Chapter 8

- **Design Concepts**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

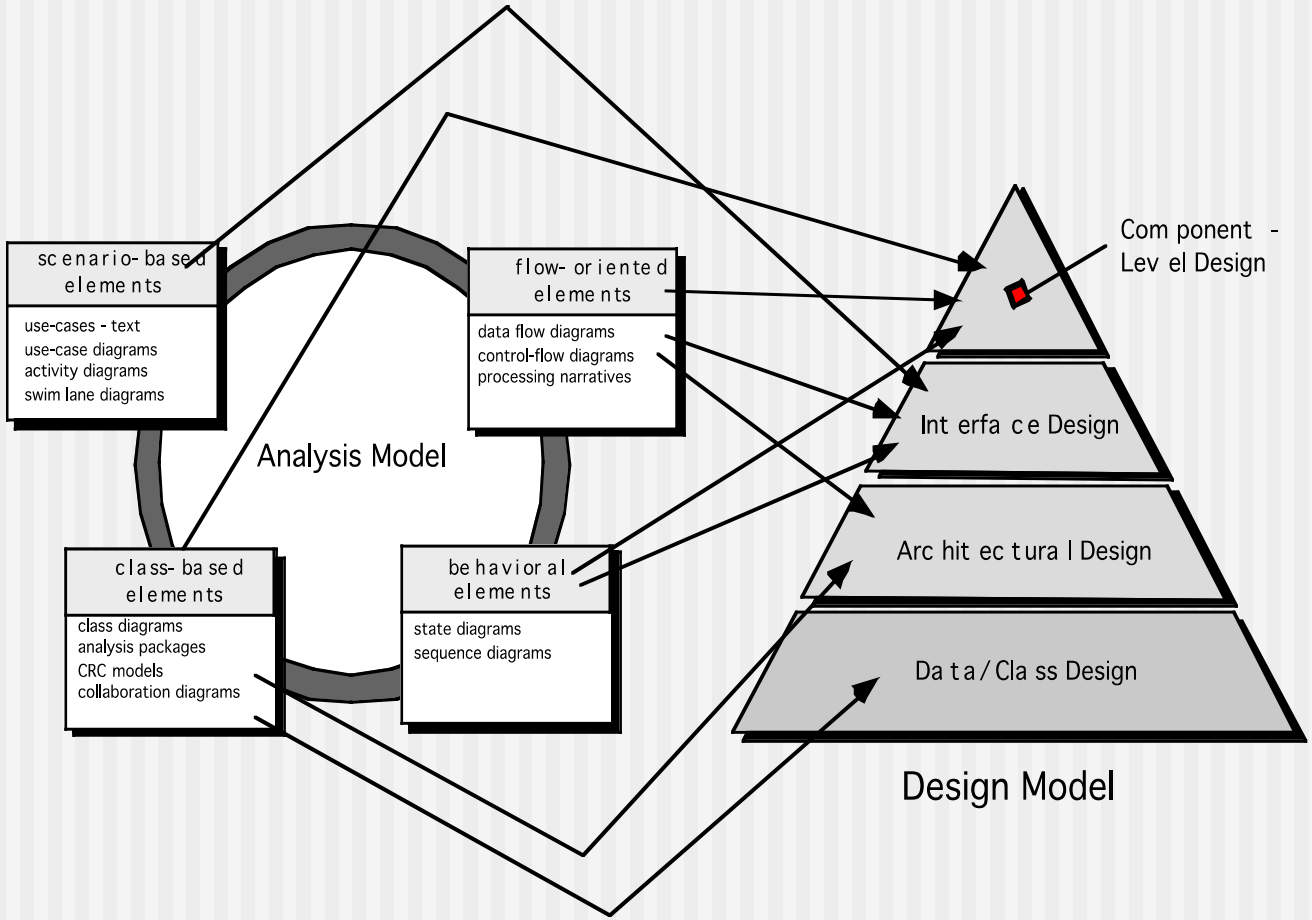
Chapter 8

For non-profit educational use only

Design

- 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.

Analysis Model -> Design Model
(Mô hình phân tích-> Mô hình thiết kế)



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.

Nguyên tắc 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.
- 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 hoá 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.

Nguyên tắc thiết kế

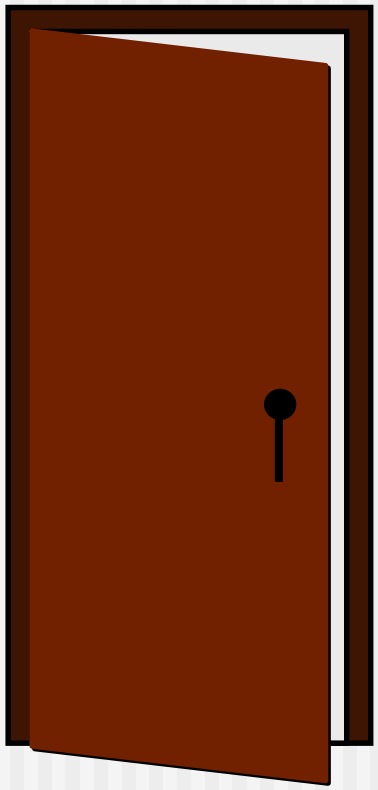
- Quá trình thiết kế không nên mắc phải ‘tunnel vision.’
- Việc thiết kế nên có thể truy ngược về mô hình phân tích.
- Việc thiết kế không nên ‘phát minh lại bánh xe’.
- Việc thiết kế nên "giảm thiểu khoảng cách trí tuệ" [DAV95] giữa phần mềm và bài toán như nó tồn tại trong thế giới thực.
- Việc thiết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

From Davis [DAV95]

Các khái niệm cơ bản

- Trừu tượng hoá—dữ liệu, thủ tục, kiểm soát
- Kiến trúc—cấu trúc tổng thể của phần mềm
- Patterns—"chuyển tải những tinh túy" của một giải pháp thiết kế đã được chứng minh
- Separation of concerns—bất kỳ vấn đề phức tạp có thể được xử lý dễ dàng hơn nếu nó được chia thành nhiều mảnh
- Modularity—mô đun hoá các dữ liệu và chức năng
- Tính ẩn—giao diện được điều khiển
- 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—Phụ lục II
- Thiết kế Lớp—cung cấp chi tiết thiết kế mà sẽ cho phép các lớp đã phân tích được thực thi

Trừu tượng dữ liệu

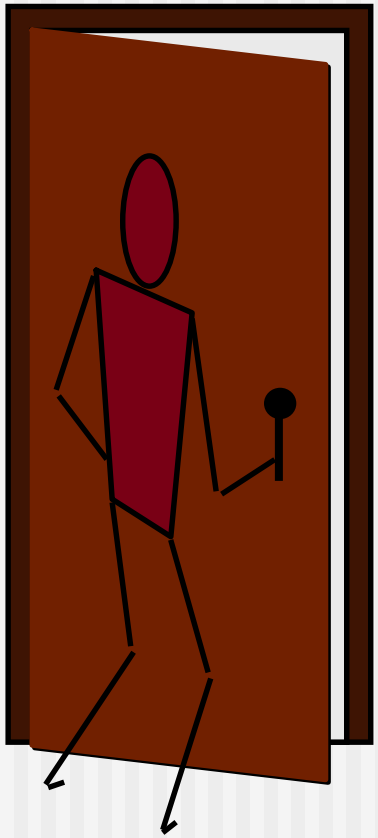


Cửa

nhà sản xuất
model số
Loại
Hướng xoay
chèn
đèn
  loại
  số lượng
Khối lượng
Cách mở

Thực thi như một cấu trúc dữ liệu

Trừu tượng thủ tục



Mở

chi tiết về
Thuật toán vào cửa

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

Kiến trúc

“Cấu trúc tổng thể của phần mềm và cách thức mà cấu trúc cung cấp tính toàn vẹn khái niệm cho một hệ thống.” [SHA95a]

Tính cấu trúc. Khía cạnh này của các đại diện thiết kế kiến trúc xác định các thành phần của một hệ thống (ví dụ, mô-đun, các đối tượng, các bộ lọc) và cách thức mà những thành phần này được đóng gói và tương tác với nhau. Ví dụ, đối tượng được đóng gói cả dữ liệu và việc xử lý các thao tác dữ liệu và tương tác thông qua việc gọi các phương pháp
Tính thêm chức năng. Các mô tả thiết kế kiến trúc nên giải quyết cách kiến trúc thiết kế đạt yêu cầu về hiệu suất, công suất, độ tin cậy, an toàn, khả năng thích ứng, và các đặc điểm khác của hệ thống.
Họ các hệ thống liên quan. Thiết kế kiến trúc sẽ dựa trên mô hình lặp lại mà thường gặp trong thiết kế của các họ của các hệ thống tương tự. Về bản chất, các thiết kế cần phải có khả năng sử dụng lại các khối xây dựng kiến trúc.

Patterns(mẫu)

Bản mẫu thiết kế pattern

***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 (sư công 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**

Phân tách các Mối quan tâm

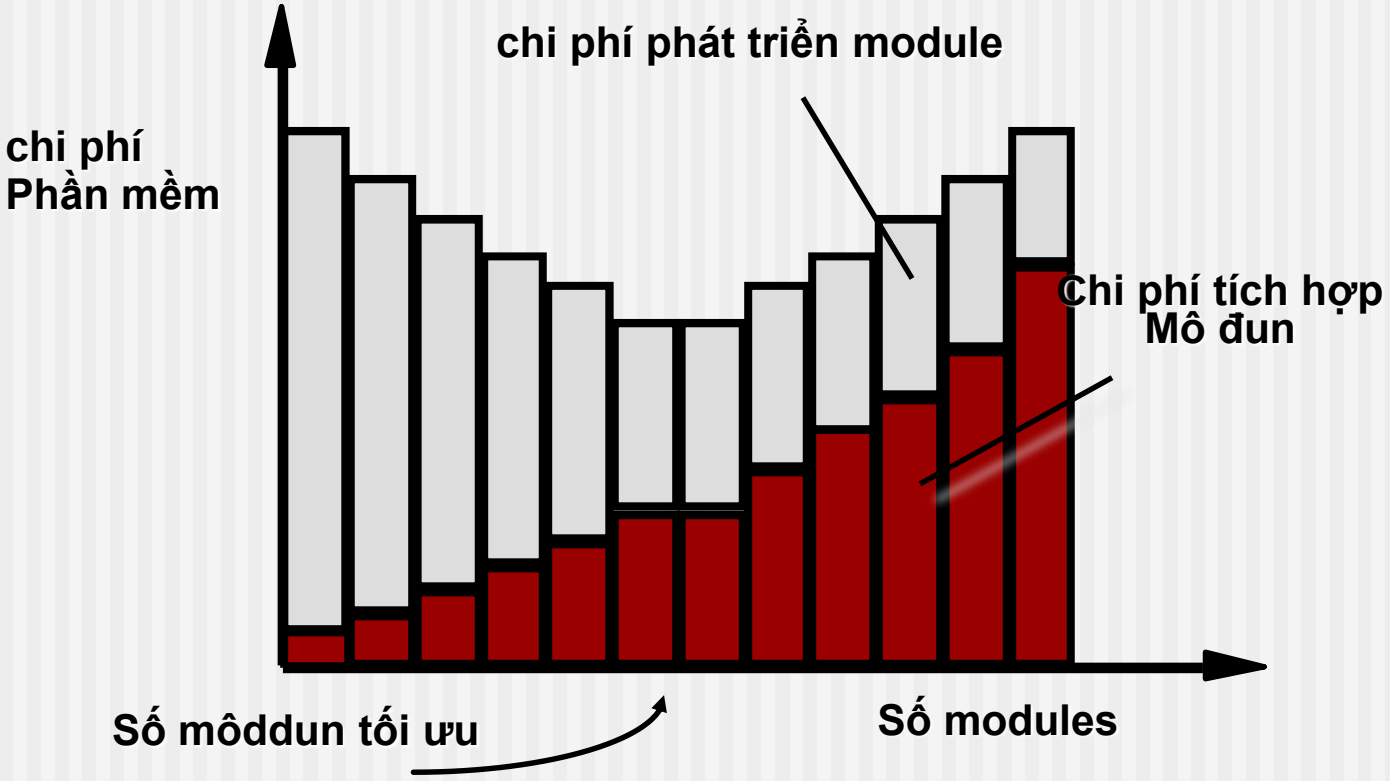
- Bất kỳ vấn đề phức tạp có thể được xử lý dễ dàng hơn nếu nó được chia thành từng mảnh mà mỗi mảnh có thể được giải quyết và / hoặc tối ưu hóa một cách độc lập
- Mối quan tâm là một tính năng hoặc hành vi được quy định như là một phần của mô hình yêu cầu cho các phần mềm
- Bằng cách tách mối quan tâm 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

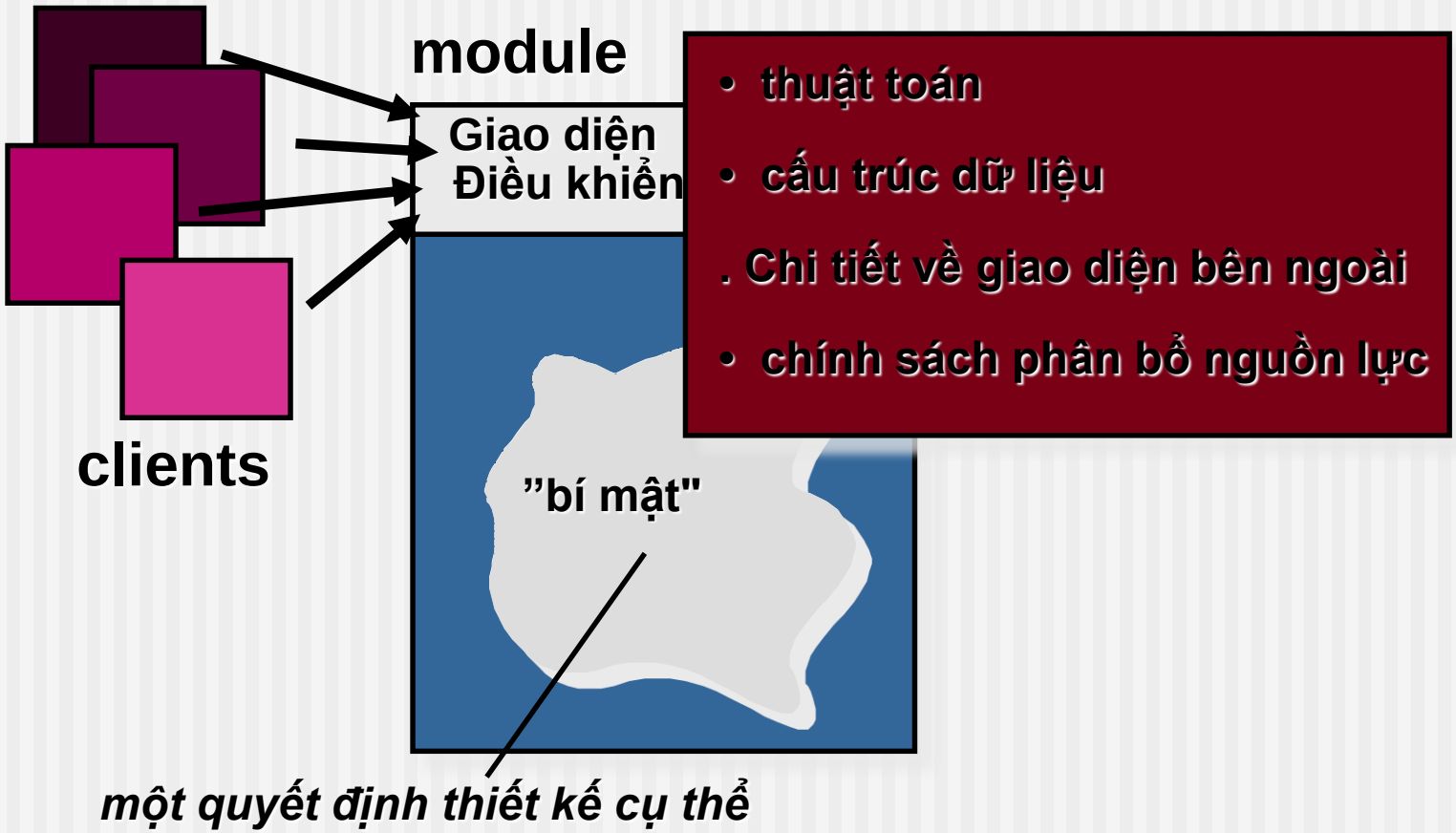
- “Mô đun là thuộc tính duy nhất của phần mềm cho phép một chương trình có thể quản lý một cách thông minh" [Mye78].
- Phần mềm nguyên khối (ví dụ, một chương trình lớn gồm một mô-đun duy nhất) có thể không được dễ dàng nắm bắt được bởi một kỹ sư phần mềm.
  - Số lượng các đường dẫn điều khiển, khoảng 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?



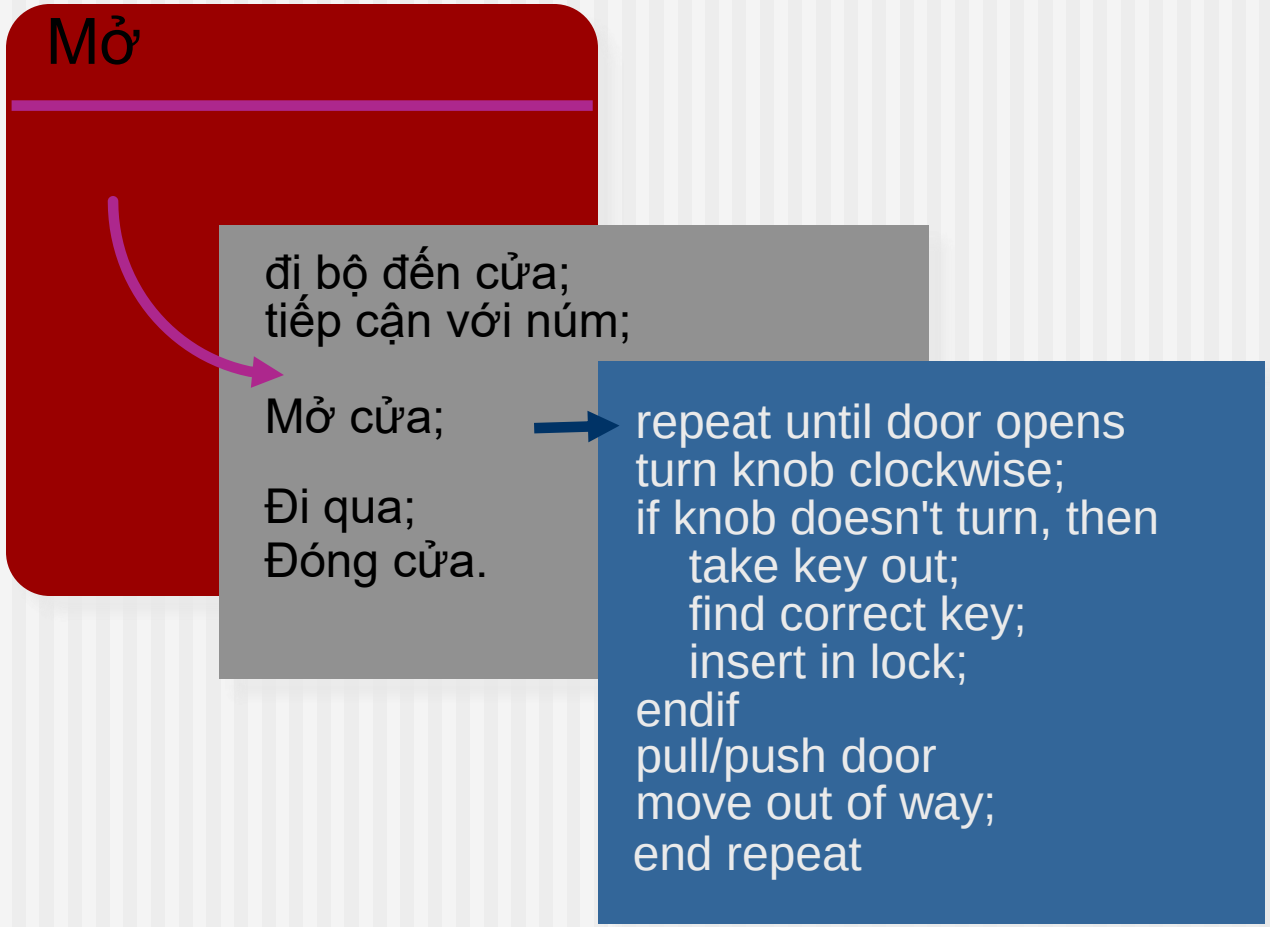
Ẩn thông tin



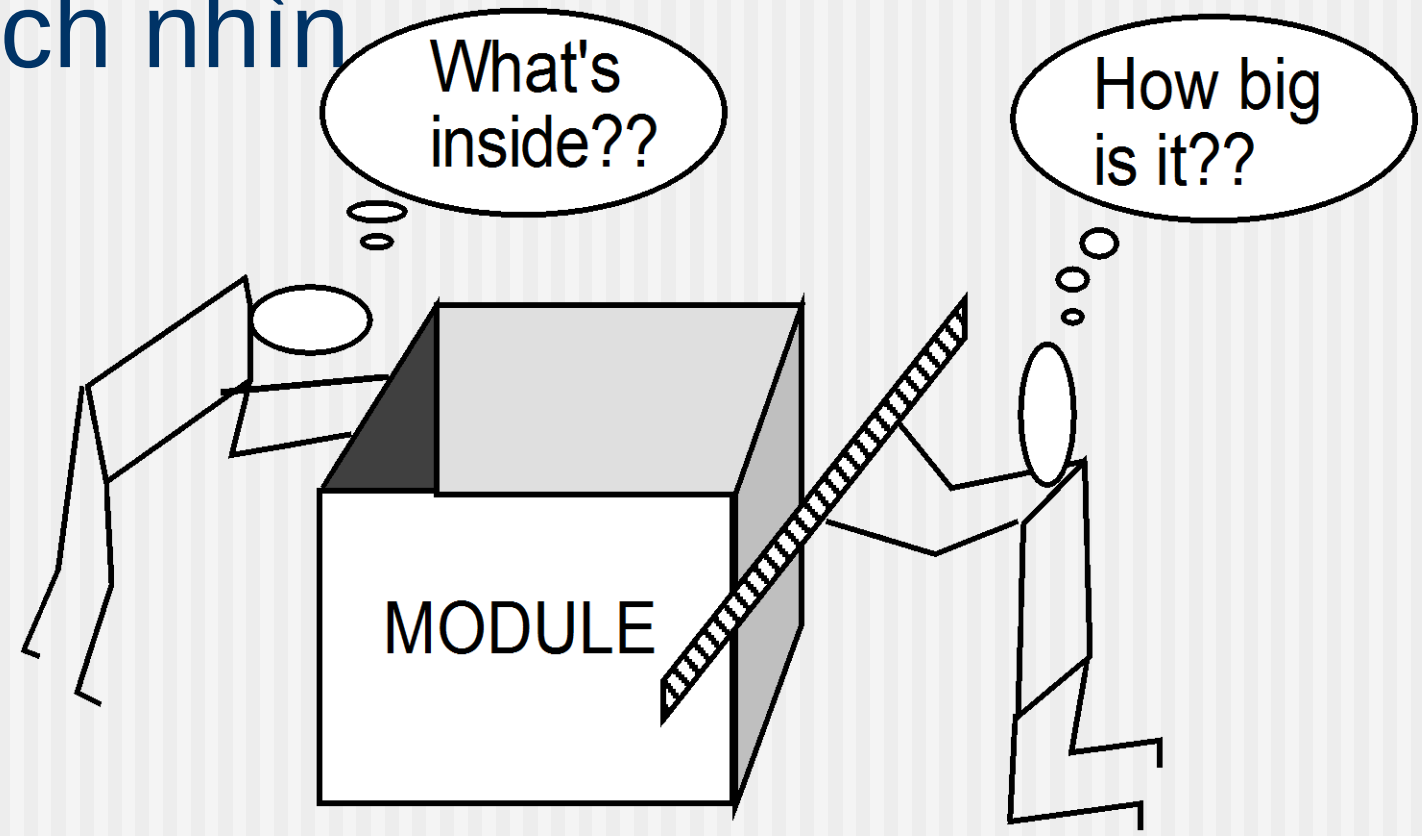
Tại sao lại Ẩn thông tin?

- 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 trong phần mềm chất lượng cao

Sàng lọc theo từng bước



Định kích thước mô đun: Hai cách nhìn



Tính độc lập chức năng

- Tính độc lập chức đạt được bằng cách phát triển các module với chức năng đơn lẻ và một "ác cảm" với tương tác quá mức với các module khác.
- *Sự gắn kết* là một dấu hiệu của sức mạnh chức năng tương đối của một module.
- Một module gắn kết thực hiện một nhiệm vụ duy nhất, đòi hỏi ít tương tác với các thành phần khác trong các phần khác của chương trình. Nói đơn giản, một mô-đun gắn kết nên (lý tưởng) làm chỉ là một việc.
- *Ghép nối* là một dấu hiệu của sự phụ thuộc lẫn nhau tương đối giữa các mô-đun.
- Ghép nối phụ thuộc vào độ phức tạp giao diện giữa các module, các điểm mà tại đó entry hoặc tham chiếu được thực hiện cho một mô-đun, và những dữ liệu truyền qua giao diện.

Các khía cạnh

- Hãy xem xét hai yêu cầu, A và B. Yêu cầu A giao nhau với yêu cầu B "nếu một phân tách phần mềm [tinh chế] được lựa chọn trong đó B không thể làm hài lòng mà không cần dùng A.[Ros04]
- Một *khía cạnh* là một đại diện của một mối quan tâm giao nhau.

Các khía cạnh—Ví dụ

- Hãy xem xét hai yêu cầu với SafeHomeAssured.com WebApp. Yêu cầu A được mô tả qua các trường hợp sử dụng camera giám sát truy cập thông qua Internet. Một tinh chỉnh thiết kế sẽ tập trung vào những mô-đun có thể cho phép một người sử dụng đăng ký để truy cập video từ camera đặt khắp không gian. Yêu cầu B là một yêu cầu an ninh chung chung mà nói rằng một người sử dụng đăng ký phải được xác nhận trước khi sử dụng SafeHomeAssured.com. Yêu cầu này được áp dụng cho tất cả các chức năng có sẵn cho người dùng SafeHome đăng ký. Vì tinh chỉnh thiết kế xảy ra, A * là một đại diện thiết kế cho yêu cầu A và B * là một đại diện thiết kế cho yêu cầu B. Vì vậy, A * và B * là đại diện của các mối quan tâm, và B * chéo cắt A *.
- Một khía cạnh là một đại diện của một mối quan tâm xuyên suốt. Do đó, các đại diện thiết kế, B *, các yêu cầu, một người sử dụng được đăng ký phải được xác nhận trước khi sử dụng SafeHomeAssured.com, là một khía cạnh của SafeHome WebApp.

Tái cấu trúc

- 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.

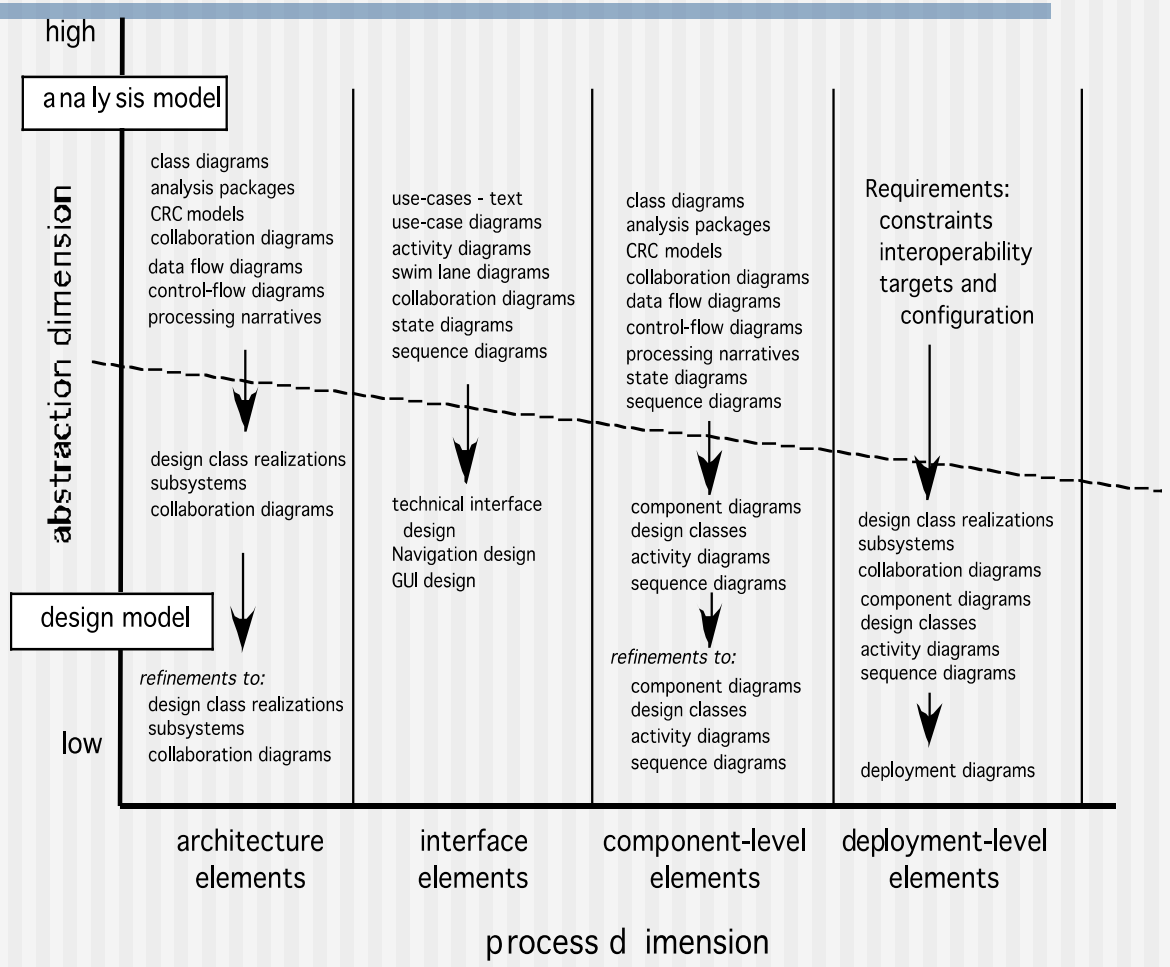
OO Các khái niệm thiết kế

- 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.

Thiết kế các lớp

- 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 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.

Mô hình thiết kế



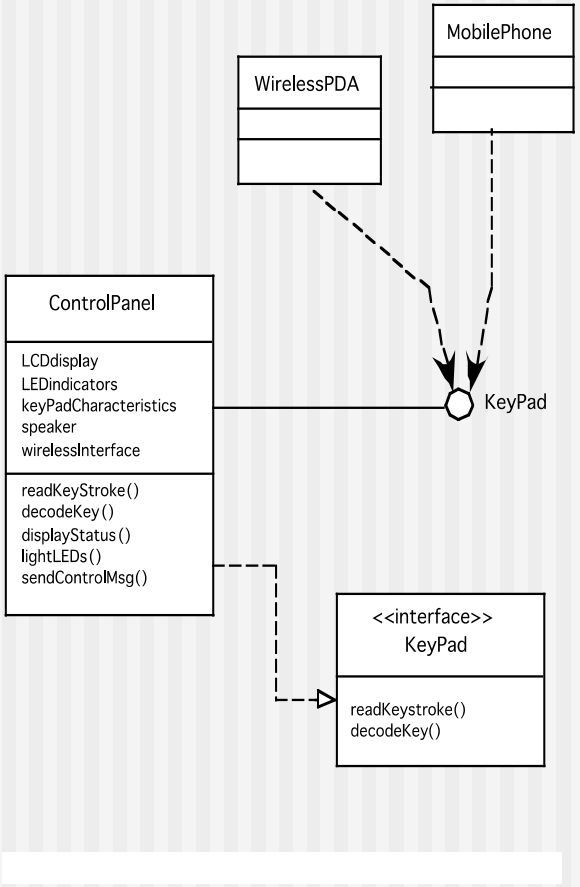
Các thành phần Mô hình thiết kế

- 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
  - Patterns và“kiểu” (Chapters 9 and 12)
- 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

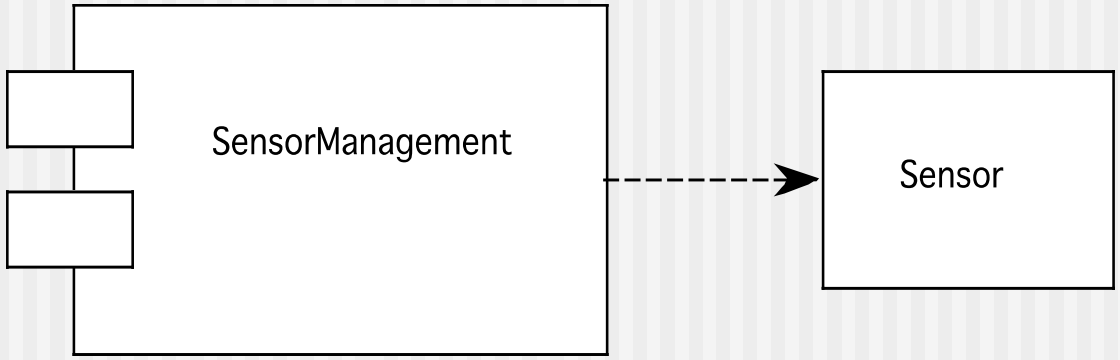
Các thành phần kiến trúc

- 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 (Chương 12) và các kiểu (Chương 9).

Các thành phần giao diện



Component Elements



Các thành phần triển khai