

Chương 5:

THIẾT KẾ VÀ XÂY DỰNG PHẦN MỀM



Mục tiêu

- Nắm vững quy trình thiết kế phần mềm để hướng tới bản thiết kế có chất lượng.
- Nắm vững các nguyên lý trong lập trình để tạo chương trình có chất lượng.

Nội dung chính

1. Tổng quan về thiết kế phần mềm
2. Quy trình thiết kế phần mềm
3. Cài đặt chương trình

1. Tổng quan về thiết kế phần mềm

- Mục đích:
 - Trả lời câu hỏi how ~ tìm giải pháp cho các vấn đề cần giải quyết bởi dự án
 - Kết quả thiết kế
 - Bản thiết kế (thiết kế mức cao – bản kiến trúc; thiết kế mức thấp – bản cấu trúc)
 - => Kiến trúc, cấu trúc ~ sự chia nhỏ
- ⇒ Giải pháp thiết kế thường dựa trên cách tiếp cận “chia để trị”
 - ~ Giải quyết vấn đề lớn thông qua việc giải quyết các vấn đề được chia nhỏ và sau đó kết hợp lại

1. Tổng quan về thiết kế phần mềm

- Giải pháp thiết kế dựa trên cách tiếp cận chia để trị
 - => ? Chia ntn để đảm bảo chất lượng thiết kế
 - Giảm chi phí, thời gian phát triển;
 - **Dễ hiểu**, dễ bảo trì, nâng cấp;
 - Hướng tới những thành phần sẵn dùng
 - Dễ lập lịch biểu, // hóa các hoạt động nhiều nhất có thể
 - **Tính kết dính thấp, tính cô kết cao.**
 - ...

1. Tổng quan về thiết kế phần mềm

- **Một số độ đo chất lượng thiết kế cơ bản**
 1. Coupling (ghép nối);
 2. Cohension (kết dính/cố kết);
 3. Understandability (dễ hiểu).

1. Tổng quan về thiết kế phần mềm

a. Tính ghép nối (*Coupling*)

- Đo sự liên kết (trao đổi dữ liệu) giữa các mô đun.
 - Ghép nối càng chặt -> chất lượng thiết kế càng thấp.
- Các mức độ ghép nối:
 - Thể hiện qua loại hình ghép nối.

Loại hình ghép nối

Ghép nối thường	✦	Loose and best
Ghép nối dữ liệu	✦	Good
Ghép nối nhân	✦	Ok
Ghép nối điều khiển	✦	Ok
Ghép nối chung	✦	Very bad
Ghép nối nội dung	✦	Tight and Worst

a. Tính ghép nối (*Coupling*)

- **Ghép nối nội dung:**

- Các mô đun dùng chung dữ liệu và điều khiển của nhau.
- Là trường hợp xấu nhất.

- Ví dụ:

- Các ngôn ngữ bậc thấp chỉ dùng biến chung,
- Lạm dụng lệnh goto trong chương trình.

a. Tính ghép nối (*Coupling*)

Ví dụ:

Mô đun 1:

10 $k=1$

20 *gosub* 100

>120 *goto* 60

$K+1$

50 *goto* 20

Mô đun 2

100 $Y=3*k*k+7*k-3$

110 *return*



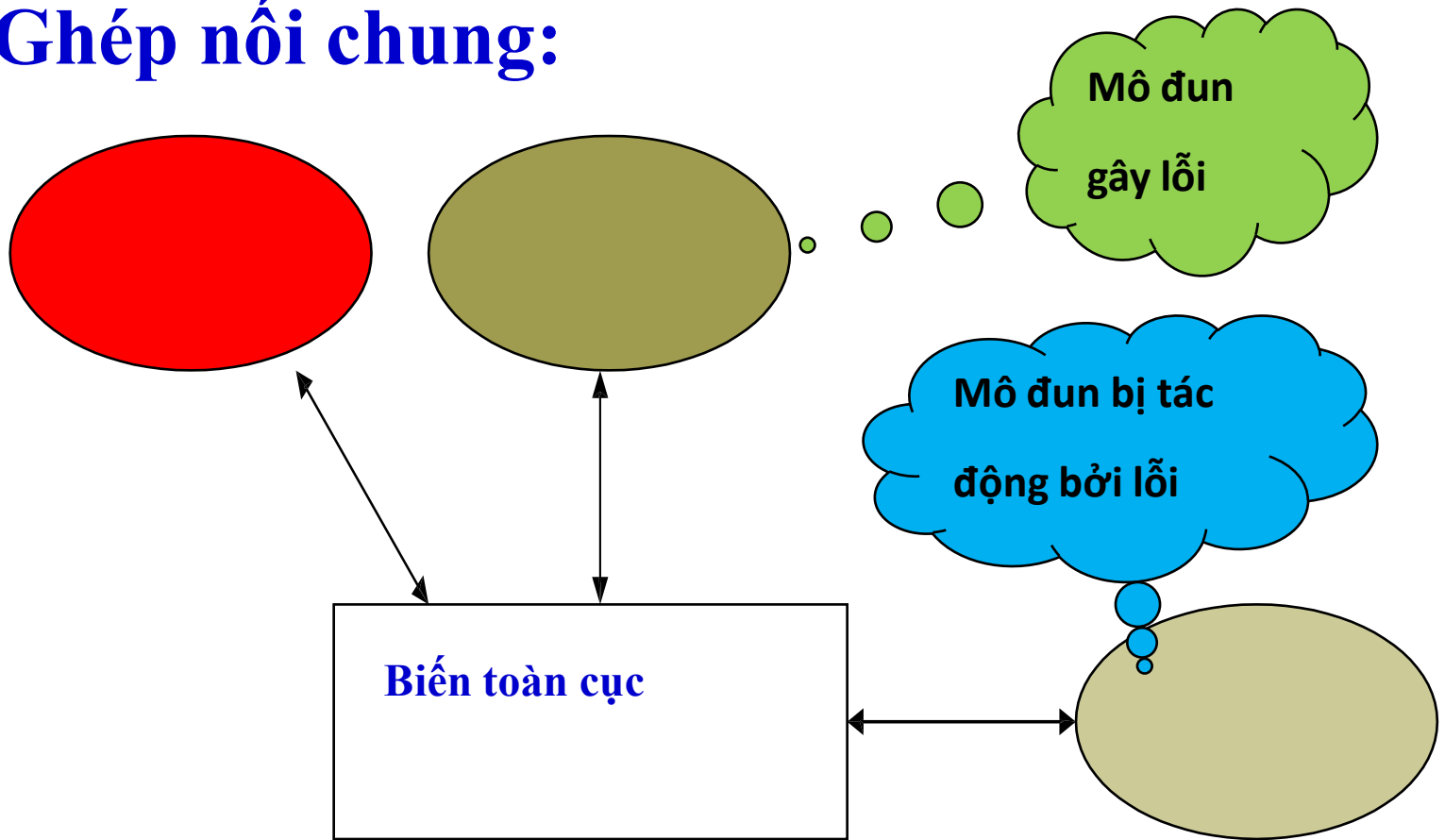
a. Tính ghép nối (*Coupling*)

■ Ghép nối chung:

- Các mô đun dùng chung dữ liệu qua biến toàn cục.
 - => *lỗi của mô đun này gây ra cho biến toàn cục có thể ảnh hưởng đến hoạt động của mô đun khác.*
- Ví dụ:

a. Tính ghép nối (*Coupling*)


Ghép nối chung:



a. Tính ghép nối (*Coupling*)

■ Ghép nối điều khiển:

- Các mô đun/thành phần trao đổi thông tin điều khiển cho nhau (*Lưu ý: mô đun/thành phần cùng mức*).
- Ví dụ:



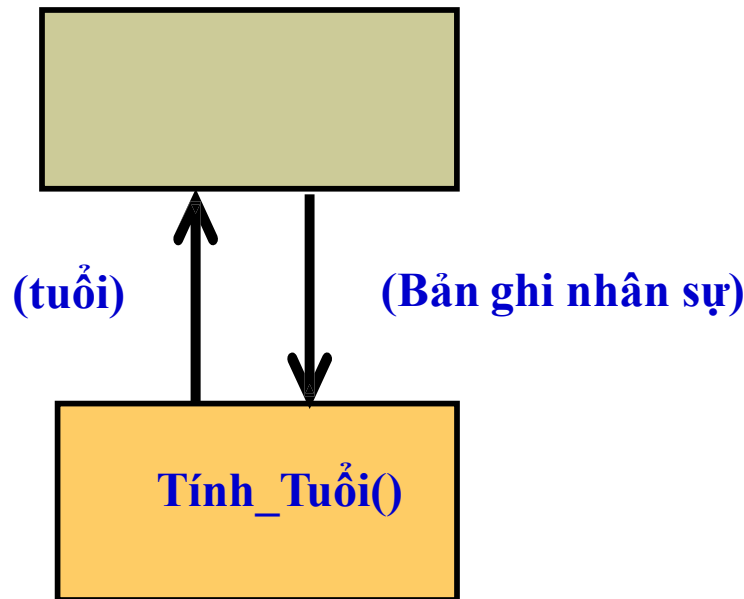
```
Procedure PrintRec  
begin  
  Display Name (name, sex);  
  ....  
end;
```

```
procedure DisplayName (in :name,sex)  
begin  
  if sex = m then  
    print Mr.  
  else  
    print Ms  
    print name  
end;
```

a. Tính ghép nối (*Coupling*)

■ Ghép nối nhân:

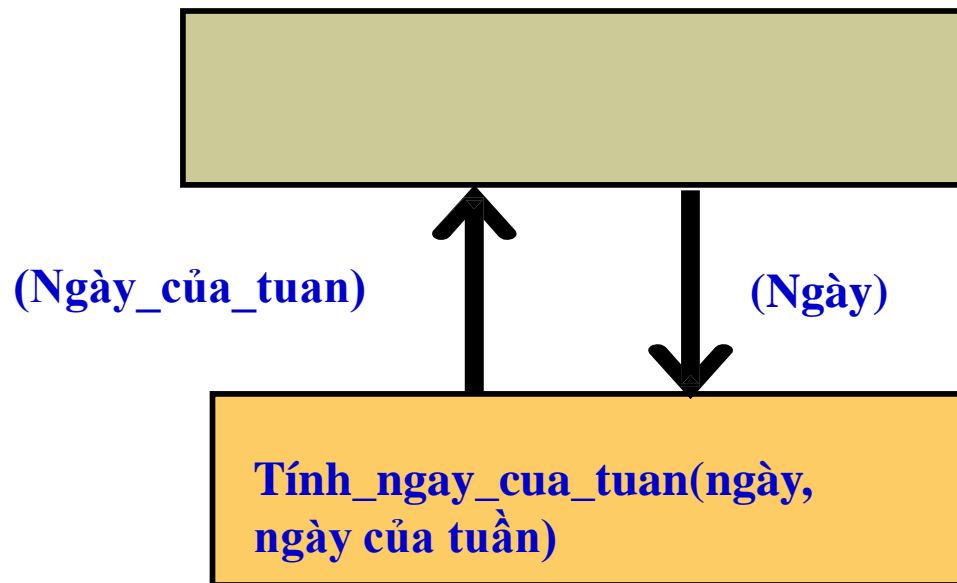
- Các mô đun trao đổi qua nhận lưu thông tin.
- Ví dụ:



a. Tính ghép nối (*Coupling*)

■ Ghép nối dữ liệu:

- Các mô đun truyền dữ liệu qua tham số, nhận kết quả trả về qua tham số.
- Ví dụ:



a. Tính ghép nối (*Coupling*)

■ Ghép nối thường:

- Các mô đun là những công việc hoàn toàn độc lập (ngang mức). Được tích hợp bởi các thành phần mức trên (điều khiển được truyền từ cha xuống)
- Ví dụ: Công việc buổi sáng:
 - *Ngủ_dậy();*
 - *Đánh_răng();*
 - *Rửa_mặt();*
 - *Ăn_sáng();*
 - *Đi_làm();*

b. Cohension (Tính kết dính)

- Đo sự phụ thuộc lẫn nhau giữa các phần tử/câu lệnh trong một module.
 - Độ kết dính cao, chất lượng thiết kế càng tốt
 - Các mức độ kết dính:

b. Cohension (Tính kết dính)

Chức năng	★	High and best
Tuần tự	★	Ok
Truyền thông	★	Ok
Thủ tục	★	Not bad
Thời điểm	★	Not bad
Logic	★	Not bad
Gom góp	★	Lowest and worst by far

b. Cohension (Tính kết dính)

- **Kết dính gom góp (Coincidental cohesion):**
 - Gom các thành phần không liên quan với nhau vào nhau.
 - Ví dụ:
 - Lập trình tuần tự/tuyến tính



b. Cohension (Tính kết dính)

- **Kết dính logic (logical cohesion):**
 - Gồm các thành phần làm chức năng logic tương tự.
 - Ví dụ:
 - Hàm xử lý các lỗi chung



b. Cohension (Tính kết dính)

- **Kết dính thời điểm/tạm thời (temporal cohesion):**
 - Gom các thành phần hoạt động cùng thời điểm.
 - Ví dụ:
 - Hàm khởi tạo, đọc dữ liệu, cấp phát bộ nhớ



b. Cohension (Tính kết dính)

- **Kết dính thủ tục (procedural cohesion):**
 - Các thành phần thực hiện theo một thứ tự xác định.
 - Ví dụ:
 - Hàm tính lương cơ bản, tính phụ cấp, tính bảo hiểm.



b. Cohension (Tính kết dính)

- **Kết dính truyền thông (communicational cohesion):**
 - Các thành phần truy cập đến cùng tập dữ liệu.
 - Ví dụ
 - Tính toán thống kê (tính max, tính min, mean, variation, ..)



b. Cohension (Tính kết dính)

■ Kết dính tuần tự (Sequential cohesion):

- Đầu ra của một thành phần là đầu vào của thành phần tiếp theo.
- Ví dụ:
 - Các hàm biến đổi từ ảnh màu -> ảnh đen trắng -> ảnh nén.



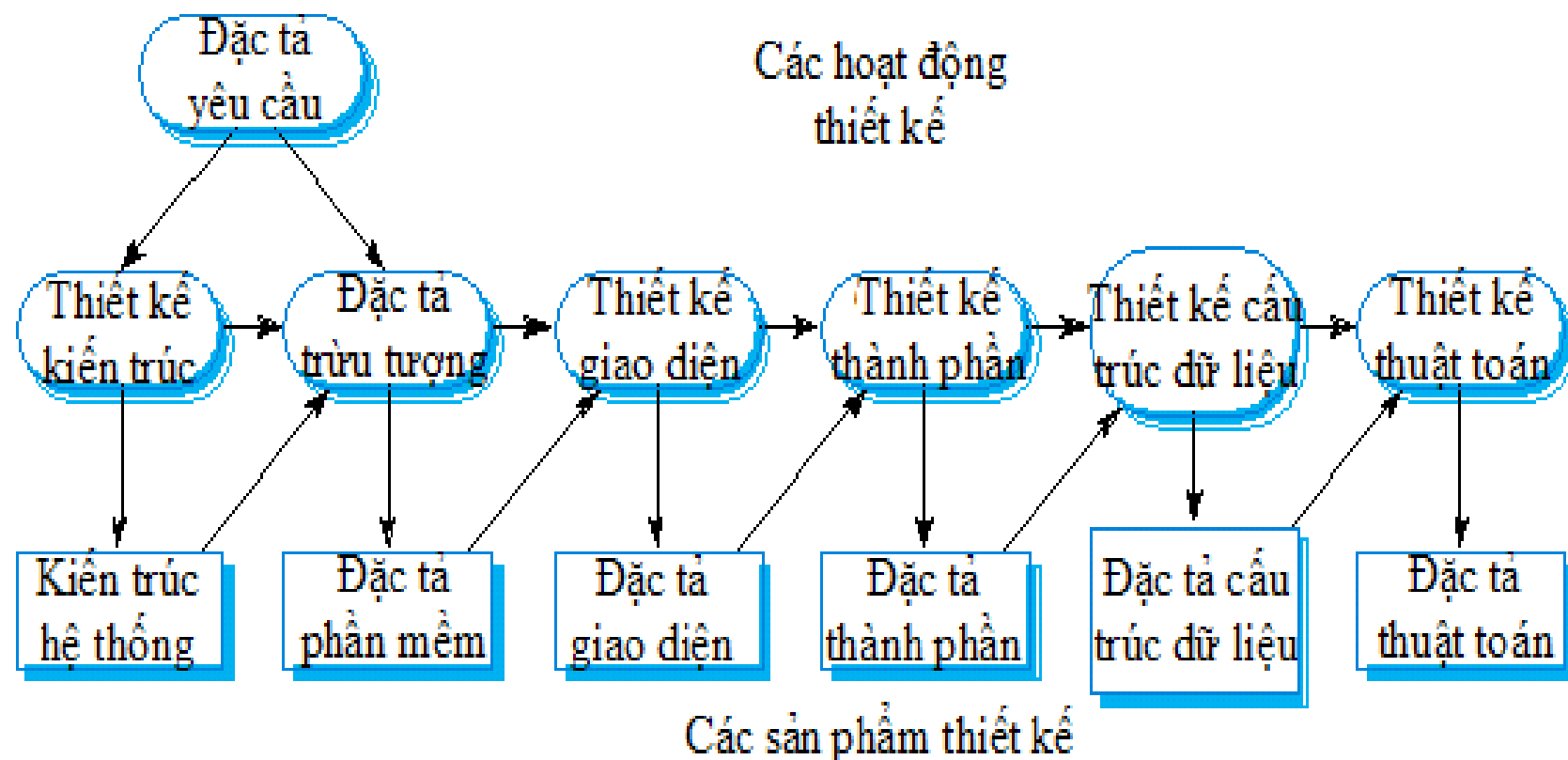
b. Cohension (Tính kết dính)

- **Kết dính chức năng (Functional cohesion):**
 - Các thành phần cùng góp phần thực hiện một chức năng.
 - Ví dụ:
 - Các thao tác trong mô đun sắp xếp
 - *Không thể thiếu thao tác nào.*
 - *Đều cùng chung nhiệm vụ sắp xếp*

c. Understandability

- Khả năng hiểu:
 - Kiến trúc hệ thống, tài liệu rõ ràng
 - Thuật toán, cấu trúc dữ liệu dễ hiểu.

2. Quy trình thiết kế phần mềm



Hình 2.7: Mô hình chung của quy trình thiết kế

2. Quy trình thiết kế phần mềm

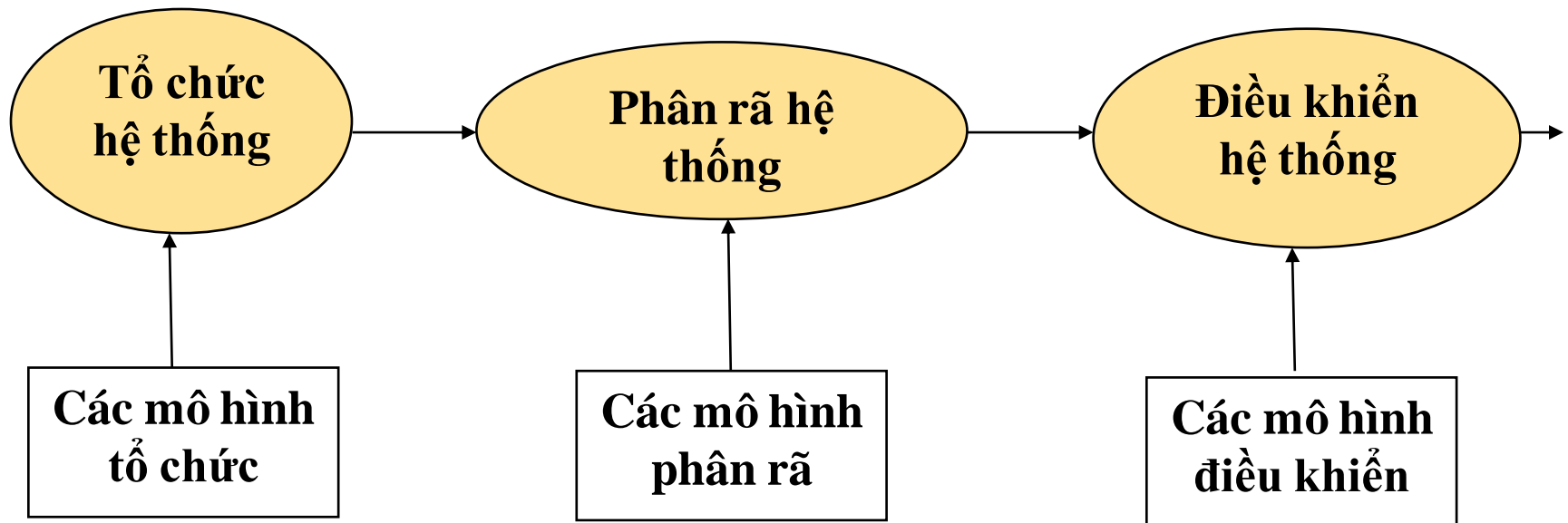
- Có thể tùy biến quy trình thiết kế chung
 - Thiết kế là hoạt động lặp lại
 - Tinh chế và hoàn thiện dần các kết quả TK.

a. Thiết kế kiến trúc

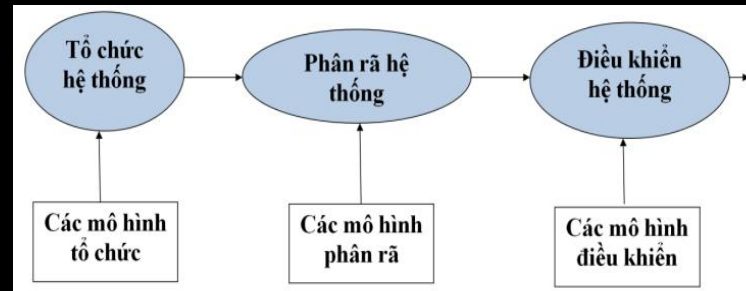
- Kiến trúc
 - ~ Cấu trúc tổng thể hệ thống
- Mục đích
 - Trả lời câu hỏi: chia để trị ntn?
 - Thành lập các phân hệ con, điều khiển các tương tác giữa chúng

a. Thiết kế kiến trúc

- Tiến trình thiết kế kiến trúc



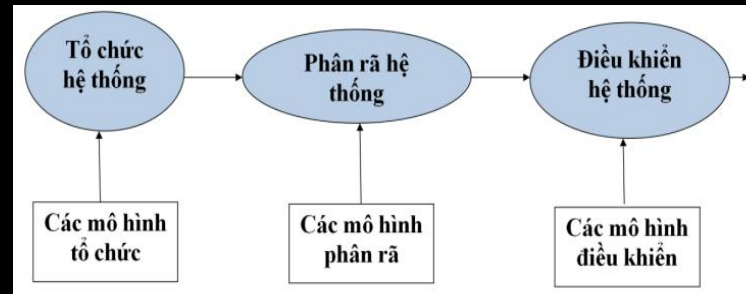
a. Thiết kế kiến trúc



i. Tổ chức hệ thống

- Lựa chọn và xây dựng mô hình kiến trúc tổng thể cho hệ thống
- Các mô tổ chức/kiến trúc?

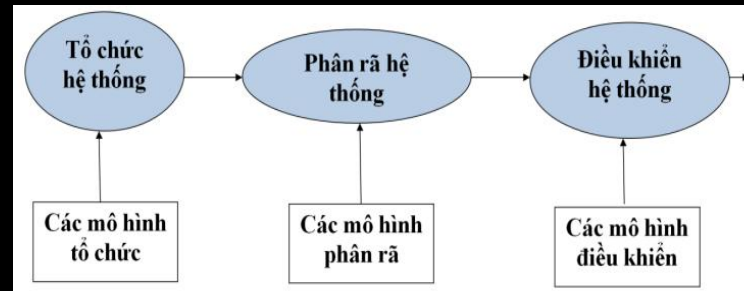
a. Thiết kế kiến trúc



i. Tổ chức hệ thống

- Các mô hình tổ chức/kiến trúc thông dụng
 - *Kiến trúc dữ liệu tập trung (Data centered Architectures)*
 - *Kiến trúc khách-dịch vụ (Client-server Architectures)*
 - *Kiến trúc phân tầng (Layered Architecture)*

a. Thiết kế kiến trúc

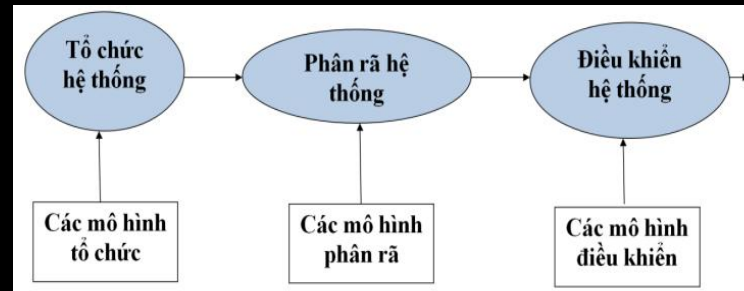


■ Mô hình kiến trúc dữ liệu tập trung

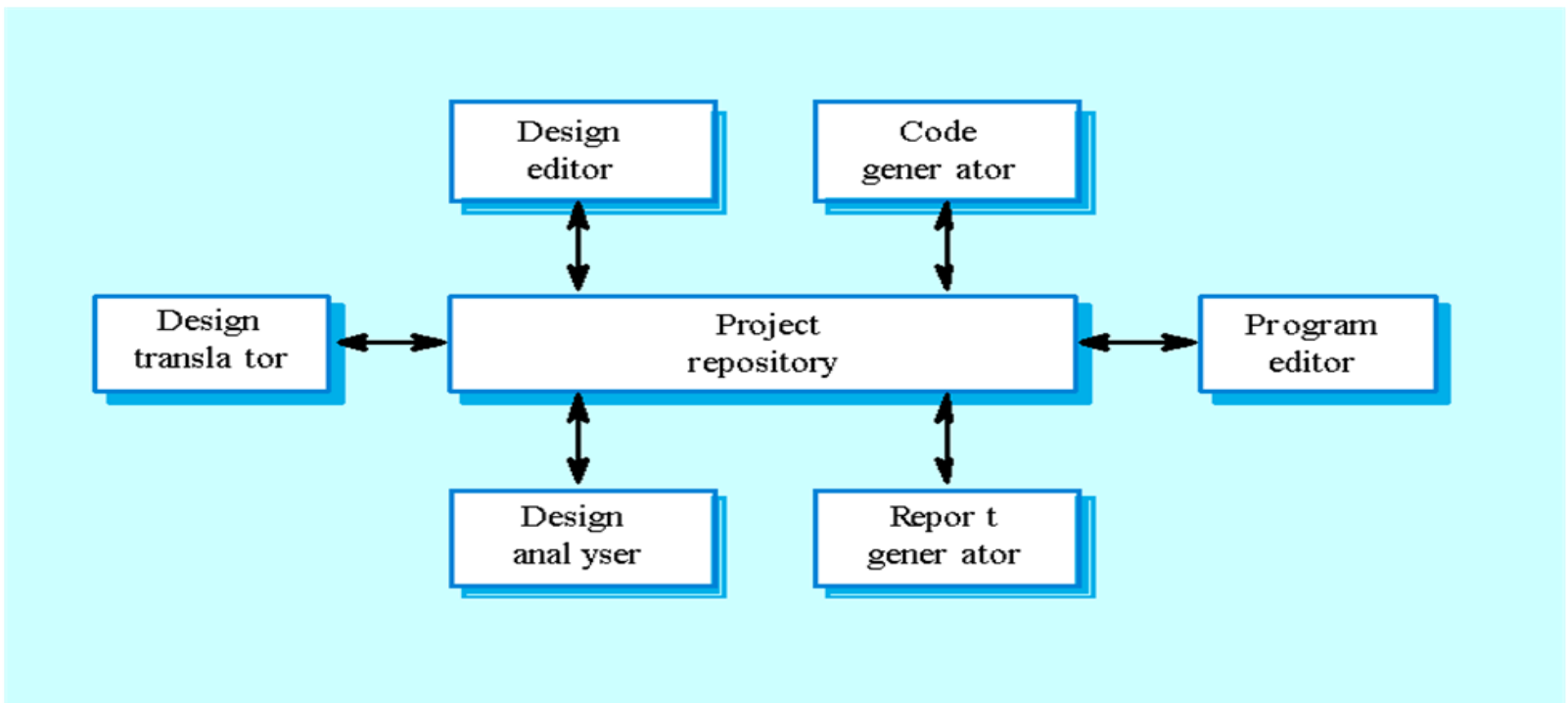
- Các phân hệ:

1. **Phân hệ quản lý CSDL dùng chung:** Quản lý các vấn đề liên quan đến dữ liệu (*Cập nhật lưu trữ, sao lưu, cấp quyền, an toàn, an ninh dữ liệu,*)
2. **Các phân hệ xử lý** (*truy cập đến CSDL để yêu cầu cung cấp dữ liệu về xử lý, hoặc cập nhật dữ liệu trong kho chung*)

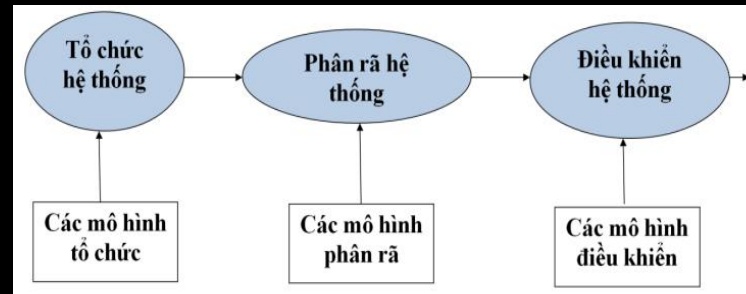
a. Thiết kế kiến trúc



- **Mô hình kiến trúc dữ liệu tập trung**
 - Ví dụ: Kiến trúc của bộ công cụ CASE



a. Thiết kế kiến trúc

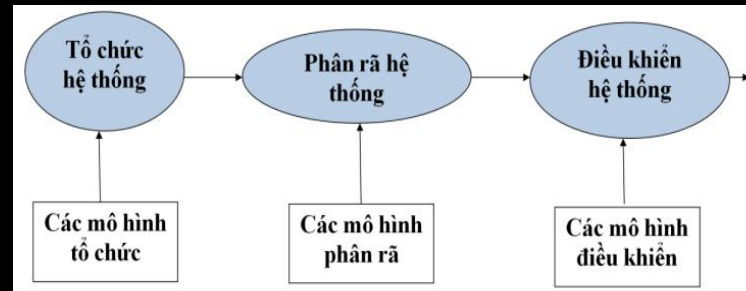


■ Mô hình kiến trúc dữ liệu tập trung

■ Ưu điểm

- Chia sẻ một khối lượng lớn dữ liệu;
- Phân hệ xử lý không cần quan tâm đến cách thức dữ liệu được quản lý ở bộ quản lý trung tâm.
- Phân hệ quản lý kho dữ liệu không cần quan tâm đến dữ liệu được xử lý như thế nào ở các phân hệ xử lý.

a. Thiết kế kiến trúc

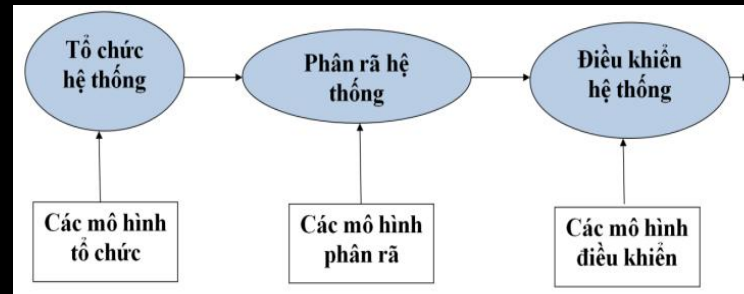


■ Mô hình kiến trúc dữ liệu tập trung

• Nhược điểm

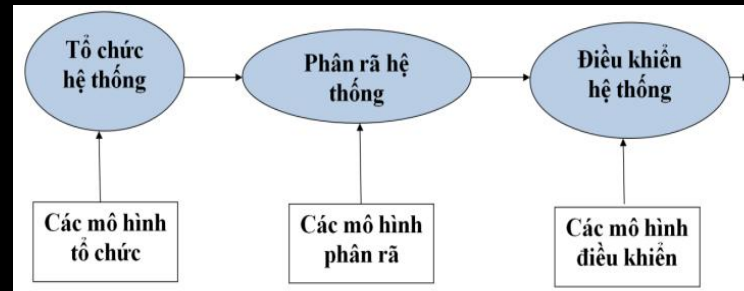
- Các hệ thống con phải chấp nhận mô hình kho chứa dữ liệu.
- Cải tiến dữ liệu là khó và đắt đỏ;
- Khó phát huy các chính sách quản lý dữ liệu riêng của từng phân hệ xử lý;
- Khó phân tán dữ liệu một cách hiệu quả.

a. Thiết kế kiến trúc

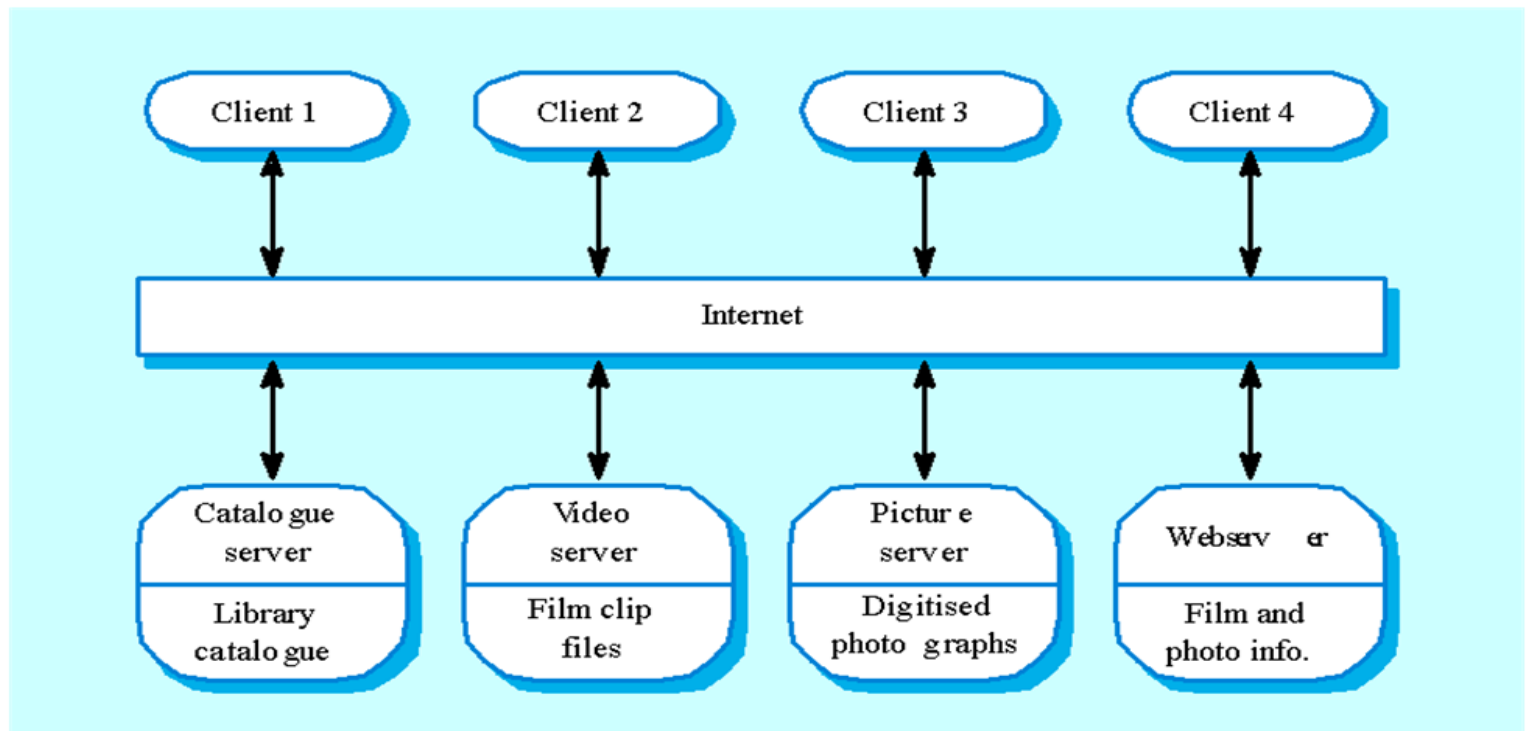


- **Kiến trúc Khách – Phục vụ (Client – server)**
 - Là mô hình hệ thống phân tán
 - Các thành phần của hệ thống được phân tán trên các server khác nhau.
 - Các thành phần kiến trúc:
 - Tập các servers độc lập cung cấp các dịch vụ
 - Tập các Client triệu gọi các dịch vụ từ các servers.
 - Mạng kết nối giữa Client và servers.

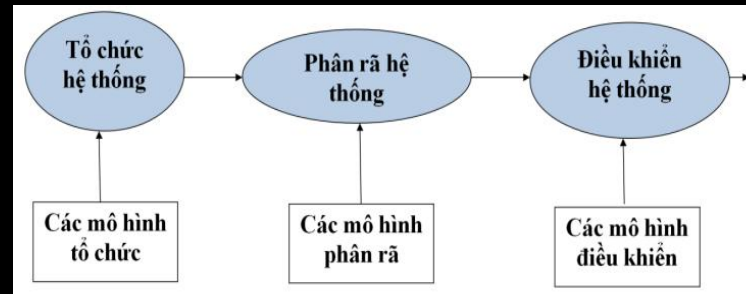
a. Thiết kế kiến trúc



- **Kiến trúc Khách – Phục vụ (Client – server)**
 - Ví dụ: kiến trúc của hệ thống quản lý phim & ảnh

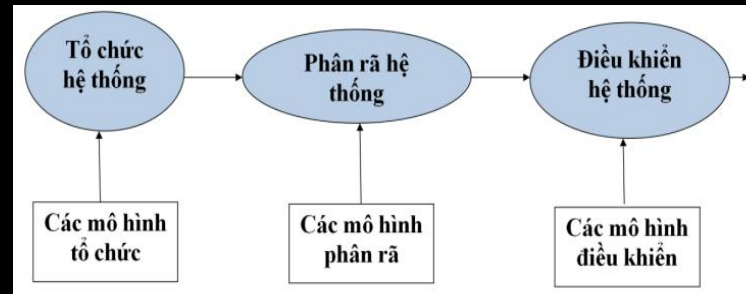


a. Thiết kế kiến trúc



- **Kiến trúc Khách – Phục vụ (Client – server)**
 - Ưu điểm
 - Dễ thêm các server mới hoặc nâng cấp các servers đang tồn tại.
 - Phù hợp với các hệ thống có kiến trúc phân tán.

a. Thiết kế kiến trúc

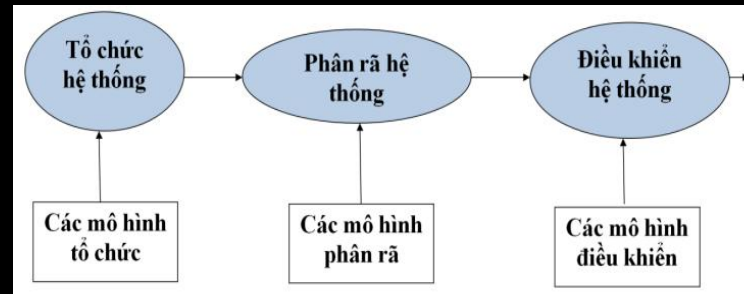


■ Kiến trúc Khách – Phục vụ (Client – server)

■ Hạn chế:

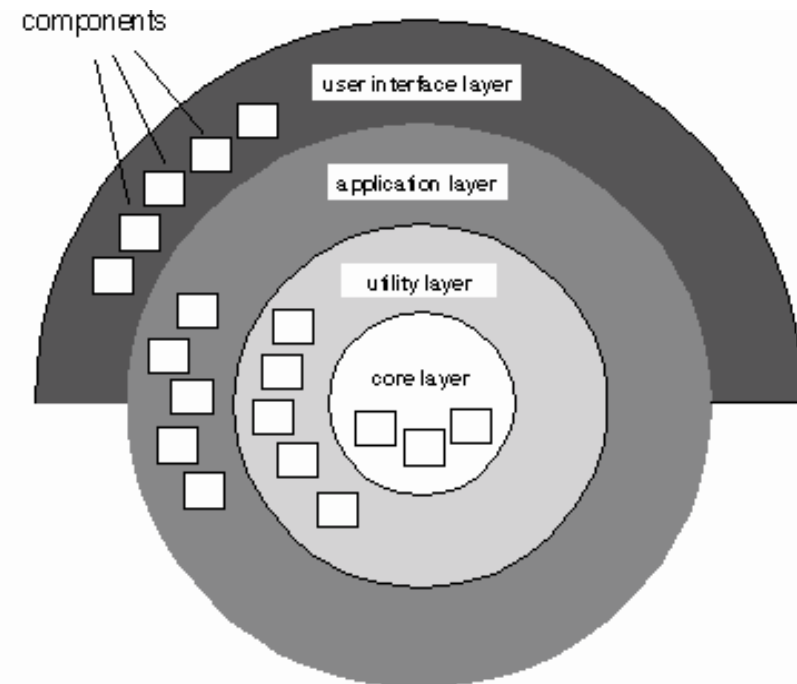
- Các phân hệ sử dụng cách tổ chức dữ liệu khác nhau (\Rightarrow *không thuận lợi khi trao đổi dữ liệu*)
- Quản lý dữ liệu dư thừa trên mỗi server;
- Không có bộ đăng ký trung tâm về tên và các dịch vụ (\Rightarrow *có thể khó tìm các dịch vụ và các servers*)

a. Thiết kế kiến trúc

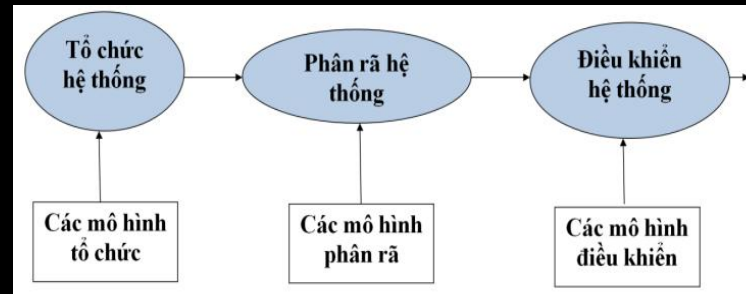


■ Kiến trúc phân tầng

- Chia hệ thống thành các tầng.
 - Mỗi tầng cung cấp một tập các dịch vụ.
- Số tầng (layer):
 - 3 tầng, ...n tầng.
- Ví dụ:
 - Xem hình



a. Thiết kế kiến trúc

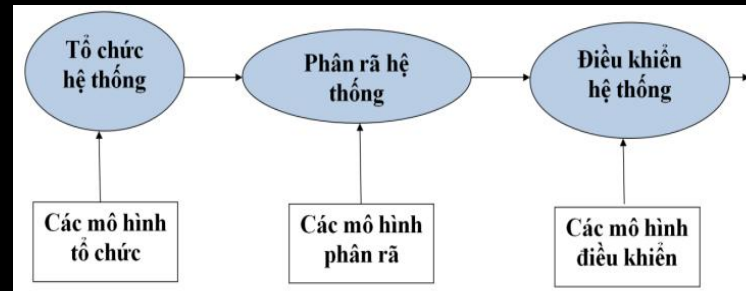


■ Kiến trúc phân tầng

■ Ưu điểm:

- Dễ bảo trì, sửa chữa.
- Nâng cao tính an ninh của hệ thống

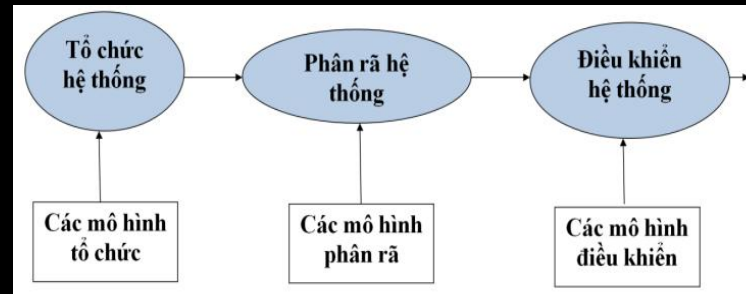
a. Thiết kế kiến trúc



ii. Phân rã hệ thống:

- Phân rã hệ thống thành các thành phần
 - Thành phần:
 - Hệ con
 - Chương trình
 - Mô đun
- Phụ thuộc vào mô hình tổ chức đã lựa chọn

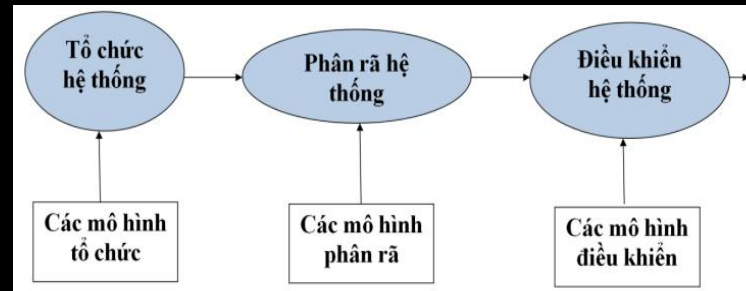
a. Thiết kế kiến trúc



ii. Phân rã hệ thống:

- Kiến trúc hệ thống nhìn dưới góc độ phân cấp
 - System → subsystem (master files) → program (temporal files) → module/class (arguments).

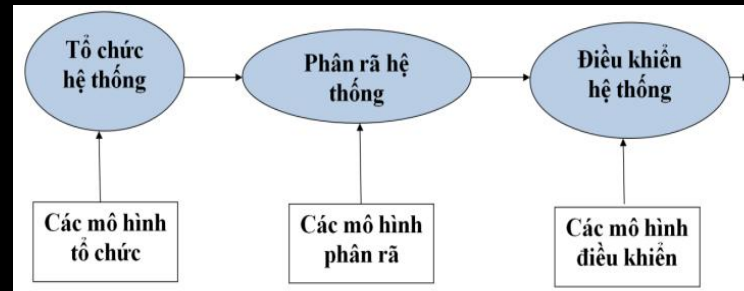
a. Thiết kế kiến trúc



iii. Điều khiển hệ thống

- Mục đích:
 - Điều khiển sự tương tác giữa các thành phần sau khi phân chia.
 - => Có những cách tiếp cận điều khiển hệ thống nào?
 - => Ví dụ về các mô hình điều khiển của từng cách tiếp cận?

a. Thiết kế kiến trúc



iii. Điều khiển hệ thống

- 2 kiểu điều khiển:

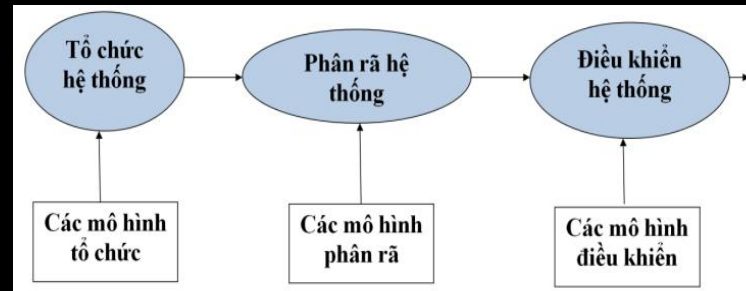
- **Điều khiển tập trung:**

- Một hệ thống chịu trách nhiệm điều khiển, khởi tạo và dừng sự thực hiện của các hệ thống con khác.

- **Điều khiển dựa trên sự kiện:**

- Mỗi hệ thống con có thể phản ứng lại với các sự kiện phát sinh từ các hệ thống con khác hoặc từ môi trường của hệ thống.

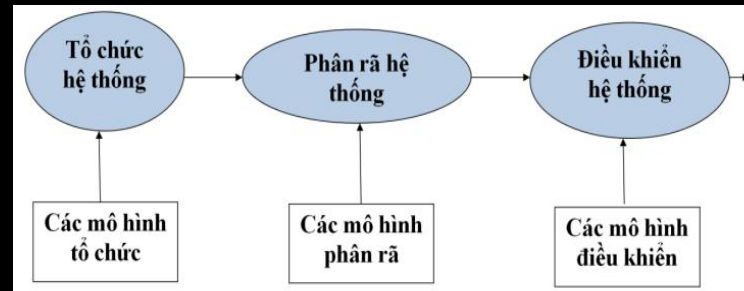
a. Thiết kế kiến trúc



■ Điều khiển tập trung

- Hai mô hình kiến trúc điều khiển tập trung được sử dụng khá phổ biến:
 - Mô hình kiến trúc gọi - trả lời.
 - Mô hình quản lý.
 -

a. Thiết kế kiến trúc

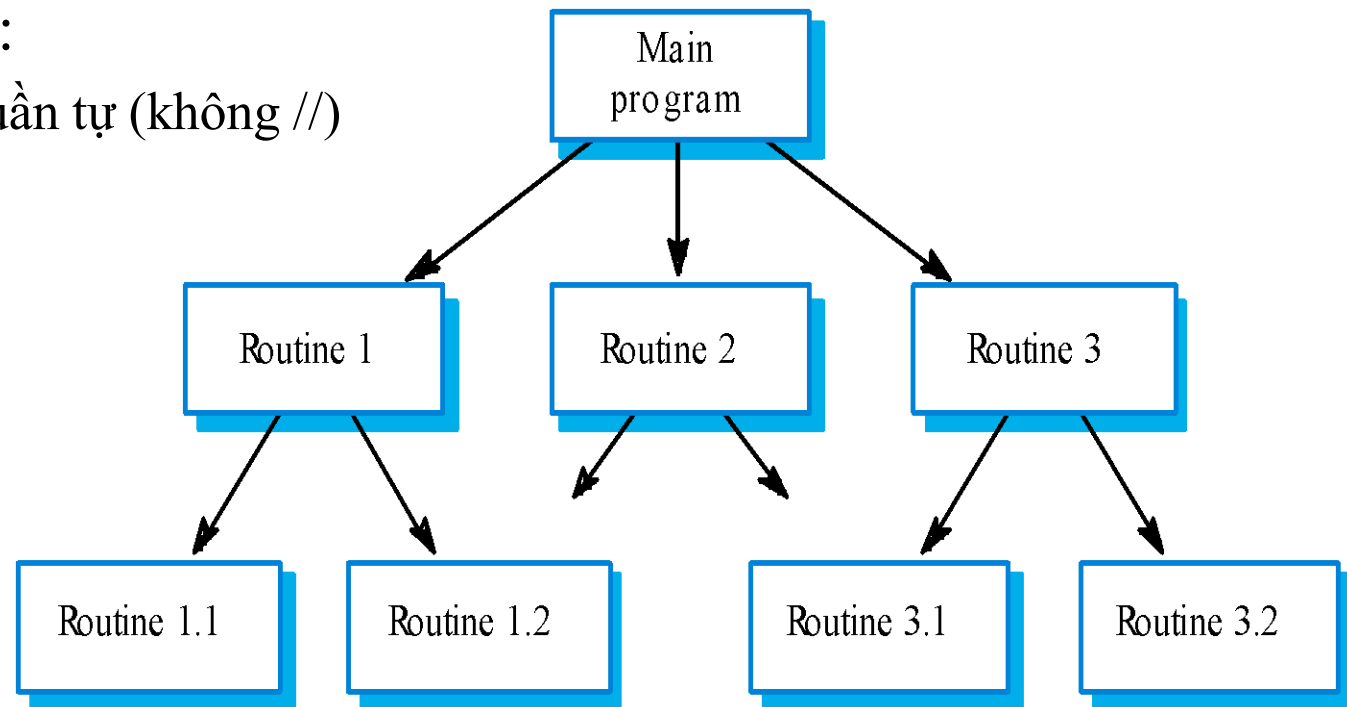


■ Điều khiển tập trung

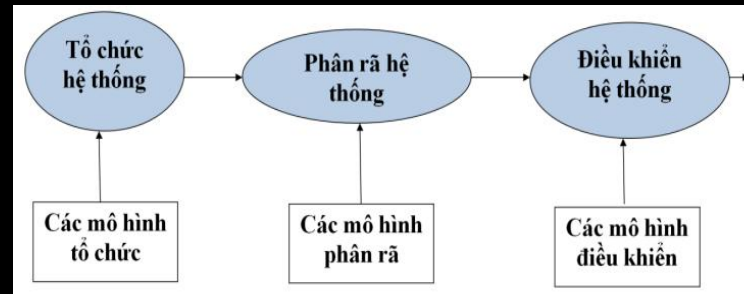
■ Mô hình call – return

■ Hạn chế:

- Xử lý tuần tự (không //)



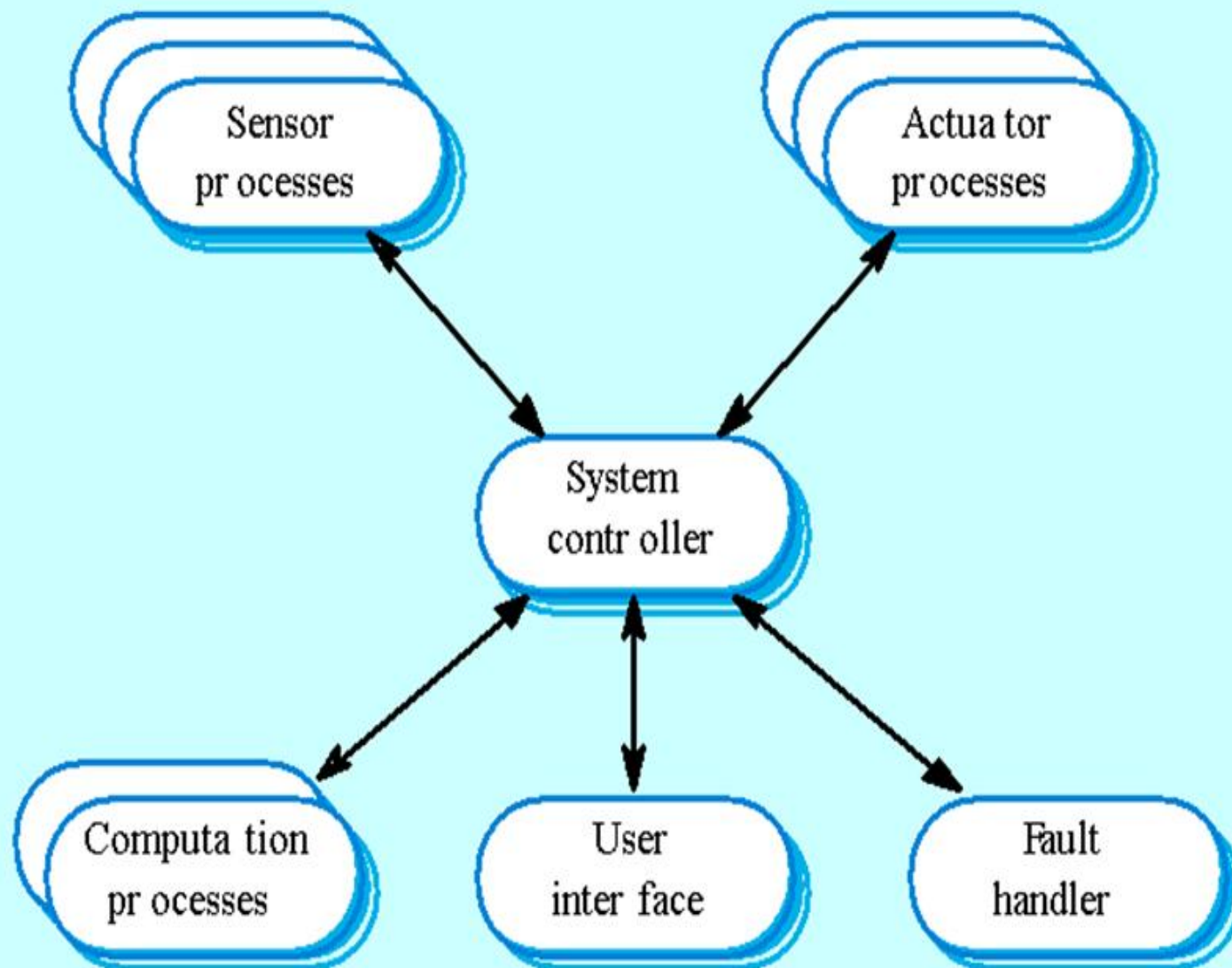
a. Thiết kế kiến trúc



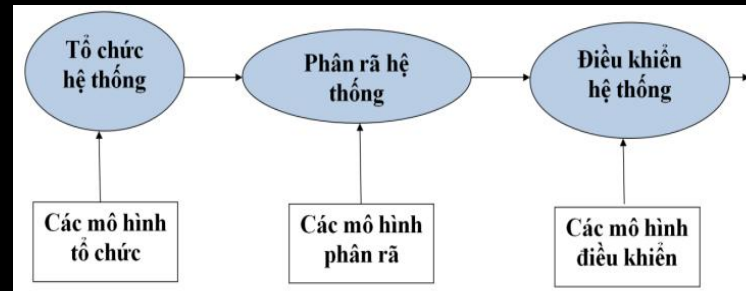
■ Điều khiển tập trung

■ Mô hình quản lý

- Trong các phân hệ có một phân hệ quản trị
 - Phân hệ quản trị: điều khiển việc dừng, khởi động và phối hợp các xử lý của hệ thống khác.
- Ưu điểm
 - Có thể áp dụng cho các hệ thống song song.
 - Nhiều phân hệ cùng thực hiện tại một thời điểm.
- Ví dụ: Mô hình quản lý của hệ thống xử lý dữ liệu thời gian thực
 - Xem hình (dưới)

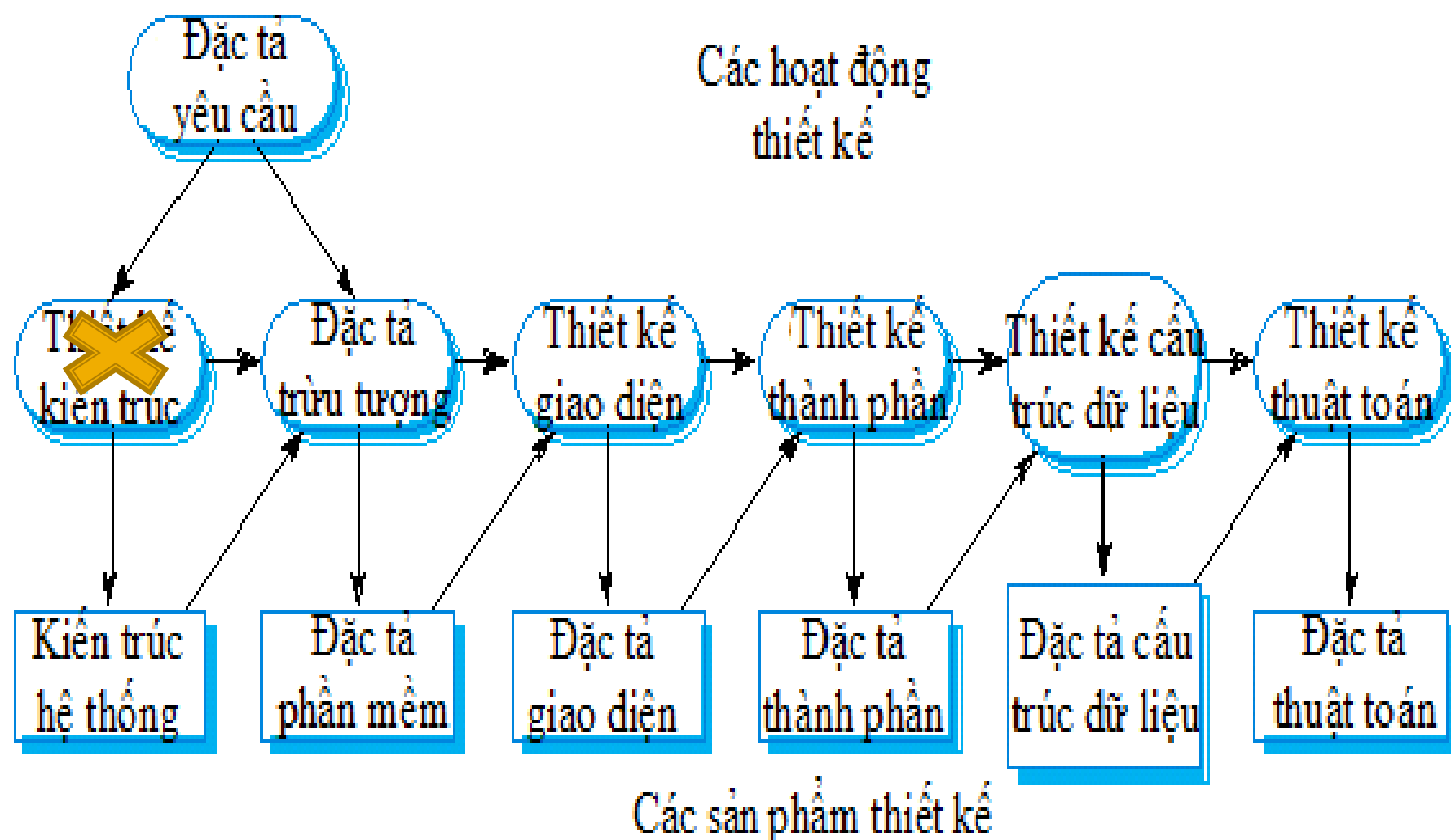


a. Thiết kế kiến trúc



■ Điều khiển hướng sự kiện

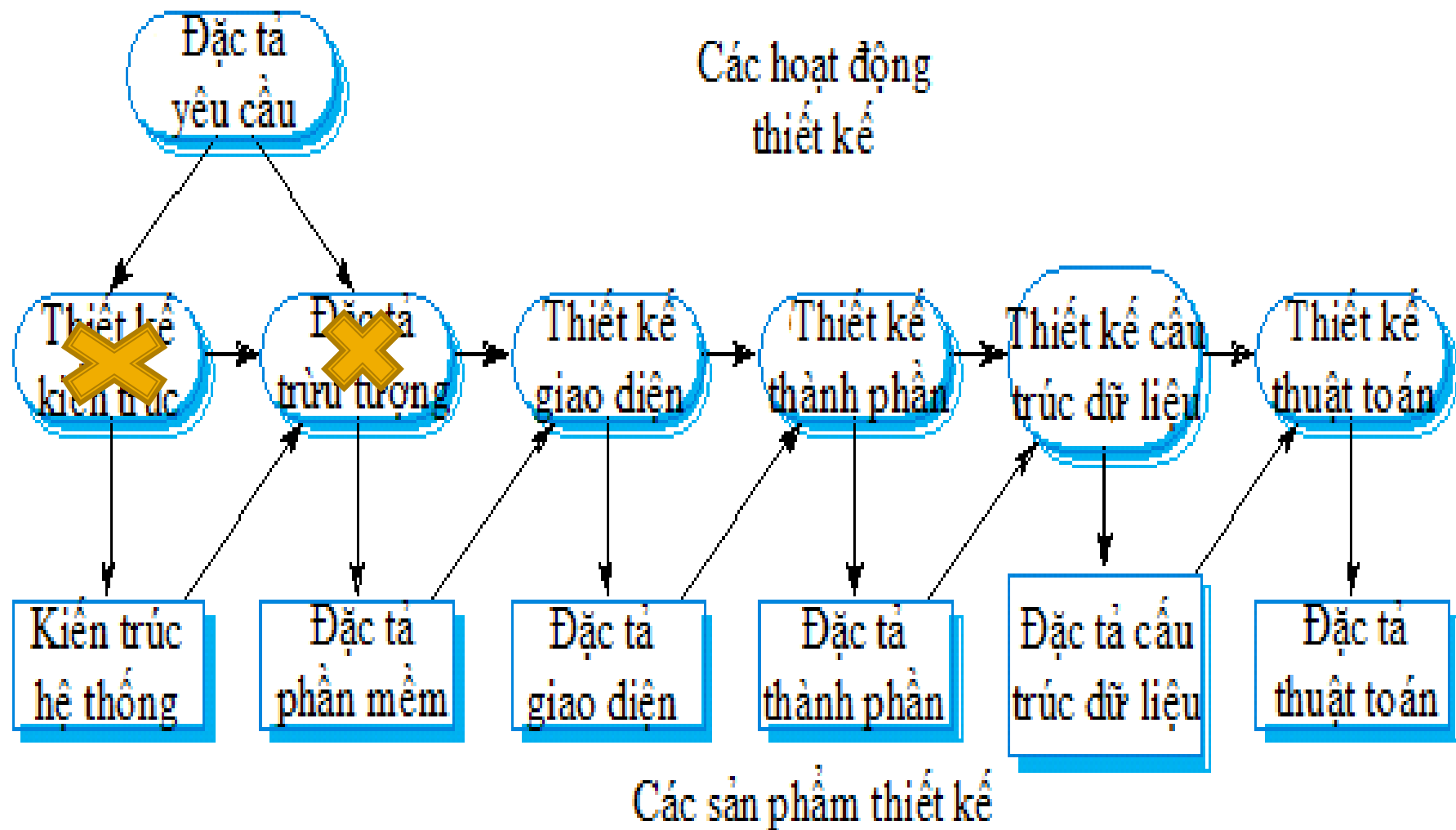
- Các phân hệ được điều khiển dựa trên sự kiện phát sinh từ bên ngoài hệ thống.
- Thường áp dụng cho các ứng dụng thời gian thực
- 2 mô hình điều khiển chính:
 - *Các mô hình thông báo*
 - *Các mô hình điều khiển ngắt*



Hình 2.7: Mô hình chung của quy trình thiết kế

b. Đặc tả trừu tượng

- Đặc tả yêu cầu của từng thành phần hệ thống (hệ con, chương trình).
 - Phân tích xác định các yêu cầu của từng phân hệ



Hình 2.7: Mô hình chung của quy trình thiết kế

c. Thiết kế giao diện

- **Tầm quan trọng:**

- Người dùng thường đánh giá hệ thống thông qua giao diện
 - Giao diện thiết kế nghèo nàn
 - Khó nghiệm thu,
 - Phần mềm không được sử dụng
 - Tạo điều kiện cho người dùng gây lỗi.

c. Thiết kế giao diện

- Người dùng
 - Có vai trò trung tâm ảnh hưởng thiết kế giao diện
 - Các đặc điểm của con người:
 - Khả năng nhớ tức thời của con người bị hạn chế
 - Người dùng luôn gây ra lỗi
 - Người sử dụng là khác nhau
 - Người dùng thích các loại tương tác khác nhau.
- => *Các nguyên tắc thiết kế giao diện.*

c. Thiết kế giao diện

- Các nguyên tắc thiết kế giao diện
 - Hướng đến sự quen thuộc của người sử dụng
 - Thống nhất
 - Tối thiểu hóa sự bất ngờ
 - Cung cấp khả năng phục hồi
 - Hướng dẫn sử dụng
 - Đa dạng trong tương tác
 - *Vận hành trực tiếp*
 - *Lựa chọn menu*
 - *Điền vào biểu mẫu (Form)*
 - *Ngôn ngữ ra lệnh*
 - *Ngôn ngữ tự nhiên*

c. Thiết kế giao diện

Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement. Only suitable where there is a visual metaphor for tasks and objects.	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users. Can become complex if many menu options.	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn Checkable	Takes up a lot of screen space. Causes problems where user options do not match the form fields.	Stock control, Personal loan processing
Command language	Powerful and flexible	Hard to learn. Poor error management.	Operating systems, Command and control systems
Natural language	Accessible to casual users Easily extended	Requires more typing. Natural language understanding systems are unreliable.	Information retrieval systems

c. Thiết kế giao diện

- Sử dụng màu trong thiết kế giao diện
 - Giúp người dùng hiểu các cấu trúc thông tin phức tạp.
 - Sử dụng để làm nổi bật các sự kiện ngoại lệ.
 - Lạm dụng màu:
 - không tốt.
 - => *cần tuân theo các hướng dẫn sử dụng màu*

c. Thiết kế giao diện

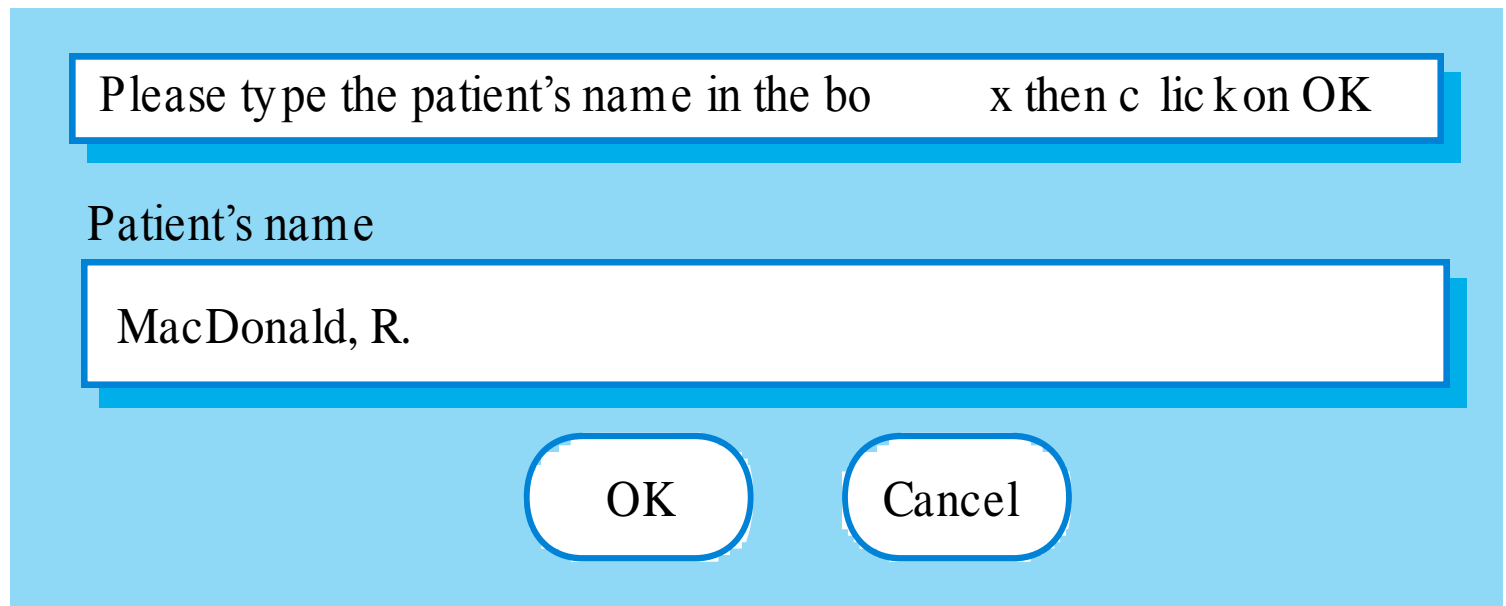
- Các hướng dẫn sử dụng màu:
 - Hạn chế số lượng màu được sử dụng
 - Nên sử dụng màu để chỉ ra các trạng thái hệ thống.
 - Sử dụng màu một cách thống nhất
 - Cẩn thận với các cặp màu tương tự.
 - Sử dụng màu phù hợp với đối tượng người dùng và miền ứng dụng.

c. Thiết kế giao diện

- Thiết kế các thông báo lỗi
 - Cần lịch sự, xúc tích, phù hợp và có cấu trúc.
 - Tránh gây căng thẳng người dùng.

c. Thiết kế giao diện

- Ví dụ: xét tình huống
 - Giả sử y tá gõ sai tên bệnh nhân muốn tìm kiếm



Please type the patient's name in the box then click on OK

Patient's name

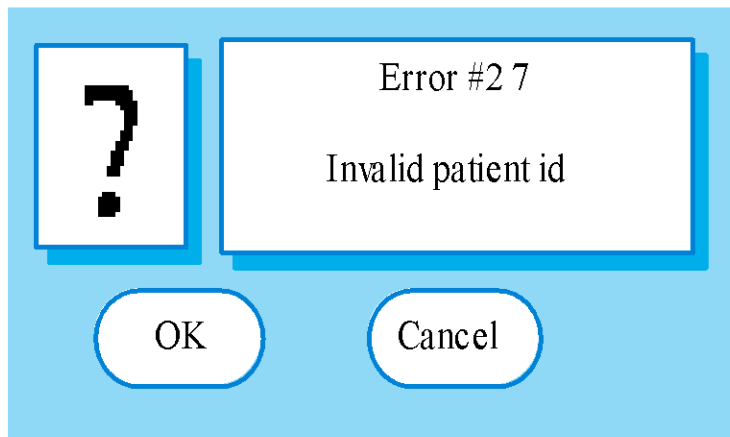
MacDonald, R.

OK Cancel

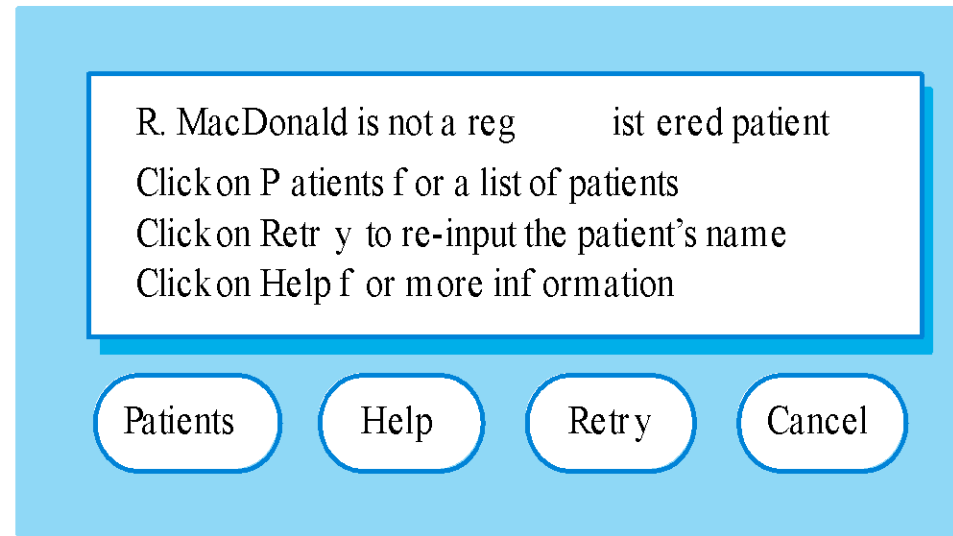
c. Thiết kế giao diện

- Ví dụ:
 - Các thông báo lỗi (tốt? Tồi?)

System-oriented error message



User-oriented error message

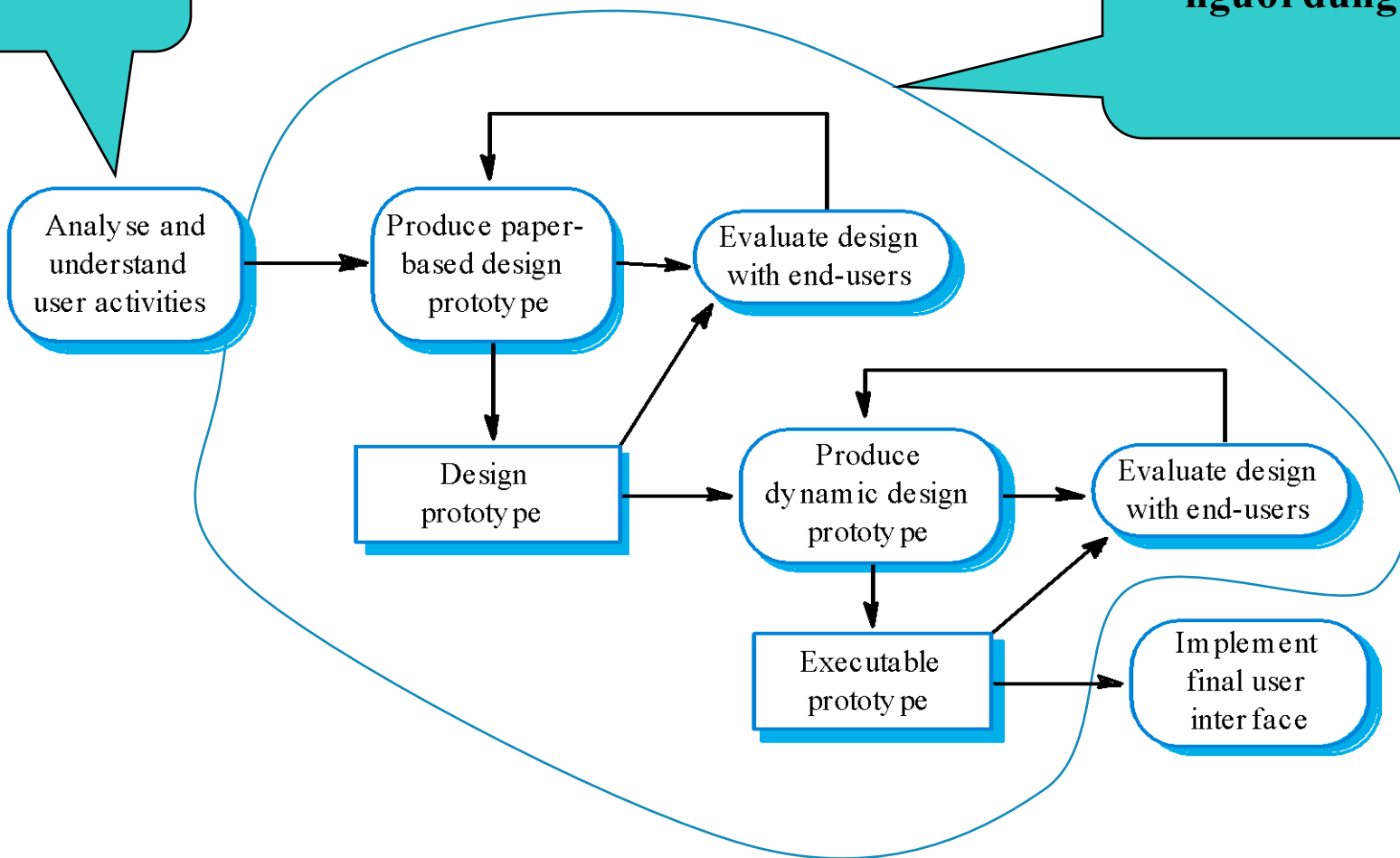


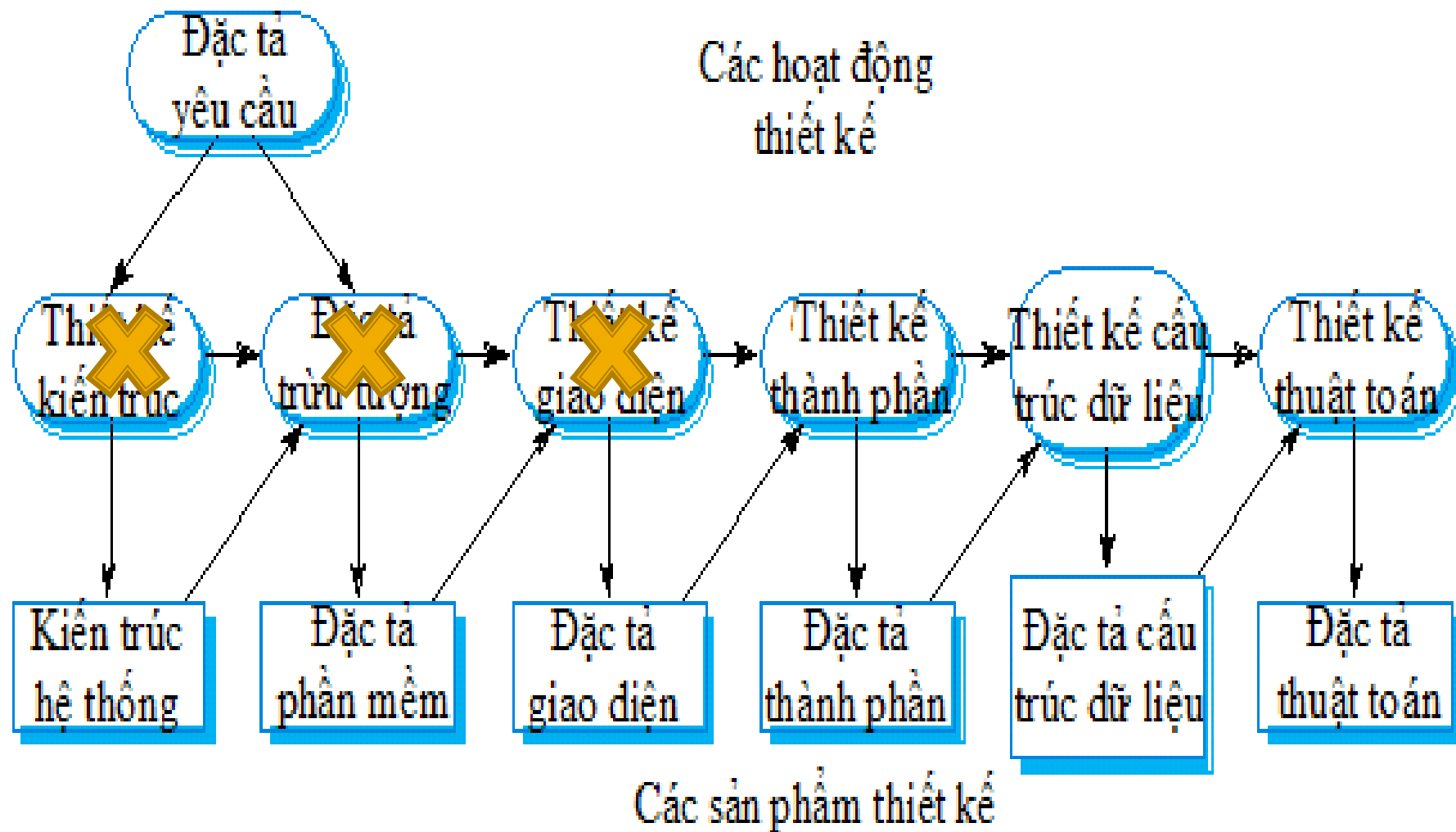
c. Thiết kế giao diện

- Tiến trình thiết kế giao diện
 - Là tiến trình lặp, cần sự tham gia của người dùng.
 - 3 hoạt động chính
 - **User analysis:**
 - *Hiểu những gì người dùng sẽ làm với hệ thống;*
 - **System prototyping.**
 - Phát triển một chuỗi các mẫu thử thực nghiệm
 - **Interface evaluation.**
 - Thử nghiệm mẫu thử cùng với người dùng

**Phân tích
người dùng**

**Các mẫu thử
người dùng**

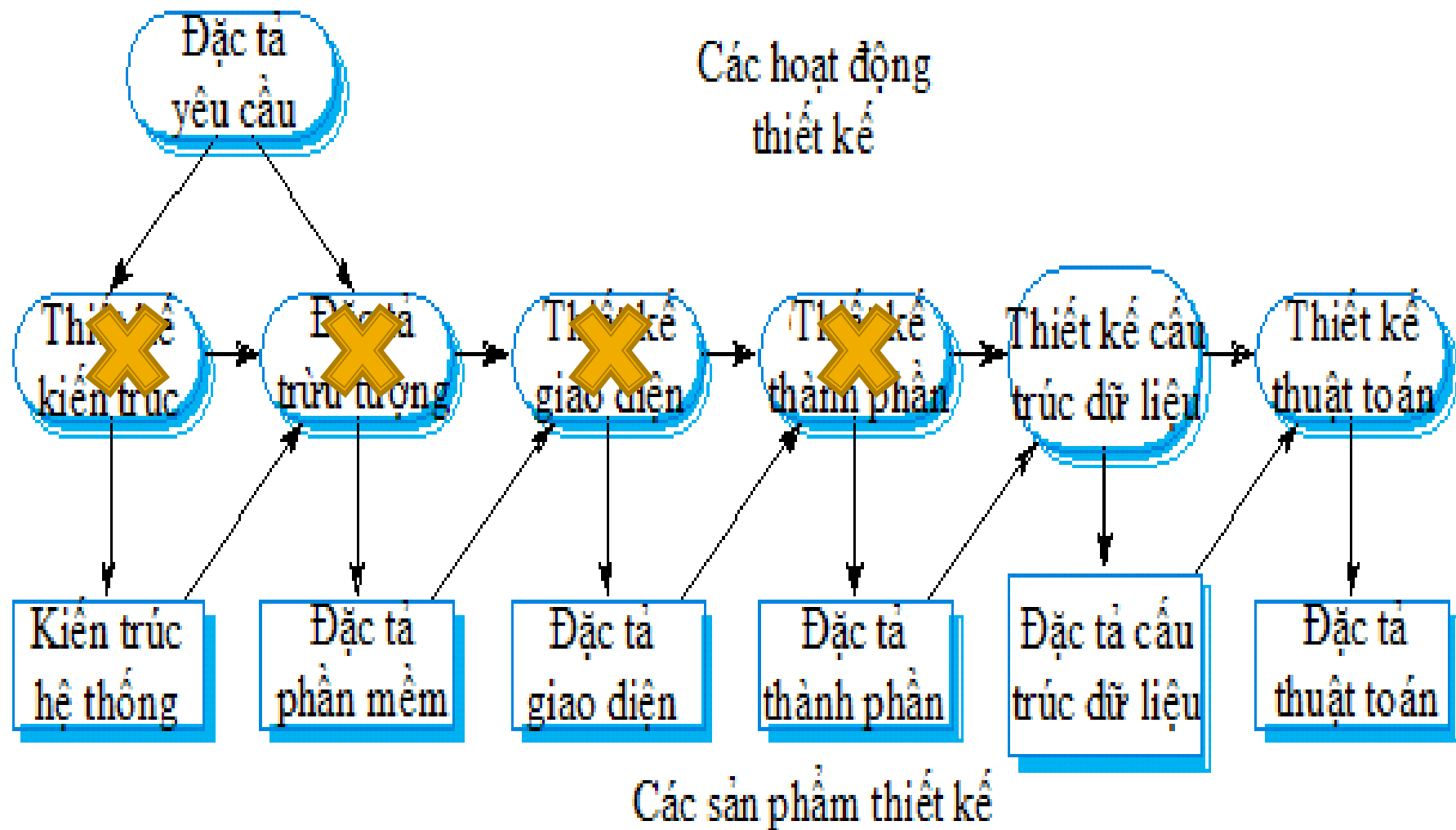




Hình 2.7: Mô hình chung của quy trình thiết kế

d. Thiết kế thành phần

- Mục đích
 - Thiết kế cấu trúc chương trình
 - Phân chia chương trình thành các module.
 - Tiếp cận theo hướng đối tượng
 - Phân chia chương trình thành các lớp đối tượng.
 - Tiếp cận theo hướng chức năng.
 - Đặc tả cấu trúc chương trình



Hình 2.7: Mô hình chung của quy trình thiết kế

e. Thiết kế cấu trúc dữ liệu

- Mục đích
 - Lựa chọn và đặc tả các cấu trúc dữ liệu của chương trình, đảm bảo
 - Thuận lợi cho các thao tác
 - Tiết kiệm không gian
 - Phản ánh đúng các dữ liệu thực tế
 - Đầu ra của TK CTDL:
 - Bản đặc tả CTDL

e. Thiết kế cấu trúc dữ liệu

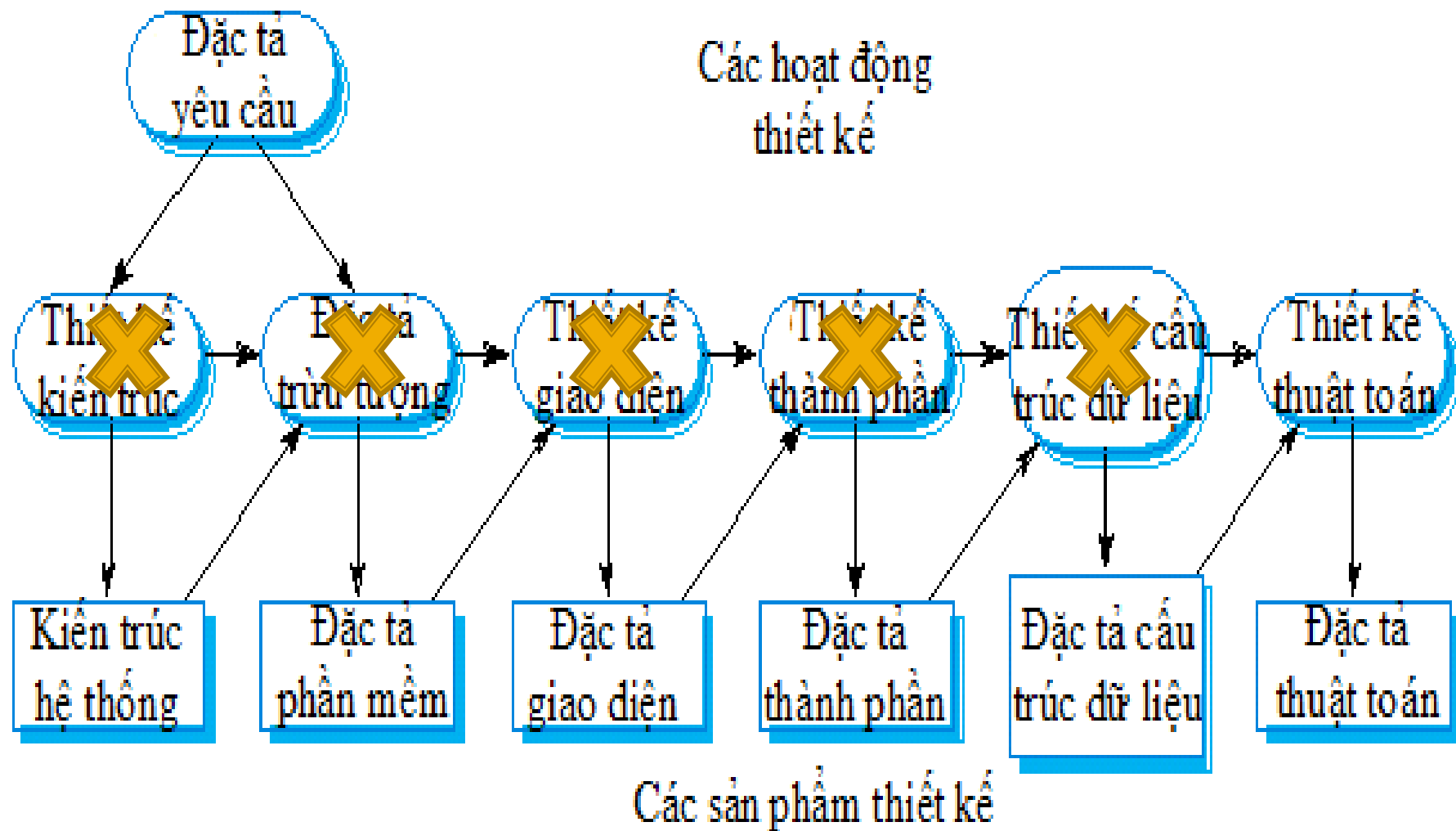
- Ví dụ:
 - Đặc tả kiểu dữ liệu phân số

$$P = \{ x/y \mid x \in \mathbb{Z}, y \in \mathbb{N} \}$$

Nếu $p = x/y \in P$ thì

$$\text{Tử}(p) = x$$

$$\text{Mẫu}(p) = y$$



Hình 2.7: Mô hình chung của quy trình thiết kế

f. Thiết kế thuật toán

- Mục đích
 - Lựa chọn và đặc tả các thuật toán được sử dụng trong chương trình, đảm bảo tính tối ưu của các thuật toán lựa chọn
 - Đầu ra:
 - Bản đặc tả thiết kế thuật toán

f. Thiết kế thuật toán

- Ví dụ: đặc tả phép toán cộng 2 phân số

$+: P \times P \rightarrow P$

$(x:P, y: P): x+y: P$

input: $x:P, y:P$

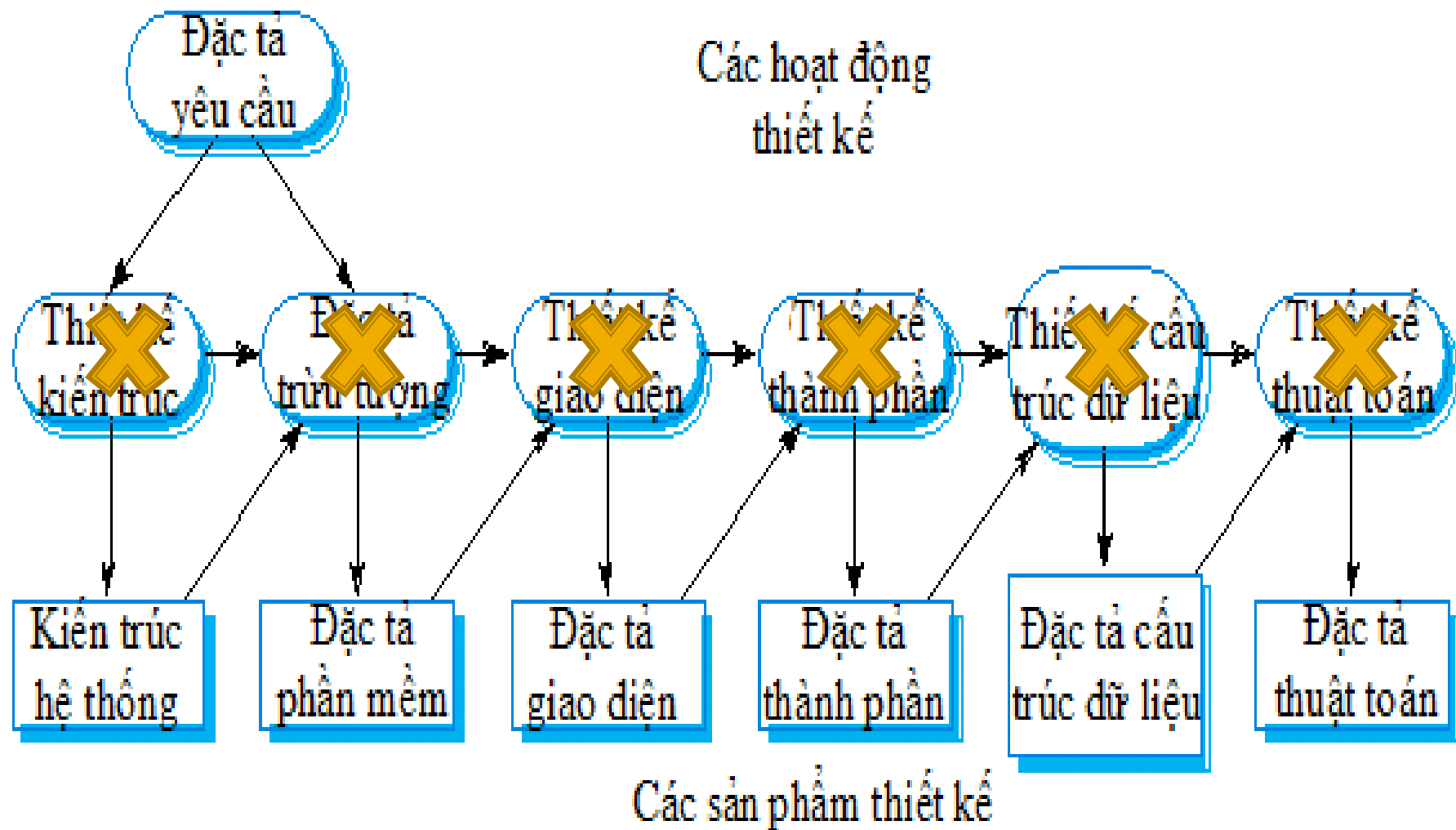
output: $z:P (*z=x+y*)$

qđms(x,y);

tu(z)=tu(x)+tu(y);

mẫu(z) = mẫu(x);

rutgon(z);



Hình 2.7: Mô hình chung của quy trình thiết kế

3. Lập trình

- Mục đích:
 - Chuyển các mô đun thiết kế thành các mô đun chương trình
 - Đảm bảo chất lượng chương trình
 - *Chạy đúng, dễ hiểu, dễ bảo trì & có thể tái sử dụng*

3. Lập trình

- Các yêu cầu đối với lập trình viên
 - Có năng lực cá nhân.
 - Hiểu biết các công cụ lập trình & các phương pháp lập trình.
 - Nắm vững các nguyên lý lập trình.
 - Có kinh nghiệm.
 - Lựa chọn ngôn ngữ cho ứng dụng
 - Quan trọng: ảnh hưởng đến chất lượng & khả năng bảo trì của ứng dụng
 - => lựa chọn ntn?

3. Lập trình

- Các yếu tố cần quan tâm khi lựa chọn ngôn ngữ cho ứng dụng
 - *Theo yêu cầu của khách hàng*
 - Nếu khách tự bảo trì phần mềm
 - *Chương trình dịch của NN*
 - Cần có chương trình dịch có chất lượng tốt
 - *Công cụ hỗ trợ*
 - Dễ dàng quá trình lập trình và bảo trì
 - *Kinh nghiệm của lập trình viên*
 - Chọn NN mà lập trình viên làm chủ.

3. Lập trình

- Lựa chọn ngôn ngữ cho ứng dụng (t.t)
 - *Tính khả chuyển*
 - Thực hiện trên nhiều máy, hđh khác nhau
 - *Lĩnh vực ứng dụng*
 - Hệ thống nhúng: C, Assembly
 - Hệ thống quản lý: .Net, VB
 - Hệ chuyên gia: Prolog
 - Mạng: .Net, Các công nghệ Java
 - Web: PHP, ASP, JSP
- ⇒ Chưa tồn tại ngôn ngữ đa năng cho mọi ứng dụng
- *Phương pháp phát triển phần mềm*
 - ~ phương pháp luận lập trình

3. Lập trình

- Một số phương pháp luận lập trình
 - Lập trình thủ tục/cấu trúc
 - *Ngôn ngữ: Pascal, C, VB...*
 - Lập trình hướng đối tượng
 - *Các NNLT như: Smalltalk, Java, C#,*
 - Lập trình hàm
 - *Sử dụng để tính toán các biểu thức*
 - *Áp dụng trong các lĩnh vực tính toán và trí tuệ nhân tạo*
 - *NNLT: Lisp, Scheme*
 - Lập trình logic
 - *XD hệ chuyên gia*
 - *Xử lý Ngôn ngữ tự nhiên.*
 - *Ngôn ngữ: Prolog, ...*

3. Lập trình

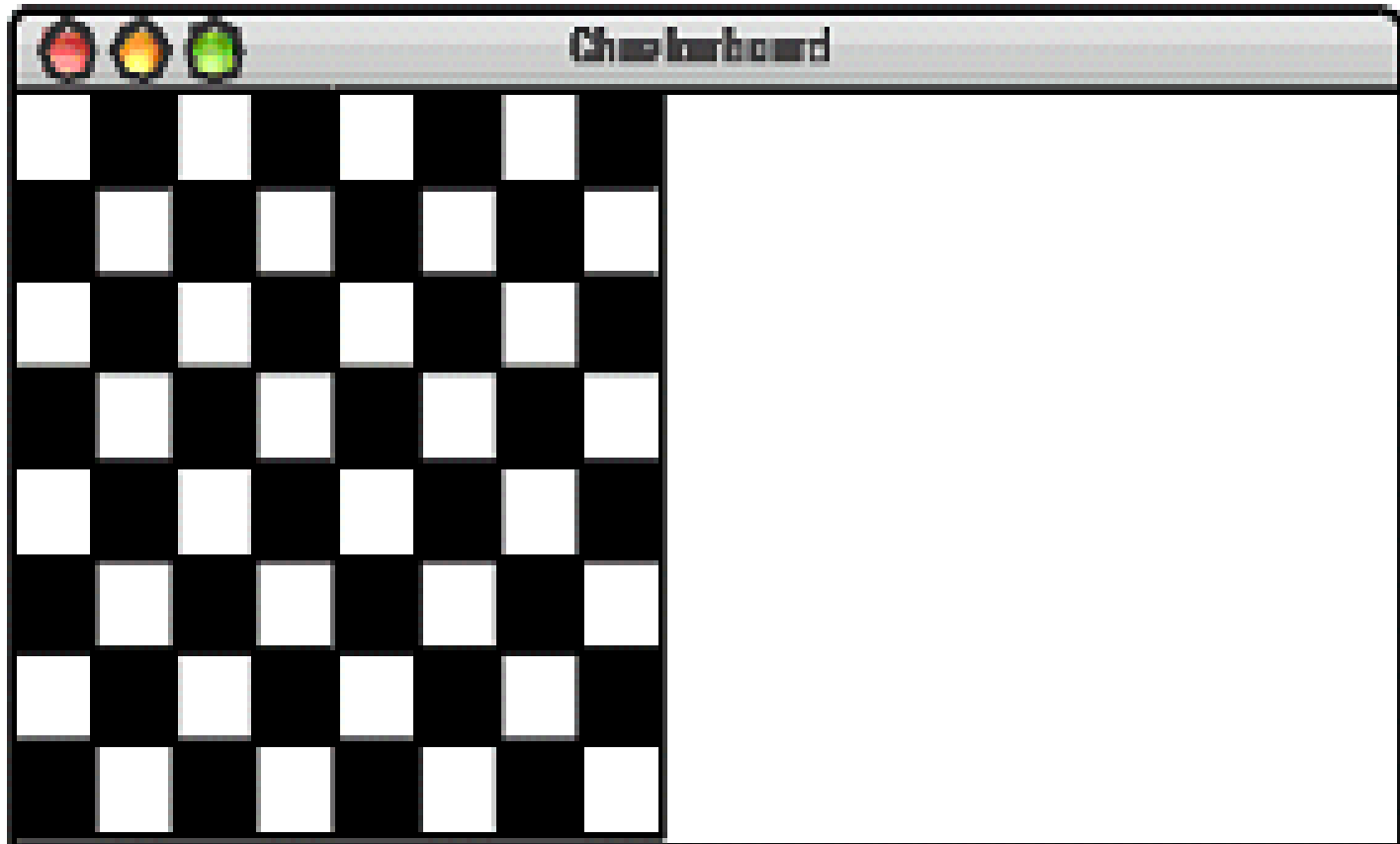
- Phong cách lập trình
 - => tạo chương trình dễ hiểu, dễ bảo trì, ít lỗi
 - Lập trình theo đúng các nguyên tắc (chuẩn, luật)
 - Các nguyên tắc lập trình?

3. Lập trình

- Một số nguyên tắc lập trình:
 - ❖ *Đặt tên: Có ý nghĩa, gọi nhớ*
 - ❖ *Trình bày: Rõ ràng, có cấu trúc, dễ hiểu*
 - ❖ *Chú thích: Đầy đủ, dễ đọc*
 - ❖ *Hạn chế sử dụng các cấu trúc khó kiểm soát: break, goto, continue, ...*
 - ❖ *Tránh sử dụng các số tối nghĩa.*
 - ❖ *Các câu lệnh cần đơn giản: ví dụ: Tránh sử dụng các câu lệnh điều khiển lồng nhau sâu quá 3 mức*
 - ❖ *Ân thông tin: Giúp rút ngắn độ dài chương trình, tránh truy cập bất hợp pháp, dễ bảo trì, dễ hiểu, giảm chi phí*
 - ❖

Phong cách lập trình

Ví dụ: Chương trình vẽ bàn cờ



Phong cách lập trình

Chương trình có phong cách tốt

```
/*
 * File: Checkerboard.java
 * -----
 * This program draws a checkerboard.
 */

import acm.graphics.*;
import acm.program.*;

/*
 * This class draws a checkerboard on the graphics window.
 * The size of the checkerboard is specified by the
 * constants NROWS and NCOLUMNS, and the checkboard fills
 * the vertical space available.
 */

public class Checkerboard extends GraphicsProgram {

    /* Number of rows */
    private static final int NROWS = 8;

    /* Number of columns */
    private static final int NCOLUMNS = 8;

    /* Runs the program */
    public void run() {
        int sqSize = getHeight() / NROWS;
        for (int i = 0; i < NROWS; i++) {
            for (int j = 0; j < NCOLUMNS; j++) {
                int x = j * sqSize;
                int y = i * sqSize;
                GRect sq = new GRect(x, y, sqSize, sqSize);
                sq.setFilled(((i + j) % 2) != 0);
                add(sq);
            }
        }
    }
}
```

Tổng kết

- Chương này giới thiệu các kiến thức cơ bản về thiết kế và lập trình.
- Sinh viên cần nắm vững các hoạt động thiết kế cũng như các nguyên tắc cơ bản để tạo các chương trình có phong cách tốt.

Các câu hỏi thảo luận

1. Đưa ra các lý do cho các trả lời của bạn, hãy gợi ý một mô hình kiến trúc phù hợp cho các hệ thống sau đây:

- Hệ thống bán vé tự động được sử dụng bởi các khách hàng tại một ga tàu.
- Hệ thống hội thảo truyền hình được điều khiển bởi máy tính mà cho phép các âm thanh, hình ảnh và dữ liệu máy tính được hiển thị cho những người tham gia tại cùng một thời điểm.
- Một robot lau sàn nhà. Mục đích là lau sạch các căn phòng. Bộ lau sàn phải có khả năng nhận biết các bức tường và các vật cản khác

Các câu hỏi thảo luận

2. Đưa ra các lý do cho trả lời của bạn, gợi ý mô hình điều khiển phù hợp cho các hệ thống sau đây:

a. Một tập hợp các công cụ phần mềm được tạo bởi các phiên bản khác nhau, nhưng phải làm việc cùng nhau

b. Một bộ điều khiển TV mà phản ứng với các tín hiệu từ các nút điều khiển từ xa

Các câu hỏi thảo luận

3. Có nên có một nghề phần mềm chuyên nghiệp về “software architecture” mà vai trò của nó làm việc độc lập với khách hàng để thiết kế một kiến trúc hệ thống phần mềm? Hệ thống này sau đó có thể được cài đặt ở một số công ty phần mềm. Những khó khăn có thể gặp phải khi thiết lập hệ thống này là gì?

Các câu hỏi thảo luận

- Tìm hiểu và ứng dụng các chuẩn lập trình trong C++