xuất hiện do sử dụng nhiều mô hình (số lượng ở đây là 2)

3.4 Mô hình tăng trưởng (incremental model)



Hình 3.4 Mô hình tăng trưởng

- Chuỗi các bước thiết kế,cài đặt,tích hợp và kiểm thử được thực hiện liên tục (tăng). Sử dụng trong một số dự án về phòng thủ không gian [Wong, 1984]



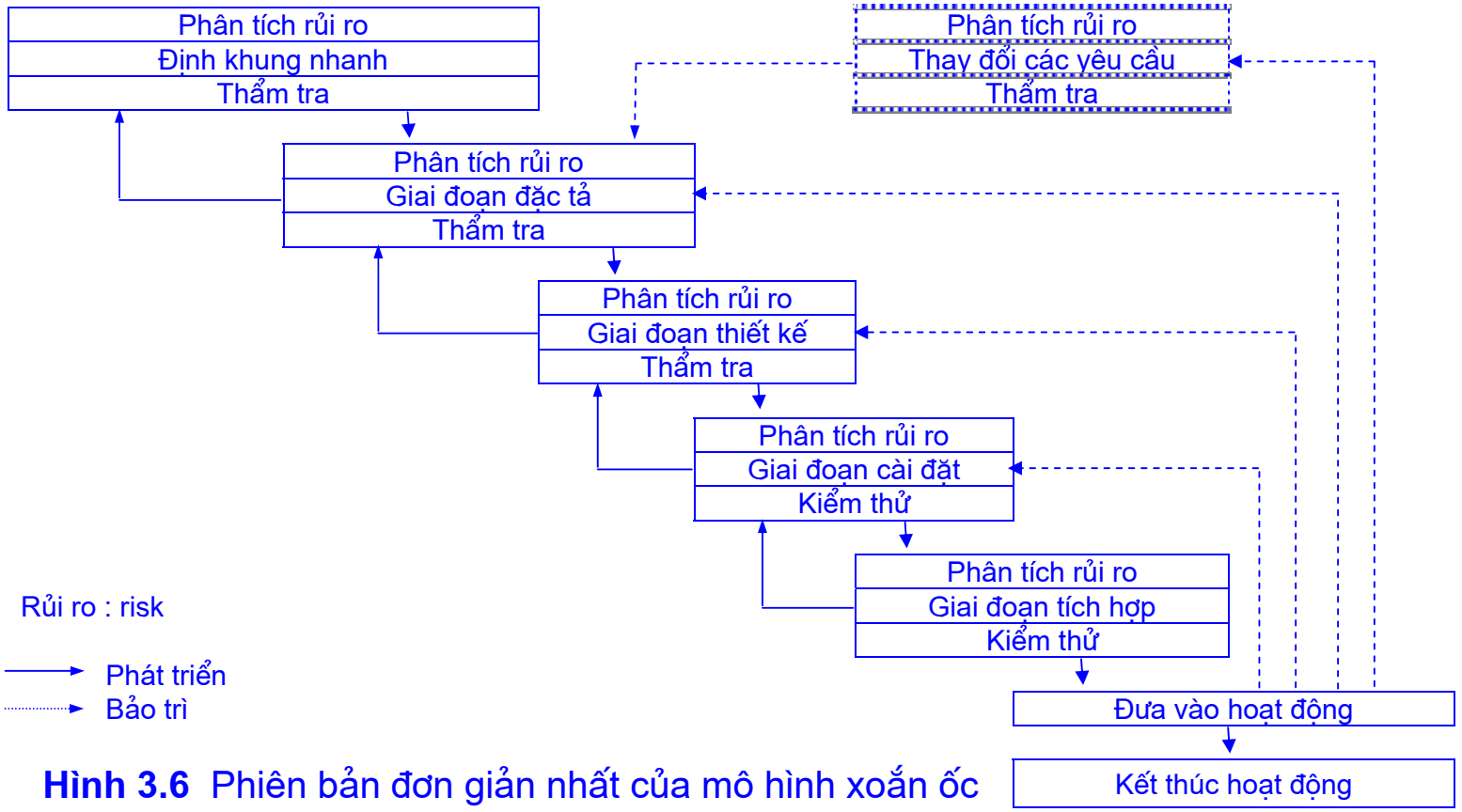
Hình 3.5 Mô hình tăng trưởng nhiều rủi ro

- Giao sản phẩm cho khách hàng sau mỗi bước xây dựng. Mỗi bước xây dựng tương đương với một tập con các yêu cầu của khách hàng.
- Một sản phẩm điển hình thường bao gồm khoảng 10-50 bước xây dựng
- Giảm *khó chịu* cho khách hàng khi phải thay đổi một sản phẩm hoàn chỉnh
- Chú ý việc phá bỏ các cấu trúc của bước xây dựng trước đó !

3.5 Mô hình đồng bộ và ổn định (synchronize-and-stabilize model)

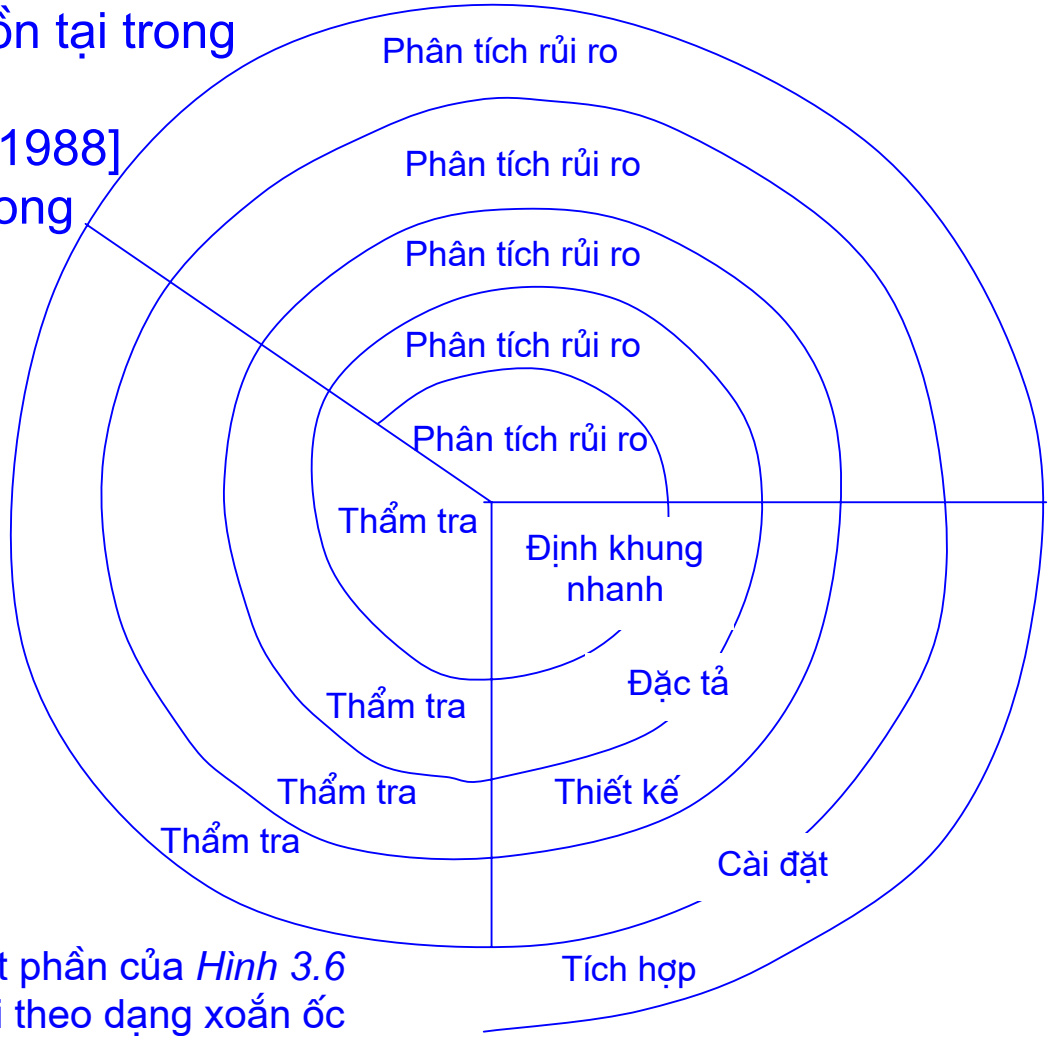
- Là một dạng khác của mô hình tăng trưởng [Cusamano và Selby, 1997]
- Được triển khai sử dụng tại công ty Microsoft, Inc.
- Các bước thực hiện:
  - dẫn dắt các phân tích yêu cầu bằng cách phỏng vấn đông đảo các khách hàng tiềm năng (potential customers)
  - rút ra tài liệu đặc tả
  - công việc được chia thành 3 hay 4 bước xây dựng: đặc điểm cấp thiết nhất, đặc điểm cấp thiết nhì,....
  - mỗi bước xây dựng được thực hiện cùng lúc bởi nhiều nhóm nhỏ
  - cuối mỗi ngày các nhóm thực hiện đồng bộ với nhau bằng cách ghép các phần việc của nhóm lại với nhau, kiểm thử và lần vết trên sản phẩm kết quả
  - hiệu chỉnh các lỗi và bước xây dựng chuyển sang trạng thái ổn định, sẽ không có bất kỳ thay đổi nào nữa trên tài liệu đặc tả
  - lặp lại bước đồng bộ

3.6 Mô hình xoắn ốc
(spiral model)

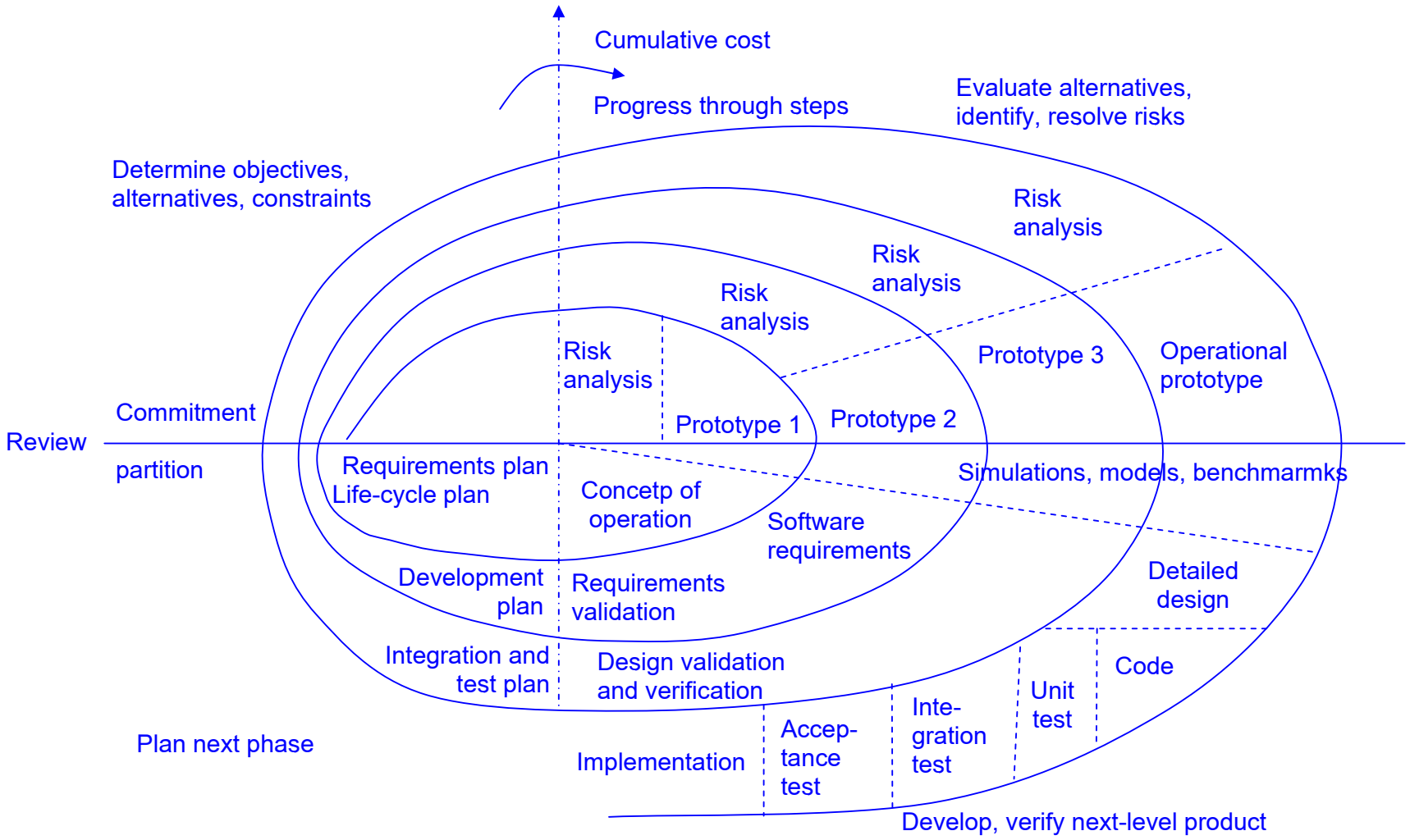


Hình 3.6 Phiên bản đơn giản nhất của mô hình xoắn ốc

- Yếu tố rủi ro hầu như luôn tồn tại trong sự phát triển của phần mềm
- Do Boehm đề xuất [Boehm,1988] nhằm giảm thiểu sự rủi ro trong quá trình phát triển
- Được sử dụng rộng rãi cho một lớp rộng các sản phẩm và gặt hái nhiều thành công [Boehm, 1988]



Hình 3.7 Một phần của *Hình 3.6* được vẽ lại theo dạng xoắn ốc



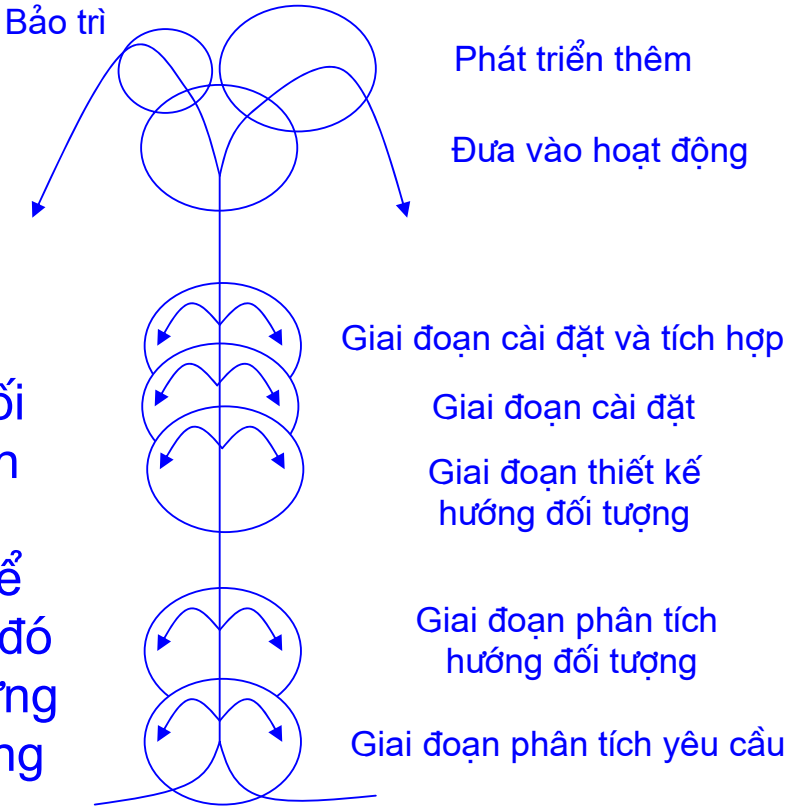
Hình 3.8 Mô hình xoắn ốc đầy đủ [Boehm, 1988]. (©1988 IEEE.)

- *Điểm mạnh*
  - hướng rủi ro (risk-driven)
  - các công việc luân phiên và chịu các ràng buộc đã hỗ trợ cho việc tái sử dụng phần mềm hiện có
  - đánh giá mức độ rủi ro
  - mục tiêu quan trọng luôn là chất lượng phần mềm
  - giảm nhẹ kiểm thử và nhanh chóng sửa chữa những lỗi xảy ra
  - bảo trì đơn giản chỉ là một vòng tròn trong xoắn ốc, như vậy không có sự phân biệt giữa phát triển và bảo trì
- *Điểm yếu*
  - hướng rủi ro
    VD: tiến hành thế nào nếu có một thành phần có độ rủi ro cao ?
  - dành riêng cho các phần mềm nội bộ có kích thước lớn [Boehm, 1988]
  - vì dự án có thể chấm dứt do các đánh giá về rủi ro, do đó sẽ rất không hay khi đã ký kết các hợp đồng, ảnh hưởng đên uy tín của công ty, rắc rối về mặt luật pháp
  - kích thước sản phẩm ảnh hưởng đến giá thành việc phân tích rủi ro

3.7 Các mô hình hướng đối tượng
(object-oriented life-cycle models)

- Đặc tính quan trọng nhất là lặp:
  - giữa các giai đoạn
  - một phần trong giai đoạn

- Mô hình vòi phun nước của [Hendreson-Sellers và Edwards, 1990]
  - vòng tròn thể hiện các giai đoạn gối lên nhau, phần thấy được phản ánh sự gối lên trên giữa các hoạt động
  - mũi tên bên trong một giai đoạn thể hiện sự lặp lại bên trong giai đoạn đó
  - vòng tròn bảo trì nhỏ hơn tượng trưng cho việc giảm bớt nhân lực cho công tác bảo trì



Hình 3.9 Mô hình vòi phun nước

- Một số mô hình khác
  - Objectory [Jacobson, Christerson và Overgaard, 1992]
  - chu trình sống đệ quy/song song (recursice/parallel)[Berard, 1993]
  - thiết kế cấu trúc hình thức khứ hồi (round-trip gestalt) [Booch, 1994]

Điểm mạnh:
- cho phép lặp
- kết hợp nhiều dạng song song (các hoạt động gối đầu)
- hỗ trợ phát triển tăng trưởng

Điểm yếu:
- nguy cơ có thể xảy ra do thông dịch không đúng những cái cần thiết
- thiếu kỷ luật trong công việc,trình tự công việc của các thành viên chuyển dịch hầu như ngẫu nhiên giữa các giai đoạn
  VD: đầu tiên là thiết kế phần một, tiếp theo là phân tích phần hai, sau đó là cài đặt phần ba,... !
- trình tự cái mới trong sự liên hệ giữa các thành phần, do trình tự làm việc ngẫu nhiên dẫn đến mới ở chỗ này nhưng lại cũ tại nơi khác !

3.8 So sánh các mô hìnhchu trình sống
(comparaison of life-cycle models)

Mô hình chu trình sống	Điểm mạnh	Điểm yếu
Mô hình xây dựng và hiệu chỉnh	Tốt đối với các chương trình ngắn không yêu cầu về bảo trì	Không đáp ứng được các chương trình tương đối lớn trở đi
Mô hình thác nước	Tiếp cận có kỷ luật Hướng tài liệu	Sản phẩm chuyển giao có thể không theo những gì khách hàng cần
Mô hình định khung nhanh	Đảm bảo sản phẩm được chuyển giao có được những gì khách hàng cần	*Xem phần 9*
Mô hình tăng trưởng	Trở lại sớm tối đa bằng cách đầu tư tiếp tục Đẩy mạnh công tác bảo trì	Đòi hỏi kiến trúc mở Có thể thoái hóa thành mô hình xây dựng và điều chỉnh
Mô hình đồng bộ và ổn định	Có được những gì khách hàng cần trong tương lai Đảm bảo các thành phần có thể tích hợp thành công	Không được sử dụng rộng rãi như tại Microsoft
Mô hình xoắn ốc	Kết hợp nhiều đặc điểm của tất cả các mô hình phía trên	Chỉ có thể sử dụng cho các sản phẩm có kích thước lớn hay cho các tổ chức Các nhà phát triển phải có khả năng phân tích rủi ro và giải quyết rủi ro
Các mô hình hướng đối tượng	Hỗ trợ việc lặp lại bên trong các giai đoạn, song song hóa giữa các giai đoạn	Có thể suy thoái thành CABTAB (thuật ngữ về sự thiếu kỷ luật trong công việc: trình tự thực hiện các công việc lung tung, bừa bãi)

Hình 3.10 So sánh giữa các mô hình chu trình sống

4

NHÓM LÀM VIỆC VÀ CÁC CÔNG CỤ NGHỀ NGHIỆP

(TEAMS AND THE TOOLS OF THEIR TRADE)

Nội dung:

- Khái quát chung
- Tiếp cận về các nhóm làm việc
- Phân tích giá thành và lợi nhuận
- Đánh giá phần mềm
- Các công cụ CASE
- Các phiên bản phần mềm

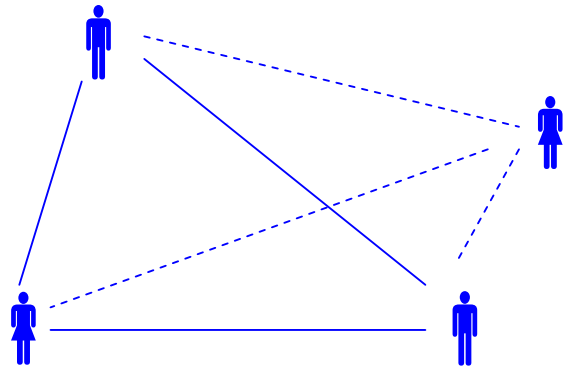
4.1 Khái quát chung
(overview)

- Dự án phần mềm chỉ có thể thành công với:
  - thành thạo, hiểu biết về công nghệ phần mềm
  - đào tạo tốt về công nghệ phần mềm
- Ngoài con người tốt, các nhóm làm việc cũng phải được tổ chức nhằm làm cho các thành viên làm việc hiệu quả và kết hợp chặt chẽ với nhau
- Công nghệ phần mềm cần hai dạng công cụ:
  - phân tích, dùng trong phát triển phần mềm. VD: công cụ phân tích giá thành, lợi nhuận; công cụ phân tích mịn dần
  - phần mềm, các sản phẩm trợ giúp các nhóm công nghệ phần mềm trong phát triển và bảo trì phần mềm. Thường gọi là các **công cụ CASE** (computer-adied software engineering tools - *CASE tools*)

4.2 Tổ chức nhóm làm việc
(team organization)

- Các sản phẩm tương đối lớn trở đi phải do những người chuyên nghiệp thực hiện và những người này được tổ chức thành nhóm làm việc (team)
- *Vấn đề đặt ra*: sản phẩm nếu do 1 người thực hiện sẽ hoàn thành trong 1 năm, 4 người thực hiện sẽ hoàn thành trong 3 tháng ?

- **Luật Brooks [Brooks, 1975]:** thêm nhân lực cho một dự án phần mềm đang thực hiện sẽ làm chậm tiến độ thực hiện của nó.
- Các giai đoạn đều có nhóm làm việc riêng nhưng vai trò đặc biệt thuộc về nhóm cài đặt (mỗi người làm việc trên một mô-đun riêng)

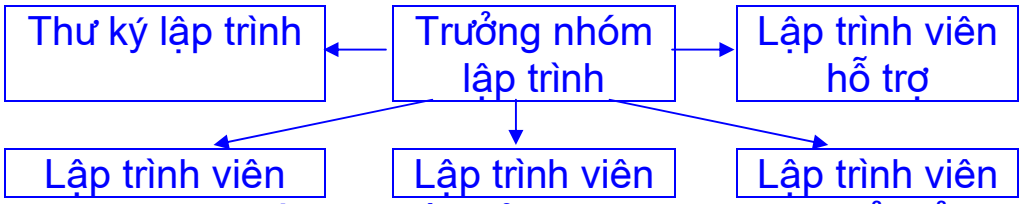


Hình 4.1 Các kênh giao tiếp khi thêm một người mới (nét đứt)

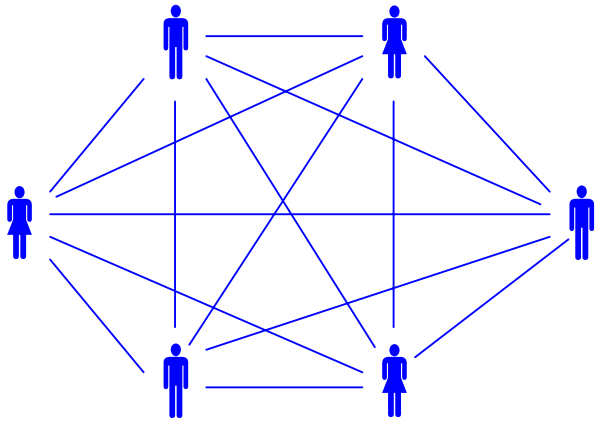
4.3 Tiếp cận nhóm làm việc dân chủ
(democratic team approach)

- Được mô tả đầu tiên bởi Weinberg [Weinberg, 1971]
- Khái niệm cơ bản là *lập trình bản ngã* (egoless programming)
  - lập trình viên gắn bó cao với mã lệnh của họ
  - các mô-đun như là sự mở rộng của chính bản thân
  - khó phát hiện lỗi
- Hướng giải quyết:
  - cấu trúc lại môi trường xã hội theo các giá trị của lập trình viên
  - khuyến khích các thành viên khác trong nhóm tìm kiếm lỗi trong các mã lệnh của mình→ thể hiện tinh thần tập thể cao
- *Nhóm làm việc dân chủ* (democratic team): ≤ 10 lập trình viên bản ngã
- Ưu điểm: thái độ tích cực để phát hiện lỗi, cảm thấy hạnh phúc trong nhóm
- Khuyết điểm: khó chấp nhận từ phía các nhà quản lý, các lập trình viên lâu năm sẽ cảm thấy khó chịu (nhất là khi được các lập trình viên trẻ tuổi giúp phát hiện lỗi !)

4.4 Tiếp cận về trưởng nhóm lập trình cổ điển
(classical chief programmer team approach)



Hình 4.3 Cấu trúc về trưởng nhóm lập trình cổ điển

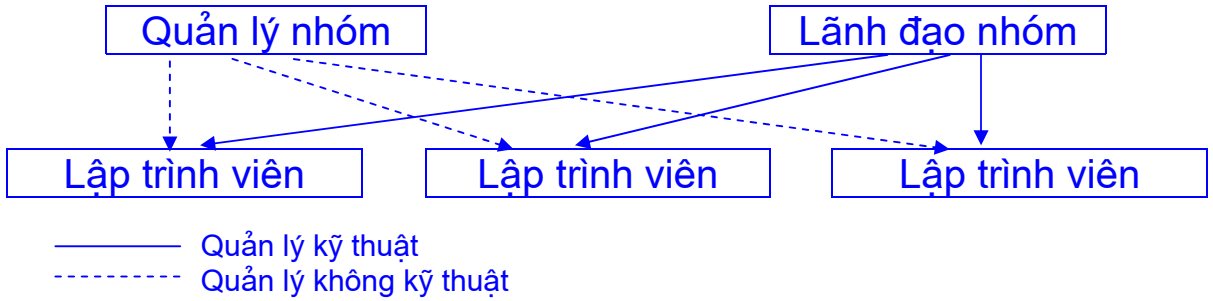


Hình 4.2 Các kênh giao tiếp

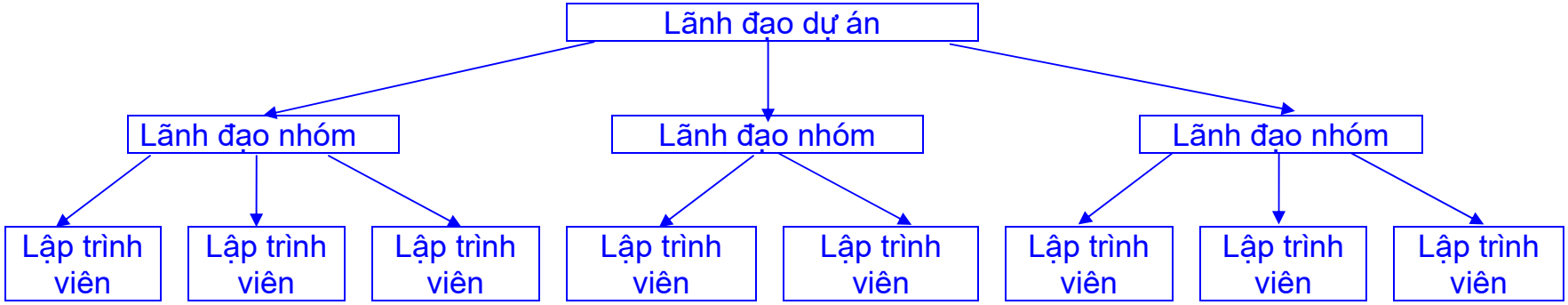
- Chính thức hóa bởi Mills [Backer, 1972]
- Các thành viên trong nhóm:
  - trưởng nhóm (chief), quản lý tốt, giỏi lập trình, xử lý các công việc khó khăn khác
  - lập trình viên hỗ trợ (back-up programmer), sẵn sàng thay thế trưởng nhóm quán xuyến các công việc khi cần
  - thư ký lập trình (secretary), bảo trì thư viện, tài liệu, danh sách mã nguồn, dữ liệu kiểm thử, JCL (job control language)
  - lập trình viên (programmer)
- Trợ giúp của các chuyên gia luật, tài chính,...trong các vấn đề liên quan

4.5 Một số cấu trúc nhóm lập trình hiện đại
(structures of modern programming team)

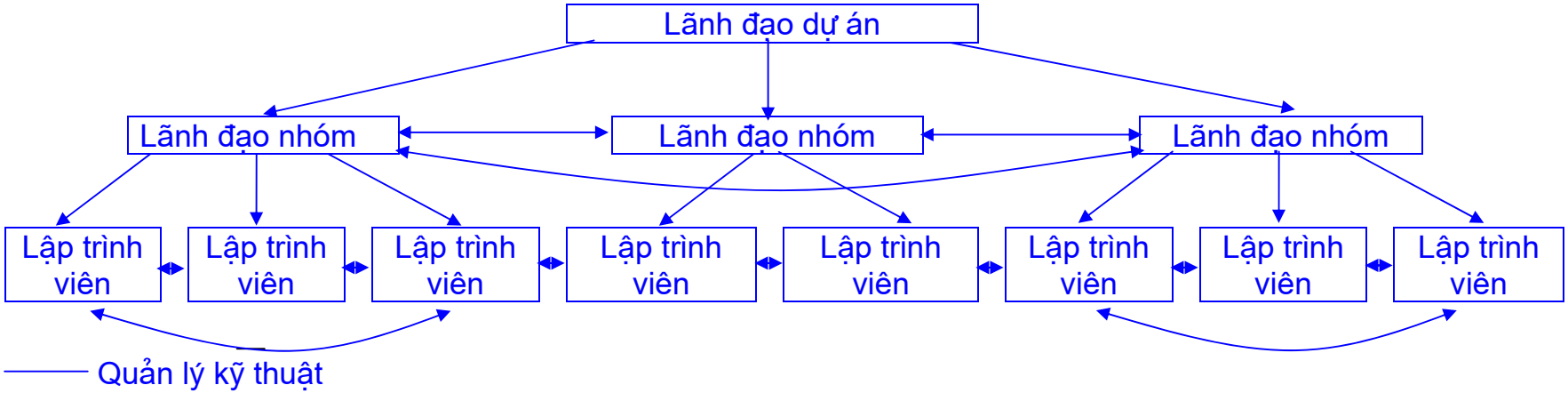
- Khó tìm được trưởng nhóm có khả năng tuyệt vời như cấu trúc cổ điển
- Tách trưởng nhóm thành 2 cá nhân
  - lãnh đạo (leader)
  - quản lý (manager)



Hình 4.4 Cấu trúc nhóm lập trình hiện đại



Hình 4.5 Cấu trúc tổ chức quản lý kỹ thuật cho các dự án lớn



Hình 4.6 Phiên bản của *Hình 4.5* với việc phi tập trung hóa các quyết định và các kênh giao tiếp kỹ thuật

4.6 Các nhóm làm việc dạng đồng bộ hoá và ổn định (synchronize-and-stabilize teams)

- Do Microsoft sử dụng [Cusamano và Selby, 1997]
  VD: Windows95 có: ≥ 11 triệu LOCs, ≥ 200 lập trình và kiểm thử viên
- Cứ mỗi 3-4 bước xây dựng được thực hiện song song bởi nhiều nhóm nhỏ
- Tổ chức của một nhóm nhỏ:
  - 1 quản lý chương trình
  - 3-8 nhà phát triển
  - 3-8 kiểm thử viên (tương ứng 1-1 với số lượng nhà phát triển)
- Công việc của một nhóm nhỏ:
  - được cung cấp tài liệu đặc tả cho toàn bộ công việc của nhóm
  - môi thành viên trong nhóm được tự do thiết kế và cài đặt phần làm việc của mình
- Ưu điểm: các lập trình viên luôn sáng tạo và đổi mới, hướng cùng mục đích
- Khuyết điểm: phải tôn trọng triệt để thời gian đưa mã nguồn vào cơ sở dữ liệu của sản phẩm để đồng bộ hóa

4.7 So sánh các tiếp cận tổ chức nhóm làm việc
(comparison of approaches to team organization)

Tổ chức nhóm làm việc	Điểm mạnh	Điểm yếu
Các nhóm dân chủ	Chất lượng mã lệnh cao vì thái độ tích cực dò tìm lỗi Đặc biệt tốt với những vấn đề phức tạp	Không thể gánh vác thêm công việc
Trưởng nhóm lập trình cổ điển	Thành công chính trong dự án báo *NewYork Times*	Không thực tế
Trưởng nhóm lập trình có bổ sung	Nhiều thành công	Không có những thành công so sánh được với dự án *NewYork Times*
Nhóm lập trình hiện đại	Quản lý nhóm/lãnh đạo nhóm xoá đi sự cần thiết của trưởng nhóm lập trình Co giãn được Hỗ trợ phi tập trung khi cần	Khó khăn sẽ nảy sinh nếu không có sự phác họa rõ ràng trách nhiệm giữa quản lý và lãnh đạo nhóm

Hình 4.7 So sánh các cách tiếp cận

4.8 Làm mịn dần (stepwise refinement)

- *Định nghĩa:* trì hoãn các quyết định chi tiết càng lâu càng tốt nhằm có thể tập trung vào những vấn đề trọng tâm nhất
- Luật Miller [Miller, 1956], con người chỉ có thể tập trung cùng một lúc tối đa là 2-7 mức thông tin (quanta of information)
- Thuật ngữ *làm mịn dần* được Wirth giới thiệu đầu tiên [Wirth, 1971]
- **Xét ví dụ sau:**

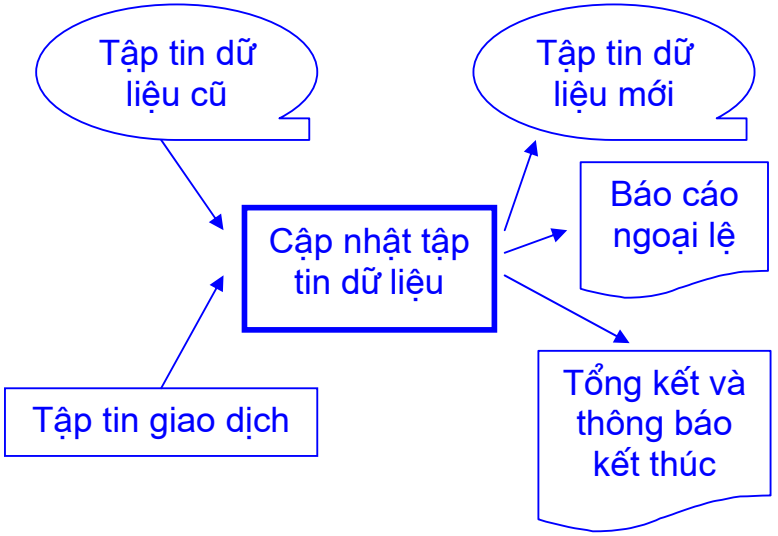
  Bài toán: Thiết kế một sản phẩm dãy các cập nhật tập tin với dữ liệu gồm có tên, địa chỉ khách thuê bao của một tờ báo tháng, *Tuổi trẻ chủ nhật*.
  - có 3 dạng giao dịch: thêm, sửa đổi và xóa với các mã tương ứng 1-2-3.
  - các giao dịch được sắp xếp theo thứ tự từ điển của tên thuê bao; nếu có nhiều giao dịch trên cùng một tên thuê bao, các giao dịch sẽ được thực hiện theo thứ tự sau: thêm, sửa đổi và xóa.

  Hai tập tin đầu vào:
  - chứa dữ liệu về các thuê bao
  - chứa các giao dịch

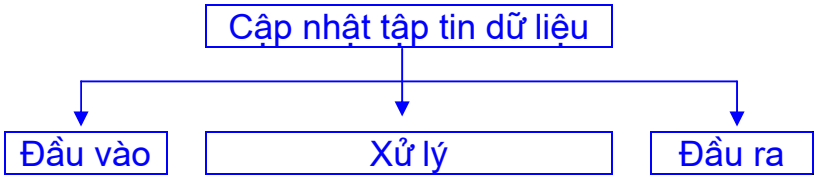
  Ba tập tin đầu ra:
  - chứa dữ liệu mới về các thuê bao
  - báo cáo về các ngoại lệ (exception)
  - tổng kết và thông báo kết thúc công việc

Kiểu giao dịch	Tên	Địa chỉ
3	Bình	
1	Hùng	18/09 đường 30/4, Tp.Cần Thơ
2	Sanh	67 đường Trần Hưng Đạo, TP.HCM
3	Sanh	
1	Toàn	88 đường Đại Cồ Việt, Hànội

Hình 4.8 Các mẩu tin giao dịch



Hình 4.9 Chuỗi các cập nhật tập tin dữ liệu



Hình 4.10 Bước thiết kế mịn thứ nhất

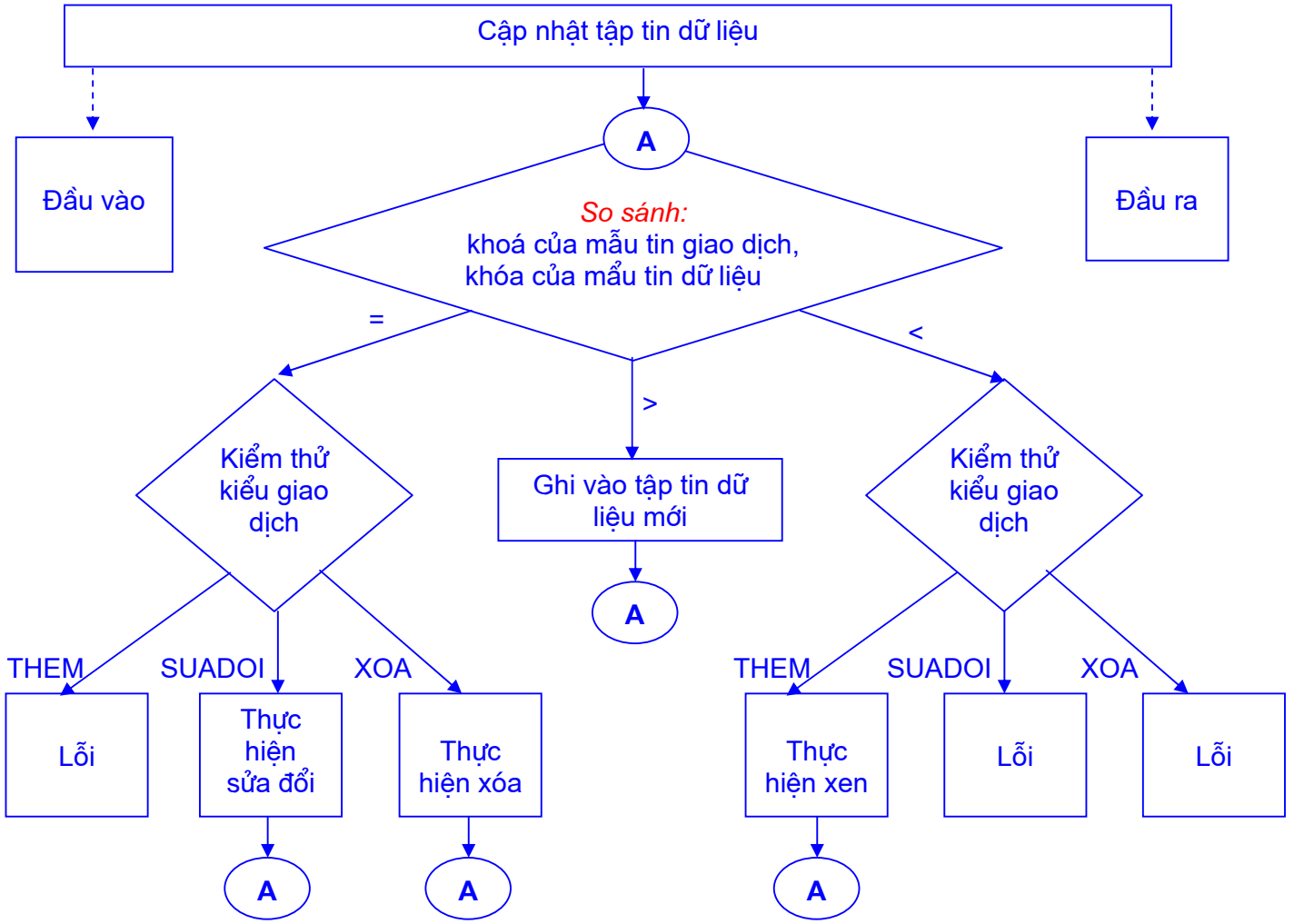
Tập tin giao dịch
3 Bình
1 Hùng
2 Sanh
3 Sanh
1 Việt

Tập tin dữ liệu cũ
Anh
Bình
Khoa
Sanh
Toàn
Tùng

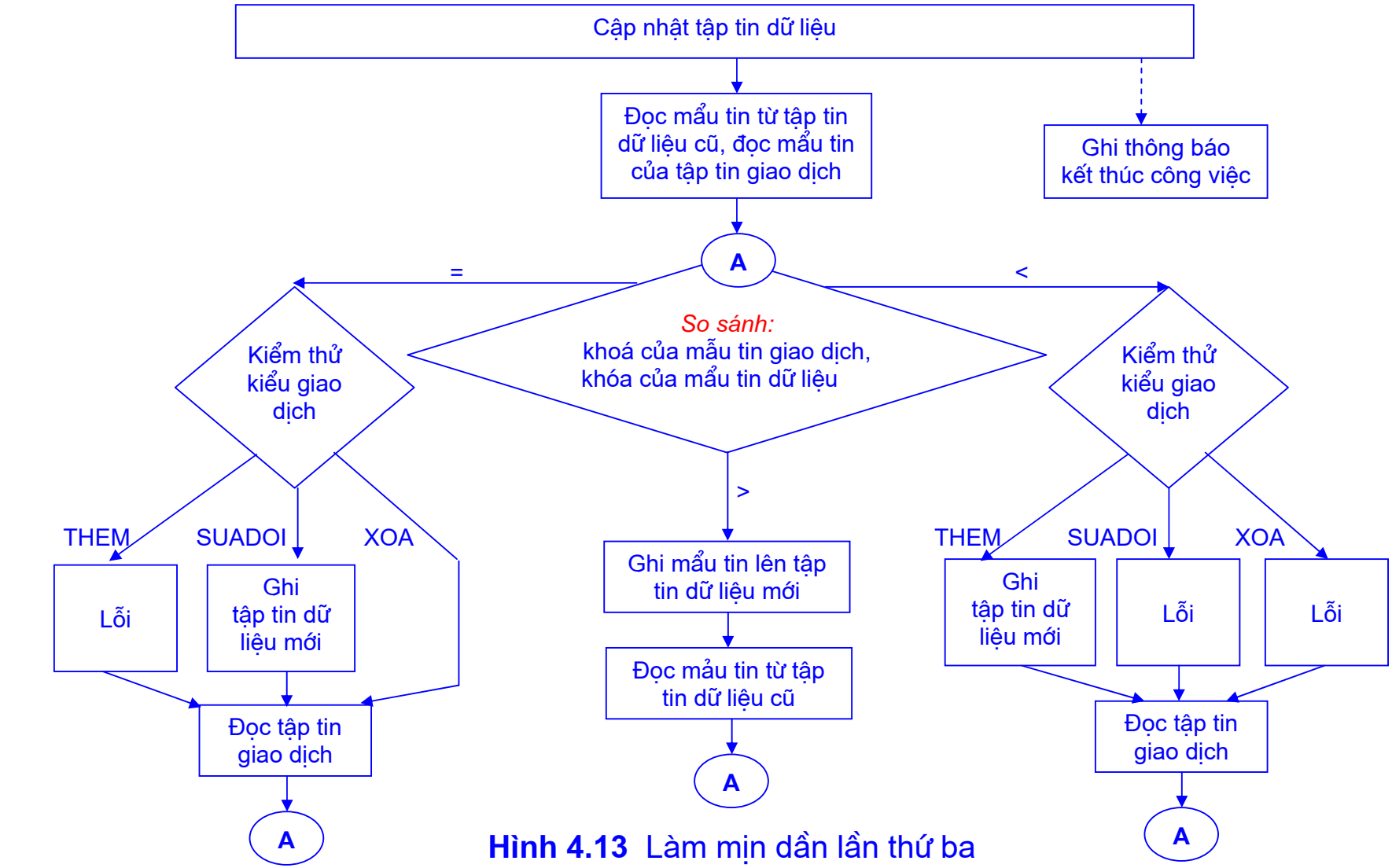
Tập tin dữ liệu mới
Anh
Hùng
Khoa
Toàn
Tùng

Báo cáo ngoại lệ
Toàn

Hình 4.11 Các tập tin giao dịch, dữ liệu cũ, dữ liệu mới và báo cáo ngoại lệ



Hình 4.12 Làm mịn lần thứ hai



Hình 4.13 Làm mịn dần lần thứ ba

4.9 Phân tích giá thành và lợi nhuận (cost-benefit analysis)

- Ước tính giá thành và lợi nhuận trong tương lai
- Một số vấn đề quan tâm:
  - sự thay đổi trị giá của tiền [Yourdon, 1989]
  - giá thành phần cứng, phần mềm
  - tiền thuê chuyên gia công nghệ phần mềm
  - tính giá trị sản phẩm như thế nào ?
  - ...
- Một số hướng giải pháp:
  - tính theo mệnh giá của đồng tiền mạnh
  - sử dụng chiến lược thích hợp để đánh giá
  - ...

4.10 Đánh giá phần mềm
(software metrics)

- Còn gọi là đo (mesurement) phần mềm
- Tiến trình đánh giá (process metrics), quá trình tiến hành đo phần mềm
- Đánh giá sản phẩm (product metrics)
- Nên đánh giá theo từng giai đoạn
- Có nhiều phương pháp đánh giá, nhìn chung có 5 yếu tố cơ bản sau:
  - kích thước (size) [LOC,...]
  - giá thành (cost) [đơn vị tiền tệ]
  - thời gian thực hiện (duration) [tháng]
  - nhân lực (effort) [người-tháng]
  - chất lượng (quality) [số lượng lỗi phát hiện được]

4.11 Phân loại công cụ CASE
(taxonomy of CASE)

- Đơn giản nhất là công cụ phần mềm *(software tool)*, trợ giúp một mặt nào đó trong sản xuất phần mềm
- *upperCASE* hay *front-end*: trợ giúp trong các giai đoạn đầu tiên như phân tích yêu cầu, đặc tả và thiết kế
- *lowerCASE* hay *back-end* : trợ giúp trong các giai đoạn cuối như cài đặt, tích hợp và bảo trì
- Workbench : công cụ thể hiện bằng đồ họa với từ điển dữ liệu, kiểm chứng tính chắc chắn, sinh báo cáo, sinh màn hình cho các giai đoạn đặc tả và thiết kế. VD: PowerBuilder, Software through Pictures, System Architect,...
  - hỗ trợ 1 hoặc 2 hoạt động (ativities)
  - hoạt động bao gồm nhiều công việc (task). VD: hoạt động mã hóa có các công việc: viết, nối kết, kiểm thử và dò lỗi
  - hỗ trợ định khung nhanh
- Môi trường *(CASE environment)*: hỗ trợ đầy đủ hoặc một phần lớn tiến trình phần mềm [Fuggetta, 1993]

- Từ điển dữ liệu *(data dictionary)*: danh sách các định nghĩa dữ liệu có trong sản phẩm
  - kiểm chứng tính chắc chắn *(consistency checker)*: phản ánh các mục trong đặc tả với thiết kế, trong thiết kế với cài đặt,...
  - sinh báo cáo (report generator): tạo các mã lệnh cho các báo cáo
  - sinh màn hình (screen generator): tạo mã lệnh cho các màn hình
- Điều khiển cấu hình (configuration control)

  VD: khi gặp lỗi, xác định chính xác phiên bản nào của sản phẩm bị lỗi

Phân tích yêu cầu
Đặc tả
Thiết kế
Cài đặt
Tích hợp
Bảo trì

(a) Công cụ

Phân tích yêu cầu
Đặc tả
Thiết kế
Cài đặt
Tích hợp
Bảo trì

(b) Workbench

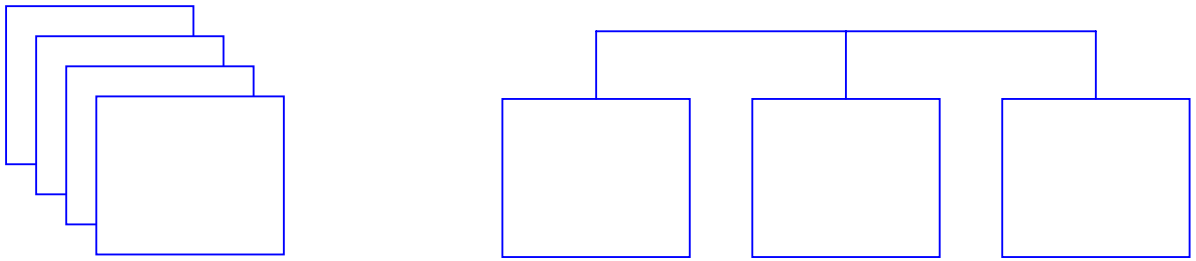
Phân tích yêu cầu
Đặc tả
Thiết kế
Cài đặt
Tích hợp
Bảo trì

(c) Môi trường

Hình 4.14 Giới thiệu công cụ, workbench và môi trường

4.12 Các phiên bản phần mềm (software versions)

- Một phiên bản mới của phần mềm sẽ ra đời mỗi khi bảo trì sản phẩm !
- Phiên bản đã sửa chữa (revisions): là phiên bản mới (new version) sau khi đã được hiệu chỉnh một lỗi. Gọi phiên bản đã sửa chữa trước là *n* thì phiên bản đã sửa chữa sau sẽ là *n+1*
- Phiên bản khác nhau (variations)
  - hai trình điều khiển máy in: in kim, in lade,...
  - sản phẩm được cài đặt trên hai hệ điều hành hay hai loại phần cứng khác nhau,...



Hình 4.16 Các dạng phiên bản khác nhau *(a) revisions (b) variations*

5

KIỂM THỬ
(TESTING)

Nội dung:

- Khái quát chung
- Vấn đề chất lượng
- Kiểm thử không dựa trên thực thi
- Kiểm thử dựa trên thực thi
- Một số dạng kiểm thử khác

5.1 Khái quát chung
(overview)

- [IEEE 610.12, 1990]
  - lỗi *(fault)* : thiếu sót về mặt kỹ thuật (bug)
  - hỏng hóc *(failure)*: hỏng hóc của sản phẩm bắt nguồn từ lỗi
- Lỗi *(error)*: tạo ra bởi người lập trình
- Thẩm tra (verification)
- Công nhận hợp lệ (validation)

5.2 Vấn đề chất lượng
(quality issue)

- Chất lượng *(quality)*: sản phẩm đáp ứng chính xác đặc tả của nó
- Đảm bảo chất lượng phần mềm *(software quality assurance-SQA)*
  - thành lập nhóm SQA
  - nhóm SQA đảm bảo sản phẩm hoạt động đúng chức năng và kiểm tra mỗi khi các nhà phát triển hoàn thành một giai đoạn nào đó
  - nhóm SQA đảm bảo chất lượng của tiến trình phần mềm
- Độc lập về quản lý (managerial independance): nhóm SQA và nhóm phát triển phải được quản lý độc lập với nhau, không can thiệp vào công việc của nhau

5.3 Kiểm thử không dựa trên thực thi (nonexecution-based testing)

5.3.1 walkthroughs

- Nhóm walkthrough khoảng 4-6 người
  - có ít nhất một đại diện thuộc nhóm đặc tả
  - nhà quản lý chịu trách nhiệm về nhóm đặc tả
  - một đại diện khách hàng
  - một đại diện của nhóm thực hiện giai đoạn kế tiếp [Daun, 1984]
  - một đại diện của nhóm SQA, làm trưởng nhóm walkthrough
- Nên chọn những người già dặn kinh nghiệm kỹ thuật [New, 1992]
- Quản lý nhóm walkthrough, có 2 cách thực hiện:
  - hướng theo thành viên: mỗi thành viên trong nhóm đưa ra danh sách chất vấn có các mục không rõ ràng hoặc không chính xác theo quan điểm của mình, đại diện nhóm đặc tả giải trình.
  - hướng theo tài liệu: người có trách nhiệm về tài liệu giải trình từng phần trong tài liệu cho nhóm đưa ra ý kiến. [IEEE 1028, 1988]

5.3.2 Thanh tra (inspection)

- Thành lập nhóm thanh tra
  - khoảng 4 người: nhóm trưởng*(moderator)*, người thiết kế*(designer)*, người cài đặt*(implementer)* và người kiểm thử *(tester)* thuộc nhóm SQA
  - khoảng 3-6 người [IEEE 1028, 1986]: một số vai trò đặc biệt như nhóm trưởng*(moderator)*, người dẫn dắt nhóm phần thiết kế *(reader)*, người viết báo cáo lỗi *(recorder)*
- Thanh tra với nhóm thanh tra, do Fagan đề xuất [Fagan, 1976] nhằm kiểm thử các thiết kế và mã lệnh, gồm 5 bước:
  - bước 1: xem xét khái quát *(overview),* các tài liệu sẽ được thanh tra như đặc tả, thiết kế, mã lệnh, kế hoạch; được đưa ra bởi chính người viết tài liệu đó; tất cả các thành viên trong nhóm sẽ nhận đầy đủ các tài liệu
  - bước 2: chuẩn bị *(preparation),* các thành viên tìm hiểu các tài liệu một cách chi tiết; danh sách các lỗi trong các lần thanh tra gần nhất

- bước 3: thanh tra *(inspection),* một thành viên duyệt qua tất cả các mục và các nhánh trong tài liệu; phát hiện các lỗi; lãnh đạo nhóm thanh tra viết báo cáo về lỗi
- bước 4: làm lại *(rework),* các cá nhân phụ trách các tài liệu sẽ sửa các lỗi được mô tả trong báo cáo về lỗi ở bước 3
- bước 5: tiếp tục *(follow-up)*, nhóm trưởng đảm bảo rằng toàn bộ các tài liệu đã được điều chỉnh; giới thiệu lỗi. [Fagan, 1986]

- Thiết lập danh sách các lỗi tiềm tàng

5.3.3 Điểm mạnh và điểm yếu (strengths and weaknesses of reviews)

- Điểm mạnh
  - rất hiệu quả trong việc tìm kiếm lỗi
  - lỗi được phát hiện sớm do đó sẽ giảm chi phí bảo trì
- Điểm yếu
  - không hiệu quả đối với phần mềm lớn, trừ khi nó được chia thành nhỏ hơn và tương đối độc lập
  - phải xem xét các tài liệu liên quan của phiên bản hiện hành, sẽ không tốt nếu như tài liệu không được cập nhật đầy đủ và chính xác

5.4 Đánh giá công tác thanh tra
(metrics for inspections)

- Phương pháp tính mật độ lỗi *(fault density)*
  - số lỗi trên một trang đặc tả hay thiết kế
  - số lỗi trên 1000LOC
- Phương pháp tính tỷ lệ phát hiện lỗi *(fault detection rate)*
  - số lượng lỗi quan trọng/không quan trọng trên một giờ
- Phương pháp tính hiệu suất dò tìm lỗi *(fault detection efficiency)*
  - số lượng lỗi quan trọng/không quan trọng trên người-giờ

5.5 Kiểm thử dựa trên thực thi (execution-based testing)

- Định nghĩa của Goodenough [Goodenough, 1979]: là tiến trình suy xét dựa vào cách thức xử lý của sản phẩm trên cơ sở thực thi sản phẩm trong một môi trường đã biết với các đầu vào chọn lọc.
- Hệ mô phỏng *(simulator)*: là một mô hình hoạt động của môi trường sản phẩm
- Một số khái niệm
  - tiện ích *(utility)*
  - độ tin cậy *(reliability)*
  - sự mạnh mẽ *(robustness)*
  - hiệu suất *(performance)*
  - sự đúng đắn *(correctness)*: một sản phẩm được xem là đúng nếu như nó đáp ứng được những đặc tả đầu ra của nó, độc lập với tài nguyên máy tính và vận hành dưới những điều kiện cho phép [Goodenough, 1979]

VD:

Đặc tả đầu vào: p: mảng n số nguyên, n>0
Đặc tả đầu ra: q: mảng n số nguyên với $q[0] \le q[1] \le ... \le q[n-1]$

Hình 5.1 Đặc tả đúng cho sắp xếp

```
void trickSort (int p[], int q[])
{
    int i;
    for (i= 0; i < n; i++) q[i] = 0;

}
```

Hình 5.2 Phương thức trickSort đáp ứng đặc tả *Hình 5.1*

Đặc tả đầu vào: p: mảng n số nguyên, n>0
Đặc tả đầu ra: q: mảng n số nguyên với $q[0] \le q[1] \le ... \le q[n-1]$
Các phần tử trong mảng q là hoán vị của các phần tử trong mảng p với giá trị không thay đổi

Hình 5.3 Đặc tả đúng cho sắp xếp

5.6 Một số dạng kiểm thử khác
(other types of testing software)

- Kiểm thử phần mềm phân tán *(testing distributed software)*
  - trên nhiều phần khác nhau của phần cứng
  - trên mạng
  - trao đổi bằng các thông báo
  - sử dụng các công cụ đặc biệt để xác định lỗi, lần vết [Wahl và Schach, 1988]
  - sử dụng tập tin lịch sử (historical file)
- Kiểm thử phần mềm thời gian thực *(testing real-time software)*
  - phụ thuộc vào thời điểm xuất hiện đầu vào và thứ tự của nó
  - khó khăn khi ứng dụng các trường hợp kiểm thử (test cases)
  - có 5 dạng tiếp cận: phân tích cấu trúc, chứng minh tính đúng đắn, kiểm thử theo hệ thống, kiểm thử thống kê và mô phỏng [Quirk, 1985]

6 GIỚI THIỆU VỀ ĐỐI TƯỢNG

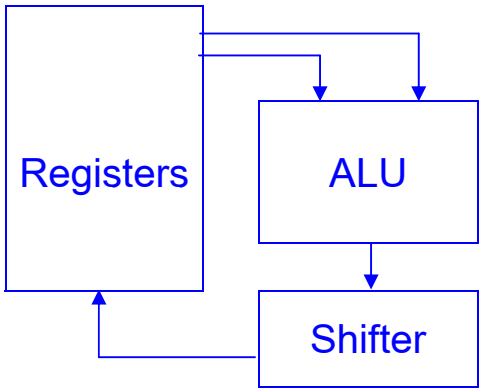
(INTRODUCTION TO OBJECTS)

Nội dung:

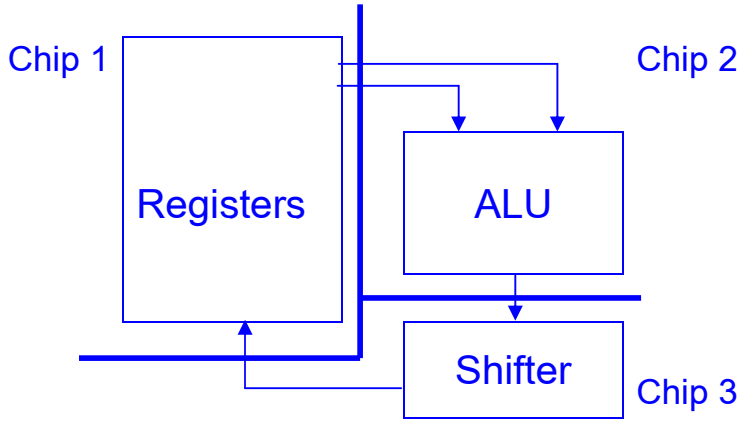
- Giới thiệu về mô-đun
- Độ gắn kết
- Nối kết
- Bao gói dữ liệu
- Kiểu dữ liệu trừu tượng
- Thông tin ẩn
- Đối tượng

6.1 Giới thiệu về mô-đun
(what is a module ?)

- Định nghĩa của [Stevens, Myers, và Constantine, 1974] về mô-đun:
  - tập hợp của một hay nhiều câu lệnh kế tiếp nhau được đặt tên,
  - các phần khác trong chương trình có thể kích hoạt với tên được đặt,
  - có tập hợp các tên biến riêng biệt.
- Mô-đun là một khối đơn các mã lệnh có thể kích hoạt giống như thủ tục, hàm hay phương thức



Hình 6.1 Thiết kế của máy vi tính



Hình 6.2 Máy vi tính *Hình 6.1* chế tạo với 3 chip

6.2 Độ gắn kết (cohesion)

- Là mức độ tương tác bên trong một mô-đun
- Myers định nghĩa 7 thể loại (mức) gắn kết [Myers, 1978b]

7.	Gắn kết chức năng	Tốt
	Gắn kết thông tin	
5.	Gắn kết truyền thông	
4.	Gắn kết thủ tục	
3.	Gắn kết thời gian	
2.	Gắn kết luận lý	
1.	Gắn kết trùng khớp	Xấu

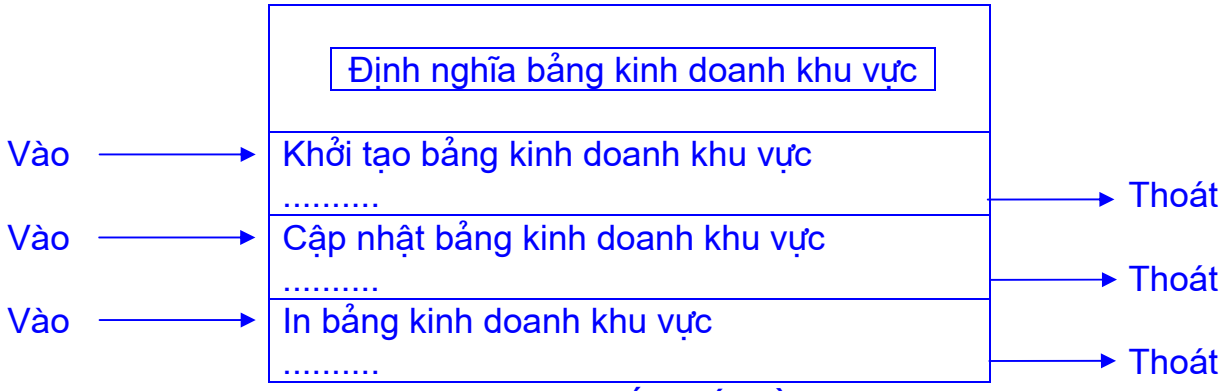
Hình 6.3 Các mức gắn kết

- *Gắn kết trùng khớp (coincidental cohesion)*: mô-đun thực hiện nhiều hành động không liên quan đến nhau

  VD: mô-đun với tên như `in dòng tiếp theo, đảo ngược chuỗi ký tự tham số thứ hai, thêm 7 cho tham số thứ 5, đổi tham số thứ tư thành số thực`

- *Gắn kết luận lý (logical cohesion)*: mô-đun thực hiện chuỗi các hành động có liên quan với nhau, một trong số đó được chọn bởi mô-đun gọi đến
  VD: mô-đun thực hiện việc thêm, xóa, cập nhật các mẩu tin trên tập tin
- *Gắn kết thời gian (temporal cohesion)*: mô-đun thực hiện chuỗi các hành động liên quan với nhau theo thời gian
  VD: mô-đun với tên như `mở tập tin cũ, tạo tập tin mới, mở tập tin giao dịch, in tập tin giao dịch, khởi tạo bảng kinh doanh khu vực, đọc mẩu tin giao dịch đầu tiên,đọc mẩu tin đầu tiên trong tập tin cũ`
- *Gắn kết thủ tục (procedural cohesion)*: mô-đun thực hiện chuỗi các hành động liên quan với nhau theo các bước đúng trình tự phát triển sản phẩm
  VD: mô-đun với tên như `đọc số hiệu bộ phận từ tập tin cơ sở dữ liệu và cập nhật, sửa chữa mẩu tin trên tập tin bảo trì`
- *Gắn kết truyền thông (communcational cohesion)*: mô-đun thực hiện chuỗi các hành động liên quan với nhau theo các bước đúng trình tự phát triển sản phẩm và nếu như mọi hành động đều được thực hiện trên dữ liệu giống nhau
  VD: mô-đun với tên như `cập nhật mẩu tin trong cơ sở dữ liệu và ghi vào sổ` hoặc `tính toán quỹ đạo và gửi ra máy in`

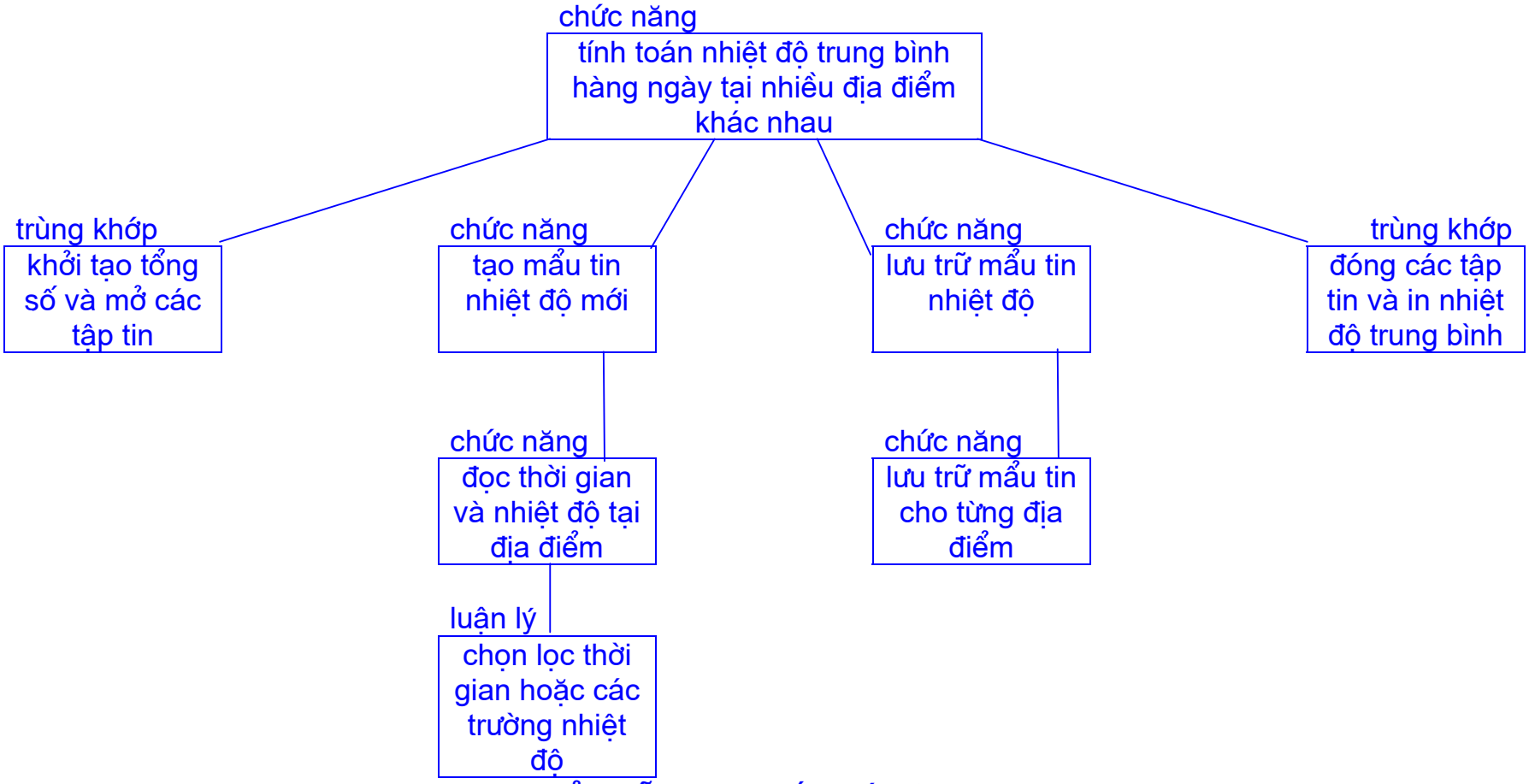
- *Gắn kết thông tin (information cohesion)*: mô-đun thực hiện một số lượng các hành động, mỗi hành động có đầu vào riêng, mã lệnh độc lập và thực hiện trên dữ liệu giống nhau
  VD:



Hình 6.4 Mô-đun gắn kết về thông tin

- Chức năng: mô-đun thực hiện một hành động hoặc nhận lấy một kết quả
  VD: mô-đun với tên như `lấy nhiệt độ lò; tính toán quỹ đạo của điện tử; ghi lên đĩa mềm; tính toán tiền hoa hồng`

❖ Ví dụ về các dạng gắn kết



Hình 6.5 Biểu diễn dạng gắn kết của từng mô-đun

6.3 Nối kết (coupling)

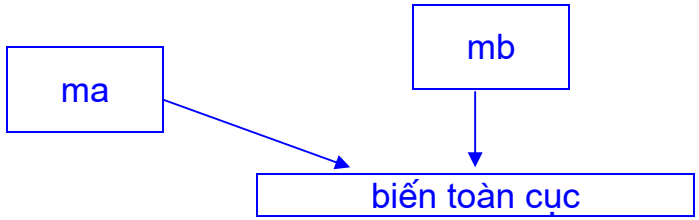
- Là mức độ tương tác giữa hai mô-đun, rất quan trọng trong đánh giá
- Các mức độ nối kết

5.	Nối kết dữ liệu	Tốt
4.	Nối kết nhãn hiệu	
3.	Nối kết điều khiển	
2.	Nối kết chung	
1.	Nối kết nội dung	Xấu

Hình 6.6 Các mức độ nối kết

- *Nối kết nội dung(content coupling)*: hai mô-đun được gọi là nối kết về nội dung nếu như chúng có thể tham khảo rực tiếp nội dung của nhau
  VD: mô-đun *p* tham khảo dữ liệu cục bộ của mô-đun *q*, hoặc mô-đun *p* thay đổi một câu lệnh của mô-đun *q*.

- *Nối kết chung (common coupling)*: hai mô-đun được gọi là nối kết chung nếu cả hai truy xuất đến các dữ liệu toàn cục giống nhau
  VD:



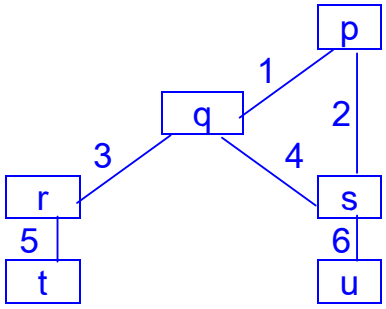
Hình 6.7 Nối kết chung

- *Nối kết điều khiển (control coupling)*: hai mô-đun được gọi là nối kết điều khiển nếu mô-đun này có thể gửi phàn tử điều khiển đến mô-đun kia (có thể điều khiển lẫn nhau)

- *Nối kết nhãn hiệu (stamp coupling)*: hai mô-đun được gọi là nối kết nhãn hiệu nếu như tham số được gửi đi là một cấu trúc dữ liệu và mô-đun được gọi chỉ thao tác trên một vài thành phần của cấu trúc dữ liệu đó

- *Nối kết dữ liệu (data coupling)*: hai mô-đun được gọi là nối kết dữ liệu nếu như tất cả các tham số đều là các mục dữ liệu thuần nhất (homogeneous data items)

- Ví dụ về nối kết



p,t và u truy xuất cùng một cơ sở dữ liệu trong cập nhật

Hình 6.8 Sơ đồ các mô-đun nối kết

số	nhập	xuất
1	kiểu máy bay	cờ trạng thái
2	-	danh sách các phần của máy bay
3	mã hàm	-
4	-	danh sách các phần của máy bay
5	số phần	phần chế tạo
6	số phần	tên phần

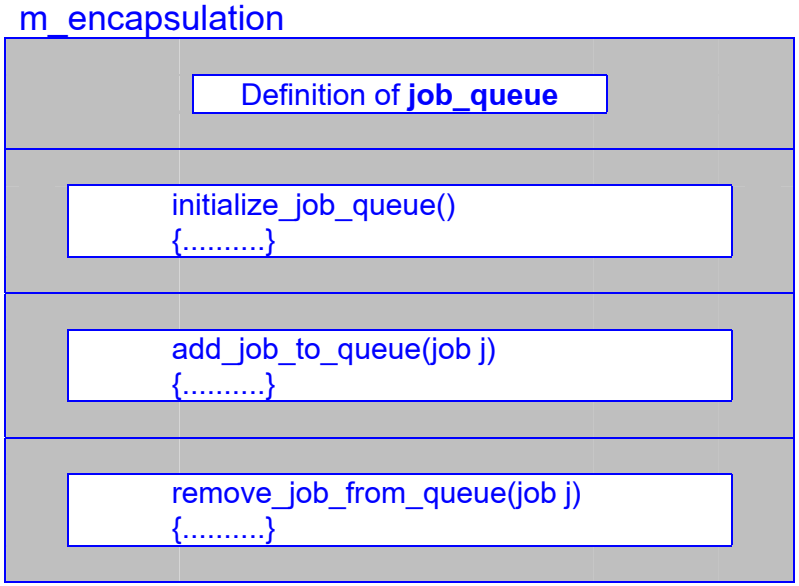
Hình 6.9 Mô tả giao diện *Hình 6.6*

	q	r	s	t	u
p	Dữ liệu	-	Dữ liệu hoặc Nhãn hiệu	Chung	Chung
q		Điều khiển	Dữ liệu hoặc Nhãn hiệu	-	-
r				Dữ liệu	-
s				-	Dữ liệu
t					Chung

Hình 6.10 Nối kết giữa các cặp mô-đun

6.4 Bao gói dữ liệu (data encapsulation)

- Là một dạng của trừu tượng hóa



Hình 6.11 Thiết kế *job_queue* trong một phần hệ điều hành sử dụng bao gói dữ liệu

6.5 Kiểu dữ liệu trừu tượng (abstract data types)

- Kiểu dữ liệu trừu tượng: là kiểu dữ liệu cùng với các thao tác trên chính nó

VD:

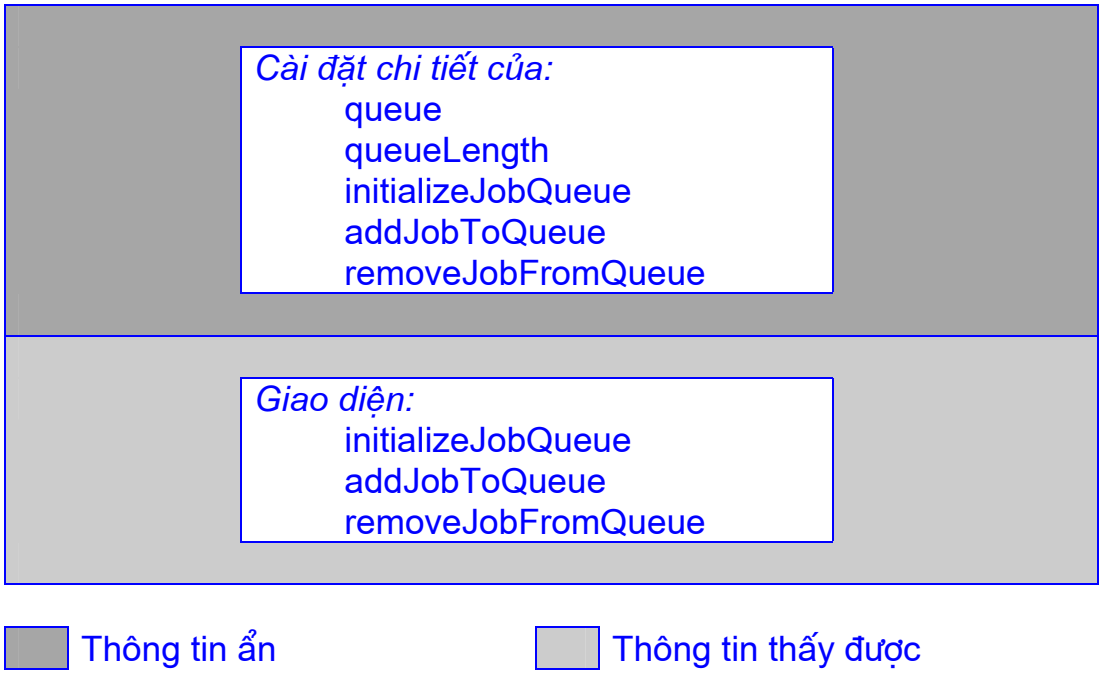
```
class JobQueue
{
    // dữ liệu
    private int queueLength;
    private int queue[25] = new int[25];

    // các phương thức
    public void initializeJobQueue()
    {...}
    public void addJobToQueue(int jobNumber)
    {...}
    public void removeJobfromQueue()
    {...}
}
```

Hình 6.12 JobQueue cài đặt trên Java như là một kiểu dữ liệu trừu tượng

- Trừu tượng hóa thủ tục (procedural abstraction)
- Trừu tượng hóa lặp (iteration abstraction) [Liskov và Guttag, 1986]

6.6 Thông tin ẩn
(information hiding)

- Do Parnas đề xuất [Parnas, 1971, 1972a, 1972b]



Hình 6.13 Kiểu dữ liệu trừu tượng với thông tin ẩn

6.7 Đối tượng (objects)

- Là kiểu dữ liệu trừu tượng hay mô-đun với thông tin được gắn kết
- Là thể hiện (instance) của một kiểu dữ liệu trừu tượng
- Hỗ trợ thừa kế, khái niệm được giới thiệu đầu tiên ở ngôn ngữ lập trình Simula67 [Dahl và Nygaard, 1966; Dahl, Myrhaug and Nygaard, 1973]
- Hỗ trợ đa hình và liên kết động
- Một số vấn đề:
  - gắn kết giữa các mô-đun được thừa kế và các mô-đun định nghĩa mới
  - nối kết chung với định nghĩa công cộng: *public*

Hình 6.14 Các khái niệm chính

7 SỬ DỤNG LẠI, DỄ DI CHUYỂN VÀ VẬN HÀNH TƯƠNG TÁC

(REUSABILITY, PORTABILITY, AND INTEROPERABILITY)

Nội dung:

- Các khái niệm về sử dụng lại
- Trở ngại của việc sử dụng lại
- Sử dụng lại trong các giai đoạn thiết kế và cài đặt
- Dễ di chuyển
- Vận hành tương tác

7.1 Các khái niệm về sử dụng lại
(reuse concepts)

- Sử dụng lại là việc lấy một bộ phận của sản phẩm này để phát triển thuận lợi sản phẩm khác (với chức năng khác)
- Bộ phận được sử dụng lại có thể là một mô-đun, một đoạn mã lệnh, một thiết kế, một phần hướng dẫn sử dụng, một tập dữ liệu kiểm thử, một ước lượng về thời gian và giá thành,...
- Có 2 dạng sử dụng lại:
  - ngẫu nhiên (accidental reuse), một số bộ phận của sản phẩm cũ vẫn được sử dụng cho sản phẩm mới
  - thảo luận (delibrate reuse), bộ phận đang được thực hiện sẽ được sử dụng lại trong tương lai
- Theo thống kê, khoảng 85% bộ phận của sản phẩm cũ được sử dụng lại
- *Một số ví dụ:*
  - các giao diện lập trình ứng dụng của Microsoft (application programming interface - API)
  - các thư viện của C, C++,...

7.2 Trở ngại của việc sử dụng lại
(impediments to reuse)

- Bản ngã, các nhà chuyên nghiệp thường viết các bộ phận từ đầu chứ không sử dụng lại của người khác
- Chất lượng của bộ phận sử dụng lại
- Phục hồi lại các bộ phận cũ hữu ích
- Giá thành cao khi sử dụng lại

- Một số trường hợp nghiên cứu
  - Raytheon Missile Systems Division: 40-60% thiết kế và mô-đun,...
  - Toshiba Software Factory: 32% tài liệu, thiết kế 33%, mã lệnh 48%,...
  - NASA Software: 35% mã lệnh, 45% mô-đun,...
  - ...

7.3 Sử dụng lại trong các giai đoạn thiết kế và cài đặt
(reuse during the design and implementation phases)

- Thư viện(libraries) hay bộ công cụ (toolkit)
  - GUI (graphical user interface), Java Abstract Windowing Toolkit,...
- Khung ứng dụng (framework): kết hợp sự điều khiển luận lý của thiết kế, xây dựng ứng dụng với các thao tác giống hệt sản phẩm trước đó
  - MacApp cho các máy Macintosh,
  - The Microsoft Foundation Class Library - MFC, Borland's Visual Component Library - VCL, Object Windows Library - OWL
- Mẫu thiết kế (design patterns)
  - Abstract Factory [Gamma, Helm, Johnson và Vlissides, 1995]
- Kiến trúc phần mềm (software architecture)

❖**Sử dụng lại và bảo trì**

Hoạt động	% giá thành	% tiết kiệm do sử dụng lại
Phát triển		
Bảo trì	33%	9.3%
	67%	17.9%

Hình 7.1 Số liệu với 40% bộ phận được được sử dụng lại

7.4 Dễ di chuyển (portability)

- Định nghĩa [Mooney, 1990] : một sản phẩm được cho là dễ di chuyển nếu với chi phí không lớn lắm có thể thực thi được trên một máy tính mới thay vì phải viết lại từ đầu
- Một số vấn đề cần quan tâm
  - không tương thích phần cứng (hardware incompatibilities)
  - không tương thích hệ điều hành (operating system incompatibilities)
  - không tương thích về số hoá phần mềm (numerical software incompatibilities). VD: 16 bits hay 32 bits
  - không tương thích trình biên dịch (compiler incompatibilities)
- Một số kỹ thuật nhằm đạt được tính dễ di chuyển
  - hệ thống phần mềm dễ di chuyển (portable system software)
  - hệ thống phần mềm ứng dụng dễ di chuyển (portable application software)
  - dữ liệu dễ di chuyển (portable data)

7.5 Vận hành tương tác (interoperability)

- *Định nghĩa:* là sự hợp tác qua lại trên các đối tượng mã lệnh từ nhiều nhà sản xuất phần mềm khác nhau, được viết trên nhiều ngôn ngữ lập trình khác nhau và thực thi trên nhiều hệ điều hành khác nhau
- Một số ví dụ
  - OLE (object linking and embedding),1990, là một phần của Windows 3.0, hỗ trợ các tài liệu phức hợp về xử lý văn bản, bảng tính,...
  - COM (component object model) là bước phát triển tiếp theo của OLE
  - ActiveX năm 1996, có nối kết với Internet; giống OLE và COM
  - DCOM (distributed COM), 1996, hỗ trợ phân tán trên các nền hệ điều hành khác nhau
  - COM+, COM3 là các phiên bản hướng đối tượng
  - CORBA (common object request broker architecture) hỗ trợ các ứng dụng phần mềm vận hành tương tác trên các máy khác nhau trong cùng một môi trường phân tán [OMG, 1993] (Object Management Group)

8 HOẠCH ĐỊNH VÀ ƯỚC LƯỢNG

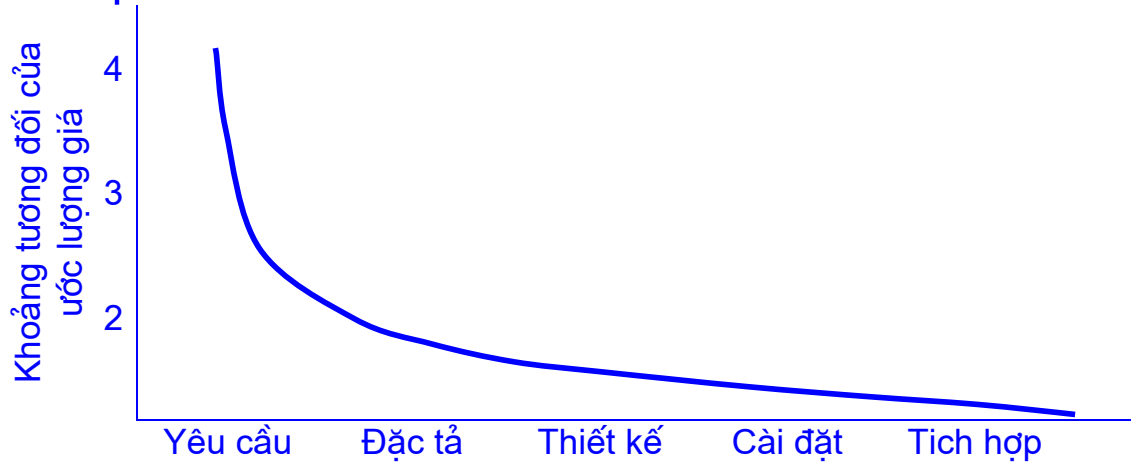
(PLANNING AND ESTIMATING)

Nội dung:

- Hoạch định và tiến trình phần mềm
- Ước lượng giá thành
- Ước lượng thời gian
- Đo kích thước sản phẩm
- Các kỹ thuật ước lượng giá thành
- Mô hình COCOMO trung gian
- Mô hình COCOMO II
- Khung kế hoạch quản lý phần mềm
- Công cụ CASE cho hoạch định và ước lượng

8.1 Hoạch định và tiến trình phần mềm
(planning and the software process)

- Không thể hoạch định toàn bộ dự án phần mềm nhay từ khi bắt đầu cho đến lúc phân phối sản phẩm cho khách hàng
- Thường có sự khác biệt giữa mô tả của khách hàng và các đặc tả của nhóm phát triển



Hình 8.1 Mô hình ước lượng các khoảng giá tương đối

8.2 Ước lượng giá thành
(estimating cost)

- Khách hàng cần phảI biết mình sẽ trả bao nhiêu tiền
- Ước lượng:
  - *thấp*, công ty thua lỗ
  - *cao*, khách hàng không đặt hàng
- Giá nội (internal cost): giá thành để phát triển phần mềm
  lương của đội ngũ phát triển, các nhà quản lý, các nhân viên hỗ trợ,...
  chi phí phần cứng, phần mềm dùng để phát triển
  các chi phí về thuê mướn ,tiện ích, lương cho các nhà quản trị cấp cao
- Giá ngoại (external cost): giá thành khách hàng phảI trả
  - giá nội
  - lợi nhuận
  - chi phí bảo trì

8.3 Ước lượng thời gian
(estimating duration)

- Khách hàng cần phải biết khi nào mình nhận được sản phẩm, khi đó cần cân nhắc các yếu tố sau khi ước lượng:
  - *nhanh*, công ty mất uy tín nếu không hoàn thành
  - *chậm*, khách hàng sẽ không đặt hàng
- Khó chính xác do quá trình ước lượng chịu ảnh hưởng của nhiều yếu tố
- Nhân tố con người (human factor) có tác động rất lớn khi ướng lượng
- Thống kê của Sackman [Sackman, Erikson và Grant, 1968]
  - về sự khác biệt trên 2 lập trình viên được đào tạo như nhau
    - kích thước sản phẩm: 6-1
    - thời gian thực thi sản phẩm : 8-1
    - thời gian phát triển: 9-1
    - viết mã lệnh: 18-1
    - thời gian gỡ rối: 28-1
  - về sự khác biệt trên 2 lập trình viên có 11 năm kinh nghiệm: 5/1

8.4 Đo kích thước sản phẩm
(metrics for the size of a product)

- Theo **số lượng dòng mã lệnh - LOC** (lines of code)
- Theo **số lượng chỉ thị đã phân phối tính theo đơn vị ngàn – KDSI** (thousand delivered source of codeI [van der Poel và Schach, 1983])

  Các vấn đề gặp phải với các phương pháp (1) và (2):
  - tính toán kích thước cho các giai đoạn khác: phân tích yêu cầu,…
  - càI đặt trên hai NNLT khác nhau: C, Java, Lisp,…
  - cách đếm số dòng mã lệnh: mã lệnh thực thi, định nghĩa dữ liệu,…
  - mã lệnh tạo công cụ dùng để hỗ trợ phát triển
  - sinh mã tự động, thiết kế giao diện trực tiếp (GUI),…
  - giá thành của sản phẩm phụ thuộc vào ước lượng LOC

- Theo **số lượng toán tử và toán hạng** (operators and operands) [Halstead, 1977; Shen, Conte và Dunsmore, 1983]

- Theo **FFP** (files-flows-processes) [van de Poel và Schach, 1983]
  - áp dụng đối với các ứng dụng xử lý dữ liệu có kích thước trung bình
  - sử dụng từ 2 đến 10 người/năm

$$S = Fi + Fl + Pr$$
$$C = d \times S$$

Trong đó:

File: tập hợp các mẩu tin (vật lý hay luận lý) có liên hệ với nhau

Flow: giao diện dữ liệu giữa sản phẩm và môl trường như màn hình, báo cáo,…

Process: về chức năng mà nói đó chính là một định nghĩa logic hay toán học dùng để thao tác trên dữ liệu

hằng số ***d***: độ đo hiệu quả của sản phẩm, tuỳ thuộc vào từng công ty

- Theo **điểm chức năng-FP** (function point) [Albrecht, 1979; Albrecht và Gaffney, 1983]
  - Các thuật ngữ:
    - UFP (unadjsuted function points) : các điểm không thích ứng
    - TCF (technical complexity factor) : nhân tố về độ phức tạp kỹ thuật
    - DI (degree of influence) : mức độ ảnh hưởng [0..70]

- Các công thức và thông số chính:

	Level of Complexity		
Component	**Simple**	**Average**	**Complex**
Input item *(Inp)*	3	4	6
Output item *(Out)*	4	5	7
Inquiry *(Inq)*	3	4	6
Master file *(Maf)*	7	10	15
Interface *(inf)*	5	7	10

Hình 8.2 Bảng giá trị các điểm chức năng

Cách tính chính xác:
UFP = tổng các điểm chức năng theo độ phức tạp
DI = tổng các mức độ ảnh hưởng
TCF = 0.65 + 0.01 × DI
FP = UFP × TCF

Ví dụ về cách tính đơn giản:
FP = 4×Inp + 5×Out + 4×Inq + 10×Maf + 7×Inf

1. Data communications
2. Distributed data processing
3. Performance criteria
4. Heavily utilized hardware
5. High transaction rates
6. Online data entry
7. End-user effciency
8. Online updating
9. Complex computations
10. Reusability
11. Ease of installation
12. Ease of opearation
13. Portability
14. Maintainability

(Các giá trị về ảnh hưởng thay đổi từ 0 đến 5)

Hình 8.3 Bảng các nhân tố kỹ thuật

- Theo **điểm chức năng mở rộng-Mk II FPs** [Symons, 1987]
  - chính xác hơn khi tính toán giá trị của UFP
  - phần mềm được chia thành các giao dịch (transaction), mỗi giao dịch bao gồm các thành phần sau:
    - một đầu vào (an input)
    - một quá trình (a process)
    - một đầu ra (an ouput)
  - giá trị của UFP được tính toán dựa trên số lượng các thành phần trong từng giao dịch
  - được sử dụng rộng rãI trên thế giới [Boehm, 1997]

- Theo **các điểm đặc điểm** (feature points) [Jones, 1991]
  - Sử dụng cho các phần mềm chịu ảnh hưởng mạnh về giảI thuật:
    - thời gian thực (real-time software)
    - nhúng (embedded software)
    - truyền thông (communication software)
  - *Đang trong thời gian kiểm chứng độ chinh xác*

- Công thức:

  Feature Points = FP – 3×Maf + 3×Alg

  Trong đó:
  - FP : số lượng điểm chức năng tính toán theo phương pháp (5)
  - Maf : số lượng tập tin chính
  - Alg : số lượng giảl thuật sử dụng

- Theo **3D-FPs** (3D function points) [Boehm, 1997]
  - phát triển bởi công ty chế tạo máy bay Boeing
  - sử dụng phương pháp tính toán dựa theo FPs
  - dùng cho các phần mềm: khoa học(scientific software),thời gian thực
  - *Chưa được công bố chính thức*

8.5 So sánh một số phương pháp đo kích thước phần mềm

	Assembler version	Ada version
Kích thước mã nguồn	70 KDSI	25 KDSI
Giá thành phát triển	$1,043,000	$590,000
KDSI trên mỗi người-tháng	0.335	0.211
Giá trung bình của mỗi câu lệnh	$14.90	$23.60
Số điểm chức năng trên mỗi người-tháng	1.65	2.92
Giá của mỗi điểm chức năng	$3,023	$1,170

Hình 8.4 So sánh giữa hai sản phẩm Assembler và Ada [Jones, 1987] (©1987 IEEE)

- Theo nghiên cứu của Jones [Jones, 1987], độ sai lệnh khi sử dụng hai phương pháp trên là:
  - KDSI : 800%
  - FP : 200% (tốt hơn rất nhiều !)

8.6 Các kỹ thuật ước lượng giá thành
(techniques of cost estimation)

- Dựa vào đánh giá của chuyên gia trên các phần mềm tương tự (expert judgement by analogy)
  - tổng hợp kết quả từ việc tham khảo ý kiến của nhiều chuyên gia
  - độ chính xác cao khi chuyên gia đã từng làm việc trên lĩnh vực đó

- Tiếp cận dưới lên (bottom-up approach)
  - chia sản phẩm thành các thành phần nhỏ hơn
  - ước lượng thời gian và giá thành trên từng thành phần độc lập nhau
  - *ưu điểm:* nhanh và chính xác hơn khi ước lượng toàn bộ
  - *khuyết điểm:* toàn bộ sản phẩm > tổng các thành phần

- Các mô hình ước lượng giá thành theo giải thuật (algorithmic cost estmation models)
  - các giá trị FPs hay FFP được sử dụng như trị đầu vào của mô hình
  - các sản phẩm được xử lý giống nhau
  - một số mô hình chính:
    - SLIM [Putnam, 1978]
    - RCA Price S [Freiman và Park, 1979]
    - COCOMO [Boehm, 1981] [Boehm, 1984b] (COnstructive COst MOdel): bao gồm 3 mô hình ước lượng từ lớn đến nhỏ (macroestimation -> microestimation)

8.7 Mô hình COCOMO trung gian
(intermediate COCOMO)

- Mức độ phức tạp và chi tiết ở mức trung bình
- Dựa trên hai yếu tố:
  - chiều dàI sản phẩm tính theo KDSI
  - mức độ khó khăn (difficulty) khi phát triển sản phẩm, có 3 dạng được cho theo bảng sau:

Software project	a_b	b_b	c_b	d_b
Dạng cơ bản (organic)	3.20	1.05	2.50	0.38
Dạng trung bình (semi-detached)	3.00	1.12	2.50	0.35
Dạng lớn (embedded)	3.60	1.20	2.50	0.32

Hình 8.5 Bảng mức độ khó khi phát triển sản phẩm

- Các công thức:

$$\textbf{Nominal Effort [NE]} = a_b \times (\textbf{KDSI})\ \exp(b_b)$$

$$\textbf{Date [D]} = c_b \times (\textbf{NE})\ \exp(d_b)$$

- Ví dụ với 12 KDSI và mức độ khó cơ bản:

$$NE = 3.2 \times (12)^{1.05} = 43 \text{ người-tháng}$$

- Tiếp theo NE được nhân với 15 hệ số phát triển được cho trong bảng sau:

Cost Drivers	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.16	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual machine volatility*		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experiences	1.29	1.13	1.00	0.91	0.82	
Programmer capabilitiy	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience*	1.21	1.10	1.00	0.90		
Programming language experiences	1.14	1.07	1.00	0.95		
Project Attributes						
Use of modern programming pratices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

* *The underlying virtual machine:* độ phức tạp về phần cứng hoặc phần mềm để hoàn thành công việc

Hình 8.6 Các hệ số nhân của mô hình COCOMO trung gian [Boehm, 1984b] (©1984 IEEE)

- Ví dụ về phần mềm xử lý truyền thông trên microprocessor với: 10 KDSI và là dạng dự án lớn:

$$NE = 2.8 \times (10)^{1.20} = 44 \text{ người-tháng}$$

với hệ số nhân kết quả có được từ Hình 8.7 là 1.35 ta được:

$$NE = 1.35 \times 44 = 59 \text{ người-tháng}$$

Cost Drivers	Situation	Rating	Effort Multiplier
Required software reliability	Serious financial consequences of software fault	High	1.15
Database size	20,000 bytes	Low	0.94
Product complexity	Communications processing	Very high	1.30
Execution time constraint	Will use 70% of available time	High	1.11
Main storage constraint	45K of 64K store (70%)	High	1.06
Virtual machine volatility*	Based on commercial microprocessor hardware	Nominal	1.00
Computer turnaround time	Two hour average turnaround time	Nominal	1.00
Analyst capabilities	Goo senior analyst	High	0.86
Applications experiences	Three years	Nominal	1.00
Programmer capabilitiy	Good senior programmers	High	0.86
Virtual machine experience*	Six months	Low	1.10
Programming language experiences	Twelve months	Nominal	1.00
Use of modern programming pratices	Most techniques in use over one year	High	0.91
Use of software tools	At basic minicomputer tool level	Low	1.10
Required development schedule	Nine months	Nominal	1.00

Hình 8.7 Các hệ số nhân cho phần mềm truyền thông trên microprocessor [Boehm, 1984b] (©1984 IEEE)

8.8 Mô hình COCOMO II
(COCOMO II)

- Bối cảnh lịch sử của COCOMO:
  - ra đời năm 1981
  - chỉ có duy nhất mô hình thác nước về chu trình sống của phần mềm
  - thực thi trên các máy mainframes
  - không nhận biết được một số công nghệ mới như: hướng đối tượng, mô hình khách-chủ
- COCOMO II [Boehm et al., 1995] được xây dựng trên cơ sở xem xét lại toàn bộ COCOMO và phức tạp hơn:
  - nhận biết các công nghệ mới
  - tương thích với các dạng chu trình sống của phần mềm hiện đại
  - nhận biết các ngôn ngữ lập trình thuộc thế hệ thứ 4
  - dựa trên 3 mô hình chính, thay đổi thông số liên quan đến *Hình 8.5*
  - 17 hệ số nhân, trong đó có 7 hệ số nhân mới,…
  - được sử dụng trong 83 dự án trên nhiều lĩnh vực khác nhau
- *Còn quá mới để đánh giá kết quả ứng dụng*

8.9 Theo dõi các ước lượng về thời gian và giá thành
(tracking duration and cost estimate)

- Phải đối chiếu kết quả có được sau khi hoàn thành với dự đoàn
- Phải điều chỉnh ngay khi dự đoán sai ảnh hưởng đến tiến trình dự án
  VD: dành cho giai đoạn đặc tả là 3 tháng và 7 người nhưng đến 4 tháng với 10 người mà vẫn chưa hoàn thành.

- Theo dõi sát sao mức độ ảnh hưởng đến tiến trình chung trong toàn bộ quá trình sử dụng việc ước lượng

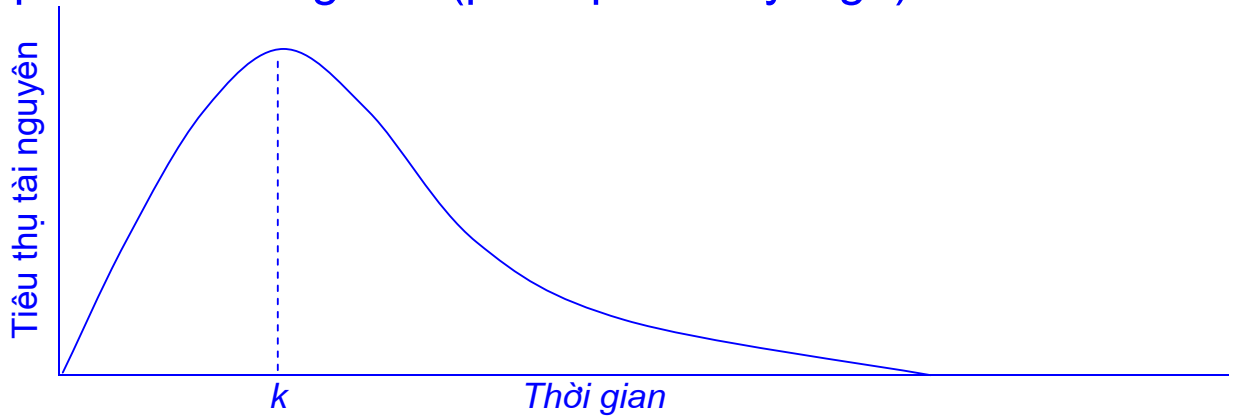
8.10 Các thành phần của kế hoạch quản lý dự án phần mềm
(components of a software project management plan - SPMP)

- Một SPMP có 3 thành phần chính
  - công việc sắp thực hiện
  - các nguồn tài nguyên: con người (giữ vai trò quan trọng nhất), phần cứng và phần mềm
  - tiền phải trả cho toàn bộ dự án
- Mức tiêu thụ tài nguyên theo thời gian [Norden, 1958] cho các dự án lớn với công thức tính xấp xỉ theo thời gian *t* (phân phối Rayleigh):

$$R_c = \frac{t}{k^2} e^{-t^2 / 2k^2}$$

với:

$(0 \le t \le \infty)$



Hình 8.8 Đường cong Rayleigh thể hiện sự tiêu hao tài nguyên theo thời gian

8.11 Khung kế hoạch quản lý dự án phần mềm
(software project management plan framework)

1. Introduction
   - 1.1 Project Overview
   - 1.2 Project Deliverables
   - 1.3 Evolution of the Software Project Management Plan
   - 1.4 Reference Materials
   - 1.5 Definitions and Acronyms
2. Project Organization
   - 2.1 Process Model
   - 2.2 Organizational Structure
   - 2.3 Organizational Boudaries and Interfaces
   - 2.4 Projects Responsibilities
3. Managerial Process
   - 3.1 Management Objectives and Priorities
   - 3.2 Assumptions, Depedencies, and Constraints
   - 3.3 Risk Management
   - 3.4 Monitoring and Controlling Mechanisms
   - 3.5 Staffing Plan
4. Technical Process
   - 4.1 Methods, Tools, and Techniques
   - 4.2 Software Documentation
   - 4.3 Project Support Function
5. Work Packages, Schedules, and Budget
   - 5.1 Work Packages
   - 5.2 Dependencies
   - 5.3 Resources Requirements
   - 5.4 Budget and Resource Allocation
   - 5.5 Schedule

Additional Components

Hình 8.9 Các thành phần trong kế hoạch quản lý dự án phần mềm của IEEE [IEEE 1058.1, 1987] (© 1987 IEEE.)

8.12 Kế hoạch quản lý dự án phần mềm theo IEEE
(IEEE software project management plan - SPMP)

1. Giới thiệu

- Cung cấp cái nhìn tổng quan về dự án và sản phẩm sắp phát triển

1.1 Khái quát về dự án:

- Mô tả tóm tắt các mục tiêu của dự án, sản phẩm sẽ được phân phối, các hoạt động, các sản phẩm kết quả trong từng công việc
- Liệt kê các giai đoạn quan trọng: các yêu cầu về tài nguyên, lịch làm việc chính và ngân sách chung

1.2 *Phân phối sản phẩm:* danh sách các mục sẽ được phân phối đến khách hàng và các ngày phân phối tương ứng

1.3 *Phát triển của SPMP:* các cập nhật thường xuyên liên quan đến khách hàng và công ty phát triển phần mềm trên các vấn đề kinh nghiệm và các thay đổi. Mô tả các hình thức thủ tục và cách thức để thay đổi kế hoạch chung

1.4 *Tài liệu tham khảo:* liệt kê các tàl liệu tham khảo sử dụng trong SPMP

1.5 *Các định nghĩa và từ viết tắt:* chứa đựng các thông tin nhằm làm cho mọi người hiểu rõ SPMP

2. Tổ chức dự án

- Chỉ rõ sản phẩm sẽ được phát triển như thế nào theo quan điểm của tiến trình xử lý phần mềm và cấu trúc tổ chức của các nhà phát triển.

2.1 Mô hình xử lý:

- Các hoạt động: thiết kế của sản phẩm,kiểm thử trên sản phẩm.
- Các chức năng của sản phẩm: quản lý dự án hoặc quản lý cấu hình.

- Đặc tả các khía cạnh mấu chốt như các giai đoạn quan trọng, các mốc ranh giới, các xem xét lại, các công việc của sản phẩm và các phân phối.

2.2 *Cấu trúc tổ chức:* mô tả cấu trúc quản lý để tổ chức phát triển sản phẩm, quan trọng là chỉ rõ quyền hạn và trách nhiệm của tổ chức bên trong

2.3 Tổ chức các ranh giới và giao diện:

- Các thành viên của dự án phải giao tiếp với khách hàng và các thành viên khác ở cơ quan của khách hàng
- Trong các dự án lớn nên có thêm một số hợp đồng con
- Ranh giới quản lý và quản trị giữa dự án và các thực thể khác phải được dự kiến
- Trong các dự án có nhiều kiểu nhóm khác nhau thì ranh giới quản lý và quản trị cũng phải được định nghĩa rõ ràng

2.4 *Trách nhiệm trong dự án:* mỗi chức năng hay hoạt động của dự án phải có xác định cá nhân chịu trách nhiệm cụ thể

3. Quy trình quản lý

3.1 Quản lý các mục đích và các quyền ưu tiên:

- Mô tả mục đích và quyền ưu tiên dành cho quản lý, cơ chế và tần xuất báo cáo, độ ưu tiên tương đối giữa các yêu cầu
- Kế hoạch làm việc và ngân sách cho dự án, các thủ tục quản lý rủi ro.

3.2 *Sự đảm đương, phụ thuộc và ràng buộc:* các đảm đương và ràng buộc trong tài liệu đặc tả

3.3 Quản lý rủi ro: liệt kê các nhân tố gắn liền với dự án và cơ chế lần vết các yếu tố rủi ro

3.4 *Các cơ chế giám sát và điều khiển:* mô tả chi tiết các cơ chế báo cáo, xem xét và sổ sách kế toán

3.5 Kế hoạch nhân sự: liệt kê số lượng và kiểu nhân lực và thời gian làm việc

4. Quy trình kỹ thuật

4.1 *Các phương thức, công cụ và kỹ thuật:* mô tả chi tiết các khía cạnh phần cứng, phần mềm, hệ điều hành dùng cho phát triển sản phẩm cũng như hệ thống đích mà sản phẩm sẽ thực thi trên đó. Một số khía cạnh khác như các kỹ thuật phát triển, các kỹ thuật kiểm thử, cấu trúc nhóm làm việc, các ngôn ngữ lập trình, các công cụ CASE. Các chuẩn kỹ thuật về tài liệu và mã lệnh, các tham khảo đến các tài liệu khác cũng như thủ tục để phát triển và sửa đổi các công việc sản phẩm.

4.2 *Tài liệu phần mềm:* tài liệu phân tích yêu cầu, các mốc quan trọng, các vạch ranh giới và các xem xét

4.3 *Các chức năng hỗ trợ dự án:* kế hoạch chi tiết hỗ trợ các chức năng như quản lý cấu hình, đảm bảo chất lượng và kế hoạch kiểm thử

5. Các gói công việc, kế hoạch và ngân sách

5.1 *Các gói công việc:* đặc tả các gói công việc và các công việc tương ứng đã được chia thành các hoạt động và nhiệm vụ

5.2 Các phụ thuộc: phụ thuộc giữa các gói công việc và các sự kiện bên ngoài

5.3 Yêu cầu tài nguyên: mô tả tất cả các yêu cầu về tài nguyên

5.4 Ngân sách và việc phân bổ tài nguyên: trình bày ngân sách cho từng thành phần, tài nguyên, các chức năng, các hoạt động và các nhiệm vụ

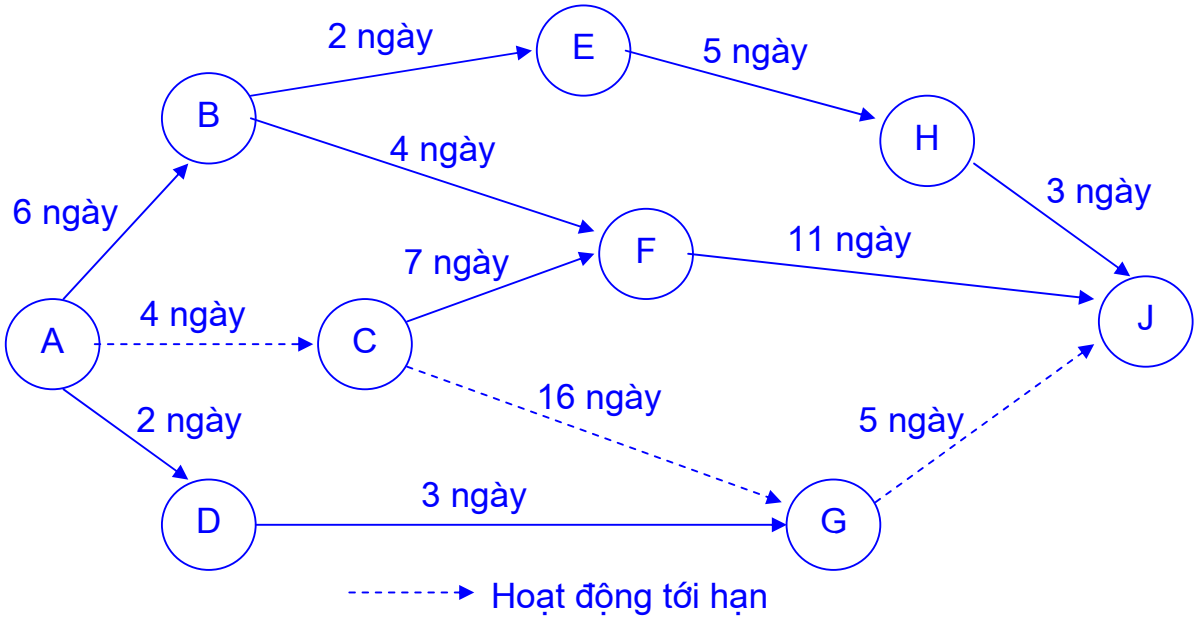
5.5 Kế hoạch làm việc: kế hoạch làm việc chi tiết cho mỗi thành phần, kế hoạch tổng thể để đảm bảo cho dự án được thực hiện đúng thời gian

Các thành phần bổ sung

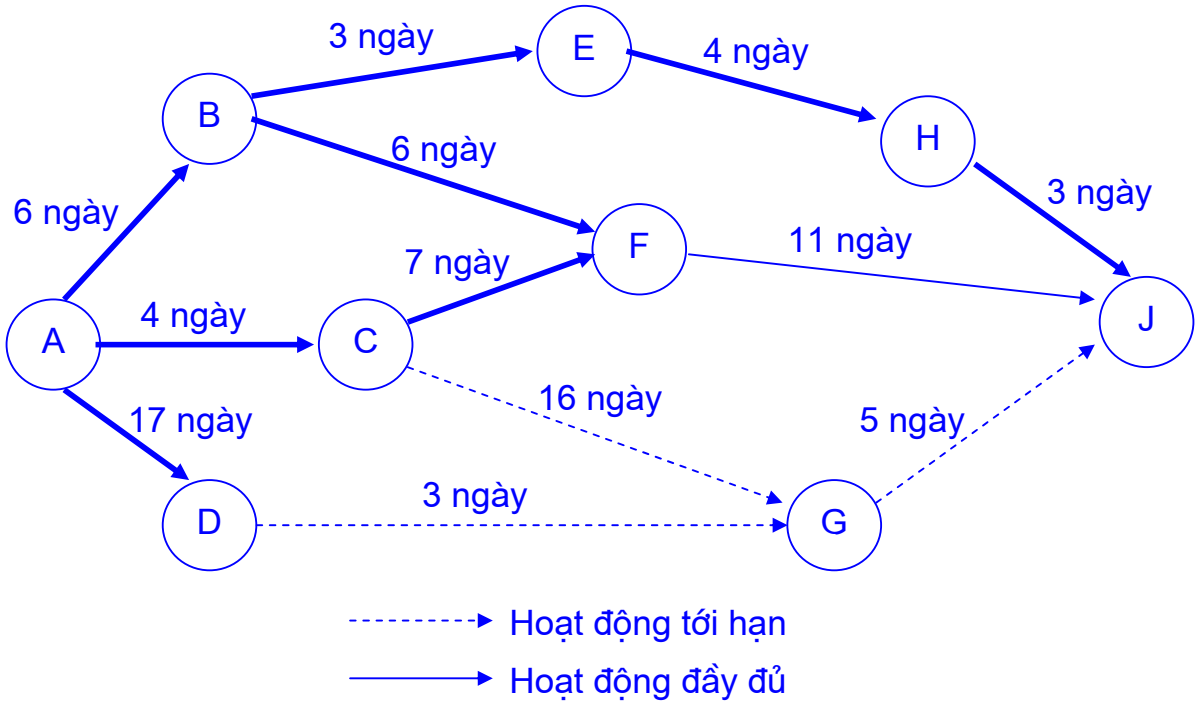
- Cần thiết trong một số dự án
- Kế hoạch quản lý các thầu phụ, kế hoạch an ninh, kế hoạch đào tạo, kế hoạch mua phần cứng, kế hoạch cài đặt và kế hoạch bảo trì

8.13 Công cụ CASE cho hoạch định và ước lượng
(CASE tools for planning and estimating)

- CPM (crotical path management)
- PERT (program evaluation review techniques) [Moder, Phillips và Davis, 1983]



Hình 8.10 Mô hình PERT với các thời gian hoạt động và đường tới hạn



Hình 8.10 Mô hình PERT được cập nhật với tại ngày thứ 17

Phần

2

CÁC GIAI ĐOẠN TRONG CHU TRÌNH SỐNG CỦA PHẦN MỀM

(THE PHASES OF THE SOFTWARE LIFE CYCLE)

9 GIAI ĐOẠN PHÂN TÍCH YÊU CẦU

(REQUIREMENTS PHASE)

Nội dung:

- Khái quát chung
- Khởi động việc phân tích yêu cầu
- Các kỹ thuật phân tích yêu cầu
- Nhân tố con người
- Sử dụng định khung nhanh để đặc tả
- Sử dụng lại mô hình định khung nhanh

9.1 Khái quát chung
(overview)

- Xác định cái mà khách hàng cần (needs) chứ không phảI cái mà khách hàng muốn (wants)
- Phân tích càng chính xác càng tốt thực trạng hiện nay của khách hàng
- Nhận biết những khả năng, những cái cần có trong sản phẩm
- Không có khái niệm phân tích yêu cầu hướng đối tượng

- Một số khó khăn chính khi thực hiện:
  - thông thường khách hàng không biết họ cần gì
  - ngay cả khi khách hàng biết rõ mình cần gì thi cũng sẽ khó khăn khi chuyển tải những thông tin này cho nhà phát triển theo hướng tin học hóa !

9.2 Khởi động việc phân tích yêu cầu (initialisation of requirements)

- Bắt đầu khi các thành viên của nhóm phân tích yêu cầu (requirements analysis team - RAT) tiếp xúc với khách hàng
- Thông thường thì khách hàng sẽ sắp xếp những buổi phỏng vấn đầu tiên (initial interviews)
- Các buổi phỏng vấn thêm sẽ được xếp lịch trong tiến trình phỏng vấn (interview process)

- Tiến trình phỏng vấn kết thúc khi nhóm RAT nhận thấy đã nắm bắt được các thông tin liên quan từ :
  - khách hàng
  - những người sử dụng tương lai của sản phẩm

9.3 Các kỹ thuật phân tích yêu cầu
(requirements analysis techniques)

- Phỏng vấn theo cấu trúc (structered interview)
  - chuẩn bị sẵn các câu hỏi cụ thể dạng đóng (specific preplanned close-ended questions) để nêu ra
    VD: *khách hàng có thể được hỏi như:* có bao nhiêu nhân viên bán hàng trong công ty ? khoảng thời gian giới hạn cho một đáp ứng yêu cầu là bao nhiêu ?
  - người đi phỏng vấn viết một báo cáo (report) cho biết các nội dung chính của buổi phỏng vấn và gửi một bản sao cho khách hàng để hiệu chỉnh
- Phỏng vấn không theo cấu trúc (unstructered interview)
  - đặt các câu hỏi dạng mở (open-ended questions) nhằm khuyến khích khách hàng nói rõ các thông tin

VD: bằng cách hỏi khách hàng tại sao không vừa ý với sản phẩm hiện hiện nay có thể co biết được nhiều khía cạnh trong cách thức kinh doanh của khách hàng

- với người phỏng vấn nhiều kinh nghiệm có thể đặt các câu hỏi mở rộng sau khi đã lắng nghe cẩn thận và dẫn dắt cuộc nói chuyện đi xa hơn, do đó sẽ có nhiều thông tin tốt
- người đi phỏng vấn viêt một báo cáo cho biết các nội dung chính của buổi phỏng vấn và gửi một bàn sao cho khách hàng để hiệu chỉnh

- Gửi bản câu hỏi (send a questionnaire)
  - gửi một bản câu hỏi đến các thành viên liên quan trong cơ quan khách hàng
  - rất hữu dụng vì tập hợp được ý kiến của hàng trăm cá nhân khác nhau
  - các ý kiến phản hồi sẽ được suy nghĩ cẩn thận và xác đáng hơn
  - khuyết điểm: khó mở rộng các câu hỏi và ít thông tin mở rộng
- Khảo sát các biểu bảng (examine the various forms)
  - thường được sử dụng trong môi trường kinh doanh

- khảo sát toàn bộ các biểu bảng được khách hàng sử dụng

VD: khảo sát một mẫu biểu được in trong một cửa hàng,

Có thể phản ánh:

- số trang in
- kích thước khổ giấy
- độ ẩm
- nhiệt độ mực in
- áp lực trên giấy

Các trường khác nhau trang mẫu biểu sẽ chỉ rõ:

- sự chuyển tiếp giữa các tác vụ in
- các giai đoạn tương đối quan trọng

- thấu hiểu các thông tin qua việc quan sát cách thức kinh doanh của khách hàng là cách cực kỳ hữu ích nhằm xác định những cái mà khách hàng cần

- Quay phim video (set up video cameras)
  - là phương pháp mới được sử dụng gần đây
  - đ ược tiến hành tại nơi làm việc nhằm ghi lại chính xác mọi diễn biến
  - nhóm RAT phải có được sự hợp tác của tất cả các thành viên

Chú ý:

- sẽ cực kỳ khó khăn để nắm bát các thông tin cần thiết khi người được quay phim cảm thấy mình:
  - bị xâm phạm đời tư
  - lo sợ
  - bị quấy rầy
- dự kiến trước các rủi ro trước khi giới thiệu máy quay phim vì có thể việc này sẽ gây ra những tức giận cho thành viên được quay phim

- Sử dụng các kịch bản (scenarios)
  - *kịch bản :* là cách thức mà người sử dụng có thể tiến hành trên sản phẩm nhằm hòan thành một số mục tiêu nào đó
    VD: kế hoạch làm giảm cân.
    - chuyên gia dinh dưỡng nhập tuổi, giống, khối lượng và các dữ liệu cá nhân khác của một bệnh nhân béo phì
    - sản phẩm in ra thực đơn cho bệnh nhân
    - kịch bản được đưa ra cho người sử dụng tương lai của sản phẩm
    - chuyên gia dinh dưỡng chỉ ra các điểm không phù hợp cho một bệnh nhân phải sử dụng các thức ăn đặc biệt được chỉ định như

người bị bệnh tiểu đường, người ăn chay hay người bị bệnh đường huyết
  - nhà phát triển cập nhật lại kịch bản trong đó người sử dụng được hỏi về chế độ ăn uống đặc biệt cần có trước khi thực đơn được in
- cho phép người sử dụng tương tác với chính bản thân họ và nhóm RAT sẽ ghi nhận lại các thông tin này
- một số cách mô tả kịch bản:
  - liệt kê các hành động có trong kịch bản
  - tạo một bản tình tiết lưu trữ chuỗi các sự kiện (chẳng hạn như một mẫu giấy trong đó co một chuỗi các biểu bảng, mỗi biểu bảng liên quan đến một màn hình và trả lời của người sử dụng)
- ưu điểm:
  - thể hiện cách đối xử của sản phẩm mà người sử dụng có thể hiểu và cảm nhận được
  - người sử dụng hiểu được kịch bản do đó sẽ đóng vai trò tích cực trong quá trình phân tích yêu cầu
  - nguồn thông tin về cái cần thực sự (real needs) của khách hàng sẽ do chính khách hàng và người sử dụng cung cấp

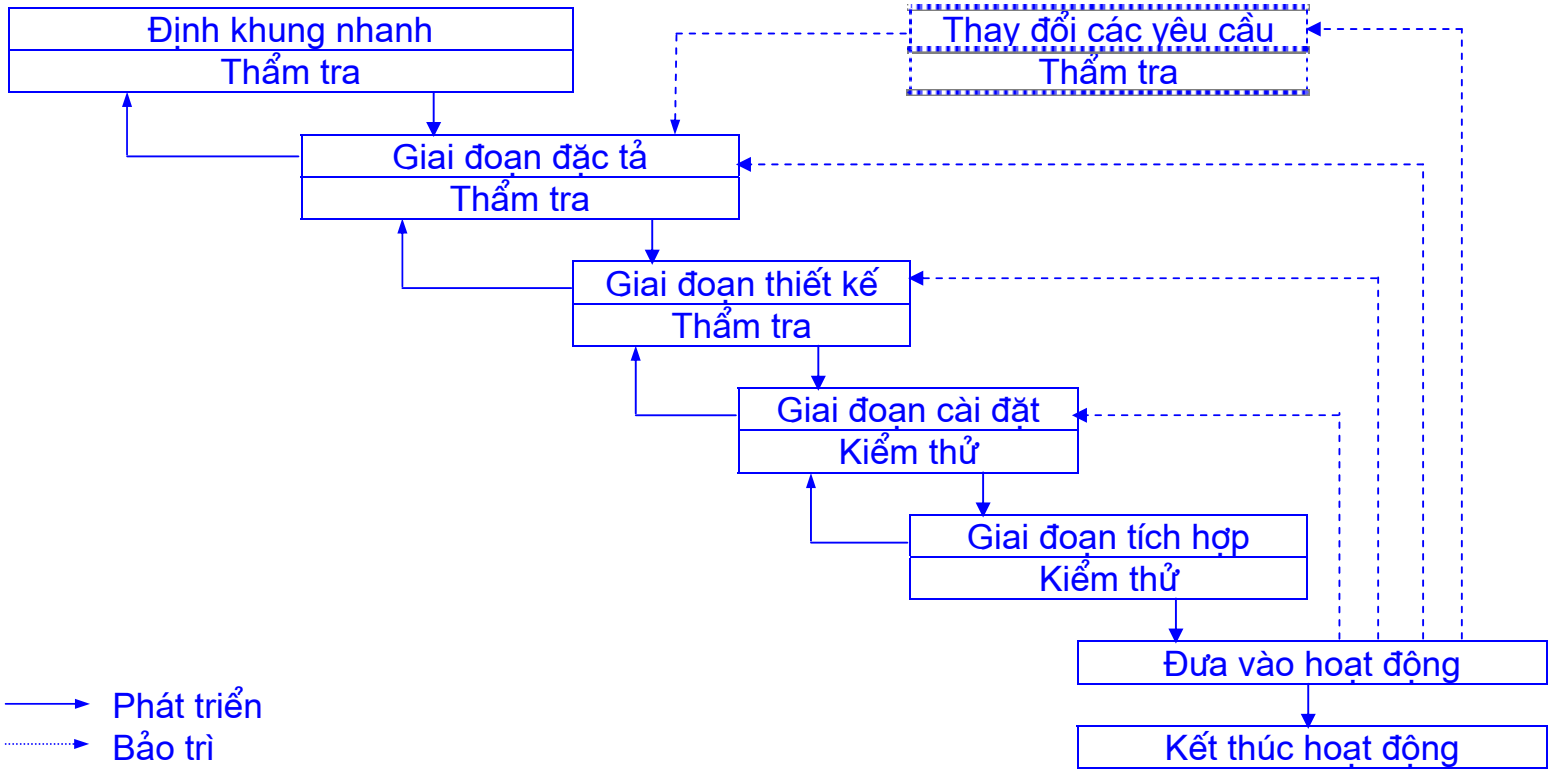
- đóng vai trò quan trọng trong việc phân tích hướng đối tượng (chính là các trường hợp sử dụng - use cases)

- Định khung nhanh (rapid prototyping)
  - xây dựng mô hình càng nhanh càng tốt
  - được xây dựng dành cho các thay đổi
  - phản ánh chức năng mà khách hàng thấy như: các màn hình nhập, các báo cáo,...
  - bỏ qua các khía cạnh như: cập nhật tập tin,...
  - khách hàng, người sử dụng tương lai và nhóm phát triển sẽ cùng nhau xem xét và ghi nhận các dữ kiện
  - các nhà phát triển sẽ liên tục thay đổi mô hình cho đến khi mô hình đã chứa đựng mọi cái cần có
  - quá trình định khung nhanh sẽ được sử dụng cho giai đoạn đặc tả
  - rất hiệu quả khi phát triển giao diện người dùng, hướng đối tượng [Capper, Colgate, Hunter và James, 1994]

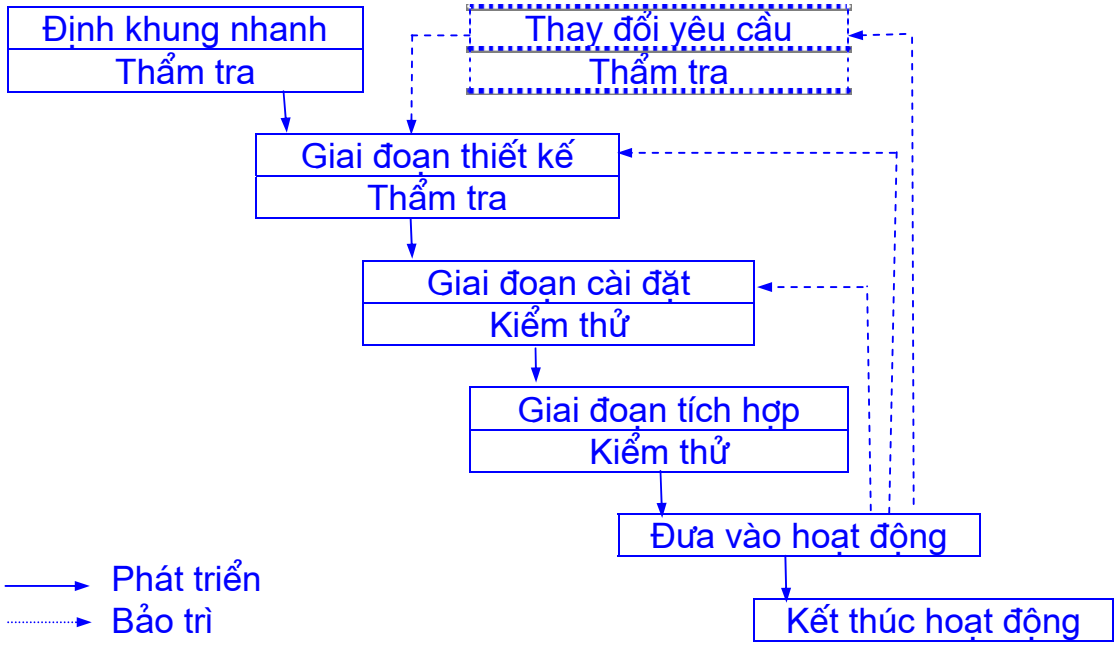
9.4 Nhân tố con người (human factors)

- Điều quan trọng là khách hàng và người sử dụng tương lai tương tác với mô hình định khung nhanh thông qua giao diện người dùng
- Thân thiện với người dùng (user friendliness): dễ dàng giao tiếp với sản phẩm phần mềm
- Sử dụng các nhân tố sau để tăng sự hấp dẫn người dùng
  - đồ họa, cửa sổ, biểu tượng, thực đơn pop-up
  - chỉ và chọn (point and click)
  - quan hệ với nhiều người dùng khác
  - giảm thời gian đào tạo để sử dụng
  - cung cấp nhiều thông tin

9.5 Sử dụng định khung nhanh để đặc tả
(rapid ptototyping as a specification technique)



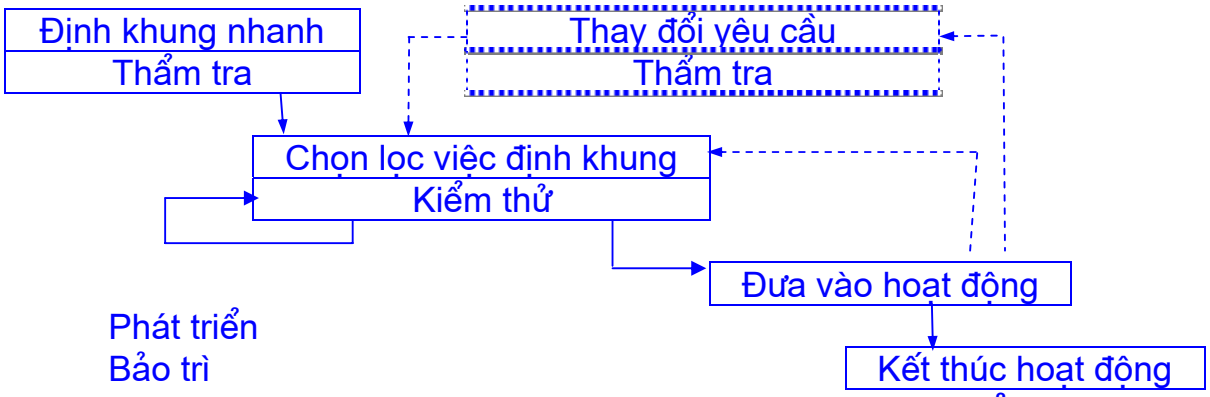
Hình 9.1 Mô hình định khung nhanh như là một kỹ thuật phân tích yêu cầu



Hình 9.2 Định khung nhanh cho đặc tả

- *Tốt nhất là nên sử dụng định khung nhanh như một kỹ thuật để phân tích yêu cầu*

9.6 Sử dụng lại mô hình định khung nhanh (reusing the rapid prototype)



Hình 9.3 Mô hình định khung nhanh với việc phát triển không thận trọng

- Phát triển thật nhanh sản phẩm phần mềm
- Không có đặc tả và thiết kế → khó bảo trì

9.7 Thiết kế ứng dụng chung (joint application design - JAD)

- Là dạng mở rộng của mô hinh định khung nhanh
- Các thành viên trong cơ quan khách hàng sẽ đóng vai trò tích cực hơn
- Các vấn đề cần quan tâm
  - kỹ thuật áp dụng cho các giai đoạn phân tích yêu cầu và đặc tả
  - các nhà phát triển và khách hàng làm việc như một nhóm chung và có trách nhiệm chung đối với kết quả đầu ra
  - nhóm làm việc sẽ thảo luận các yêu cầu cần có, thiết kế các màn hình và báo cáo, xây dựng mô hình định khung nhanh, rút ra các đặc tả
  - chủ yếu dựa trên sự đồng thuận (consensus)

9.8 Kiểm thử trong giai đoạn phân tích yêu cầu
(testing during the requirements phase)

- Do nhóm SQA tiến hành
  - đảm bảo các cá nhân trong cơ quan khách hàng có tương tác với mô hình định khung nhanh
  - phân tích các đề xuất
- Có thể thành lập một hội đồng của khách hàng có trách nhiệm phân tích các đề xuất của khách hàng

9.9 Đánh giá giai đoạn phân tích yêu cầu
(metrics for the requirements phase)

- Tần xuất thay đổi các yêu cầu (phỏng vấn, kịch bản)
- Số lượng các yêu cầu thay đổi trong các giai đoạn còn lại (áp dụng cho toàn bộ các kỹ thuật) → phân tích lại khi có quá nhiều thay đổi
- Số lần mỗi đặc điểm được xây dựng (định khung nhanh)

10 GIAI ĐOẠN ĐẶC TẢ
(SPECIFICATION PHASE)

Nội dung:

- Khái quát chung
- Tài liệu đặc tả
- Đặc tả không hình thức
- Một số kỹ thuật đặc tả bán hình thức
- Một số kỹ thuật đặc tả hình thức
- So sánh các kỹ thuật đặc tả
- Kiểm thử trong giai đoạn đặc tả
- Đánh giá giai đoạn đặc tả

10.1Khái quát chung
(overview)

- Tài liệu đặc tả phải đáp ứng được hai yêu cầu mâu thuẫn nhau
  - rõ ràng và dễ hiểu đối với khách hàng (dễ thuyết phục)
  - đầy đủ và chi tiết vì đây là nguồn thông tin duy nhất dành cho nhóm thiết kế
- Các lỗi xảy ra trong giai đoạn này sẽ ảnh hưởng đến các giai đoạn còn lại của toàn bộ tiến trình

- Các kỹ thuật đặc tả
  - theo cấu trúc
  - hướng đối tượng

10.2Tài liệu đặc tả
(the specification document)

- Là hợp đồng (contract) giữa khách hàng và nhà phát triển
- Phải bao gồm các ràng buộc mà sản phẩm phải đáp ứng
  - thời hạn phân phối sản phẩm cho khách hàng
  - sản phẩm được cài đặt để chạy thử song song với sản phẩm hiện hành cho đến khi khách hàng chấp nhận
  - dễ dàng chuyển đổi trên các phần cứng hay hệ điều hành khác nhau
  - có độ tin cậy cao
  - hoạt động tốt 24/24 giờ (nếu có yêu cầu)
  - thời gian đáp ứng nhanh
    VD: 95% các truy vấn dạng 4 phải được trả lời trong khoảng 0.25s
- Thành phần sống còn là tập các tiêu chuẩn chấp thuận
- Giải pháp chiến lược (solution strategy), là cách tiếp cận chung để tạo ra sản phẩm

10.3Đặc tả không hình thức (informal specifications)

- Sử dụng ngôn ngữ tự nhiên

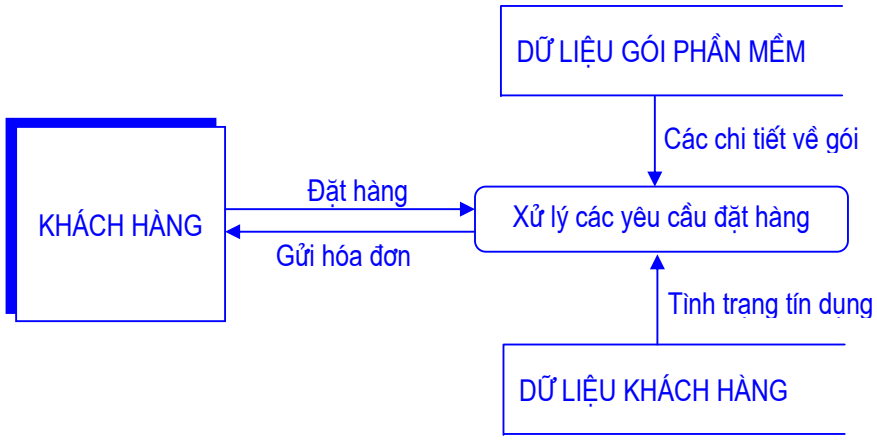
  VD: BV.4.2.5 Nếu doanh thu của tháng hiện tại thấp hơn với doanh thu dự kiến thì một báo cáo sẽ được in ra, trừ phi hiệu số doanh thu giữa doanh thu và doanh thu dự kiến của tháng hiện tại nhỏ hơn phân nửa hiệu số doanh thu tương tự như trên của tháng trước đó hoặc hiệu số doanh thu hiện tại này nhỏ hơn 5%.

- Thường có nhiều lỗi xảy ra
- Ngôn ngữ tự nhiên không phải là phương cách tốt để đặc tả sản phẩm

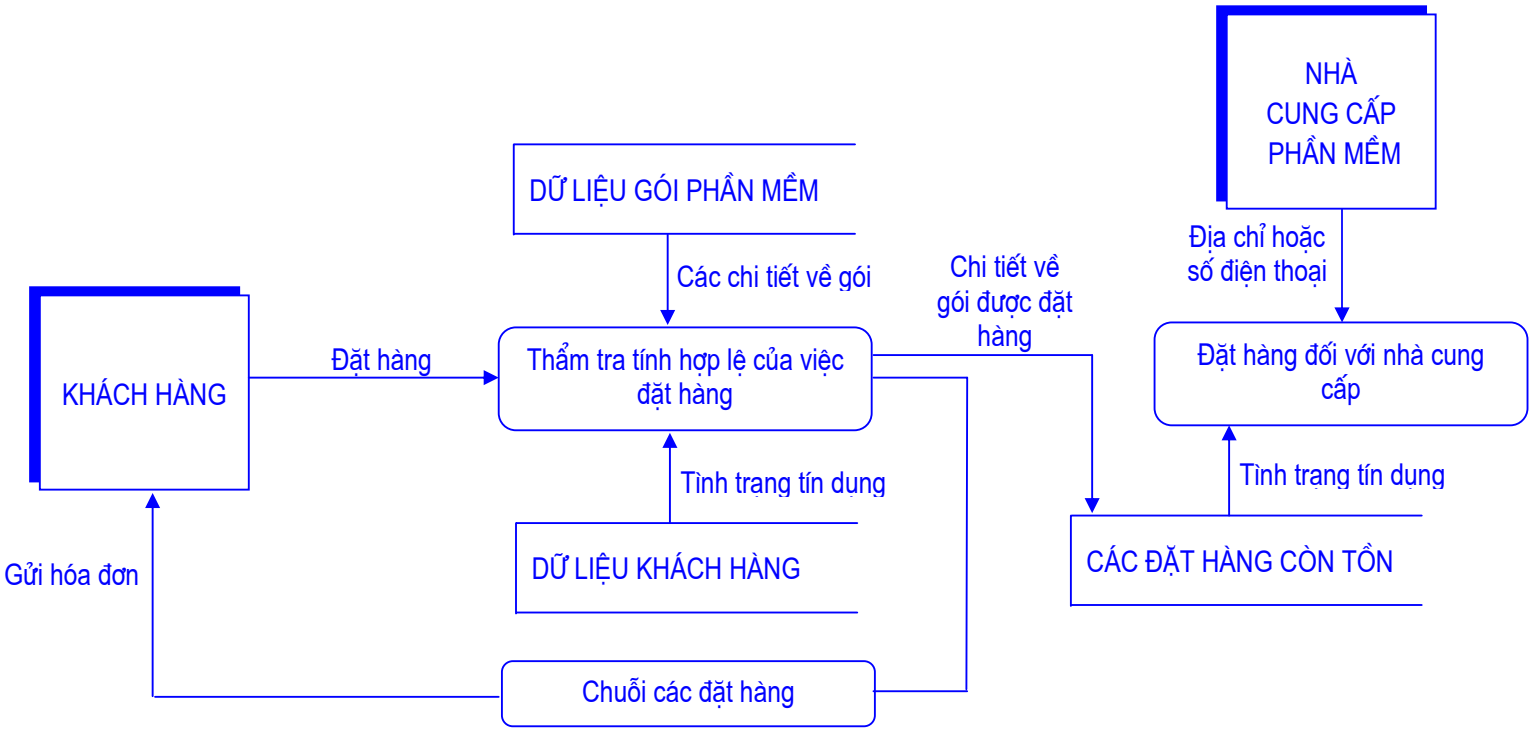
10.4Một số kỹ thuật đặc tả bán hình thức
(the semiformal specification techniques)

- Phân tích theo cấu trúc sử dụng đồ họa và được ứng dụng rộng rãi
  - Gane và Sarsen [Gane và Sarsen, 1979]
  - Yourdon và Constantine [Yourdon và Constantine, 1979]
  - DeMarco [deMarco, 1978],
- PSL/PSA [Teichroew và Hershey, 1977] (problem statement language/problem statement analyzer) là kỹ thuật hỗ trợ dựa trên máy tính
- SADT [Ross, 1985] (structural analysis and design technique)
- SREM <*đọc là shrem*> [Alford, 1985] (software requirements engineering method) dựa trên kỹ thuật máy hữu hạn trạng thái, bao gồm các thành phần sau:
  - RSL: ngôn ngữ đặc tả
  - REVS: tập các công cụ thực hiện các mối liên hệ trong việc đặc tả (chuyển đổi sang cơ sở dữ liệu automate,...)
  - DCDS: kỹ thuật thiết kế
- Mô hình thực thể-quan hệ

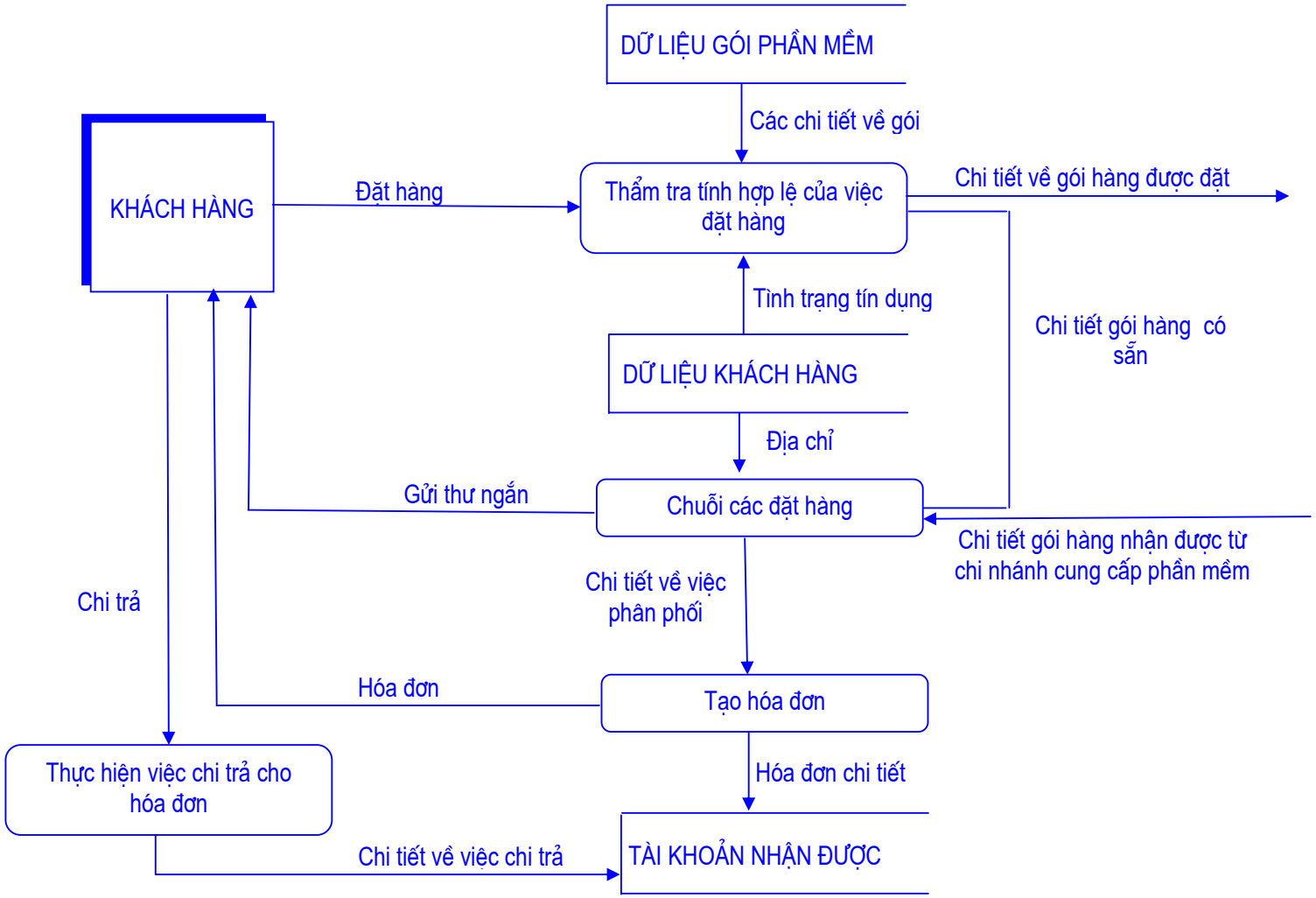
- Ví dụ về cửa hàng mua bán phần mềm (software shop) theo phương pháp Gane và Sarsen
  - *bước 1:* vẽ DFD (data flow diagram)



Hình 10.1 DFD: bước làm mịn thứ nhất

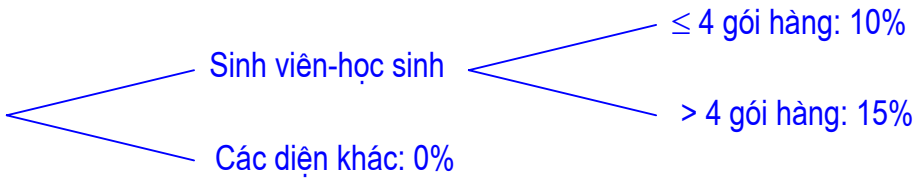


Hình 10.2 DFD: bước làm mịn thứ hai



Hình 10.3 DFD: một phần trong bước làm mịn thứ ba

- *bước 2:* quyết định các phần sẽ được tin học hóa và cách thức tiến hành
- *bước 3:* chi tiết hóa các dòng dữ liệu
  VD: dòng dữ liệu đặt hàng: số, chi tiết về khách hàng, về gói hàng
- *bước 4:* định nghĩa mối quan hệ giữa các tiến trình
  VD: giảm giá cho sinh viên-học sinh

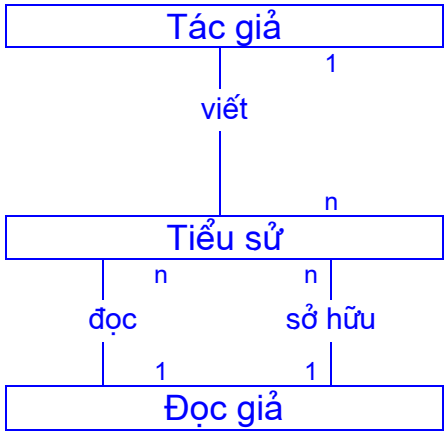


Hình 10.4 Cây quyết định cho cửa hàng phần mềm

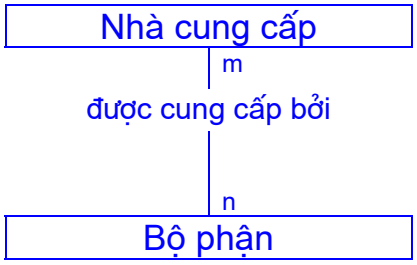
- *bước 5:* định nghĩa kho dữ liệu
- *bước 6:* định nghĩa tài nguyên vật lý
- *bước 7:* xác định các đặc tả đầu vào và đầu ra
- *bước 8:* hoàn thiện kích thước
- *bước 9:* xác định các yêu cầu về phần cứng

10.6Mô hình thực thể-quan hệ
(entity-relationship modeling - ERM)

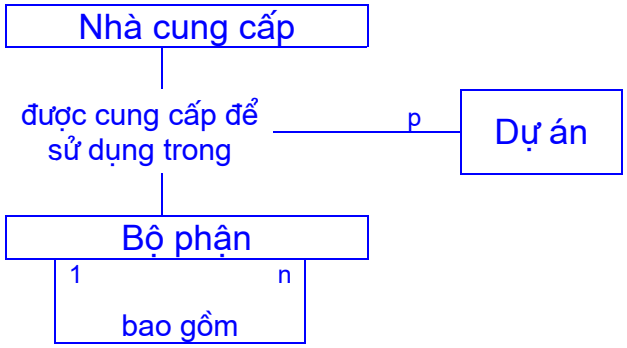
- Là kỹ thuật đặc tả bán hình thức hướng dữ liệu (semiformal data-oriented)
- Được sử dụng rộng rãi trong việc đặc tả cơ sở dữ liệu
- Bao gồm các thực thể và các quan hệ



Hình 10.5 Biểu đồ ER đơn giản



Hình 10.6 Biểu đồ ER nhiều-nhiều



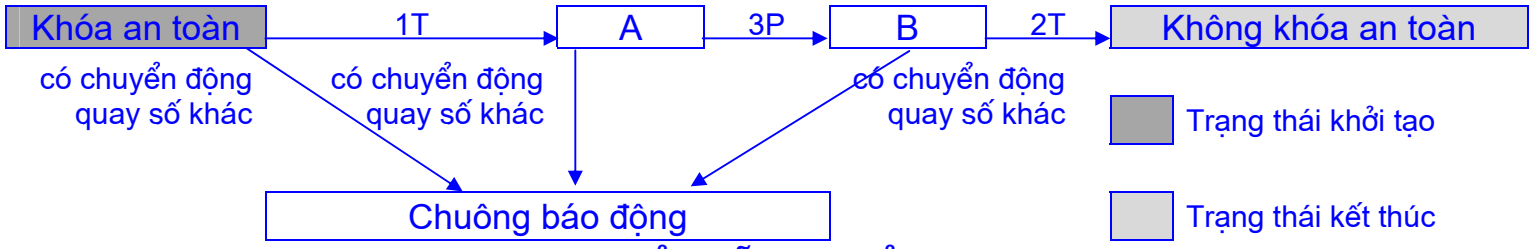
Hình 10.7 Biểu đồ ER phức tạp

10.7Máy hữu hạn trạng thái
(finite state machines - FSM)

- Hữu dụng trong các ứng dụng có các trạng thái (state) và có sự dịch chuyển (transition) giữa các trạng thái
- Thường ứng dụng thực đơn giao diện người dùng được điều khiển
- Một FSM có 5 thành phần
  - tập các trạng thái J
  - tập các đầu vào K
  - tập các dịch chuyển T, xác định các trạng thái chuyển tiếp theo từ trạng thái hiện hành
  - trạng thái bắt đầu S
  - tập các trạng thái kết thúc F

❖ Ví dụ về bộ điều khiển an toàn

J = {Khóa an toàn, A, B, Không khóa an toàn, Chuông báo động }
K = {1T, 1P, 2T, 2P, 3T, 3P}
T = Hình 10.9
S = {Khóa an toàn}
F = {Khóa an toàn, Chuông báo động}



Hình 10.8 FSM biểu diễn các tổ hợp khóa an toàn

Trạng thái hiện hành / *Quay số*	**Bảng các trạng thái tiếp theo**		
	Khóa an toàn	A	B
1T	A	Chuông báo động	Chuông báo động
1P	Chuông báo động	Chuông báo động	Chuông báo động
2T	Chuông báo động	Chuông báo động	Không khóa an toàn
2P	Chuông báo động	Chuông báo động	Chuông báo động
3T	Chuông báo động	Chuông báo động	Chuông báo động
3P	Chuông báo động	B	Chuông báo động

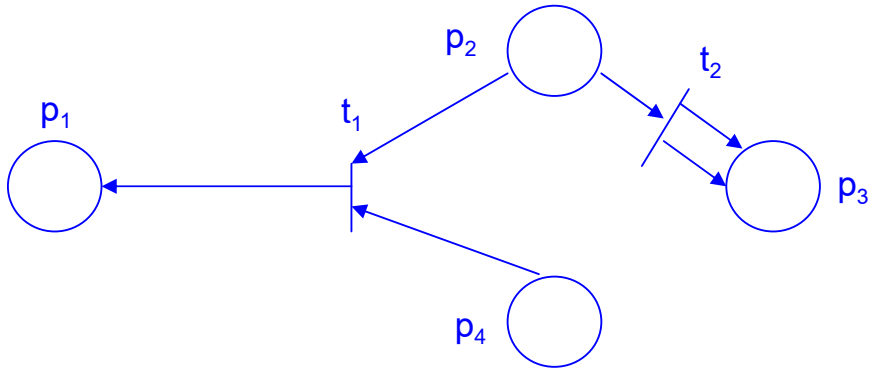
Hình 10.9 Bảng chuyển dịch cho FSM

10.8Một số kỹ thuật đặc tả hình thức
(the formal specification techniques)

- Ana [Luckham và von Henke, 1985], là ngôn ngữ đặc tả cho Ada
- Gist [Balzer, 1985] dùng để mô tả các tiến trình
- VDM [Jones, 1986b; Bj∅rner, 1987] cho ngữ nghĩa
- CSP [Hoare, 1985] biểu diễn các sự kiện và các tiến trình với môi trường làm việc
- Mạng Petri
- Z

- Hướng thời gian do Carl Adam Petri đề xuất [Petri, 1962]
- Có 4 thành phần chính
  - tập các vị trí P
  - tập các phép biến đổi T
  - hàm đầu vào L
  - hàm đầu ra O

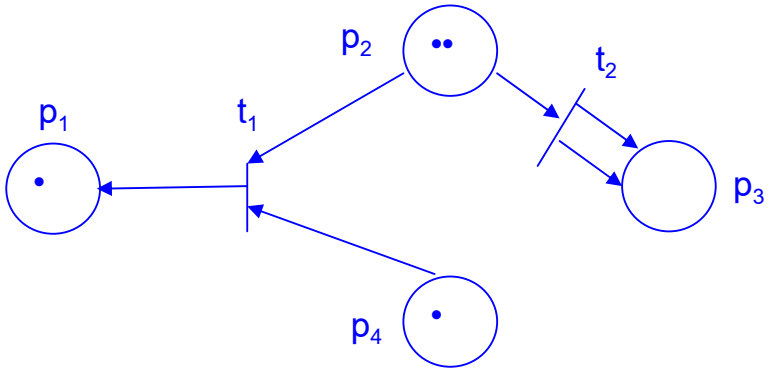
VD: $P = \{p_1, p_2, p_3, p_4\}$
$T = \{t_1, t_2\}$
$I(t_1) = \{p_2, p_4\}$, $I(t_2) = \{p_2\}$
$O(t_1) = \{p_1\}$, $O(t_2) = \{p_3, p_3\}$



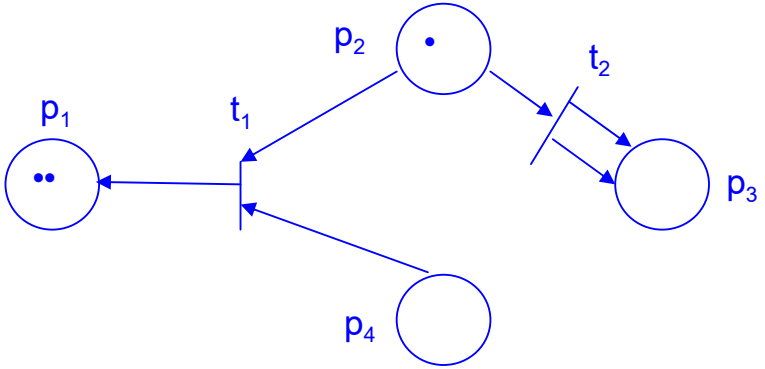
Hình 10.11 Mạng Petri

- Định nghĩa hình thức hơn của Peterson [Peterson, 1981] với một cấu trúc mạng Petri là bộ tứ C=(P,T,I,O)
  - $P = \{p_1, p_2, ..., p_n\}$ là tập hữu hạn các vị trí, $n \geq 0$
  - $T = \{t_1, t_2, ..., t_m\}$ là tập hữu hạn các biến đổi, $m \geq 0$, $P \cap T = \varnothing$
  - $I: T \rightarrow P^{\infty}$ ánh xạ từ các phép biến đổi sang các vị trí
  - $O: T \rightarrow P^{\infty}$ ánh xạ từ các phép biến đổi sang các vị trí

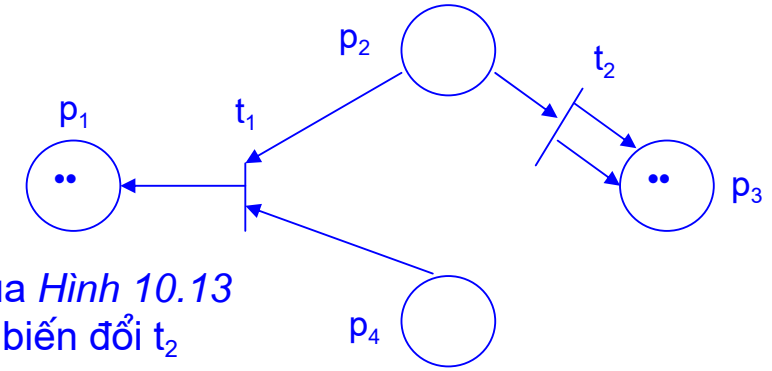
- Mạng Petri có đánh dấu (marking) khi gắn thêm các *tokens*
- Định nghĩa của Peterson [Peterson, 1981] cho đánh dấu
  - thêm M:P→{0,1,2,...} , tập các số nguyên không âm
  - trở thành bộ 5: (P,T,I,O,M)



Hình 10.12 Mạng Petri có đánh dấu

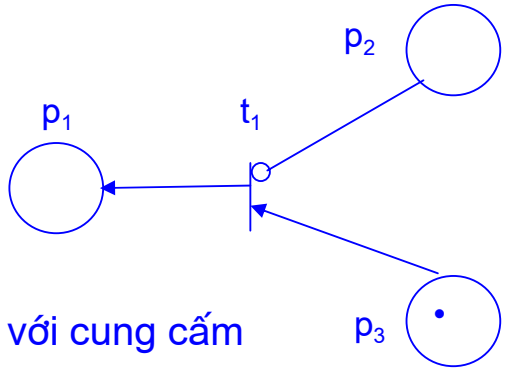


Hình 10.13 Mạng Petri của *Hình 10.12* sau khi bắn sang phép biến đổi t_1



Hình 10.14 Mạng Petri của *Hình 10.13* sau khi bắn sang phép biến đổi t_2

- Cung cấm (inhibitor arc): phép biến đổi có thể thực hiện mà không cần có token trong vị trí nối

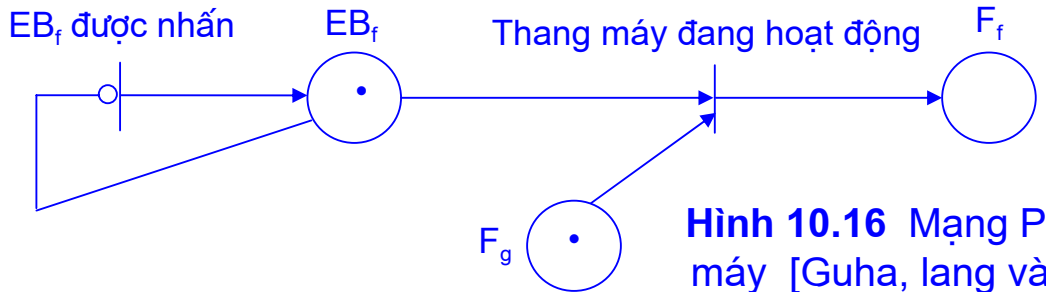


Hình 10.15 Mạng Petri với cung cấm

❖Bài toán thang máy

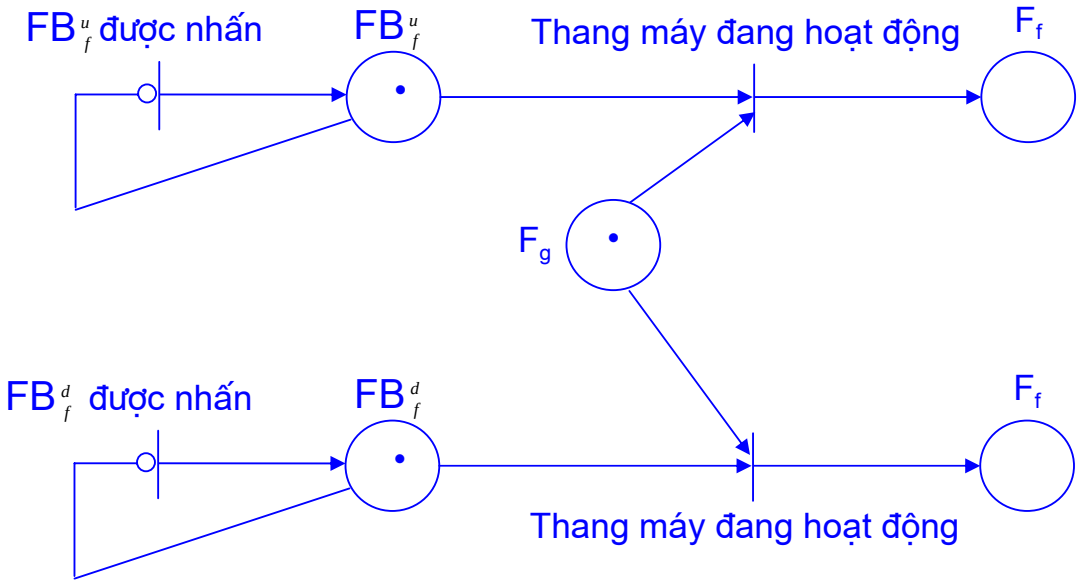
Có n thang máy trong tòa nhà m tầng. Mỗi tầng được xem như một vị trí F_f ($1 \le f \le m$) và mỗi thang máy là một token. Một token trong F_f có nghĩa là thang máy đó đang ở tầng f.

Ràng buộc 1: mỗi thang máy có m nút, nút tương ứng với các tầng sẽ sáng lên khi được nhấn và tắt khi đã đi đến tầng tương ứng. Gọi EB_f ($1 \le f \le m$) là các nút của thang máy tương ứng với tầng f và $EB_{f,e}$ ($1 \le f \le m$, $1 \le e \le n$) là nút f của thang máy e.



Hình 10.16 Mạng Petri biểu diễn một nút trong thang máy [Guha, lang và Bassiouni, 1987]. (©1987 IEEE)

Ràng buộc 2: Mỗi tầng (trừ tầng đầu tiên và cuối cùng) có 2 nút chỉ hướng đi lên hay đi xuống. Các nút này sẽ sáng lên khi được nhấn và tắt khi thang máy đi đến và di chuyển theo hướng mong muốn. FB^u_f và FB^d_f (1< f <m).



Hình 10.17 Mạng Petri biểu diễn các nút chỉ hướng đi [Guha, lang và Bassiouni, 1987]. (©1987 IEEE)

Ràng buộc 3: Một thang máy không có yêu cầu co nghĩa là nó đang dừng tại tầng hiện tại và các cửa của nó ở trạng thái đóng. Khi đó biến đổi *Thang máy đang hoạt động* sẽ bị ngắt.

- *Có thể dùng mạng Petri để đặc tả*

10.10 Z

- Ngôn ngữ đặc tả hình thức nổi tiếng về tính hiệu quả [Spivey, 1992]
- Cơ sở lý thuyết : lý thuyết tập hợp, lý thuyết hàm, toán rời rạc,...
- Một số ký hiệu sử dụng: $\exists, \supset, \Rightarrow, \oplus$,...
- Các bước tiến hành: xác định các tập hợp, kiểu dữ liệu và hằng; định nghĩa trạng thái; xác định trạng thái khởi tạo; các phương thức hoạt động
- Ưu điểm:
  - dễ dàng xác định lỗi đặc tả trong các giai đoạn về sau
  - cực kỳ chính xác; giảm thiểu sự không rõ ràng, mâu thuẫn,... so với đặc tả không hình thức
  - dễ dàng chứng minh tính đúng đắn
  - dễ dàng đào tạo sử dụng
  - giảm chi phí phát triển phầm mềm
  - dễ dàng thuyết phục khách hàng khi viết lại bằng ngôn ngữ tự nhiên
- Sử dụng rộng rãi trong công nghệ phần mềm với các ứng dụng lớn tại các nước Châu Âu và Mỹ

10.11 So sánh các kỹ thuật đặc tả

Phương pháp đặc tả	Thể loại	Điểm mạnh	Điểm yếu
Ngôn ngữ tự nhiên	Không hình thức	Dễ học Dễ sử dụng Dễ hiểu đối với khách hàng	Không chính xác Đặc tả có thể không rõ ràng, mâu thuẫn và /hoặc không đầy đủ
Mô hình thực thể-quan hệ PSL/PSA SADT SREM Phân tích hệ thống theo cấu trúc	Bán hình thức	Khách hàng có thể hiểu được Chính xác hơn các phương pháp không hình thức	Không chính xác như các phương pháp hình thức Nhìn chung thì khó định lượng thời gian
Anna CSP Máy hữu hạn trạng thái mở rộng Gist Mạng Petri VDM Z	Hình thức	Cực kỳ chính xác Có thể giảm các lỗi đặc tả Có thể giảm chi phí và nhân lực Có thể hỗ trợ việc chứng minh tính chính xác	Khó học Khó sử dụng Khách hàng hầu như không thể hiểu được

10.12 Kiểm thử trong giai đoạn đặc tả
(testing during the specification phase)

- Đánh giá sự chính xác của tài liệu đặc tả
- Thanh tra (inspection) [Fagan, 1976] cho giai đoạn thiết kế và viết mã lệnh
  VD:nhóm thanh tra (team of inspectors) sẽ đối chiếu lại tài liệu đặc tả với một danh sách kiểm tra (checklist)
- Một dữ kiện điển hình trong danh sách thanh tra đặc tả bao gồm: đã chỉ rõ các tài nguyên phần cứng?, đã chỉ rõ các tiêu chuẩn chấp thuận ?

10.13 Đánh giá công việc đặc tả
(metrics for the specification phase)

- Đánh giá 5 đại lượng cơ bản: kích thước (số trang tài liệu đặc tả, kích thước sản phẩm đích,...), giá thành, thời gian, nhân lực, chất lượng (thống kê lỗi)
- Sử dụng từ điển dữ liệu (data dictionary)

11

GIAI ĐOẠN PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG

(OBJECT-ORIENTED ANALYSIS PHASE)

Nội dung:

- Khái quát chung
- Mô hình các trường hợp sử dụng
- Mô hình lớp
- Mô hình động

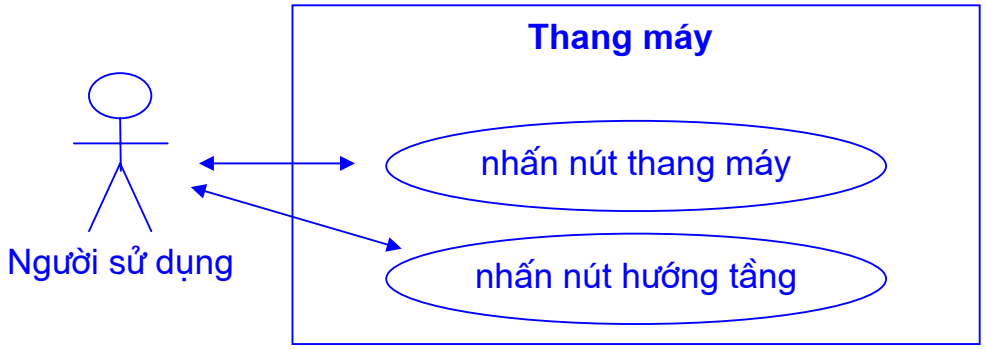
10.1Khái quát chung
(overview)

- Tốt hơn đặc tả (phân tích) cấu trúc
- Là kỹ thuật đặc tả bán hình thức trong các kỹ thuật hướng đối tượng
- UML (unified modeling language)

- Có 3 bước chính
  - mô hình các trường hợp sử dụng
  - mô hình lớp
  - mô hình động

10.2Mô hình các trường hợp sử dụng
(use-case modeling)

- Xác định các chức năng của sản phẩm, cung cấp cái nhìn trên tổng thể các chức năng cho nên còn được gọi là mô hình chức năng
- Thể hiện sự tương tác giữa các lớp trong sản phẩm với người sử dụng
- Trình bày dưới 2 dạng
  - sơ đồ trường hợp sử dụng
  - các kịch bản kết hợp
- Kịch bản là thể hiện của trường hợp sử dụng



Hình 11.1 Các trường hợp sử dụng trong bài toán thang máy

1. Người A nhấn nút UP tại tầng 3 để yêu cầu thang máy. Người A muốn đi đến tầng 7.
2. Nút UP sáng lên.
3. Một thang máy đến tầng 3. Trong thang máy này có người B vào thang máy từ tầng 1 và đi đến tầng 9.
4. Nút UP tắt.
5. Thang máy mở cửa, người A vào thang máy.
6. Người A nhấn nút đi tầng 7.
7. Nút tầng 7 sáng lên.
8. Cửa thang máy đóng.
9. Thang máy đi đến tầng 7.
10. Nút tầng 7 tắt.
11. Cửa thang máy mở cho phép người A đi ra khỏi thang máy.
12. Bộ định thời hoạt động. người A đi ra ngoài.
13. Cửa thang máy đóng lại sau khi đã hết thời gian.
14. Thang máy tiếp tục đi đến tầng 9 với người B.

Hình 11.2 Một kịch bản thông thường

1. Người A nhấn nút UP tại tầng 3 để yêu cầu thang máy. Người A muốn đi đến tầng 1.
2. Nút UP sáng lên.
3. Một thang máy đến tầng 3. Trong thang máy này có người B vào thang máy từ tầng 1 và đi đến tầng 9.
4. Nút UP tắt.
5. Thang máy mở cửa, người A vào thang máy.
6. Người A nhấn nút đi tầng 1.
7. Nút tầng 1 sáng lên.
8. Cửa thang máy đóng.
9. Thang máy đi đến tầng 9.
10. Nút tầng 9 tắt.
11. Cửa thang máy mở cho phép người B đi ra khỏi thang máy.
12. Bộ định thời hoạt động. người B đi ra ngoài.
13. Cửa thang máy đóng lại sau khi đã hết thời gian.
14. Thang máy tiếp tục đi đến tầng 1 với người A.

Hình 11.3 Một kịch bản không thông thường

10.3Mô hình lớp
(class modeling)

- Tách ra các lớp và các thuộc tính và thể hiện bằng sơ đồ thực thể-quan hệ

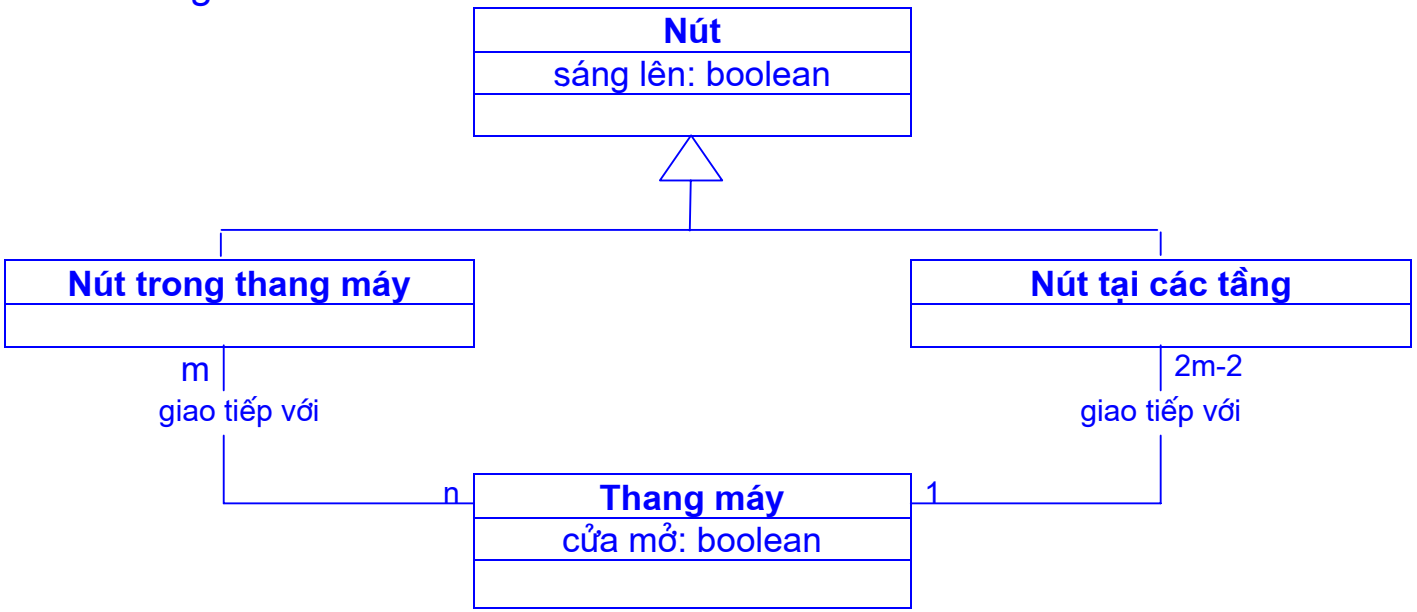
- Tách tên (noun extraction), nhằm chỉ ra các lớp có thể có với 3 bước sau:
  - định nghĩa súc tích vấn đề: định nghĩa sản phẩm trên những nét chính nhất với những câu đơn.

    VD: Các nút trong các thang máy và tại các tầng điều khiển sự di chuyển *n* thang máy của tòa nhà *m* tầng.
  - chiến lược không hình thức: lấy ra các ràng buộc một cách không hình thức để giải quyết vấn đề

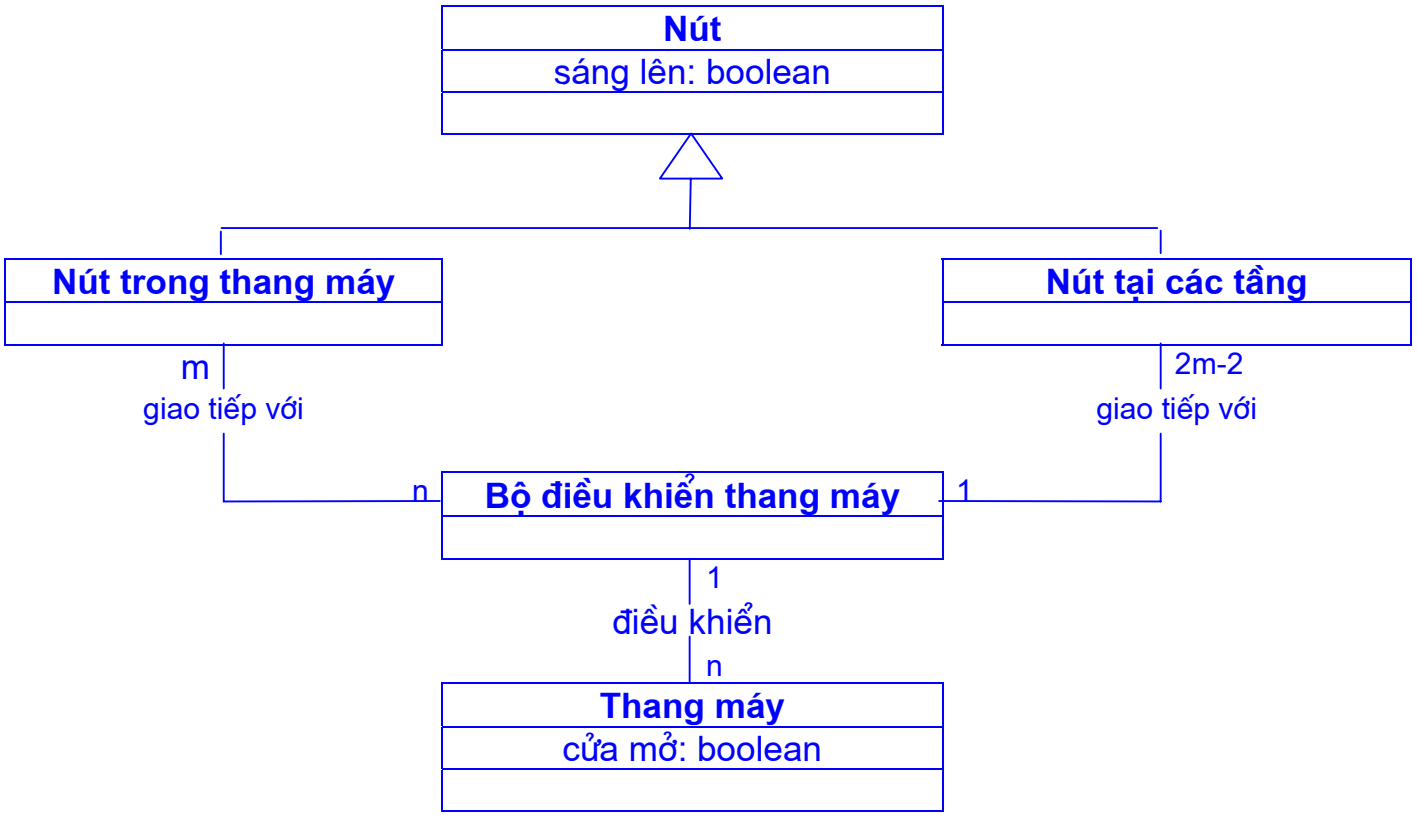
    VD: Các nút dùng trong các thang máy và tại các tầng điều khiển sự di chuyển *n* thang máy của tòa nhà *m* tầng. Các nút sẽ sáng lên khi được nhấn với yêu cầu một thang máy dừng tại tầng xác định nào đó và ánh đèn tắt đi khi yêu cầu đã được đáp ứng. Khi một thang máy không có yêu cầu nào thì dừng tại tầng hiện hành với các cửa đóng.

- hình thức hóa chiến lược: xác định các tên trong chiến lược không hình thức (kể các các nội dung nằm ngoài biên vấn đề) và sử dụng các tên như là các ứng cử viên cho các tên lớp

  VD: **Các nút** dùng trong **các thang máy** và tại **các tầng** điều khiển **sự di chuyển** *n* **thang máy** của tòa nhà *m* tầng. **Các nút** sẽ sáng lên khi được nhấn với yêu cầu một **thang máy** dừng tại tầng xác định nào đó và **ánh đèn** tắt đi khi **yêu cầu** đã được đáp ứng. Khi một **thang máy** không có **yêu cầu** nào thì dừng tại **tầng** hiện hành với các **cửa** đóng.



Hình 11.4 Sơ đồ lớp đầu tiên



Hình 11.5 Sơ đồ lớp thứ hai

- Khung lớp trách nhiệm và hợp tác (class-responsibility-collaboration CRC) do nhóm phát triển điền vào:
  - tên lớp
  - các chức năng (trách nhiệm)
  - danh sách các lớp gọi các chức năng này (hợp tác)

  Phân tích:
  - *ưu điểm:*
    - được thực hiện bởi một nhóm làm việc do đó giảm thiểu các thiết sót hay các mục không chính xác trong lớp;
    - dễ dàng xác định sơ đồ lớp có đầy đủ và chính xác hay chưa;
    - rất tốt trong việc xác định các chức năng và khả năng hợp tác giữa các lớp.
  - *nhược điểm:* không phải là cách tìm lớp tốt nếu như nhóm làm việc chưa có kinh nghiệm trên các lĩnh vực liên quan;

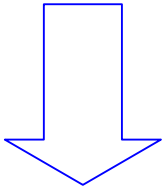
LỚP **Bộ điều khiển thang máy**
TRÁCH NHIỆM 1. Bật nút trong thang máy 2. Tắt nút trong thang máy 3. Bật nút tại tầng 4. Tắt nút tại tầng 5. Mở cửa thang máy 6. Đóng cửa thang máy 7. Di chuyển thang máy lên một tầng 8. Di chuyển thang máy xuống một tầng
HỢP TÁC 1. Lớp **Nút trong thang máy** 2. Lớp **Nút tại các tầng** 3. Lớp **Thang máy**

Hình 11.6 CRC đầu tiên của lớp **Bộ điều khiển thang máy**

10.4Mô hình động
(dynamic modeling)

- Xây dựng sơ đồ trạng thái, là việc mô tả sản phẩm dưới dạng mô hình trạng thái hữu hạn
  - trạng thái
  - sự kiện
  - vị từ
- Thực hiện trên từng lớp
- Tương tự FSM

Hình 11.7 Sơ đồ trạng thái cho lớp **Bộ điều khiển thang máy**



12

GIAI ĐOẠN THIẾT KẾ
(DESIGN PHASE)

Nội dung:

- Khái quát chung
- Thiết kế và trừu tượng hóa
- Thiết kế hướng sự kiện
- Thiết kế hướng dữ liệu
- Thiết kế hướng đối tượng
- Kiểm thử
- Đánh giá

12.1Khái quát chung
(overview)

- Hàng trăm kỹ thuật thiết kế đã ra đời trong hơn 30 năm qua
- Thiết kế hướng sự kiện (action-oriented)
  - phân rã sản phẩm thành các mô-đun có tính chặt chẽ cao và ít gắn kết với nhau
- Thiết kế hướng dữ liệu (data-oriented)
  - phụ thuộc vào cấu trúc dữ liệu mà các xử lý được thực hiện trên đó
  - các kỹ thuật nổi tiếng như [Jackson, 1975;1983], [Wanier, 1976;1981] và [Orr, 1981]
- Thiết kế hướng đối tượng
  - dạng tổng hợp, bao gồm cả sự kiện và dữ liệu

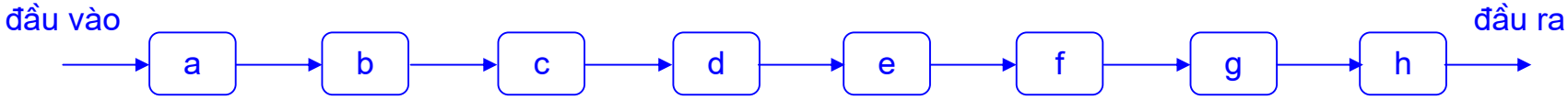
- Đầu vào: tài liệu đặc tả, cho biết sản phẩm phải làm gì *(what ?)*
- Đầu ra: để đạt được những công việc đã mô tả ở đầu vào, sản phẩm phải thực hiện như thế nào *(how ?)*

12.2Thiết kế và trừu tượng hóa
(design and abstraction)

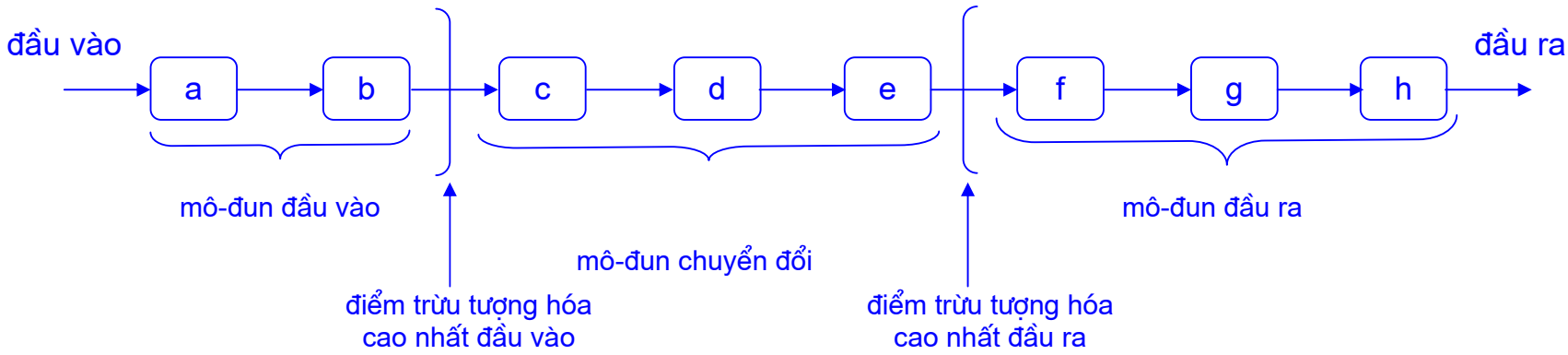
- Giai đoạn thiết kế phần mềm có 3 hoạt động chính: kiến trúc, chi tiết và kiểm thử
- Thiết kế kiến trúc *(architectural design, general design, logical design, high-level design)*: theo quan điểm trừu tượng hóa là phân chia sản phẩm thành các mô-đun
- Thiết kế chi tiết *(detailed design, modular design, physical design, low-level design)*: chi tiết hóa từng mô-đun
  - chọn giải thuật
  - chọn cấu trúc dữ liệu
- Kiểm thử thiết kế *(design testing)*

12.3Phân tích dòng dữ liệu
(data flow analysis - DFA)

- Thiết kế hướng sự kiện, tạo ra các mô-đun với tính chặt chẽ cao
  - đầu vào: sơ đồ dòng dữ liệu (data flow diagram - DFD)
  - sau khi hoàn thành DFD, nhà thiết kế phần mềm phải hoàn tất các thông tin vào/ra của từng module
- Điểm trừu tượng hóa cao nhất đầu vào *(point of highest abstraction of input)*: điểm biến chuyển dữ liệu đầu vào thành dữ liệu nội tại
- Điểm trừu tượng hóa cao nhất đầu ra *(point of highest abstraction of output)*: điểm biến chuyển dữ liệu nội tại thành đầu ra

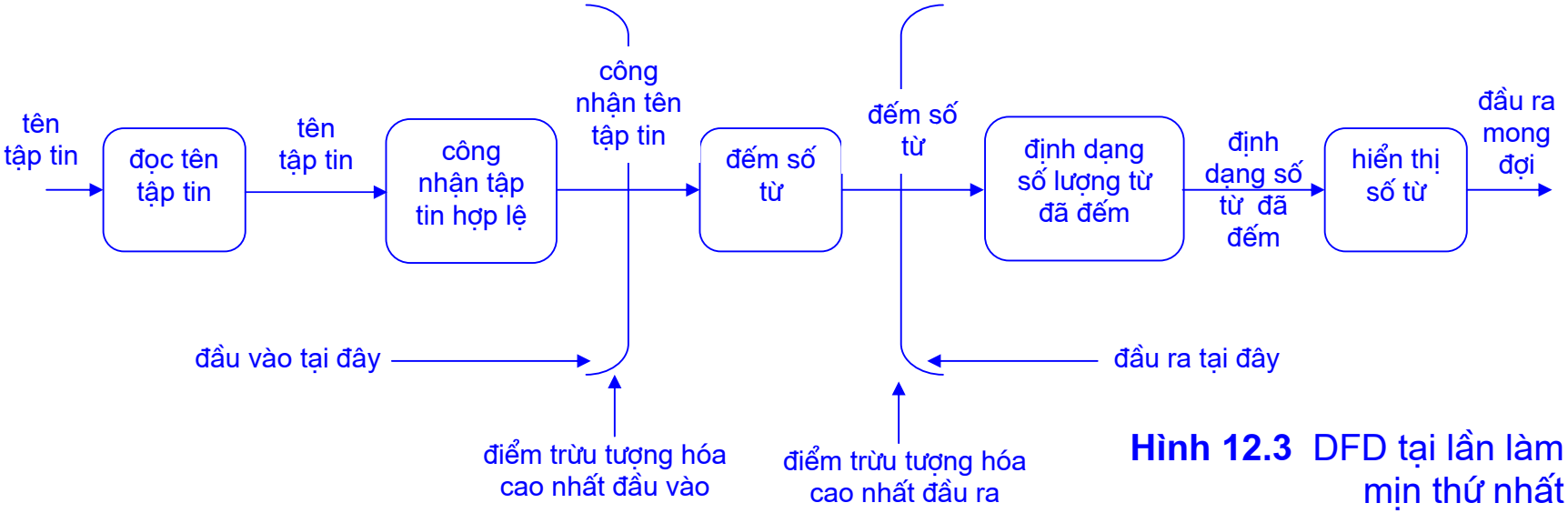


Hình 12.1 Thể hiện dữ liệu và sự kiện của sản phẩm bằng DFD

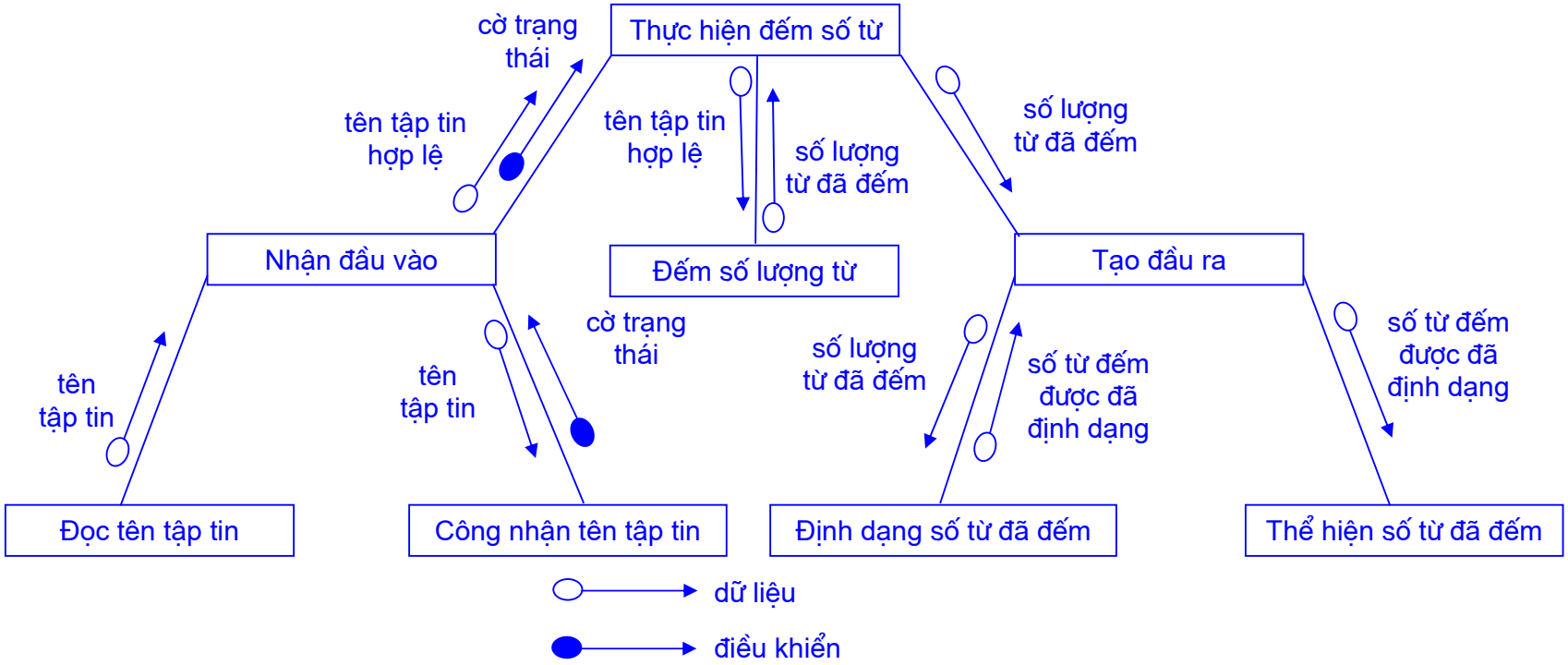


Hình 12.2 Các điểm trừu tượng hóa cao nhất đầu vào và đầu ra

❖Ví dụ về DFA : đếm số từ (words) trong tập tin



Hình 12.3 DFD tại lần làm mịn thứ nhất



Hình 12.4 Biểu đồ cấu trúc

❖Thiết kế chi tiết 4 mô-đun

Tên mô-đun	**Đọc tập tin**
Kiểu trả về	String
Các tham số đầu vào	không
Các tham số đầu ra	không
Các thông báo lỗi	không
Các tập tin truy xuất	không
Các tập tin có thay đổi trên đó	không
Các mô-đun được gọi	không
Mô tả	Sản phẩm được thi hành khi người dùng gõ lệnh: word count <tên tập tin> Sử dụng một lời gọi hệ thống, mô-đun này sẽ truy xuất nội dung chuỗi lệnh do người sử dụng nhập vào, tách ra tên tập tin và trả về kết quả là tên tập tin đã tách được.

Tên mô-đun	**Công nhận tên tập tin hợp lệ**
Kiểu trả về	boolean
Các tham số đầu vào	tên tập tin: String
Các tham số đầu ra	không
Các thông báo lỗi	không
Các tập tin truy xuất	không
Các tập tin có thay đổi trên đó	không
Các mô-đun được gọi	không
Mô tả	Mô-đun này tạo một lời gọi hệ thống để xác định sự tồn tại của tập tin với tham số đầu vào là tên tập tin. Mô-đun trả về kết quả *true* nếu tập tin đã tồn tại và *false* nếu ngược lại.

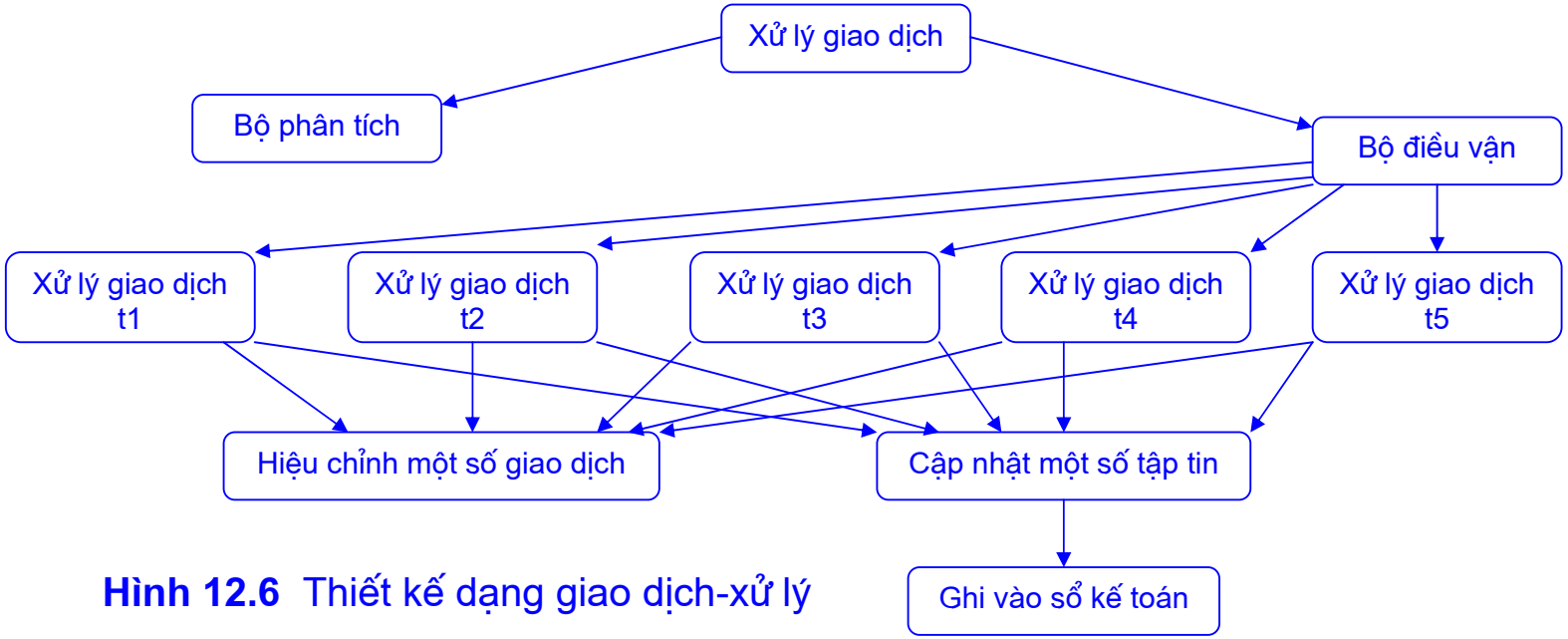
Tên mô-đun	**Đếm số lượng từ**
Kiểu trả về	integer
Các tham số đầu vào	tên tập tin hợp lệ: String
Các tham số đầu ra	không
Các thông báo lỗi	không
Các tập tin truy xuất	không
Các tập tin có thay đổi trên đó	không
Các mô-đun được gọi	không
Mô tả	Mô-đun này xác định với tên tập tin hợp lệ đầu vào là tập tin văn bản. Khi đó mô-đun sẽ trả về số từ có trong tập tin văn bản , ngược lại trả về -1.

Tên mô-đun	**Tạo đầu ra**
Kiểu trả về	void
Các tham số đầu vào	số lượng từ: integer
Các tham số đầu ra	không
Các thông báo lỗi	không
Các tập tin truy xuất	không
Các tập tin có thay đổi trên đó	không
Các mô-đun được gọi	**Định dạng số từ** các tham số: số từ:integer, số từ đã được định dạng:String **Thể hiện số từ đã đếm** các tham số: số từ đã được định dạng:integer: String
Mô tả	Mô-đun này lấy tham số đầu vào là số từ đã đếm được bằng cách gọi mô-đun **Định dạng số từ** và sau đó gọi mô-đun **Thể hiện số từ đã đếm** để thể hiện số từ đã đếm được.

Hình 12.5 Thiết kế chi tiết các mô-đun

12.4Phân tích giao dịch
(transaction analysis - TA)

- Thiết kế hướng sự kiện
- Một giao dịch là một thao tác theo quan điểm của người sử dụng sản phẩm
  VD: xử lý một yêu cầu, in ra danh sách các đặt hàng trong ngày

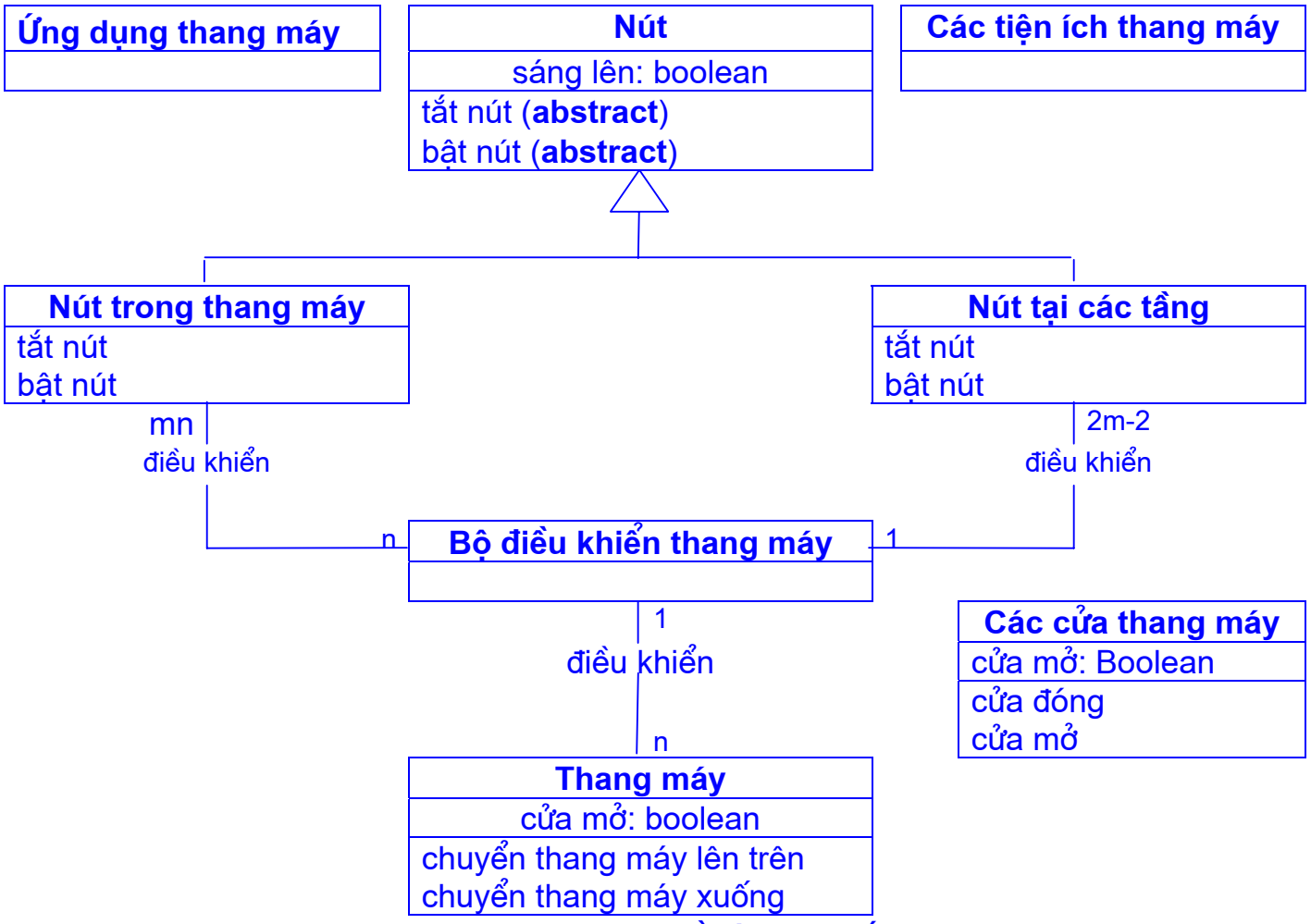


Hình 12.6 Thiết kế dạng giao dịch-xử lý

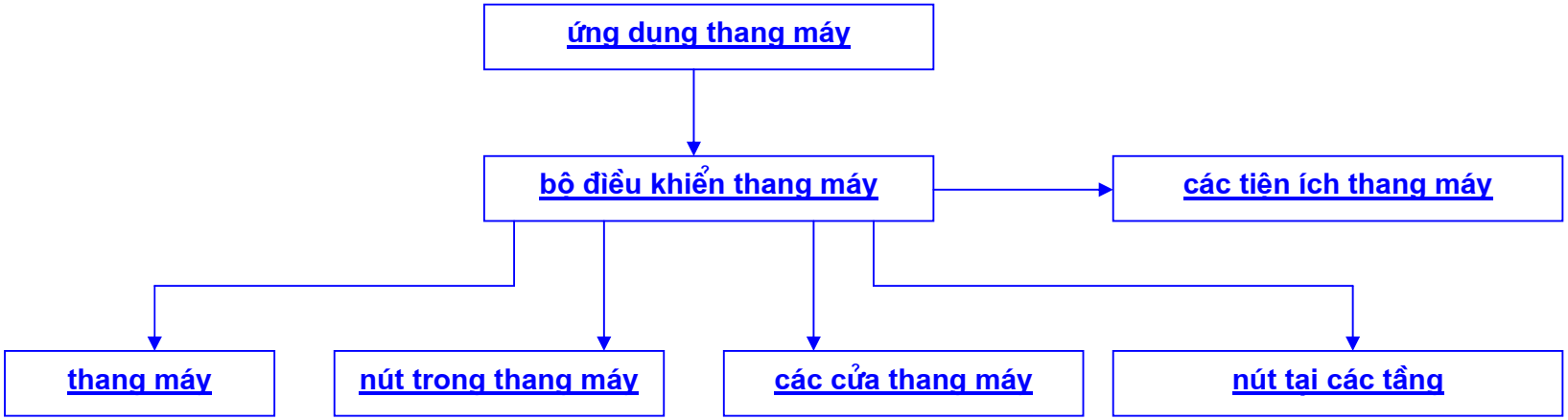
12.5Thiết kế hướng đối tượng
(object-oriented design - OOD)

- Thiết kế sản phẩm thành các đối tượng(object) là các thể hiện của các lớp (classe) hay các lớp con (subclass)
- Các ngôn ngữ lập trình hướng đối tượng thông dụng như Smalltalk [Goldberg và Robson, 1989], C++ [Stroustrup, 1991], Eiffel [Meyer, 1992b], Ada95 [ISO/IEC 8652, 1995] và Java [Flanagan, 1996]
- Khi cài đặt trên các ngôn ngữ lập trình không hướng đối tượng tiến hành thiết kế trên các kiểu dữ liệu trừu tượng (abstract data type design)

- Bao gồm 4 bước:
  - xây dựng sơ đồ tương tác cho từng kịch bản
  - xây dựng sơ đồ lớp chi tiết
  - thiết kế sản phẩm theo các đối tượng của khách hàng
  - tiến hành thiết kế chi tiết



Hình 12.7 Sơ đồ lớp chi tiết



Hình 12.8 Quan hệ khách hàng-đối tượng

12.6Kiểm thử trong giai đoạn thiết kế
(testing during the design phase)

- Tìm thấy lỗi trong giai đoạn này là rất quan trọng
- Có thể sử dụng
  - walkthroughs
  - thanh tra tương tự như trong giai đoạn đặc tả nhưng có thể không có đại diện của khách hàng
- Phải phản ánh được hướng thiết kế

12.7Đánh giá giai đoạn thiết kế
(metrics for the design phase)

- Có nhiều phương pháp đánh giá trên các mặt của giai đoạn thiết kế
  - số lượng các mô-đun: đánh giá thô về kích thước của sản phẩm
  - độ gắn kết của mô-đun: đánh giá về chất lượng

- độ nối kết giữa các mô-đun: thống kê về lỗi

- Độ phức tạp của thiết kế chi tiết *M* bằng số lượng quyết định nhị phân cộng với 1 [McCabe, 1976] (hay số lượng nhánh trong một mô-đun)
  VD: độ phức tạp khi viết một mô-đun *toascii*:
  - có sử dụng switch có 128 nhánh : 128
  - sử dụng bảng chuyển đổi trực tiếp: 1
- [Henry và Kafura, 1981] $M = length \times (fan\text{-}in \times fan\text{-}out)^2$
  - length : kích thước mô-đun
  - fan-in : số lượng các luồng đi vào mô-đun + số lượng cấu trúc dữ liệu mà mô-đun truy xuất
  - fan-out : số lượng các luồng đi ra mô-đun + số lượng các cấu trúc dữ liệu toàn cục mà mô-đun cập nhật
- Phương pháp thành công nhất cho thiết kế hướng đối tượng: CDM [Kitchenham, Pickard và Linkman, 1990; Shepperd, 1990]
- Một số phương pháp khác : [Chidamber và Kemerer, 1994], [Binkley and Schach, 1996;1997;1998]

13

GIAI ĐOẠN CÀI ĐẶT

(IMPLEMENTATION PHASE)

Nội dung:

- Khái quát chung
- Kỹ năng lập trình tốt
- Viết mã lệnh chuẩn
- Lựa chọn trường hợp kiểm thử mô-đun
- Các phương pháp tạo dữ liệu kiểm thử
- Kỹ thuật Cleanroom

13.1Khái quát chung
(overview)

- Quá trình chuyển đổi từ thiết kế chi tiết sang mã lệnh
- Do nhiều người thực hiện (programming-in-the-many)
- Lựa chọn ngôn ngữ lập trình
  - phụ thuộc vào hợp ngữ của máy tính
  - phụ thuộc vào số lượng ngôn ngữ lập trình sẵn có
  - thói quen sử dụng ngôn ngữ lập trình
- Các ngôn ngữ lập trình thế hệ thứ tư (fourth generation languages – 4GL): Focus, Nature,...
  - mã máy (1)
  - hợp ngữ (2)
  - ngôn ngữ mức cao (3) : FORTRAN, ALGOL 60, COBOL,...

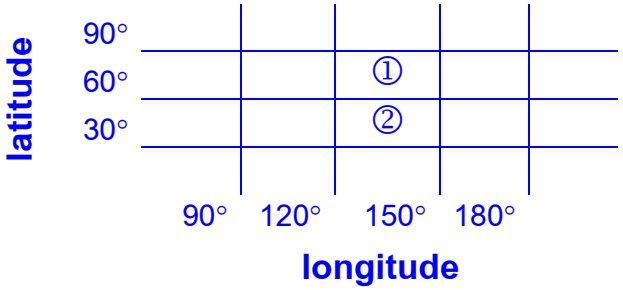
  $\Rightarrow$ Mục tiêu là sản phẩm sẽ do chính người lập trình sử dụng (end-user programming)
- Có đánh giá rủi ro khi chọn ngôn ngữ lập trình

13.2Kỹ năng lập trình tốt
(good programming practice)

- Hiểu rõ ngôn ngữ *(language-specific)*
- Sử dụng tên biến thích hợp và có nghĩa *(use of consistent and meaningful variable names*)
  - có nghĩa theo quan điểm của các nhà lập trình bảo trì
  - chú ý đến ngôn ngữ mẹ đẻ của các lập trình viên, thống nhất ngôn ngữ để đặt tên biến (tiếng Anh,...)
  - tên biến phải rõ ràng và không gây lầm lẫn
  - dễ dàng hiểu các mã lệnh
- Chú thích tự thân *(the issue of self-documenting code)*
  - không có các dòng chú thích
  - các tên biến phải được diễn giải ngay từ đầu *(prologue comments)*
- Nên có các chú thích bên trong mô-đun *(inline comments)*
- Sử dụng tham số *(use of parameters)*
- Dễ đọc *(code layout for increased readability)*, sử dụng các cặp dấu ngoặc, canh đầu dòng, các dòng trắng để định rõ các công việc,...

- Thông tin tối thiểu của một mô-đun *(the minimum information)*
  - tên mô-đun
  - mô tả vắn tắt các công việc mô-đun phải thực hiện
  - tên của lập trình viên
  - ngày viết mô-đun
  - ngày mô-đun được chấp thuận và được chấp thuận bởi ai
  - các tham số
  - danh sách các tên biến (nên theo thứ tự chữ cái) và cách sử dụng
  - tên các tập tin mà mô-đun có truy xuất
  - tên các tập tin bị thay đổi bởi mô-đun (nếu có)
  - nhập/xuất của mô-đun (nếu có)
  - các khả năng lỗi xảy ra
  - tên tập tin sẽ được sử dụng để kiểm thử
  - danh sách các cập nhật đã được tiến hành với ngày tương ứng, người chấp thuận
  - các lỗi đã biết (nếu có)

- Các lệnh *if* lồng nhau *(nested* if *statement)*



Hình 13.1 Các tọa độ trên bản đồ

```
if (latitude>30 && longitude>120)
{
  if (latitude<=60 && longitude<=150)
        mapSquareNo = 1;
  else if (latitude<=90 && longitude<=150)
        mapSquareNo = 2;
  else
        System.out.println("Not on the map");
}
else
        System.out.println("Not on the map");
```

Hình 13.2 Định dạng tốt nhưng nhiều *if* lồng nhau

```
if (latitude>30 && longitude>120) { if (latitude<=60 && longitude<=150) mapSquareNo = 1;   else if (latitude<=90 &&
longitude<=150) mapSquareNo = 2;    else System.out.println("Not on the map");} else System.out.println("Not on the
map");
```

Hình 13.3 Định dạng xấu và nhiều *if* lồng nhau

```
if (longitude>120 && longitude<=150  && latitude>30 && latitude<=60)
        mapSquareNo = 1;
  else if (longitude>120 && longitude<=150  && latitude>60 && latitude<=90)
        mapSquareNo = 2;
  else
        System.out.println("Not on the map");
```

Hình 13.4 Các câu *if* chấp nhận được

13.3Viết mã lệnh chuẩn
(coding standards)

- Thống nhất quy ước về cách đặt tên mô-đun, tên biến,...
- Nên sử dụng các quy tắc sau:
  - độ lồng nhau của lệnh *if* tối đa là 3
  - mỗi mô-đun có khoảng 35 đến 50 mã lệnh thực thi
  - không sử dụng lệnh *goto*, có thể sử dụng để bắt lỗi
- Chịu sự kiểm thử của nhóm SQA

- Có khả năng sử dụng lại (reuse)
  - một số phần trong đặc tả, hợp đồng, kế hoạch, thiết kế, các mô-đun
  - một số thiết bị phần cứng liên quan

13.4Lựa chọn trường hợp kiểm thử mô-đun (module test case selection)

- Một mô-đun phải chịu hai lần kiểm thử
  - không hình thức (informal testing), do lập trình viên tiến hành
  - theo phương pháp (methodical testing), do nhóm SQA thực hiện sau khi khi lập trình viên xác nhận rằng mô-đun đã vận hành tốt:
    - → không dựa trên việc thực thi (nonexecution-based testing)
    - → dựa trên việc thực thi (execution-based testing)
- Cách kiểm thử dở nhất là sử dụng dữ liệu kiểm thử bừa bãi, khi đó sẽ không có đủ thời gian để thực hiện
- Các kiểm thử tốt nhất là xây dựng các trường hợp kiểm thử có hệ thống, các bộ dữ liệu kiểm thử được tạo ra có chọn lọc

13.5Các phương pháp tạo dữ liệu kiểm thử
(constructing test data to test a module)

- Kiểm thử dựa trên đặc tả, không chú ý đến mã lệnh
  - *các tên gọi khác:* hộp đen (black-box), cấu trúc (structural), dữ liệu dẫn (data-driven), chức năng (functional), xuất/nhập dẫn (input/output driven)
  - VD: 5 dạng hoa hồng và 7 dạng khấu hao, số trường hợp kiểm thử ít nhất sẽ là 35
- Kiểm thử dựa trên mã lệnh, không chú ý đến đặc tả; mọi phân nhánh trong mô-đun phải được thực thi ít nhất một lần
  - *các tên gọi khác*: hộp kính (glass-box), hộp trắng (white-box), hành vi (behavioral), logic dẫn (logic-driven), định hướng đường đi (path-oriented)

13.6Kỹ thuật kiểm thử dạng hộp đen
(black-box module-testing techniques)

- Kiểm thử tương đương và phân tích giá trị biên (equivalence testing and boundary value analysis)
  - lớp tương đương
  - phân tích giá trị biên trong khoảng (R_1,R_2) sẽ có 5 trường hợp kiểm thử: $<R_1$, $=R_1$, $>R_1$và $<R_2$, $=R_2$, $>R_2$
- Kiểm thử chức năng (functional testing)
  - dựa trên dữ liệu theo từng chức năng

```
<hàm mức cao> ::== if <biểu thức điều kiện>
                      <hàm mức thấp 1>;
                   else
                      <hàm mức thấp 2>;
```

⇒ *<biểu thức điều kiện>*, *<hàm mức thấp 1>*, *<hàm mức thấp 2>* còn *<hàm mức cao>* sẽ kiểm thử dạng hộp kính(ở phần tiếp theo)

13.7Kỹ thuật kiểm thử dạng hộp kính
(glass-box module-testing techniques)

- Kiểm thử cấu trúc lệnh, phân nhánh và đường đi (statement, branch, and path coverage)
  - lệnh: các chuỗi dữ liệu thử phải đảm bảo mỗi lệnh được thực hiện ít nhất một lần. VD:

```
if (s > 1 && t == 0)
    x = 9;

Trường hợp kiểm thử: s = 2, t = 0.
```

  - phân nhánh: các chuỗi dữ liệu thử phải đảm bảo mỗi nhánh được thực hiện ít nhất một lần

    ⇒ *Còn gọi kiểm thử cấu trúc với hai dạng trên*
  - đường đi: hiệu quả nhất, kiểm thử tất cả các hướng đi có thể
    - sử dụng phương pháp định nghĩa toàn bộ các đường đi có sử dụng (all-definition-use-path coverage) nhằm giảm thiểu số lượng đường đi phải kiểm thử [Rapps và Weyuker, 1985]
    - ứng với mỗi đường đi tạo một bộ dữ liệu kiểm thử

- Đo độ phức tạp (complexity metrics)
  - số lượng dòng lệnh [Basili và Hutchens, 1983; Takahashi và Kamayachi, 1985]
  - số lượng các quyết định nhị phân + 1 [McCabe, 1976]
  - độ đo Halstead
    - n_1: số lượng các toán tử khác nhau
    - n_2: số lượng các toán hạng khác nhau
    - N_1: tổng số các toán tử
    - N_2: tổng số các toán hạng. VD:

```
if (k < 2)
{
    if (k > 3)
        x = x*k;
}

các toán tử khác nhau: if ( < ) { > = * ; }
các toán hạng khác nhau: k 2 3 x
n1 = 10, n2 = 4, N1 = 13, N2 = 7
```

  - kích thước dữ liệu: O, Ω

13.8Kỹ thuật Cleanroom

- Đề xuất bởi [Cobb và Mills, 1990; Dyer, 1992; Linger, 1994], tổ hợp một số kỹ thuật phát triển phần mềm khác nhau
  - mô hình tăng trưởng
  - các kỹ thuật đặc tả và thiết kế hình thức
  - kỹ thuật kiểm thử mô-đun không dựa trên thực thi: đọc mã lệnh, walkthroughs và thanh tra

Một số ứng dụng:

- Ericsson Telecom OS32 với 350000 dòng lệnh do 70 người thực hiện (1.0 lỗi /KLOC),...
- 17 sản phẩm với 1 triệu dòng lệnh (2.3 lỗi/KLOC) [Linger, 1994]

14 GIAI ĐOẠN CÀI ĐẶT VÀ TÍCH HỢP

(IMPLEMENTATION AND INTEGRATION PHASE)

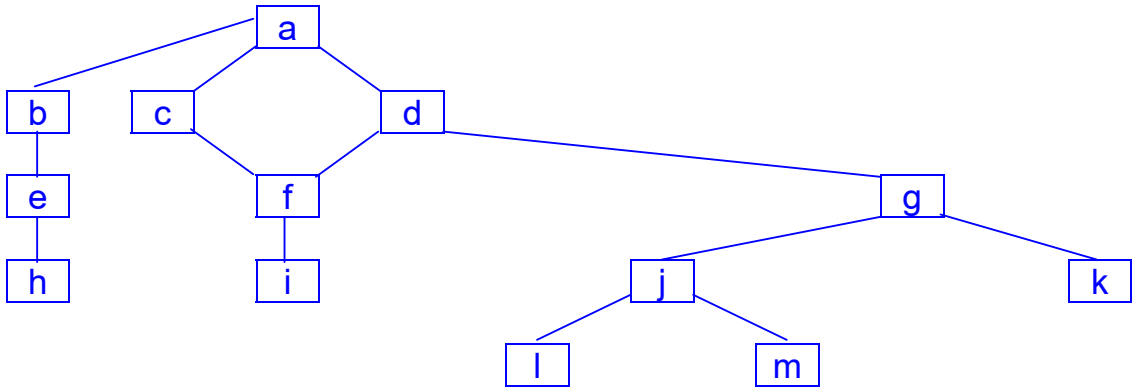
Nội dung:

- Khái quát chung
- Cài đặt và tích hợp
- Cài đặt và tích hợp dạng trên xuống
- Cài đặt và tích hợp dạng dưới lên
- Cài đặt và tích hợp dạng hỗn hợp

14.1Khái quát chung
(overview)

- *Sẽ không tốt* nếu như việc cài đặt được thực hiện riêng rẽ và sau đó tích hợp lại toàn bộ và được kiểm thử chung
- Sẽ tốt hơn nếu như việc cài đặt và tích hợp được thực hiện song song

- Xét ví dụ sau:



Hình 14.1 Sơ đồ điển hình về sự liên kết giữa các mô-đun

⇒ Rất khó khăn khi kiểm thử mô-đun *a* riêng biệt trước khi kiểm thử các mô-đun *b*, *c* và *d* !

14.3Cài đặt và tích hợp dạng trên xuống

(top-down implementation and integration)

- Nếu mô-đun mA gọi mô-đun mB thì mA được cài đặt và tích hợp trước mB
  VD: *(Hình 14.1)* , các thứ tự có thể là:
  `a,b,c,d,e,f,g,h,i,j,k,l` hay `a,b,e,h,c,d,f,i,g,j,k,l`
- Khi mô-đun mNew được thêm vào, thì lỗi xảy ra chỉ có thể tại các vị trí:
  - mô-đun mNew
  - giao diện (≥1) giữa mNew và phần còn lại hiện có trong sản phẩm
- Có thể chia thành 2 dạng
  - mô-đun logic *(logic module)*: tổ hợp các dòng điều khiển quyết định trong sản phẩm. VD: *(Hình 14.1)* `a,b,c,d` và có thể là `g,j`
  - mô-đun hoạt động *(operational module)*: hoạt động thật sự của sản phẩm. VD: *(Hình 14.1)* `e,f,h,i,k,l,m`

  ⇒ Các mô-đun hoạt động phải được cài đặt trước các mô-đun logic
- Khó khăn khi sử dụng lại các mô-đun
- Lập trình bảo vệ*(defensive programming)*: kiểm tra an toàn khi gọi mô-đun

14.4Cài đặt và tích hợp dạng dưới lên

(bottom-up implementation and integration)

- Nếu mô-đun mA gọi mô-đun mB thì mB được cài đặt và tích hợp trước mA

  VD: *(Hình 14.1)*

  - thứ tự duy nhất là: `l,m,h,i,j,k,e,f,g,b,c,d,a`
  - nên phân chia các mô-đun như sau:

    người 1: `h,e,b`

    người 2: `i,f,c`

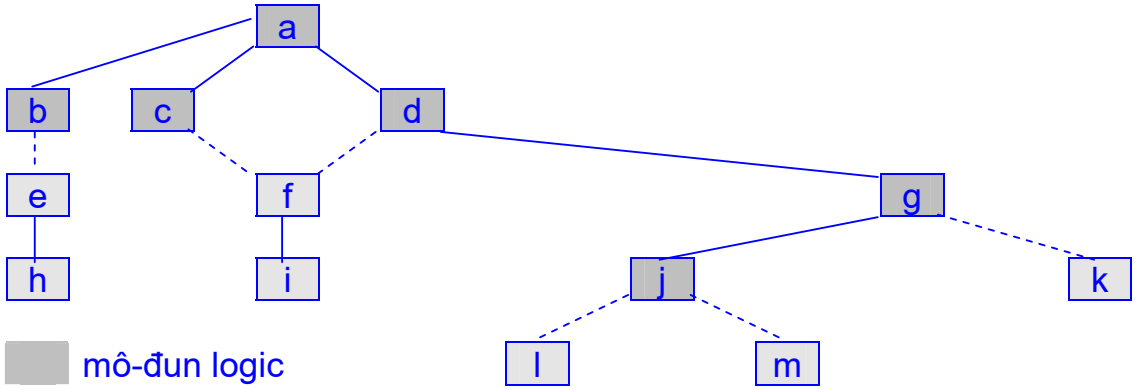
    người 3: `l,m,j,k,g,d`

    tích hợp 3&2, sau khi tích hợp `b,c,d` thì cài đặt và tích hợp `a`
- Nếu gặp lỗi ở các mô-đun logic thì sẽ khó khăn khi lần vết sửa đổi trở lại trên các mô-đun đã thực hiện cài đặt và tích hợp

14.5Cài đặt và tích hợp dạng hỗn hợp

(sandwich implementation and integration)

- Xét ví dụ sau:



Hình 14.2 Cài đặt và tích hợp dạng hỗn hợp

Cài đặt:

- trên xuống: `a,b,c,d,g,j`
- dưới lên : `e,f,h,i,k,l,m`

14.6Tổng kết các phương pháp tiếp cận

(summary of implementation and integration approaches)

Cách tiếp cận	Điểm mạnh	Điểm yếu
Cài đặt trước, tích hợp sau	-	Không cô lập được lỗi Chậm phát hiện lỗi thiết kế chính
Cài đặt và tích hợp trên xuống	Cô lập được lỗi Sớm phát hiện lỗi thiết kế chính	Các mô-đun có khả năng sử dụng lại không được kiểm thử đầy đủ
Cài đặt và tích hợp dưới lên	Cô lập được lỗi Các mô-đun có khả năng sử dụng lại được kiểm thử đầy đủ	Chậm phát hiện lỗi thiết kế chính
Cài đặt và tích hợp hỗn hợp	Cô lập được lỗi Sớm phát hiện lỗi thiết kế chính Các mô-đun có khả năng sử dụng lại được kiểm thử đầy đủ	

Hình 14.3 Tổng kết các phương pháp tiếp cận

14.7Cài đặt và tích hợp các sản phẩm hướng đối tượng

(implementation and integration of object-oriented products)

- Cài đặt và tích hợp dạng trên xuống: tương tự như tiếp cận cổ điển
- Cài đặt và tích hợp dạng dưới lên: các đối tượng gửi thông báo đến những đối tượng đã được cài đặt và tích hợp sau khi bản thân đã được cài đặt và tích hợp
- Cài đặt và tích hợp hỗn hợp

14.8 Kiểm thử

- Kiểm thử tích hợp giao diện GUI, khó khăn khi kiểm thử người dùng sử dụng chuột, kích hoạt thực đơn,...
  ⇒ *Dùng các ngôn ngữ scripts, công cụ kiểm thử đặc biệt*

- Kiểm thử sản phẩm, do nhóm SQA thực hiện theo trình tự sau:
  - hộp đen
  - từng mô-đun (hoặc từng đối tượng)
  - dạng thô trên toàn sản phẩm
  - đặc biệt *(stress)*: khi login,...
  - khối lượng *(volume)*: với nhiều tập tin đầu vào
  - xem xét lại toàn bộ các tài liệu sẽ trao cho khách hàng (thỏa mãn SPMP)
  - đối chiếu giữa tài liệu và sản phẩm

  ⇒ *Có thể dùng kịch bản cho các phần mềm dạng hướng đối tượng*

- Chấp nhận kiểm thử (acceptance testing)
  - thực hiện giữa khách hàng và nhóm SQA
  - phải thực hiện trên dữ liệu thực tế chứ không đơn thuần là dữ liệu dùng để kiểm thử
  - nếu sản phẩm mới thay thế sản phẩm đã có sẵn, trong tài liệu đặc tả phải có một điều khoản cho phép sản phẩm mới được cài đặt song song với sản phẩm cũ cho đến khi khách hàng chấp thuận

- Kết thúc giai đoạn kiểm thử thì công việc của các nhà phát triển xem như kết thúc và sản phẩm chuyển sang giai đoạn bảo trì

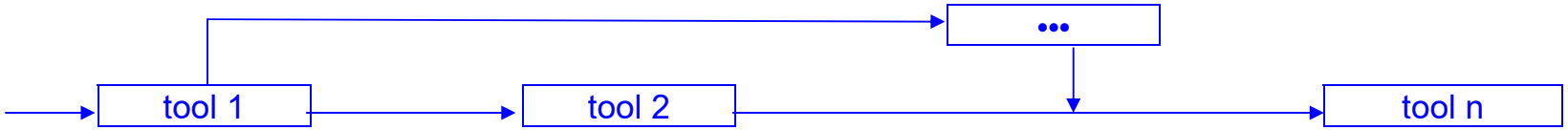
14.9Tích hợp môi trường

(integrated environments)

- Nhằm mục tiêu tất cả các công cụ trong cùng một môi trường có cùng giao diện người dùng (user interface integration)
  ⇒ *Nhìn thấy và cảm nhận được (look and feel)*

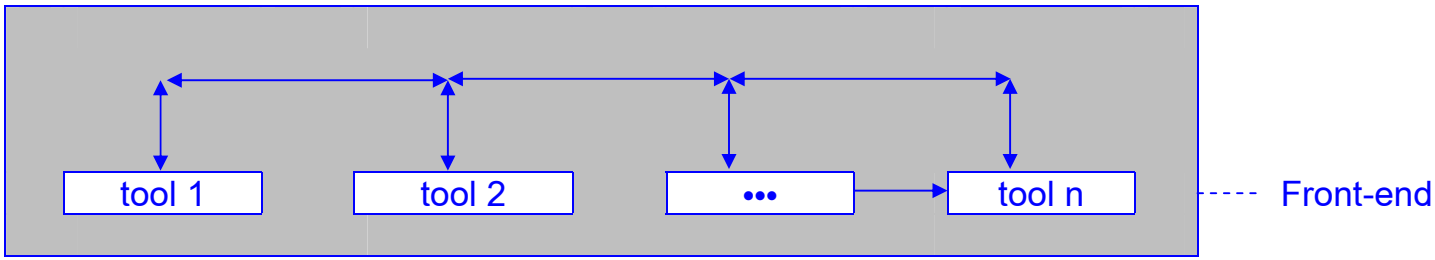
- Tích hợp các tiến trình (process integration)
  - môi trường thường hỗ trợ cho một tiến trình phát triển phần mềm đặc biệt nào đó
  - còn gọi là môi trường kỹ thuật nền (technique-based enviroment)
  - một số môi trường thương mại: *Analyst/Designer* theo phương pháp Yourdon [Yourdon, 1989], Statemate [Hrel và al., 1990], Rose [Booch, 1994],...

- Tích hợp công cụ (tool integration), tất cả các công cụ giao tiếp với nhau thông qua các định dạng dữ liệu giống nhau. VD: theo dạng mã ASCII
  - tích hợp công cụ dòng dữ liệu (data stream tool integration)
    VD: các dòng dữ liệu nhập/xuất đều dưới dạng mã ASCII



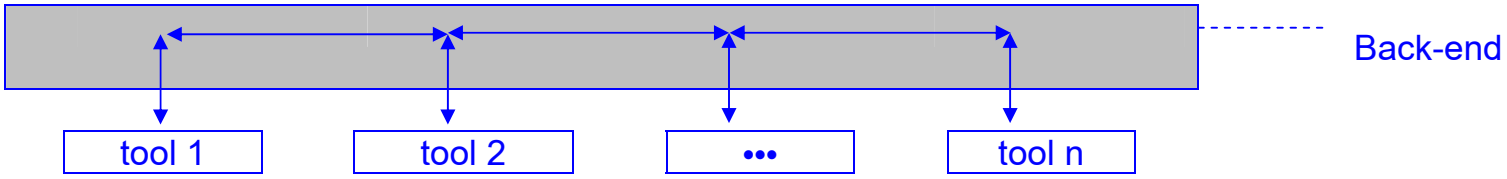
Hình 14.4 (a) Tích hợp công cụ dòng dữ liệu

  - tích hợp front-end (front-end tool inegration), các công cụ được nhúng. Môi trường thương mại: SoftBench [Riehle, 1991] dành cho sản xuất phần cứng, CT dành cho sản xuất phần mềm



Hình 14.4 (b) Tích hợp công cụ front-end

- tích hợp back-end (back-end tool inegration), sử dụng cơ sở dữ liệu chung. Môi trường thương mại: AD/Cycle [Mercurio, Meyers, Nisbet và Radin, 1990]



Hình 14.4 (c) Tích hợp công cụ back-end

- Các khuôn dạng tích hợp khác (other forms of integration)
  - tích hợp nhóm *(team integration)*, tạo hiệu quả và cách giao tiếp trong nhóm phát triển
  - tích hợp quản lý *(management integration)*, hỗ trợ cho quản lý

14.10 Đánh giá giai đoạn cài đặt và tích hợp

(metrics for the implementation and integration phase)

- Có nhiều phương pháp đo độ phức tạp tại giai đoạn này
- Công thức [Brettschneider, 1989] xác định thời gian kiểm thử

$$\frac{\ln\left(\frac{f_{target}}{0.5 + f_{target}}\right)}{\ln\left(\frac{0.5 + f_{target}}{f_{total} + f_{target}}\right)} \times t_h$$

Trong đó: (*lỗi – failure)*

- f_{target} : số lượng lỗi dự đoán
- f_{total} : số lỗi thực sự xảy ra sau đó
- t_h : thời gian kiểm thử xảy ra lỗi
- ln : lô-ga-rít theo cơ số *e*

VD: Giả sử sản phẩm có 50000 LOC, hợp đồng qui định mỗi KDSI có ít hơn 0.02 lỗi. Sản phẩm được kiểm thử 400 giờ, trong thời gian này có 20 lỗi xảy ra và đã thực thi 50 giờ kể từ lỗi cuối cùng.

Ta có: $f_{target} = 0.02 \times 50000 = 1$, $f_{total} = 20$, $t_h = 400 - 50 = 350$ giờ

kết quả là 54 giờ (thiếu 4 giờ) $\rightarrow$ phải kiểm thử thêm 4 giờ nữa, nếu trong 4 giờ này có lỗi xảy ra thì phải tiếp tục áp dụng công thức.

$\Rightarrow$ *Lặp lại thao tác này cho đến khi không còn thời gian thiếu nữa*

15

GIAI ĐOẠN BẢO TRÌ

(MAINTENANCE PHASE)

Nội dung:

- Khái quát chung
- Sự cần thiết của bảo trì
- Những đòi hỏi đối với các nhà lập trình bảo trì
- Quản lý bảo trì
- Bảo trì phần mềm hướng đối tượng
- So sánh kỹ năng bảo trì và kỹ năng phát triển
- Kiểm thử
- Đánh giá

15.1Khái quát chung
(overview)

- Giai đoạn bảo trì bắt đầu sau khi khách hàng đã chấp thuận sản phẩm và cần có các thay đổi trên sản phẩm
- Các thể hiện của bảo trì: mã nguồn, tài liệu, hướng dẫn sử dụng,...
- Còn gọi là sự tiến triển (evolution) để chỉ rõ sự phát triển của sản phẩm thay vì gọi đó là bảo trì

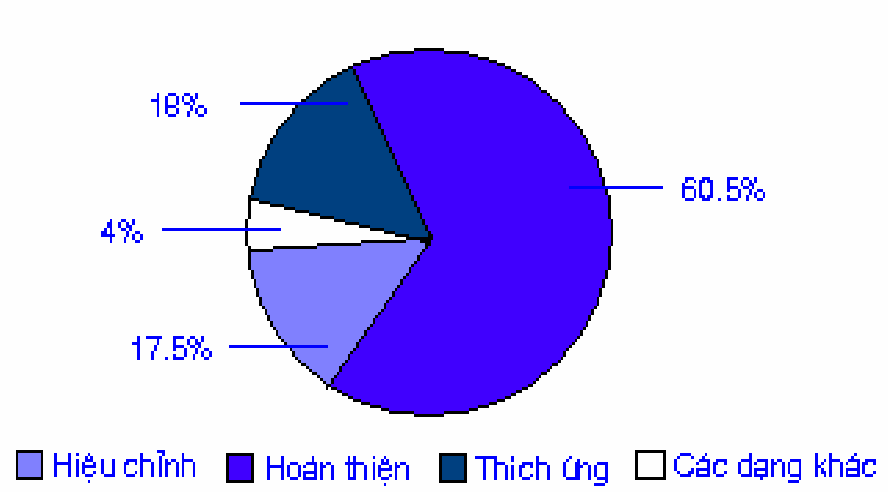
15.2Sự cần thiết của bảo trì
(why maintenance is necessary)

- Hiệu chỉnh *(corrective maintenance)*: khoảng 17.5%; các lỗi đặc tả , thiết kế, tài liệu, mã nguồn hay các dạng khác
  VD: Nghiên cứu trên 69 công ty của [Lientz, Swanson và Tompkins]

- Hoàn thiện (*perfective maintenance*): khoảng 60.5%; các thay đổi về mã lệnh nhằm hoàn thiện hiệu năng của sản phẩm
  VD: Khách hàng yêu cầu thêm một số chức năng hay sửa đổi sản phẩm để tăng tốc độ xử lý

- Thích ứng *(adaptive maintenance)*: khoảng 18%; các thay đổi nhằm tác động lại những thay đổi trong môi trường mà sảnphẩm đang vận hành. Khách hàng phải chịu chi phí (external imposed)
  VD: thay đổi trình biên dịch, hệ điều hành hay phần cứng

- Các dạng khác *(other types of maintenance)*: khoảng 4%; thuộc các dạng khác ngoài ba dạng kể trên



Hình 15.1 Các khoảng thời gian cho mỗi dạng bảo trì

15.3Những đòi hỏi đối với các nhà lập trình bảo trì
(what is required of maintenance programmers)

- Hình thành thuật ngữ nhà lập trình bảo trì (maintenance programmer - MP)
- Đây là khía cạnh khó khăn nhất, nhiều thách thức, của một sản phẩm phần mềm vì đụng chạm đến tất cả các giai đoạn trong tiến trình xây dựng phần mềm
- Nghịch lý hiện nay tại các công ty:
  - xem nhẹ công tác bảo trì
  - giao các công đoạn bảo trì cho các lập trình viên mới

VD: Xem xét các khả năng xảy ra khi một báo cáo về lỗi sản phẩm không làm việc giống như trong hướng dẫn sử dụng được chuyển đến cho một MP; các thông tin được điền vào theo quan điểm của người sử dụng.

Các lý giải có thể có:

- không có lỗi, do người sử dụng hiểu không chính xác hướng dẫn sử dụng hoặc sử dụng sản phẩm không đúng cách
- hướng dẫn sử dụng được viết không chính xác
- lỗi tại mã nguồn

- MP phải có kỹ năng lần vết (debugging skills) tốt để xác định chính xác vị trí lỗi
- Lỗi hồi qui (regression fault): sửa chữa lỗi có quan tâm đến các lỗi khác trong sản phẩm
- Chuẩn bị tài liệu chi tiết cho toàn bộ sản phẩm cũng như cho từng mô-đun riêng biệt sau khi sửa chữa xong

- Được xem như là dịch vụ hậu mãi (after-sales service), giữ khách hàng bằng cách cung cấp những dịch vụ bảo trì tốt nhất
- Là chuẩn mực cho sự thành công của công ty phần mềm

15.4Quản lý bảo trì
(management of maintenance)

- Xây dựng cơ chế cho phép có những thay đổi trên sản phẩm khi bảo trì
- Lãnh đạo nhóm SQA và lãnh đạo nhóm phát triển phần mềm phải độc lập với nhau
- Các báo cáo lỗi *(fault reports)*
  - người sử dụng điền các thông tin về lỗi trên các chức năng
  - đủ thông tin để MP có thể tái tạo lại lỗi
- Ủy quyền thay đổi trên sản phẩm *(authorizing changes to the product)*
  - xác định lỗi, thay đổi mã nguồn, cố định mã nguồn
  - kiểm thử qui hồi (regression testing) trên toàn bộ sản phẩm
  - cập nhật các tài liệu để phản ánh các thay đổi
  - có thể cập nhật tài liệu về đặc tả cũng như thiết kế
  - tạo phiên bản mới
  - chuyển đến nhóm SQA để xác nhận lại (nhưng không được can thiệp vào công việc của các lập trình viên)

- Bảo đảm công tác bảo trì *(ensuring maintainability)*
  - việc bảo trì phải được thực hiện nhiều lần
  - tạo nhiều phiên bản
  - có kế hoạch bảo trì trong suốt tiến trình phần mềm
  - ghi nhận cẩn thận các thông tin kỹ thuật
  - tài liệu phải được hoàn tất và hiệu chỉnh chu đáo, phản ánh chính xác mọi thành phần của phiên bản hiện hành
- Vấn đề về sự lặp lại công tác bảo trì *(problem of repeated maintenance)*
  - khách hàng thường xuyên thay đổi các yêu cầu
  - nên đưa ra mô hình làm việc, khi có thay đổi thì khách hàng sẽ phải trả thêm chi phí phát triển

15.5Bảo trì phần mềm hướng đối tượng
(maintenance of object-oriented software)

- Dễ dàng bảo trì các đối tượng
  - do các khái niệm độc lập nên dễ dàng xác định vị trí nhằm hiệu chỉnh hay nâng cao
  - các thay đổi chỉ tác dụng bên trong đối tượng nên giảm thiểu các lỗi hồi qui

- Khó khăn:
  - MP phải nghiên cứu toàn bộ các cây thừa kế
  - khi cài đặt trên ngôn ngữ lập trình hướng đối tượng: vấn đề đa hình và động
  - khi lần vết các thừa kế liên tục nhau khi có một lớp nào đó có một số thay đổi

15.6 So sánh kỹ năng bảo trì và kỹ năng phát triển
(maintenance skills versus development skills)

Khả năng	Kỹ năng bảo trì	Kỹ năng phát triển
Xác định nguyên nhân gây ra lỗi	Hiệu chỉnh	Kiểm thử tích hợp và kiểm thử phát triển
Thực hiện hiệu quả các chức năng mà không có tài liệu thích hợp	Hoàn thiện, thích ứng	Đặc tả, thiết kế, cài đặt và tích hợp, kiểm thử
Nắm vững các vấn đề liên quan trên các giai đoạn	Đòi hỏi như nhau	

15.7Kiểm thử giai đoạn bảo trì
(testing during the maintenance phase)

- Khó khăn khi phải nắm vững toàn bộ sản phẩm
- Cách tiến hành:
  - sử dụng các tình huống kiểm thử (test cases) để đảm bảo sản phẩm vẫn còn vận hành tốt sau khi đã có cập nhật
  - thay đổi một số tình huống kiểm thử
  - lưu trữ toàn bộ các tình huống kiểm thử với kết quả cần đạt tương ứng
- Sử dụng kiểm thử hồi qui

15.8Đánh giá giai đoạn bảo trì
(metrics for the maintenance phase)

- Sử dụng cách đánh giá cho các giai đoạn liên quan như trong quá trình phát triển
- Ngoài ra còn có:
  - số lượng báo cáo lỗi
  - phân loại lỗi theo độ khó và kiểu lỗi
  - thông tin về trạng thái hiện hành của báo cáo lỗi

  VD: 13 báo cáo lỗi nghiêm trọng đã được xử lý,
  03 báo cáo lỗi nghiêm trọng chưa được xử lý.