

25 YEARS ANNIVERSARY

SOKT

The central graphic for the 25th anniversary features the number "25" in large, bold, white letters. A circular arc above the "2" contains the text "YEARS ANNIVERSARY" in a smaller, white, sans-serif font. Below the "25" is the acronym "SOKT" in a large, white, sans-serif font. The entire graphic is set against a red background with a subtle dotted pattern.

**ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

Nhập môn Công nghệ Phần mềm

(Introduction to Software Engineering)

CHƯƠNG 7

Thiết kế phần mềm

Mục tiêu của bài học

Sinh viên sẽ được trang bị các kiến thức sau:

- Các khái niệm liên quan tới Thiết kế phần mềm
- Giới thiệu về hai giai đoạn thiết kế phần mềm: thiết kế kiến trúc và thiết kế chi tiết
- Phân biệt tính móc nối (Coupling) và tính kết dính (Cohesion) trong thiết kế phần mềm

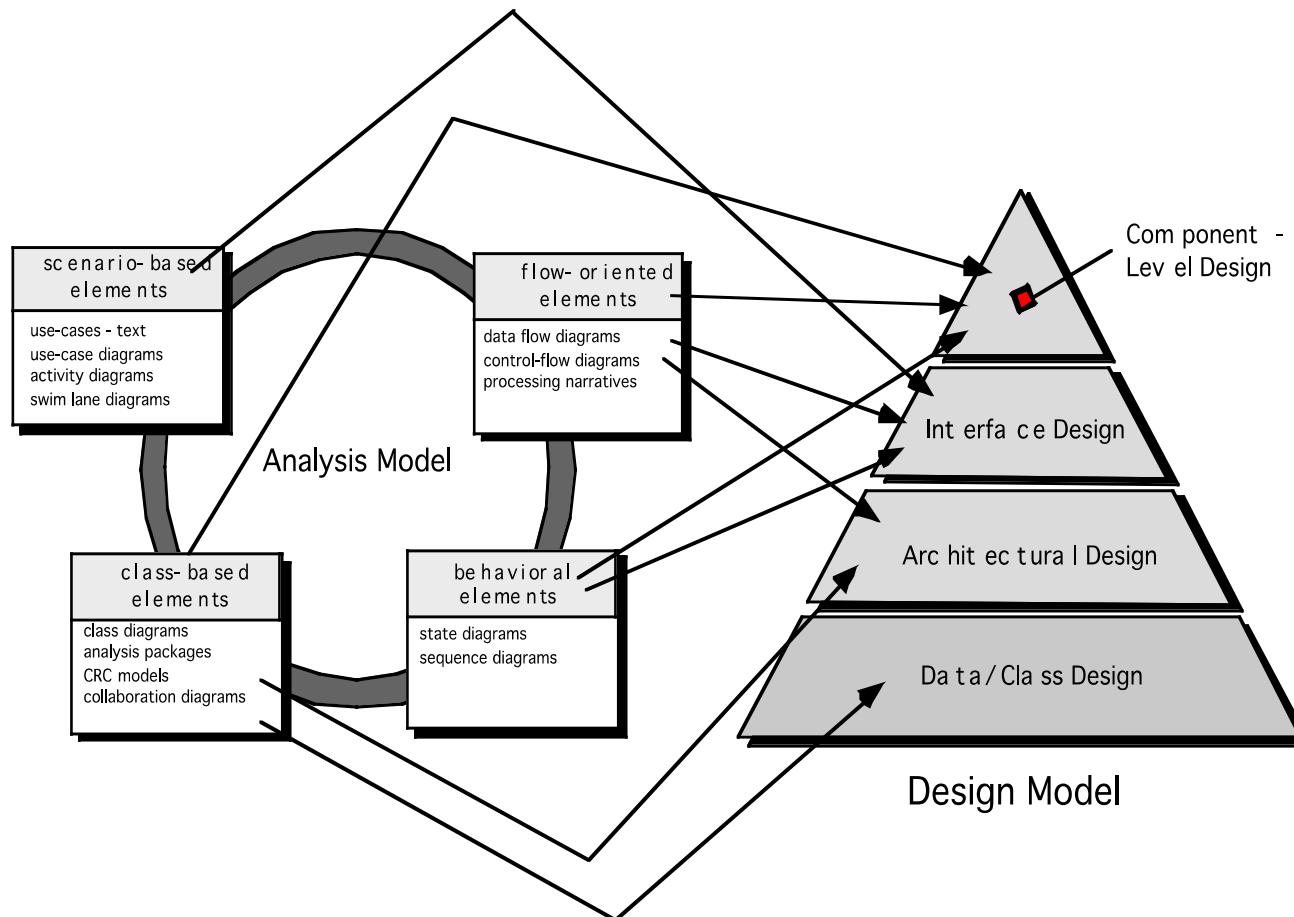
Nội dung

- 1. Tổng quan thiết kế phần mềm**
2. Các khái niệm trong thiết kế phần mềm
3. Thiết kế kiến trúc phần mềm
4. Thiết kế chi tiết phần mềm
5. Tính móc nối (Coupling) và tính kết dính (Cohesion)

1.1 Khái niệm về thiết kế phần mềm

- Thiết kế tạo ra các biểu diễn và dữ kiện của hệ thống phần mềm cần xây dựng từ kết quả phân tích yêu cầu để có thể tiếp tục thực hiện giai đoạn tiếp theo trong tiến trình phát triển phần mềm.
- Kết quả của pha thiết kế là mô hình thiết kế của phần mềm
 - Mô hình thiết kế đủ chi tiết để giai đoạn lập trình có thể thực hiện
 - Là phương tiện để trao đổi thông tin và đảm bảo chất lượng
 - Mô hình thiết kế dễ sửa đổi hơn mã chương trình, cung cấp cái nhìn tổng thể đồng thời có nhiều mức chi tiết

Mô hình phân tích-> Mô hình thiết kế



These slides are designed to accompany Software Engineering:
A Practitioner's Approach, 7/e (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

1.1 Khái niệm về thiết kế phần mềm

- Thiết kế xuất phát từ kết quả phân tích yêu cầu, giúp trả lời câu hỏi “Như thế nào?”
 - Mô tả một hoặc nhiều giải pháp, giúp đánh giá các giải pháp, lựa chọn giải pháp tốt nhất
- Thiết kế thường mô tả ở một mức trừu tượng nhất định, sử dụng các mô hình (khác với cài đặt chi tiết khi lập trình)
- Nếu không có thiết kế hoặc thiết kế tồi:
→ làm tăng công sức viết mã chương trình, tăng công sức bảo trì, khó khăn khi cần sửa đổi, mở rộng

1.1 Khái niệm về thiết kế phần mềm

- Theo Mitch Kapor, người đã tạo ra Lotus 1-2-3, giới thiệu trong “Tuyên ngôn về thiết kế phần mềm” trên Dr. Dobbs Journal. Thiết kế phần mềm tốt nên thể hiện:
 - **Sự ổn định (Firmness):** Một chương trình không nên có bất cứ lỗi nào làm hạn chế chức năng của nó.
 - **Tiện nghi (Commodity):** Một chương trình nên phù hợp với mục đích đã định của nó .
 - **Sự hài lòng (Delight):** Trải nghiệm sử dụng chương trình nên làm hài lòng người dùng.

1.1 Khái niệm về thiết kế phần mềm

- Các giai đoạn thiết kế: hoạt động thiết kế xuất hiện trong các mô hình phát triển phần mềm khác nhau, gồm hai giai đoạn thiết kế chính:
 - Thiết kế kiến trúc / Thiết kế mức cao (high level design)
 - Mô hình tổng thể của hệ thống
 - Cách thức hệ thống được phân rã thành các mô đun
 - Mối quan hệ giữa các môđun
 - Cách thức trao đổi thông tin giữa các môđun
 - Thực hiện bởi nhiều mức trừu tượng
 - Thiết kế chi tiết / Thiết kế mức thấp (low level design)
 - Thiết kế chi tiết lớp, module, thiết kế thuật toán, thiết kế dữ liệu, thiết kế giao diện,...

1.2 Quan hệ giữa thiết kế và chất lượng

- **Thiết kế phải thực hiện tất cả các yêu cầu rõ ràng** chứa trong mô hình phân tích, và nó phải đáp ứng tất cả các yêu cầu tiềm ẩn khách hàng mong muốn .
- **Thiết kế phải là một hướng dẫn dễ hiểu dễ đọc** cho những người tạo ra code và cho những người kiểm thử và sau đó hỗ trợ cho phần mềm.
- **Thiết kế nên cung cấp một bức tranh hoàn chỉnh của phần mềm**, giải quyết vấn đề dữ liệu, chức năng, và hành vi từ một quan điểm thực thi.

1.2 Quan hệ giữa thiết kế và chất lượng

- **Một thiết kế nên thể hiện một kiến trúc mà** (1) đã được tạo ra bằng cách sử dụng phong cách kiến trúc hoặc các pattern được công nhận , (2) bao gồm các thành phần mang những đặc tính thiết kế tốt và (3) có thể được thực hiện một cách tiến hóa
 - Đối với các hệ thống nhỏ hơn, thiết kế đôi khi có thể được phát triển tuyến tính.
- **Một thiết kế nên môđun hoá:** đó là, phần mềm nên được phân chia thành các thành phần hợp lý hoặc hệ thống con
- **Một thiết kế cần có biểu diễn riêng biệt** của dữ liệu, kiến trúc, giao diện, và các thành phần.

1.3 Nguyên tắc Chất lượng

- Một thiết kế nên dẫn đến các cấu trúc dữ liệu thích hợp cho các lớp sẽ được thực thi và được rút ra từ mô hình dữ liệu có thể nhận biết.
- Một thiết kế nên dẫn đến các thành phần mang những đặc tính chức năng độc lập.
- Một thiết kế nên dẫn đến giao diện mà giảm sự phức tạp của các kết nối giữa các thành phần và với môi trường bên ngoài
- Một thiết kế nên được chuyển hóa bằng cách sử dụng một phương pháp lặp lại được dẫn dắt bởi các thông tin thu được trong quá trình phân tích các yêu cầu phần mềm.
- Một thiết kế nên được đại diện bằng một ký hiệu truyền đạt hiệu quả ý nghĩa của nó.

1.4 Nguyên tắc Thiết kế

- Quá trình thiết kế không nên mắc phải 'tunnel vision.'
- Việc thiết kế nên có thể truy ngược về mô hình phân tích.
- Việc thiết kế không nên 'phát minh lại bánh xe'.
- Việc thiết kế nên "giảm thiểu khoảng cách trí tuệ" [DAV95] giữa phần mềm và bài toán như nó tồn tại trong thế giới thực.
- Việc thiết kế nên biểu lộ tính đồng nhất và tích hợp.

1.4 Nguyên tắc Thiết kế

- Việc thiết kế nên được cấu trúc để thích ứng với thay đổi.
- Việc thiết kế nên được cấu trúc để làm suy thoái (degrade) nhẹ nhàng, ngay cả khi đang gặp phải dữ liệu bất thường, các sự kiện, hoặc điều kiện hoạt động .
- Thiết kế không phải là coding, coding không phải là thiết kế.
- Việc thiết kế nên được đánh giá về chất lượng khi nó được tạo ra, chứ không phải sau thực tế.
- Việc thiết kế cần được xem xét để giảm thiểu lỗi khái niệm (ngữ nghĩa).

Nội dung

1. Tổng quan thiết kế phần mềm
- 2. Các khái niệm trong thiết kế phần mềm**
3. Thiết kế kiến trúc phần mềm
4. Thiết kế chi tiết phần mềm
5. Tính móc nối (Coupling) và tính kết dính (Cohesion)

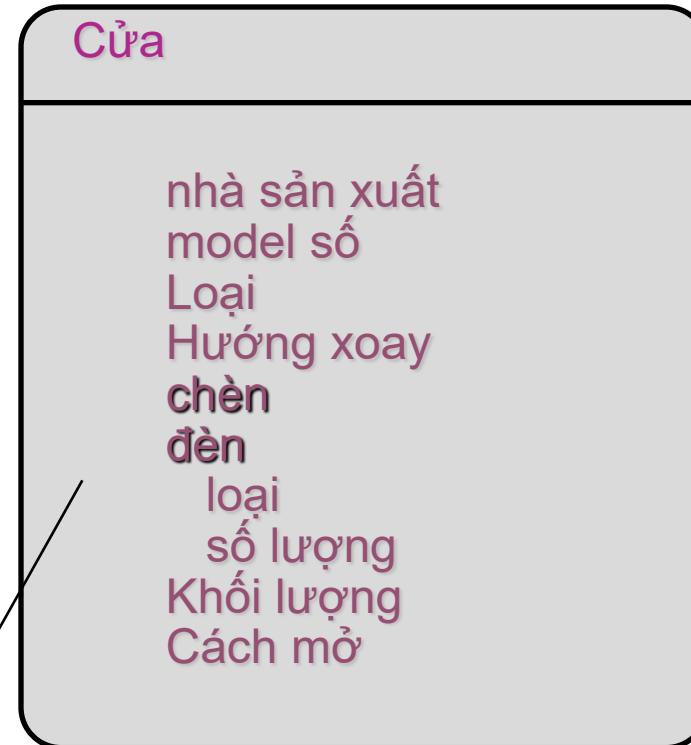
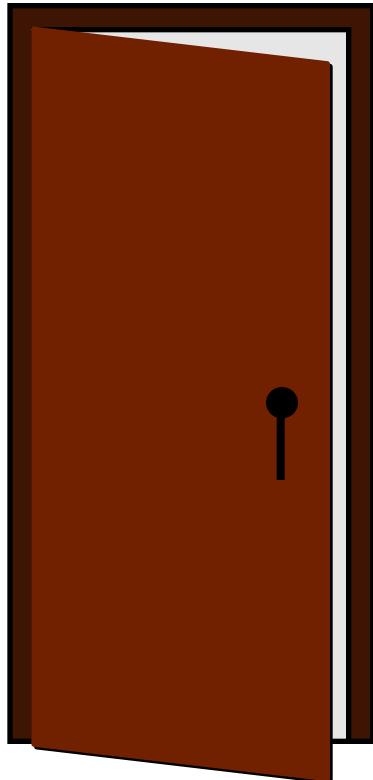
2.1 Các khái niệm cơ bản trong thiết kế phần mềm

- **Trừu tượng hoá**—dữ liệu, thủ tục, kiểm soát
- **Kiến trúc**—cấu trúc tổng thể của phần mềm
- **Mẫu thiết kế (Design Patterns)**—"chuyển tải những tinh túy" của một giải pháp thiết kế đã được chứng minh
- **Phân tách mối quan tâm (Separation of concerns)**—bất kỳ vấn đề phức tạp có thể được xử lý dễ dàng hơn nếu nó được chia thành nhiều mảnh
 - Bằng cách tách mối quan tâm ra nhỏ hơn, và các mảnh dễ quản lý hơn, một vấn đề mất ít hơn công sức và thời gian để giải quyết.
- **Mô đun hoá (Modularity)**—mô đun hoá các dữ liệu và chức năng
- **Tính ẩn**—giao diện được điều khiển

2.1 Các khái niệm cơ bản trong thiết kế phần mềm (2)

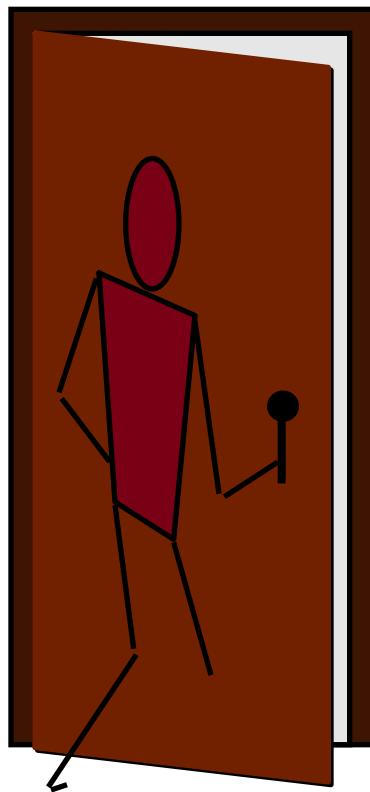
- **Độc lập chức năng** — Các hàm đơn mục đích và khớp nối thấp
- **Sàng lọc**—xây dựng chi tiết cho tất cả các khái niệm trừu tượng
- **Các khía cạnh**—một cơ chế cho sự hiểu biết các yêu cầu tổng thể ảnh hưởng đến thiết kế như thế nào
- **Refactoring**— một kỹ thuật tái tổ chức nhằm đơn giản hóa thiết kế
- **OO design concepts**—các khái niệm về thiết kế hướng đối tượng
- **Thiết kế Lớp**—cung cấp chi tiết thiết kế mà sẽ cho phép các lớp đã phân tích được thực thi

Ví dụ: Trừu tượng dữ liệu



Thực thi như một cấu trúc dữ liệu

Ví dụ: Trừu tượng thủ tục



thực hiện với "kiến thức" về các
đối tượng được kết hợp với vào cửa

Khuôn mẫu thiết kế (Design Patterns)

- Mỗi chúng ta đều đã trải qua vấn đề thiết kế và từng nghĩ: **Liệu đã có ai phát triển một lời giải cho vấn đề này chưa?**
 - Điều gì sẽ xảy ra nếu đã có một cách tiêu chuẩn để mô tả một vấn đề (để bạn có thể nhìn nhận nó), và một phương pháp có tổ chức để trình bày lời giải cho vấn đề đó?
- **Khuôn mẫu thiết kế** là một phương pháp có hệ thống để mô tả các vấn đề và giải pháp, cho phép các cộng đồng công nghệ phần mềm có thể nắm bắt kiến thức thiết kế theo cách có thể tái sử dụng

Khuôn mẫu thiết kế (Design Patterns)

- *Mỗi khuôn mẫu mô tả một vấn đề xảy ra hết lần này đến lần khác trong môi trường của chúng ta và sau đó mô tả cốt lõi của giải pháp cho vấn đề đó theo một cách mà bạn có thể sử dụng nó hàng triệu lần mà không bao giờ phải làm lại một việc lần thứ hai.*

Christopher Alexander, 1977

- “một quy tắc ba phần diễn tả một mối quan hệ giữa một ngũ cảnh, một vấn đề, và một giải pháp.”

Các loại khuôn mẫu

- **Architectural patterns** Mẫu kiến trúc mô tả các vấn đề thiết kế trên diện rộng được giải quyết bằng cách tiếp cận cấu trúc.
- **Data patterns** Mẫu dữ liệu mô tả vấn đề dữ liệu và các giải pháp mô hình dữ liệu có thể được dùng để giải quyết vấn đề trên
- **Component patterns** Mẫu thành phần (còn gọi là các mẫu thiết kế) nhằm đến các vấn đề liên quan với việc phát triển các hệ thống con và các thành phần, cách thức chúng giao tiếp với nhau, và vị trí của chúng trong một kiến trúc lớn hơn
- **Interface design patterns**. Mẫu thiết kế giao diện mô tả các vấn đề giao diện người dùng thông thường và giải pháp bao gồm các đặc trưng cụ thể của người dùng cuối.
- **WebApp patterns**. Mẫu WebApp giải quyết tập hợp vấn đề có thể gặp phải khi xây dựng Ứng dụng web và thường kết hợp nhiều mô hình khác nhau.

Các loại khuôn mẫu (2)

- **Khuôn mẫu khởi tạo (Creational patterns)** tập trung vào việc khởi tạo, sáng tác và biểu diễn của các đối tượng, ví dụ:
 - Abstract factory pattern
 - Factory method pattern
- **Khuôn mẫu cấu trúc (Structural patterns)** tập trung vào các vấn đề và các giải pháp liên quan đến cách các lớp và các đối tượng được tổ chức và tích hợp để xây dựng một cấu trúc lớn hơn, ví dụ như:
 - Adapter pattern
 - Aggregate pattern
- **Khuôn mẫu hành vi (Behavioral patterns)** giải quyết các vấn đề liên quan đến việc phân công trách nhiệm giữa các đối tượng và cách thức mà giao tiếp được thực hiện giữa các đối tượng, ví dụ như:
 - Chain of responsibility pattern:
 - Command pattern:

Ví dụ: thông tin của một mẫu thiết kế

Bản mẫu thiết kế

Tên Pattern—mô tả bản chất của mô hình trong một tên ngắn nhưng ý nghĩa

Intent (ý định)—mô tả các mô hình và những gì nó làm

Also-known-as—liệt kê các từ đồng nghĩa cho các pattern

Motivation(Động lực)—cung cấp một ví dụ về vấn đề

Applicability (Khả năng áp dụng)—lưu ý tình huống thiết kế cụ thể, trong đó mô hình được áp dụng

Structure(cấu trúc)—mô tả các lớp được yêu cầu để thực hiện mô hình

Participants (thành phần tham gia)—mô tả trách nhiệm của các lớp được yêu cầu để thực hiện pattern

Collaborations (sự cộng tác)—mô tả cách những thành phần tham gia cộng tác để thực hiện trách nhiệm của mình

Consequences(hệ quả)—mô tả các "lực lượng thiết kế" có ảnh hưởng đến các mô hình và các đánh đổi tiềm năng phải được xem xét khi mô hình được thực hiện

Related patterns(patterns liên quan)—tham khảo chéo liên quan đến các mẫu thiết kế

Khung (frameworks)

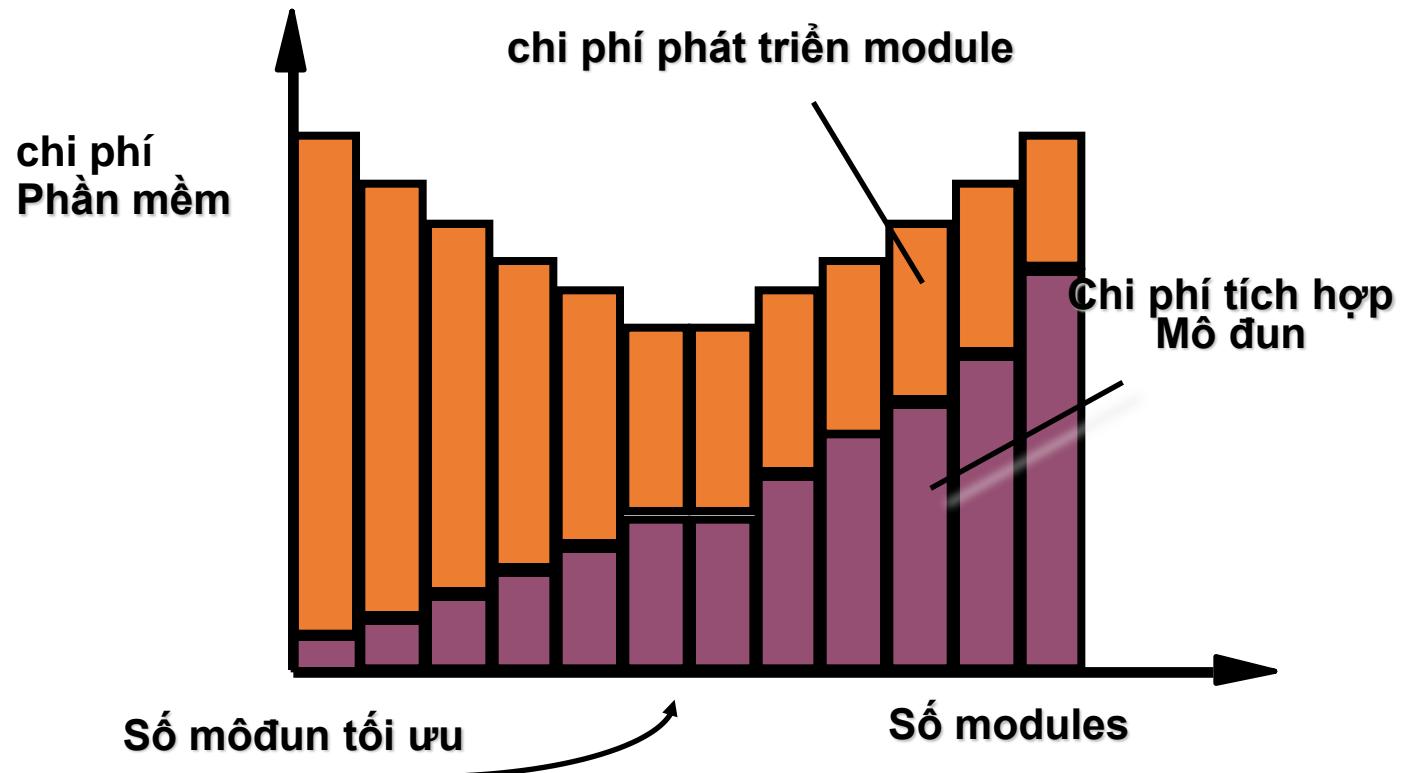
- Bản thân khuôn mẫu có thể không đủ để phát triển một thiết kế đầy đủ
 - Trong một số trường hợp, cần thiết phải cung cấp một cơ sở hạ tầng thực hiện cụ thể, được gọi là một khung (**framework**) cho công việc thiết kế.
- Một **framework** không phải là một khuôn mẫu kiến trúc, mà là một bộ khung với một tập hợp các “**plug points**” (còn được gọi là hooks hay slots) cho phép nó thích ứng với một miền vấn đề xác định.
 - Những “**plug points**” cho phép bạn tích hợp các lớp vấn đề cụ thể hoặc chức năng trong các bộ khung.

Mô đun hoá (Modularity)

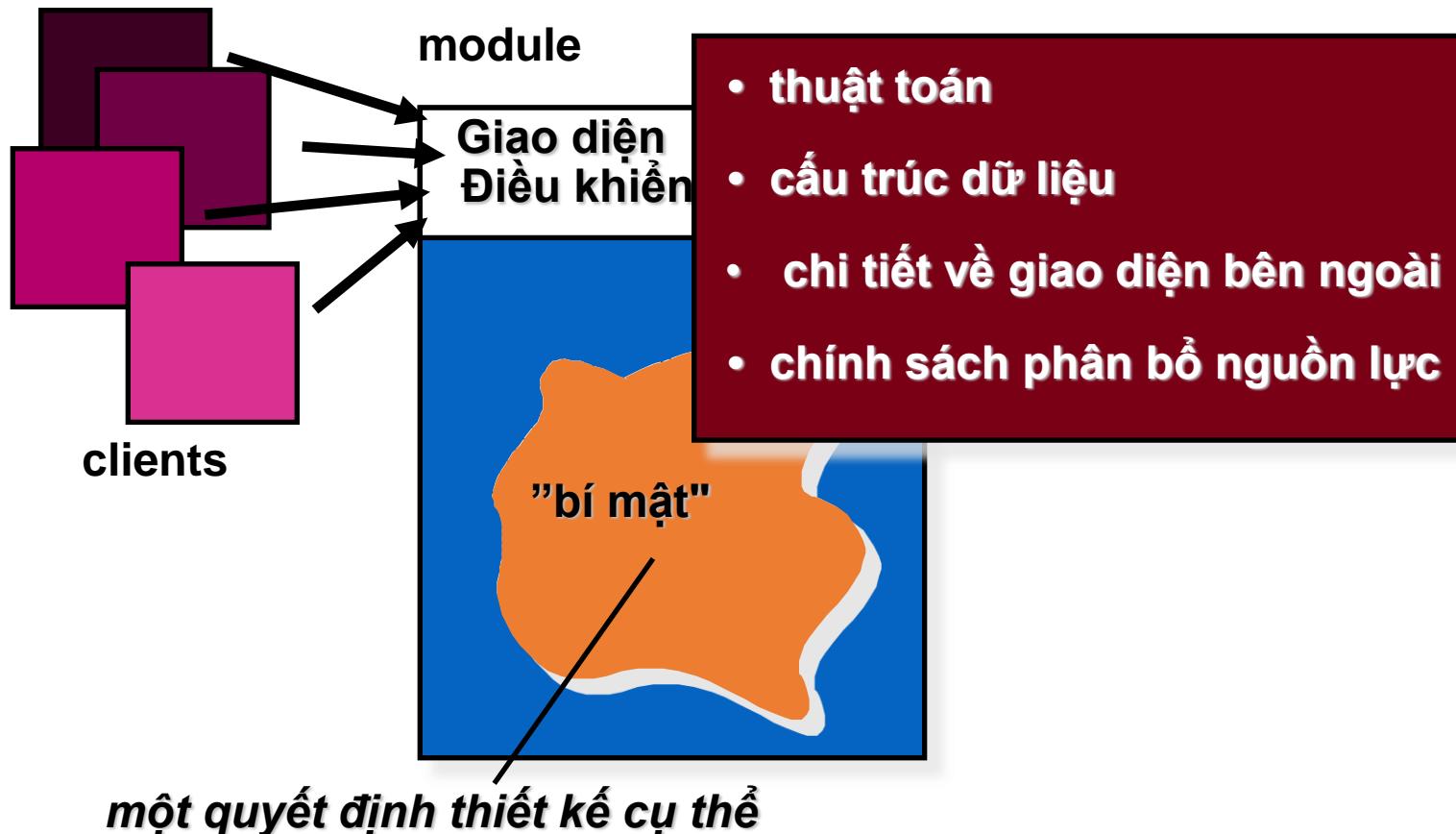
- "Mô đun là thuộc tính duy nhất của phần mềm cho phép một chương trình có thể quản lý một cách thông minh" [Mye78].
- Phần mềm nguyên khối (ví dụ, một chương trình lớn gồm một mô-đun duy nhất) có thể không được dễ dàng nắm bắt được bởi một kỹ sư phần mềm.
 - Số lượng các đường dẫn điều khiển, khoảng thời gian tham khảo, số lượng các biến, và độ phức tạp tổng thể sẽ làm cho việc hiểu được gần như không thể.
- Trong hầu hết các trường hợp, bạn nên phá vỡ thiết kế thành nhiều module, hy vọng sẽ làm cho việc hiểu biết dễ dàng hơn và như một hệ quả, giảm chi phí cần thiết để xây dựng các phần mềm.

Modularity: sự đánh đổi

**Số "chính xác" các mô đun
cho một thiết kế phần mềm cụ thể?**



Ân thông tin



Tại sao lại Ân thông tin?

- Làm giảm khả năng "tác dụng phụ"
- Hạn chế ảnh hưởng chung của quyết định thiết kế cục bộ
- Nhấn mạnh truyền thông qua giao diện điều khiển
- Không khuyến khích việc sử dụng các dữ liệu toàn cục
- Dẫn đến đóng gói, một thuộc tính của thiết kế chất lượng cao
- Kết quả tạo ra phần mềm chất lượng cao

Tái cấu trúc (Refactoring)

- Fowler [FOW99] định nghĩa cấu trúc lại theo cách sau đây:
 - "Tái cấu trúc là quá trình thay đổi một hệ thống phần mềm trong một cách mà nó không làm thay đổi hành vi bên ngoài của mã [thiết kế] nhưng cải thiện cấu trúc bên trong của nó."
 - Khi phần mềm được refactored, thiết kế hiện có được kiểm tra về sự
 - Dư thừa
 - Yếu tố thiết kế không sử dụng
 - Các thuật toán không hiệu quả hoặc không cần thiết
 - Cấu trúc dữ liệu xây dựng kém hoặc không phù hợp
 - Hoặc bất kỳ sự thất bại thiết kế khác có thể được điều chỉnh để mang lại một thiết kế tốt hơn.

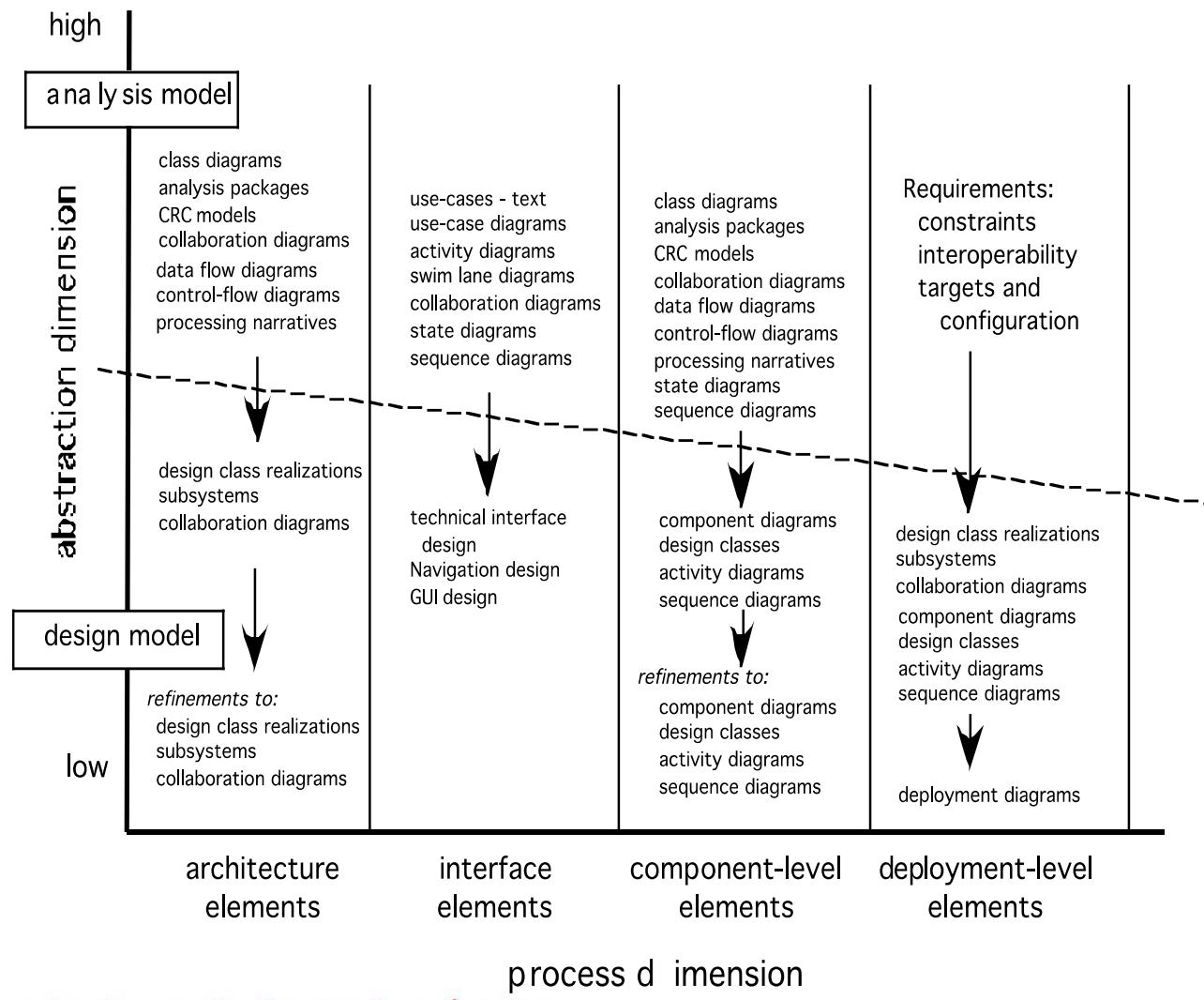
2.2 Các khái niệm thiết kế Hướng đối tượng

- **Các lớp thiết kế**
 - Các lớp thực thể
 - Các lớp biên
 - Các lớp điều khiển
- **Sự thừa kế**—tất cả các trách nhiệm của một lớp cha được ngay lập tức được thừa kế bởi tất cả các lớp con
- **Thông điệp**—khuyến khích một số hành vi xảy ra trong đối tượng nhận
- **Đa hình**—một đặc tính mà làm giảm đáng kể nỗ lực cần thiết để mở rộng thiết kế.

Các lớp thiết kế

- Các lớp phân tích được tinh chỉnh trong quá trình thiết kế để trở thành **các lớp thực thể**
- **Các lớp biên** phát triển trong thiết kế để tạo ra giao diện (ví dụ, màn hình tương tác hoặc báo cáo) mà người dùng thấy và tương tác với phần mềm.
 - Các lớp biên được thiết kế với trách nhiệm quản lý các đối tượng cách thực thể được đại diện cho người sử dụng.
- **Các lớp điều khiển** được thiết kế để quản lý
 - Việc tạo ra hoặc cập nhật các đối tượng thực thể;
 - Sự tức thời của các đối tượng biên khi họ có được thông tin từ các đối tượng thực thể;
 - Truyền thông phức tạp giữa các tập của các đối tượng;
 - Xác nhận của dữ liệu trao đổi giữa các đối tượng hoặc giữa người sử dụng và với ứng dụng.

2.3 Mô hình thiết kế



Các thành phần trong Mô hình thiết kế

- **Các thành phần dữ liệu**

- Mô hình dữ liệu -> cấu trúc dữ liệu
- Mô hình dữ liệu -> kiến trúc cơ sở dữ liệu

- **Các thành phần kiến trúc**

- Miền ứng dụng
- Các lớp phân tích, mối quan hệ, hợp tác và hành vi của chúng được chuyển thành chứng ngộ thiết kế
- Có thể áp dụng các mẫu thiết kế và “kiểu”

- **Các thành phần giao diện**

- Giao diện người dùng (UI)
- Giao diện bên ngoài đến các hệ thống khác, các thiết bị, mạng, hoặc các nhà sản xuất khác hoặc người dùng thông tin
- Giao diện nội bộ giữa các thành phần thiết kế khác nhau.

- **Các yếu tố cấu thành**

- **Các thành phần triển khai**

Các thành phần kiến trúc

- Các mô hình kiến trúc [Sha96] có nguồn gốc từ ba nguồn:
 - Thông tin về miền ứng dụng cho xây dựng phần mềm;
 - Các thành phần mô hình yêu cầu cụ thể như sơ đồ luồng dữ liệu và phân tích các lớp, các mối quan hệ và sự hợp tác của chúng, và
 - Sự sẵn có của mô hình kiến trúc và các kiểu

Các thành phần giao diện

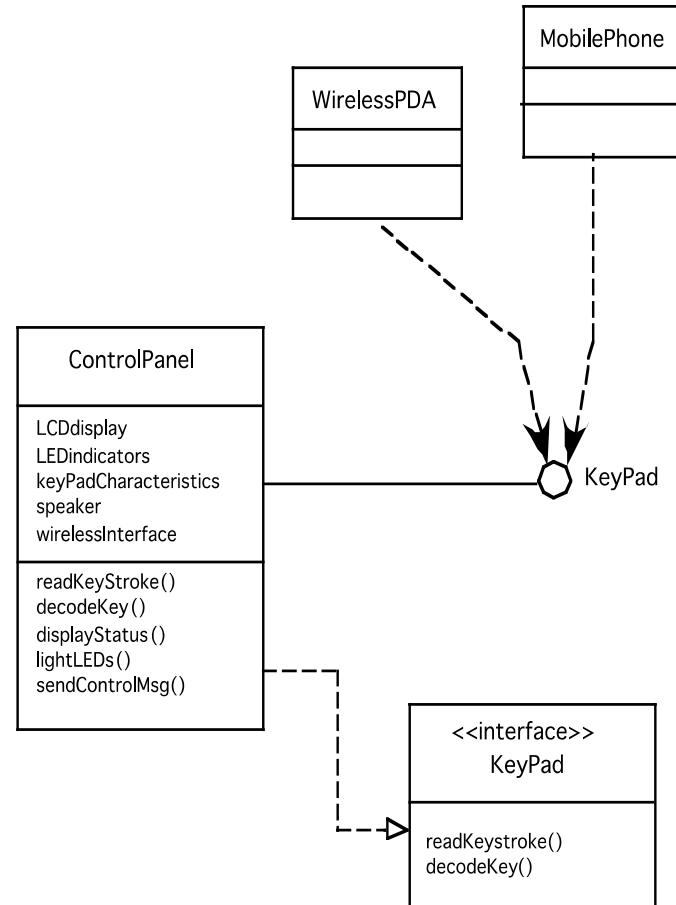


Figure 9.6 UML interface representation for ControlPanel

Các thành phần triển khai

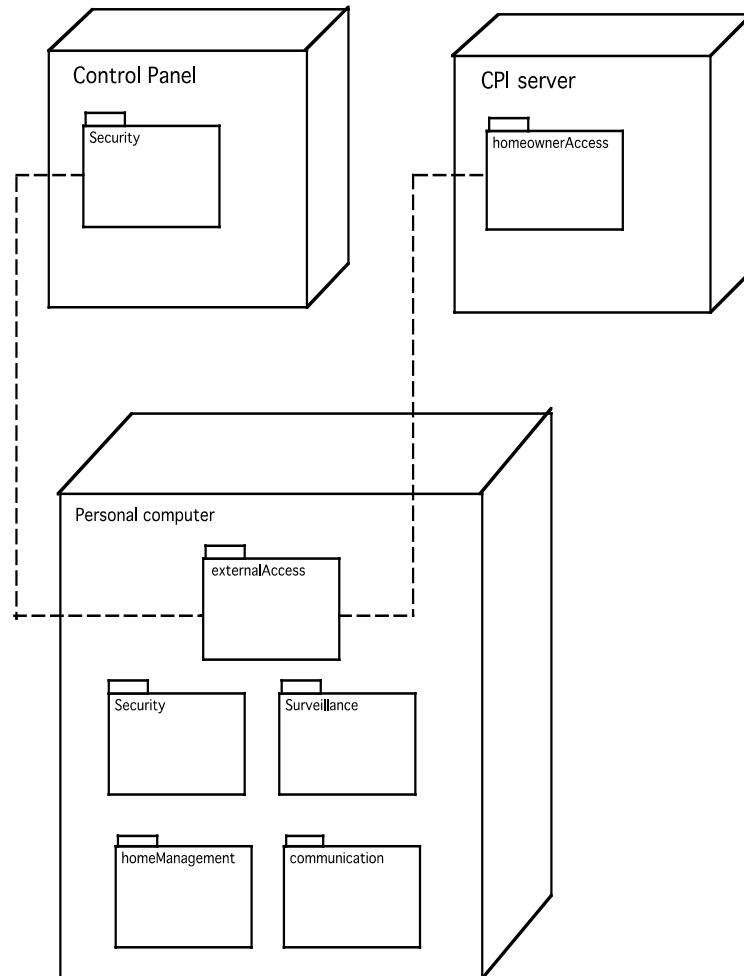


Figure 9.8 UML deployment diagram for *SafeHome*

Nội dung

1. Tổng quan thiết kế phần mềm
2. Các khái niệm trong thiết kế phần mềm
- 3. Thiết kế kiến trúc phần mềm**
4. Thiết kế chi tiết phần mềm
5. Tính móc nối (Coupling) và tính kết dính (Cohesion)

3.1 Tại sao cần thiết kế kiến trúc?

- Các kiến trúc không phải là phần mềm hoạt động. Thay vào đó, nó là một đại diện cho phép một kỹ sư phần mềm:
 1. Phân tích hiệu quả của thiết kế trong việc đáp ứng các yêu cầu đề ra,
 2. Xem xét lựa chọn thay thế kiến trúc khi thay đổi thiết kế vẫn tương đối dễ dàng, và
 3. Giảm thiểu rủi ro gắn liền với việc xây dựng các phần mềm.

3.1 Tại sao cần thiết kế kiến trúc?

- **Đại diện của kiến trúc phần mềm là một tạo khả năng** cho truyền thông giữa tất cả các bên (các bên liên quan) quan tâm đến sự phát triển của một hệ thống dựa trên máy tính.
- **Những kiến trúc làm nổi bật thiết kế quyết định ban đầu** mà sẽ có một tác động sâu sắc trên tất cả các công việc kỹ thuật phần mềm sau và, quan trọng hơn, vào sự thành công cuối cùng của hệ thống như là một thực thể hoạt động.
- **Kiến trúc "tạo thành một mode minh bạch tương đối** **nhỏ** về cách hệ thống được cấu trúc và cách các thành



3.2 Mô tả kiến trúc

- The IEEE Standard định nghĩa một mô tả kiến trúc (AD) là một "một tập hợp các sản phẩm để tài liệu hóa một kiến trúc"
 - Các mô tả chính nó được đại diện bằng cách sử dụng nhiều quan điểm, nơi từng xem là "một đại diện của cả một hệ thống từ quan điểm của một tập hợp có liên quan của [các bên liên quan] các mối quan tâm"

Thể loại kiến trúc

- **Thể loại** ngũ ý một phân loại cụ thể trong lĩnh vực phần mềm tổng thể.
- Trong mỗi thể loại, bạn gặp phải một số tiểu phân loại.
 - Ví dụ, trong các thể loại của các tòa nhà, bạn sẽ gặp phải những phong cách chung sau đây: nhà ở, căn hộ, chung cư, cao ốc văn phòng, tòa nhà công nghiệp, nhà kho, vv.
 - Trong mỗi phong cách chung, phong cách cụ thể hơn có thể được áp dụng. Mỗi phong cách sẽ có một cấu trúc có thể được mô tả bằng một tập các mô hình dự đoán được.

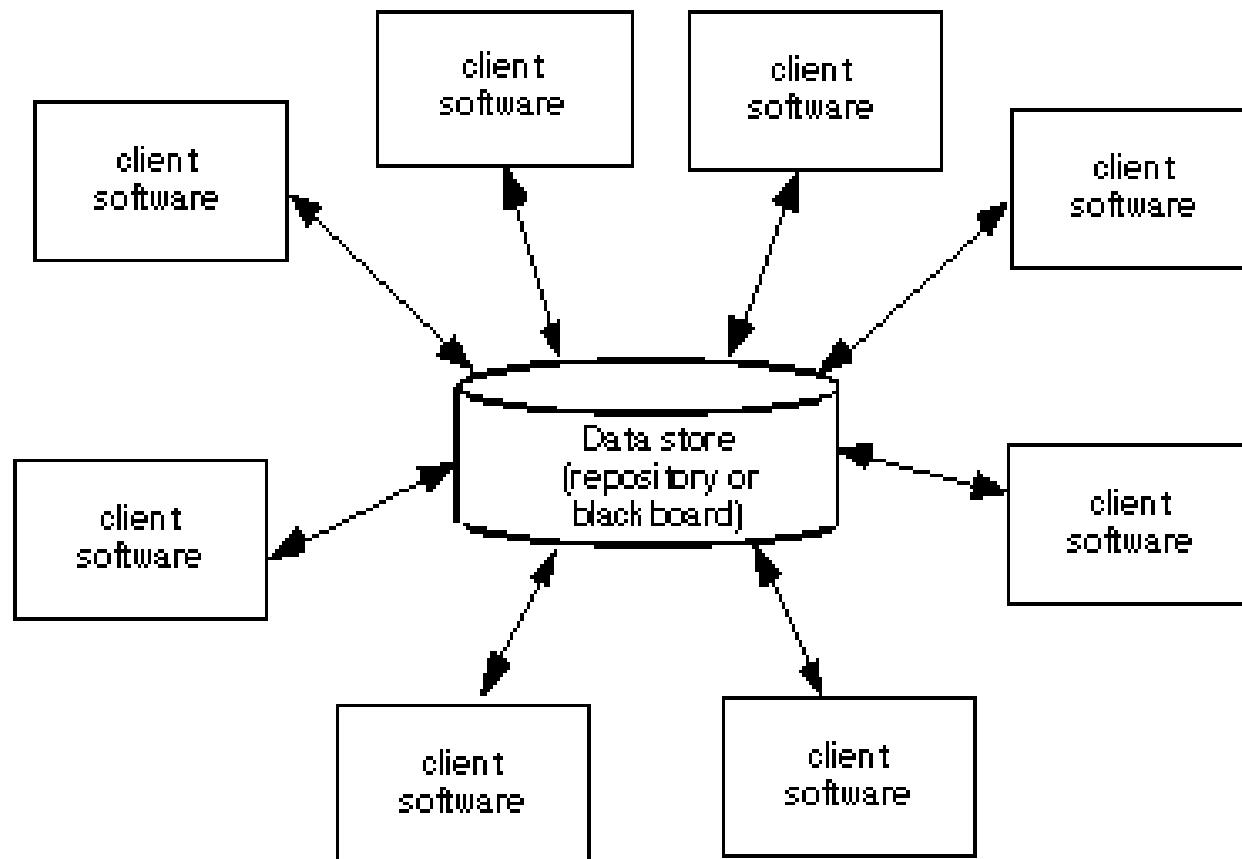
3.3 Kiểu kiến trúc

- **Mỗi phong cách mô tả một loại hệ thống bao gồm:**
 - (1) một tập hợp các thành phần (ví dụ, một cơ sở dữ liệu, mô đun tính toán) thực hiện một chức năng cần thiết của một hệ thống,
 - (2) một tập hợp các kết nối cho phép "truyền thông, phối hợp và hợp tác" giữa các thành phần,
 - (3) khó khăn để xác định cách các thành phần có thể được tích hợp để tạo thành hệ thống, và
 - (4) các mô hình ngữ nghĩa cho phép một nhà thiết kế phải hiểu được tính chất tổng thể của hệ thống bằng cách phân tích các đặc tính được biết đến của các bộ phận cấu thành của nó.

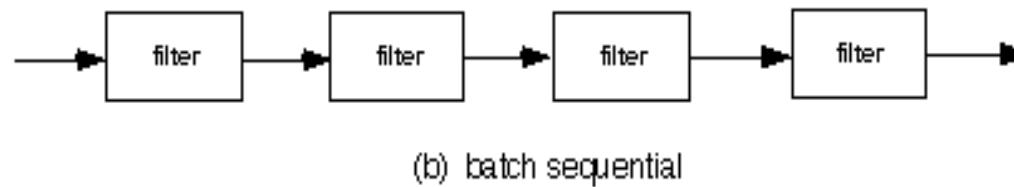
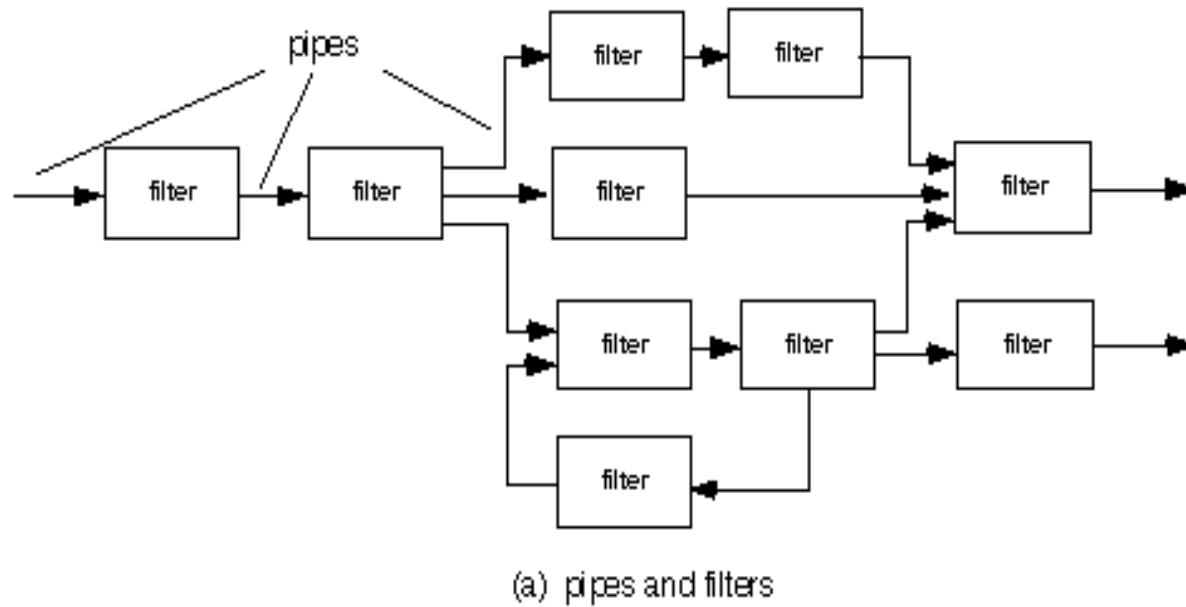
3.3 Kiểu kiến trúc

- Một số kiểu kiến trúc hệ thống:
 - Kiến trúc lấy dữ liệu làm trung tâm
 - Kiến trúc luồng dữ liệu
 - Kiến trúc gọi và trả về
 - Kiến trúc phân lớp
 - Kiến trúc hướng đối tượng
 - ...

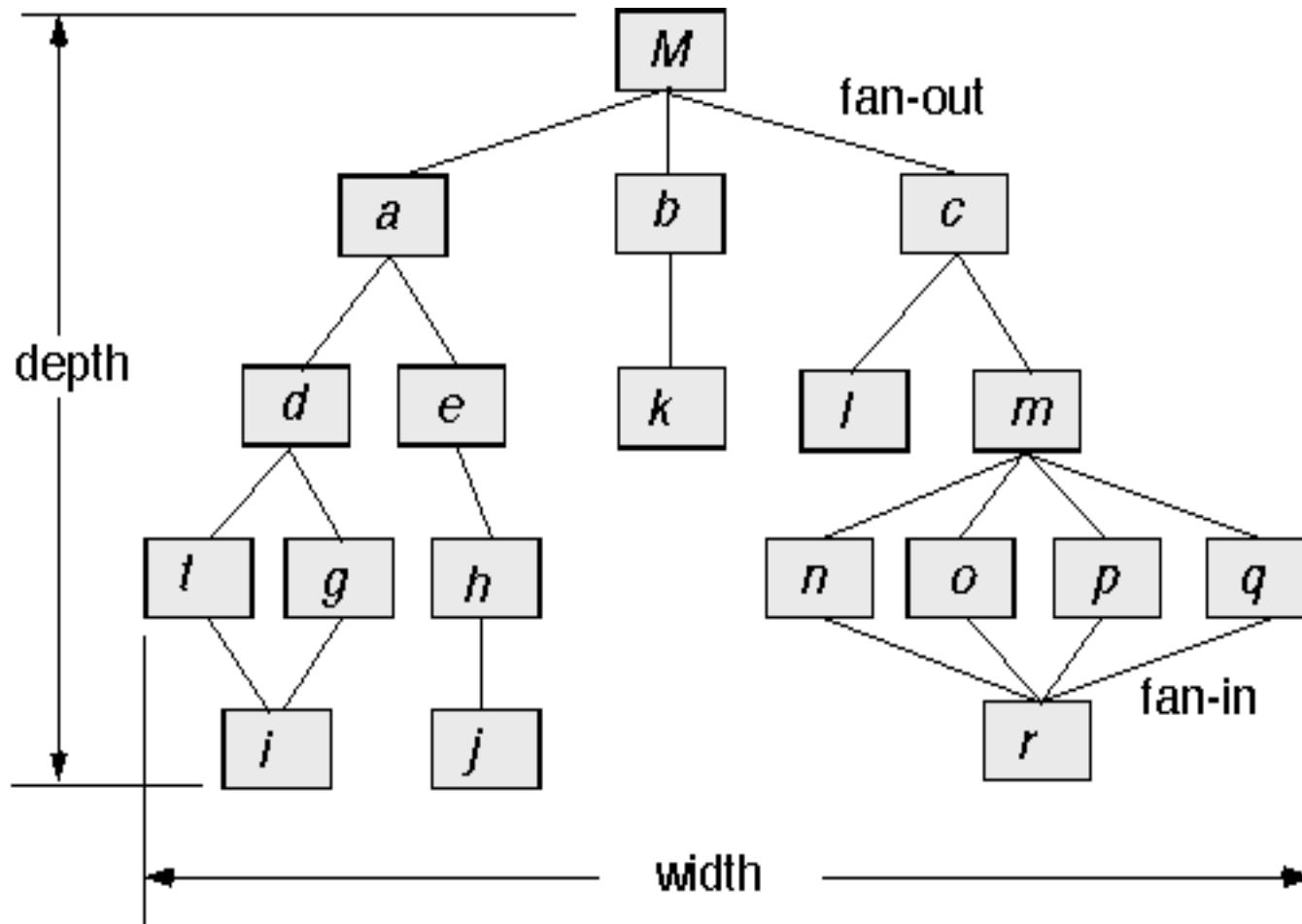
Kiến trúc lấy dữ liệu làm trung tâm



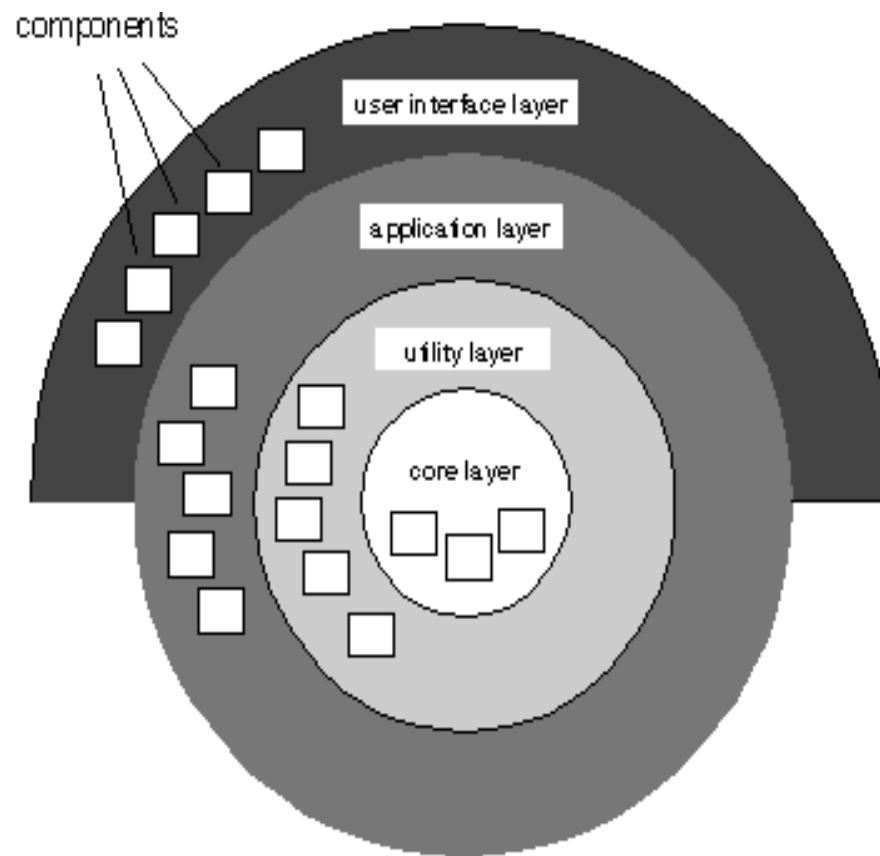
Kiến trúc luồng dữ liệu



Kiến trúc gọi và trả về



Kiến trúc phân lớp



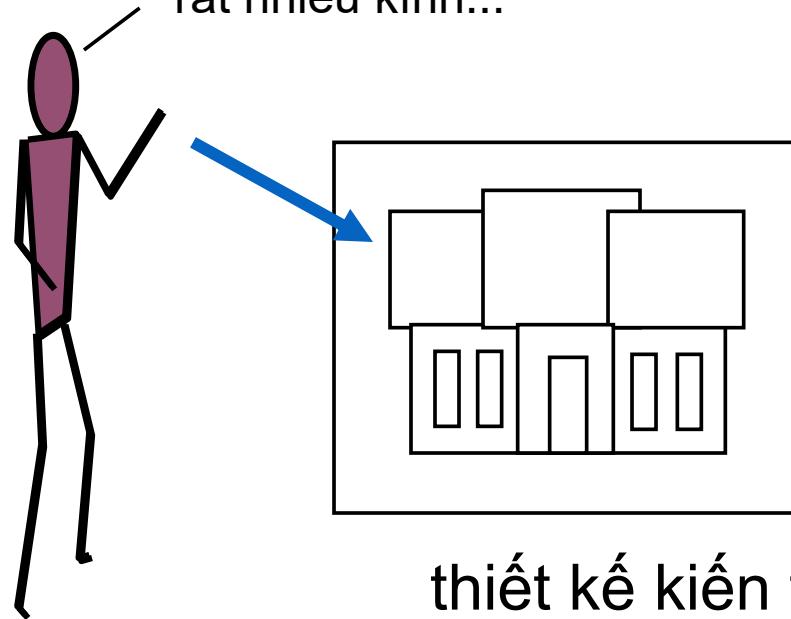
3.4 Thiết kế kiến trúc

- Phần mềm phải được đặt trong bối cảnh
 - Thiết kế cần xác định các thực thể bên ngoài (các hệ thống khác, thiết bị, con người) mà phần mềm tương tác với và bản chất của sự tương tác
- Một tập hợp các nguyên mẫu kiến trúc cần được xác định
 - Một nguyên mẫu là một trừu tượng (tương tự như một lớp) đại diện cho một phần tử của hệ thống hành vi
 - Các nhà thiết kế xác định cấu trúc của hệ thống bằng cách xác định và tinh chỉnh các thành phần phần mềm thực hiện từng nguyên mẫu

Ví dụ

yêu cầu khách hàng

"bốn phòng ngủ, ba phòng tắm,
rất nhiều kính..."



thiết kế kiến trúc

Các bước thiết kế kiến trúc

1. Thu thập các kịch bản.
2. Gợi ý các yêu cầu, ràng buộc, và đặc tả môi trường.
3. Mô tả các phong cách / mô hình kiến trúc đã được lựa chọn để giải quyết các tình huống và yêu cầu:
 - quan điểm mô-đun
 - góc nhìn quá trình
 - quan điểm dòng dữ liệu
4. Đánh giá chất lượng thuộc tính bằng cách coi mỗi thuộc tính trong sự cô lập.
5. Xác định mức độ nhạy cảm của các thuộc tính chất lượng với các thuộc tính kiến trúc khác nhau cho một phong cách kiến trúc cụ thể.
6. Kiến trúc ứng cử viên phê bình (được phát triển trong bước 3) bằng cách sử dụng phân tích độ nhạy được thực hiện ở bước 5.

Nội dung

1. Tổng quan thiết kế phần mềm
2. Các khái niệm trong thiết kế phần mềm
3. Thiết kế kiến trúc phần mềm
- 4. Thiết kế chi tiết phần mềm**
5. Tính móc nối (Coupling) và tính kết dính (Cohesion)

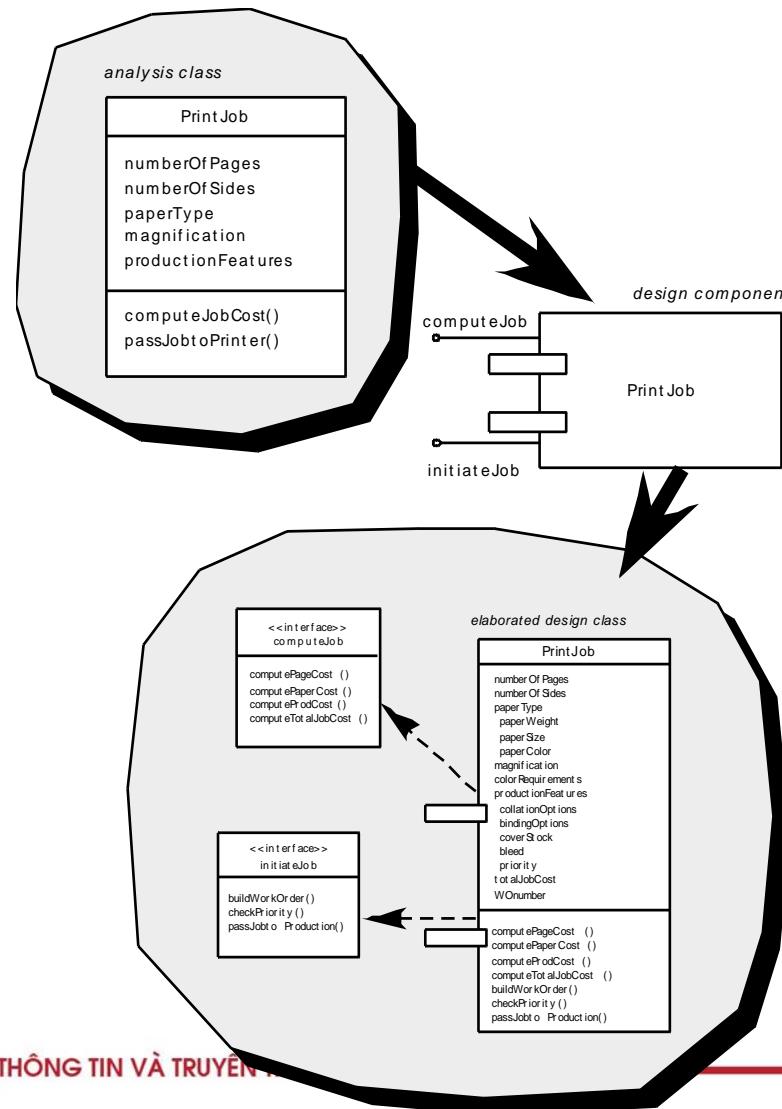
4.1 Thiết kế chi tiết

- Thiết kế chi tiết: chỉ ra chi tiết và đầy đủ về **thành phần**, tạo điều kiện xây dựng phần mềm trong pha sau đó → thiết kế mức thành phần
- Các hoạt động thiết kế chi tiết:
 - Thiết kế chi tiết lớp, module,
 - Thiết kế thuật toán,
 - Thiết kế dữ liệu,
 - Thiết kế giao diện,
 - ...

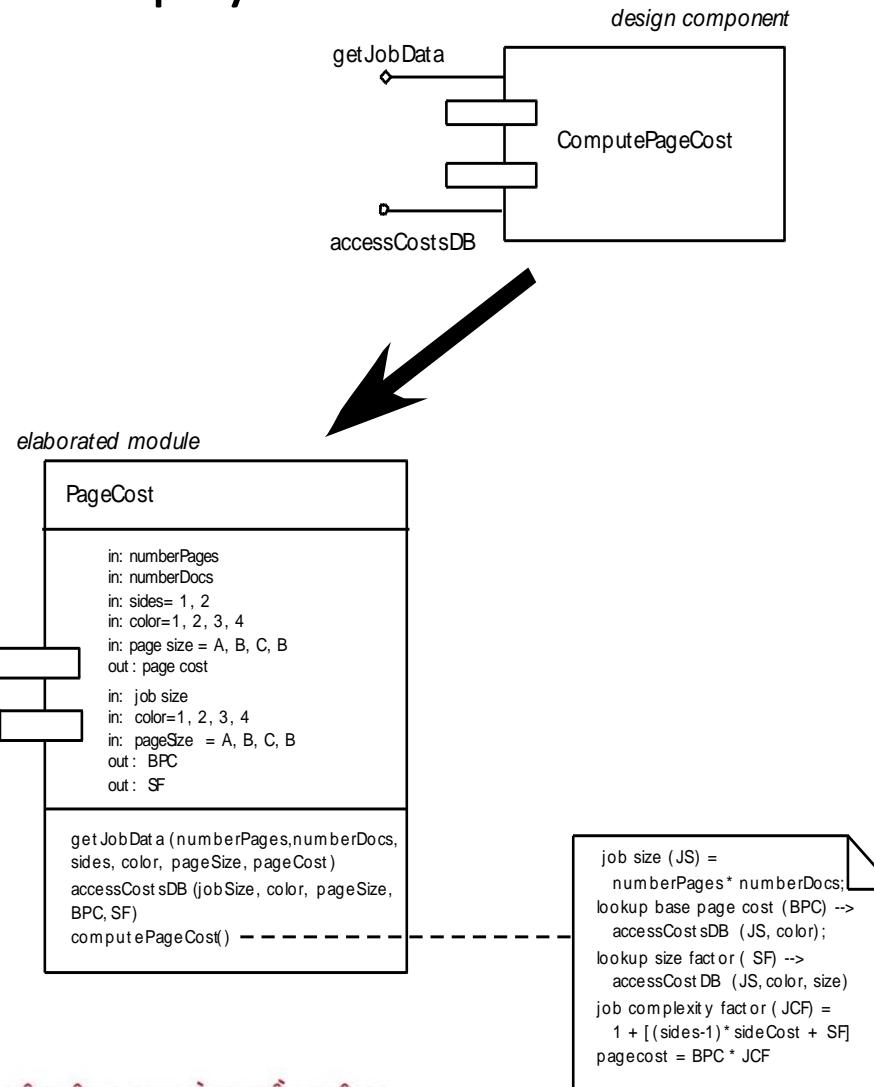
Thế nào là một thành phần?

- *OMG Unified Modeling Language Specification* [OMG01] định nghĩa một thành phần (component) là:
 - Một phần có tính mô-đun, có thể triển khai và thay thế của một hệ thống mà đóng gói việc thực thi và đưa ra một tập các giao diện.
- **Góc nhìn hướng đối tượng:** Một thành phần bao gồm một tập các lớp cộng tác.
- **Góc nhìn quy ước:** Một thành phần bao gồm logic xử lý, cấu trúc dữ liệu bên trong cần thiết để triển khai logic đó và một giao diện cho phép thành phần được gọi và dữ liệu được truyền đến nó.

Thành phần hướng đối tượng



Thành phần quy ước



Nguyên lý thiết kế cơ bản

- **Nguyên lý mở-đóng (OCP)**: Một mô-đun (thành phần) nên được mở cho việc mở rộng nhưng nên đóng cho việc hiệu chỉnh.
- **Nguyên lý thay thế Liskov (LSP)**: Các lớp con nên thay thế được các lớp chs.
- **Nguyên lý tráo đổi phụ thuộc (DIP)**: Phụ thuộc vào trừu tượng hóa, không phụ thuộc vào kết cấu.
- **Nguyên lý tách biệt giao diện (ISP)**: Nhiều giao diện client cụ thể thì tốt hơn một giao diện mục đích chung.
- **Nguyên lý phát hành và tái sử dụng tương đương (REP)**: "Nguyên tắc tái sử dụng là nguyên tắc phát hành". Nói cách khác, nếu một thành phần được coi là có thể tái sử dụng thì nó phải là một đơn vị có thể thay thế được.
- **Nguyên lý đóng chung (CCP)**: Các lớp thay đổi cùng nhau thì thuộc về nhau
- **Nguyên lý tái sử dụng chung (CRP)**. Các lớp không được tái sử dụng cùng nhau thì không nên nhóm lại với nhau"

Source: Martin, R., "Design Principles and Design Patterns," downloaded from <http://www.objectmentor.com>, 2000.

Hướng dẫn thiết kế

- **Thành phần:**

- Quy ước đặt tên nên được thiết lập cho các thành phần được xác định như là một phần của mô hình kiến trúc rồi sau đó tinh chỉnh và xây dựng như là một phần của mô hình mức thành phần

- **Giao diện:**

- Giao diện cung cấp thông tin quan trọng về sự giao tiếp và cộng tác (đồng thời cũng giúp chúng ta đạt được OPC)

- **Phụ thuộc và kế thừa:**

- Việc mô hình hóa sự phụ thuộc từ trái sang phải và sự kế thừa từ dưới lên trên.

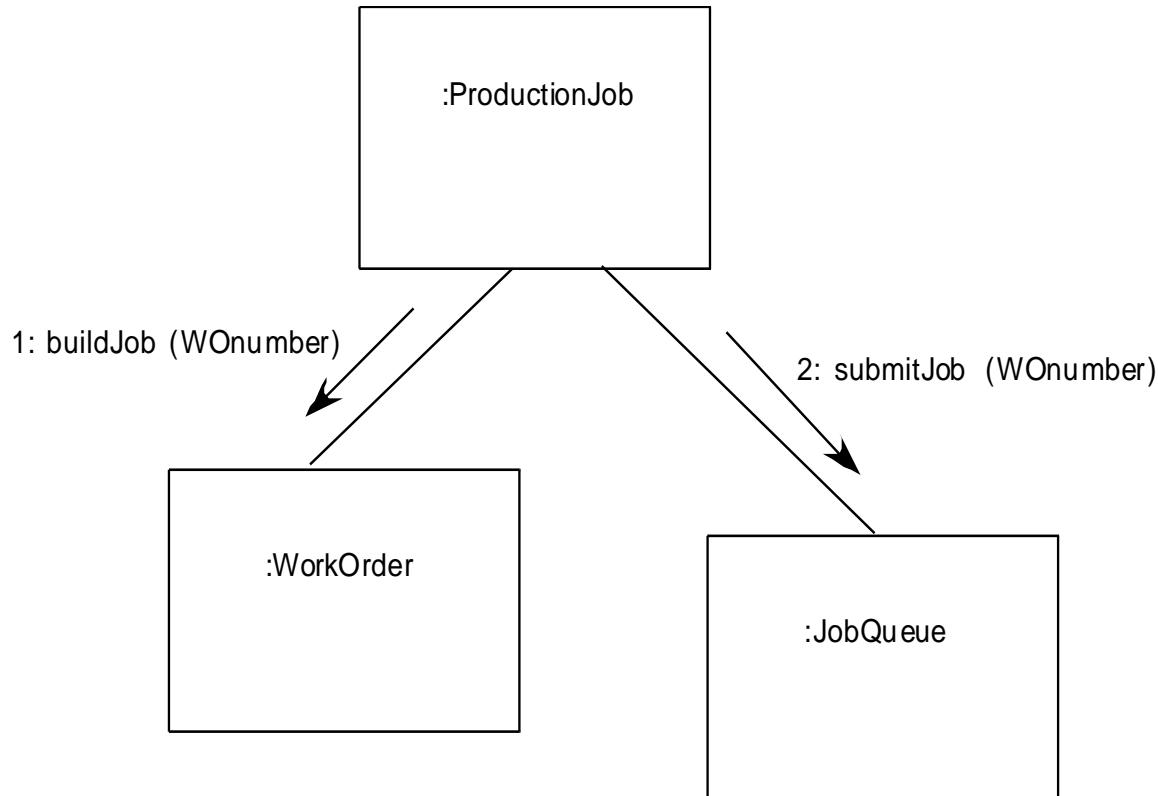
Thiết kế mức thành phần - I

- Bước 1. Xác định tất cả các lớp thiết kế tương ứng với miền vấn đề.
- Bước 2. Xác định tất cả các lớp thiết kế tương ứng với kết cấu hạ tầng.
- Bước 3. Xây dựng tất cả các lớp không có được như là thành phần tái sử dụng.
- Bước 3a. Xác định chi tiết thông điệp khi các lớp hoặc các thành phần cộng tác.
- Bước 3b. Xác định các giao diện thích hợp cho mỗi thành phần.

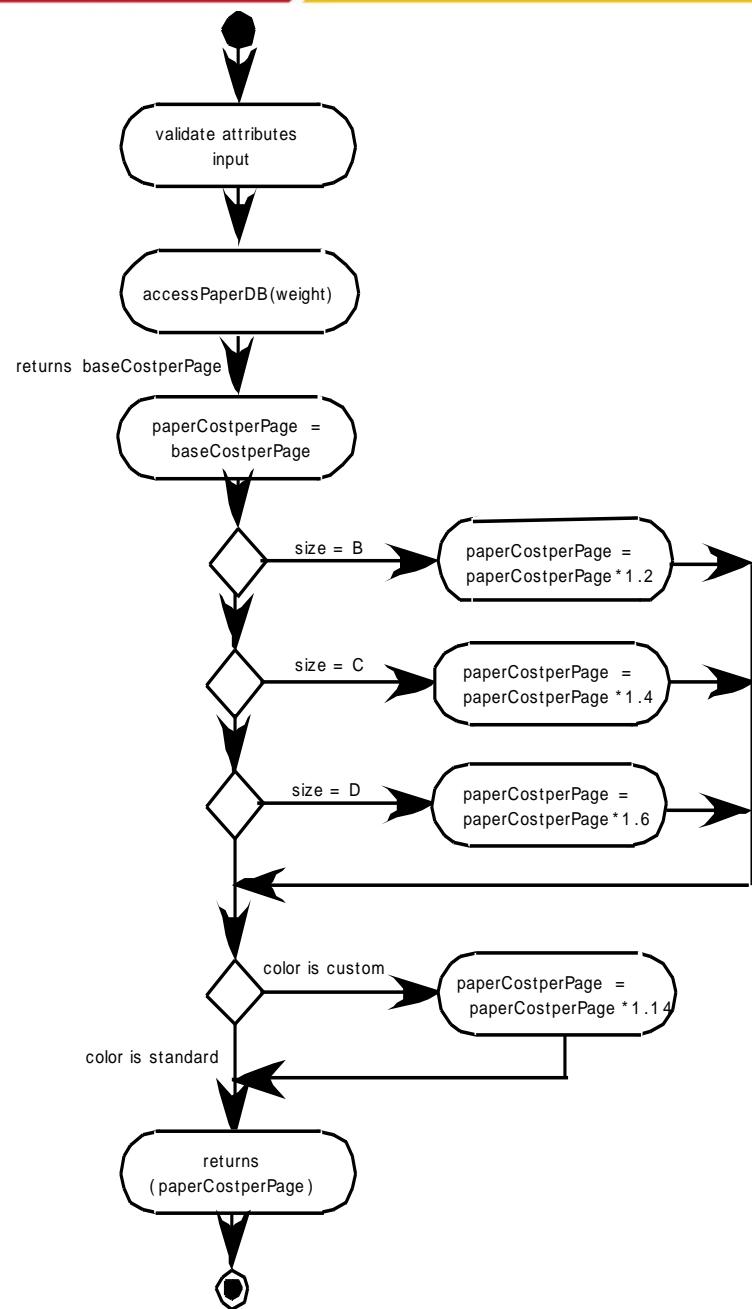
Thiết kế mức thành phần - II

- Step 3c. Xây dựng các thuộc tính và xác định kiểu dữ liệu và cấu trúc dữ liệu cần thiết để thực hiện chúng.
- Step 3d. Mô tả luồng xử lý trong từng hoạt động cụ thể.
- Step 4. Mô tả các nguồn dữ liệu bền vững (cơ sở dữ liệu và các tập tin) và xác định các lớp cần thiết để quản lý chúng.
- Step 5. Phát triển và xây dựng biểu diễn hành vi cho một lớp hoặc một thành phần.
- Step 6. Xây dựng sơ đồ triển khai để cung cấp thêm thông tin về chi tiết thực hiện.
- Step 7. Nhân tố hóa mỗi thể hiện thiết kế mức thành phần và luôn luôn xem xét phương án thay thế.

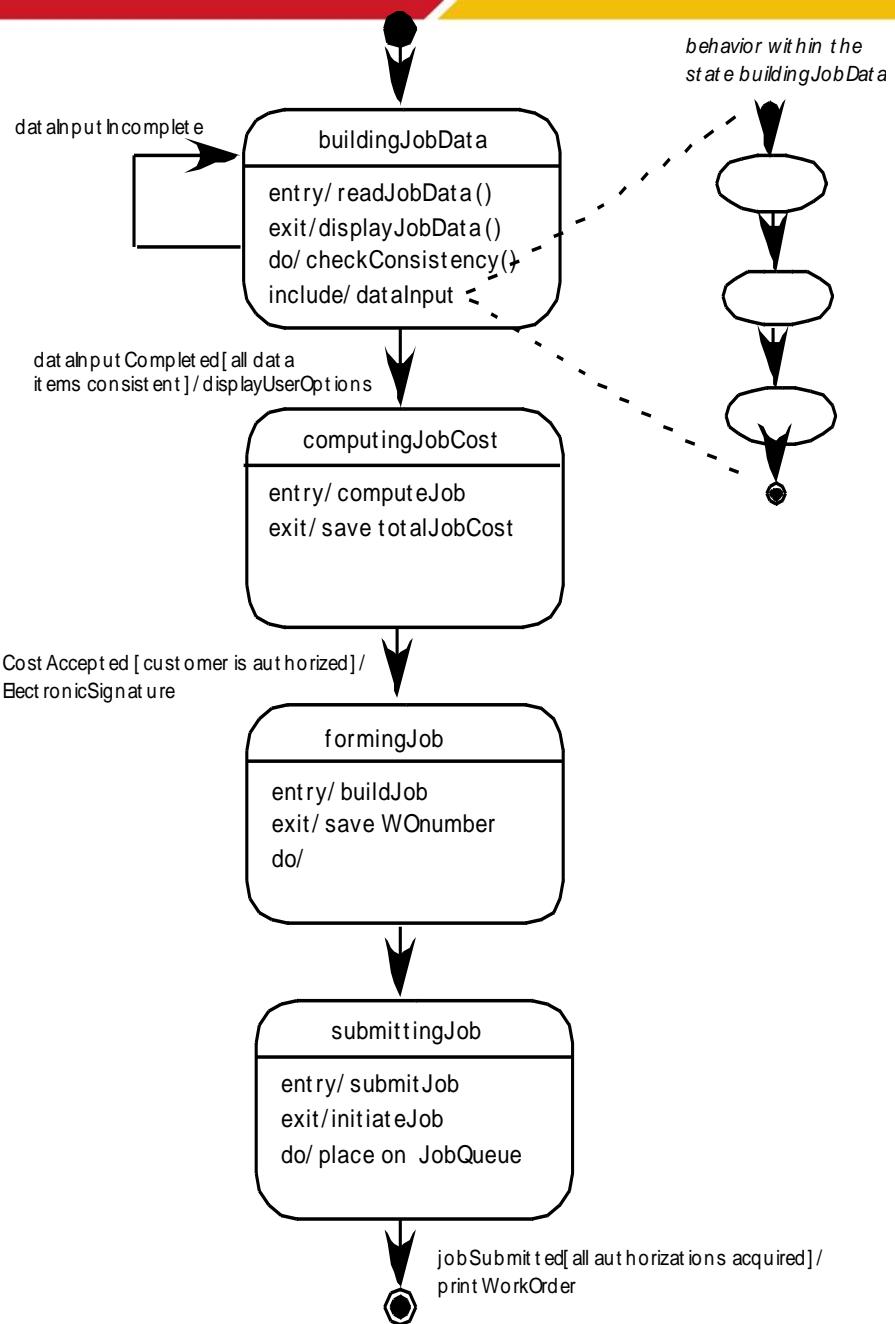
Sơ đồ cộng tác



Sơ đồ hoạt động



Sơ đồ trạng thái



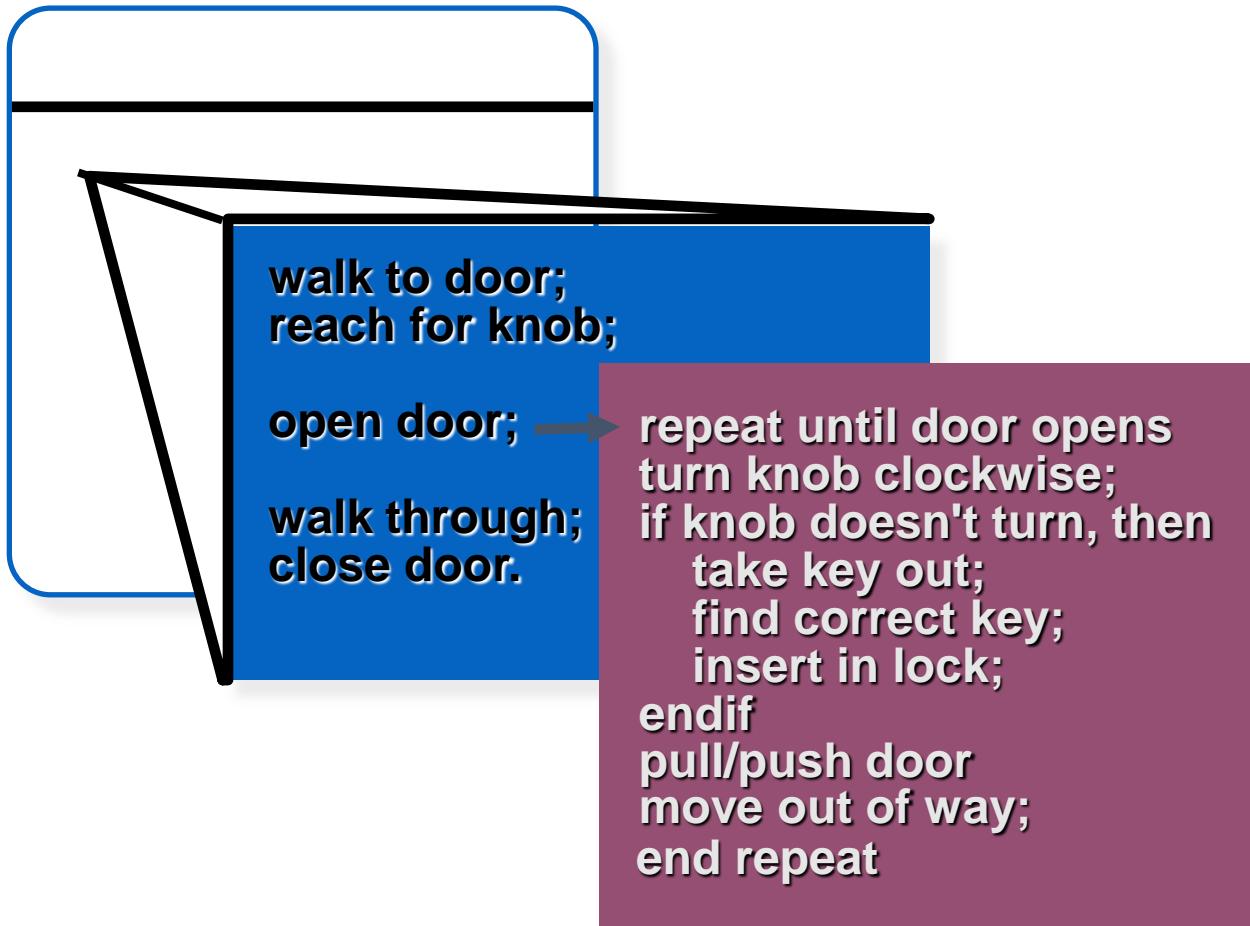
Thiết kế thành phần quy ước

- Việc thiết kế các xử lý logic được quy định bởi các nguyên tắc cơ bản của thiết kế thuật toán và lập trình có cấu trúc.
- Việc thiết kế các cấu trúc dữ liệu được định nghĩa bởi các mô hình dữ liệu được phát triển cho hệ thống.
- Việc thiết kế các giao diện được quy định bởi sự hợp tác mà một thành phần phải có ảnh hưởng.

Thiết kế thuật toán

- Là hoạt động thiết kế sát nhất với việc coding.
- Cách tiếp cận:
 - Xem xét mô tả thiết kế cho các thành phần
 - Sử dụng tinh chỉnh từng bước để phát triển thuật toán
 - Sử dụng lập trình có cấu trúc để thực hiện logic có tính thủ tục
 - Sử dụng ‘phương pháp chính thống’ để chứng minh logic.

Tinh chỉnh từng bước



Mô hình biểu diễn thiết kế thuật toán

- Trình bày các thuật toán ở mức độ chi tiết mà có thể được xem xét lại chất lượng.
- Tùy chọn:
 - Đồ họa (e.g. flowchart, box diagram)
 - Mã giả (e.g., PDL)
 - Ngôn ngữ lập trình
 - Bảng quyết định

Thiết kế dữ liệu

- Tìm kiếm biểu diễn logic cho các phần tử dữ liệu đã được nhận diện trong giai đoạn phân tích yêu cầu.
- Thiết kế các cấu trúc dữ liệu của chương trình và cơ sở dữ liệu
- Các tiêu chí thiết kế dữ liệu
 - Không dư thừa dữ liệu
 - Tối ưu hóa không gian lưu trữ
- Được biểu diễn ở các mức trừu tượng khác nhau

Một số nguyên tắc thiết kế dữ liệu

- Nhận diện cả cấu trúc dữ liệu và tác vụ truy xuất
- Chú ý sử dụng từ điển dữ liệu
- Trì hoãn thiết kế dữ liệu mức thấp cho đến cuối giai đoạn này
- Che giấu biểu diễn bên trong của cấu trúc dữ liệu
- Nên áp dụng kiểu ADT trong thiết kế cũng như trong lập trình

Nội dung

1. Tổng quan thiết kế phần mềm
2. Các khái niệm trong thiết kế phần mềm
3. Thiết kế kiến trúc phần mềm
4. Thiết kế chi tiết phần mềm
5. **Tính móc nối (Coupling) và tính kết dính (Cohesion)**

Chất lượng của thiết kế

- Thế nào là một thiết kế tốt?
 - Easy for Developing, Reading & Understanding
 - Easy for Communication
 - Easy for Extending (add new features)
 - Easy for Maintenance

Chất lượng của thiết kế

- Mỗi module có tính cố kết cao (high cohesion)
 - Mỗi module là một đơn vị logic
 - Toàn bộ module cùng đóng góp thực hiện một mục tiêu
- Liên kết lỏng lẻo (loose coupling) giữa các module
 - Ít ràng buộc, ít phục thuộc lẫn nhau
- Dễ hiểu
- Định nghĩa rõ ràng
 - Các module và quan hệ giữa chúng

Cohesion và Coupling

- Cohesion
 - Gắn kết đề cập đến mức độ mà các phần tử của một mô-đun thuộc về nhau. Tính gắn kết là thước đo mức độ liên quan hoặc tập trung của các trách nhiệm của một mô-đun đơn lẻ.
- Coupling
 - Ghép nối / Khớp nối hoặc phụ thuộc là mức độ mà mỗi chương trình mô-đun phụ thuộc vào từng mô-đun khác.

“Loose coupling and high cohesion”

Sự gắn kết (cohesion)

- Góc nhìn quy ước:
 - “Tư duy đơn lẻ” của một mô-đun: mục tiêu thiết kế chung của việc tách các mối quan tâm cho thấy một mô-đun nên giải quyết một nhóm mối quan tâm duy nhất
- Góc nhìn hướng đối tượng:
 - Sự gắn kết nghĩa là một thành phần hoặc một lớp chỉ đóng gói các thuộc tính và hoạt động liên quan chặt chẽ với nhau và với bản thân thành phần hoặc lớp đó.

Sự gắn kết (cohesion)

- Các mức của sự gắn kết:

- Chức năng
- Tầng
- Truyền thông
- Liên tục
- Thủ tục
- Phụ thuộc thời gian
- Lợi ích



Ghép nối (coupling)

- Góc nhìn quy ước:
 - Là mức độ mà một thành phần kết nối với các thành phần khác và với thế giới bên ngoài.
- Góc nhìn hướng đối tượng:
 - Một thước đo chất lượng mức độ kết nối của các lớp với lớp khác.
- Các mức của ghép nối:
 - Nội dung
 - Chung
 - Điều khiển
 - Đánh dấu
 - Dữ liệu
 - Lời gọi công thức
 - Loại sử dụng
 - Sự lồng ghép và bao hàm



Tổng kết

- Tầm quan trọng của thiết kế phần mềm có thể được phát biểu bằng một từ “**chất lượng**”.
- *Thiết kế là nơi chất lượng phần mềm được nuôi dưỡng trong quá trình phát triển*: cung cấp cách biểu diễn phần mềm có thể được xác nhận về chất lượng, là cách duy nhất mà chúng ta có thể chuyển hóa một cách chính xác các yêu cầu của khách hàng thành sản phẩm hay hệ thống phần mềm cuối cùng.
- Thiết kế phần mềm là **công cụ giao tiếp** làm cơ sở để có thể mô tả một cách đầy đủ các dịch vụ của hệ thống, để quản lý các rủi ro và lựa chọn giải pháp thích hợp.

Tổng kết (2)

- Thiết kế phần mềm phục vụ như một **nền tảng cho mọi bước kỹ nghệ phần mềm và bảo trì**.
- Không có thiết kế có nguy cơ sản sinh một hệ thống không ổn định - một hệ thống sẽ thất bại.
- Một hệ thống phần mềm rất khó xác định được chất lượng chừng nào chưa đến bước kiểm thử. **Thiết kế tốt là bước quan trọng đầu tiên để đảm bảo chất lượng phần mềm.**



25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank
you for
your
attentions
!

