

# CÔNG NGHỆ PHẦN MỀM

## SOFTWARE ENGINEERING

**TS. Võ Đức Hoàng**

Khoa Công nghệ thông tin

Trường Đại học Bách khoa - Đại học Đà Nẵng

- Chương 1: Giới thiệu Công nghệ phần mềm
- Chương 2: Các mô hình phát triển phần mềm
- Chương 3: Phân tích và đặc tả yêu cầu
- Chương 4: Các kỹ thuật đặc tả
- Chương 5: Thiết kế
- Chương 6: Lập trình và ngôn ngữ lập trình
- Chương 7: Kiểm thử
- Chương 8: Quản trị dự án

## ■ ***Giáo trình chính:***

- Andrew Troelsen, *Pro C# 5.0 and The .NET 4.5 Framework*, Apress, 2012.

## ■ ***Tài liệu tham khảo:***

- Simon Kendal, *Object Oriented Programming using C#*, BookBoon, 2012.
- Microsoft MSDN, *C# Programming Guide for Visual Studio 2013*.
- Joe Mayo, *LINQ Programming*, McGraw-Hill Education, 2009.
- Andrew Clymer, *Pro Asynchronous Programming with .NET*, Apress, 2013...

# Thiết kế phần mềm (5)

- Là một quá trình để chuyển đổi các yêu cầu của người dùng thành một số hình thức phù hợp, giúp người lập trình thực hiện và mã hóa phần mềm.
- Có thể được sử dụng trực tiếp để triển khai trong các ngôn ngữ lập trình.
- Là bước đầu tiên trong SDLC (Vòng đời thiết kế phần mềm – Software Design Life Cycle)
  - Chuyển từ miền vấn đề sang miền giải pháp.
  - Cố gắng chỉ định cách thực hiện các yêu cầu được đề cập trong SRS

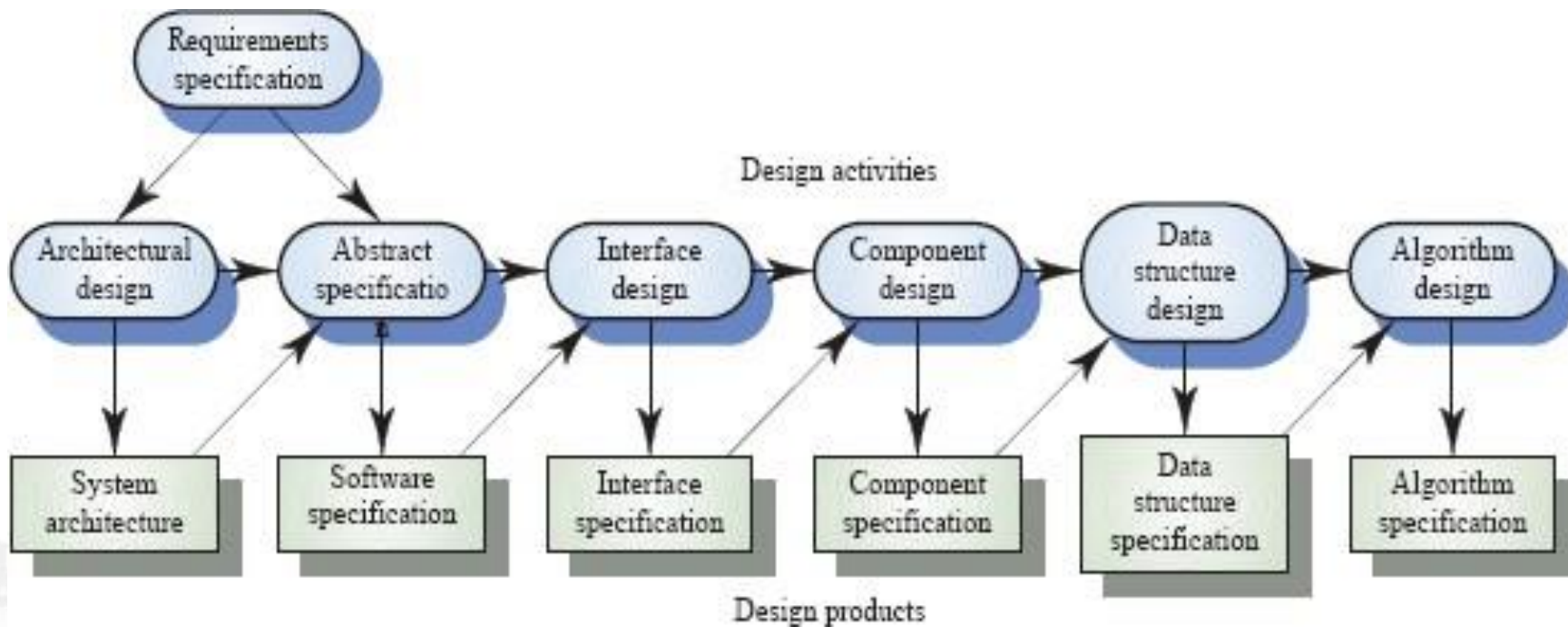


# Thiết kế phần mềm-các mức



- Hoạt động thiết kế xuất hiện trong các mô hình phát triển khác nhau
- Ba mức thiết kế chính
  - Thiết kế kiến trúc (Architectural Design)
    - Phân tích giải pháp thành các thành phần tương tác với nhau
    - Định nghĩa giao diện giữa các thành phần
    - Định nghĩa vấn đề được giải quyết bởi mỗi thành phần
    - Có thể được thực hiện bởi nhiều mức trừu tượng
  - Thiết kế mức cao (High-level Design)
    - Phân rã các thành phần thành các hệ con, các mô-đun và mô tả sự tương tác giữa các mô-đun
  - Thiết kế chi tiết (Detailed Design)
    - Cấu trúc logic của từng mô-đun và giao diện của chúng để giao tiếp với các mô-đun khác.
    - ~~Thiết kế thuật toán, cấu trúc dữ liệu,...~~

# Thiết kế phần mềm - các giai đoạn



- *Architectural design*
  - Xác định các hệ thống con
- *Abstract specification*
  - Đặc tả các hệ thống con
- *Interface design*
  - Mô tả giao diện các hệ thống con
- *Component design*
  - Phân tích hệ thống con thành các thành phần
- *Data structure design*
  - Các cấu trúc dữ liệu lưu trữ dữ liệu của bài toán
- *Algorithm design*
  - Thiết kế thuật toán cho các hàm/mô-đun



## ■ Architectural design (Thiết kế kiến trúc)

- Đặc tả các thành phần chính của một hệ thống, vai trò, tính chất, giao diện của chúng, và các mối quan hệ và tương tác giữa chúng.
- Cấu trúc tổng thể của hệ thống được chọn, nhưng các chi tiết bên trong của các thành phần chính bị bỏ qua.
- Các vấn đề trong thiết kế kiến trúc bao gồm:
  - Phân ra hệ thống thành các thành phần chính.
  - Phân bổ vai trò chức năng cho các thành phần.
  - Xác định giao diện của các thành phần
  - Thuộc tính hiệu suất và khả năng mở rộng, thuộc tính tiêu thụ tài nguyên, thuộc tính độ tin cậy, v.v.
  - Giao tiếp và tương tác giữa các thành phần.
  - Thiết kế bên trong của các thành phần chính được bỏ qua cho đến giai đoạn cuối cùng của thiết kế.

## ■ Interface Design - Thiết kế giao diện:

- Đặc tả sự tương tác giữa một hệ thống và môi trường của nó.
- Giai đoạn này tiến hành ở mức độ trừu tượng cao đối với hoạt động bên trong của hệ thống, tức là trong quá trình thiết kế giao diện, phần bên trong của các hệ thống hoàn toàn bị bỏ qua và hệ thống được coi là một hộp đen.
- Tập trung vào đối thoại giữa hệ thống đích và người dùng, thiết bị và các hệ thống khác mà nó tương tác. Phát biểu vấn đề thiết kế được tạo ra trong bước phân tích vấn đề cần xác định con người, các hệ thống khác và các thiết bị được gọi chung là các tác nhân.
- Thiết kế giao diện nên bao gồm các chi tiết sau:
  - Mô tả chính xác các sự kiện trong môi trường hoặc thông điệp từ các tác nhân mà hệ thống phải phản hồi.
  - Mô tả chính xác các sự kiện hoặc thông điệp mà hệ thống phải tạo ra.
  - Đặc tả về dữ liệu và các định dạng của dữ liệu vào và ra hệ thống.
  - Đặc tả mối quan hệ thứ tự và thời gian giữa các sự kiện hoặc thông điệp đến và các sự kiện hoặc đầu ra đi.

## ■ Detailed Design - Thiết kế chi tiết

- Đặc tả các yếu tố bên trong của tất cả các thành phần hệ thống chính, các thuộc tính, các mối quan hệ, các xử lý và thường là thuật toán và cấu trúc dữ liệu của chúng.
- Thiết kế chi tiết có thể bao gồm:
  - Phân rã các thành phần hệ thống chính thành các đơn vị chương trình.
  - Phân bổ trách nhiệm chức năng cho các đơn vị.
  - Giao diện người dùng
  - Trạng thái đơn vị và thay đổi trạng thái
  - Dữ liệu và tương tác điều khiển giữa các đơn vị
  - Đóng gói dữ liệu và cài đặt, bao gồm các vấn đề về phạm vi và khả năng nhìn thấy của các yếu tố chương trình
  - Thuật toán và cấu trúc dữ liệu

- Là một kỹ thuật để phân chia một hệ thống phần mềm thành nhiều mô-đun độc lập và tách rời, mà có khả năng thực hiện (các) nhiệm vụ một cách độc lập.
- Các mô-đun này có thể hoạt động như các cấu trúc cơ bản cho toàn bộ phần mềm.
  - Xu hướng thiết kế các mô-đun sao cho chúng có thể được thực thi và / hoặc được biên dịch riêng rẽ và độc lập.
- Thiết kế mô-đun tuân theo các quy tắc “chia để trị” giải quyết vấn đề



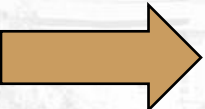
- Phần mềm là tập các mô-đun tương tác với nhau
- Mô-đun hóa đóng vai trò quan trọng để có được phần mềm có chất lượng với chi phí thấp
- Mục đích thiết kế hệ thống
  - Xác định các mô-đun có thể
  - Xác định tương tác giữa các mô-đun



- Ý nghĩa
  - Dễ xây dựng, dễ thay thế, sửa đổi
- Nguyên lý của mô đun hóa
  - Quan điểm chia để trị
    - $C(p1 + p2) > C(p1) + C(p2)$
    - $E(p1 + p2) > E(p1) + E(p2)$

Trong đó: C: độ phức tạp

E: nỗ lực thực hiện

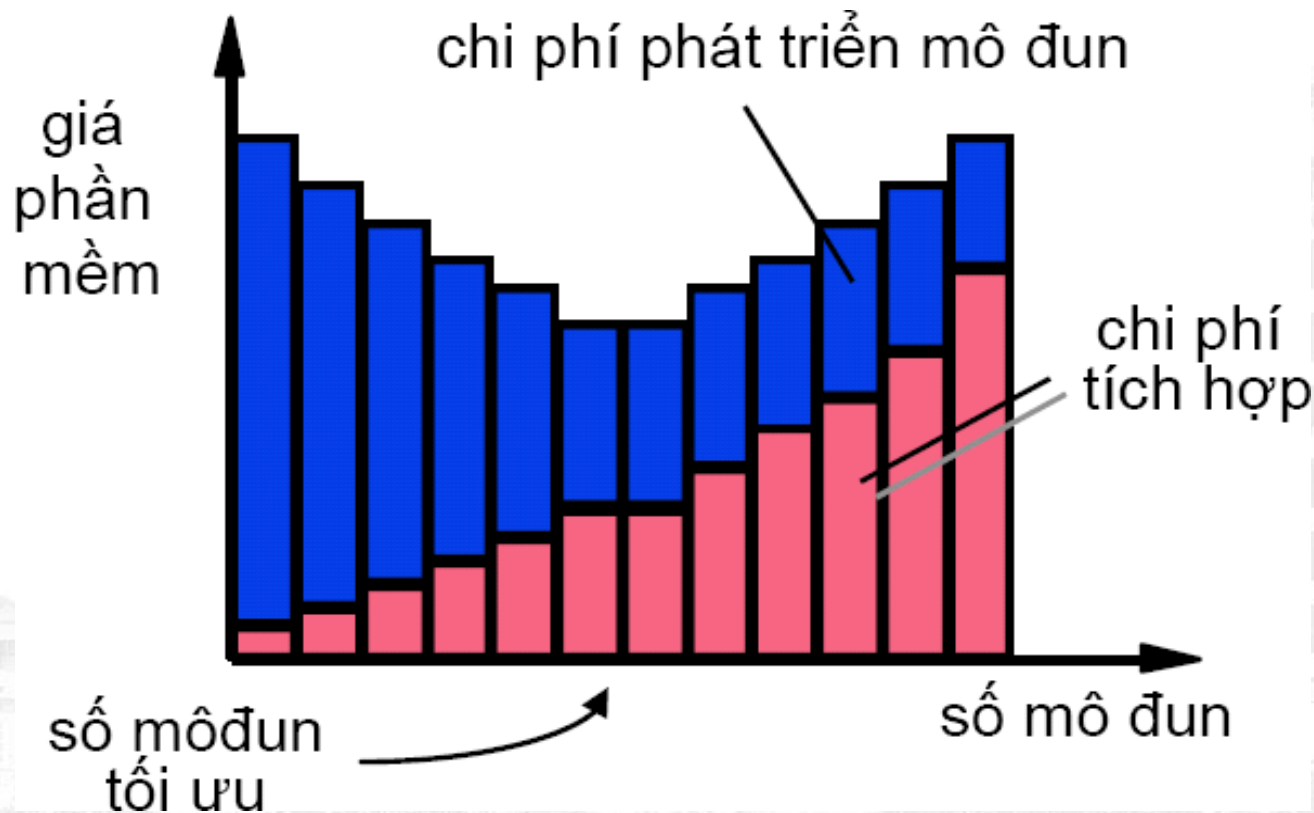
- 
- Giảm độ phức tạp, cục bộ, dễ sửa đổi
  - Có khả năng phát triển song song
  - Dễ sửa đổi, dễ hiểu nên dễ tái sử dụng

## ■ Ưu điểm của mô-đun hóa:

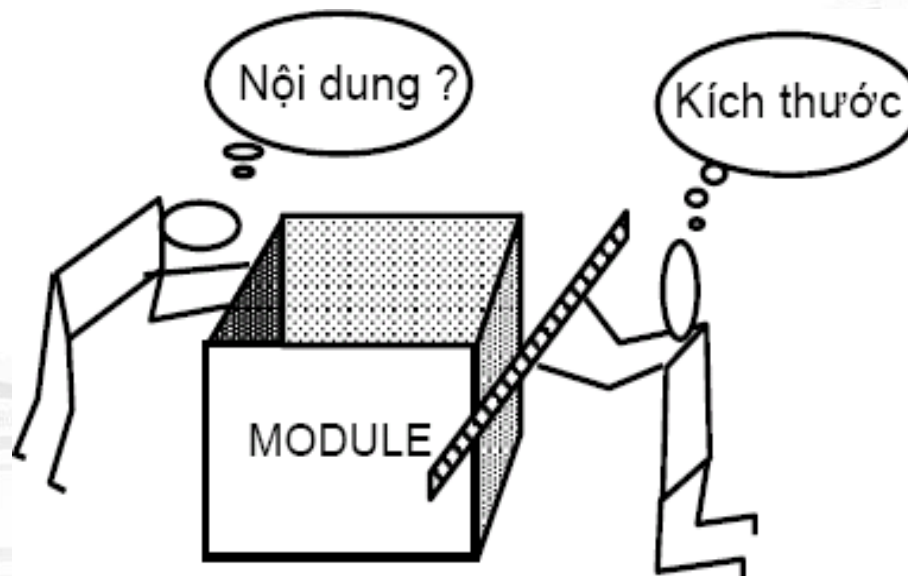
- Các thành phần nhỏ hơn dễ bảo trì hơn
- Chương trình có thể được phân chia dựa trên các khía cạnh chức năng
- Mức độ trừu tượng mong muốn có thể được đưa vào chương trình
- Các thành phần có độ kết dính cao (high cohesion) có thể được sử dụng lại
- Thực thi đồng thời có thể được thực hiện
- Khả năng bảo mật

# Thiết kế - hướng mô-đun (5)

- Số lượng mô-đun => cần xác định số mô-đun tối ưu



- Kích cỡ mô-đun xác định dựa trên khái niệm độc lập chức năng
  - Dễ hiểu, dễ sửa đổi, dễ tái sử dụng



# Tính đồng thời - Concurrency



- Được thực hiện bằng cách chia phần mềm thành nhiều đơn vị thực thi độc lập (như các mô-đun) và thực hiện song song.
  - Cung cấp khả năng cho phần mềm thực thi nhiều phần mã song song với nhau.
- Các lập trình viên và nhà thiết kế cần phải nhận ra các mô-đun có thể được thực hiện song song
- Ví dụ:
  - Tính năng kiểm tra chính tả trong trình xử lý văn bản là một mô-đun, thực thi cùng trình xử lý văn bản.



# Thiết kế phần mềm-sự thay đổi



- Sự thay đổi = tính chất đặc trưng của phần mềm
- Dự báo thay đổi là cần thiết
  - Giảm chi phí bảo trì
- Dự báo thay đổi là khó khăn
  - Sự thay đổi thường không xác định trước
  - Nhiều yếu tố thay đổi cùng lúc
  - Thời điểm thay đổi là khó có thể biết trước

# Thiết kế phần mềm-sự thay đổi



- Các yếu tố có thể thay đổi
  - Thuật toán
  - Cấu trúc dữ liệu
  - Biểu diễn dữ liệu bên ngoài
  - Thiết bị ngoại vi
  - Môi trường xã hội
  - Yêu cầu khách hàng

- Tạo mô hình cài đặt của phần mềm
  - Là phương tiện trao đổi thông tin để đảm bảo chất lượng
  - Dễ hiểu, dễ sửa đổi hơn mã chương trình
  - Có nhiều mức chi tiết; cung cấp cái nhìn tổng thể
- Nếu không có thiết kế; hoặc thiết kế tồi
  - Làm tăng công sức mã hóa
  - Làm tăng công sức bảo trì

# Coupling và Cohesion

- Khi một chương trình phần mềm được mô đun hóa, các tác vụ của nó được chia thành nhiều mô đun dựa trên một số đặc điểm.
- Các mô-đun là tập hợp các lệnh cùng nhau thực hiện một số nhiệm vụ.
- Mỗi mô-đun được xem xét là một thực thể duy nhất nhưng có thể có liên quan với nhau để cùng làm việc
- Có một số độ đo chất lượng của một thiết kế mô-đun và sự tương tác giữa các mô-đun.
  - Tính ghép nối (Coupling)
  - Tính gắn kết (Cohesion)

# Gắn kết – Cohesion (1)

- **Gắn kết** – độ đo sự phụ thuộc lẫn nhau của các thành phần trong một môđun
  - Sự gắn kết càng lớn, thiết kế chương trình càng tốt.
  - Mỗi module chỉ nên thực hiện một chức năng
  - Mọi thành phần nên tham gia thực hiện chức năng đó
  - Gắn kết cao thì tính cục bộ cao (độc lập chức năng);
  - Dễ hiểu, dễ sửa đổi

functional  
sequential  
communicational  
procedural  
temporal  
logical

coincidental

high and best  
ok  
still ok  
not bad at all  
still not bad at all  
still not bad at all  
  
lowest and worst by far



# Gắn kết – Cohesion (2)

## ■ Gắn Kết ngẫu nhiên (coincidental cohesion)

- Các thành phần không liên quan đến nhau
- Vd:

`print_next_line,`

`reverse_string_of_characters_comprising_second_parameter,`

`add_7_to_fifth_parameter, convert_fourth_parameter_to_floating_point`

## ■ Gắn kết logic (logical cohesion)

- Các thành phần làm chức năng logic tương tự
- Vd: hàm xử lí lỗi chung

## ■ Gắn kết thời điểm (temporal cohesion)

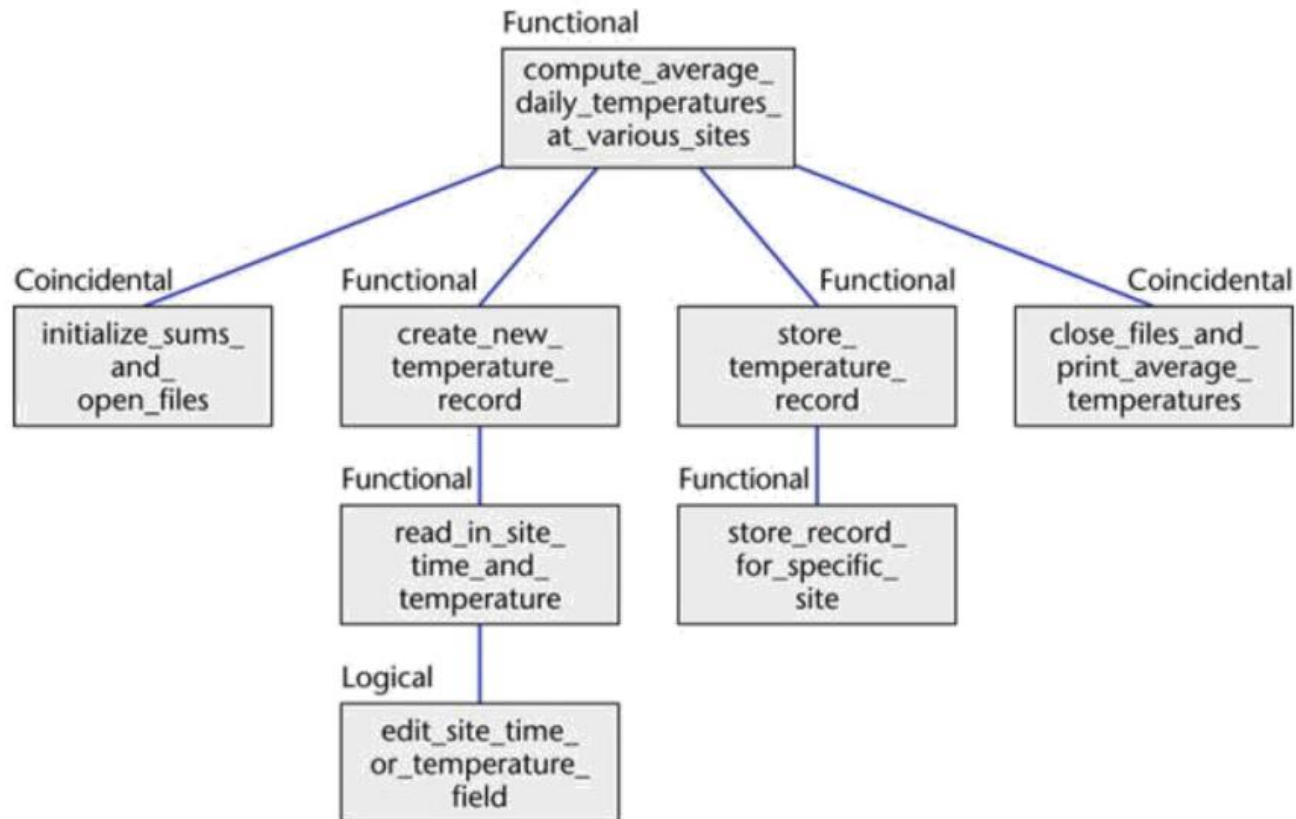
- Các thành phần hoạt động cùng thời điểm
- Vd: hàm khởi tạo (đọc dữ liệu, cấp phát bộ nhớ)

# Gắn kết – Cohesion (3)

- **Gắn kết thủ tục (procedural cohesion)**
  - Các thành phần tạo có một thứ tự xác định
  - Vd: tính lương cơ bản, tính phụ cấp, tính bảo hiểm
- **Gắn kết truyền thông (communicational cohesion)**
  - Các thành phần truy cập cùng dữ liệu
  - Vd: thống kê (tính max, min, mean, variation...)
- **Gắn kết tuần tự (sequential cohesion)**
  - Output của một thành phần là input của thành phần tiếp theo
  - Vd: ảnh màu -> đen trắng -> ảnh nén
- **Gắn kết chức năng (functional cohesion)**
  - Các thành phần cùng góp phần thực hiện một chức năng
  - Vd: sắp xếp

# Gắn kết – Cohesion (4)

## ■ Ví dụ:



# Tính ghép nối – Coupling (1)

## ■ Ghép nối – độ đo sự liên kết giữa các mô đun

- Mức độ quan hệ của các module
- Module nên ghép nối lỏng lẻo
- Càng lỏng lẻo càng dễ sửa đổi thiết kế

normal coupling

data coupling

stamp coupling

control coupling

common coupling

content coupling

loose and best

still very good

ok

ok

very bad

tight and worst

# Tính ghép nối – Coupling (2)



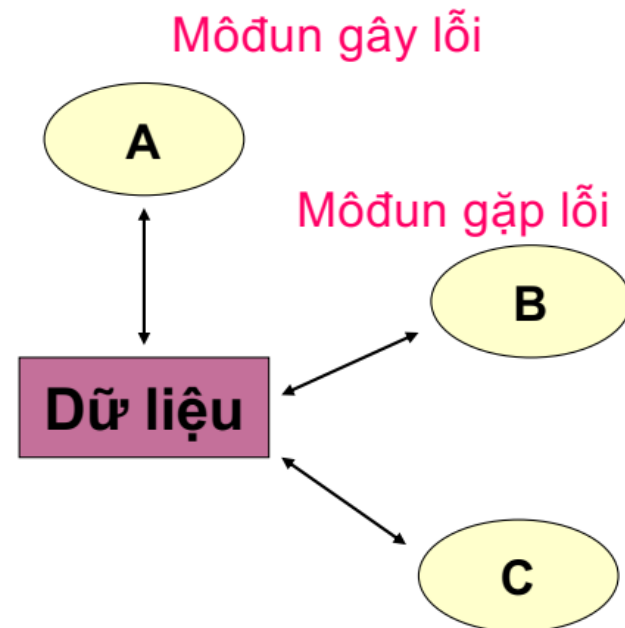
- Ghép nối nội dung (content coupling)
  - Là trường hợp xấu nhất
  - Khi một mô-đun có thể truy cập trực tiếp hoặc sửa đổi hoặc tham chiếu nội dung của mô-đun khác, nó được gọi là ghép nối nội dung.
  - Các mô-đun dùng lẫn dữ liệu của nhau
    - Các ngôn ngữ bậc thấp không có biến cục bộ
    - Lạm dụng lệnh goto
- Ví dụ 1:
  - Module p thay đổi một câu lệnh của module q
- Ví dụ 2:
  - Module p tham chiếu đến dữ liệu cục bộ của module q dưới dạng sự thay thế số bên trong q
- Ví dụ 3:
  - Module p phân thành một nhãn cục bộ trong module q



# Tính ghép nối – Coupling (3)

## ■ Ghép nối chung (common coupling)

- Khi nhiều mô-đun có quyền truy cập đọc và ghi vào một số dữ liệu toàn cục
  - Các module trao đổi dữ liệu thông qua biến tổng thể
  - Lỗi của module này có thể ảnh hưởng đến hoạt động của module khác
  - Khó sử dụng lại các module
- Ví dụ: Modules B và C truy cập cùng một CSDL và cả hai hai có thể read và write cùng một bản ghi



# Tính ghép nối – Coupling (4)

- Ghép nối điều khiển (control coupling)
  - Hai mô-đun được gọi là ghép nối điều khiển nếu một trong số chúng quyết định chức năng của mô-đun kia hoặc thay đổi luồng thực thi của chúng.
  - Các module trao đổi thông tin điều khiển
  - Làm cho thiết kế khó hiểu, khó sửa đổi, dễ nhầm

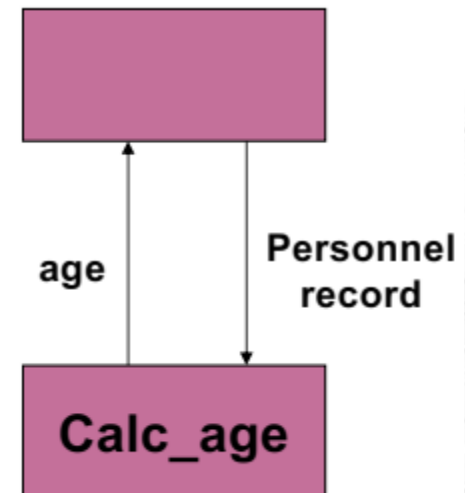
```
procedure PrintRec is  
begin  
    DisplayName (name, sex);  
    ....  
end PrintRec;
```

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

# Tính ghép nối – Coupling (5)

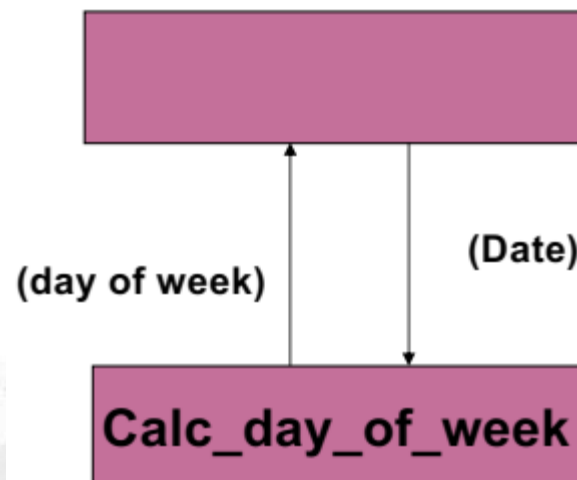
## ■ Ghép nối nhãn (stamp coupling)

- Khi nhiều mô-đun chia sẻ cấu trúc dữ liệu chung và hoạt động trên phần khác nhau của nó
  - Các module trao đổi thừa thông tin
  - Module có thể thực hiện chức năng ngoài ý muốn
  - Làm giảm tính thích nghi



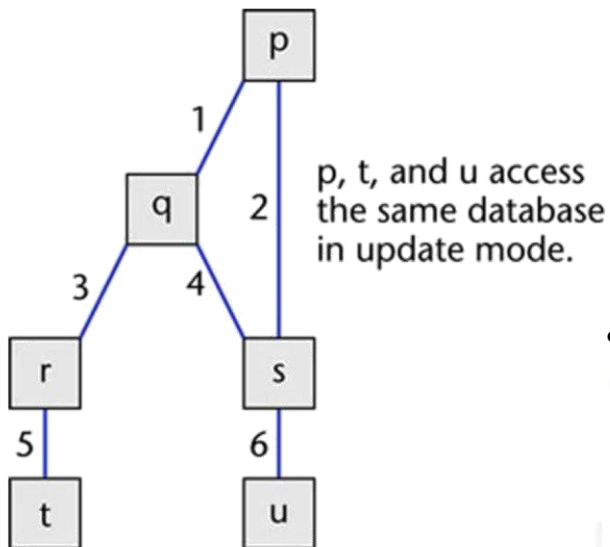
# Tính ghép nối – Coupling (6)

- Ghép nối dữ liệu (data coupling)
  - khi hai mô-đun tương tác với nhau bằng cách truyền dữ liệu (dưới dạng tham số). Nếu một mô-đun chuyển cấu trúc dữ liệu làm tham số, thì mô-đun nhận sẽ sử dụng tất cả các thành phần của nó.



# Tính ghép nối – Coupling (7)

## ■ Ví dụ



## ● Mô tả giao diện

Number	In	Out
1	aircraft_type	status_flag
2	list_of_aircraft_parts	—
3	function_code	—
4	list_of_aircraft_parts	—
5	part_number	part_manufacturer
6	part_number	part_name

## • Ghép nối giữa tất cả các cặp môđun

	q	r	s	t	u
p	Data	—	{ Data or stamp	Common	Common
q		Control	{ Data or stamp	—	—
r			—	Data	—
s				—	Data
t					Common



## ■ Tiêu chí

- Tính mô đun hóa
- Tính che dấu thông tin

## ■ Độ đo chất lượng thiết kế

- Tính ghép nối giữa các mô đun (coupling)
- Tính gắn kết các thành phần trong mô đun (cohesion)
- Tính hiểu được (understandability)
- Tính thích nghi được (adaptability)

=> Phụ thuộc bài toán, không có phương pháp tổng quát

- Đầu ra của quy trình thiết kế phần mềm là tài liệu thiết kế, mã giả, sơ đồ logic chi tiết, sơ đồ tiến trình và mô tả chi tiết tất cả các yêu cầu chức năng hoặc phi chức năng.
- Giai đoạn tiếp theo, đó là việc thực hiện phần mềm, phụ thuộc vào tất cả các kết quả đầu ra được đề cập ở trên.
- Cần phải xác minh đầu ra trước khi chuyển sang giai đoạn tiếp theo. Càng phát hiện sớm bất kỳ sai sót nào, nó càng tốt hoặc có thể không được phát hiện cho đến khi thử nghiệm sản phẩm.
- Nếu kết quả đầu ra của giai đoạn thiết kế ở dạng ký pháp hình thức, thì nên sử dụng các công cụ để xác minh nếu không có thể sử dụng đánh giá (review) thiết kế để xác minh (verification) và xác nhận (validation).
- Bằng cách tiếp cận xác minh có cấu trúc, người đánh giá có thể phát hiện các lỗi có thể gây ra bằng cách xem xét một số điều kiện. Một đánh giá thiết kế tốt là rất quan trọng đối với thiết kế phần mềm tốt, độ chính xác và chất lượng

# CHIẾN LƯỢC THIẾT KẾ

- Thiết kế phần mềm là một quá trình để khái niệm hóa các yêu cầu phần mềm thành triển khai cài đặt phần mềm.
- Thiết kế phần mềm lấy yêu cầu của người dùng làm thách thức và cố gắng tìm giải pháp tối ưu.
- Trong khi phần mềm đang được khái niệm hóa, một kế hoạch được vạch ra để tìm ra thiết kế tốt nhất có thể để thực hiện giải pháp dự định.

# Thiết kế có cấu trúc (1)



- Là khái niệm hoá vấn đề thành một số yếu tố được tổ chức tốt của giải pháp.
  - Tập trung vào thiết kế giải pháp.
  - Lợi ích của thiết kế có cấu trúc là, nó giúp hiểu rõ hơn về cách giải quyết vấn đề. Thiết kế có cấu trúc cũng giúp đơn giản hơn cho người thiết kế để tập trung vào vấn đề chính xác hơn.
- Thiết kế có cấu trúc chủ yếu dựa trên chiến lược “chia và trị” (divide and conquer), trong đó một vấn đề được chia thành nhiều vấn đề nhỏ và mỗi vấn đề nhỏ được giải quyết riêng lẻ cho đến khi toàn bộ vấn đề được giải quyết.
- Các phần nhỏ của vấn đề được giải quyết bằng các mô-đun giải pháp.
  - Thiết kế cấu trúc nhấn mạnh rằng các mô-đun này được tổ chức tốt để đạt được giải pháp chính xác.
- Các mô-đun được sắp xếp theo thứ bậc. Họ giao tiếp với nhau



# Thiết kế có cấu trúc (2)

- Các mô-đun được sắp xếp phân cấp, có giao tiếp với nhau.
- Một thiết kế có cấu trúc tốt luôn tuân theo một số quy tắc để giao tiếp giữa nhiều mô-đun:
- Tính gắn kết - nhóm tất cả các yếu tố liên quan đến chức năng.
- Tính ghép nối- giao tiếp giữa các mô-đun khác nhau.

=> Một thiết kế có cấu trúc tốt có tính gắn kết cao (high cohesion) và ghép nối thấp (low coupling).

# Thiết kế hướng chức năng (1)



- Hệ thống bao gồm nhiều hệ thống con nhỏ hơn được gọi là chức năng.
  - Các chức năng này có khả năng thực hiện nhiệm vụ quan trọng trong hệ thống.
  - Hệ thống được xem xét là là khung nhìn đỉnh của tất cả các chức năng.
- Thiết kế hướng chức năng kế thừa một số tính chất của thiết kế có cấu trúc trong đó sử dụng phương pháp chia để trị
  - Cơ chế thiết kế này chia toàn bộ hệ thống thành các chức năng nhỏ hơn, cung cấp phương tiện trừu tượng bằng cách che giấu thông tin và hoạt động của chúng ..
  - Các mô-đun chức năng này có thể chia sẻ thông tin với nhau bằng cách truyền thông tin và sử dụng thông tin toàn cục.

# Thiết kế hướng chức năng (2)



- Khi một chương trình gọi một hàm, hàm sẽ thay đổi trạng thái của chương trình, đôi khi các mô-đun khác không chấp nhận được.
- Thiết kế hướng chức năng hoạt động tốt khi không quan tâm trạng thái hệ thống và chương trình / chức năng làm việc trên các đầu vào thay vì trên trạng thái.
- Quá trình thiết kế
  - Toàn bộ hệ thống được xem như luồng dữ liệu trong hệ thống bằng sơ đồ luồng dữ liệu.
  - DFD mô tả cách các chức năng thay đổi dữ liệu và trạng thái của toàn bộ hệ thống.
  - Toàn bộ hệ thống được chia nhỏ một cách hợp lý thành các đơn vị nhỏ hơn được gọi là các chức năng trên cơ sở hoạt động của chúng trong hệ thống.
  - Mỗi chức năng sau đó được mô tả ở mức lớn.

# Thiết kế hướng đối tượng (1)



- Thiết kế hướng đối tượng làm việc xoay quanh các thực thể và đặc điểm của chúng thay vì các chức năng liên quan đến hệ thống phần mềm.
  - Chiến lược thiết kế này tập trung vào các thực thể và đặc điểm của nó.
  - Toàn bộ khái niệm về giải pháp phần mềm xoay quanh các thực thể tham gia.
- Các khái niệm quan trọng của Thiết kế hướng đối tượng:
  - Đối tượng - Tất cả các thực thể liên quan đến thiết kế giải pháp được gọi là các đối tượng. Ví dụ, người, ngân hàng, công ty và khách hàng được coi là đối tượng. Mỗi thực thể có một số thuộc tính liên quan đến nó và có một số phương thức để thực hiện trên các thuộc tính.
  - Lớp - Một lớp là một mô tả tổng quát về một đối tượng. Một đối tượng là một thể hiện của một lớp. Lớp định nghĩa tất cả các thuộc tính, mà một đối tượng có thể có và các phương thức, định nghĩa chức năng của đối tượng.
  - Các thuộc tính được lưu trữ dưới dạng các biến và chức năng được xác định bằng phương thức hoặc thủ tục.



# Thiết kế hướng đối tượng (2)



- Đóng gói - các thuộc tính (biến dữ liệu) và phương thức (thao tác trên dữ liệu) được gói lại với nhau được gọi là đóng gói.
  - Đóng gói không chỉ kết hợp thông tin quan trọng của một đối tượng với nhau, mà còn hạn chế quyền truy cập dữ liệu và phương thức từ thế giới bên ngoài. Đây được gọi là ẩn thông tin.
- Kế thừa - cho phép các lớp tương tự tổ chức phân cấp trong đó các lớp dưới hoặc lớp con có thể thực hiện và sử dụng lại các biến và phương thức được phép từ các lớp cha trực tiếp của chúng.
  - Thuộc tính này của OOD được gọi là thừa kế.
  - Giúp dễ dàng xác định lớp cụ thể và tạo các lớp tổng quát từ các lớp cụ thể.
- Đa hình - Các ngôn ngữ OOD cung cấp một cơ chế trong đó các phương thức thực hiện các nhiệm vụ tương tự nhưng khác nhau về đối số, có thể được gán cùng tên.
  - Điều này được gọi là đa hình, cho phép một giao diện duy nhất thực hiện các nhiệm vụ cho các loại khác nhau. Tùy thuộc vào cách hàm được gọi, phần mã tương ứng sẽ được thực thi.



# Thiết kế hướng đối tượng (3)



- Quá trình thiết kế phần mềm có thể được coi là một chuỗi các bước được xác định rõ. Mặc dù nó thay đổi theo cách tiếp cận thiết kế (hướng chức năng hoặc hướng đối tượng, nhưng nó có thể có các bước sau:
  - Một thiết kế giải pháp được tạo ra từ yêu cầu hoặc sơ đồ hệ thống được sử dụng trước đó và / hoặc sơ đồ tuần tự hệ thống.
  - Các đối tượng được xác định và nhóm thành các lớp thay mặt cho sự giống nhau về đặc tính thuộc tính.
  - Phân cấp lớp và quan hệ giữa chúng được xác định.
  - Khung ứng dụng được định nghĩa.

## ■ Thiết kế từ trên xuống

- Một hệ thống bao gồm nhiều hơn một hệ thống con và nó chứa một số thành phần. Các hệ thống con và các thành phần này có thể có tập hợp các hệ thống con và các thành phần và tạo cấu trúc phân cấp trong hệ thống.
- Thiết kế từ trên xuống lấy toàn bộ hệ thống phần mềm làm một thực thể và sau đó phân tách nó để đạt được nhiều hơn một hệ thống con hoặc thành phần dựa trên một số đặc điểm. Mỗi hệ thống con hoặc thành phần sau đó được coi là một hệ thống và phân hủy hơn nữa. Quá trình này tiếp tục cho đến khi đạt được mức thấp nhất của hệ thống trong hệ thống phân cấp từ trên xuống.
- Thiết kế từ trên xuống bắt đầu với một mô hình hệ thống tổng quát và tiếp tục xác định phần cụ thể hơn của nó. Khi tất cả các thành phần được cấu thành, toàn bộ hệ thống ra đời.
- Thiết kế từ trên xuống phù hợp hơn khi giải pháp phần mềm cần được thiết kế từ đầu và không biết chi tiết cụ thể.

## ■ Thiết kế từ dưới lên

- Mô hình thiết kế từ dưới lên bắt đầu với hầu hết các thành phần cơ bản và cụ thể nhất.
  - Nó tiến hành với việc xây dựng các thành phần cấp cao hơn bằng cách sử dụng các thành phần cơ bản hoặc cấp thấp hơn.
  - Nó tiếp tục tạo ra các thành phần cấp cao hơn cho đến khi hệ thống mong muốn không được phát triển thành một thành phần duy nhất. Với mỗi cấp độ cao hơn, mức trừu tượng tăng lên.
- Chiến lược từ dưới lên phù hợp hơn khi một hệ thống cần được tạo ra từ một số hệ thống hiện có, trong đó các phần tử nguyên thủy cơ bản có thể được sử dụng trong hệ thống mới hơn.

- Cả hai cách tiếp cận từ trên xuống và từ dưới lên đều không thực tế riêng lẻ. => **sử dụng kết hợp cả hai**

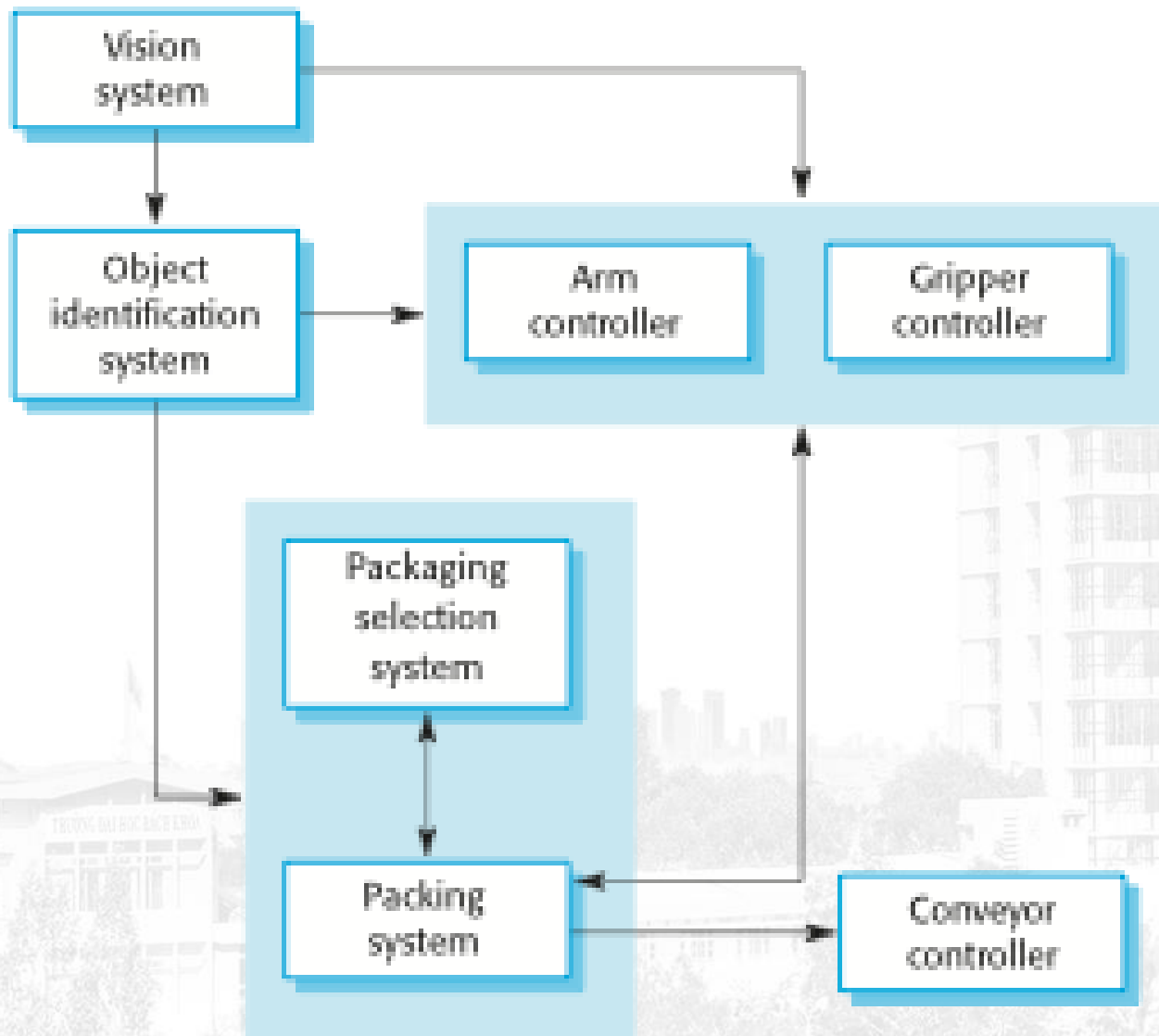
# THIẾT KẾ KIẾN TRÚC

- Quá trình thiết kế để xác định các hệ thống con tạo nên một hệ thống và khung để điều khiển và giao tiếp giữa các hệ thống con là thiết kế kiến trúc.
- Đầu ra của quá trình thiết kế này là một mô tả về kiến trúc phần mềm.



- Giai đoạn sớm của quá trình thiết kế hệ thống.
- Biểu diễn mối liên kết giữa đặc tả và quy trình thiết kế.
- Thường được thực hiện song song với một số hoạt động đặc tả.
- Liên quan đến việc xác định các thành phần chính của hệ thống và giao tiếp giữa chúng.

# Kiến trúc của hệ thống điều khiển robot đóng gói



- **Kiến trúc nhỏ** có liên quan đến kiến trúc của các chương trình cá nhân.
  - Ở cấp độ này, chúng ta quan tâm đến cách một chương trình riêng lẻ được phân tách thành các thành phần.
- **Kiến trúc lớn** liên quan đến kiến trúc của các hệ thống doanh nghiệp phức tạp bao gồm các hệ thống, chương trình và các thành phần chương trình khác.
  - Các hệ thống doanh nghiệp này được phân phối trên các máy tính khác nhau, có thể được sở hữu và quản lý bởi các công ty khác nhau.

- Truyền thông bên liên quan
  - Kiến trúc có thể được sử dụng như một tiêu điểm để thảo luận của các bên liên quan hệ thống.
- Phân tích hệ thống
  - Có nghĩa là phân tích xem hệ thống có thể đáp ứng các yêu cầu phi chức năng của nó hay không.
- Tái sử dụng quy mô lớn
  - Kiến trúc có thể được sử dụng lại trên một loạt các hệ thống
  - Kiến trúc dòng sản phẩm có thể được phát triển.

- Phương pháp sử dụng phổ biến biểu diễn kiến trúc: Các sơ đồ khối đơn giản, phi hình thức hiển thị các thực thể và các mối quan hệ
- Hạn chế: chúng thiếu ngữ nghĩa, không thể hiện các loại quan hệ giữa các thực thể cũng như các thuộc tính hữu hình của các thực thể trong kiến trúc.
- Phụ thuộc vào việc sử dụng các mô hình kiến trúc.
- Các yêu cầu đối với ngữ nghĩa mô hình phụ thuộc vào cách các mô hình được sử dụng.



# Sơ đồ “Box and line”

- Rất trừu tượng - không thể hiện bản chất của các mối quan hệ thành phần cũng như các thuộc tính có thể nhìn thấy bên ngoài của các hệ thống con.
- Tuy nhiên, hữu ích cho việc giao tiếp với các bên liên quan và lập kế hoạch dự án.

- Như một cách để tạo thuận tiện việc thảo luận về thiết kế hệ thống
  - Một khung nhìn kiến trúc mức cao của một hệ thống là rất hữu ích để giao tiếp với các bên liên quan của hệ thống và lập kế hoạch dự án vì nó không bị lộn xộn với chi tiết.
  - Các bên liên quan có thể liên kết đến nó và hiểu một khung nhìn trừu tượng về hệ thống.
    - Họ có thể thảo luận về toàn bộ hệ thống mà không bị nhầm lẫn bởi chi tiết.
- Như một cách để ghi lại một kiến trúc đã được thiết kế
  - Mục đích ở đây là tạo ra một mô hình hệ thống hoàn chỉnh cho thấy các thành phần khác nhau trong một hệ thống, giao diện và các kết nối của chúng.

- Thiết kế kiến trúc là một quy trình sáng tạo vì thế các quy trình khác nhau tùy thuộc vào loại hệ thống được phát triển.
- Một số quyết định chung liên quan tất cả các quy trình thiết kế và các quyết định này ảnh hưởng đến các đặc điểm phi chức năng của hệ thống.

# Các quyết định thiết kế kiến trúc

- Có một kiến trúc ứng dụng chung có thể được sử dụng?
- Hệ thống sẽ được phân chia như thế nào?
- Những phong cách kiến trúc nào là phù hợp?
- Cách tiếp cận nào sẽ được sử dụng để cấu trúc hệ thống?
- Làm thế nào hệ thống sẽ được phân tách thành các mô-đun?
- Chiến lược kiểm soát nào nên được sử dụng?
- Thiết kế kiến trúc sẽ được đánh giá như thế nào?
- Kiến trúc nên được tài liệu hoá như thế nào?

- Các hệ thống trong cùng một miền thường có kiến trúc tương tự phản ánh các khái niệm miền.
- Các dòng sản phẩm ứng dụng được xây dựng xung quanh một kiến trúc cốt lõi với các biến thể đáp ứng các yêu cầu cụ thể của khách hàng.
- Kiến trúc của một hệ thống có thể được thiết kế theo một trong những kiểu kiến trúc khác
  - nắm bắt được bản chất của một kiến trúc và có thể được khởi tạo theo những cách khác nhau.



- **Hiệu suất**
  - Cục bộ hoá các hoạt động quan trọng và giảm thiểu truyền thông.
  - Sử dụng các thành phần lớn hơn là các thành phần nhỏ (mịn).
- **Bảo mật**
  - Sử dụng một kiến trúc lớp với các tài sản quan trọng ở các lớp bên trong.
- **Sự an toàn**
  - Cục bộ hóa các tính năng đặc biệt an toàn ở một số ít hệ thống con.
- **Khả dụng**
  - Bao gồm các thành phần và cơ chế dự phòng cho khả năng chịu lỗi.
- **Bảo trì**
  - Sử dụng các thành phần nhỏ, có thể thay thế.

- Những khung nhìn hay quan điểm nào hữu ích khi thiết kế và tài liệu hoá một kiến trúc hệ thống?
- Những ký pháp nào nên được sử dụng để mô tả các mô hình kiến trúc?
- Mỗi mô hình kiến trúc chỉ thể hiện một khung nhìn hoặc quan điểm của hệ thống.
  - Có thể chỉ ra cách một hệ thống được phân tách thành các mô-đun, cách các xử lý thời gian chạy tương tác hoặc các cách khác nhau trong đó các thành phần hệ thống được phân phối trên một mạng.
  - Đối với cả thiết kế và tài liệu, cần biểu diễn nhiều khung nhìn của kiến trúc phần mềm.

# 4 + 1 view model of software architecture



- **Khung nhìn logic:** cho thấy các sự trừu tượng hoá trong hệ thống dưới dạng các đối tượng hoặc các lớp đối tượng.
- **Khung nhìn quy trình:** cho thấy làm thế nào, tại thời điểm chạy, hệ thống bao gồm các quá trình tương tác.
- **Khung nhìn phát triển:** cho thấy phần mềm được phân rã như thế nào để phát triển.
- **Khung nhìn vật lý:** cho thấy phần cứng hệ thống và cách các thành phần phần mềm được phân tán trên các bộ xử lý trong hệ thống.
- Liên quan đến việc sử dụng các trường hợp sử dụng hoặc kịch bản (+1)

- Các mẫu là một phương tiện để biểu diễn, chia sẻ và sử dụng lại tri thức.
- Một mẫu kiến trúc là một sự mô tả cách điều của các thiết kế tốt, đã được dùng thử và thử nghiệm trong các môi trường khác nhau.
- Các mẫu nên bao gồm thông tin về khi nào hữu ích và khi nào không hữu ích.
- Các mẫu có thể được biểu diễn bằng mô tả dạng bảng và đồ họa.

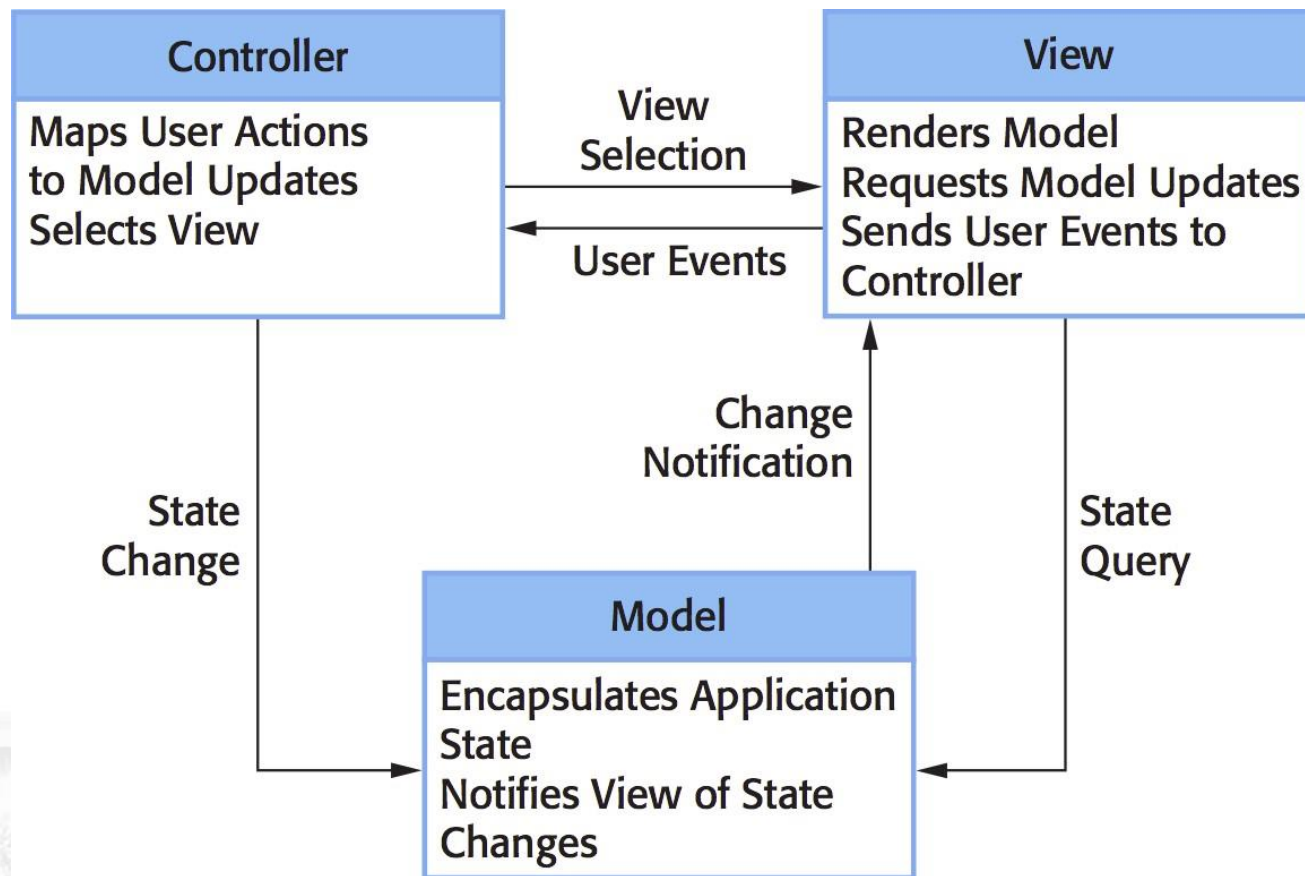
# Mẫu kiến trúc MVC Model-View-Controller



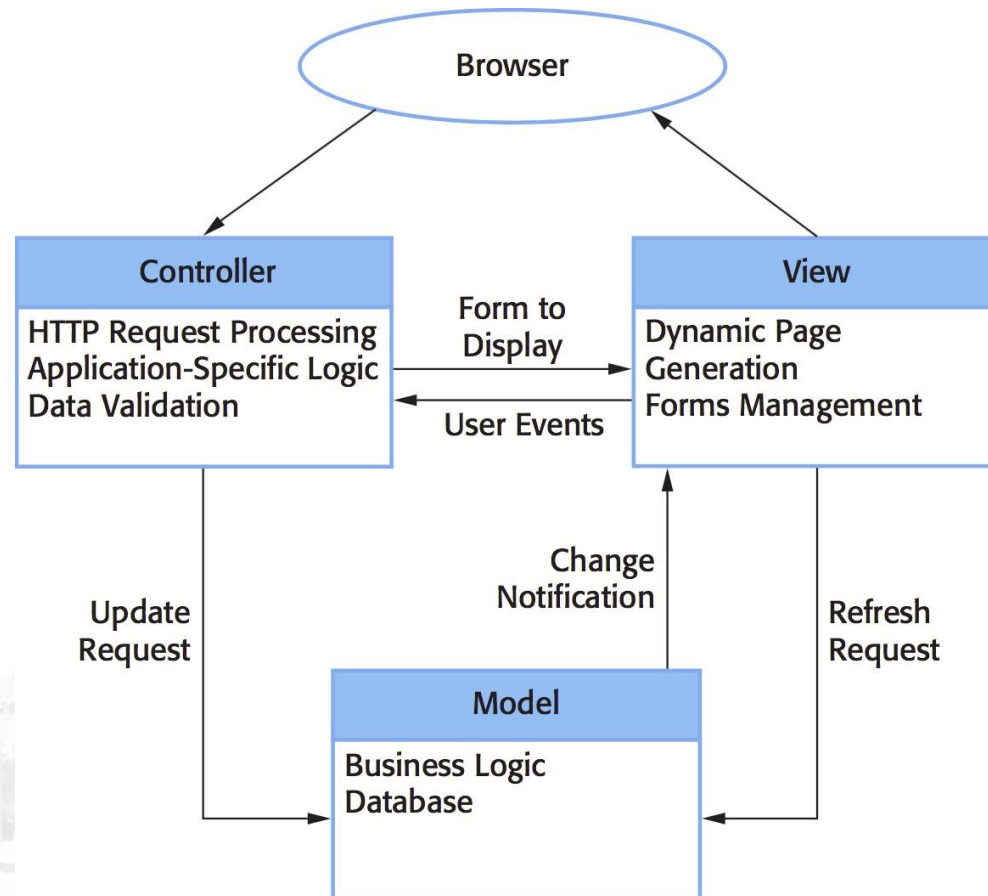
Tên	MVC (Model-View-Controller)
Mô tả	<p>Tách trình bày và tương tác khỏi dữ liệu hệ thống. Hệ thống được cấu trúc thành ba thành phần logic tương tác với nhau.</p> <p>Thành phần <b>Model</b> quản lý dữ liệu hệ thống và các hoạt động liên quan đến dữ liệu đó.</p> <p>Thành phần <b>View</b> xác định và quản lý cách trình bày dữ liệu cho người dùng.</p> <p>Thành phần <b>Controller</b> quản lý tương tác người dùng (ví dụ: nhấn phím, nhấp chuột, v.v.) và chuyển các tương tác này cho View và Model.</p>
Ví dụ	Kiến trúc của ứng dụng web sử dụng mẫu MVC
Sử dụng khi nào	Được sử dụng khi có nhiều cách để xem và tương tác với dữ liệu. Cũng được sử dụng khi các yêu cầu trong tương lai về tương tác và trình bày dữ liệu chưa được biết.
Ưu điểm	Cho phép dữ liệu thay đổi độc lập với cách biểu diễn của nó và ngược lại. Hỗ trợ trình bày cùng một dữ liệu theo các cách khác nhau với các thay đổi được thực hiện trong một sự trình bày được hiển thị trong tất cả chúng
Nhược điểm	Có thể làm phát sinh thêm bổ sung và tăng độ phức tạp mã khi mô hình dữ liệu và tương tác đơn giản.



# Tổ chức của MVC (Model-View-Controller)



## ■ Ví dụ:



- Được sử dụng để mô hình hoá giao tiếp của các hệ thống con.
- Tổ chức hệ thống thành một tập hợp các lớp (hoặc máy trừu tượng) mỗi lớp cung cấp một tập dịch vụ.
- Hỗ trợ sự phát triển tăng trưởng của các hệ thống con trong các lớp khác nhau. Khi một giao diện lớp thay đổi, chỉ có lớp liền kề bị ảnh hưởng.

Tên	Kiến trúc phân lớp
Mô tả	Tổ chức hệ thống thành các lớp với chức năng liên quan kết hợp với mỗi lớp. Một lớp cung cấp dịch vụ cho lớp trên nó vì thế các lớp cấp thấp nhất biểu diễn cho các dịch vụ cốt lõi mà có khả năng được sử dụng trong toàn hệ thống.
Ví dụ	Một mô hình lớp của một hệ thống để chia sẻ các tài liệu bản quyền được lưu trữ trong các thư viện khác nhau.
Sử dụng khi nào	Được sử dụng khi xây dựng các tiện ích mới trên các hệ thống hiện có; khi sự phát triển được thực hiện bởi một số nhóm với mỗi nhóm chịu trách nhiệm cho một lớp chức năng; khi có yêu cầu về bảo mật đa cấp.
Ưu điểm	Cho phép thay thế toàn bộ lớp miễn là giao diện được duy trì. Các tiện ích dự phòng (ví dụ: xác thực) có thể được cung cấp trong mỗi lớp để tăng độ tin cậy của hệ thống.
Nhược điểm	Trong thực tế, việc cung cấp một sự tách biệt rõ ràng giữa các lớp thường rất khó khăn và một lớp mức cao có thể phải tương tác trực tiếp với các lớp mức thấp hơn là thông qua lớp ngay bên dưới nó. Hiệu suất có thể là một vấn đề vì nhiều mức độ giải thích của một yêu cầu dịch vụ vì nó được xử lý ở mỗi lớp.

**User Interface**

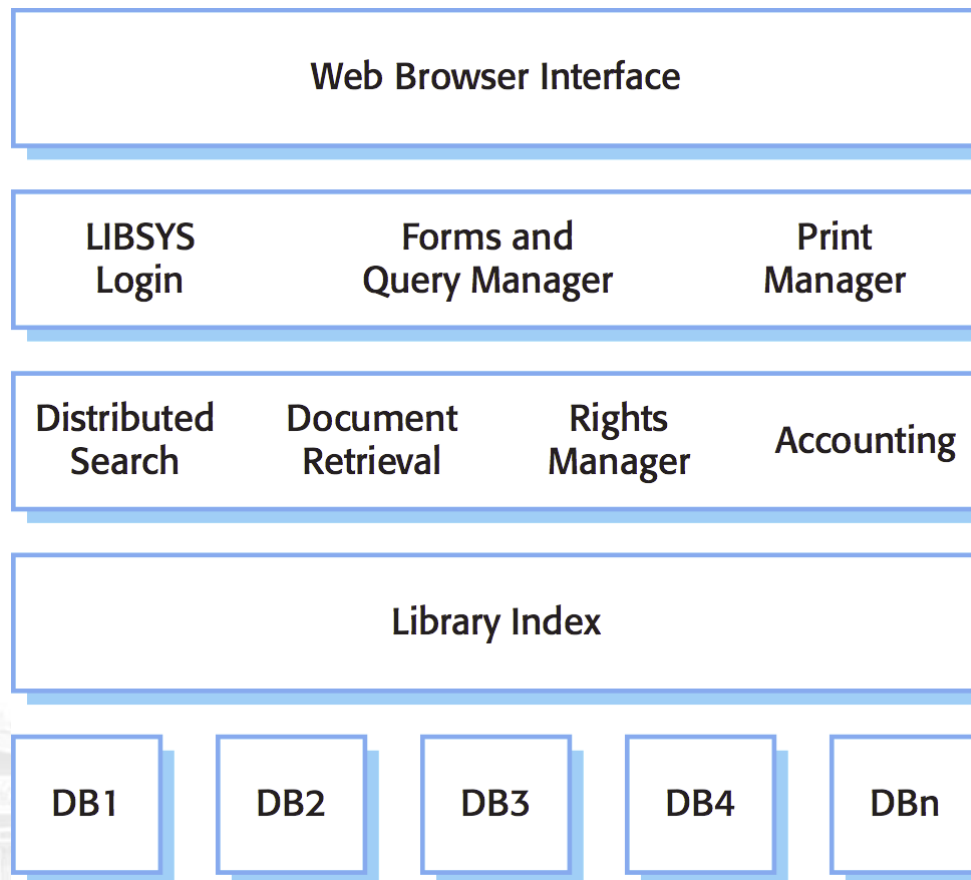
**User Interface Management  
Authentication and Authorization**

**Core Business Logic/Application Functionality  
System Utilities**

**System Support (OS, Database etc.)**



# Kiến trúc của hệ thống LIBSYS



# Những điểm chính

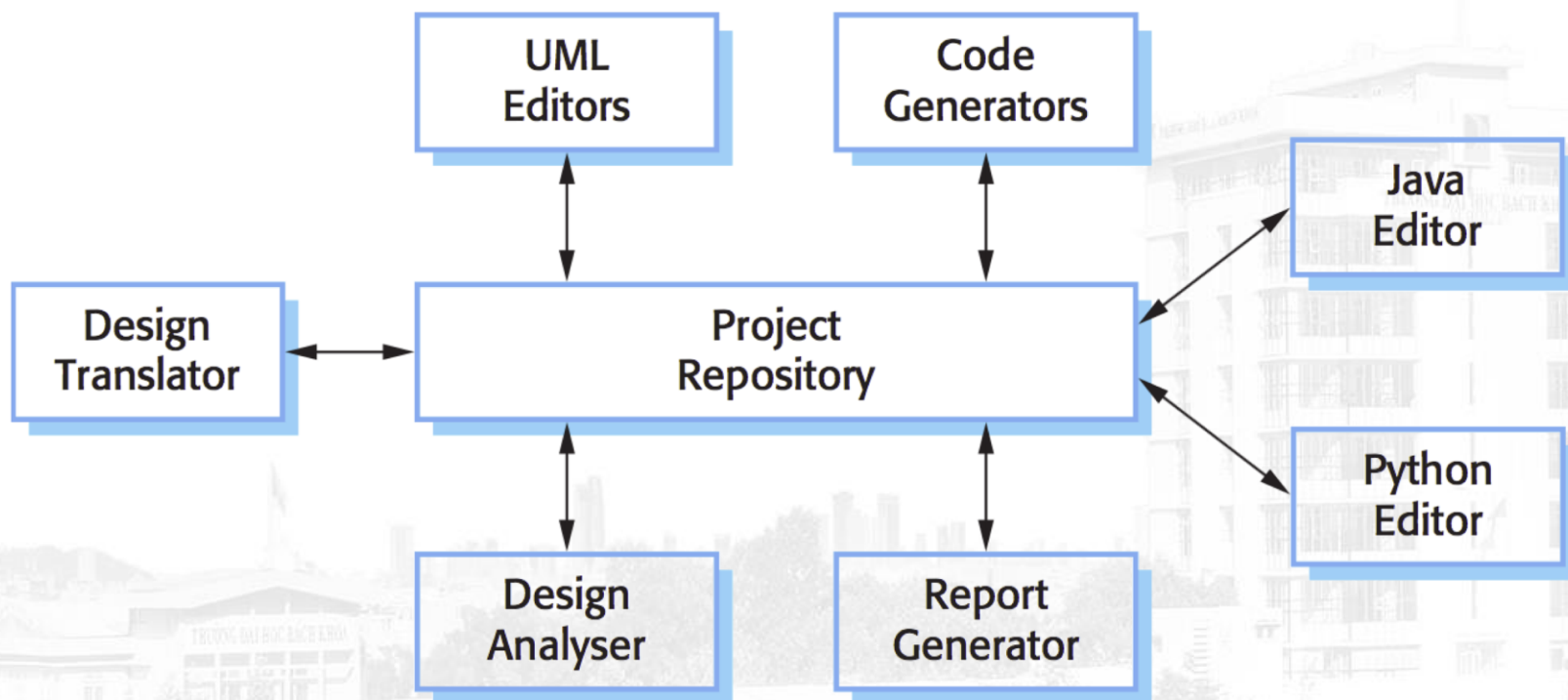
- Kiến trúc phần mềm là một mô tả về tổ chức một hệ thống phần mềm như thế nào
- Các quyết định thiết kế kiến trúc bao gồm các quyết định về loại ứng dụng, phân tằm của hệ thống, các kiểu kiến trúc sẽ được sử dụng.
- Kiến trúc có thể được tài liệu hoá từ nhiều quan điểm hoặc khung nhìn khác nhau như khung nhìn khái niệm, khung nhìn logic, khung nhìn quá trình và khung nhìn phát triển.
- Các mẫu kiến trúc là một phương tiện tái sử dụng kiến thức về kiến trúc hệ thống chung.
  - mô tả kiến trúc, giải thích khi nào nó có thể được sử dụng và mô tả những ưu điểm và nhược điểm của nó.

- Các hệ thống con cần trao đổi dữ liệu, có thể được thực hiện theo hai cách:
  - Dữ liệu chia sẻ được lưu trữ trong cơ sở dữ liệu trung tâm hoặc kho lưu trữ và có thể được truy cập bởi tất cả các hệ thống con;
  - Mỗi hệ thống con duy trì cơ sở dữ liệu riêng và truyền dữ liệu cho các hệ thống phụ con.
- Khi một lượng lớn dữ liệu được chia sẻ, mô hình kho lưu trữ được sử dụng phổ biến nhất, đây là một cơ chế chia sẻ dữ liệu hiệu quả.

Tên	Repository pattern
Mô tả	Tất cả dữ liệu trong một hệ thống được quản lý trong một kho lưu trữ trung tâm có thể truy cập được đối với tất cả các thành phần hệ thống. Các thành phần không tương tác trực tiếp, chỉ thông qua các kho lưu trữ.
Ví dụ	Ví dụ về IDE trong đó các thành phần sử dụng kho lưu trữ thông tin thiết kế hệ thống. Mỗi công cụ phần mềm tạo ra thông tin có sẵn để sử dụng bởi các công cụ khác.
Sử dụng khi nào	Nên sử dụng mẫu này khi có một hệ thống trong đó khối lượng thông tin lớn được tạo ra phải được lưu trữ trong một thời gian dài. Cũng có thể sử dụng nó trong các hệ thống điều hướng dữ liệu trong đó việc đưa dữ liệu vào kho lưu trữ sẽ kích hoạt một hành động hoặc công cụ.
Ưu điểm	Các thành phần có thể độc lập, không cần biết về sự tồn tại của các thành phần khác. Những thay đổi được thực hiện bởi một thành phần có thể được truyền tới tất cả các thành phần. Tất cả dữ liệu có thể được quản lý một cách nhất quán (ví dụ: các bản sao lưu được thực hiện cùng một lúc) vì tất cả đều ở một nơi.
Nhược điểm	Kho lưu trữ là một điểm thất bại duy nhất vì vậy các vấn đề trong kho lưu trữ ảnh hưởng đến toàn bộ hệ thống. Có thể không hiệu quả trong việc tổ chức tất cả các giao tiếp thông qua kho lưu trữ. Phân phối kho lưu trữ trên một số máy tính có thể khó khăn.



## ■ Ví dụ: Kiến trúc kho cho IDE

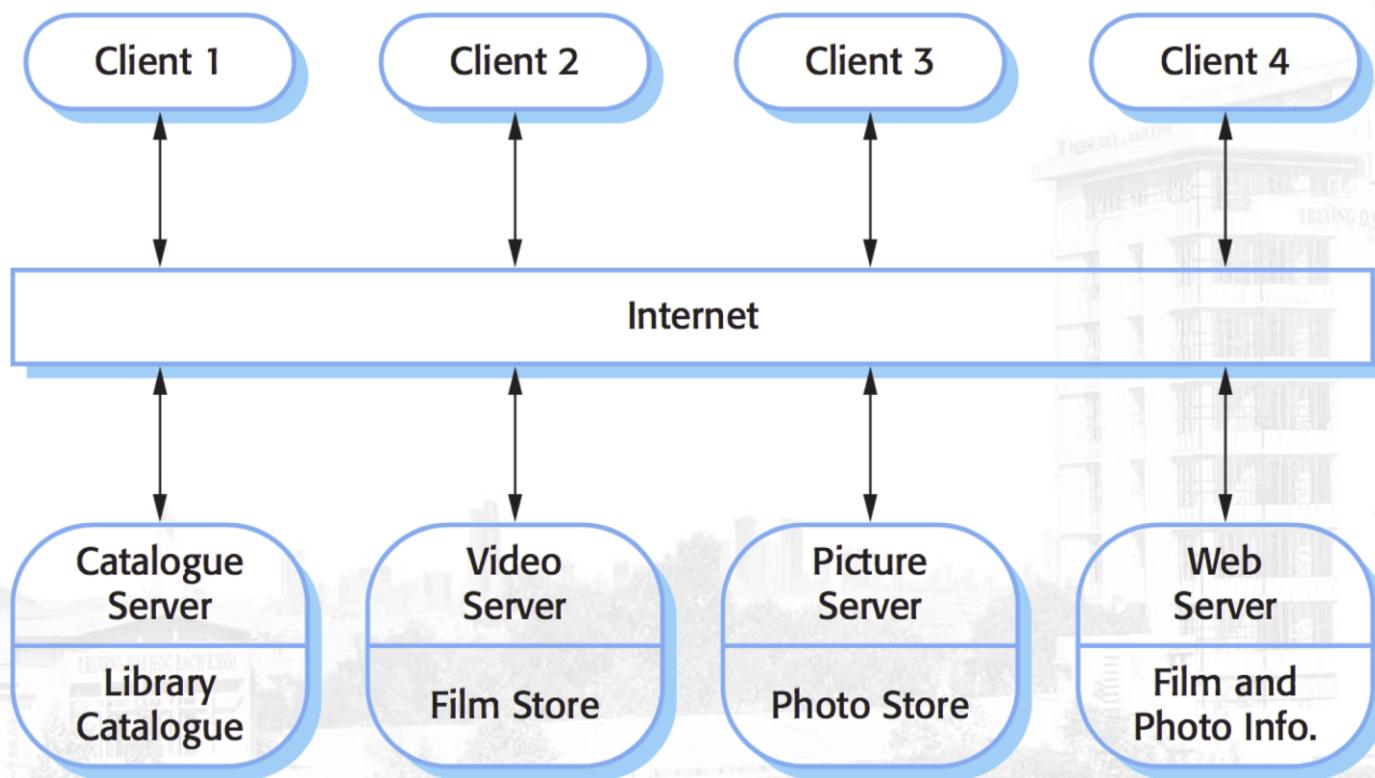




# Kiến trúc Client-server

Tên	Kiến trúc Client-server
Mô tả	Trong kiến trúc <b>client-server</b> , chức năng của hệ thống được tổ chức thành các dịch vụ, với mỗi dịch vụ được phân phối từ một máy chủ riêng biệt. Khách hàng là người dùng của các dịch vụ này và truy cập máy chủ để sử dụng chúng.
Ví dụ	ví dụ về thư viện phim và video / DVD được tổ chức cấu trúc hệ thống client-server
Sử dụng khi nào	Được sử dụng khi dữ liệu trong cơ sở dữ liệu dùng chung phải được truy cập từ một loạt các vị trí. Bởi vì các máy chủ có thể được sao chép, cũng có thể được sử dụng khi tải trên hệ thống thay đổi.
Ưu điểm	Ưu điểm chính của mô hình này là các máy chủ có thể được phân phối trên một mạng. Chức năng chung (ví dụ: dịch vụ in) có thể có sẵn cho tất cả khách hàng và không cần phải thực hiện bởi tất cả các dịch vụ.
Nhược điểm	Mỗi dịch vụ là một điểm thất bại đơn lẻ nên dễ bị từ chối tấn công dịch vụ hoặc lỗi máy chủ. Hiệu suất có thể không thể đoán trước vì nó phụ thuộc vào mạng cũng như hệ thống. Có thể có vấn đề quản lý nếu máy chủ được sở hữu bởi các tổ chức khác nhau.

- Ví dụ: Kiến trúc client-server cho thư viện phim



# Kiến trúc “Pipe and filter”

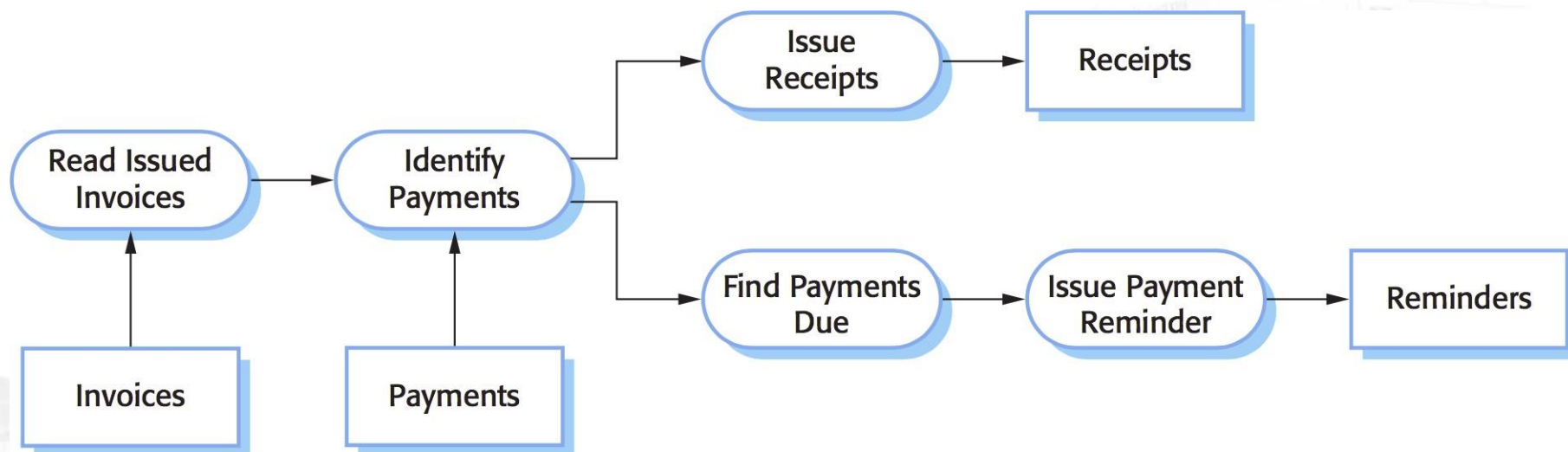
- Biến đổi chức năng xử lý đầu vào để tạo ra đầu ra.
- Có thể được gọi là một mô hình đường ống và bộ lọc (như trong UNIX shell).
- Các biến thể của phương pháp này rất phổ biến. Khi các phép biến đổi là tuần tự, đây là một mô hình tuần tự lô được sử dụng rộng rãi trong các hệ thống xử lý dữ liệu.
- Không thực sự phù hợp cho các hệ thống tương tác.

# Mẫu “pipe and filter”

Tên	Pipe and filter
Mô tả	<p>Việc xử lý dữ liệu trong một hệ thống được tổ chức sao cho mỗi thành phần xử lý (bộ lọc) rời rạc và thực hiện một kiểu chuyển đổi dữ liệu.</p> <p>Các luồng dữ liệu (như trong một đường ống) từ thành phần này sang thành phần khác để xử lý.</p>
Ví dụ	<p>ví dụ về hệ thống đường ống và bộ lọc được sử dụng để xử lý hóa đơn.</p>
Sử dụng khi nào	<p>Được sử dụng phổ biến trong các ứng dụng xử lý dữ liệu (cả xử lý theo lô và theo giao dịch) trong đó các đầu vào được xử lý ở các giai đoạn riêng biệt tạo các đầu ra tương ứng.</p>
Ưu điểm	<p>Dễ hiểu và hỗ trợ tái sử dụng. Tiến trình công việc phù hợp với cấu trúc của nhiều quy trình kinh doanh. Sự tiến hóa dễ dàng bằng cách bổ sung các phép biến đổi. Có thể được thực hiện như là một hệ thống tuần tự hoặc đồng thời.</p>
Nhược điểm	<p>Định dạng truyền dữ liệu cần được thống nhất giữa các biến đổi giao tiếp nhau. Mỗi phép biến đổi phải phân tích cú pháp các đầu vào của nó và tách đầu ra của nó về dạng đã thỏa thuận. Điều này làm tăng chi phí hệ thống và có thể có nghĩa là không thể sử dụng lại các phép biến đổi chức năng sử dụng các cấu trúc dữ liệu không tương thích.</p>

# Kiến trúc “pipe and filter”

- Ví dụ sử dụng kiến trúc “pipe and filter” trong xử lý hóa đơn.





- Các hệ thống ứng dụng được thiết kế để đáp ứng nhu cầu tổ chức.
- Khi các nghiệp vụ có nhiều điểm chung, các hệ thống ứng dụng cũng có xu hướng có một kiến trúc chung phản ánh các yêu cầu của ứng dụng.
- Kiến trúc ứng dụng chung là kiến trúc cho một loại hệ thống phần mềm có thể được cấu hình và điều chỉnh để tạo ra một hệ thống đáp ứng các yêu cầu cụ thể.

# Các kiến trúc ứng dụng



- Là điểm khởi đầu cho thiết kế kiến trúc.
- Là một checklist thiết kế
- Như một cách tổ chức công việc của nhóm phát triển.
- Là một phương tiện để đánh giá các thành phần để tái sử dụng.
- Là một từ vựng để trao đổi về các loại ứng dụng.

# Ví dụ về các loại ứng dụng



- Ứng dụng xử lý dữ liệu
  - Các ứng dụng hướng dữ liệu xử lý dữ liệu theo lô mà không có sự can thiệp rõ ràng của người dùng trong quá trình xử lý.
- Ứng dụng xử lý giao dịch
  - Các ứng dụng tập trung vào dữ liệu xử lý các yêu cầu của người dùng và cập nhật thông tin trong cơ sở dữ liệu hệ thống.
- Hệ thống xử lý sự kiện
  - Các ứng dụng trong đó các hành động của hệ thống phụ thuộc vào việc diễn giải các sự kiện từ môi trường hệ thống.
- Hệ thống xử lý ngôn ngữ
  - Các ứng dụng mà ý định của người dùng được mô tả bằng ngôn ngữ hình thức được hệ thống xử lý và giải thích.

# Các ví dụ kiểu ứng dụng

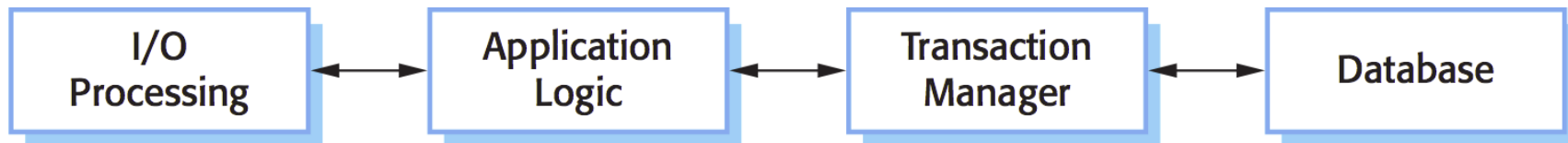


- Hệ thống xử lý giao dịch
  - Hệ thống thương mại điện tử;
  - Hệ thống đặt phòng.
- Hệ thống xử lý ngôn ngữ
  - Trình biên dịch;
  - Thông dịch lệnh.

- Xử lý yêu cầu của người dùng về thông tin từ cơ sở dữ liệu hoặc yêu cầu cập nhật cơ sở dữ liệu.
- Từ góc độ người dùng, một giao dịch là:
  - Bất kỳ chuỗi hoạt động thỏa mãn một mục tiêu;
  - Ví dụ: tìm thời gian của các chuyến bay từ London đến Paris.
- Hệ thống xử lý giao dịch thường là hệ thống tương tác trong đó người dùng thực hiện các yêu cầu không đồng bộ cho dịch vụ, sau đó được xử lý bởi người quản lý giao dịch.

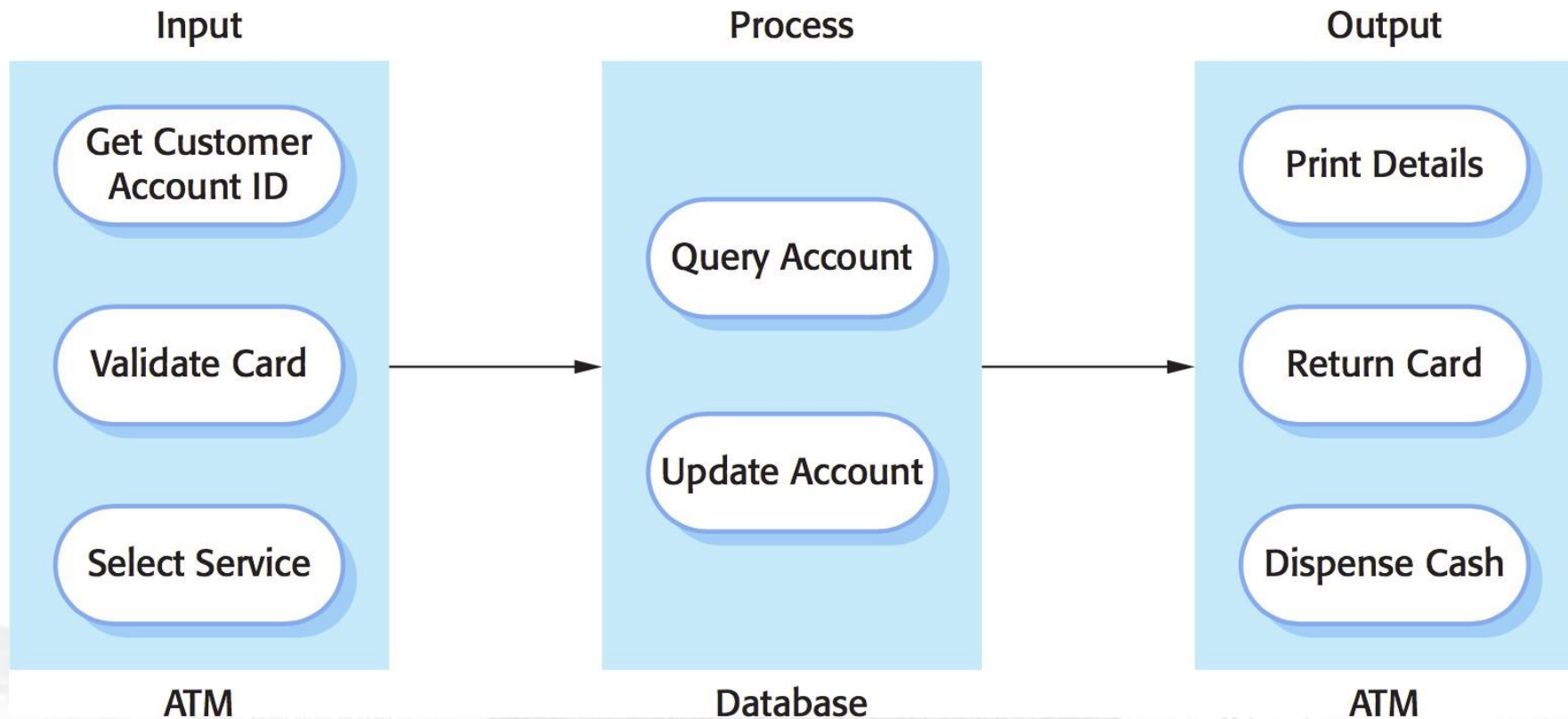


# Cấu trúc của ứng dụng xử lý giao dịch



- Người dùng đưa ra yêu cầu cho hệ thống thông qua thành phần xử lý I / O.
- Yêu cầu được xử lý bởi một số logic cụ thể của ứng dụng.
- Một giao dịch được tạo và chuyển đến một trình quản lý giao dịch, thường được nhúng trong hệ thống quản lý cơ sở dữ liệu
- Sau khi người quản lý giao dịch đảm bảo rằng giao dịch được hoàn thành đúng, nó báo hiệu cho ứng dụng xử lý đã kết thúc.

# Kiến trúc phần mềm cho hệ thống ATM



- Hệ thống thông tin có kiến trúc chung có thể được tổ chức thành kiến trúc phân lớp.
- Đây là các hệ thống dựa trên giao dịch vì sự tương tác với các hệ thống này thường liên quan đến các giao dịch cơ sở dữ liệu.
- Các lớp bao gồm:
  - Giao diện người dùng
  - Thông tin liên lạc người dùng
  - Lấy thông tin
  - Cơ sở dữ liệu hệ thống

# Hệ thống thông tin trên web



- Các hệ thống quản lý thông tin và tài nguyên hiện nay thường là các hệ thống dựa trên web, giao diện người dùng được triển khai bằng trình duyệt web.
- Ví dụ:
  - hệ thống thương mại điện tử là hệ thống quản lý tài nguyên dựa trên Internet, chấp nhận đơn đặt hàng điện tử cho hàng hóa hoặc dịch vụ và sau đó sắp xếp việc giao các hàng hóa hoặc dịch vụ này cho khách hàng.
  - Trong hệ thống thương mại điện tử, lớp dành riêng cho ứng dụng bao gồm chức năng bổ sung hỗ trợ giỏ hàng, trong đó người dùng có thể đặt một số mặt hàng trong các giao dịch riêng biệt, sau đó thanh toán tất cả chúng trong một giao dịch.

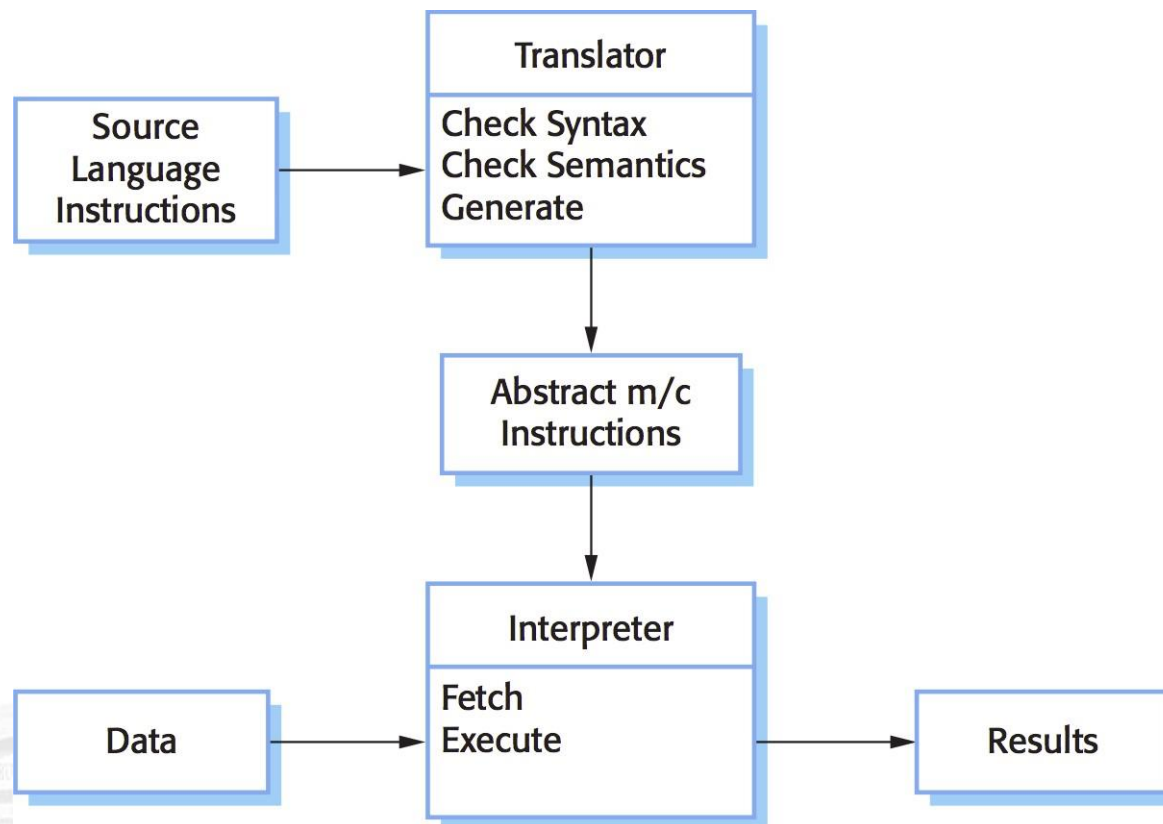


- Các hệ thống này thường được triển khai kiến trúc client- server nhiều tầng
  - Máy chủ web (web server) chịu trách nhiệm cho tất cả các giao tiếp người dùng, với giao diện người dùng được triển khai bằng trình duyệt web;
  - Máy chủ ứng dụng (application server) chịu trách nhiệm triển khai logic cụ thể của ứng dụng cũng như các yêu cầu lưu trữ và truy xuất thông tin;
  - Máy chủ cơ sở dữ liệu (database server) chuyển thông tin đến và đi từ cơ sở dữ liệu và xử lý quản lý giao dịch.



- Chấp nhận một ngôn ngữ tự nhiên hoặc nhân tạo làm đầu vào và tạo ra một số cách biểu diễn khác của ngôn ngữ đó.
- Có thể bao gồm một thông dịch viên để thực hiện các chỉ dẫn bằng ngôn ngữ đang được xử lý.
- Được sử dụng trong các tình huống trong đó cách dễ nhất để giải quyết vấn đề là mô tả thuật toán hoặc mô tả dữ liệu hệ thống

# Kiến trúc của hệ xử lý ngôn ngữ tự nhiên



# Những điểm chính

- Các mô hình kiến trúc ứng dụng giúp chúng ta hiểu và so sánh các ứng dụng, xác nhận thiết kế hệ thống ứng dụng và đánh giá các thành phần quy mô lớn để tái sử dụng.
- Hệ thống xử lý giao dịch là hệ thống tương tác cho phép thông tin trong cơ sở dữ liệu được truy cập và sửa đổi từ xa bởi một số người dùng.
- Hệ thống xử lý ngôn ngữ được sử dụng để dịch văn bản từ ngôn ngữ này sang ngôn ngữ khác và để thực hiện các hướng dẫn được chỉ định trong ngôn ngữ đầu vào. Chúng bao gồm một trình dịch và một máy trừu tượng thực thi ngôn ngữ được tạo.

# THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

- Là giao diện ứng dụng front-end mà người dùng tương tác để sử dụng phần mềm.
  - Người dùng có thể thao tác và điều khiển phần mềm cũng như phần cứng bằng giao diện người dùng.
  - giao diện người dùng được tìm thấy ở hầu hết mọi nơi có công nghệ kỹ thuật số, từ máy tính, điện thoại di động, ô tô, máy nghe nhạc, máy bay, tàu thủy, v.v.
- Giao diện người dùng là một phần của phần mềm và được thiết kế theo cách mà nó được mong đợi sẽ cung cấp cho người dùng cái nhìn sâu sắc về phần mềm.
- UI cung cấp nền tảng cơ bản cho tương tác giữa người và máy tính.
  - UI có thể là đồ họa, dựa trên văn bản, dựa trên video âm thanh, tùy thuộc vào sự kết hợp phần cứng và phần mềm.
  - UI có thể là phần cứng hoặc phần mềm hoặc kết hợp cả hai.

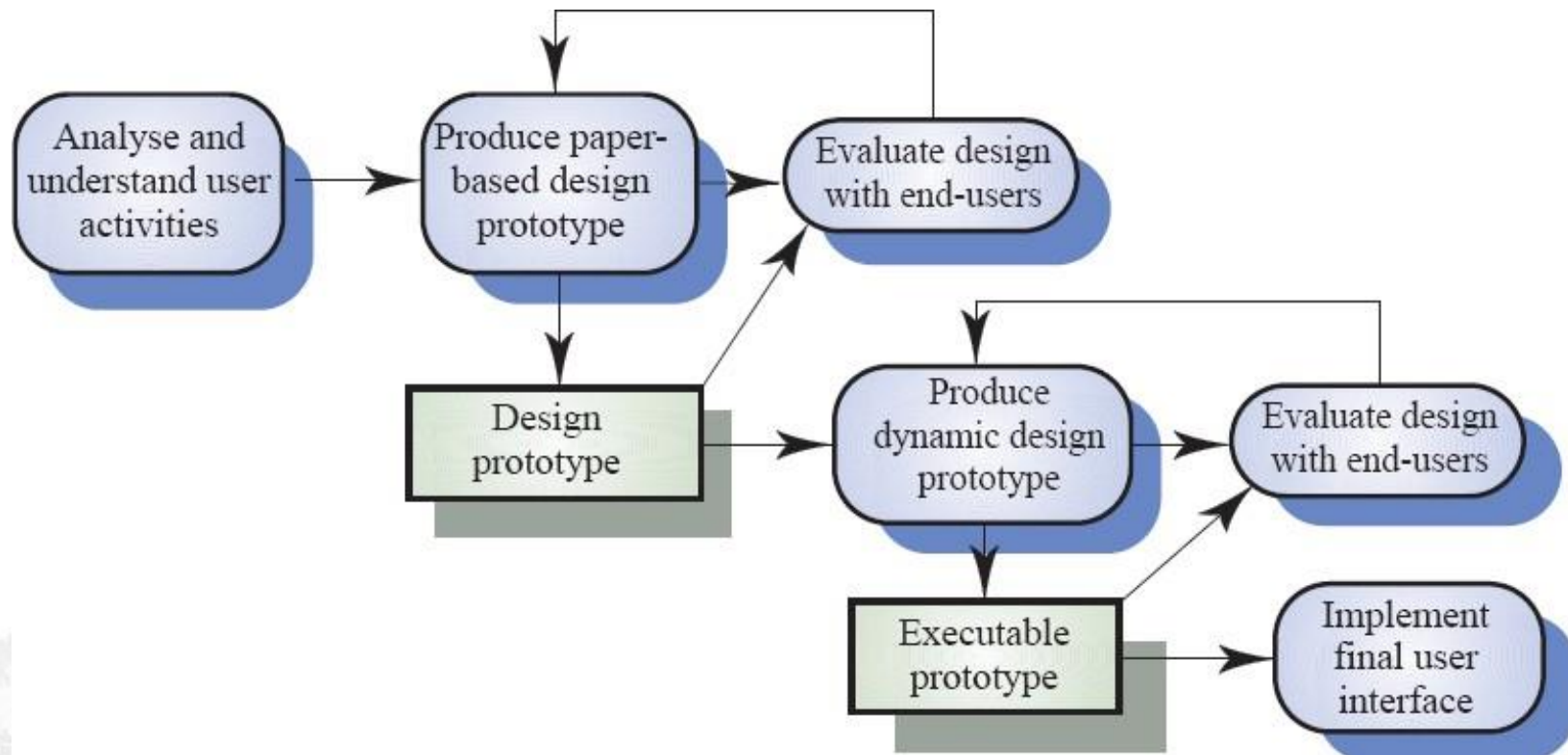


## Thiết kế giao diện người dùng (User Interface)

- Là một khâu trong thiết kế phần mềm
  - Hướng người dùng
  - Làm bản mẫu, người dùng đánh giá
- Có vai trò quan trọng
  - Cho phép người dùng sử dụng hệ thống
  - Quyết định hiệu năng hệ thống
  - Người dùng đánh giá hệ thống qua giao diện

=> Cần dễ sử dụng + hiệu quả

# Thiết kế giao diện người dùng



- Phần mềm trở nên phổ biến hơn nếu giao diện người dùng là:
  - Hấp dẫn
  - Đơn giản để sử dụng
  - Đáp ứng trong thời gian ngắn
  - Rõ ràng để hiểu
  - Phù hợp trên tất cả các màn hình giao tiếp

Yếu tố người dùng cần được xem xét trong thiết kế

- Nhu cầu của người dùng hệ thống
- Kinh nghiệm, năng lực
  - khả năng dùng bàn phím, mouse,...
  - tốc độ phản ứng, khả năng nhớ thao tác...
- Sở thích, văn hóa, lứa tuổi
  - màu sắc, ngôn ngữ, biểu tượng

Một số nguyên lý thiết kế giao diện người dùng

- Tính thân thiện người dùng
- Tính nhất quán
- Tính ít bất ngờ
- Tính phục hồi
- Hướng dẫn người dùng
- Tính đa dạng về người dùng



Hai vấn đề chính cần giải quyết

- ***Tương tác người dùng***: cách người dùng đưa thông tin vào cho hệ thống
- ***Biểu diễn thông tin***: cách hệ thống trình diễn thông tin cho người dùng

=>Giải pháp được xem xét theo góc độ

- Thiết bị tương tác người dùng
- Cách hệ thống trình diễn - chủng loại giao diện
- Mô hình tương tác

## Thiết bị vào và ra

- Màn hình
- Bàn phím
- Mouse, bút từ, ...
- Màn hình cảm biến
- Mic/Speaker
- Smart cards,...

=> Cả thiết bị lẫn phương thức đều đang tiến hóa - nhận dạng tiếng nói, chữ viết...

# Thiết kế GDND – Loại giao diện



- UI được chia thành hai loại:
  - Giao diện dòng lệnh
  - Giao diện đồ họa người dùng

- Là phương thức tương tác đầu tiên
  - Nhập lệnh/dữ liệu từ bàn phím
  - CLI cung cấp một dấu nhắc lệnh, nơi người dùng nhập lệnh và nhập liệu cho hệ thống.
  - Người dùng cần nhớ cú pháp của lệnh và cách sử dụng.
  - CLI trước đó không được lập trình để xử lý lỗi người dùng một cách hiệu quả.
- Lệnh là một tham chiếu dựa trên văn bản đến tập hợp các chỉ dẫn (instruction), dự kiến sẽ được hệ thống thực thi. Có các phương thức như macro, script giúp người dùng dễ dàng thao tác.

- Dễ cài đặt so với GUI
  - thực hiện thông qua hàm chuẩn của ngôn ngữ
  - sử dụng ít tài nguyên máy tính hơn so với GUI.
- Có khả năng tổ hợp lệnh để tạo các lệnh phức tạp
  - phối hợp các filter, tạo các lô xử lý (batch)
  - có thể lập trình bằng (Unix) shell
  - có thể tự động hóa
- Thao tác thực hiện tuần tự
  - khó sửa lỗi thao tác trước đó
- Không phù hợp với người dùng ít kinh nghiệm
  - khó học, khó nhớ
  - dễ nhầm
  - đòi hỏi kỹ năng sử dụng bàn phím



# Giao diện dòng lệnh

Command Parameters

```
Pinaca:lib gopal$ ls -al
total 12264
drwxr-xr-x 21 gopal staff 714 Jul 2 2013 .
drwxr-xr-x 14 gopal staff 476 Oct 24 09:20 ..
-rw-r--r-- 1 gopal staff 15264 Jul 2 2013 annotations-api.jar
-rw-r--r-- 1 gopal staff 54142 Jul 2 2013 catalina-ant.jar
-rw-r--r-- 1 gopal staff 134215 Jul 2 2013 catalina-ha.jar
-rw-r--r-- 1 gopal staff 257520 Jul 2 2013 catalina-tribes.jar
-rw-r--r-- 1 gopal staff 1581311 Jul 2 2013 catalina.jar
-rw-r--r-- 1 gopal staff 1801636 Jul 2 2013 ecj-4.2.2.jar
-rw-r--r-- 1 gopal staff 46085 Jul 2 2013 el-api.jar
-rw-r--r-- 1 gopal staff 123241 Jul 2 2013 jasper-el.jar
-rw-r--r-- 1 gopal staff 599428 Jul 2 2013 jasper.jar
-rw-r--r-- 1 gopal staff 88690 Jul 2 2013 jsp-api.jar
-rw-r--r-- 1 gopal staff 177598 Jul 2 2013 servlet-api.jar
-rw-r--r-- 1 gopal staff 6873 Jul 2 2013 tomcat-api.jar
-rw-r--r-- 1 gopal staff 796527 Jul 2 2013 tomcat-coyote.jar
-rw-r--r-- 1 gopal staff 235411 Jul 2 2013 tomcat-dbcp.jar
-rw-r--r-- 1 gopal staff 77364 Jul 2 2013 tomcat-i18n-es.jar
-rw-r--r-- 1 gopal staff 48693 Jul 2 2013 tomcat-i18n-fr.jar
-rw-r--r-- 1 gopal staff 51678 Jul 2 2013 tomcat-i18n-ja.jar
-rw-r--r-- 1 gopal staff 124006 Jul 2 2013 tomcat-jdbc.jar
-rw-r--r-- 1 gopal staff 23201 Jul 2 2013 tomcat-util.jar
Pinaca:lib gopal$
```

Cursor

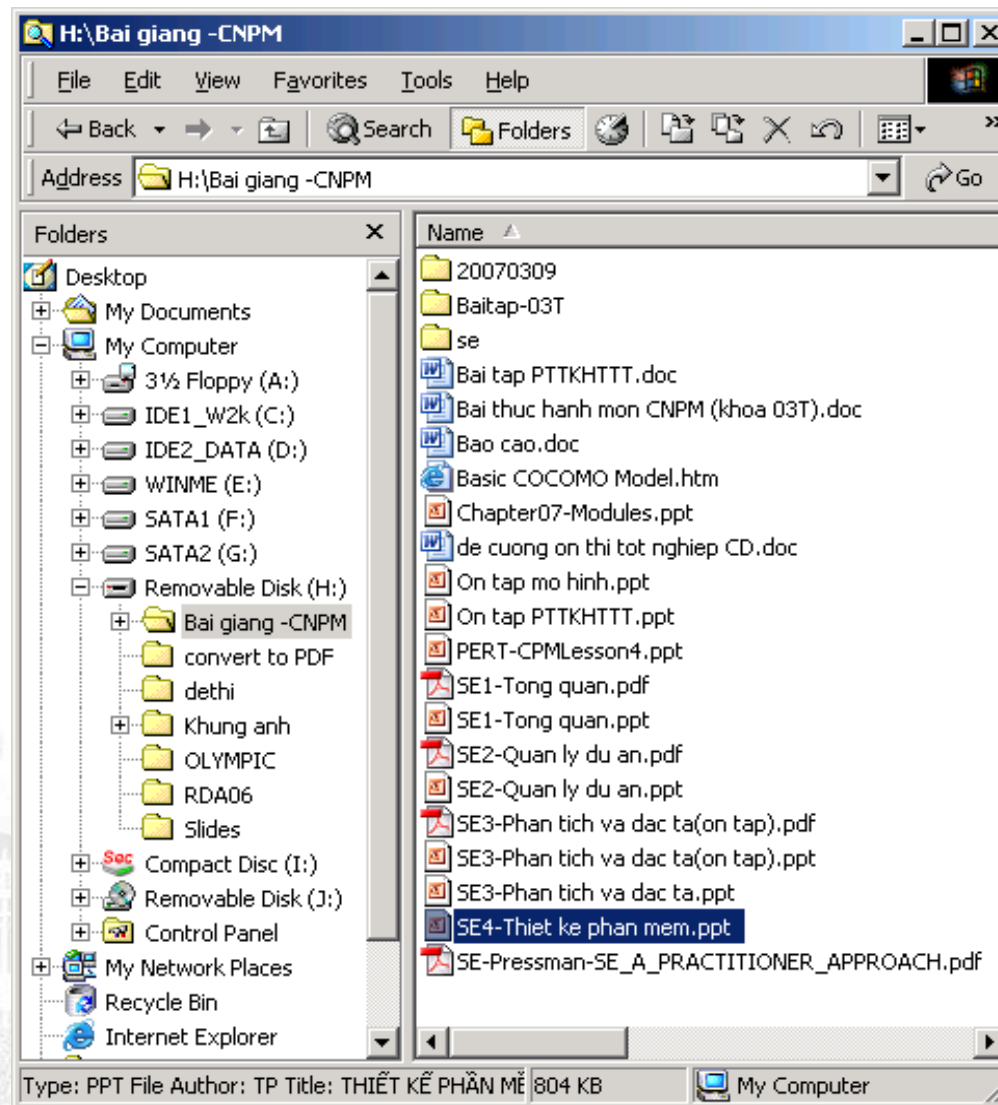
Command Prompt

10.10.10.10 - PuTTY

```
drwxr-xr-x 27 dungdv dungdv 4096 Mar 7 16:24 dungdv
drwx----- 9 hanhdd hanhdd 4096 Mar 2 17:47 hanhdd
drwxr-xr-x 32 hanv hanv 4096 Oct 14 16:49 hanv
drwx----- 23 501 501 4096 Nov 13 16:09 hieulq
drwx----- 29 hungpn hungpn 4096 Mar 21 16:43 hungpn
drwx----- 2 root root 16384 Sep 24 2003 lost+fo
drwx----- 9 tmct tmct 4096 Jan 25 12:27 tmct
drwx----- 3 fosg tuanlt 4096 Oct 11 15:25 tuanlt
hanhdd@selablinux:/home/staff$ ls -l
total 44
drwxr-xr-x 27 dungdv dungdv 4096 Mar 7 16:24 dungdv
drwx----- 9 hanhdd hanhdd 4096 Mar 2 17:47 hanhdd
drwxr-xr-x 32 hanv hanv 4096 Oct 14 16:49 hanv
drwx----- 23 501 501 4096 Nov 13 16:09 hieulq
drwx----- 29 hungpn hungpn 4096 Mar 21 16:43 hungpn
drwx----- 2 root root 16384 Sep 24 2003 lost+fo
drwx----- 9 tmct tmct 4096 Jan 25 12:27 tmct
drwx----- 3 fosg tuanlt 4096 Oct 11 15:25 tuanlt
hanhdd@selablinux:/home/staff$
```

- Giao diện người dùng đồ họa cung cấp các phương tiện đồ họa người dùng để tương tác với hệ thống. GUI có thể là sự kết hợp của cả phần cứng và phần mềm. Sử dụng GUI, người dùng hiểu, sử dụng phần mềm.
- GUI tiêu tốn nhiều tài nguyên hơn CLI.
  - Với công nghệ tiên tiến, các lập trình viên và nhà thiết kế tạo ra các thiết kế GUI phức tạp hoạt động với hiệu quả, độ chính xác và tốc độ cao hơn.
- GUI cung cấp một bộ các thành phần để tương tác với phần mềm hoặc phần cứng.
  - Mỗi thành phần đồ họa cung cấp một cách để làm việc với hệ thống.
  - Dễ học, dễ sử dụng, thuận tiện với người ít kinh nghiệm

# Giao diện đồ họa



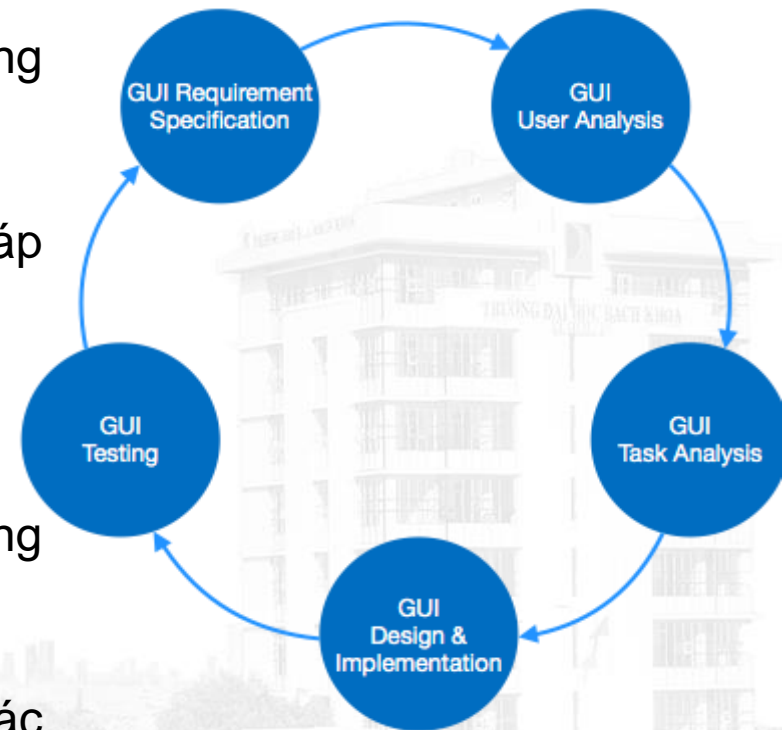
# Các hoạt động thiết kế giao diện (1)

## ■ Thu thập yêu cầu GUI

- danh sách tất cả các yêu cầu chức năng và phi chức năng của GUI.
- có thể lấy từ người dùng và giải pháp phần mềm hiện có của họ.

## ■ Phân tích người dùng

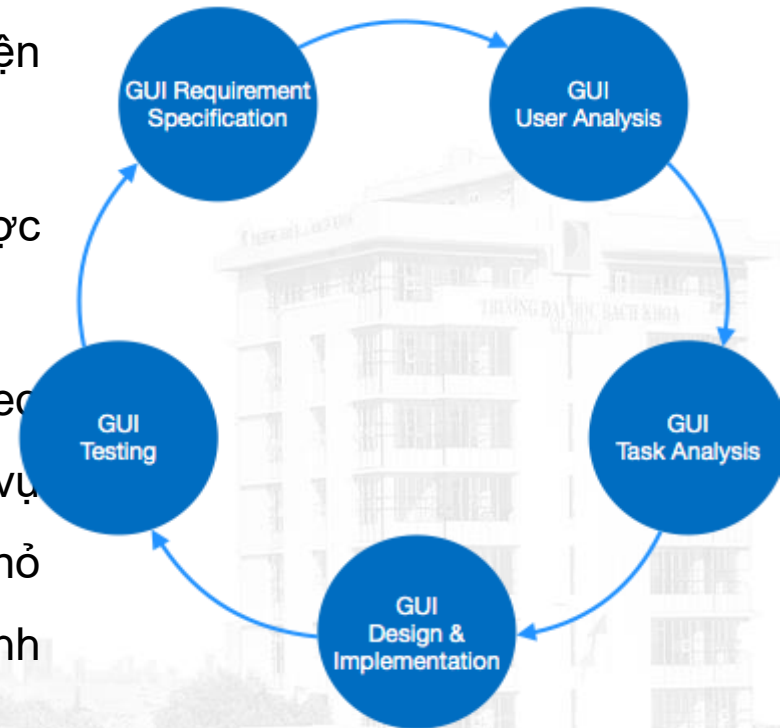
- nghiên cứu những người sẽ sử dụng GUI phần mềm.
- Đối tượng mục tiêu quan trọng khi các chi tiết thiết kế thay đổi theo kiến thức và mức độ năng lực của người dùng.





## ■ Phân tích nhiệm vụ

- phân tích nhiệm vụ nào được thực hiện bằng giải pháp phần mềm.
- trong GUI, không quan trọng nó sẽ được thực hiện như thế nào.
- Các tác vụ có thể được biểu diễn theo cách phân cấp, nghĩa là từ một tác vụ chính và chia nó thành các tác vụ phụ nhỏ hơn. Tác vụ cung cấp các mục tiêu để trình bày GUI.
- Luồng thông tin giữa các tác vụ phụ xác định luồng nội dung GUI trong phần mềm.



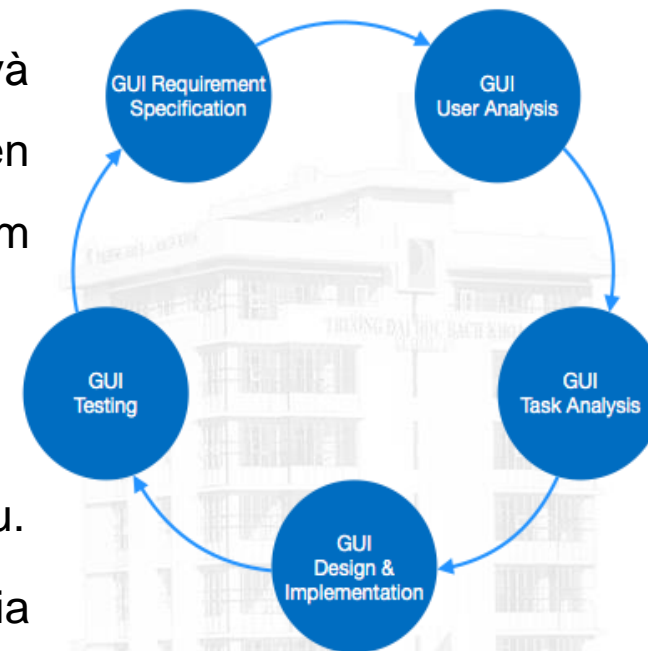


## ■ Thiết kế & triển khai GUI

- sau khi có thông tin về các yêu cầu, nhiệm vụ và môi trường người dùng, thiết kế GUI và thực hiện mã và nhúng GUI với phần mềm giả hoặc làm việc trong nền.

## ■ Kiểm tra

- có thể được thực hiện theo nhiều cách khác nhau.
- Tổ chức có thể có kiểm tra nội bộ, sự tham gia trực tiếp của người dùng và phát hành phiên bản beta là một vài trong số đó.
- Kiểm tra có thể bao gồm khả năng sử dụng, khả năng tương thích, chấp nhận người dùng, vv



- Các quy tắc sau đây được đề cập là các quy tắc vàng cho thiết kế GUI, được mô tả bởi Shneiderman và Plaisant trong cuốn “Thiết kế giao diện người dùng”.
  - **Nhất quán** - Các chuỗi hành động nhất quán nên được yêu cầu trong các tình huống tương tự. Thuật ngữ giống hệt nhau nên được sử dụng trong lời nhắc, menu và màn hình trợ giúp. Sử dụng các lệnh nhất quán.
  - **Sử dụng các lối tắt** - Người dùng mong muốn giảm số lượng tương tác tăng theo tần suất sử dụng. Chữ viết tắt, phím chức năng, lệnh ẩn và tiện ích macro rất hữu ích cho người dùng chuyên gia.
  - **Phản hồi thông tin** - Đối với mỗi hành động thao tác, cần có một số phản hồi hệ thống. Đối với các hành động thường xuyên và nhỏ, phản hồi phải khiêm tốn, trong khi đối với các hành động không thường xuyên và lớn, phản hồi phải thực chất hơn.

- Các quy tắc sau đây được đề cập là các quy tắc vàng cho thiết kế GUI, được mô tả bởi Shneiderman và Plaisant trong cuốn “Thiết kế giao diện người dùng”.
  - **Thiết kế hộp thoại để tăng năng suất** - Phản hồi thông tin khi hoàn thành một nhóm hành động mang lại cho người vận hành sự hài lòng về thành tựu, cảm giác nhẹ nhõm, tín hiệu để loại bỏ các kế hoạch dự phòng.
  - **Xử lý lỗi đơn giản** - Càng nhiều càng tốt, thiết kế hệ thống để người dùng không mắc lỗi nghiêm trọng. Nếu xảy ra lỗi, hệ thống sẽ có thể phát hiện ra nó và đưa ra các cơ chế đơn giản, dễ hiểu để xử lý lỗi.
  - **Cho phép đảo ngược hành động dễ dàng** - Tính năng này làm giảm sự lo lắng, vì người dùng biết rằng các lỗi có thể được hoàn tác. Dễ dàng đảo ngược các hành động khuyến khích khám phá các lựa chọn lạ.
  - **Giảm tải bộ nhớ ngắn hạn** - Giới hạn xử lý thông tin của con người trong bộ nhớ ngắn hạn đòi hỏi màn hình phải đơn giản, nhiều màn hình được hợp nhất, giảm tần số chuyển động của cửa sổ và thời gian đào tạo đủ được phân bổ cho mã, ghi nhớ, và các chuỗi hành động.

