

Trường Đại Học Bách Khoa Tp. Hồ Chí Minh
Khoa Công Nghệ Thông Tin

Môn học

CÔNG NGHỆ PHẦN MỀM



GIỚI THIỆU MÔN HỌC



- **Nhắc nhở**

- ◆ Mã số 501095

- Soát tín chỉ: 2

- Phân phối giờ 2(2.1.4)

- ◆ Môn học trước: Toán Tin Học (501302)

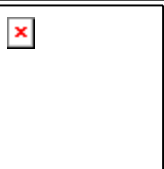
- **Tóm tắt nội dung**

- ◆ Các khái niệm cơ bản của Công Nghệ Phần Mềm: các mô hình phát triển phần mềm, phân tích yêu cầu, thiết kế kiểm tra...

- ◆ 2 trường phái chính: cấu trúc (cấp trên) & hướng đối tượng

- ◆ Chuẩn UML và việc áp dụng nó trong phương pháp hướng đối tượng

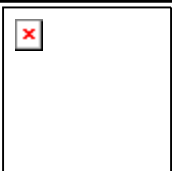
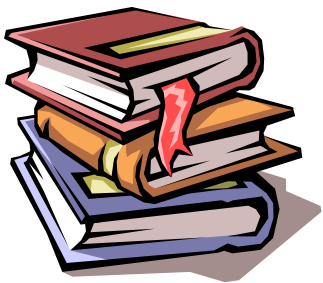
- ◆ Thực hành tại phòng Lab: làm quen với công cụ Rational Rose



GIỚI THIỆU MÔN HỌC (t.t)

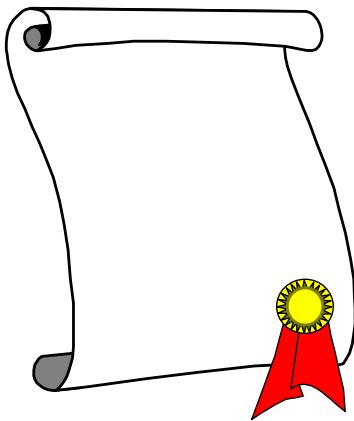
- Tài liệu tham khảo

- ◆ [1] Software Engineering - A practitioner's approach, R.S. Pressman, McGraw-Hill, 1997
- ◆ [2] OMG Unified Modeling Language Specification, version 1.3, Object Management Group (www.omg.org), 1999
- ◆ [3] UML Toolkit, Hans-Erik Eriksson & Magnus Penker, 1998
- ◆ [4] Object-Oriented Software Engineering, A Use-Case Driven Approach, I. Jacobson, ACM Press/Addison-Wesley, 1992
- ◆ [5] Object-Oriented Analysis and Design with Applications, G. Booch, The Benjamin Cummings Publishing Company, 1994



GIỚI THIỆU MÔN HỌC (t.t)

Hình thức đánh giá



- Thi giữa kỳ trước nghiệm không sâu đúng tài liệu, chiếm 20 % kết quả cuối cùng
- Thi cuối kỳ trước nghiệm không sâu đúng tài liệu, chiếm 80 % kết quả cuối cùng

NỀN CÖÔNG

Chöông 1: Giöõ thiäu veà Coâng Ngheä Phaàn Meäm

Chöông 2: Phaân tích yêu cầu theo phöông pháp coảñieän

Chöông 3: Các khai niệm cô bản của mô hình höông ñoä tööng

Chöông 4: Mô hình nghiệp vụ và thu thập yêu cầu

Chöông 5: Phaân tích yêu cầu höông ñoä tööng

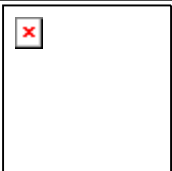
Chöông 6: Cô sô của thiết kế phaàn meäm và phöông pháp thiết kế coảñieän

Chöông 7: Thiết kế höông ñoä tööng

Chöông 8: Hiệän thöc và triệän khai hệ thống

Chöông 9: Kyö thuaät kiểm tra phaàn meäm

Chöông 10: Chiếän thuaät kiểm tra phaàn meäm



Chương 1

GIỚI THIỆU VỀ CÔNG NGHỆ PHẦN MỀM

❖ Một số khái niệm

❖ Các mô hình phát triển phần mềm



NOI DUNG

1.1. Một số khái niệm

1.1.1. Khuôn hoàng phần mềm

1.1.2. Nền nhóa

1.1.3. Chu trình (*process*), phương pháp (*method*), công cụ (*tool*)

1.1.4. Một cách nhìn tổng quan về công nghệ phần mềm

1.1.5. Mô hình CMM

1.2. Các mô hình phát triển phần mềm

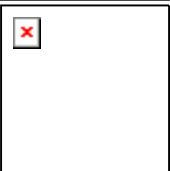
1.2.1. Mô hình tuần tời tuyến tính

1.2.2. Mô hình *prototype*

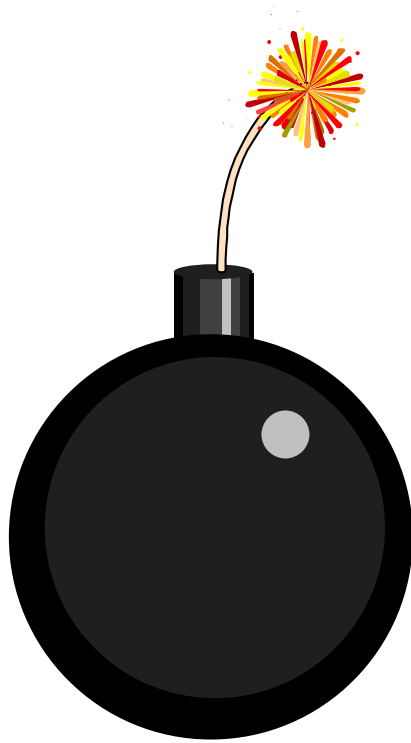
1.2.3. Mô hình xoắn ốc

1.2.4. Mô hình tầng dần

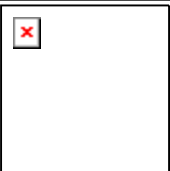
1.2.3. Mô hình RAD



KHUNG HOANG PHAN MEM



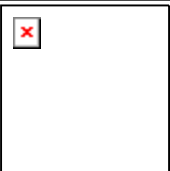
- Phần mềm được viết ngay thời điểm xuất hiện các hệ máy tính và ngôn ngữ lập trình đầu tiên
- Trên thực tế sản xuất phần mềm không đáp ứng kịp yêu cầu của người sử dụng



KHUNG HOANG PHAN MEM (t.t)

Các dữ liệu quan sát nổi

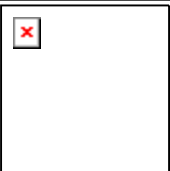
- ◆ Có 6 nền tảng triển khai thì có 2 bỏ hủy
- ◆ Trung bình thời gian thực hiện thực tế là 50 % (còn lại 200-300%)
- ◆ Các nền tảng lớn đã thất bại
- ◆ 3/4 các hệ thống lớn có lỗi khi thực thi
- ◆ Quá trình phân tích yêu cầu (5 % công sức): nền tảng 55 % lỗi, còn 18 % phát hiện nổi
- ◆ Quá trình thiết kế (25 % công sức): nền tảng 30 % lỗi, còn 10 % phát hiện nổi
- ◆ Quá trình mã hóa kiểm tra và bảo trì: nền tảng 15 % lỗi, còn 72 % phát hiện nổi



KHUNG HOANG PHAN MEM (t.t)

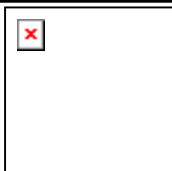
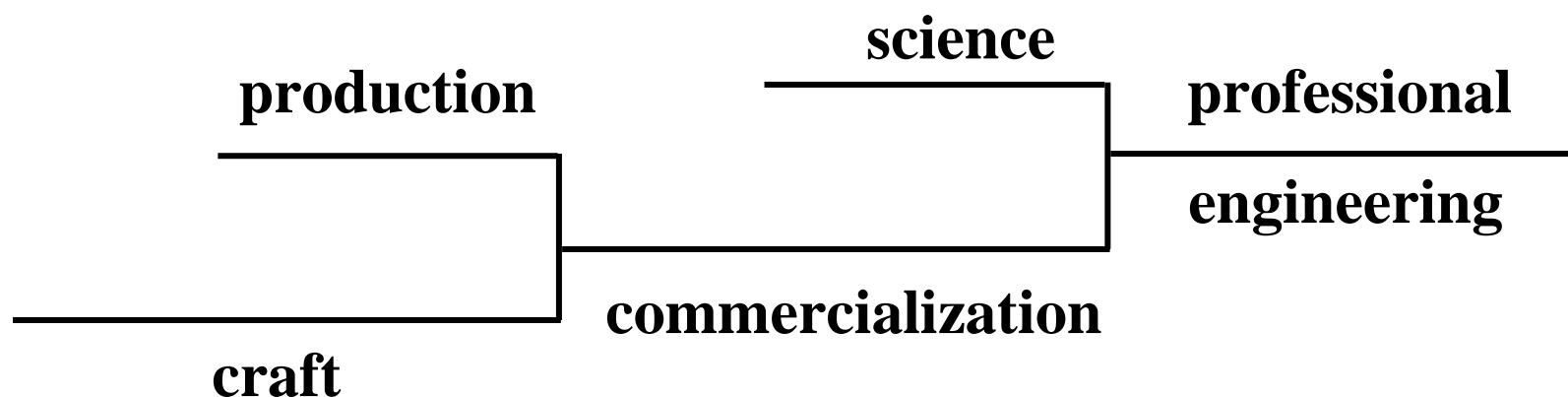
Nguyên nhân

- ◆ Phát triển phần mềm giống như một nghệ thuật, chờ đợi xem như một ngành khoa học
- ◆ Quá trình phát triển phần mềm chờ đợi thống nhất
- ◆ Phải viết lại s/w mỗi khi có sự thay đổi về ngôn ngữ h/w hoặc o/s
- ◆ Chờ đợi một chuẩn cho việc nỗ lực hiệu suất và sản phẩm
- ◆ Nỗ lực tập của phần mềm quá cao nối với 1 “kiến trúc sư”
- ◆ Kỹ thuật cần tái sử dụng lại trong các yêu cầu phần mềm
- ◆ Làm việc nhóm không tuân thủ luật gây ra các lỗi



KHUÔNG HOÀNG PHẦN MỀM (t.t)

Hồ sơ công nghệ sản xuất phần mềm chuyên nghiệp



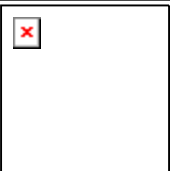
NÒNGH NGHÓA

Nòngh nghóa coảñieñ (cuả Fritz Bauer)

Công Nghệ Phần Mềm là sỡi thiết lập và sỡi dùng các nguyên tắc

khoa hoặi nhằm mục ñích tạo ra các phần mềm một cách kinh tế mà

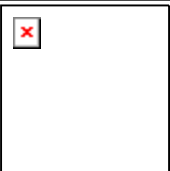
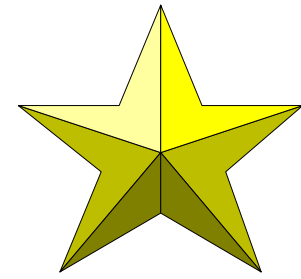
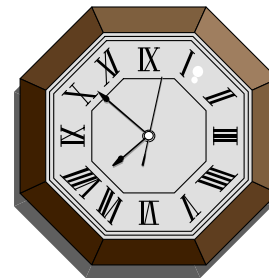
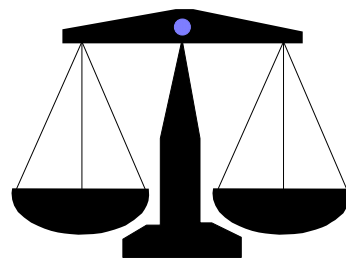
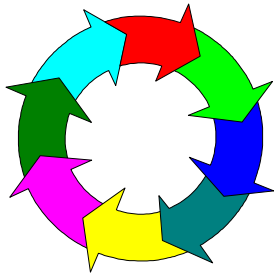
các phần mềm ñó hoạt ñộng hiệu quả và tin cậy trên các máy tính.



NH NGHÓA (t.t)

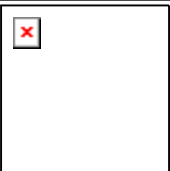
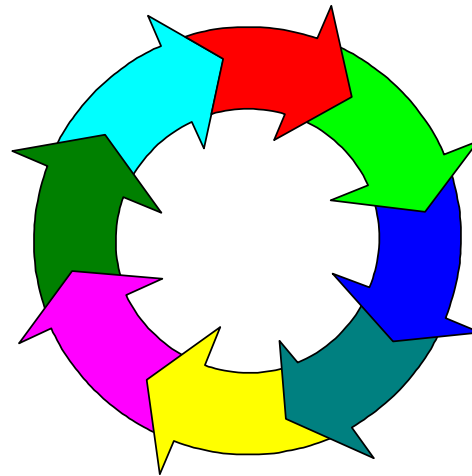
Nh nghĩa khác: Công Nghệ Phần Mềm

- Là các quy trình ứng dụng và công nghệ để hỗ trợ việc xây dựng cho sự phát triển, thực thi và bảo trì các hệ thống thông tin và phần mềm
- Tập trung vào quy trình, sự nỗ lực, sản phẩm, tính ứng dụng thời gian và chất lượng



CHU TRÌNH

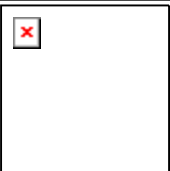
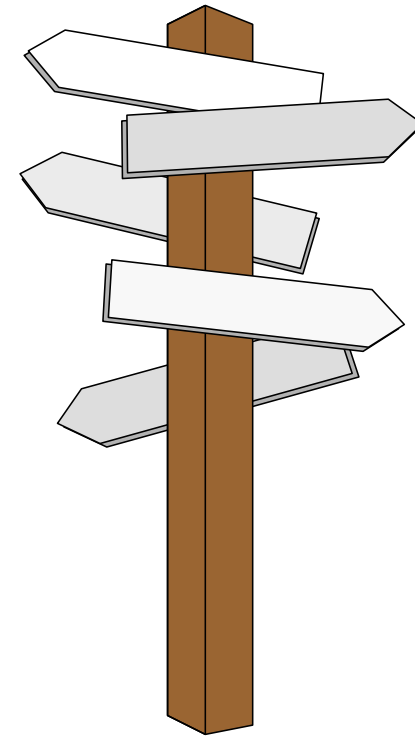
Chu trình (*process*) là hình thức mô tả bao gồm các tiêu chuẩn phải được thiết lập để triển khai công nghệ phần mềm.



PHÖÔNG PHÁP

Phöông pháp (*method*) chæ ra cách thöc hiên nhöng công viêc cụ thể ("*how to*"):

- ◆ phân tích yêu cầu
- ◆ thiết kế
- ◆ xây dựng chương trình
- ◆ kiểm tra
- ◆ sửa lỗi
- ◆ ...



CÔNG CỤ

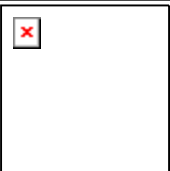


- Công cụ (*tool*) cung cấp các hỗ trợ tối ưu hay bản tối ưu hoá với chu trình và phương pháp
- Các công cụ được tích hợp tạo thành CASE (*Computer Aided Software Engineering*)

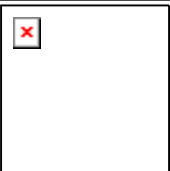
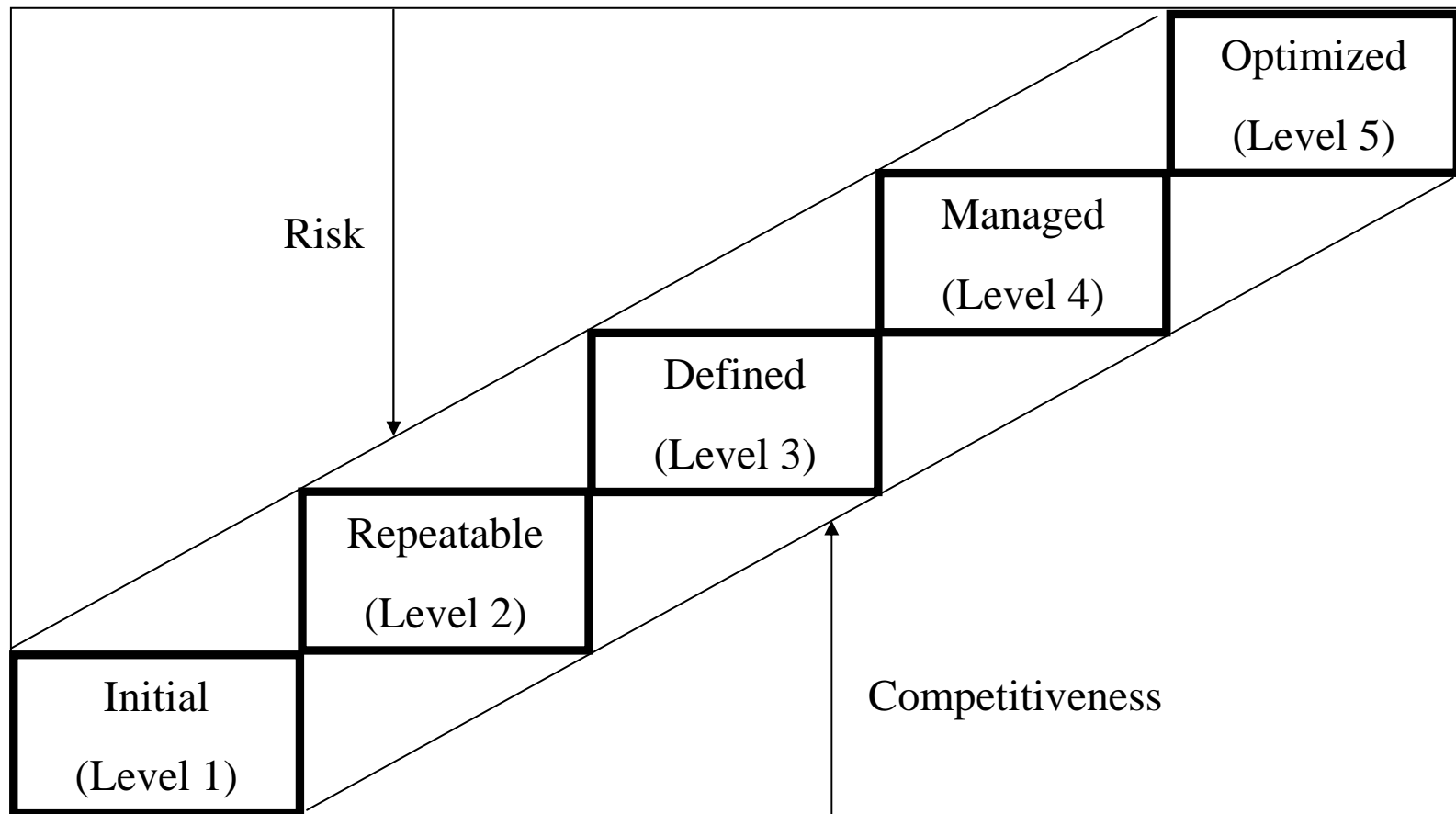
MO' CACH NHIN TONG QUAN VE CNPM

Gom 3 giai n'oa'n lon

- ◆ Giai n'oa'n n'ong ngh'oa: Phan t'ich he' th'ong (*system engineering*), Hoach n'ong n'e' tai (*software project management*), Phan t'ich ye'u cau (*requirement analysis*).
- ◆ Giai n'oa'n phat trien: Thiet ke' phan mem (*software design*), sinh ma' (*code generation*), kiem tra phan mem (*software testing*)
- ◆ Giai n'oa'n bao tri: So'a loi (*correction*), thay noi moi tro'ng th'oc thi (*adaptation*), tang co'ng (*enhancement*)

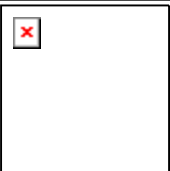


MO HÌNH CMM



CÁC MÔ HÌNH PHÁT TRIỂN PHẦN MỀM

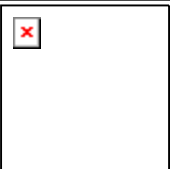
- Mô hình tuần tự tuyến tính: cổ điển
- Mô hình *prototyping*: *prototype*
- Mô hình xoắn ốc: hình giải ro
- Mô hình tăng dần: các bước lặp
- Mô hình RAD: thời gian phát triển ngắn



MO HÌNH TUẦN TỐI TUYẾN TÍNH



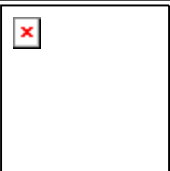
- Mô hình phát triển phần mềm tuần tự
- Các công việc tiếp nối nhau một cách tuần tự
- Nền tảng cho các phương pháp phân tích, thiết kế kiểm tra...



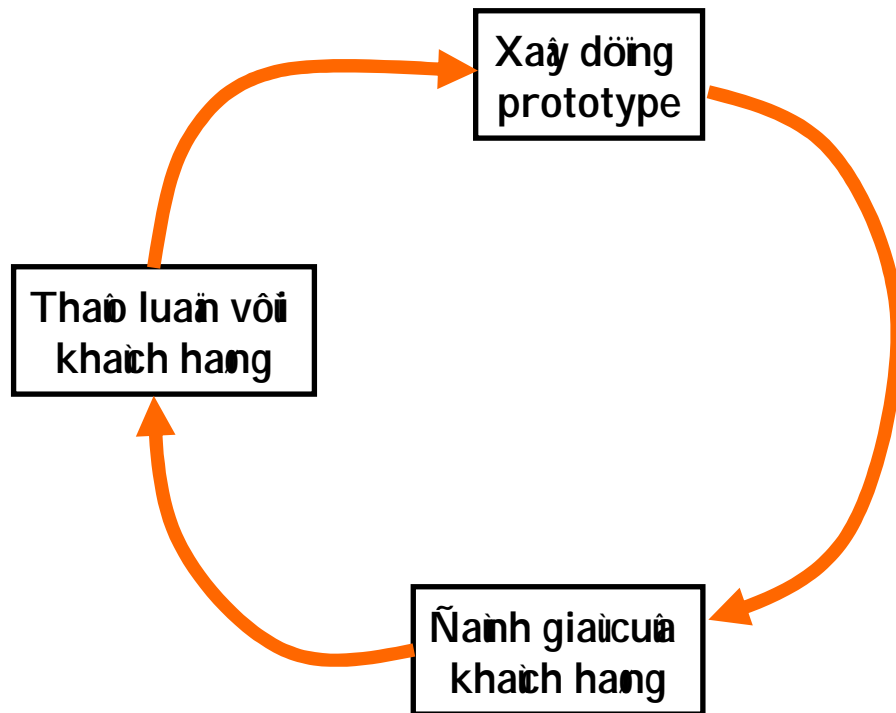
MO HÌNH TUẦN TỐI TUYẾN TÍNH (t.t)

Bước loà một số khuyết điểm

- ◆ Bản chất của phát triển phần mềm là quá trình lặp đi lặp lại cho đến khi không phải tuần tởi
- ◆ Bắt bước khách hàng nêu tất cả yêu cầu một cách chính xác và này nui ngay từ ban đầu
- ◆ Khách hàng thông phải chờ đợi rất lâu để thấy được phiên bản này tiên của sản phẩm
- ◆ Tồn tại "delay" trong nhóm làm việc

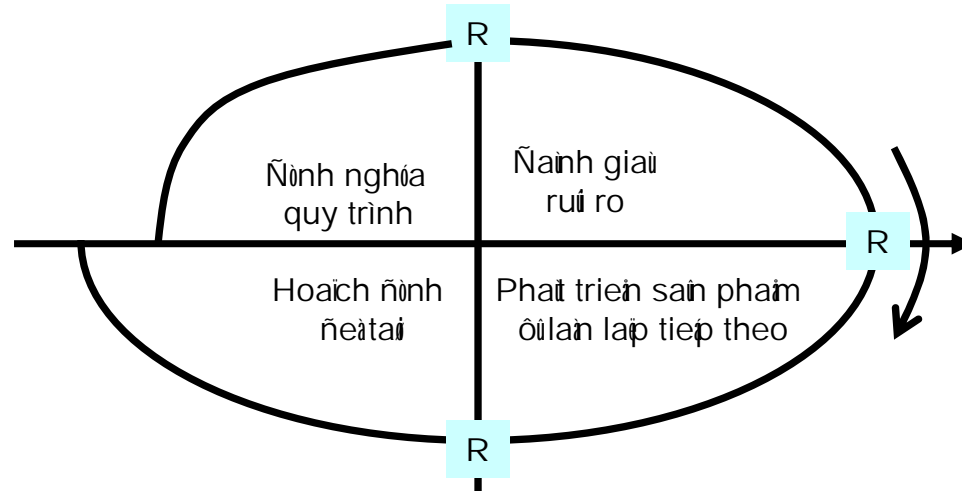


MO HÌNH *PROTOTYPING*

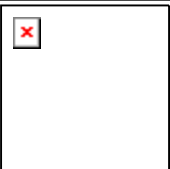


- *Prototype* nhỏ là một công cụ để thể hiện chính xác yêu cầu của khách hàng
- *Prototype* có thể bỏ "throw-away"
- Một số khuyết điểm
 - ◆ Khách hàng có thể thích nghi và phát triển hoàn thành sản phẩm một khi thấy những cái *prototype* này
 - ◆ Các *prototype* thông thường không hoạt động hiệu quả

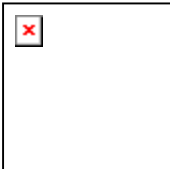
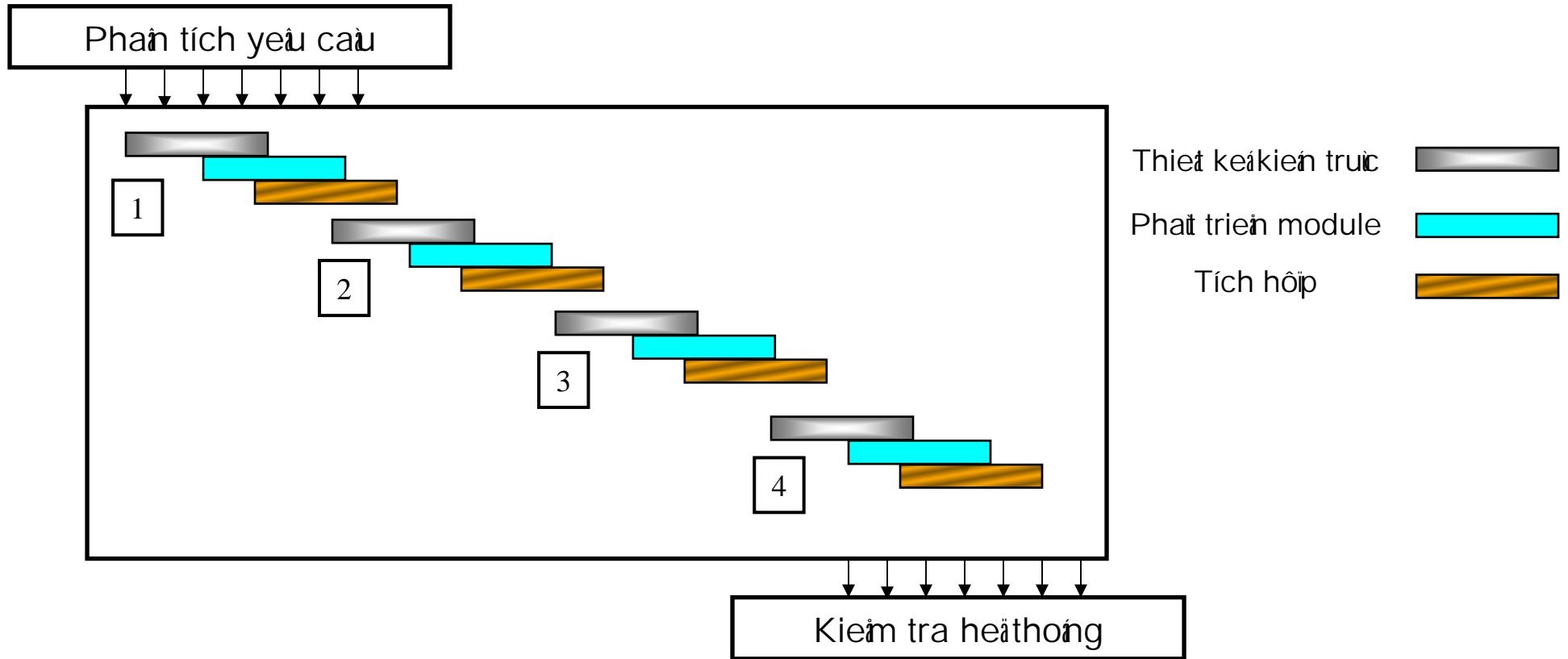
MO HÌNH XOÁN ỐC



- Nöôc thöc hiên theo môt chuôi lap kieu xoán ốc, mõi lan lap cái thiên san phẩm
- Còiphöông pháp ñành giai rui ro
- Còithểáp dụng *prototype*
- Mõi lan lap nööc cái thiên cho thích nghi với bản chất của ñềài

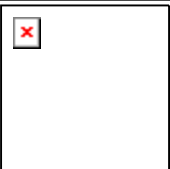


MO HÌNH TẦNG DÀN

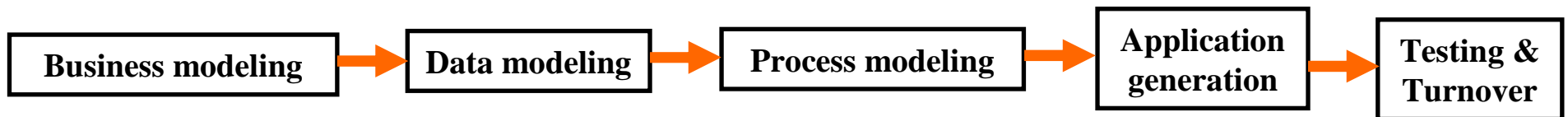


MO HÌNH TẦNG DÀN (t.t)

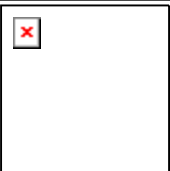
- Các bước (*iteration*) này tập trung vào yêu cầu của phần mềm và thiết lập một kiến trúc ổn định cho hệ thống (ít phải thay đổi sau này)
- Các bước sau tập trung vào việc xây dựng sản phẩm để cuối cùng chuyển sang giai đoạn kiểm tra hệ thống
- Mỗi bước hiện thực một phần cuối trong toàn bộ yêu cầu của hệ thống
- Quá trình xây dựng và kiểm tra thuật kiểm tra theo kiểu tầng dần và dựa trên phương pháp kiểm tra hồi quy.



MO HÌNH RAD



- Rapid Application Development là mô hình tuần hoàn tối ưu tính công nghệ gian phát triển rất ngắn
- Số lượng các thành phần có sẵn càng nhiều càng tốt
- Số lượng công cụ lập trình càng tối ưu càng sinh lợi cho người không phải các ngôn ngữ truyền thống



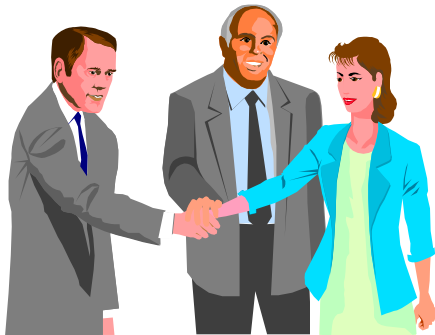
Chương 2

PHÂN TÍCH YÊU CẦU THEO PHƯƠNG PHÁP COẢNỈEỦ

- ❖ Môhình phân tích
- ❖ DFD & STD
- ❖ Tồniễn döliệu

GIỚI THIỆU

- Khách hàng và nhà phát triển gặp nhau để thảo luận về yêu cầu của hệ thống phần mềm cần xây dựng
- Nhà phát triển kiểm chứng lại (*validate*) yêu cầu và biểu diễn nó bằng mô hình phân tích
- Mô hình phân tích (WHAT?): các chức năng, dữ liệu *input & output*, các trạng thái khác nhau...



NOI DUNG

2.1. Các yếu tố cấu thành của mô hình phân tích

2.2. Mô hình chức năng và dòng thông tin

2.2.1. Lựa chọn dòng chảy dữ liệu với các ký hiệu cơ bản

2.2.2. Mô hình của Ward và Mellor

2.2.3. Mô hình của Hatley & Pirbhai

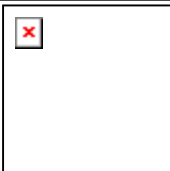
2.3. Mô hình hành vi phần mềm

2.4. Kỹ thuật phân tích yêu cầu

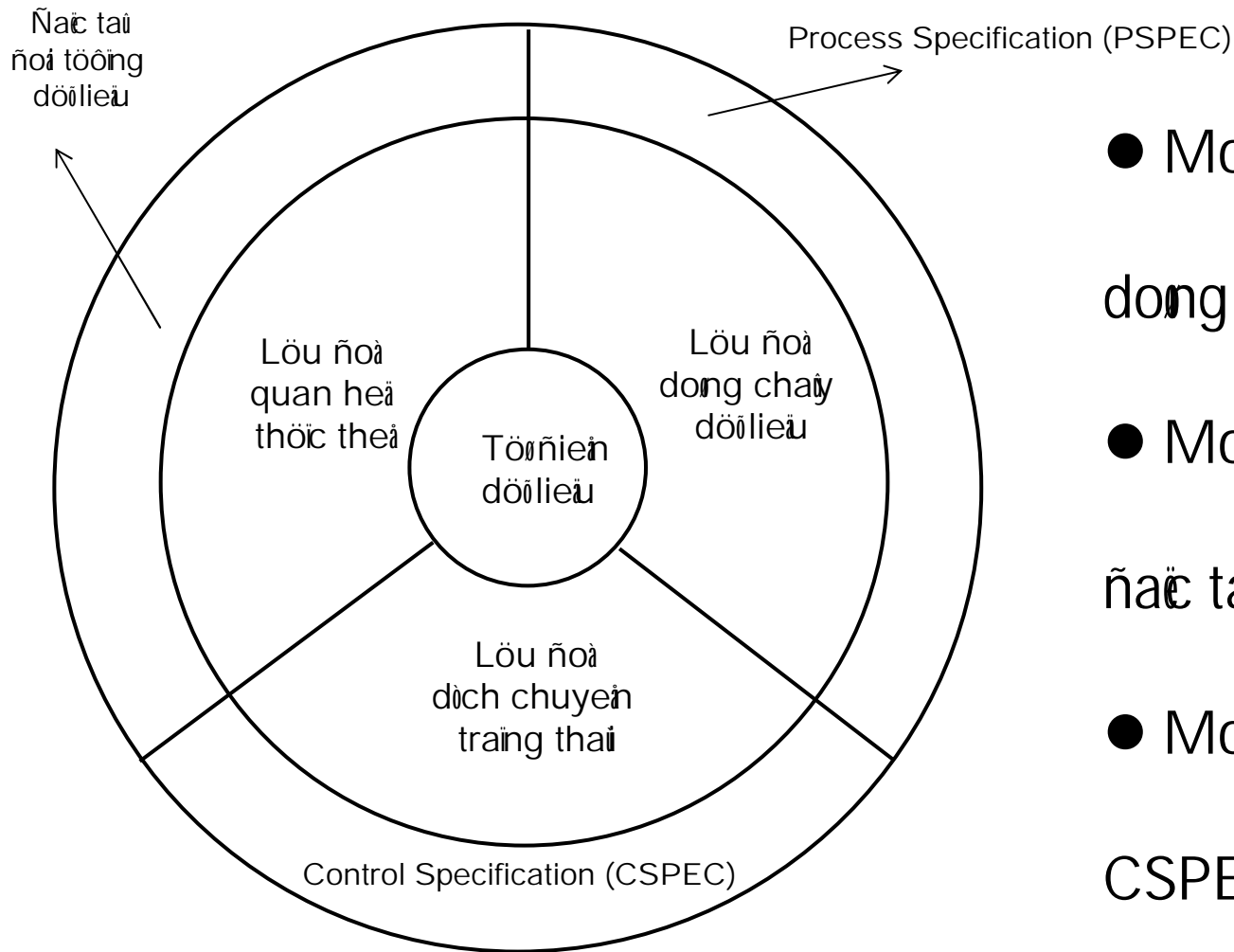
2.4.1. Xây dựng DFD

2.4.2. Viết PSPEC

2.5. Tổ chức dữ liệu



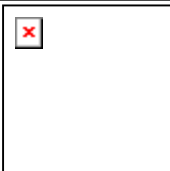
CÁC YẾU TỐ CƠ BẢN CỦA MÔ HÌNH PHÂN TÍCH



- Mô hình chức năng và dòng thông tin: DFD, PSPEC
- Mô hình dữ liệu: ERD, ñăng tài ñoà tổng dữ liệu
- Mô hình hành vi: STD, CSPEC

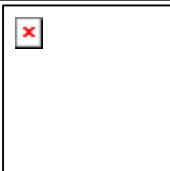
MO HÌNH CHỜC NÀNG VÀ DONG THÔNG TIN

- Mô tả dòng thông tin di chuyển (*flow*) xuyên qua các hệ thống thiên về phần mềm.
- Thông tin *input* cũng như *output* có thể ở nhiều dạng khác nhau: file, bàn phím, trên mạng, ổ thiết bị, kết xuất ra màn hình và máy in...
- Các giải thuật xử lý cũng rất đa dạng



MO HÌNH CHỒI NÀNG VÀ DONG THÔNG TIN (t.t)

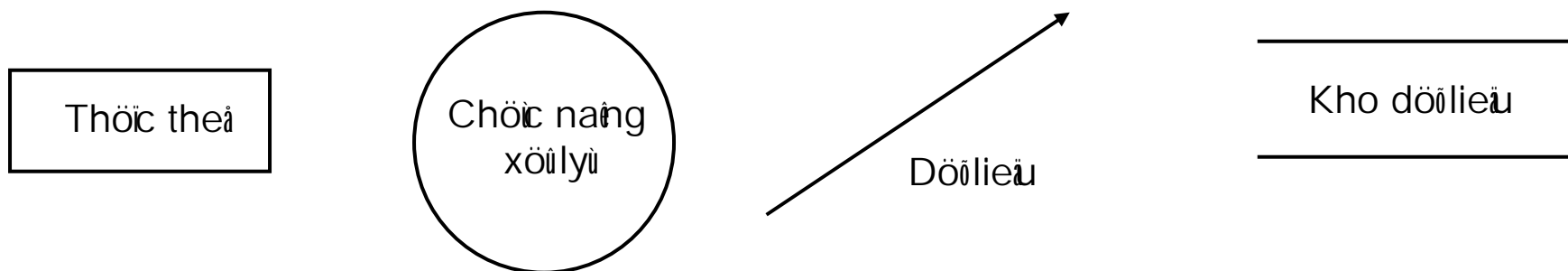
- Lồu ñoàdong chày dồlieu DFD (*Data Flow Diagram*) cung cấp 4 kyùhieù cô baù ñeảmoảhình sồ di chuyền của dong thông tin
- DFD ñồôc môừơng ñeảmoảhình các hệthống thời gian thồc
 - ◆ Môừơng của Ward vàMellor (tồ ñồc: [1], trang 312)
 - ◆ Môừơng của Hatley & Pirbhai (tồ ñồc: [1], trang 315)



LÖÖC ÑOÀDONG CHAY DÖÖLIEÜ (DFD)

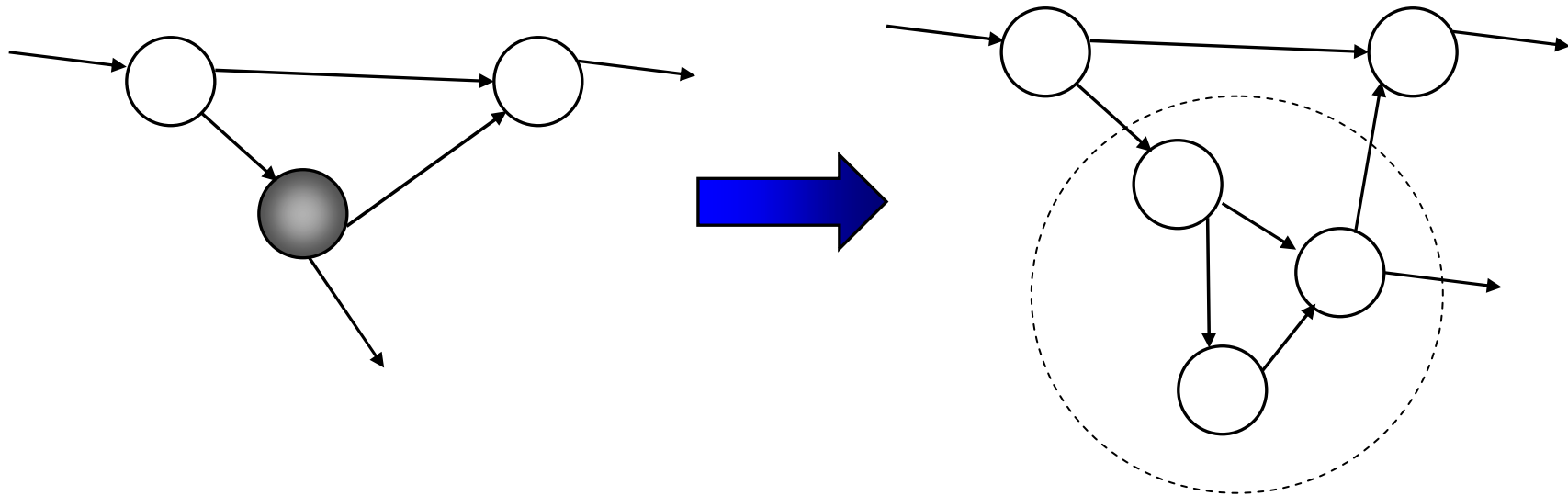
- 4 phàn töüchính

- ◆ Thöc theä tao ra hoacé tieü thui thông tin, nằm bên ngoài biên giới của phạm vi thông tin hệthống
- ◆ Chöc năng xöülyü thöc hiệñ chöc năng nào ñoù tieü thui vàtao ra thông tin, nằm bên trong phạm vi thông tin hệthống
- ◆ Thông tin hay döölíeü
- ◆ Kho döölíeü: lóu tröö döölíeü mà ñöök söüduñg böü nhiều chöc năng xöülyü

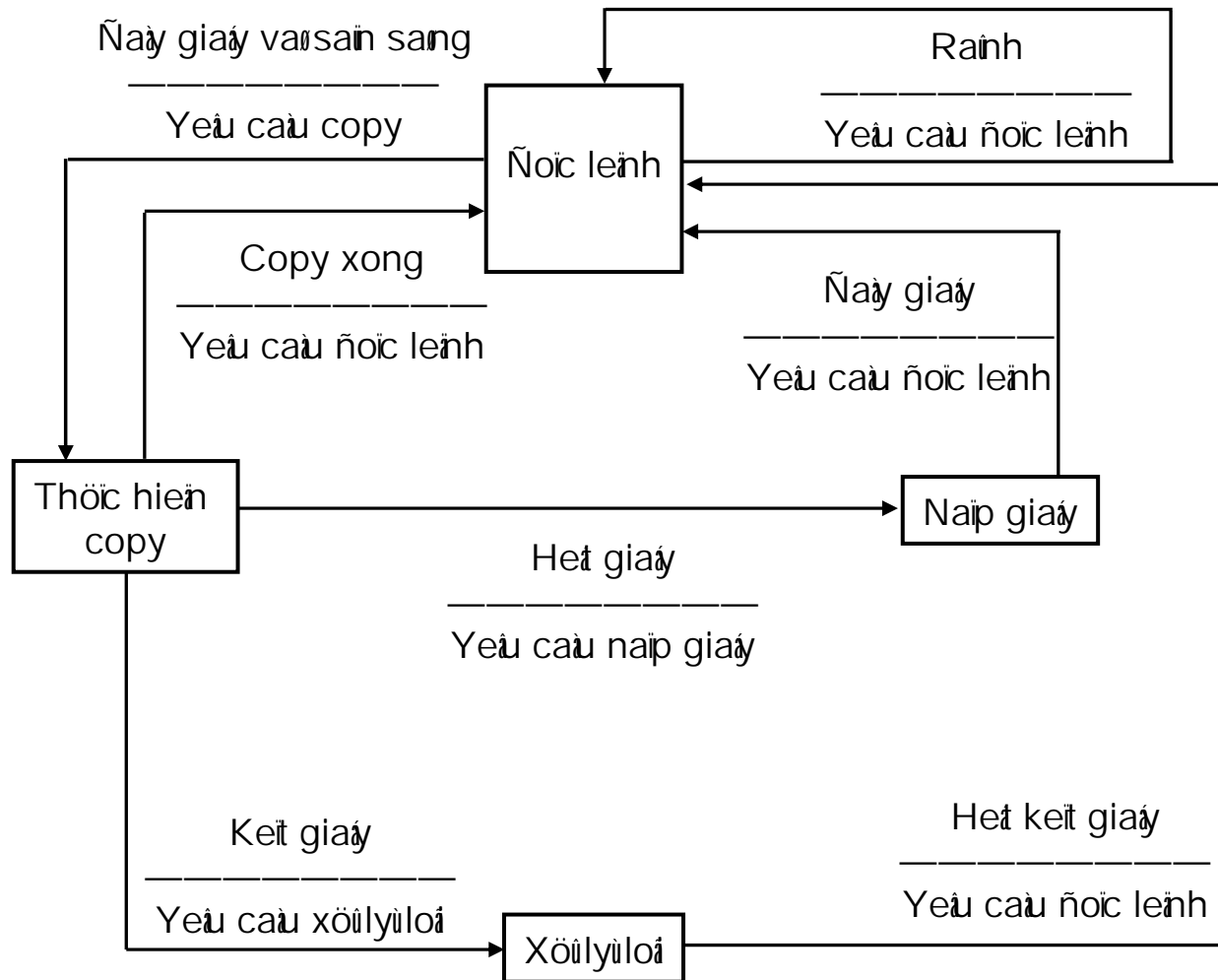


LÖÖIC ÑOÀDONG CHAY DÖÖLIEÜ (t.t)

- DFD ñöôc xay döng qua nhieu möc khac nhau: möc 0, 1, 2...
- DFD möc sau chi tiet hôn möc tröôc
- Process Specification (PSPEC) bo sung cho DFD
- Tính liên tục của dòng dữ liệu



MO HÌNH HÀNH VI PHẦN MỀM



● Lööc ñoà dồch chuyể
trăng thẩ (STD) the hie

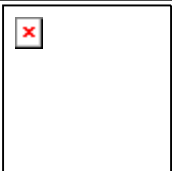
◆ Các trăng thẩ khac
nhau của hệ thố

◆ Söi dòch chuyể giöa
các trăng thẩ ñö

● Ví dụ: miêu tả hoạt
ñöng của máy photocopy

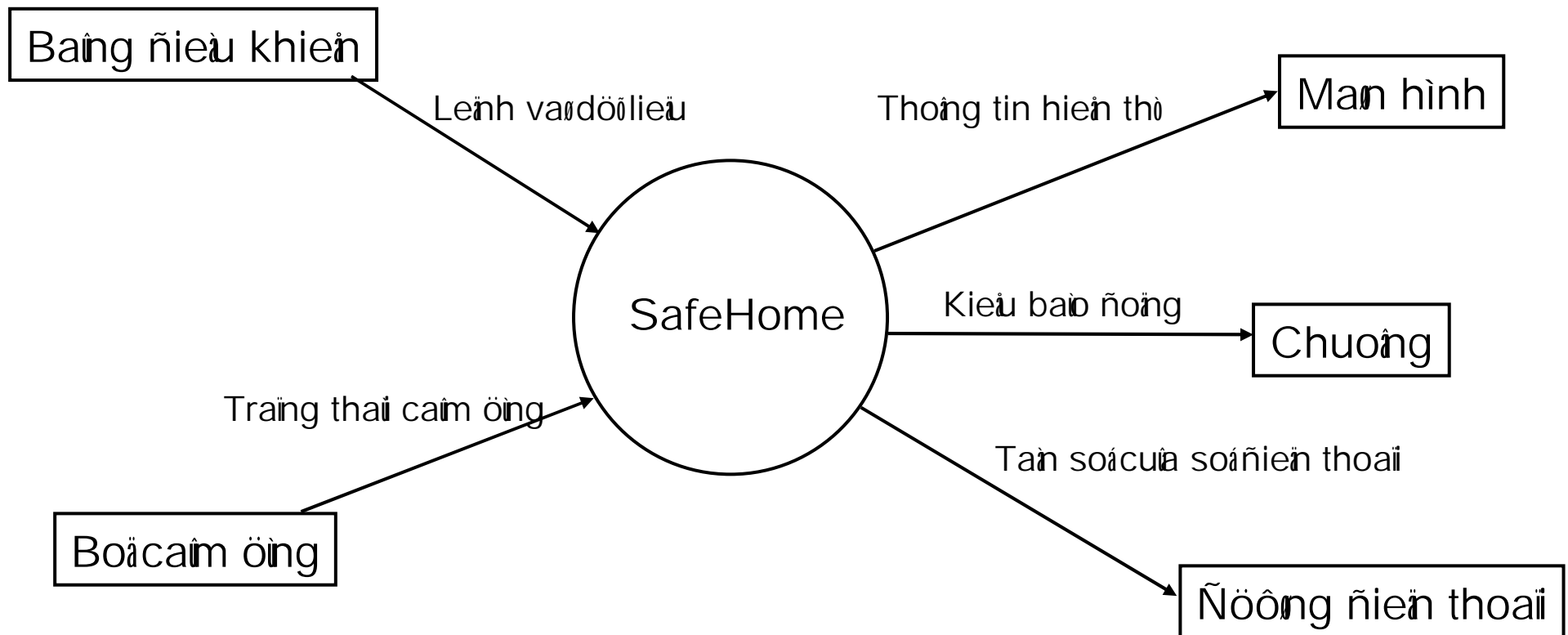
KỸ THUẬT PHÂN TÍCH YÊU CẦU

- Thiết lập nền văn miêu tả chức năng (*processing narrative*) cho hệ thống cần xây dựng
- Xây dựng DFD ở các mức khác nhau
 - ◆ Thiết lập sơ đồ ngữ cảnh (DFD mức 0)
 - ◆ Phân hoạch DFD vào các mức cao hơn
 - ◆ Sử dụng phương pháp duyệt và phân.
 - ◆ Luôn luôn tuân theo tính liên tục của dòng dữ liệu
- Viết PSPEC cho các chức năng của DFD mức cao nhất

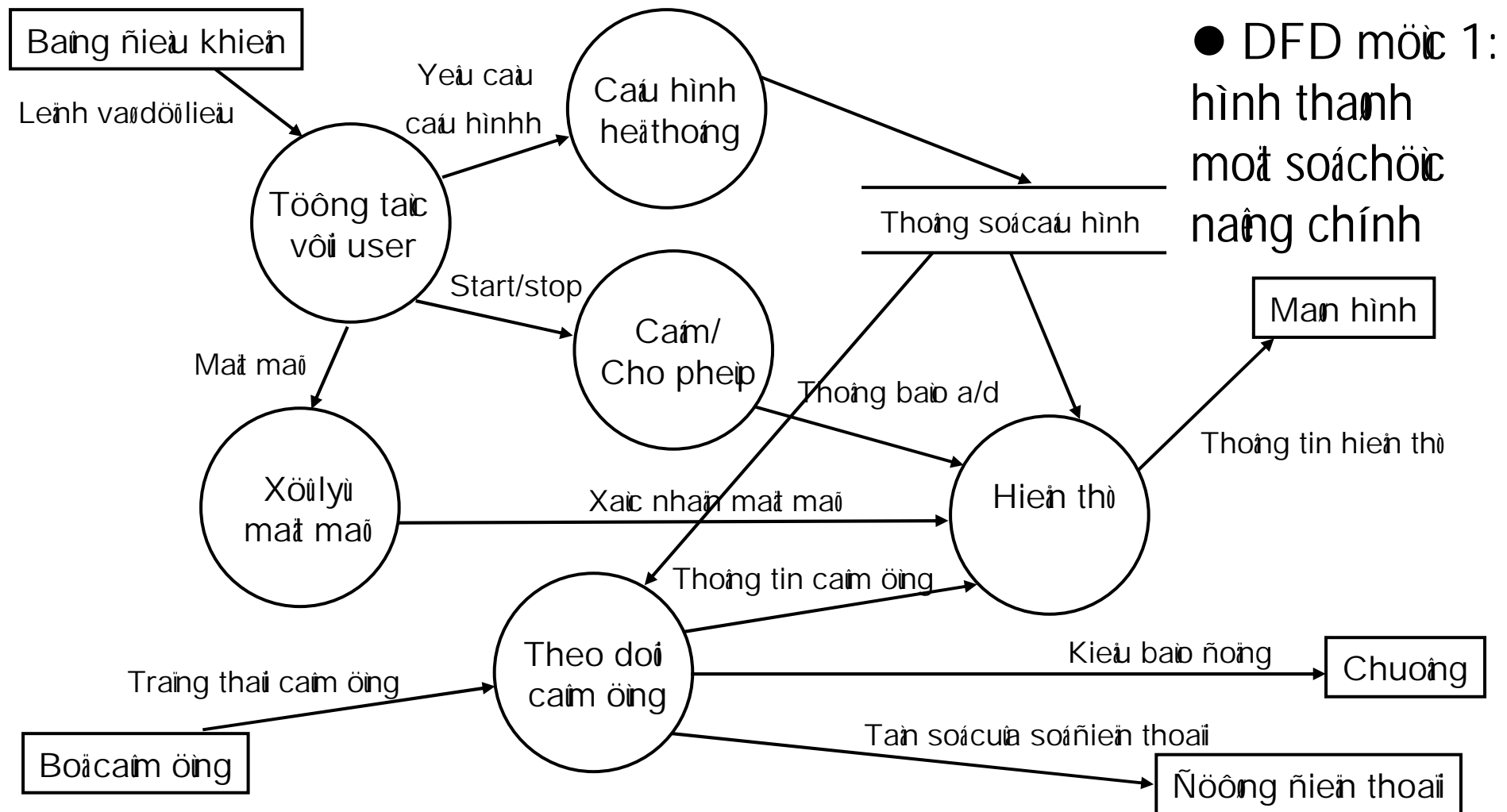


XÂY DỰNG DFD

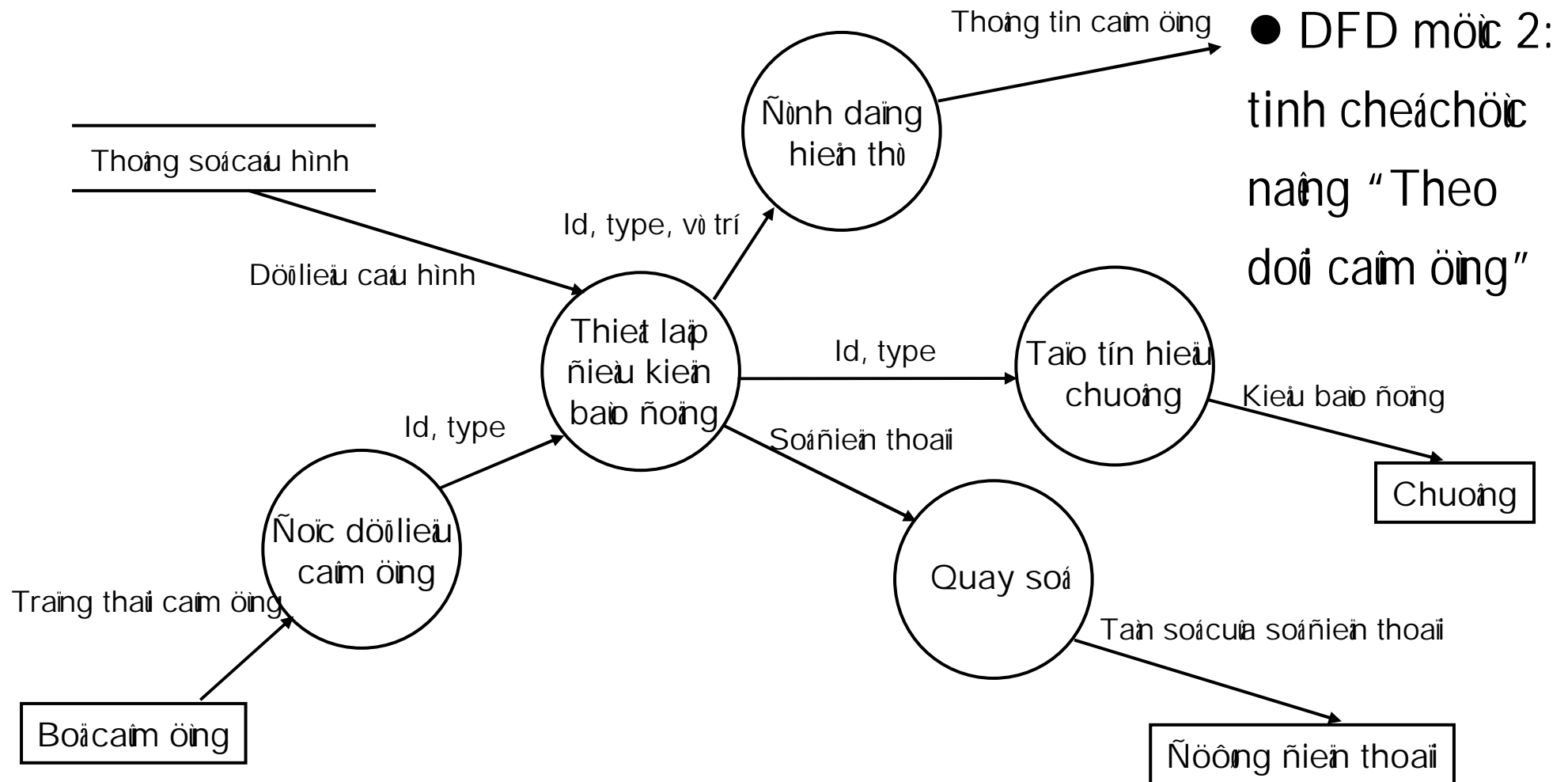
- Phần mềm SafeHome: Thiết lập hoàn văn miêu tả xử lý
- DFD mô tả ngữ cảnh: nhận diện các thực thể và dữ liệu *input, output*



XÂY DỰNG DFD (t.t)



XÂY DỰNG DFD (t.t)



VIẾT PSPEC

- Có thể viết PSPEC bằng một trong 2 cách
 - ◆ Ngôn ngữ tự nhiên (tổng từ *processing narrative*)
 - ◆ Ngôn ngữ PDL - là ngôn ngữ giúp thể hiện kiến trúc và giao tiếp của chức năng xử lý

TỔNG TIỀN DỮ LIỆU

- Nhiều phần tử được tạo ra trong mô hình phân tích: dữ liệu, chức năng, kiến trúc...
- Phải có một cách thức quản lý các phần tử này sao cho hiệu quả tổng tiền dữ liệu
- Nhìn nghĩa:
Tổng tiền dữ liệu là một danh sách có thứ tự của tất cả các phần tử dữ liệu cần thiết cho hệ thống. Các phần tử được nhìn nghĩa chính xác và chất lượng cho các phân tích việc và khách hàng cũng chia sẻ một suy nghĩ về chúng.
- Tổng tiền dữ liệu thông thường hiện thức nhờ là một phần của công cụ CASE.
- Mỗi phần tử bao gồm những thông tin: tên, bí danh, nội dung ô tô màu/nhà thiết kế, các thuộc tính và thông tin phụ trợ

TỔNG TIỀN DỮ LIỆU (t.t)

Ví dụ phân tích dữ liệu số tiền thoại

- ◆ Tên: Số tiền thoại
- ◆ Bí danh: Không
- ◆ Nội dung ô đầu/nhờ theo: *output của Thiết lập nhiều kiện báo nợ*
input của Quay số
- ◆ Các tài liệu dung:
 - số tiền thoại = [môi trường nhà phòng | số bên ngoài]
 - môi trường nhà phòng = [2001 | 2002 ... | 2009]
 - số bên ngoài = 9 + [số nhà phòng | số tổng dài]
 - số nhà phòng = tiền toán <chuỗi 4 ký số>
 - số tổng dài = (1) + mã vùng + số nhà phòng
 - tiền toán = [795 | 799 | 874 | 877]

TỔNG KẾT

- Phân tích yêu cầu theo pp công nghệ bao gồm: mô hình chức năng và dòng thông tin (DFD), mô hình dữ liệu (ERD) và mô hình hành vi (STD)
- Lược đồ DFD cơ bản có 4 ký hiệu và nó mô tả rõ ràng về biểu diễn chức năng hệ thống thời gian thực
- Xây dựng DFD mức 0 rồi đến các mức cao hơn; chuỗi các toán tính liên tục của dòng dữ liệu
- Tạo nên dữ liệu giúp quản lý và tra cứu các phần tử dữ liệu



Chương 3

CÁC KHÁI NIỆM CƠ BẢN CỦA MÔ HÌNH HỒNG NỘI TỔNG

- ❖ Lớp vỏ nội tổng, sợi hồng bao ☺ ☺
- ❖ Thuộc tính, tài vui, thông điệp
- ❖ Bao gói, thỏa ước
- ❖ Tính nửa hình, tính vĩnh cửu

NOI DUNG

3.1. Nội tổng va lôp, ñoàng bao

3.2. Thuoã tính

3.3. Tài vui

3.4. Thôñg ñieþ

3.5. Bao goþ

3.6. Thôñ keãva *override*

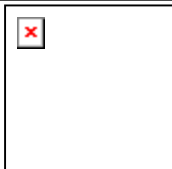
3.7. Tính ña hình

3.8. Tính vónh cõu



GIỚI THIỆU

- Mô hình hướng đối tượng giới thiệu một quan niệm lập trình (và phân tích/thiết kế) khác hẳn so với trường phái cổ điển (cấu trúc)
- Bắt đầu nhen nhóm vào những năm cuối 60s và đầu 90s trở nên rất phổ biến trong công nghiệp phần mềm
- Những ngôn ngữ hướng đối tượng đầu tiên: Smalltalk, Eiffel. Sau đó xuất hiện thêm: Object Pascal, C++, Java...
- Hình thành các phương pháp phân tích/thiết kế hướng đối tượng

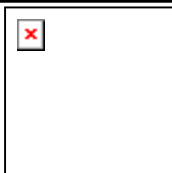
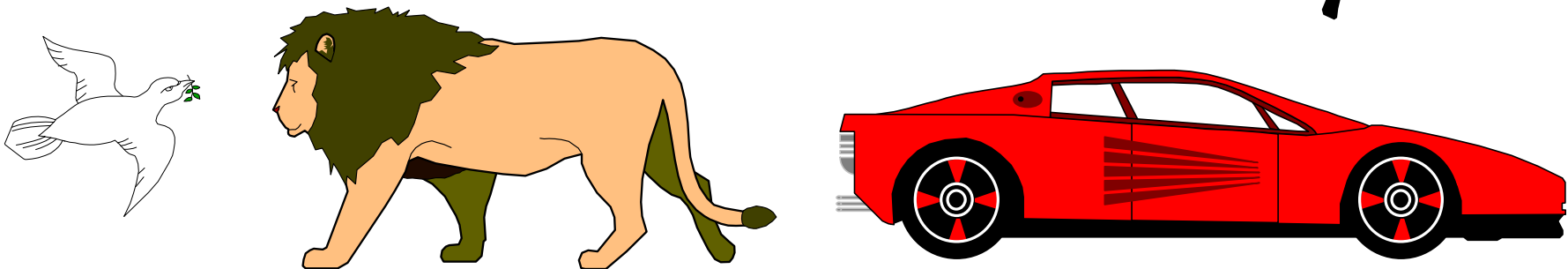


ÑOÁ TÖÔNG vaø LÔP

- Môahình höông ñoá töông quan niềm theágiôù bao gồm các ñoá töông (*object*) sinh sống vaø töông tác với nhau

- Ñoá töông bao gồm

- ◆ döù liệu: mang một giá trò nhất ñình
- ◆ tác vụ: thực hiện một công việc nào ñoù



NOI TÖÔNG vaø LÖP (t.t)

- Lööp (*class*) ñình nghóa möt tap höp caùc taùc vụ vaø thuoäc tính maø ñaùc taùc naøy ñuù caùc truoäc vaø haønh vi cuøa caùc ñoái töông.
- Ñoái töông (con goïi laø minh duy (*instance*)) ñöôùc cuøi theà hoä töø löp
- Caùc ngoân ngữ laäp trình höông ñoái töông
 - ◆ Khai baø löp: töông töï nhö khai baø möt kieäu do ngoân ngữ ñình nghóa
 - ◆ Khai baø ñoái töông: bieán cuøa kieäu löp



ÑÔNG BAO

Circle
Radius: float - x: float - y: float
+ Draw(w: Window) <u>+ GetClass(): String</u>

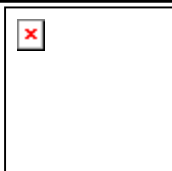
<u>c1 : Circle</u>
Radius = 1.3 x = 3.2 y = 1.7

- Ñông bao: việc gộp thuộc tính và các tài vụ trong một ñối tượng ñồng thời giới hạn cách truy xuất các thuộc tính ñó (thông phải thông qua các tài vụ get/set)



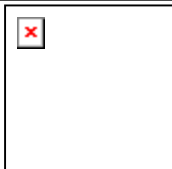
THUỘC TÍNH

- Thuộc tính (*attribute*) là một vùng chứa dữ liệu (nôn hoặc toả hợp) của lớp.
- Dữ liệu mà thuộc tính thể hiện nằm trong một khoảng giá trị nào đó xác định bởi kiểu.
- Giá trị của tất cả thuộc tính xác định trạng thái của một tổng
◆ Ví dụ: một tổng của Circle có (Radius, x, y) = (1.2, 3.4, 5.3)



THUỘC TÍNH (t.t)

- Thuộc tính có thể là bộ che dấu hoặc truy xuất nội bộ bên ngoài
 - ◆ Một số ngôn ngữ lập trình (và UML): public, protected, private
- Có 2 loại tầm vóc
 - ◆ Tầm vóc lớp: thuộc tính chung cho tất cả các đối tượng của một lớp
 - ◆ Tầm vóc đối tượng: thuộc tính của từng đối tượng (có thể mang giá trị khác nhau)
- Mức của thuộc tính chia ra số lượng dữ liệu mà bản thân thuộc tính có thể nắm giữ 0..1, 1, *, 5..8



TÀI VUI

- Tài vui (*operation*) là một dịch vụ có thể yêu cầu từ phía nhà tổng nhả hoặc hiện hành vi.
- Dấu hiệu nhận dạng của tài vui (*signature*) xác định các thông số có thể truyền cũng như kết quả trả về
- Phương thức (*method*) là phần hiện thực của tài vui

TÀI VUI (t.t)

- Tài vui có thể bảo che dấu hoặc truy xuất ở bên ngoài
 - ◆ Một số ngôn ngữ lập trình (và UML): public, protected, private
- Tài vui có thể override trong các lớp con thừa kế
 - ◆ Trừu tượng (*abstract*): không có hiện thực
- Một số ngôn ngữ lập trình cho phép định nghĩa
 - ◆ Tài vui khởi tạo (*constructor*): được gọi khi tạo ra đối tượng mới
 - ◆ Tài vui hủy (*destructor*): được gọi khi đối tượng sắp bị hủy

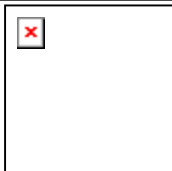


VÍ DỤ về LỚP/NOI TÖÔNG - JAVA

```
class abstract HTMLObject {
    protected static final int LEFT = 0;
    protected static final int MIDDLE = 1;
    protected static final int RIGHT = 2;
    private int alignment = LEFT;
    protected Vector objects = null;
    HTMLObject( ) {           // constructor
        objects = new Vector ( 5 );
    }
    public void setAlignment( int algnmt ) {
        alignment = algnmt;
    }
    public int getAlignment( ) {
        return alignment;
    }
    public abstract String toHTML( ); // abstract operation
}
```

THÔNG NIỆP

- Thông điệp là một phép gọi tác viên nên một nơi tổng cử thể
- Thông điệp bao gồm 3 phần
 - ◆ Nơi tổng đích
 - ◆ Dấu hiệu nhận dạng của tác viên muốn gọi
 - ◆ Danh sách thông số gọi



THÔNG ÑIỆP (t.t)

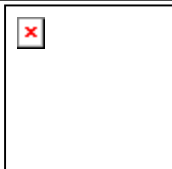
- Nói với các ngôn ngữ lập trình

- ◆ Nói tổng quát: biến nói tổng hoặc bản thân nói tổng muốn gửi thông điệp (*self, this*)

- ◆ Dấu hiệu nhận dạng của các ví dụ muốn gửi: tên các ví dụ trùng nhau \Rightarrow xem các thông số gửi

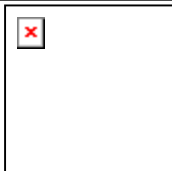
- ◆ Danh sách thông số gửi: nhớ phép gửi hàm bình thường, chuỗi kiểu khi truyền vào kết quả trả về

- ◆ Ví dụ: `aCircle.SetRadius(3);` `aCircle.Draw(pWnd);`



BAO GỘP

- Bao gộp (*aggregation*) là quan hệ giữa hai đối tượng
- Một đối tượng bao lấy đối tượng kia
- Quan hệ này thường xảy ra trong thế giới thực, ví dụ
 - ◆ Xe hơi bao gồm: bánh xe, động cơ, khung xe...
 - ◆ Trang HTML bao gồm: text, hình ảnh, tiêu đề các liên kết...
 - ◆ Checkbox, ComboBox, Slider... nằm trong một hộp thoại



BAO GỘP (t.t)

- Hai dạng liên kết giữa hai tổng nguồn và hai tổng đích
 - ◆ Chất chéo hai tổng đích nước ta ra và huyển nông thôn với hai tổng nguồn
 - ◆ Long le: chu kỳ sống của hai tổng đích lập nhau
- Quan hệ bao gộp cũng nước áp dụng cho lớp
- Một số ngôn ngữ lập trình hỗ trợ hai dạng liên kết: biến & con trỏ



VÍ DỤ về BAO GỒP - C++

```
class Geometry {           // abstract base class
public:
    Geometry( );
    ~Geometry( );
    virtual void Draw( Window *pWnd ) = 0; // abstract operation

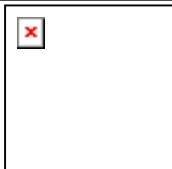
protected:
    int      xPos, yPos;
    double   xScale, yScale;
    COLORREF color;
};

class Group : public Geometry {
public:
    Group( );
    ~Group( );
    virtual void Draw( Window *pWnd ); // override
private:
    Geometry **ppGeo; // pointer container
    int      geoCount;
};
```



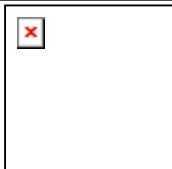
THỎA KẾ VÀ OVERRIDE

- Thỏa kế (*inheritance*) là quan hệ giữa hai lớp
- Lớp con thừa hưởng tất cả thuộc tính và các ví của lớp cha
- Lớp con (*subclass*) là dưới thể của lớp cha (*superclass*); lớp cha là tổng quát của lớp con
- Quan hệ này cũng thường được ghi nhận trong thế giới thực, ví dụ
 - ◆ Họ bà, sỏi nếu là thú
 - ◆ Button, Checkbox và Dialog nếu là Window
 - ◆ Hình tròn, hình chữ nhật, hình *ellipse* nếu là hình vẽ 2D.



THỎA KEÁ VÀ OVERRIDE (t.t)

- Hai loai thỏa keá ñôn thỏa keá và ña thỏa keá
- Ñôn thỏa keá mỗi lớp con có nhiều nhất là một lớp cha
- Ña thỏa keá
 - ◆ Mỗi lớp con có một hoặc nhiều lớp cha
 - ◆ Nảy sinh hai vấn đề ñể ñối với tên các thành phần (*member*) của lớp cha và thỏa keá lại



THỎA KEÁ VÀ OVERRIDE (t.t)

- Lớp con có thể *override* lại một số tài sản của lớp cha.
- Phải giữ nguyên dấu vết nhận dạng (*signature*) của tài sản bị *override*; chẳng thể thay đổi phương thức (phần hiển thức) của nó
- Nó có thể lập trình hoặc không hỗ trợ *thỏa keá*
- Một số ngôn ngữ đưa ra khái niệm phương thức ảo (*virtual*)

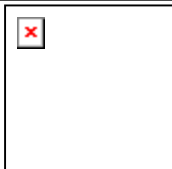
VÍ DỤ về THỎA KEÁ và OVERRIDE - JAVA

```
class HTMLDocument extends HTMLObject {
    private String title = null;
    // other attributes...
    HTMLDocument( ){
    }
    public void setTitle( String ttl ) {
        if ( ttl != null )
            title = ttl;
    }
    public String getTitle( ) {
        return title;
    }
    public String toHTML( ) { // override
        StringBuffer html = new StringBuffer;
        // additional implementation...
        return html.toString( );
    }
}
```

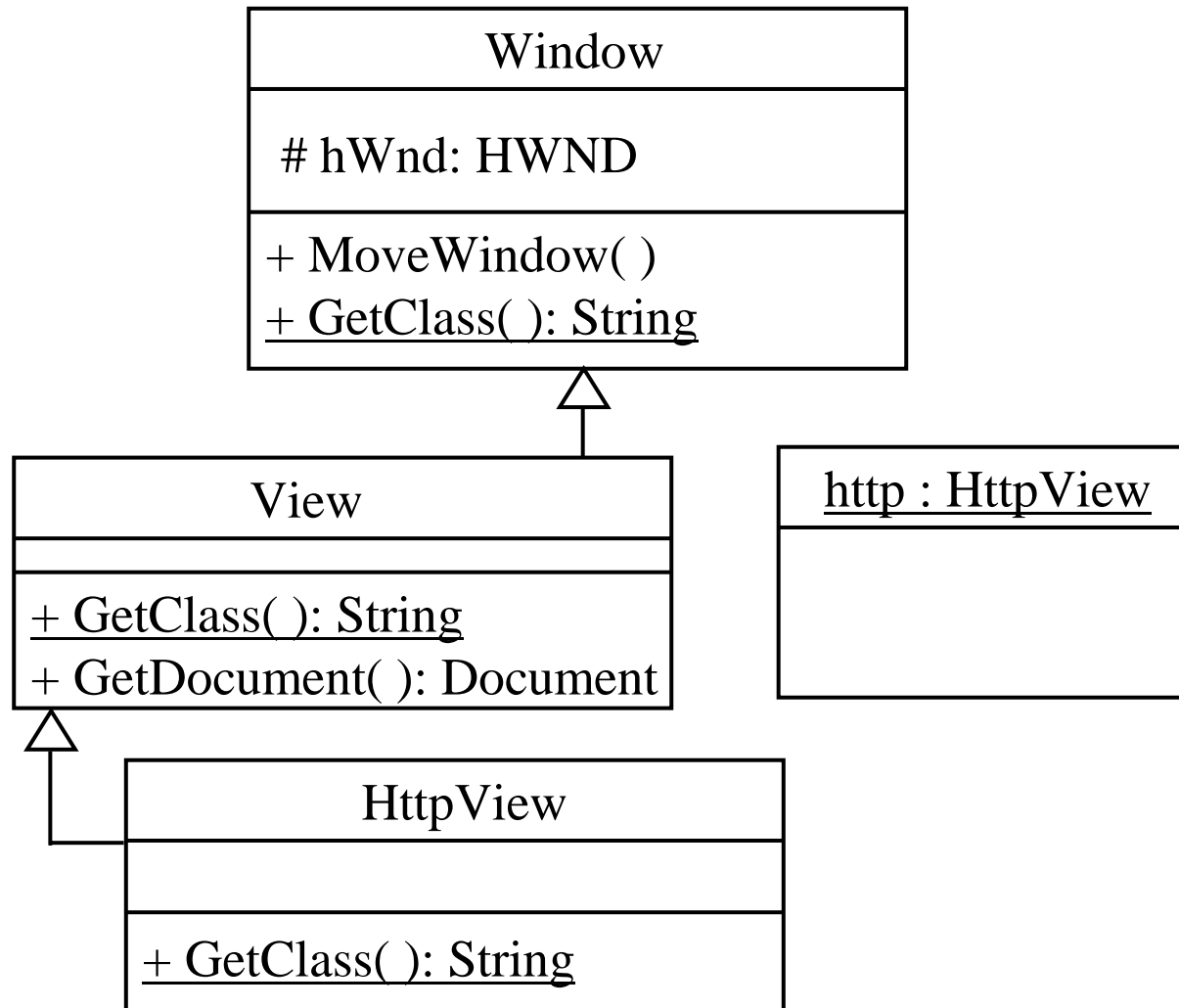


TÍNH ÑA HÌNH

- Mỗi ngôn ngữ mang nhiều biến thể khác nhau: của chính lớp ngôn ngữ và của các lớp con \Rightarrow tính đa hình (*polymorphism*)
- Tổng hợp kiểu: kiểu của lớp con luôn tổng hợp với kiểu lớp cha
- Một số ngôn ngữ lập trình định nghĩa khái niệm liên kết mạnh
 - ◆ Hàm hoặc gọi tên các thông tin xác định trong thời gian thực thi chứ không phải biên dịch
 - ◆ Mỗi ngôn ngữ có một bảng phương thức ảo



TÍNH NÃ HÌNH (t.t)



Ví dụ: nối tổng http

nhớ xem nhớ thuộc

kiểu của HttpView,

View vào Window



TÍNH VĨNH CỜU

- Chu kỳ sống của nơ-ron: khoảng thời gian từ lúc nơ-ron được tạo ra đến lúc nó bị hủy diệt.
- Thông thường chu kỳ sống của nơ-ron gói gọn trong thời gian chôn trình thức thi

TÍNH VĨNH CỬU (t.t)

- Chu kỳ sống của một đối tượng có thể vượt ra khỏi thời gian của chương trình \Rightarrow tính vĩnh cửu (*persistence*)
 - ◆ Một đối tượng được cất vào bộ nhớ vĩnh cửu khi chương trình kết thúc
 - ◆ Khi cần thiết có thể khôi phục lại một đối tượng vào bộ nhớ chính
 - ◆ Chưa lâu trở lại trạng thái của một đối tượng
 - ◆ Ngôn ngữ C++ và Java: streaming

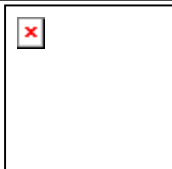


TỔNG KẾT

- Mô hình hööng ñoá tööng quan niếm theágiöü bao gồm các ñoá tööng söng chung và tööng tác với nhau: ☹️ ☹️

- Các ñaéc ñiếm chính

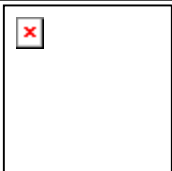
- ◆ Nöng bao: möí ñoá tööng bao gồm döülieu và tác vui. Các tác vui thiết lập nên hạnh vi của ñoá tööng. Các ñoá tööng ñöüc phân loaiï bằng löp
- ◆ Các ñoá tööng tööng tác với nhau bằng cách gửi thông ñiép
- ◆ Giöüa các löp/ñoá tööng có theá tồn tại quan hệ bao gôp và thöa keá
- ◆ Tính ña hình: ñoá tööng mang nhiều bömaät
- ◆ Tính vönh cöü: ñoá tööng có theá “ngüi”



Chương 4

MO HÌNH NGHIỆP VỤ VÀ THU THẬP YÊU CẦU

- ❖ *Actor & use-case*
- ❖ *Mo hình use-case*



NOI DUNG

4.1. Nhãn diễn các vai trò (*actor*)

4.1.1. Khái niệm *actor*

4.1.2. Nhãn diễn *actor*

4.1.3. *Actor* trong UML

4.2. Nhãn diễn các trường hợp sử dụng (*use-case*)

4.1.1. Khái niệm *use-case*

4.1.2. Tìm kiếm *use-case*

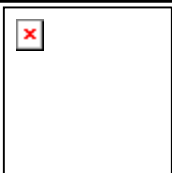
4.1.3. *Use-case* trong UML

4.3. Thiết lập các mối quan hệ

4.3.1. Quan hệ liên kết (*association*)

4.3.2. Quan hệ giao tiếp, gộp vào môi trường

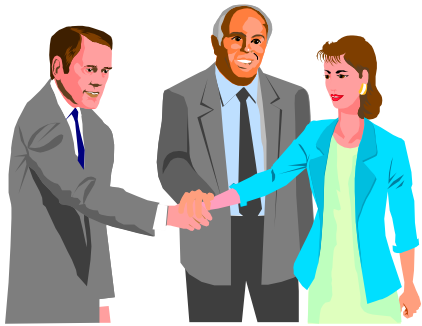
4.4. Xây dựng mô hình *use-case*



GIỚI THIỆU

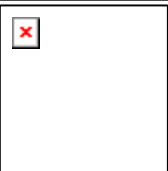
- Khách hàng và nhà phát triển gặp nhau cùng thảo luận về yêu cầu của hệ thống phần mềm cần xây dựng

- Mô hình nghiệp vụ sẽ được thiết lập nên môi trường nhà phát triển lần khách hàng trong việc kiểm chứng lại và thống nhất yêu cầu phần mềm và vai trò các tài nhân bên ngoài



KHÁI NIỆM ACTOR

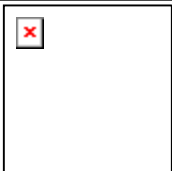
- *Actor* xác định một hoặc nhiều vai trò mà người hoặc vật sẽ đóng vai khi tương tác với hệ thống phần mềm
- *Actor* nằm ngoài phạm vi của hệ thống
 - ◆ Chưa quan tâm các thông điệp mà *actor* gửi hay nhận
 - ◆ Không quan tâm cấu trúc bên trong của *actor*
- Phân loại *actor*
 - ◆ Chuyên / Thời yếu
 - ◆ Tích cực / Thuỷ động



NHÂN DIỄN CÁC ACTOR

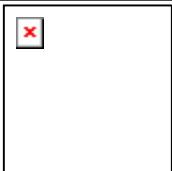
- Trau lồi một số câu hỏi nhỏ

- ◆ Ai là người sử dụng chức năng chính của hệ thống ?
- ◆ Ai cần sử dụng hệ thống để thực hiện công việc thông nhất của họ ?
- ◆ Ai phải thực hiện công việc bảo dưỡng, quản trị và giao cho hệ thống hoạt động ?
- ◆ Hệ thống sẽ kiểm soát thiết bị phần cứng nào ?
- ◆ Hệ thống năng xây dựng cần tương tác với những hệ thống khác hay không ?
- ◆ Ai hoặc vật thể nào quan tâm đến hay chịu ảnh hưởng bởi kết quả mà hệ thống phần mềm tạo ra ?

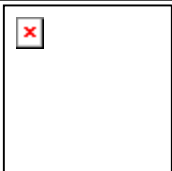
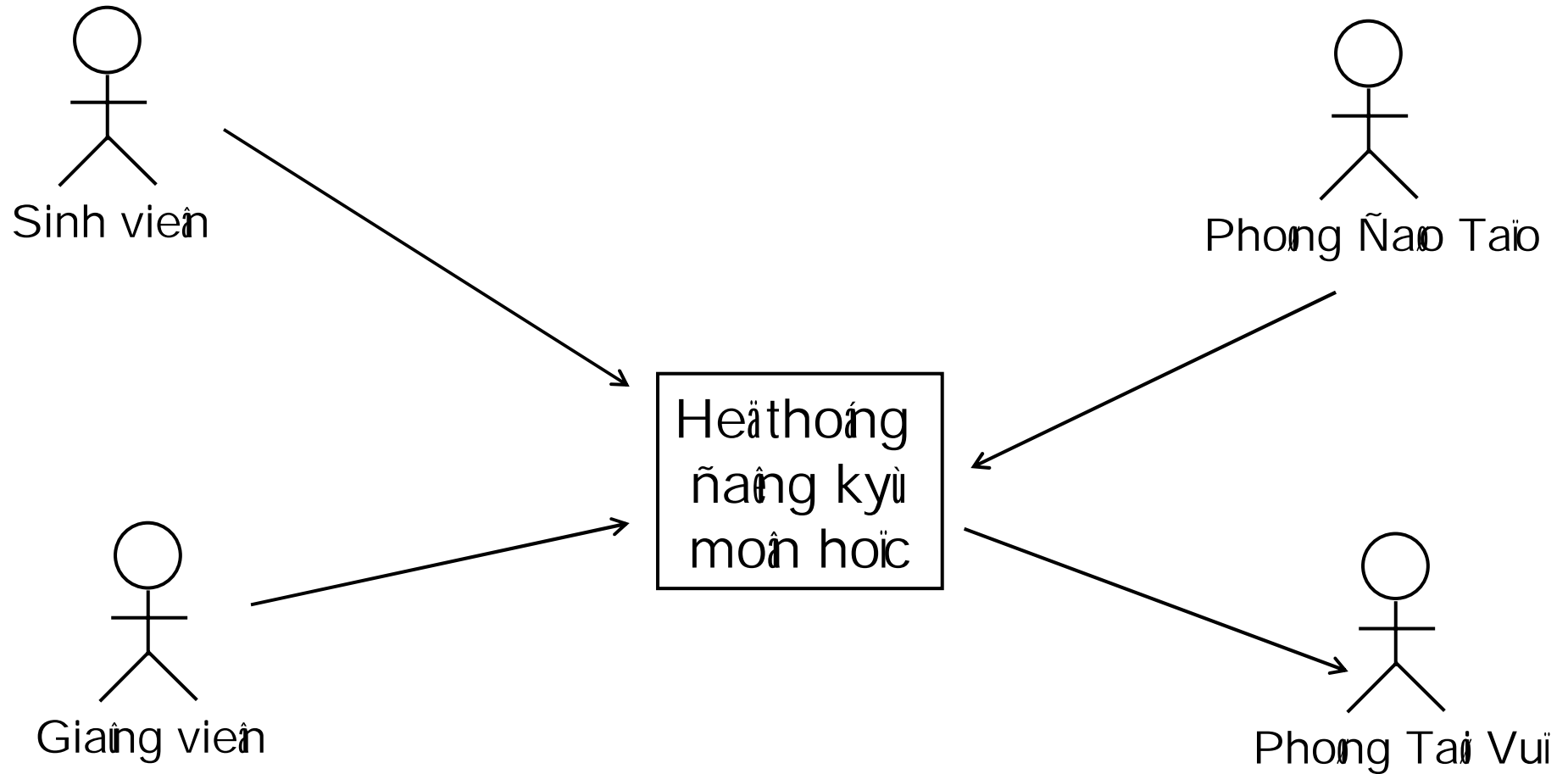


ACTOR trong UML

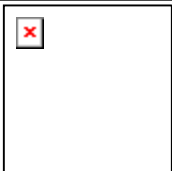
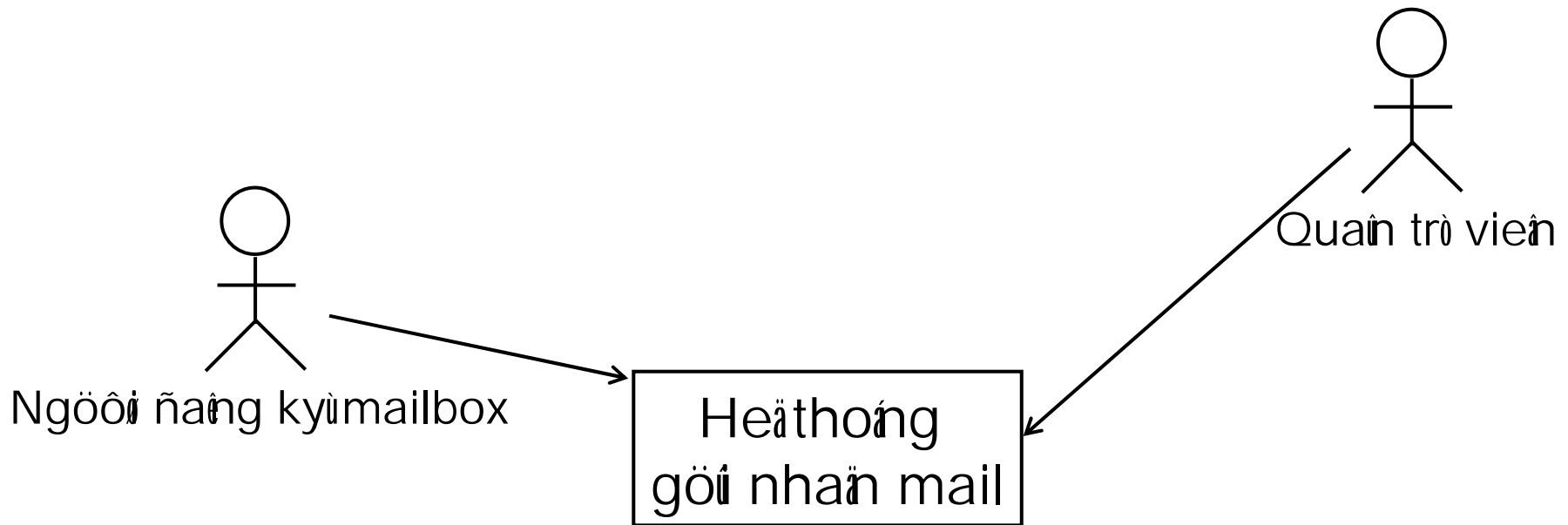
- Actor được biểu diễn bằng ký hiệu hình người
- Actor được xem là một lớp (class) có stereotype là <<actor>>
- Giữa các actor có thể có quan hệ tương quan
◆ Ví dụ: Sinh viên, giảng viên và khách thuê nhà là các thành viên của hệ thống quản lý nhà ở
- Ví dụ: một hệ thống năng lượng hoặc trong trường hợp này



ACTOR trong UML (t.t)

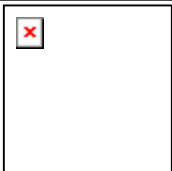


ACTOR trong UML (t.t)



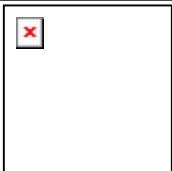
KHÁI NIỆM *USE-CASE*

- *Use-case* biểu diễn một chức năng của hệ thống phần mềm
- *Use-case* nội bộ biểu diễn bằng một chuỗi các thông điệp trao đổi bên trong hệ thống và một hoặc một số thông điệp trao đổi với *actor*
- Một số quy ước
 - ◆ *Use-case* luôn luôn nội bộ bắt đầu bằng thông điệp đến từ *actor*
 - ◆ *Use-case* phải hoàn tất: chuỗi thông điệp phải kết thúc bằng kết quả cuối cùng
 - ◆ Lỗi thông gặp: chia nhỏ *use-case* trở thành những chức năng vụn vặt



KHÁI NIỆM *USE-CASE* (t.t)

- *Niệm mô hình* là một vị trí trong *use-case* mà tại đó có thể xác định chuỗi sự kiện của một *use-case* khác
- *Use-case* có thể chứa nhiều kiện rẽ nhánh, xử lý lỗi, ngoại lệ..
- Minh diễn của *use-case* là kịch bản (*scenario*): miêu tả chuỗi trình tự các sự kiện



TÌM KIẾM *USE-CASE*

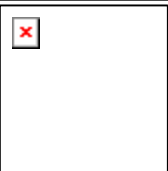
- Trau lòi một số câu hỏi nhỏ

- ◆ *Actor* yêu cầu chức năng gì của hệ thống ?

- ◆ *Actor* cần phải nhớ, tạo, xóa sửa đổi hoặc lưu trữ thông tin nào nào của hệ thống không ?

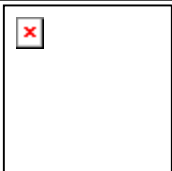
- ◆ *Actor* cần thiết phải nhớ các cảnh báo và những sự kiện trong hệ thống, hay *actor* cần phải báo hiệu cho hệ thống về vấn đề nào nào không ?

- ◆ Hệ thống có thể hoặc rồi một số công việc thông nhất của *actor* nào nào hay không ?



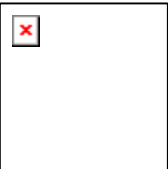
TÌM KIẾM *USE-CASE* (t.t)

- Một số câu hỏi khác cần chú ý
 - ◆ Hệ thống cần dữ liệu *input/output* nào ? Dữ liệu nào nên tồn tại ?
 - ◆ Những khối nào liên quan đến hiện thực của hệ thống hiện tại (chẳng hạn hệ thống quản lý bằng giấy tờ nên được thay thế bằng hệ thống quản lý trên máy tính) ?

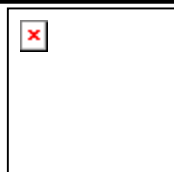
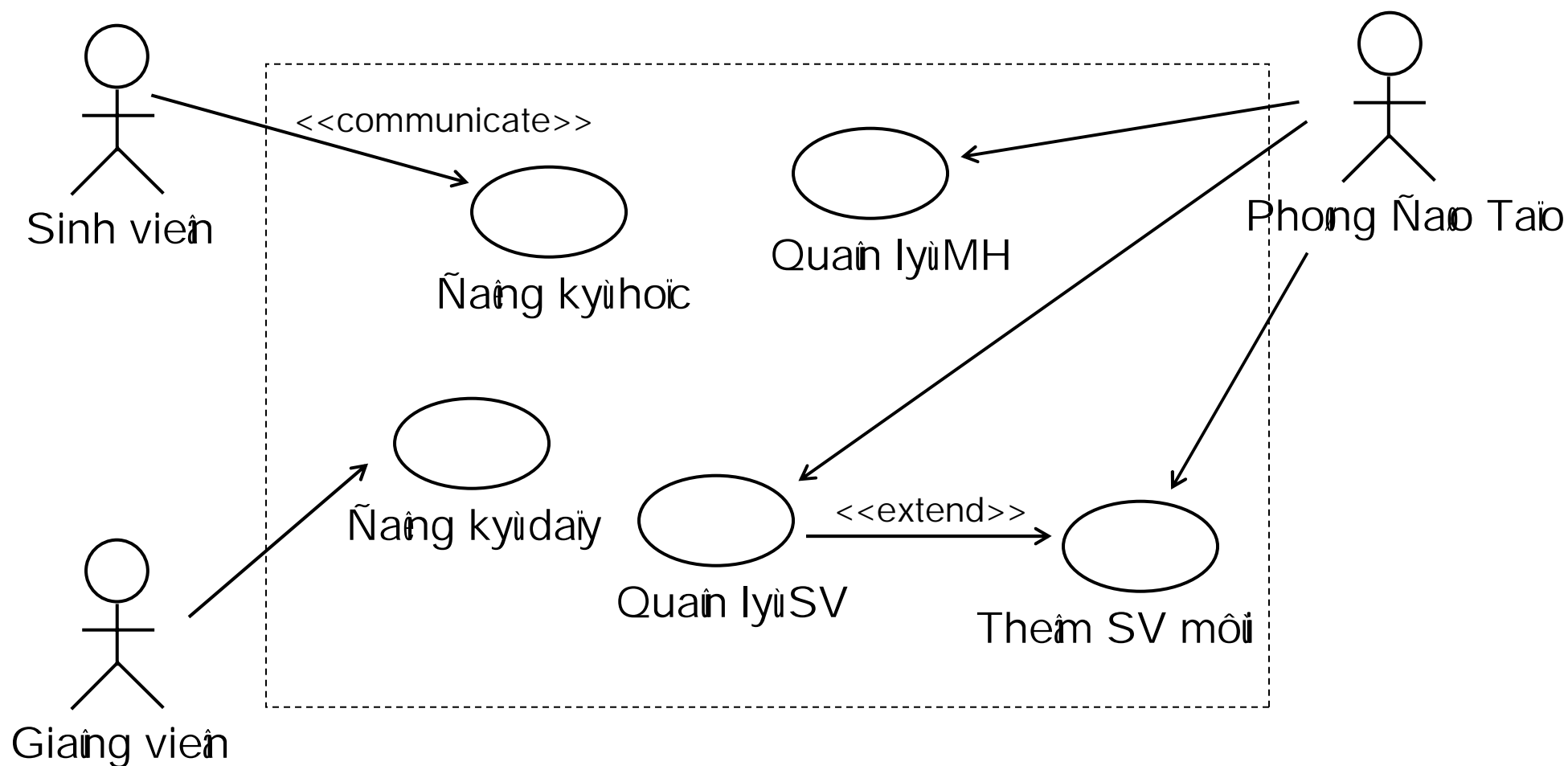


USE-CASE trong UML

- *Use-case* được biểu diễn bằng hình *ellipse*
 - Giữa *use-case* và *actor* thông qua quan hệ liên kết
 - Giữa các *use-case* cũng có quan hệ liên kết hoặc tổng quát hoá
 - Ví dụ: một hệ thống năng lượng điện hoặc theo tín hiệu trong trường
- nhà hoặc

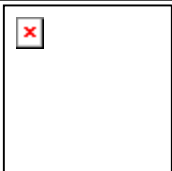


USE-CASE trong UML (t.t)



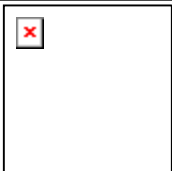
THIẾT LẬP CÁC MỐI QUAN HỆ

- Quan hệ giữa *actor* với *actor*
- Quan hệ giữa *actor* với *use-case*
- Quan hệ giữa *use-case* với *use-case*
- UML mô tả ra quan hệ liên kết (*association*)



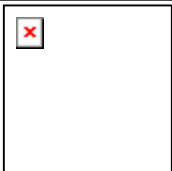
QUAN HỆ LIÊN KẾT

- Quan hệ liên kết chỉ ra một quan hệ cụ thể giữa hai bên
 - ◆ Trong thực tế hành khách với lái xe, sinh viên với giáo viên, giảng viên với môn học...
- Một số tính chất liên quan
 - ◆ Tên của liên kết
 - ◆ Một chiều hay 2 chiều
 - ◆ Bậc: số lượng thực thể tham gia vào liên kết tại mỗi bên



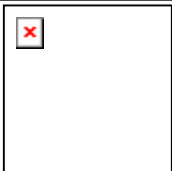
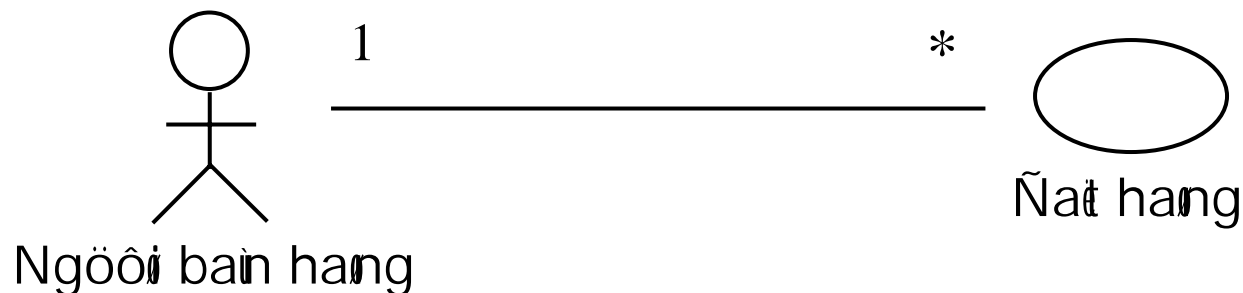
QUAN HỆ LIÊN KẾT trong UML

- UML biểu diễn liên kết nhờ là một *hình thức* (hai chiều) hoặc *đường thẳng* (một chiều)
- Có thể áp dụng *stereotype*:
 - ◆ <<include>>
 - ◆ <<extend>>
 - ◆ <<communicate>>
 - ◆ ...



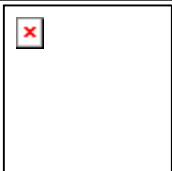
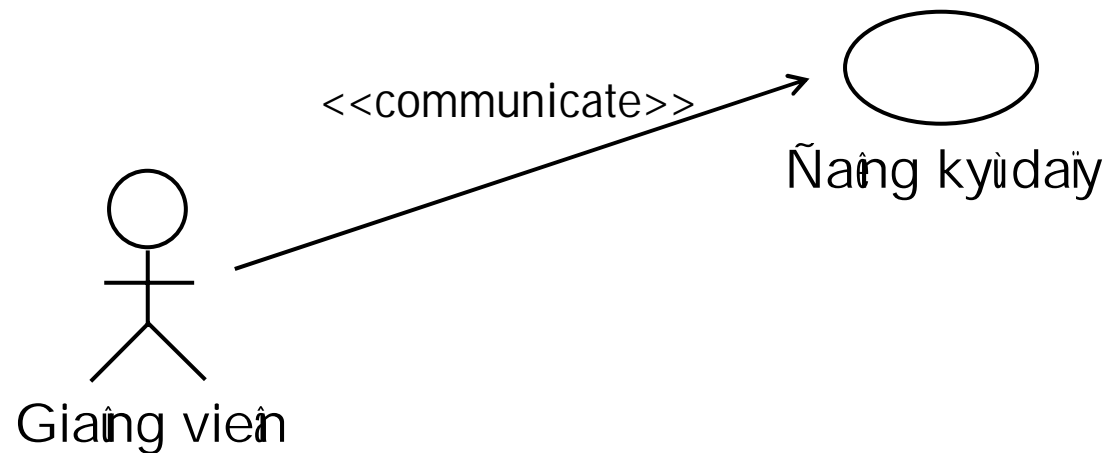
LIÊN KẾT trong MÔ HÌNH NGHIỆP VỤ

- Liên kết là quan hệ duy nhất giữa *actor* và *use-case*
- Có thể là một chiều hoặc hai chiều
 - ◆ *actor* kích hoạt *use-case* và nhận kết quả về là liên kết 2 chiều
 - ◆ *actor* kích hoạt *use-case*, không quan tâm kết quả về là liên kết 1 chiều



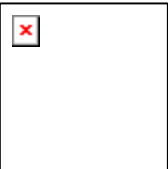
QUAN HỆ GIAO TIẾP

- Là quan hệ liên kết có *stereotype* là `<<communicate>>`
- Dùng để liên kết giữa *actor* với *use-case* mà nó kích hoạt



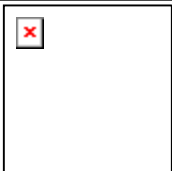
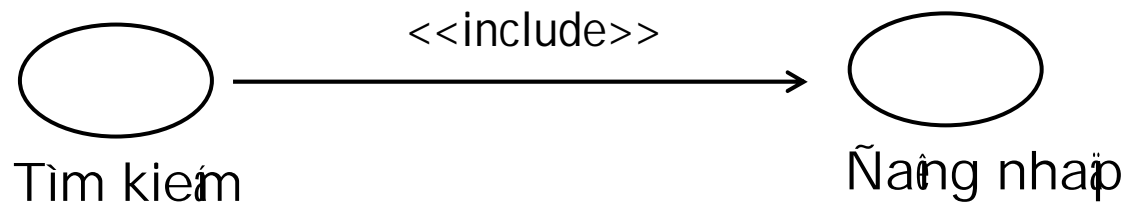
QUAN HỆ GỒP

- Là quan hệ liên kết có *stereotype* là <<include>>
- Dùng để liên kết giữa 2 *use-case*
- Trong *use-case* nguồn có một nhiệm vụ môi trường mà tại nó bắt buộc phải thêm *use-case* đích vào



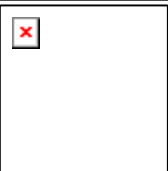
QUAN HỆ GỒP (t.t)

- Tài liệu mô tả, diễn tiến của *use-case* nguồn tìm thời ngừng lại nếu chuyển sang diễn tiến của *use-case* đích
- Khi kết thúc *use-case* đích, diễn tiến của *use-case* nguồn lại tiếp tục



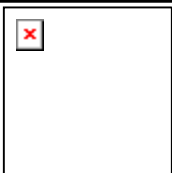
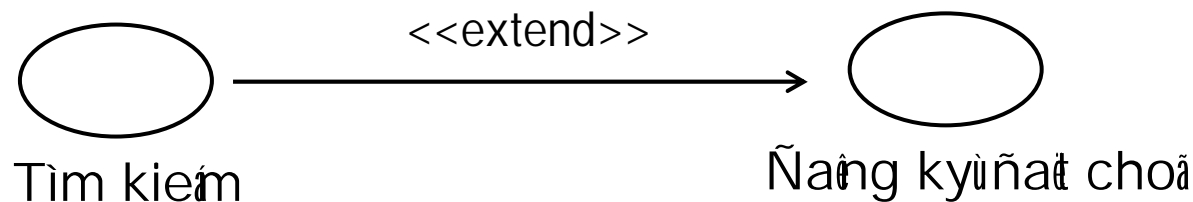
QUAN HỆ MÔI TRƯỜNG

- Là quan hệ liên kết có *stereotype* là <<extend>>
- Dùng để liên kết giữa 2 *use-case*
- Trong *use-case* nguồn có một nhiệm vụ môi trường mà tài nguyên có thể (hoặc không) phải thêm *use-case* khác vào
- Thêm hay không phụ thuộc vào điều kiện riêng hành hoặc tổng tài nguyên phía *actor*



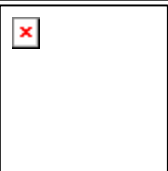
QUAN HỆ MÔI TRƯỜNG (t.t)

- Tại điểm môi trường, nếu một môi trường thì diễn tiến của *use-case* nguồn tạm thời ngừng lại và chuyển sang diễn tiến của *use-case* đích
- Khi kết thúc *use-case* đích, diễn tiến của *use-case* nguồn lại tiếp tục



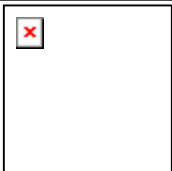
XÂY DỰNG MÔ HÌNH *USE-CASE*

- Các yêu cầu của phần mềm được miêu tả trong mô hình *use-case*
- Mô hình *use-case* bao gồm các lược đồ *use-case* (*use-case diagram*) và (có thể) một số *package*
- Mỗi lược đồ *use-case* bao gồm các *actor*, *use-case* và các mối quan hệ
- Có thể sử dụng *package* để gom một số *use-case* liên quan tạo thành một bộ chức năng con của hệ thống

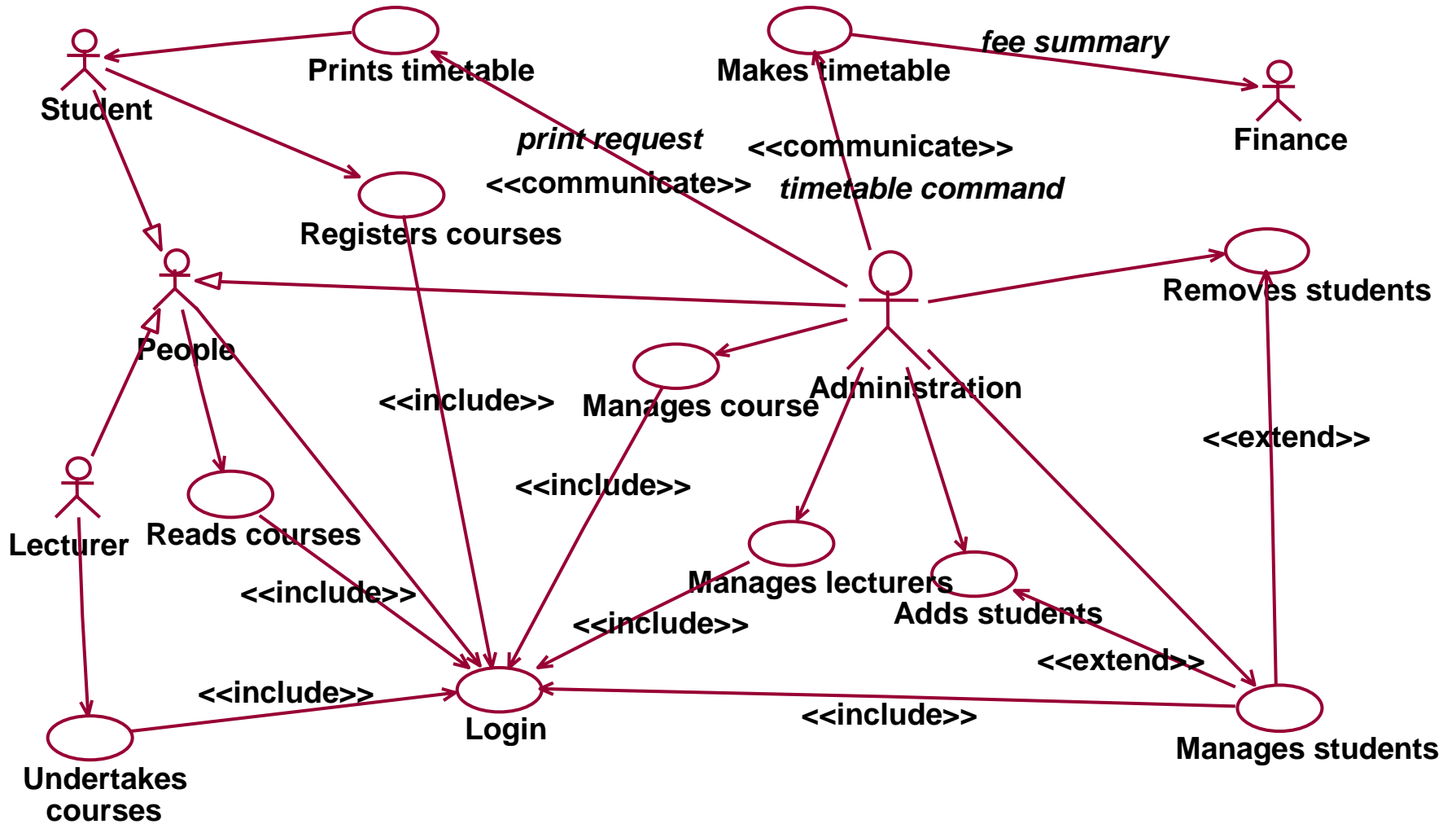


XÂY DỰNG MÔ HÌNH *USE-CASE* (t.t)

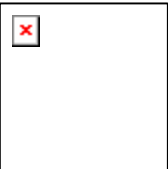
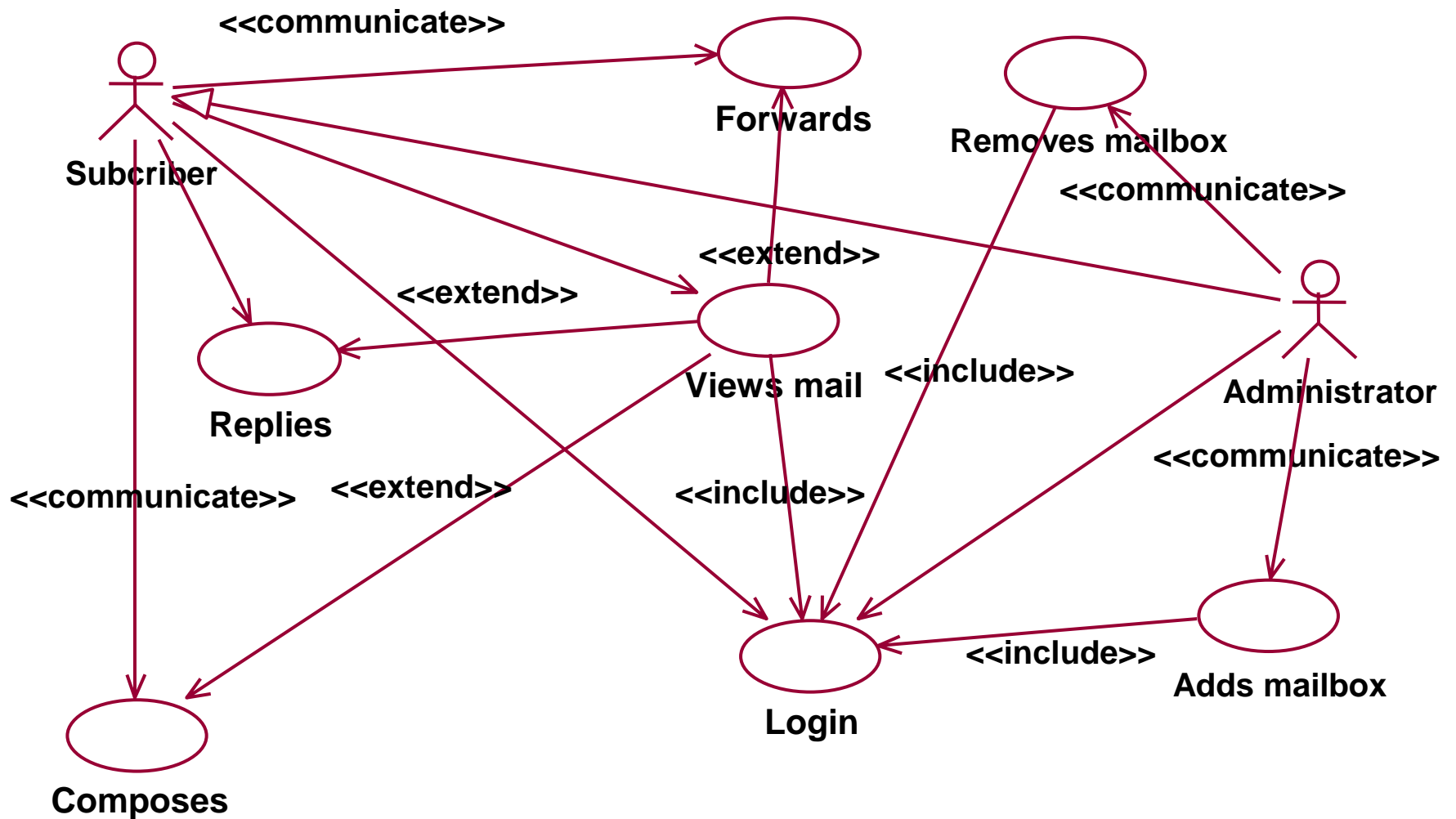
- Các quan hệ có thể xảy ra trong lược đồ *use-case*
 - ◆ Quan hệ liên kết giữa *actor* và *use-case*: một chiều hoặc hai chiều, thông có *stereotype* là <<communicate>>
 - ◆ Quan hệ môi trường hay gộp giữa 2 *use-case*: quan hệ liên kết với *stereotype* <<extend>> hay <<include>>
 - ◆ Quan hệ tổng quát hoá (*generalization*) giữa các *actor*: nhiều *actor* có vai trò của một *actor* tổng quát
 - ◆ Quan hệ tổng quát hoá giữa các *use-case*: nhiều *use-case* là trường hợp cụ thể của một *use-case* tổng quát



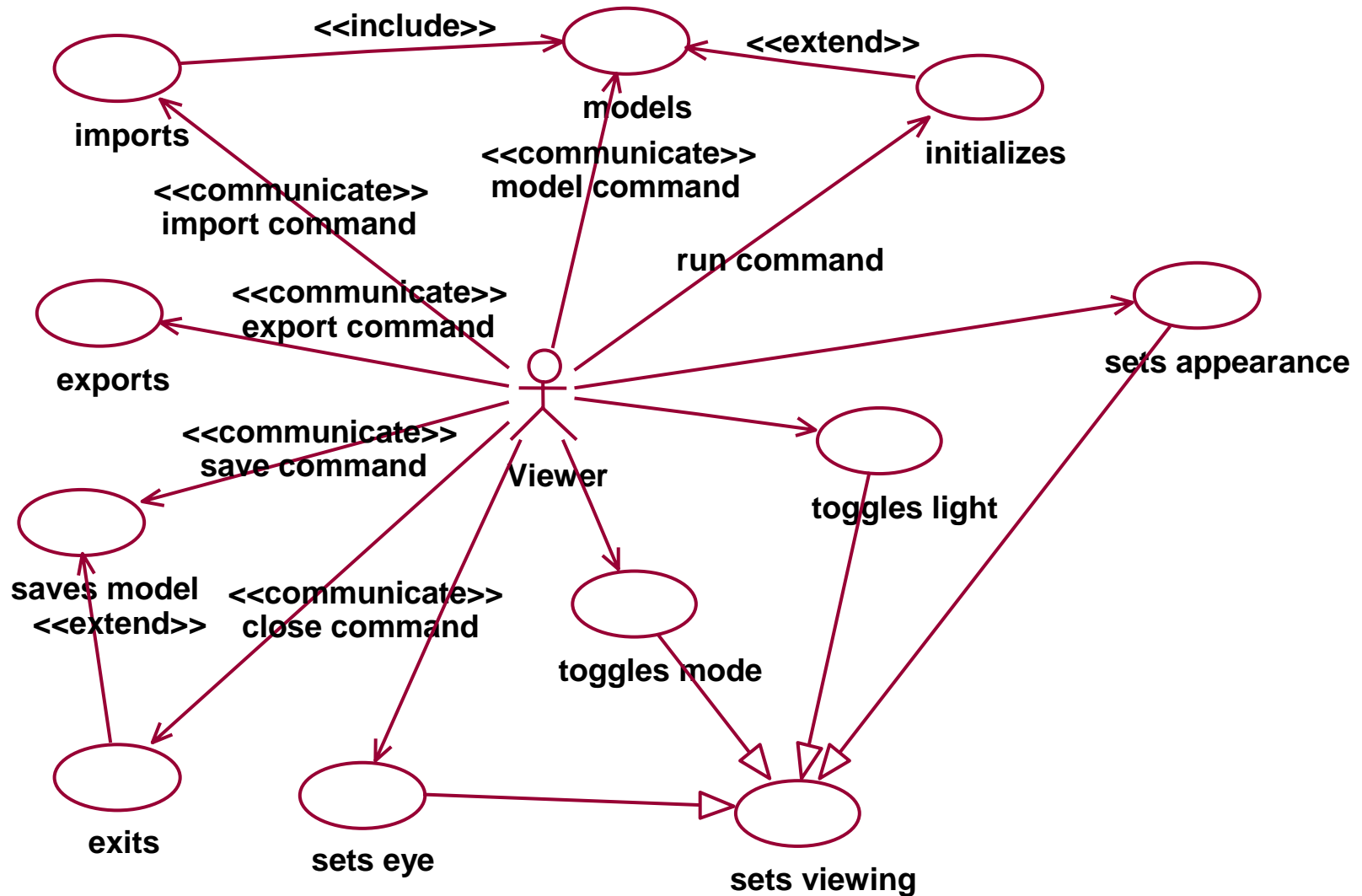
XAÂY DÖÖNG MÖÄHINH *USE-CASE* (t.t)



XÂY DỰNG MÔ HÌNH *USE-CASE* (t.t)



XÂY DỰNG MÔ HÌNH *USE-CASE* (t.t)



TỔNG KẾT

- Mô hình nghiệp vụ thể hiện các chức năng của hệ thống phần mềm và các mối liên quan



- UML định nghĩa mô hình *use-case* bao gồm các *actor*, các *use-case* và các lược đồ *use-case*
- Tiếp theo mô hình nghiệp vụ là mô hình các nỗ lực phân tích

Chương 5

PHÂN TÍCH YÊU CẦU HỒNNG NỘI TÖÖNG

- ❖ Nhận diện nội tÖÖng/lÖp
- ❖ LÖÖc nÖlÖp

NOI DUNG

5.1. Nhận diện các nội tổng/lớp

5.1.1. Nội tổng/lớp thóc thea

5.1.2. Nội tổng/lớp biên

5.1.3. Nội tổng/lớp nhiều khiên

5.2. Nhận diện các thuộc tính

5.2.1. Kiểu dữ liệu của thuộc tính

5.2.2. Bậc của thuộc tính

5.2.3. Mức nối truy xuất thuộc tính

5.3. Nhận diện các tài vui

NOI DUNG (t.t)

5.4. Nhaän dieän löp cô sôu

5.4.1. Nhaän dieän caà thuoc tính/tàc vuichung

5.4.2. Quan heätöng quatt hoà (*generalization*)

5.5. Nhaän dieän caà moà quan heä

5.5.1. Quan heälieän keä (*association*)

5.5.2. Quan heäbao göp (*aggregation*)

5.6. Xay döng löôc ñoälöp

5.7. Thiet lap caà package

GIỚI THIỆU



- Mô hình nghiệp vụ biểu diễn các chức năng phần mềm cần xây dựng dưới dạng các *use-case*
- Mô hình phân tích sẽ tìm kiếm các mối tổng "sống" trong ngữ cảnh của phần mềm
- Các mối tổng sẽ tương tác với nhau để tạo nên các chức năng mô tả bởi *use-case*

GIỚI THIỆU (t.t)

- Mô hình phân tích tập trung mô tả vai trò và cấu trúc của các node tổng
- Chờ quan tâm đến hành vi của thể và nhiệm vụ chi tiết của chúng trong ngữ cảnh của hệ thống
- Nguyên tắc: mô hình phân tích phải độc lập với o/s, ngôn ngữ lập trình, công cụ phát triển



NHÂN DIỆN NÓI TƯỜNG/LỘP

- Dựa vào các tài liệu tổng *use-case* để tìm kiếm các nói tường
- Các nói tường thường xuất hiện trong các danh từ hay nhóm danh từ
- Một số lưu ý
 - ◆ Không nên dùng nói tường để biểu diễn một dữ liệu nào (nên xem là thuộc tính của nói tường khác)
 - ◆ Nói tường/lớp phải thực hiện cần thiết cho sự hoạt động của hệ thống
 - ◆ Nói tường/lớp \neq bảng cơ sở dữ liệu
 - ◆ Nói tường/lớp \neq *actor*



NHAN DIEN NOI TÖÔNG/LÖP (t.t)

● Phân loại nói töông/löp

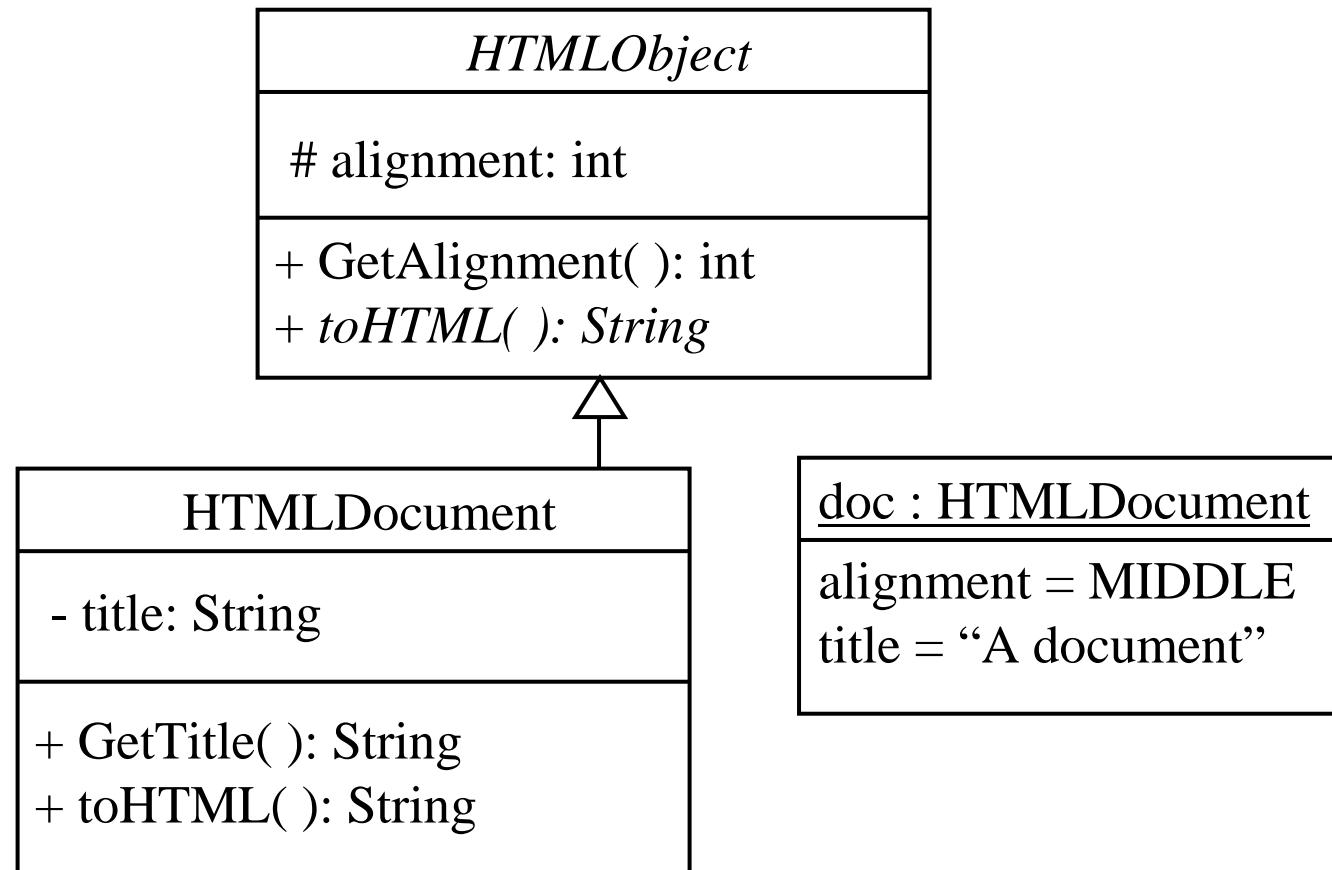


- ◆ Nói töông thöc theä (*entity*): biéu dién các thông tin thiết yếu của hệ thống, có thể ẩn hoặc lộ trong cơ sở dữ liệu
- ◆ Nói töông biên (*boundary*): thể hiện chức năng giao tiếp với *actor*
- ◆ Nói töông khiển (*control*): khiển các nói töông khác

NHÂN DIỄN ÑOÁ TÖÔNG/LÔP (t.t)

- Trong UML, lớp ñöôc biểu diễn bằng một hình chöõnhäđ gồm 3 phần: tên, các thuộc tính và các tác vụ
- Có thể áp dụng *stereotype* cho lớp: <<entity>>, <<boundary>>, <<control>>...
- Ñoá töông cũng ñöôc biểu diễn bằng hình chöõnhäđ, thông thường gồm 2 phần: tên ñoá töông + tên lớp (ñöôc gạch chân), giao trò các thuộc tính (trạng thái của ñoá töông)

NHÃN DIỄN NỘI TỔNG/LỚP (t.t)



NÓI TÖÔNG/LÖP THÖC THEÁ

- Biểu diễn cho các thực thể xuất hiện một cách tự nhiên trong hệ thống
- Thông tin về các mối tương tác thực thể có thể phải nhớ lâu lâu dài (*database, file...*)
- Trong UML, nhớ gắn *stereotype* <<entity>>
- Đánh dấu các thuộc tính của chúng

NÓI TÖÔNG/LÖP THÖC THEÁ(t.t)

● Ví dụ:

Message <<entity>>
subject: String # sent: Date # content: String
+ GetSubject(): String + toString(): String

- ◆ Nói với hệ thống năng kỳ môn hoặc hệ tin chæ qua WEB, nhận diện các nói töông thök theánhö: thông tin SV, thông tin GV, nhóm lớp hoặc, năng kỳ môn, soạn sinh viên...
- ◆ Nói với hệ thống mail, nhận diện các nói töông thök theánhö: hộp thö, thông điệp mail...
- ◆ Nói töông nöông nöng möc, nốt gây vaø bân nöa trong chöông trình vẽ beà mặt nöa hình



NỘI TỒN/LỘP BIÊN

- Thực hiện chức năng giao tiếp với *actor*
- Thông chứa các phần tử hoặc nhiều khiên giao diện người dùng (nút nhấn, hộp danh sách, tùy chọn, menu...)
- Trong UML, nó có gắn *stereotype* <<boundary>>
- Nhận biết các thuộc tính và các vai trò trong mô hình phân tích



ÑOÁ TÖÔNG/LÔP BIÊN (t.t)

- Ví dụ:

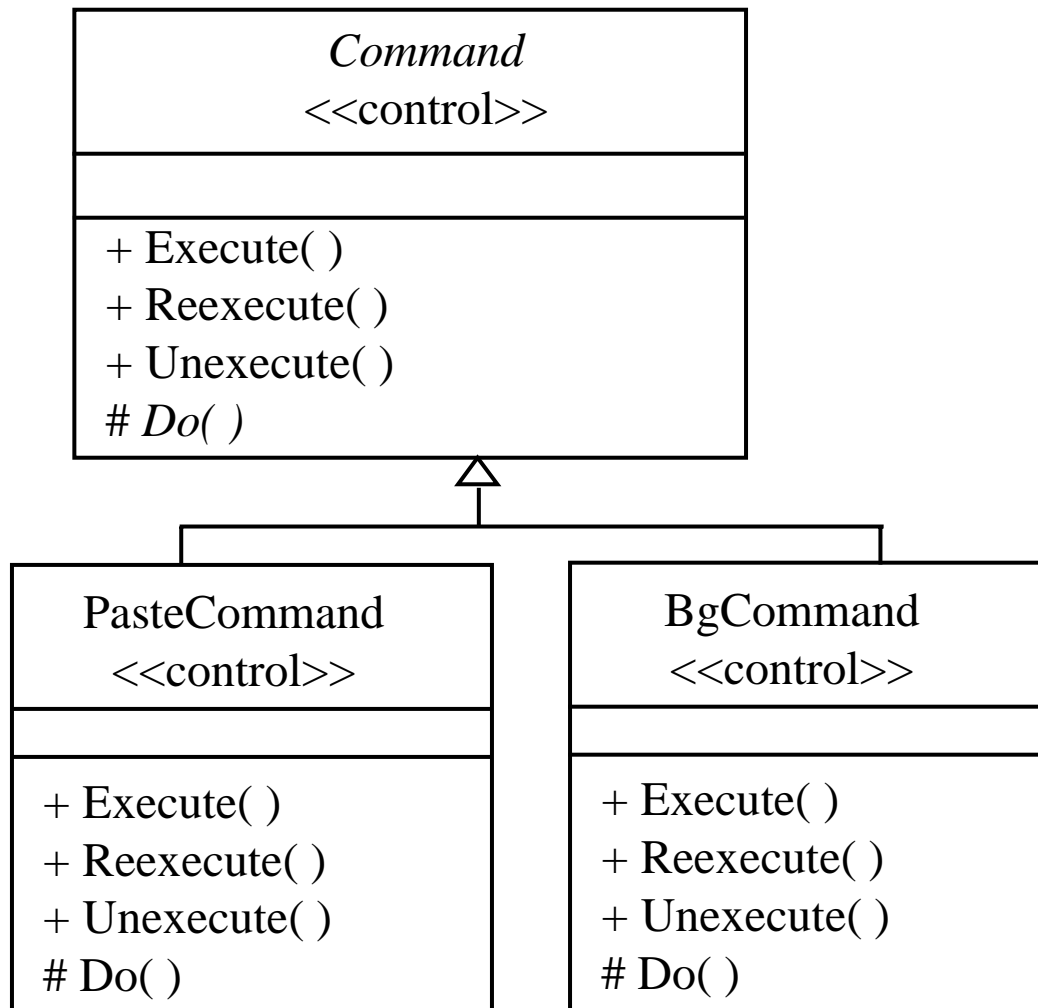
MailView <<boundary>>

- ◆ Nói với hệ thống năng khiếu hoặc hệ thống qua WEB, nhận diện các nói tổng biên nhö: RegisterForm, StudentForm...
- ◆ Nói với hệ thống mail, nhận diện các nói tổng biên nhö: MailView, MailCompose...

NỘI TỔNG/LỚP NHIỆM KHIỂN

- Cùng làm việc với nhiệm vụ khác hoặc
- (Nên gần hơn) Những lớp không phải là lớp thực thể và lớp biên
- Trong UML, nên gắn *stereotype* <<control>>
- Lớp biên thông thường có quan hệ liên kết hoặc phụ thuộc với các lớp khác

NGHĨA TỔNG/LỘP NGHĨA KHIỂN (t.t)



● Ví dụ:

◆ Nghĩa tổng biểu diễn
một số lệnh thông
thông nhớ cắt, dán,
thay nội thông số ảnh
trong hiện thời nào đó...

NHÂN DIỆN CÁC THUỘC TÍNH

- Dựa vào các tài liệu tổng *use-case*, tìm kiếm các danh từ hoặc nhóm danh từ liên quan đến nội tổng năng xét
- Trả lời câu hỏi: những thành phần nào cấu thành nội tổng năng xét ?
- Lưu ý cung cấp nội tổng trong các ngữ cảnh khác nhau chúng ta có thể tìm thấy các thuộc tính khác nhau



NHÂN DIỆN CÁC THUỘC TÍNH (t.t)

- Nên xác định (tuy nhiên không bắt buộc) trong mô hình phân tích
 - ◆ Kiểu của thuộc tính: một số kiểu cơ bản
 - ◆ Giá của thuộc tính: số ít hoặc số nhiều
 - ◆ *Visibility* của thuộc tính: mức nào cho phép truy xuất thuộc tính từ bên ngoài
- UML: thuộc tính được miêu tả trực tiếp hoặc thông qua quan hệ với các lớp khác



KIEU DỮ LIỆU CỦA THUỐC TÍNH

- Một số kiểu cơ bản của các ngôn ngữ lập trình: integer, float, double, long, char...
- Một số kiểu cơ bản khác: string, date, time...
- UML cho phép nhìn nhận hóa tất cả các kiểu dữ liệu trên



BÀI CỬA THUỐC TÍNH

- Bài cử cửa thuốc tính: số ít hay số nhiều
- Nếu thuốc tính không cần tự động minh: dùng dấu [] để chỉ số nhiều hoặc số lượng chính xác
- Trường hợp thuốc tính không miêu tả thông qua quan hệ với các lớp khác: UML cho phép thể hiện bài trên quan hệ (ví dụ: 1, 0, *, 2..9, 0..n)

MÔC ÑOÄTRUY XUAÁT THUOÁC TÍNH

- UML ñình nghĩa 3 môc ñoät truy xuất thuộc tính (*visibility*)
 - ◆ *public* (+): có thể truy xuất thuộc tính từ tất cả các vị trí khác nhau
 - ◆ *protected* (#): bản thân lớp ñang xét và các lớp con của nó có thể truy xuất thuộc tính
 - ◆ *private* (-): chỉ có lớp ñang xét có thể truy xuất thuộc tính
- Thông thường nên ñặt môc ñoät truy xuất thuộc tính là *private* hoặc *protected* (cho các lớp cô sô), không nên là *public*. Thuộc tính nên ñược truy xuất thông qua các vi get/set

NHÂN DIỄN CÁC TÁC VỤ

- Đưa vào các tác vụ của từng *use-case*, tìm kiếm các năng lực hoặc nhóm năng lực liên quan nên nói tổng năng lực
- Chú ý xem nói tổng năng lực tạo ra và bù đắp những thiếu sót nào ?
Trong thời gian ngắn/nhận thông tin ra sao ?
- Các nói tổng biên các tác vụ nhận lệnh từ *actor*.

NHÂN DIỄN CÁC TẠC VUI (t.t)

- Xem xét mối nối truy xuất của các vui tổng tối nhỏ với các thuộc tính; các các vui tổng có *visibility* là + hoặc #
- Một số các vui không xuất hiện một cách tự nhiên trong mô hình phân tích \Rightarrow mô hình thiết kế sẽ nghiên cứu kỹ các nhiệm vụ hành vi của tổng nối tổng

VÍ DỤ về NHÂN DIỆN THUỘC TÍNH

StudentInfo <<entity>>
- name: String - code: Long - dateOfBirth: Date - addr: String - acaYear: Date - department - home: String - socialAid
+ GetName(): String + GetCode(): Long

LecturerInfo <<entity>>
- name: String - code: String - dateOfBirth: String - addr: String - degree - title: String - division - health - experience: Date
+ GetName(): String + GetCode(): String

- Hệ thống nâng kỳ môn học hết tín chỉ qua WEB - Nhân diện các thuộc tính cho các môn học: StudentInfo, LecturerInfo

VÍ DỤ về NHÃN DIỄN THUỘC TÍNH (t.t)

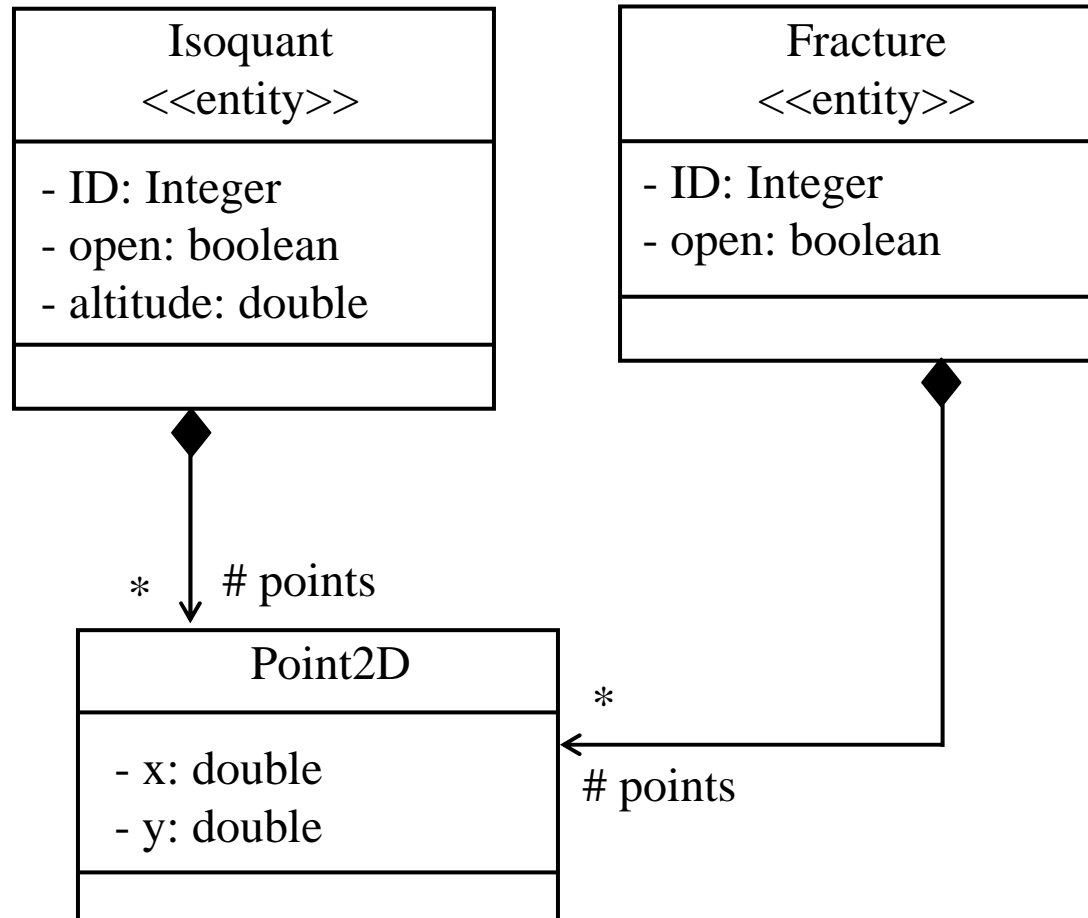
CourseOffering <<entity>>
- courseName: String - courseCode: String - offering: int - session - credit: int - prerequisite

Catalog <<entity>>
- acaYear: Date - semester

- Hệ thống nâng kỳ môn học hết tín chỉ qua WEB - Nhận diện các thuộc tính cho các mối tổng: CourseOffering, Catalog



VÍ DỤ về NHÃN DIỄN THUỘC TÍNH (t.t)



- Chương trình biểu diễn bề mặt nhôa hình - Nhãn diễn các thuộc tính cho các nĩa tổng: Isoquant, Fracture

NHÂN DIỆN LỚP CÔ SÔU

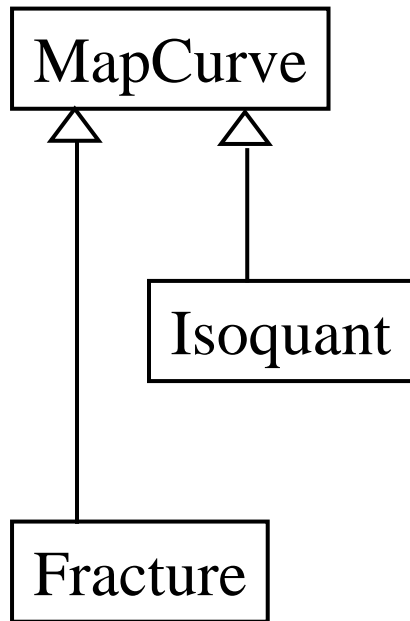
- Lớp cô sôu (*base class*) được nhân diện sau khi đã nhân diện các lớp con
- Sự xuất hiện của lớp cô sôu làm cho mô hình phân tích có tính dụng lại cao (*reusability*) và dễ mở rộng (*scalability*)
- UML hỗ trợ quan hệ tổng quát hóa (*generalization*)
- Lớp cô sôu trừu tượng (không thể tạo ra đối tượng) có tên in nghiêng

NHÂN DIỆN các THUỘC TÍNH/TÀI VUI CHUNG

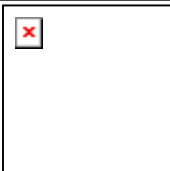
- Nói với các nhà tổng/lớp thực tập tìm các thuộc tính chung để hình thành lớp cơ sở
- Ví dụ
 - ◆ Trong hệ thống quản lý thư viện qua WEB: các nhà tổng Book, Magazine có một số thuộc tính chung \Rightarrow hình thành lớp LibraryItem
 - ◆ Nói với hệ thống nâng kỳ môn học tín chỉ qua WEB: lớp PeopleInfo là lớp cơ sở của StudentInfo và LecturerInfo
 - ◆ Chương trình vẽ biểu đồ nhà hình: lớp MapCurve là lớp cơ sở của những vòng tròn Isoquant và vết gãy Fracture



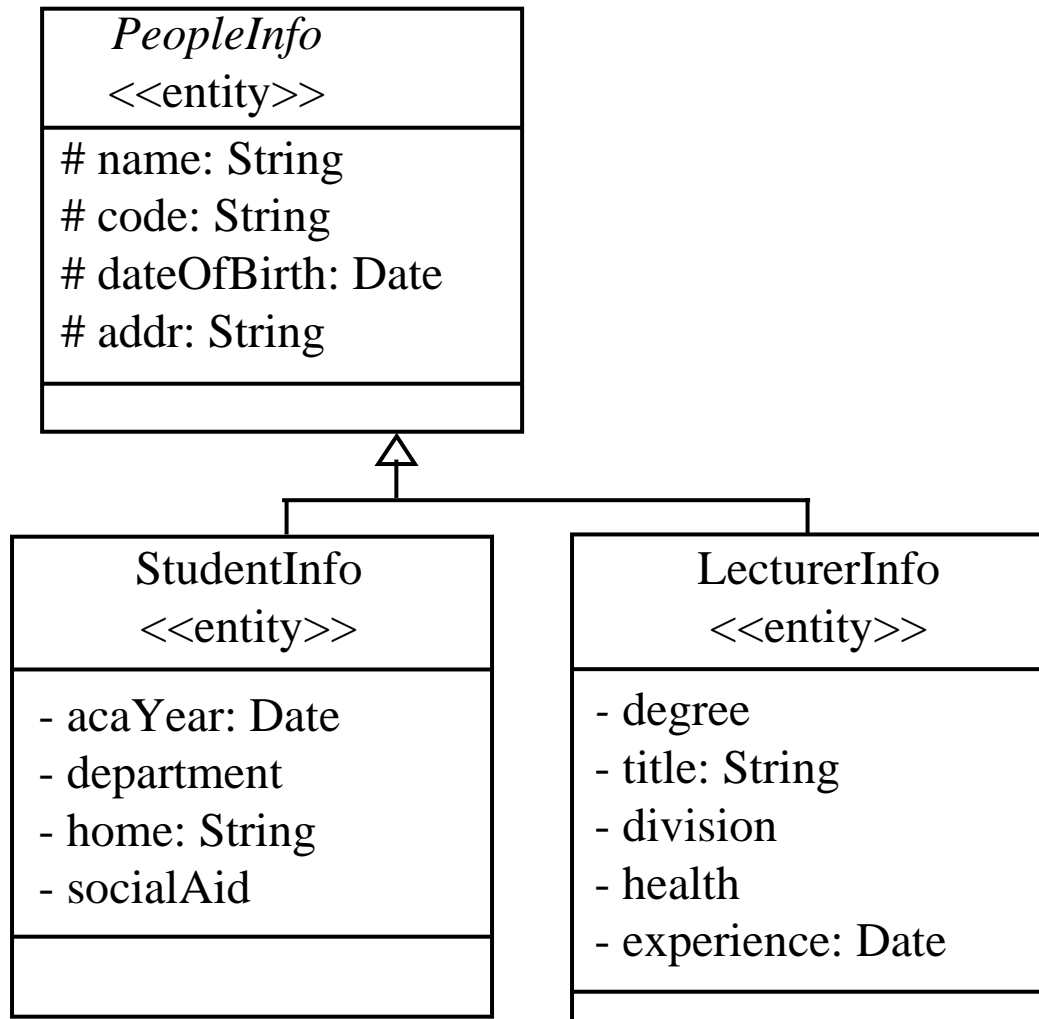
QUAN HỆ TỔNG QUAN



- UML nhìn nghĩa quan hệ tổng quát hoá giữa một lớp tổng quát hơn với một lớp cụ thể hơn: lớp cụ thể hơn có thể có các thuộc tính, các phương pháp của lớp kia.
- Ký hiệu: mũi tên có đầu là một tam giác nhỏ
- Lớp tổng quát hơn nằm về phía mũi tên



QUAN HỆ TỔNG QUAN HOÀ (t.t)



- Ví dụ: trong hệ thống nâng kỹ năng hoặc tín chấp qua WEB, lớp PeopleInfo là tổng quát hoá của StudentInfo và LecturerInfo

NHAN DIEN CAI MOI QUAN HE

- Trong mô hình phân tích các đối tượng/lớp có quan hệ với nhau
- Một số quan hệ mà UML hỗ trợ
 - ◆ Tổng quát hóa (*generalization*)
 - ◆ Liên kết (*association*)
 - ◆ Bao gộp (*aggregation*)
- Các quan hệ khác nữa có thể áp dụng cho mô hình thiết kế
 - ◆ Phụ thuộc (*dependency*)
 - ◆ Cụ thể hóa (*realization*)

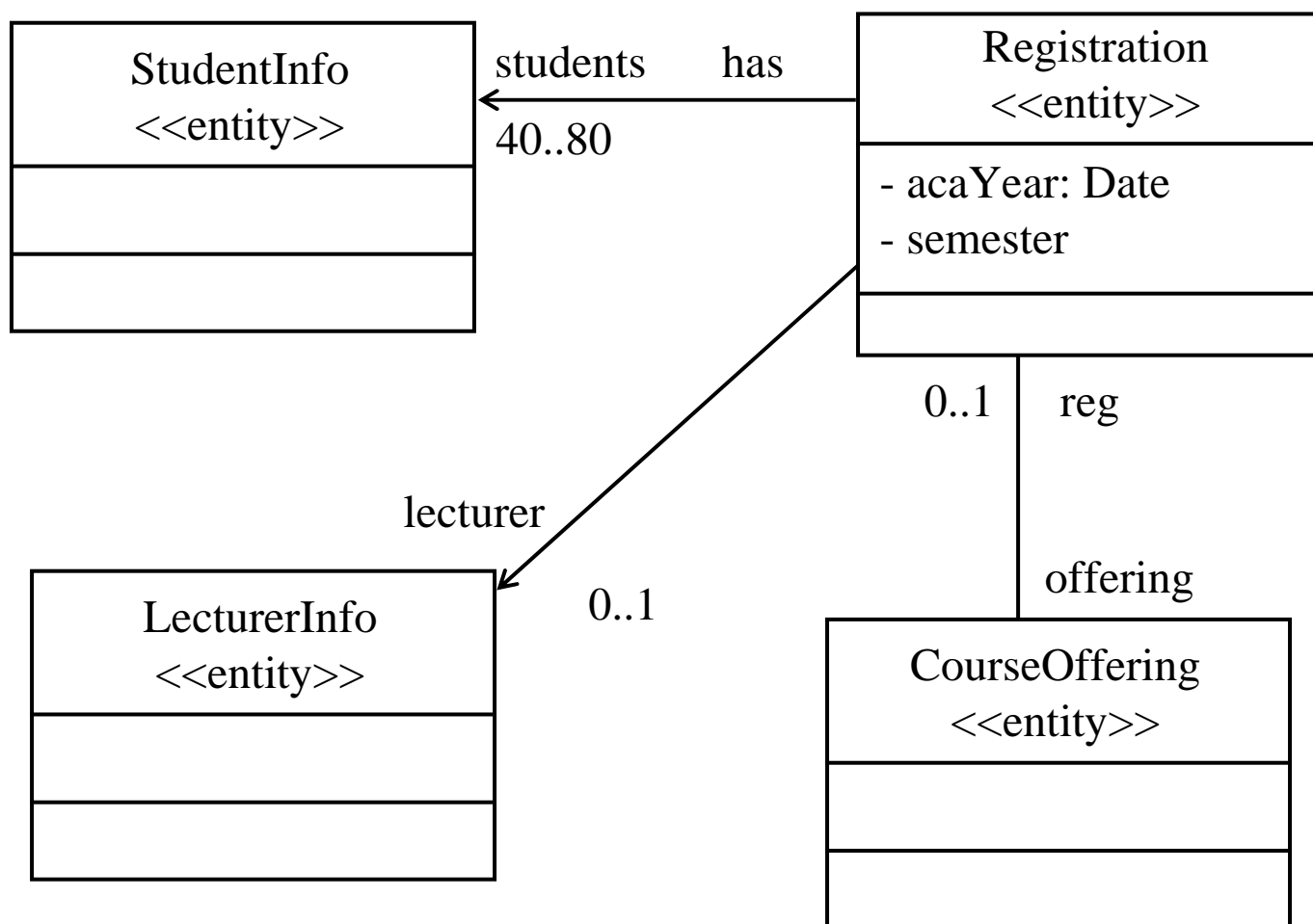


QUAN HỆ LIÊN KẾT

- Về ứng dụng và kỹ thuật giống nhờ quan hệ liên kết trong mô hình nghiệp vụ
- Áp dụng cho 2 lớp có mối tương quan mang ứng dụng nhất định
- Chú ý ghi rõ (nếu có thể)•
 - ◆ Xác định vai trò của mỗi lớp trong quan hệ
 - ◆ Tên của chính quan hệ liên kết



QUAN HỆ LIÊN KẾT (t.t)



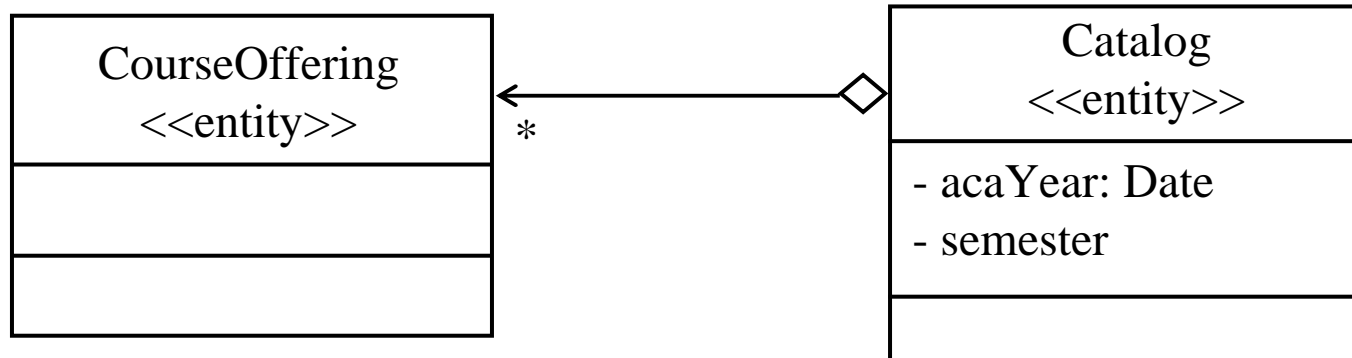
- Ví dụ: lớp Registration liên kết với lớp StudentInfo, LecturerInfo và CourseOffering

QUAN HỆ BAO GỒP

- UML định nghĩa quan hệ bao gộp là tổng hợp các thành phần của quan hệ liên kết, khi mà một hoặc nhiều liên kết trở thành một bao gộp (*aggregation*)
- Lớp của một bao gộp sẽ bao hàm lớp kia
- Ký hiệu của một bao gộp là một hình chữ nhật hoặc không tô nền
- Có hai dạng bao gộp
 - ◆ Chia sẻ (*shared*): chia sẻ giữa các bao gộp khác nhau
 - ◆ Hoàn toàn (*composite*): sở hữu đầy đủ



QUẢN LÝ BÁO GÓP (t.t)



● Ví dụ:

- ◆ Nội với hệ thống năng kiểm tra hoặc tin cậy qua WEB, lớp Catalog bao gồm lớp CourseOffering
- ◆ Cần có giao diện báo góp hoàn toàn thành cuốn và menu



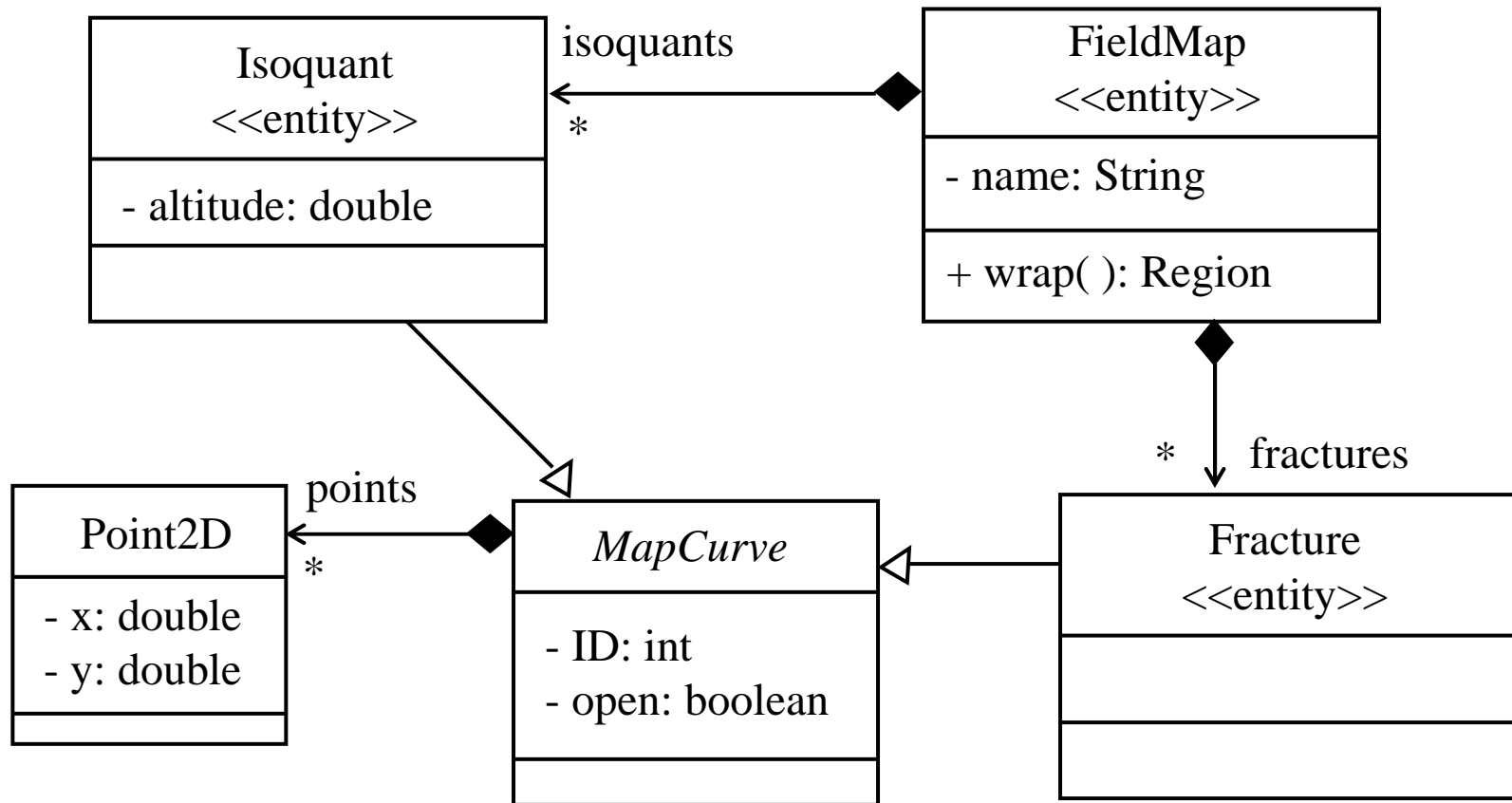
XÂY DỰNG LÖÖC ÑOÀ LỚP

- Lööc ñoà lớp (*class diagram*) biểu diễn cấu trúc của một số lớp và quan hệ giữa chúng \Rightarrow mô tả khía cạnh tĩnh (*static*) của hệ thống
- Hệ thống phức tạp có nhiều lớp \Rightarrow cần xây dựng nhiều lööc ñoà lớp, mỗi lööc ñoà mô tả một phần của hệ thống
- Lööc ñoà lớp ñược bổ sung và hoàn thiện trong mô hình thiết kế (thêm một số lớp, chi tiết các thuộc tính và các vùi, làm rõ các quan hệ)



XÂY DỰNG LÖÖC ÑOÀ LỘP (t.t)

- Ví dụ: một lược ñoà lớp của chương trình hiển thò bề mặt ñòa hình



THIẾT LẬP CÁC PACKAGE

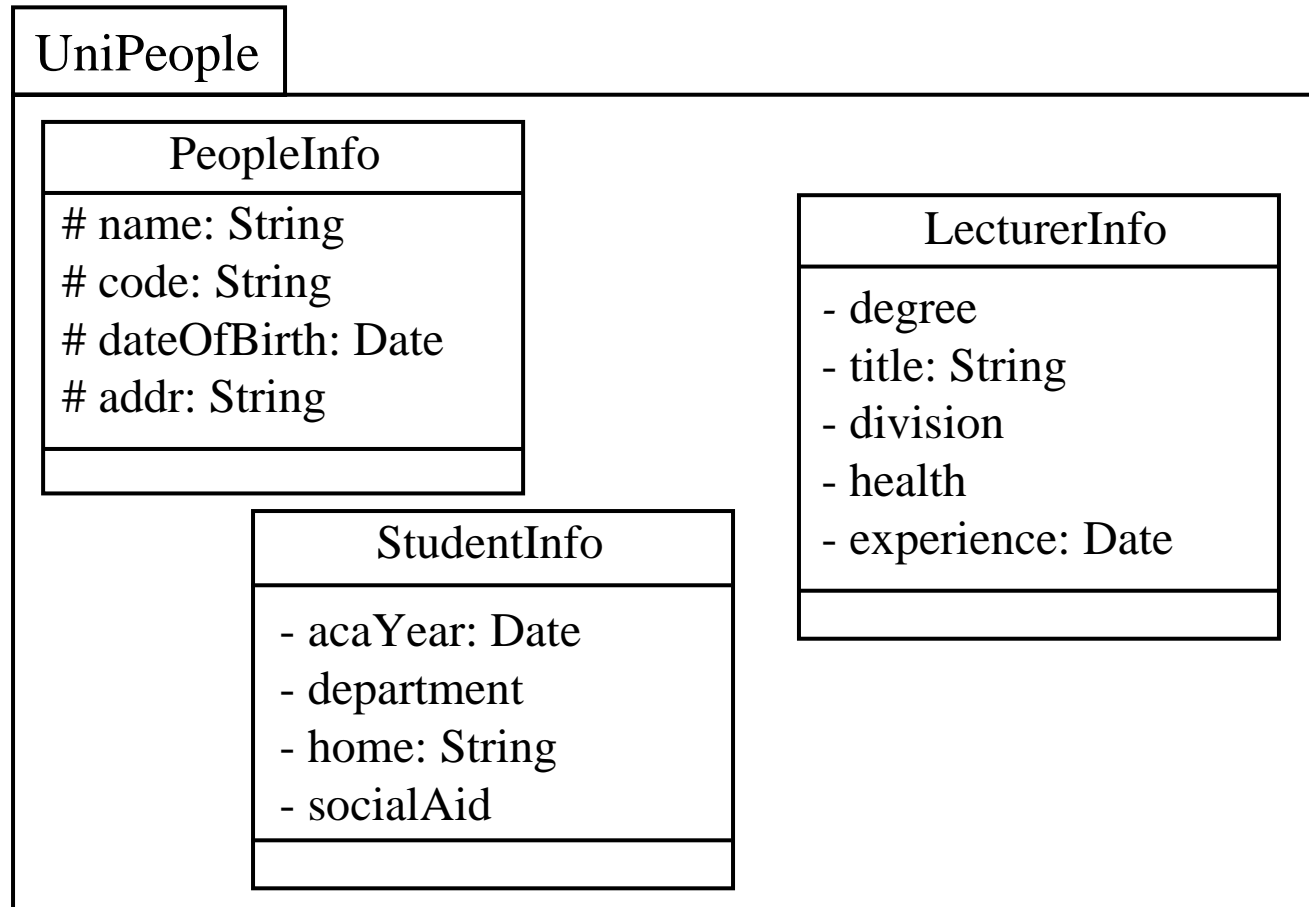
- *Package* là một cơ chế để tổ chức các phần tử vào các nhóm có liên hệ và đồng nghĩa với nhau
- *Package* có thể *import* các phần tử từ một *package* khác
- Có thể chia ra quan hệ giữa các *package*
 - ◆ Phụ thuộc
 - ◆ Tổng quát hoá



THIẾT LẬP CÁC PACKAGE (t.t)

- Mục đích truy xuất của *package*
 - ◆ *Private*: chỉ nội bộ các *package import* nội bộ truy xuất nội dung
 - ◆ *Protected*: giống như *private* nhưng cho phép thêm các *package* dẫn xuất
 - ◆ *Public*: các *package* khác có thể truy xuất nội dung
 - ◆ *Implementation*: không cho phép *import*, có thể áp dụng cho các phần tử bên trong *package*

THIẾT LẬP CÁC PACKAGE (t.t)



- Ví dụ: package

UniPeople chứa
các lớp liên quan
nên thông tin con
người

TỔNG KẾT

- Mô hình phân tích nhận diện các khối tổng/lớp: thời thế biến, nên khiến
- Nhận diện các thuộc tính và mối so sánh vui, tuy nhiên chờ làm rõ hành vi của chúng (\Rightarrow mô hình thiết kế)
- UML hỗ trợ mô tả phần tử lớp, khối tổng, lược đồ lớp, *package*



Chương 6

CÔNG SƠ CỤ THIẾT KẾ PHẦN MỀM VÀ PHƯƠNG PHÁP THIẾT KẾ CƠ BẢN

- ✦ Tổng hợp hoạt động xây dựng
- ✦ Phân chia *module* hiệu quả
- ✦ Thiết kế dữ liệu, xây dựng, thuật toán, giao diện



NOI DUNG

6.1. Các cơ sở của thiết kế phần mềm

6.1.1. Trừu tượng hoá (*abstraction*)

6.1.2. Tinh chế (*refinement*)

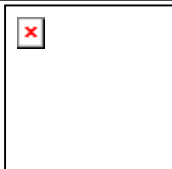
6.1.3. Phân chia *module* (*modularity*)

6.1.4. Kiến trúc phần mềm

6.1.5. Cấu trúc dữ liệu

6.1.6. Thuật toán

6.1.7. Chứng minh tính đúng đắn



NOI DUNG (t.t)

6.2. Phân chia module hiểu qua

6.2.1. Nội kết dính (*cohesion*)

6.2.2. Sự liên kết (*coupling*)

6.2.3. Các *heuristics* cho phân chia module

6.3. Thiết kế dữ liệu

6.4. Thiết kế kiến trúc

6.4.1. Dùng *transform* và dùng *transaction*

6.4.2. Ảnh hưởng dùng *transform*

6.4.3. Ảnh hưởng dùng *transaction*

6.5. Thiết kế giao diện người dùng

6.6. Thiết kế thuật toán



GIỚI THIỆU



- Thiết kế phần mềm là công việc đầu tiên của giai đoạn phát triển
- Thiết kế tạo ra các biểu diễn và điều kiện của hệ thống phần mềm cần xây dựng từ kết quả phân tích yêu cầu nếu có thể đang hiện thời sau này
- Là lĩnh vực tổng hợp mọi phương pháp phát triển với nhiều phương pháp khác nhau

TRÒU TÖÔNG HOÀU

- Quá trình thiết kế trải qua nhiều mức tròu tồông hoàikhác nhau
 - ◆ Mức cao nhất: vấn ãnềcần thiết kế ãnữộc mô ãt ãmôt cách tổng quát sôũ dụng thuật ngữ ãhồông vấn ãnề
 - ◆ Các mức thấp hơn: ãhồông ãn thu ãtức xôũ ãi chi tiết; kết hợp các thuật ngữ ãhồông ãn hiển thốc
 - ◆ Mức thấp nhất: vấn ãn ãnữộc mô ãt ã theo cách cõ ãh hiển thốc trực tiếp
- Phân loại tròu tồông hoài thu ãtức và ã dữ liệu



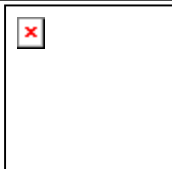
TRÒU TÖÔNG HOÀ(t.t)

- Tròu töông hoàthuât

- ◆ Làchuỗi các lệnh liên tiếp thực hiện chớ năng nào ñó
- ◆ Ví dụ: môicô (bao gồm ñi ñến cô, cầm lấy tay ñám, xoay tay ñám, kéo cánh cô, ñi vào...); thêm một phần tửvào danh sách còthòitôi (xác ñịnh vò trí, chen phần tửmôi)

- Tròu töông hoàdồlieu

- ◆ Làtổhợp đồlieu màtâm một ñó töông đồlieu (liên hệtôi ñó töông thực theãtrong UML). Ví dụ: hàng, chòng, cánh cô...
- ◆ Một sốngôn ngữlập trình hoãtrở kiểu ADT vàtemplate



TINH CHEÁ

- Tinh chế là quá trình làm rõ vấn đề
- Tinh chế và trừu tượng hoá là hai khái niệm bù trừ nhau: càng tinh chế thì càng hạ thấp mức trừu tượng hoá



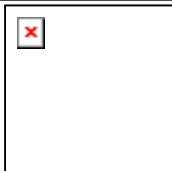
PHÂN CHIA *MODULE*

- Khái niệm *module* đã xuất hiện khoảng 4 thập niên trước đây
- Phần mềm được xây dựng bằng cách phân chia thành nhiều *module*, sau đó sẽ được tích hợp lại
- Phân chia *module* làm cho việc quản lý phần mềm khoa học hơn
- Giả sử $C(x)$: độ phức tạp của x , $E(x)$: công sức cần thực hiện x .

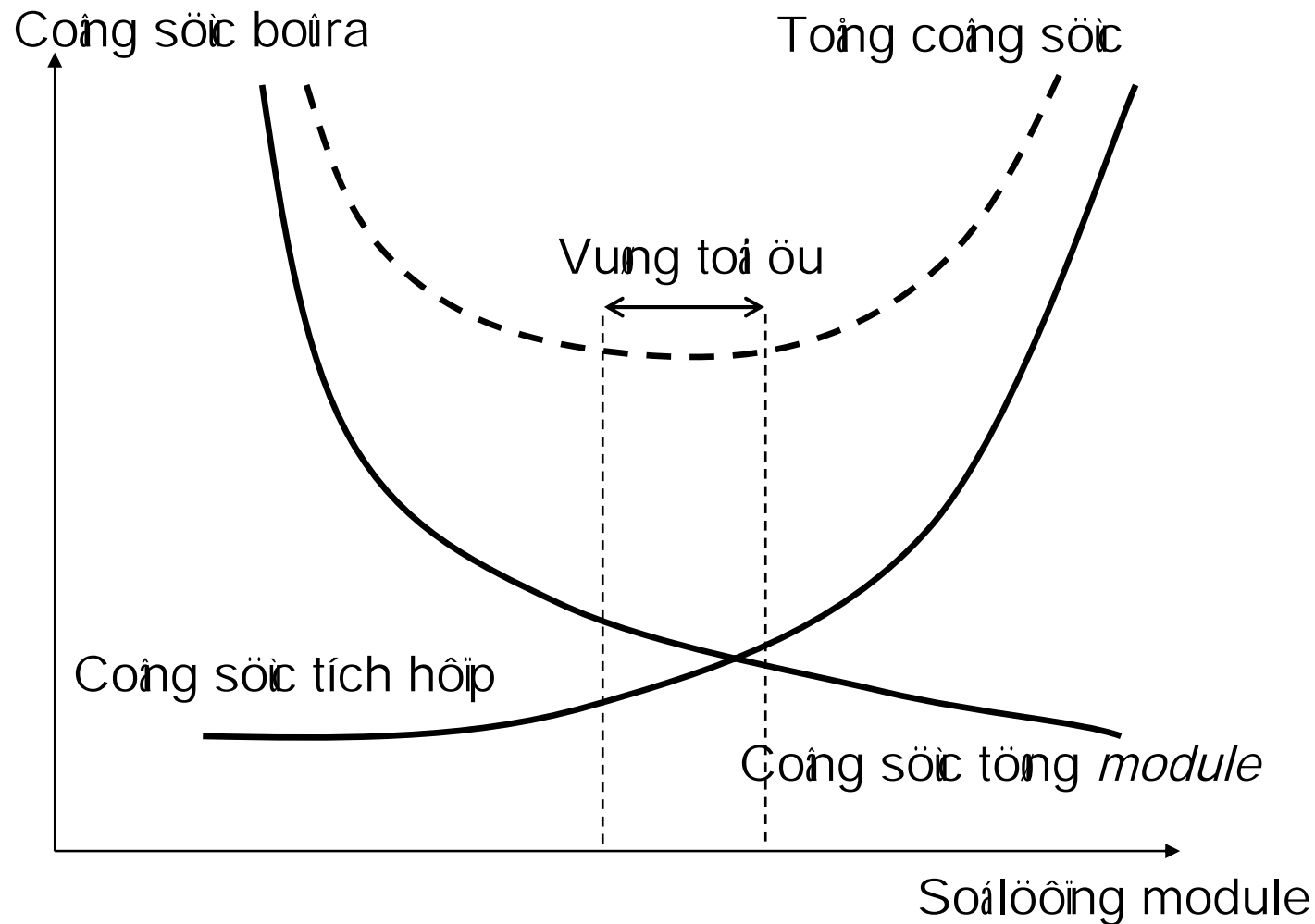
Rõ ràng: nếu $C(p1) > C(p2)$ thì $E(p1) > E(p2)$.

- Nếu phân chia $p = p1 + p2$ ta thấy (một cách trực quan):

$$C(p1 + p2) > C(p1) + C(p2) \Rightarrow E(p1 + p2) > E(p1) + E(p2)$$



PHÂN CHIA *MODULE* (t.t)

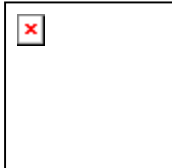


● Soá lööng *module* phụ thuộc vào ñoãphòic tập của hệthống phần mềm cần xây dựng \Rightarrow quá ít hoặc quá nhiều *module* ñều không tốt

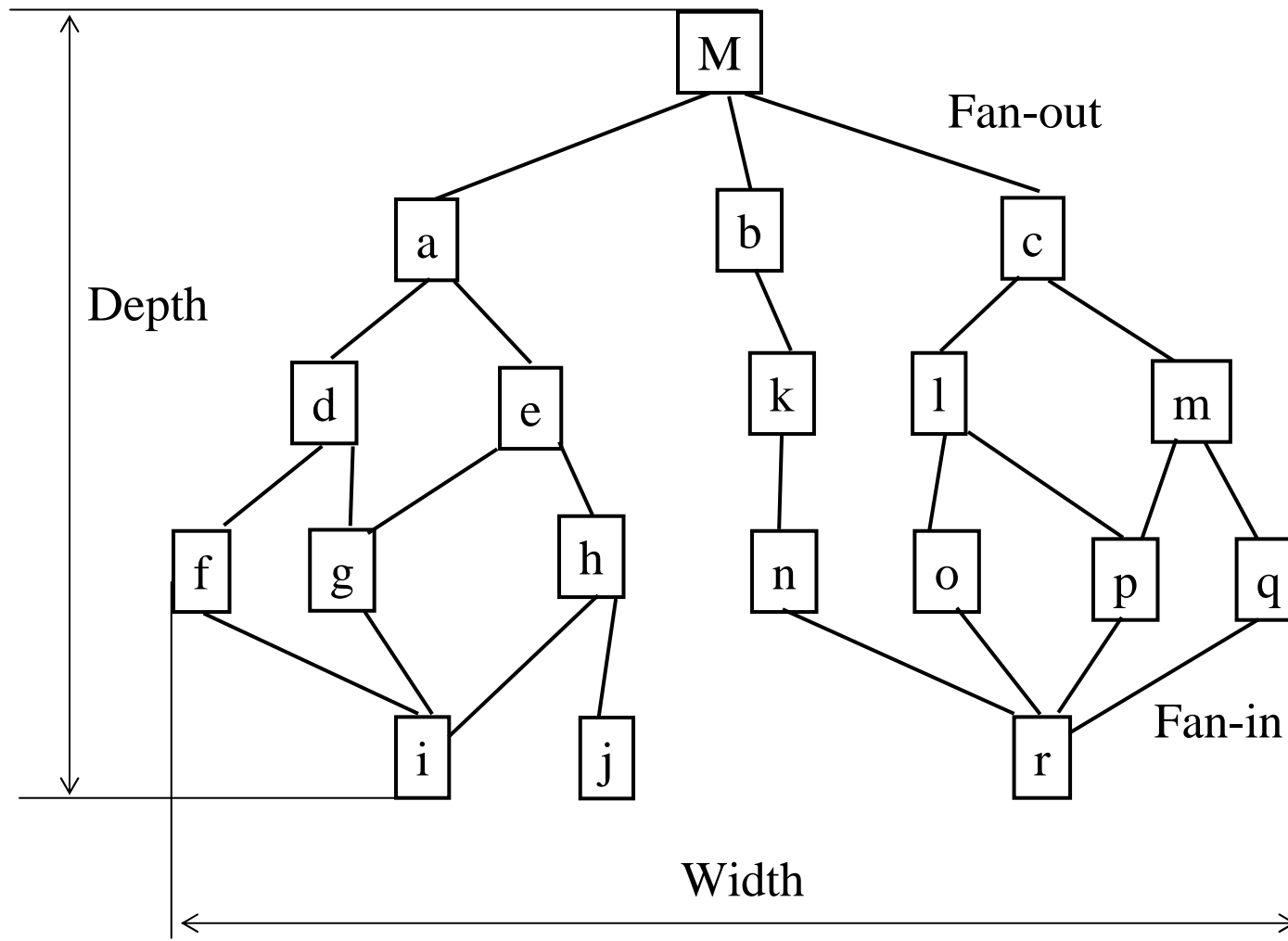


KIẾN TRÚC PHẦN MỀM

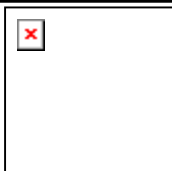
- Kiến trúc phần mềm mô tả các thành phần (*component*) kiến tạo nên hệ thống phần mềm và mối giao tiếp giữa các thành phần đó
- Thành phần có thể là
 - ◆ Các *module* mã nguồn
 - ◆ Các file thực thi (*.dll, *.exe, *.class...)
 - ◆ Các thành phần của kiến trúc hệ thống: ActiveX *control*, *bean*...
 - ◆ Các trang HTML, *.asp, *.jsp...



KIẾN TRÚC PHẦN MỀM (t.t)

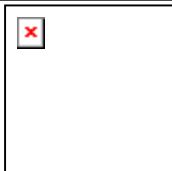


● Số node
phân cấp
nội dung
mô tả
số phân ra
các module.



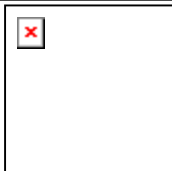
CAU TRUC DÖÖLIEÜ

- Cấu trúc dữ liệu mô tả sự tổ chức, phương thức truy xuất, mối liên kết và các xử lý khác của thông tin
- Dữ liệu tồn tại dưới cấu trúc dữ liệu tồn tại nhất chẳng bao gồm một phần tử thông tin mà có thể được truy xuất bằng một danh sách
- Một số dạng phổ biến: *vector*, ma trận, mảng nhiều chiều, danh sách liên kết, hàng, cột, cây nhị phân...
- Các biểu diễn các mối liên hệ khác nhau



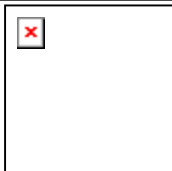
THUẬT THUẬT

- Thuật tập trung vào chi tiết xử lý của mỗi *module*.
- Cung cấp các tài chi tiết của
 - ◆ Chuỗi sự kiện
 - ◆ Vòng lặp
 - ◆ Quyết định rẽ nhánh
 - ◆ Cấu trúc dữ liệu



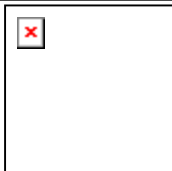
CHE DAU THÔNG TIN

- Che dấu thông tin là một trong những nguyên lý quan trọng của việc phân chia *module*
- Các *module* giao tiếp với nhau bằng những thông tin thật sơ cần thiết
- Những thông tin về thuật toán và dữ liệu cục bộ của mỗi *module* phải được che dấu khỏi các *module* khác
- Lợi ích: kiểm soát được thay đổi và sửa lỗi dễ dàng



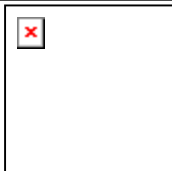
PHÂN CHIA *MODULE* HIỆU QUẢ

- Phân chia *module* là bước trong giai đoạn thiết kế
- Tuy nhiên: phân chia kiến trúc phần mềm thành một bộ các *module* nhỏ thế nào là tốt nhất ?
- Tiêu chí quan trọng nhất: tính độc lập chức năng của các *module*
- Tính độc lập chức năng được đo bằng 2 tiêu chuẩn: nội kết dính (*cohesion*) và sự liên kết (*coupling*)



ÑOÄKEÁT DÍNH

- Ñoäkeát dínđ dùng ñeãño söi phuï thuôc lẫn nhau giöa nhöng tàic vui (*task*) của một *module*
- *Module* có ñoäkeát dínđ cao nhất khi nó chæ ñaím nhận ñuöng một tàic vui \Rightarrow keát dínđ chöïc năng
- Thiét keákieán trüic phần mềm: có gáng tăng ñoäkeát dínđ



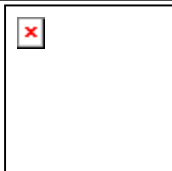
ỔNG KẾT ĐỊNH (t.t)

- Có nhiều mức ổn kết định (từ thấp đến cao)
 - ◆ ngẫu nhiên: các tác vụ không liên hệ với nhau
 - ◆ luận lý các tác vụ liên quan logic với nhau
 - ◆ nhất thời: các tác vụ phải thực thi trong một khoảng thời gian
 - ◆ giao tiếp: các tác vụ sử dụng chung một dữ liệu nào đó
 - ◆ tuần tự: các tác vụ phải thực hiện theo một trật tự nhất định
 - ◆ chờ đợi: chờ một tác vụ



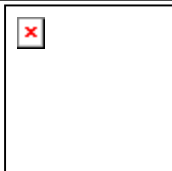
SÖI LIÊN KẾT

- Söi liên kết dùng ñể ñối ñối quá trình giao tiếp giữa các *module*:
giao tiếp của *module* chöa nhiều tài liệu và nhiều thông số gởi thì
söi liên kết càng cao
- Thiết kế kiến trúc phần mềm: có gắng giảm söi liên kết



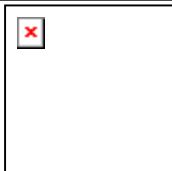
SƠ LƯỢC (t.t)

- Có nhiều mức độ liên kết (từ cao đến thấp)
 - ◆ liên kết nội dung: sử dụng dữ liệu và điều khiển của *module* khác
 - ◆ liên kết chung: có sử dụng chung dữ liệu toàn cục
 - ◆ liên kết ngoại vi: *module* phụ thuộc vào một I/O nào đó
 - ◆ liên kết điều khiển: thông số truyền ảnh hưởng đến điều khiển
 - ◆ liên kết *stamp*: truyền cấu trúc dữ liệu phức tạp
 - ◆ liên kết dữ liệu: truyền các thông số đơn giản



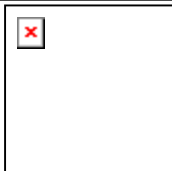
CÁC HEURISTICS cho PHÂN CHIA MODULE

- Sửa lại thiết kế ban đầu nếu cần thiết để giảm số liên kết
- Khi chiều sâu tăng, hạn chế *fan-out* trong khi sử dụng *fan-in*
- Giảm thiểu ảnh hưởng của một *module* nằm bên trong tầm nhìn khiến của nó
- Loại bỏ độ trễ trong giao tiếp của các *module*
- Ưu tiên các *module* tại rìa, hạn chế các *module* nhiều ràng buộc
- Nâng gởi các *module* nếu cần tính khả thi chuyển (portability)



THIẾT KẾ DỮ LIỆU

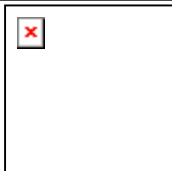
- Tìm kiếm biểu diễn luận lý cho các phần tử dữ liệu nhằm nhận diện trong giai đoạn phân tích yêu cầu
- Thiết kế các cấu trúc dữ liệu của chương trình và cơ sở dữ liệu
- Thực hiện tinh chế tổng thể



THIẾT KẾ DỮ LIỆU (t.t)

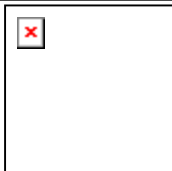
- Một số nguyên tắc

- ◆ Nhận diện cấu trúc dữ liệu và tác vụ truy xuất
- ◆ Xây dựng tổng thể dữ liệu
- ◆ Trì hoãn thiết kế dữ liệu một thập cho đến cuối giai đoạn này
- ◆ Che dấu biểu diễn bên trong của cấu trúc dữ liệu
- ◆ Phát triển một thư viện các cấu trúc dữ liệu + tác vụ thông gặp
- ◆ Nên áp dụng kiểu ADT trong thiết kế cũng như trong lập trình

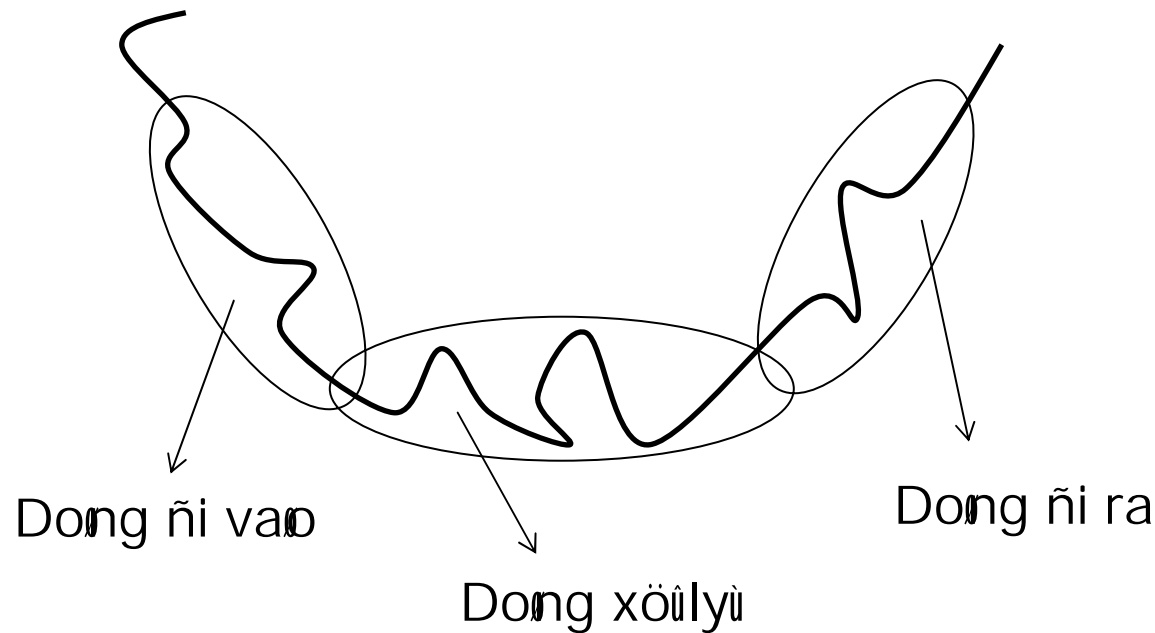


THIẾT KẾ KIẾN TRÚC

- Mục tiêu là xây dựng số lượng phân cấp *module* từ DFD
- Nâng nền móng để thiết kế chi tiết thuật toán và dữ liệu
- Phân biệt dòng *transform* và dòng *transaction* trong DFD
- Thực hiện ảnh xạ cho từng vùng của DFD tùy theo nội dung *transform* hay *transaction*



DONG TRANSFORM VÀ TRANSACTION

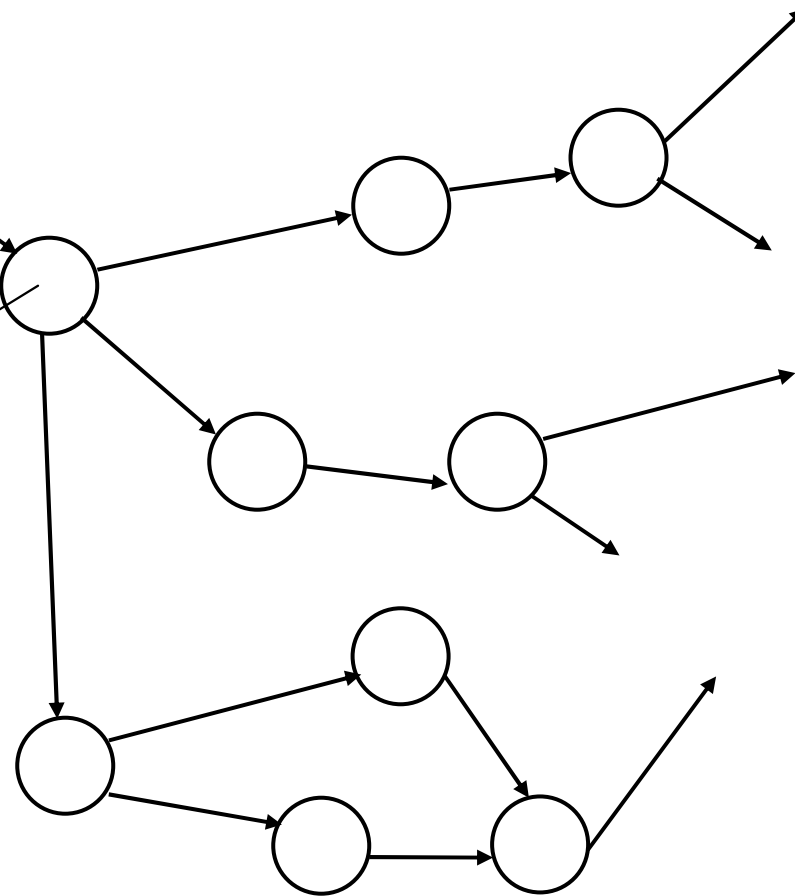


- Dong *transform* bao gồm 3 phần: dong ñi vào, dong xöülyù và dong ñi ra

DONG TRANSACTION VÀ TRANSACTION (t.t)

Dong ñi vào

T-center



- Dong *transaction* bao gồm: dong ñi vào, T-center và các ñöông xöù lyù ñaàu ra

- T-center: Chæ coù một ñöông ra ñöôc kích hoạt tại một thời ñiểm



ẢNH XÃI DONG *TRANSFORM*

- Tối ưu [1], trang 377



- Trang 163 -

Khoa Công Nghệ Thông Tin - Môn Công Nghệ Phần Mềm -
Chương 6: Cơ sở của thiết kế phần mềm và phương pháp thiết kế cấu trúc

ẢNH XÃI DONG TRANSACTION

- Tồi ão [1], trang 387

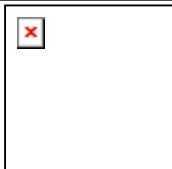


- Trang 164 -

Khoa Công Nghệ Thông Tin - Môn Công Nghệ Phần Mềm -
Chương 6: Cơ sở của thiết kế phần mềm và phương pháp thiết kế cấu trúc

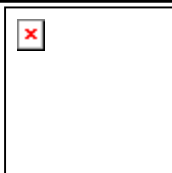
THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

- Phần mềm cần có giao diện thân thiện với người sử dụng
- Một số tiêu chuẩn giao diện
 - ◆ Thời gian nạp ồng của hệ thống: giảm trung bình và nhiễu
 - ◆ Phương tiện trợ giúp người sử dụng: tích hợp + *add-on*
 - ◆ Kiểm soát thông tin lỗi: hiển thị nguyên nhân lỗi và cách khắc phục
 - ◆ Đặt tên nhân: ngắn gọn và rõ ràng



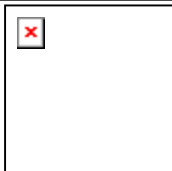
THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG (t.t)

- Công cụ thiết kế giao diện nên có những tính năng sau
 - ◆ Quản lý thiết bị nhập (bàn phím, chuột)
 - ◆ Hiệu chỉnh thông tin input
 - ◆ Kiểm soát lỗi và hiển thị thông báo lỗi
 - ◆ Cung cấp trợ giúp và hiển thị thông báo nhắc nhở
 - ◆ Cung cấp *feedback* (ví dụ nhớ từ nào đã hiển thị ký tự nào hình vẽ)
 - ◆ Kiểm soát cửa sổ và vùng, khả năng cuộn
 - ◆ Thiết lập giao tiếp giữa chương trình với giao diện (vd: hàm nhập ờng)
 - ◆ Cách ly chương trình với các hàm quản lý giao diện
 - ◆ Cho phép tùy biến giao diện



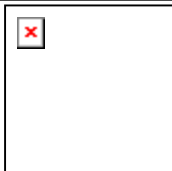
THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG (t.t)

- Một số hướng dẫn chung
 - ◆ Nên dùng nhất (menu, lệnh, hiển thị...)
 - ◆ Nên cung cấp *feedback* cho người dùng
 - ◆ Yêu cầu xác nhận những tác vụ mang tính phá hủy (xóa file, account)
 - ◆ Nên hỗ trợ UNDO, REDO
 - ◆ Hạn chế thông tin phải ghi nhớ giữa 2 tác vụ liên tiếp
 - ◆ Tối ưu trong trình bày hộp thoại và di chuyển *mouse*
 - ◆ Chấp nhận lối tắt phím người sử dụng
 - ◆ Cung cấp trợ giúp trực tuyến
 - ◆ Dùng ngôn từ đơn giản và ngắn gọn nhất tên các lệnh



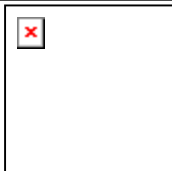
THIẾT KẾ GIAO DIỆN NGỒI DUNG (t.t)

- Nói với thông tin hiển thị
 - ◆ Chọn hiển thị những thông tin phù hợp với ngữ cảnh hiển thị
 - ◆ Dùng tên, từ viết tắt và màu gợi nhớ
 - ◆ Cho phép tổng tài trực quan
 - ◆ Tạo thông báo lỗi rõ ràng
 - ◆ Hiển thị dữ liệu ở nhiều dạng khác nhau trong cửa sổ
 - ◆ Thiết lập biểu diễn tổng thể
 - ◆ Sử dụng không gian màn hình một cách tối ưu



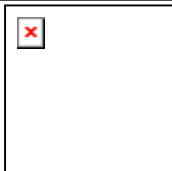
THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG (t.t)

- Nói với thông tin *input*
 - ◆ Hạn chế *input* trực tiếp (có thể chọn lựa từ một số dữ liệu có sẵn)
 - ◆ Nên đồng nhất giữa thông tin *input* và hiển thị
 - ◆ Nên cho phép tùy biến *input*
 - ◆ Cảnh cáo chức năng không thích hợp trong giao diện hiển thị
 - ◆ Cho phép *input* nhiều dạng khác nhau
 - ◆ Nên cho người sử dụng kiểm soát dòng sự kiện tổng thể
 - ◆ Tối ưu tính các giao trò *input* cho người sử dụng nếu có thể



THIẾT KẾ THUẬT

- Thiết lập thuật giải cho các *module* nào kiến tạo sao cho có thể dễ dàng mã hóa bằng ngôn ngữ lập trình có cấu trúc
- Có thể biểu diễn thuật giải bằng
 - ◆ Lưu nội thuật giải: note [1], trang 407
 - ◆ Ký hiệu dạng bảng : note [1], trang 409
 - ◆ Ngôn ngữ PDL



NGÔN NGỮ PDL

- Ngôn ngữ PDL vay mượn từ vựng của ngôn ngữ tự nhiên và cú pháp của ngôn ngữ lập trình cấu trúc. Nó có các tính chất sau:
 - ◆ Cú pháp chặt chẽ của các toán tử hoặc toán tử cấu trúc, khai báo dữ liệu, phân chia *module*
 - ◆ Cú pháp tối đa của ngôn ngữ tự nhiên giúp miêu tả xử lý
 - ◆ Phương tiện mô tả dữ liệu nên cũng nhớ dữ liệu tổ hợp
 - ◆ Có chế độ nhúng chương trình con và phương cách gọi



NGÔN NGÖPDL (t.t)

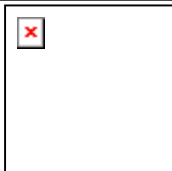
```
procedure AnalyzeTriangle( a, b, c: in real; type: out string)
begin
    sort a, b, c so that a >= b >= c;
    if ( c > 0 and a < b + c )
        if ( a = c )
            type := "Equilateral"
        else
            if ( a = b or b = c )
                type := "Isosceles"
            else
                if ( a*a = b*b + c*c )
                    type := "Right"
                else
                    type := "Scalene"
            end if
        end if
    else
        type := "Error"
    end if
end
```



TỔNG KẾT



- Các công cụ của thiết kế phần mềm: trừu tượng hoá tính che giấu tổng thể, phân chia *module*, cấu trúc dữ liệu, chương trình con, che dấu thông tin
- Phân chia *module* hiệu quả tăng độ kết dính và giảm số liên kết
- Thiết kế cơ thể phần mềm bao gồm 4 công đoạn: t/k dữ liệu, t/k kiến trúc, t/k giao diện người máy và t/k thuật toán



Chương 7

THIẾT KẾ HỒÔNG NỘI TỒÔNG

- ◆ Hạnh vi của nội tồông
- ◆ Hoàn chỉnh các tài liệu

NOI DUNG

7.1. Thiết kế giao diện

- 7.1.1. Khái niệm mô hình nền
- 7.1.2. Tổng tài giữa các nội tổng
- 7.1.3. Sợ cộng tài (*collaboration*)
- 7.1.4. Miêu tài trình tài
- 7.1.5. Lồôc nồatrăng thầi (*statechart diagram*)
- 7.1.6. Lồôc nồahoạt nồôg (*activity diagram*)

7.2. Hoàn chỉnh nắ tài tính

- 7.2.1. Nhắn diển thếm mắ sốlồp thiết kế
- 7.2.2. Nắ tài chi tiế các thuốc tính
- 7.2.3. Nhắn diển chính xắ các tài vui
- 7.2.4. Hoàn chỉnh lồôc nồalồp



GIỚI THIỆU

- Giai đoạn thiết kế quan tâm đến “HOW”:

- ◆ Thời điểm các thông điệp trao đổi, thông số của thông điệp
- ◆ Thuật giải của các vụ nạp òng
- ◆ Cấu trúc dữ liệu cho các thuật tính
- ◆ *Framework* (*console, document/view, 3-tier...*)

- Thiết kế cũng chú ý đến môi trường

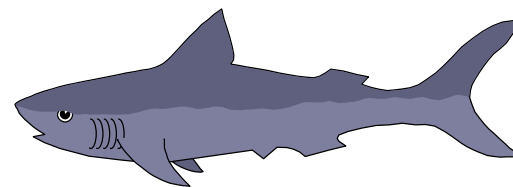
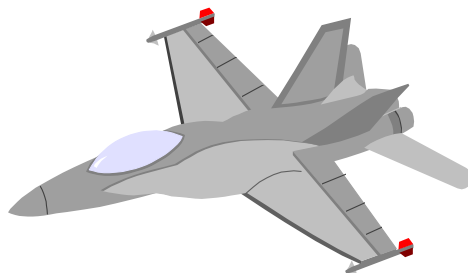
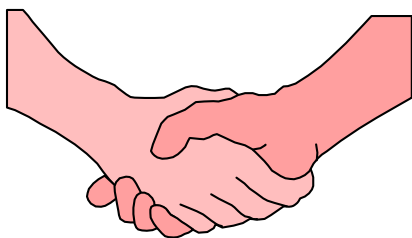
- ◆ Ngôn ngữ lập trình và thư viện lập trình (Hoặc Vector, List, Map... hay không? Hoặc *template* hay không?...)
- ◆ Kiến trúc hệ thống (COM, CORBA hay EJB)

⇒ Thiết lập mô hình động (*dynamic modeling*) và chi tiết hóa mô hình tĩnh



KHAI NIỆM MÔ HÌNH ÑỒNG

- Lööc ñoà lôp chæ mô ta ù khía cãnh tñnh củ a heã thoãg
- Hãnh vi củ a heã thoãg ñoõc mô ta ù bằg mô hình ñoõg bao gồm
 - ◆ Tõõg tãc giõa cãc ñoã tõõg: cõõg tãc hay trìn tõi
 - ◆ Trãõg thãi củ a ñoã tõõg/lõp
 - ◆ Quã trìn hoãt ñoõg củ a lôp/ñoã tõõg



TÔNG TÁC GIỮA CÁC NƠI TÔNG

- Nơi tông tông tác với nhau (*interaction*) bằng cách gửi/nhận kích thích (*stimulus*)
- *Actor* cũng có thể gửi kích thích đến nơi tông
- Kích thích khiến một tác vụ thực thi, một nơi tông nào đó tạo ra hay hủy, hoặc gây ra một tín hiệu
- Thông điệp (*message*) là tác vụ của kích thích



TỔNG TÀI GIỮA CÁC NỘI TỔNG (t.t)

● Các loại thông tin

◆ Nội gian



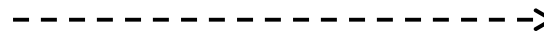
◆ Nội bộ



◆ Bất nội bộ



◆ Trao đổi với giới hạn



SÔI CÔNG TÁC

- Công tác (*collaboration*) là hình thức tập hợp các thành phần tham gia vào quan hệ giao tiếp chung
- Các thành phần tham gia là vai trò mà mỗi tổng/lớp đóng vai khi tương tác với nhau
- Các vai trò của mỗi tổng thường chia thành hóa mỗi với một mức nhất nào đó
- Loại sơ đồ công tác (*collaboration diagram*) được thiết lập để biểu diễn một *use-case* hoặc một tác vụ



SƠ ĐỒ CÔNG TÁC (t.t)

- Lược đồ công tác là một đồ liên kết các vai trò
- Quan hệ liên kết được dùng để kết nối các vai trò với nhau
- Có thể chia ra tên vai trò cho các liên kết
- Thông tác được thể hiện bằng gối/nhấn thông điệp
- Mỗi thông điệp được thể hiện bằng mũi tên (nhớ nhớ miêu tả) công với phần này tại



SỒI CÔNG TÁC (t.t)

- Các thông lệ nội hành số theo kiểu phân cấp

- ◆ 3.4.2 xây ra sau 3.4.1 và cả hai nội lồng (*nested*) trong 3.4

- ◆ 3.4.3a và 3.4.3b xây ra nội lồng và nội lồng trong 3.4

- Cấu pháp tổng quát của thông lệ

predecessor guard-condition sequence-expression return-value := message-name argument-list

- Ví dụ: 2/ 1.3.1: p := find(specs)

1.1, 4.2/ 3.2 *[i:=1..6]: invert(x, color)



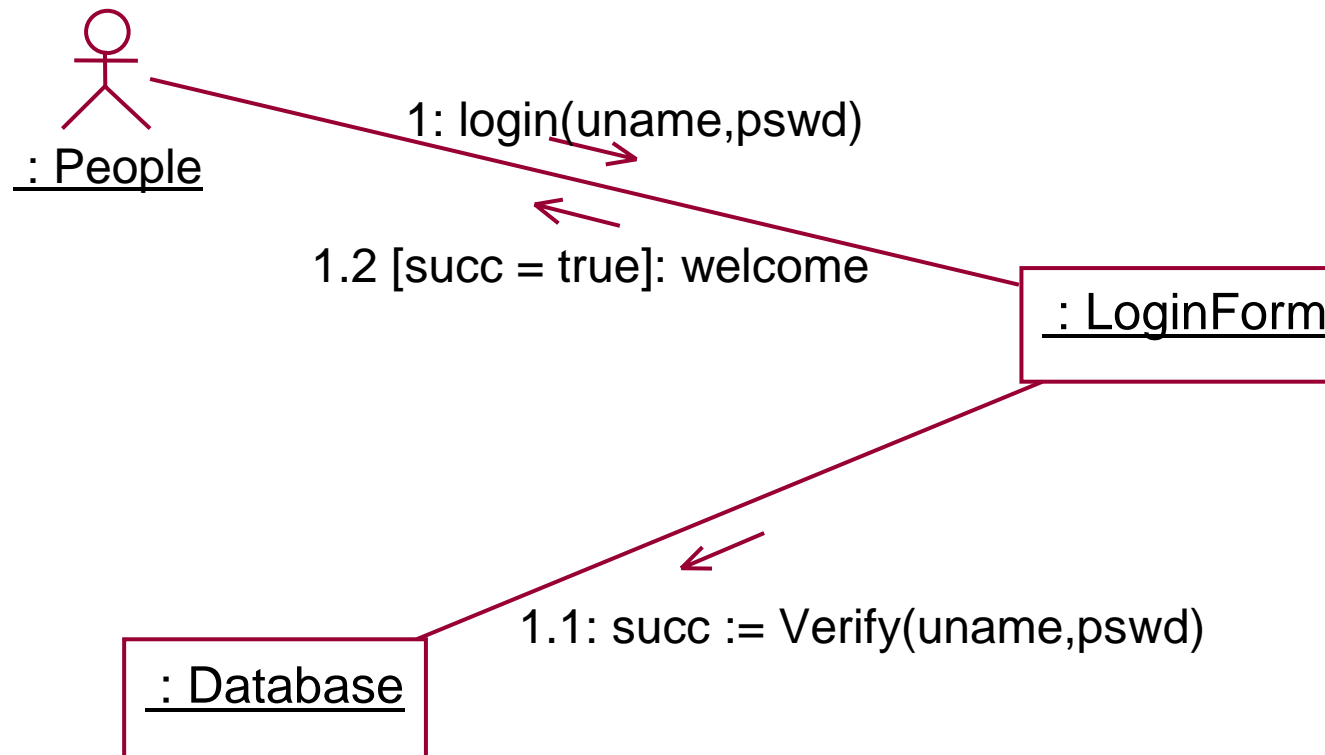
SƠ CÔNG TÁC (t.t)

- Lược đồ công tác có thể thiết lập ở một trong 2 dạng:
 - ◆ Dạng cui thể mỗi vai trò được biểu diễn bằng một ký hiệu của nó tổng cui thể các thông nghiệp được trao đổi trên các dòng liên kết
 - ◆ Dạng này tại mô tả các lớp; các dòng liên kết được ảnh hưởng vào các thông nghiệp
- Thiết lập lược đồ công tác giúp cui thể hoá (*realize*) các *use-case* và hiển diễn thêm một số tài vụ của các mối tổng/lớp phân tích

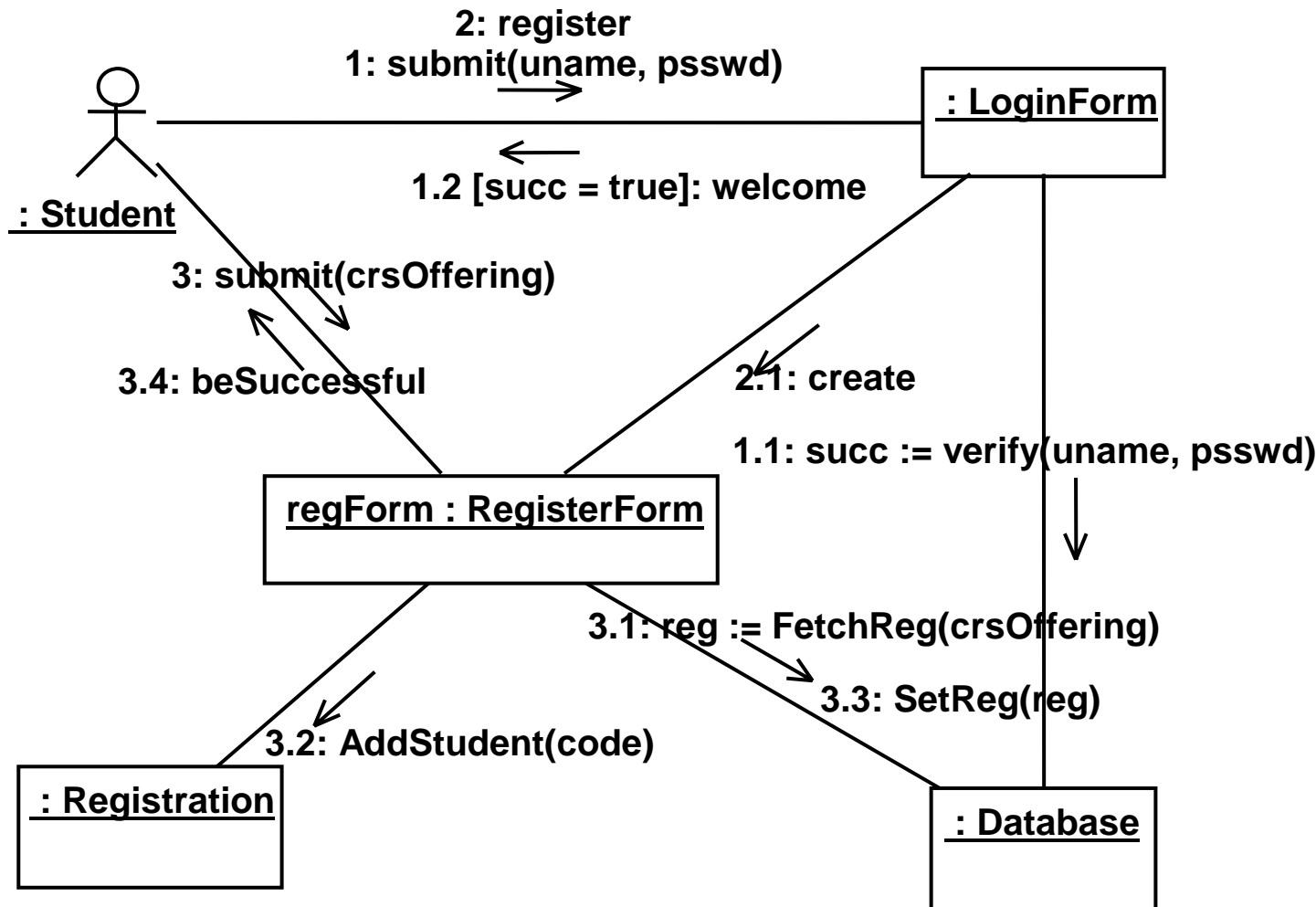


SỒI CÔNG TÁC (t.t)

- Ví dụ: lược đồ công tác mô tả cho *use-case* Login của hệ thống năng kỳ môi học tin cậy qua WEB



SỒI CÔNG TÁC (t.t)



- Ví dụ: lờc ñoà công tác mõi cũi the cho *use-case* Registers course của hệ thống ñang ky ñơn hoặ tín chæ qua WEB

MIEU TAÙTRÌNH TÖI

- Lööc ñoàcoäng taù mieu taùsöitöông taù theo khía cảnh không gian
- Ñeânhañ mañh trình töi của töông taù \Rightarrow dung lööc ñoàtuañ töi
(*sequence diagram*)
- Lööc ñoàtuañ töi mieu taùcaùc ñoá töông töông taù với nhau theo
thoí gian söng của ñoà
- Caùc thoäng ñieäp ñöôc trao ñoá theo trình töi thoí gian
- Caùc möá lieä keä không ñöôc theähieñ trong lööc ñoà



MIEU TAÛTRÌNH TÖI (t.t)

- Lööc ñoàtuàn töi coi2 dăng
 - ◆ Dăng tổng quát: thểhiệncăuvong lập vàreĩnhành
 - ◆ Dăng củitheả mieu taũmôtkịch bản củitheả
- Thôĩ gian sóng củamỗĩ ñoả töĩng ñoũc môataũtheo mộĩ ñoĩng thăng ñoĩng
- Thôĩng thôĩng thôĩ gian troĩ theo chieũ töĩtreĩn xuong dũũ
- Ít khi quan tâm ñeĩn khoảng thôĩ gian, thôĩng chăquan tâm ñeĩn trĩnh töimaũthoĩ



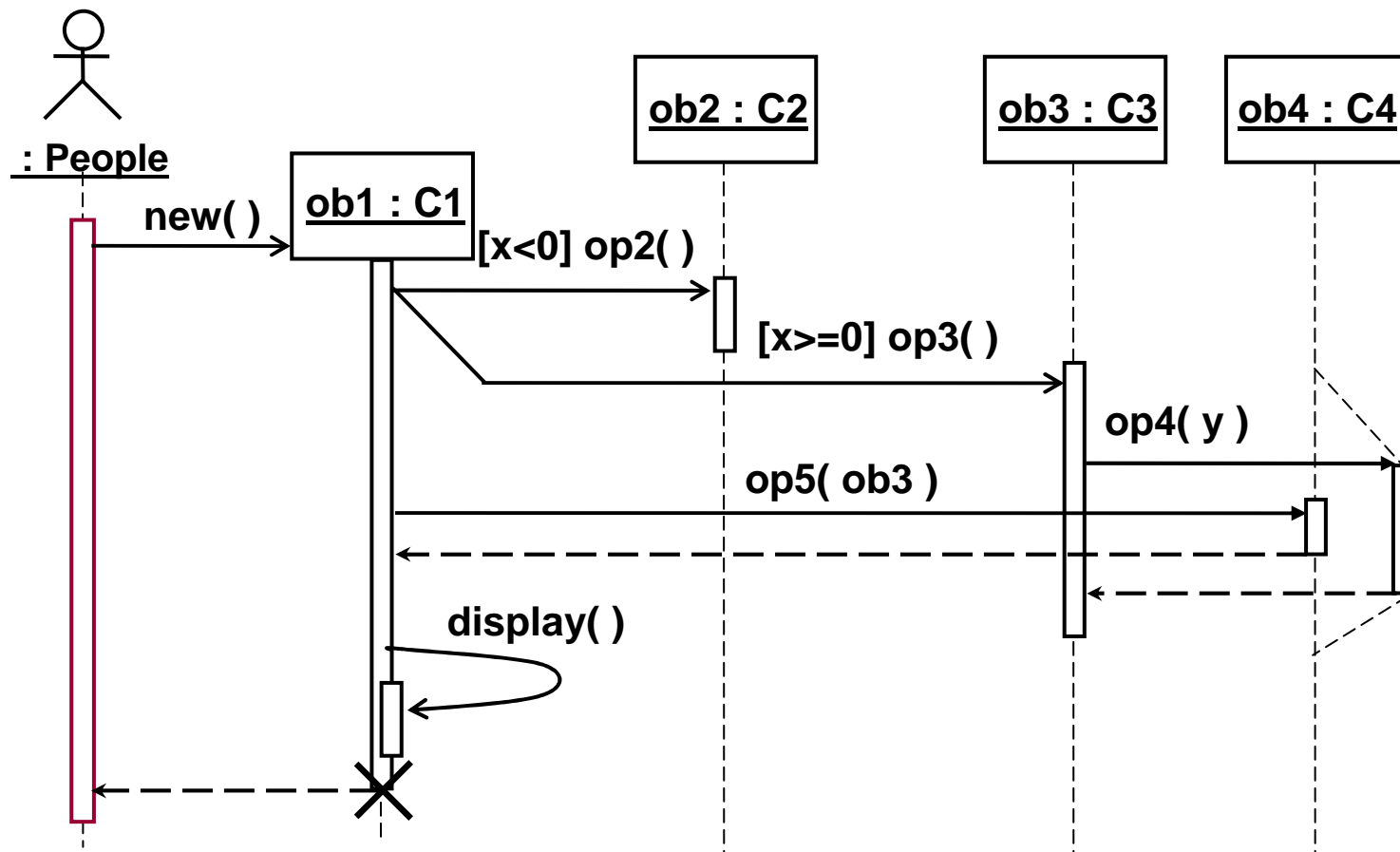
MIEU TAÙTRÌNH TÔI (t.t)

- Thanh hình chổnhat môataũsõithốc thi của một tác vĩ ñeĩ ñap òng
lại thông ñiệp gổĩ ñeĩn
- Ñoĩ ñại của thanh chổnhat phan ảnh thời gian thốc thi của tác vĩ
vũ tính chat lòng nhau (*nested*) giổa chung
- Các dòng *text* phũ trôi (môataũtác vĩ, rang bước thời gian...) ñoĩc
viết ôũleàtraũ



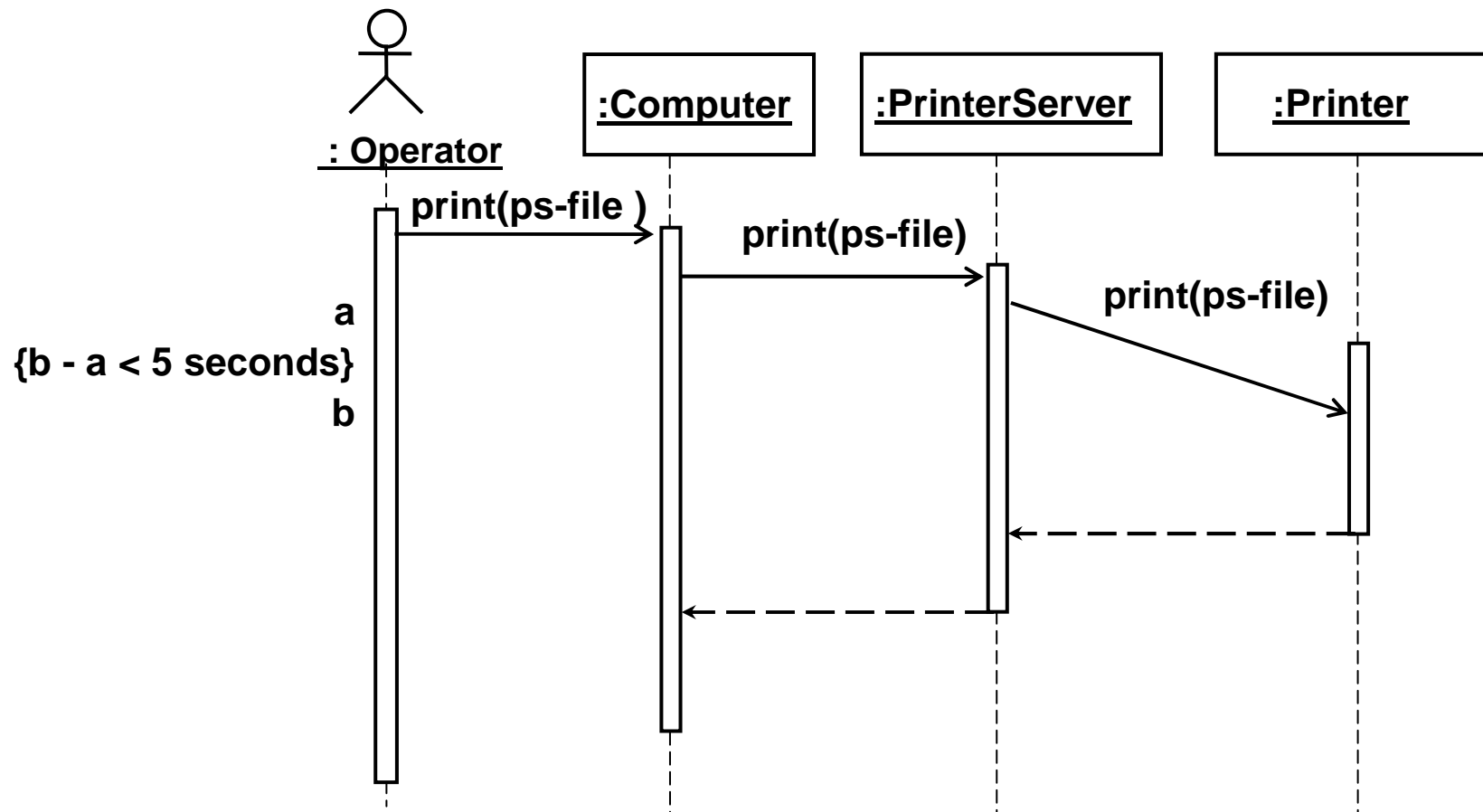
MIEU TAÙTRÌNH TÖI (t.t)

- Ví dụ: l   c n   t   n t   i d   ng t   ng qu   t



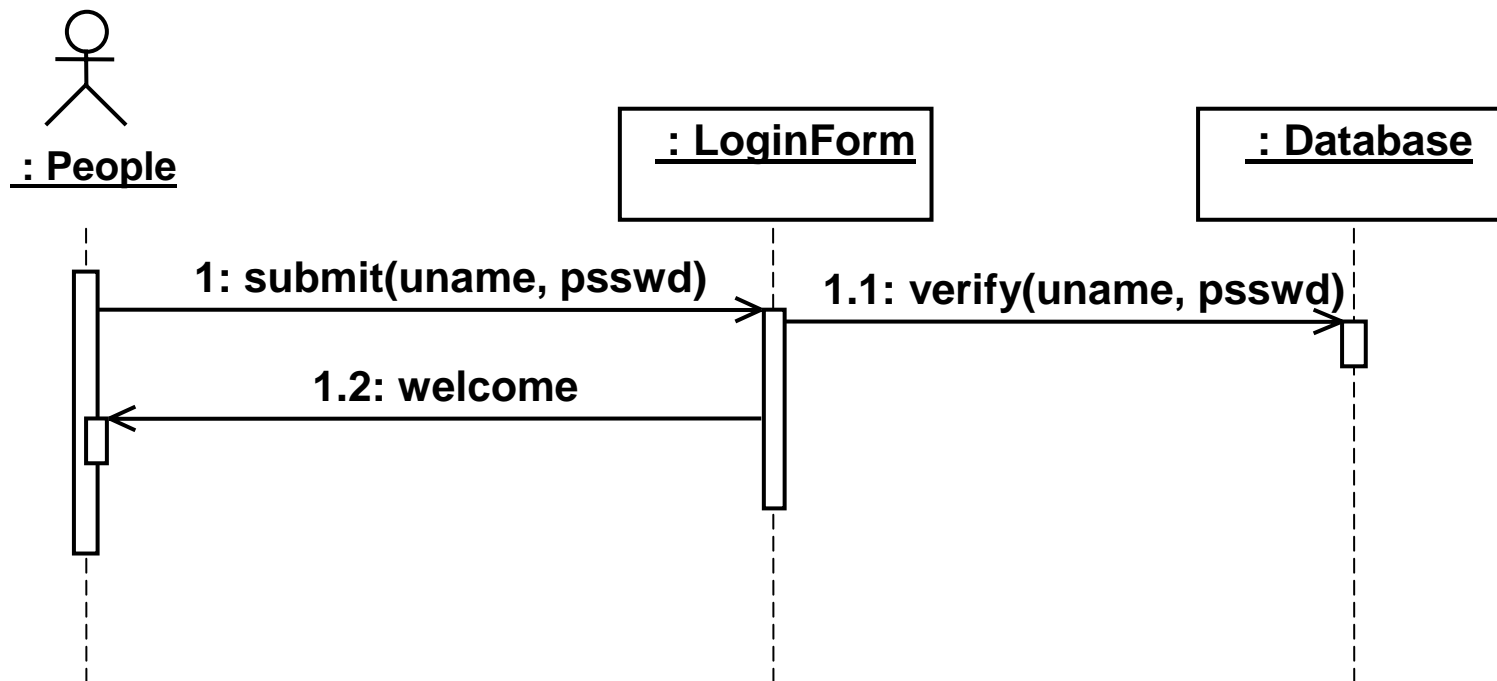
MIEU TAÙTRÌNH TÖI (t.t)

- Ví dụ: l   c nh  at  n t  i d  ng t  ng qu  t

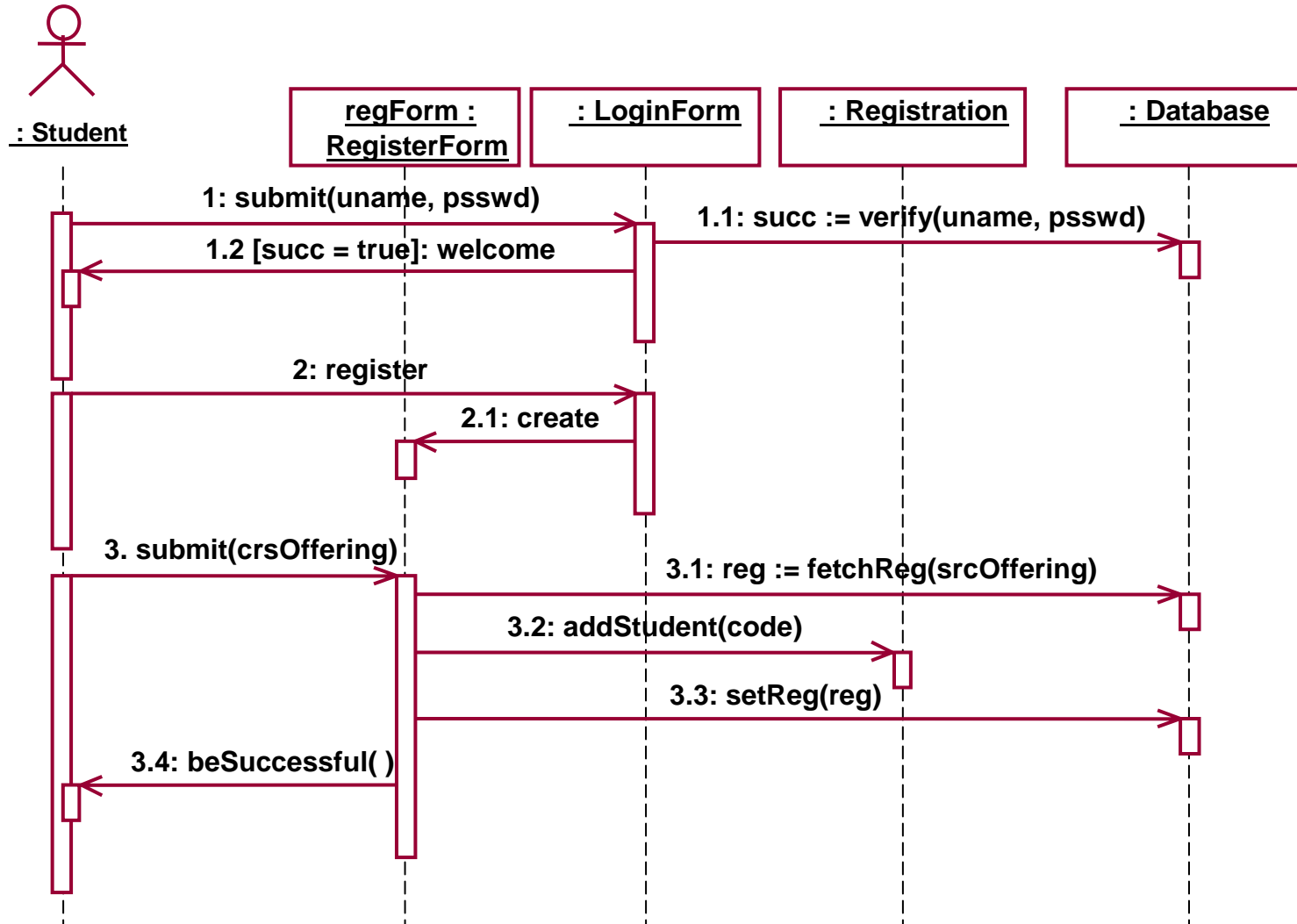


MIEU TAÙTRÌNH TÖI (t.t)

- Ví dụ: l   c n   t   n t   i d   ng c   i th    cho *use-case* Login c   a h    th   ng n   ng ky   m   n ho   c t   n ch    qua WEB



MIEU TAÙTRÌNH TÔI (t.t)



- Ví dụ:
lỗi ñoà
tuàn tôi
đang củ
thể cho
use-case
Register
courses

LÖÖC ÑOÀ TRAİNG THAI

- Chuẩn UML ñưa ra lööc ñoà traing thai ñeà biểu diễn hành vi của một phần tử bất kỳ bằng cách chĩa ra ñập òng của nó ñối với các sự kiện bên ngoài
- Thông thông lööc ñoà traing thai ñược áp dụng cho ñối tượng/lớp
⇒ biểu diễn hành vi của lớp
- Traing thai của mỗi ñối tượng (*ñình nghĩa gốc ?*) ít nhiều sẽ bị thay ñổi trong suốt chu kỳ sống của ñối tượng

LÖÖC ÑOÀ TRĂNG THAI (t.t)

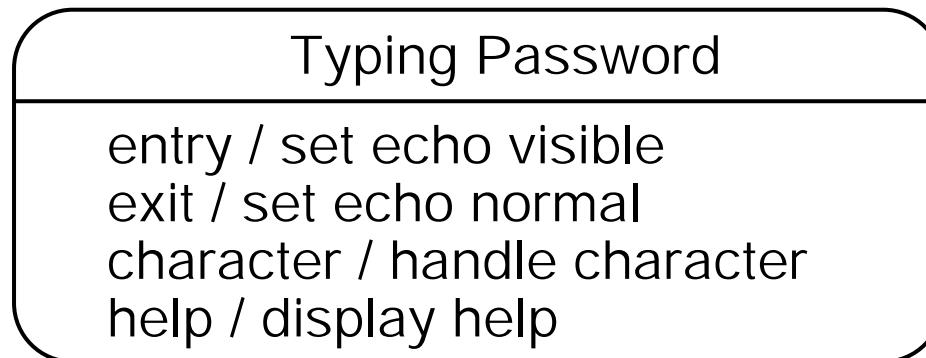
- Trăng thai ñôn giaân laømoät tình trăng trong ñôi sống ñoá töông hoặc moät töông taic của ñoá töông maøtheo ñoùñoá töông thoaũmoät ñieàu kieän, thöïc hieän moät công vieäc hoặc ñôi moät söi kieän naø ñoù
- Thông thông moät ñoá töông naèm ôumöät trăng thai trong moät khoảng thời gian nhất ñònh \Rightarrow ñoùseõdöch chuyeän töøtrăng thai nay sang trăng thai khác
- Trăng thai tổng hõp laøtrăng thai coutheañoöôc phan raõveàcaic trăng thai ñôn giaân



LÖÖC ÑOÀTRÀNG THAI (t.t)

- Trong UML ký hiệu của trạng thái là một hình chữ nhật trong góc vuông có chia làm nhiều phần phân cách nhau bằng các nét ngang:

- ◆ Phần tên
- ◆ Phần miêu tả các hành động bên trong



LÖÖC ÑOÀ TRĂNG THAI (t.t)

- Tên trạng thái là duy nhất trong lược ñoà còitheảkhông còitheả(trạng thái vô danh)
- Các hành ñộng bên trong: các hành ñộng hoặc tài vụ ñể thực hiện khi ñó tổng ñảm ôitrạng thái ñang xét; còicủipháp ñỗ sau
action-label '/' action-expression
- Một số hành ñộng (*action-label*) ñể quy ñể trỏ ñể:
 - ◆ entry: thực hiện hành ñộng tại thời ñiểm bắt ñầu trạng thái
 - ◆ exit: thực hiện hành ñộng tại thời ñiểm kết thúc trạng thái
 - ◆ do: thực hiện hành ñộng suốt trạng thái hoặc cho ñến khi kết thúc ñó
 - ◆ include: triệu gọi một máy trạng thái con khác



LÖÖC ÑOÀTRÀNG THAI (t.t)



- Các nhân hành ñoãg khác chæ ra sõi kiệã kích hoãt hành ñoãg tồõg òõg trong biệũ thồic hành ñoãg (*action-expression*)

- Cùiphãp củã biệũ thồic hành ñoãg

```
event-name '(' parameter-list ')' '[' guard-condition ']'  
'' action-expression
```



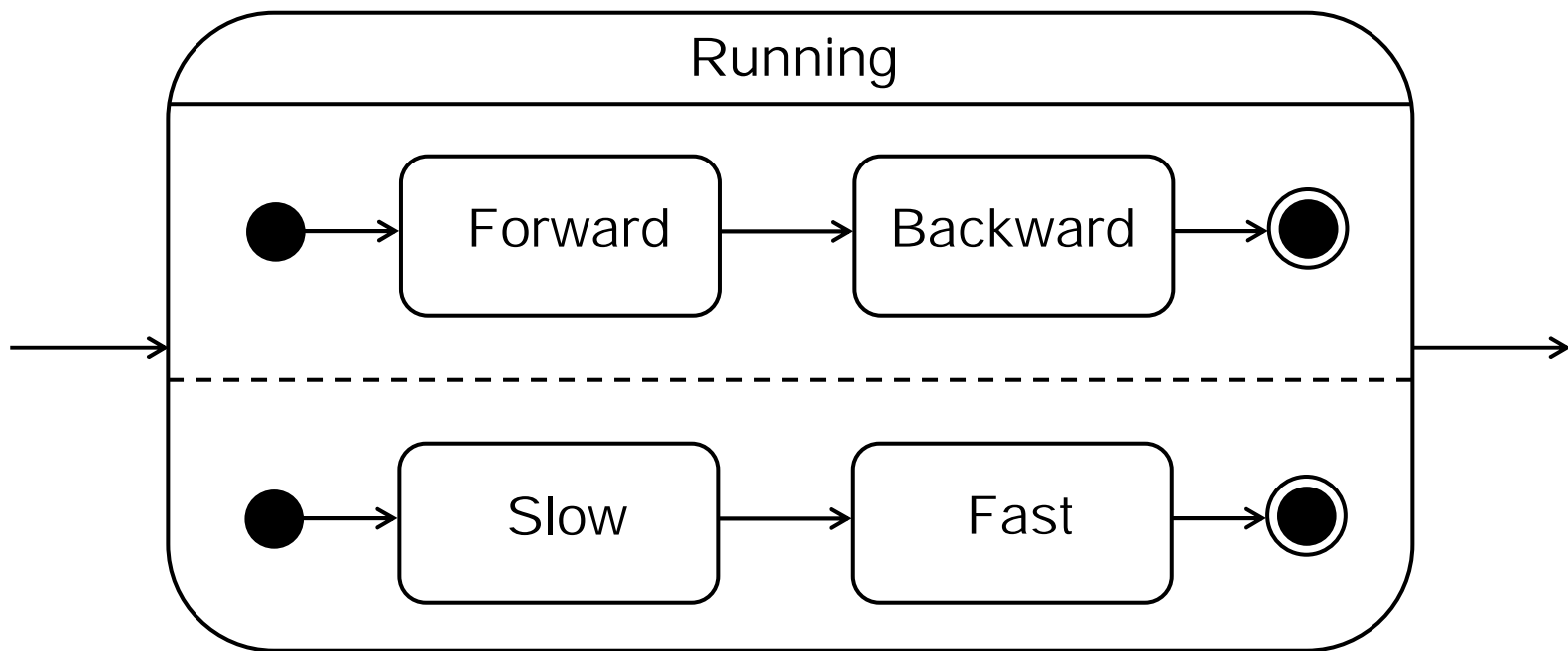
LÖÖC ÑOÀ TRĂNG THAI (t.t)

- Trăng thai bắt màu: khi nối tổng nước tạo ra hoặc trăng thai tổng hợp nước xác định; ký hiệu 
- Trăng thai kết thúc: khi nối tổng bù huyết hoặc trăng thai tổng hợp tròn nên không xác định; ký hiệu 
- Trăng thai tổng hợp (*composite*) nước phân ra thành nhiều trăng thai con nối tiếp hoặc các trăng thai con loại trừ nhau



LÖÖC ÑOÀTRÀNG THAI (t.t)

- Ví dụ: phân rã trạng thái tổng hợp Running



LÖÖC ÑÒÀ TRĂNG THAI (t.t)

- Söi kiện (*event*) kích hoạt dòng chuyển trạng thái, có thể là:
 - ◆ Một hoặc nhiều kiện trùng nhau (chuyển khác với *guard-condition*)
 - ◆ Một hoặc tổng nhận tín hiệu từ một hoặc tổng khác
 - ◆ Một phép gọi tài vụ
 - ◆ Một khoảng thời gian nhất định qua kể từ một sự kiện nào
- Cú pháp của sự kiện: **event-name** ' (' **parameter-list** ') '
- Sự kiện có thể thuộc về *package* của lớp đang mô tả hoặc không thuộc về lớp nào



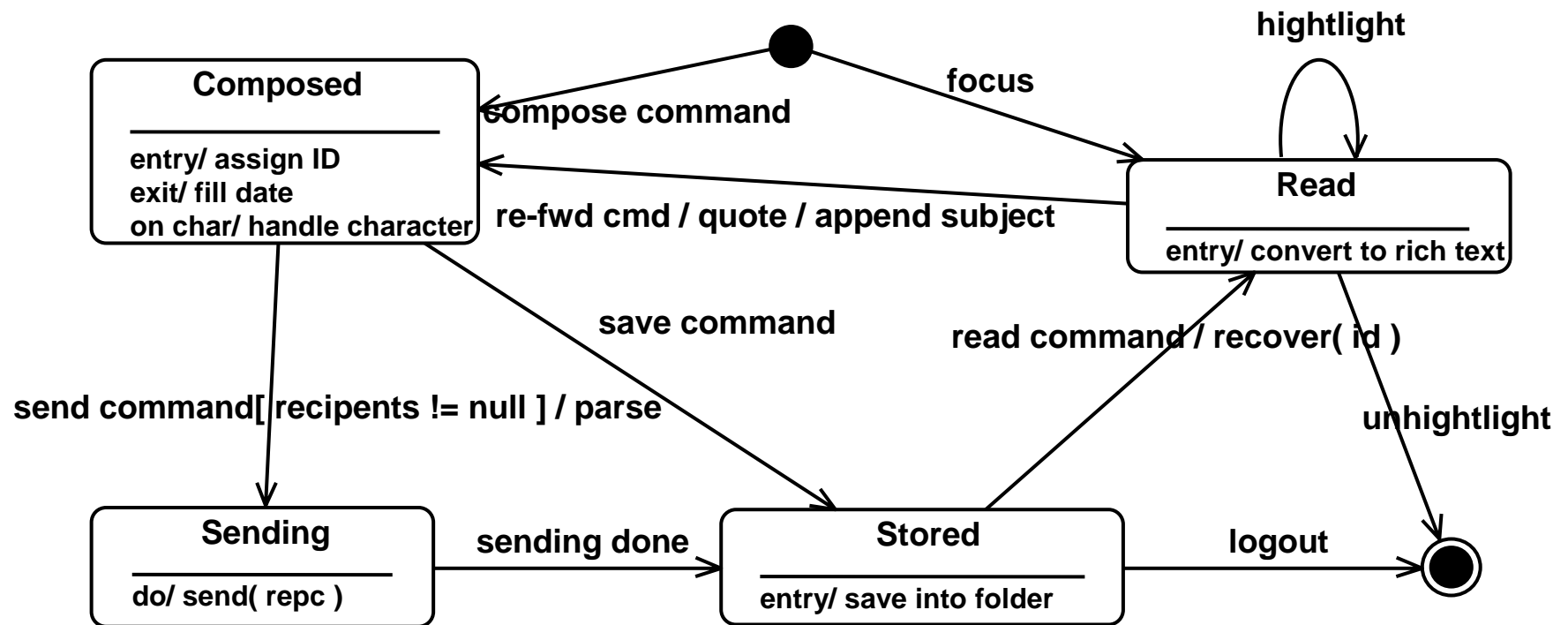
LÖÖC ÑOÀTRĂNG THAI (t.t)

- Dịch chuyển trạng thái là quan hệ giữa hai trạng thái theo một số điều kiện. Nếu trạng thái đầu tiên thỏa mãn các điều kiện sẽ chuyển sang trạng thái tiếp theo. Nếu không thỏa mãn sẽ tiếp tục ở trạng thái đầu tiên.
- Một ký hiệu nhỏ một mũi tên hướng từ trạng thái nguồn đến trạng thái đích và một nhãn.
- Nhãn cú pháp: **event-signature** '[' **guard-condition** ']'
/'/' action-expression

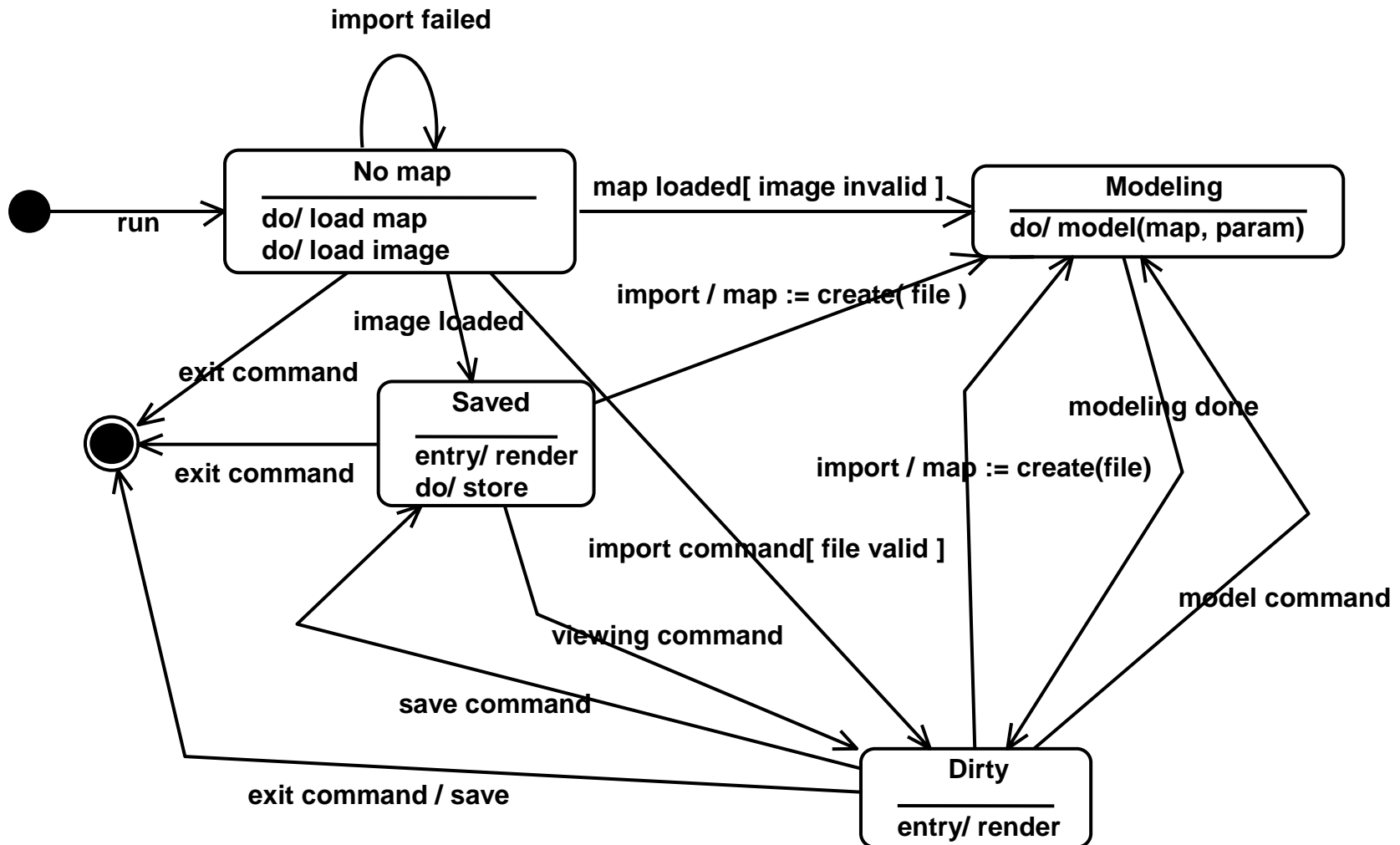


LÖÖC ÑOÀTRÀNG THAI (t.t)

- Ví dụ: lööc ñoàtràng thai của lớp Message

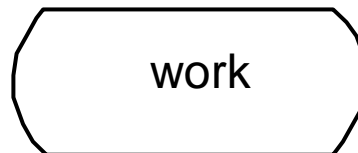


LÖÖC ÑOÀ TRẠNG THAI (t.t)



LÖÖC ÑOÀHOẠT ÑÔNG

- Lööc ñoàhoạt ñông (*activity diagram*) là một biến thể của lööc ñoà trạng thái trong ñoù trạng thái là sõi thöc thi một hành ñông và sõi dòch chuyển ñöôc kích hoạt khi hành ñông hoàn tất
- Ñöôc dùng ñeà mô tả một thuật toán hay thuật giải \Rightarrow tập trung vào các hành ñông
- Mỗi hành ñông ñöôc ký hiệu bằng hình vẽ nhỏ sau

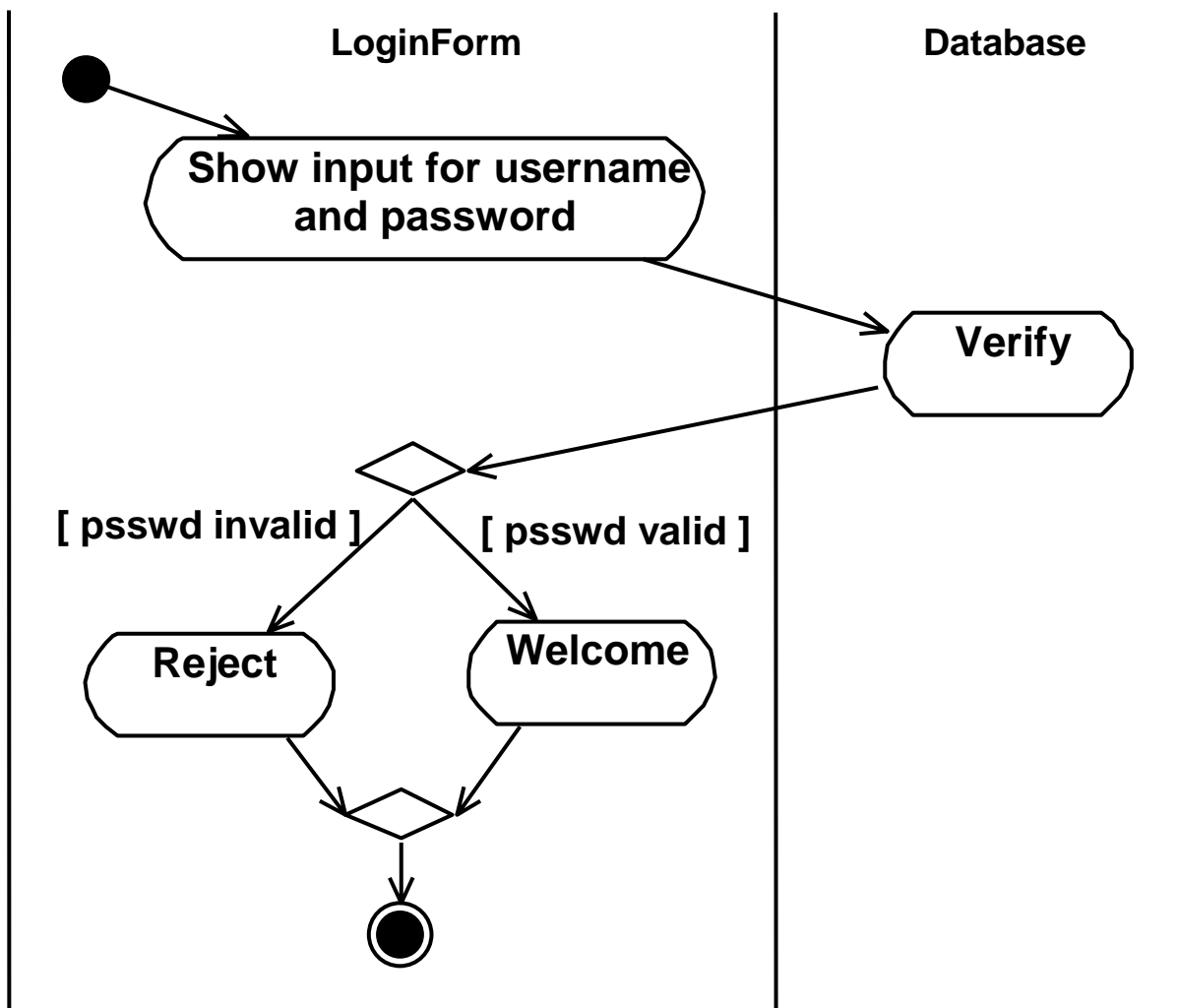


LÖÖC ÑOÀHOAÏT ÑOÑG (t.t)

- Quyết ñòngh reĩn hành: hình thoi còimột ñöông vào vớinhieun hành ra, mỗi hành ñöôcc gắnmột *guard-condition*
- Các hành ra ñöôcc nhậpláibằnmột hình thoi khác
- Mỗi “ñöông bơi” (*swimlane*) ñàĩ diễnmột lớp hoặcmột *actor*

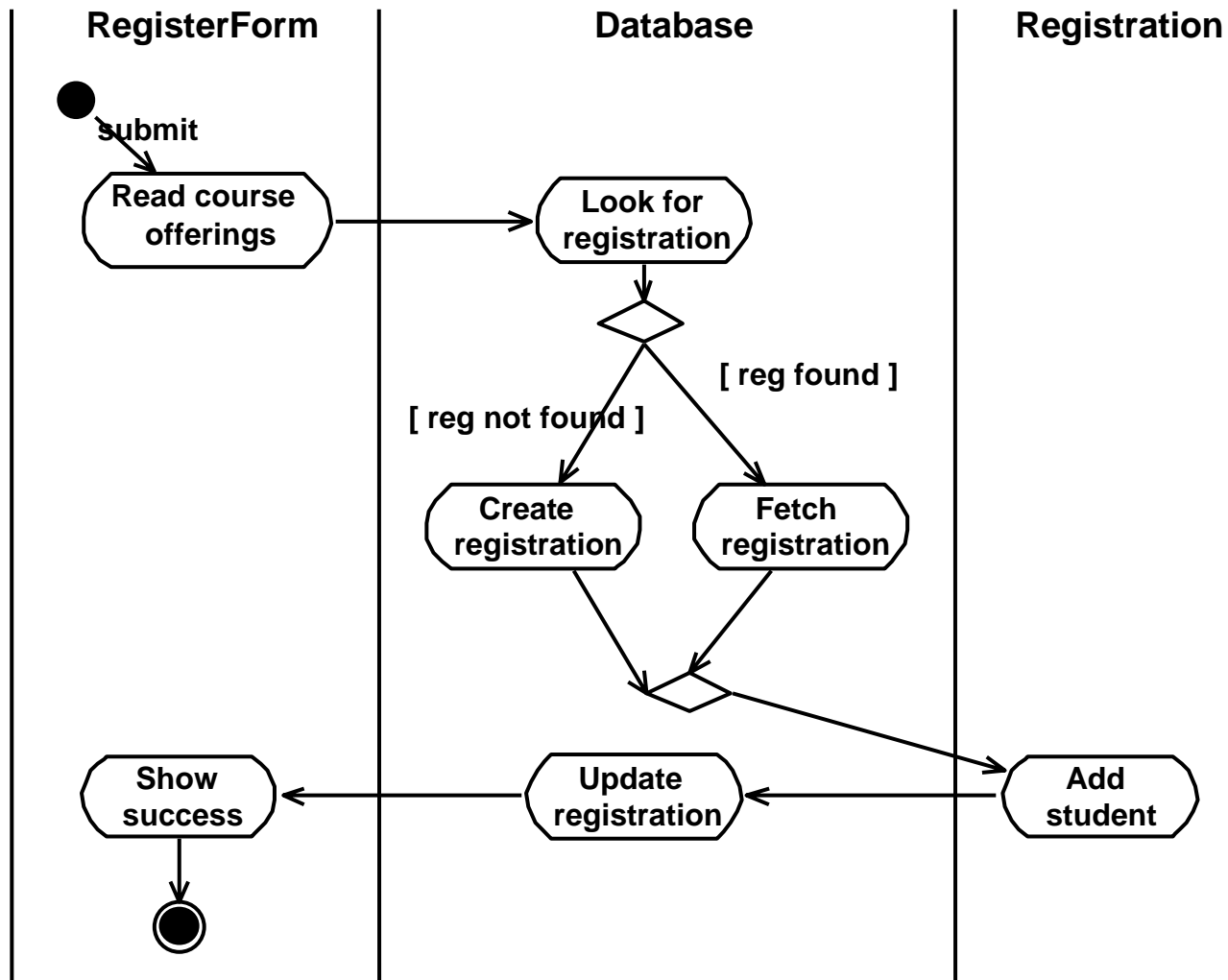


LÖÖC ÑOÀ HOẠT ÑOÀNG (t.t)



- Ví dụ: l   c    a ho  t     ng cho t  c v  i submit c  a LoginForm

LÖÖĬC ÑOÀHOAĬT ÑOŊG (t.t)



- Ví dụ: lỗi ãoàhoait ñoàng cho taic vui submit của RegisterForm



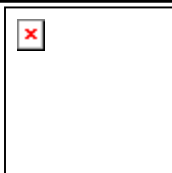
NHÂN DIỆN THÊM MỘT SỐ LỚP THIẾT KẾ

- Mô hình thiết kế phần nào chịu ảnh hưởng từ ngôn ngữ lập trình, thư viện hỗ trợ, *framework*, hệ điều hành và loại máy tính
- Một số lớp sẽ xuất hiện khi áp dụng những yếu tố trên
 - ◆ Ngôn ngữ lập trình: *template*, CObject...
 - ◆ Thư viện hỗ trợ: lớp Date, Time, List, Map, vector, iostream...
 - ◆ *Framework*: Applet, Panel, CDocument, CView, HttpServlet...
 - ◆ Hệ điều hành: các lớp thao tác file, mô tả môi trường network, các phần tử giao diện....

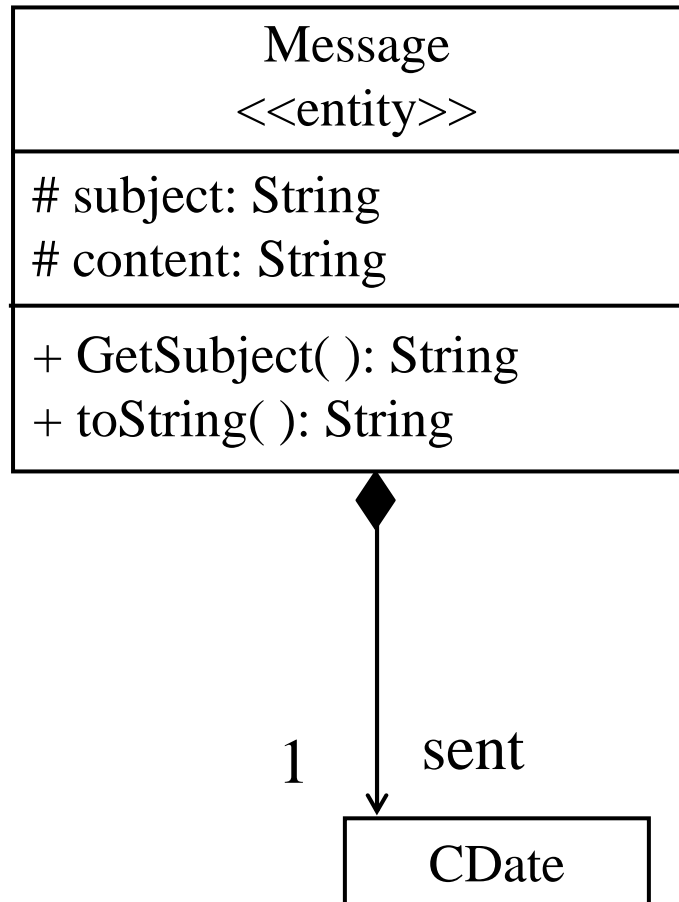


NHAN DIEN THEM MOT SO LOP THIET KE (t.t)

- Một số lớp khác xuất hiện làm chức năng duyệt (*iterate*) một lớp khác hay thực hiện các tính toán phức tạp...
- Xây dựng trực tiếp các lớp do thói quen hay ngôn ngữ cung cấp, hoặc
- Tạo ra lớp mới bằng cách thừa kế hay tích hợp các lớp cũ sẵn, ví dụ
`CArray<Isoquant, Isoquant&>`
- Bổ sung các lớp mới vào lược đồ lớp đang tồn tại cập nhật các mối quan hệ mới (bao gồm, phụ thuộc)



NAÏC TAÛCHI TIẾT CÀC THUOC TÍNH



- Trong mô hình phân tích cần phải xác rõ kiểu (hoặc cấu trúc dữ liệu) và một số truy xuất của các thuộc tính
- Có thể chọn một lớp cung cấp bộ hỗ trợ lập trình nếu cần thể hiện kiểu hay cấu trúc dữ liệu \Rightarrow bổ sung lớp của hỗ trợ và quan hệ bao gộp vào lược đồ lớp

NHAN DIEN CHINH XAC CAC TAC VUI

- Các lược đồ mô tả hành vi (công tác, tuần tởi, trạng thái, hành nỏng) giúp nhản diện chính xác các tác vui của các lớp
- Dữ liệu các thông nỏp hay hành nỏng ñể xác ñịnh *signature* của các tác vui
- Ví dụ: nhản diện một số tác vui của lớp Database

```
setReg( reg: Registration );
```

```
fetchReg( crsOff: CourseOffering) : Registration;
```



NHAN DIEN CHINH XAC CAC TAC VUI (t.t)

- Ví dụ: nhân diện một tác vui của lớp ChildView

```
render( );
```

```
store( );
```

```
load( );
```

```
model( map: FieldMap, param );
```

- Ví dụ: nhân diện một tác vui của lớp LoginForm

```
submit( uname: String; psswd: String );
```

```
makeWelcome( );
```



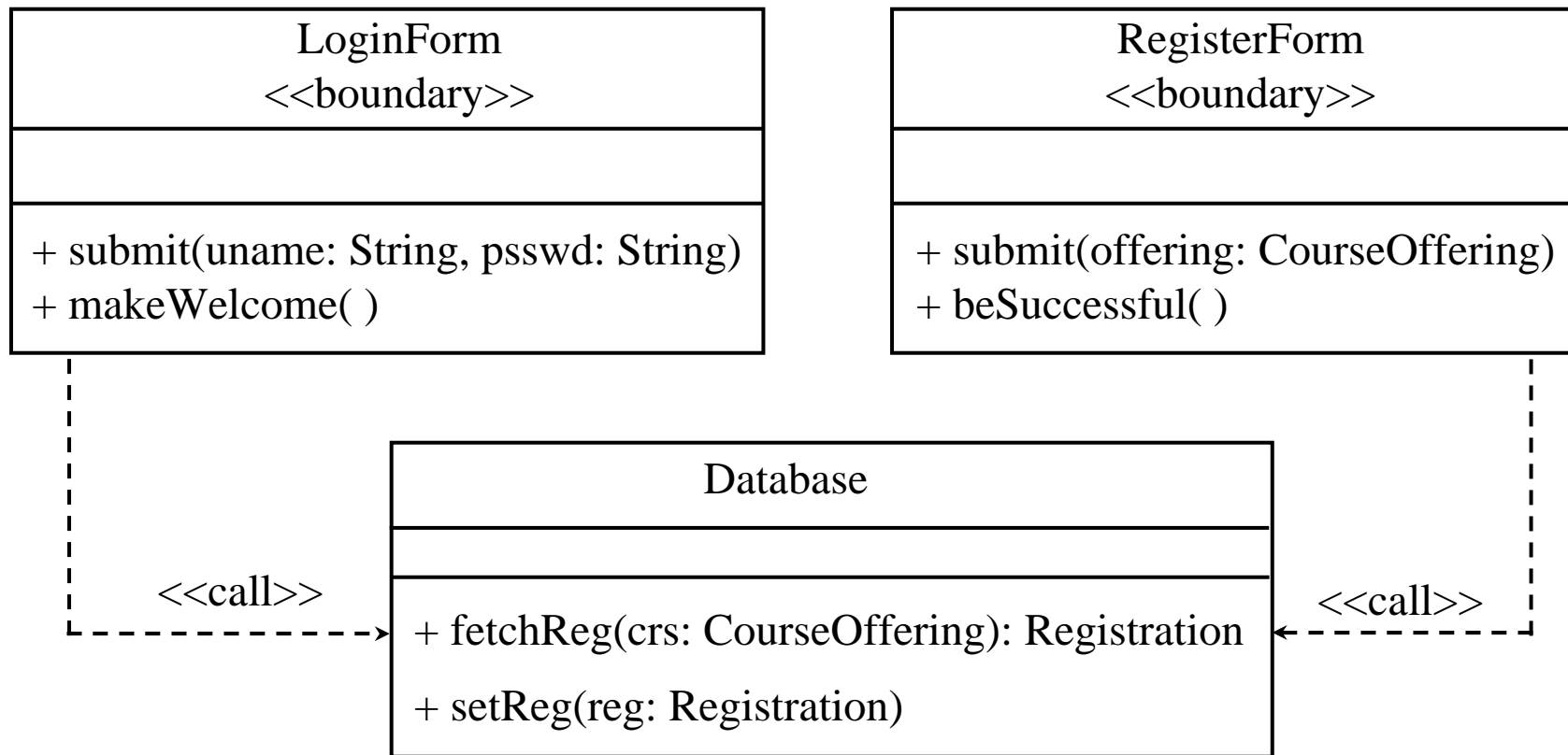
HOÀN CHỈNH LỘ TRÌNH

- Cập nhật các lớp mới, thuộc tính, tài nguyên và các mối quan hệ mới
- UML hình thức quan hệ phụ thuộc (*dependency*) giữa 2 lớp hoặc *package*: thay đổi một lớp, *package* kèm theo thay đổi lớp, *package* kia
- Ký hiệu của quan hệ phụ thuộc là mũi tên nét đứt: lớp, *package* ở phía đầu mũi tên phụ thuộc vào lớp, *package* phía cuối mũi tên
- Một số *stereotype* quy ước thông thường: <<call>>, <<instantiate>>, <<import>>, <<refine>>, <<realize>>, <<derive>>, <<trace>>

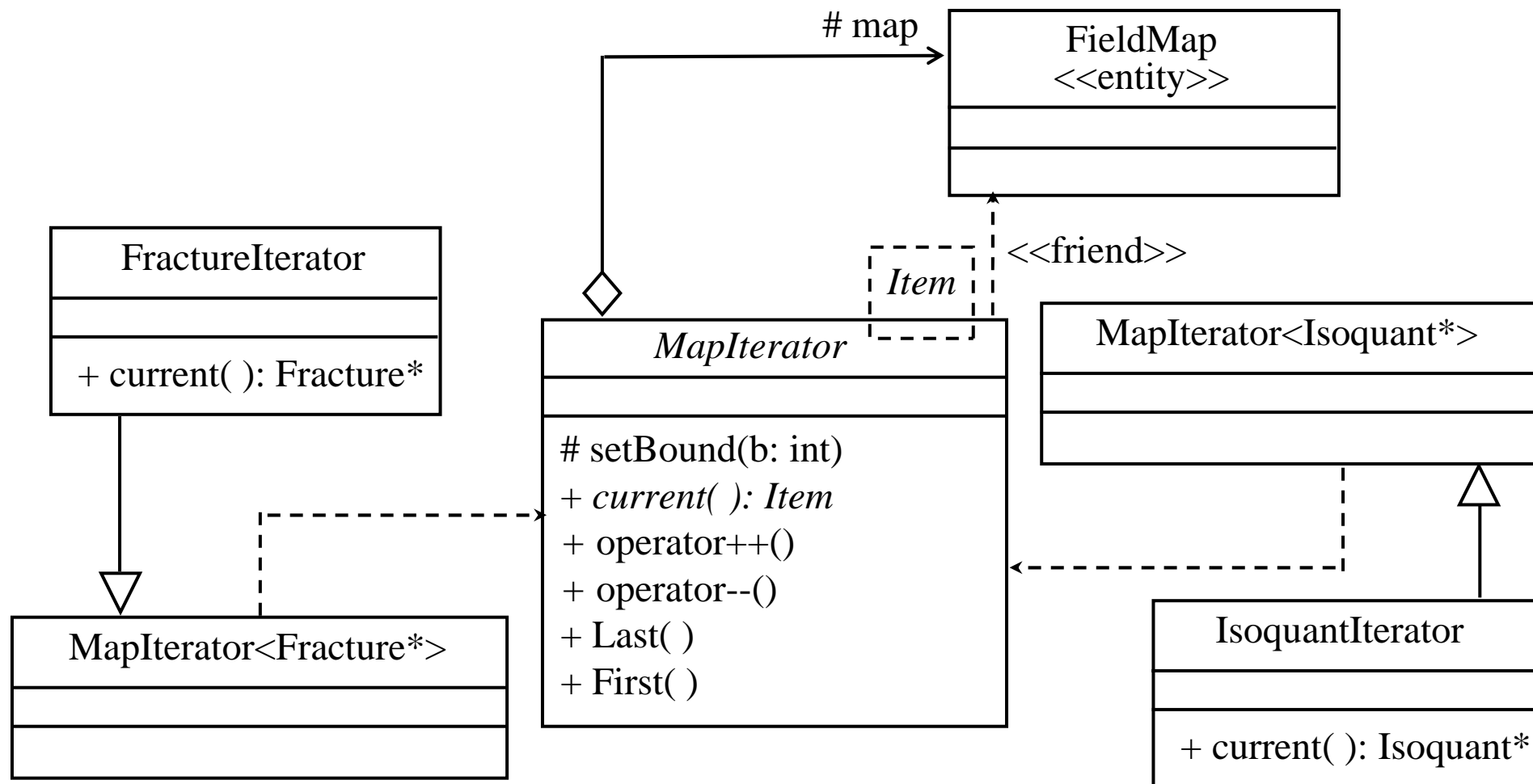


HOÀN CHỈNH LÖÖC ÑOÀ LÖP (t.t)

- Ví dụ: thêm l  c ñ  l  p cho h   th  ng ñ  ng ky   m  n ho  c



HOÀN CHỈNH LÖÖC ÑOÀ LÖP (t.t)



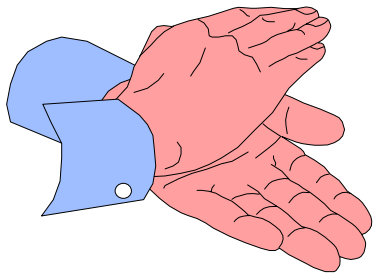
HOÀN CHỈNH LÖÖC ÑOÀ LỘP (t.t)

- Chuỗi sử dụng *package* để tổ chức các phần liên quan với nhau: các lớp về bản đồ hình, về thông tin sinh viên/giảng viên, về cửa sổ giao diện, về các servlet...
- Các *package* thể hiện kiến trúc phần mềm, thông thường chứa ảnh hưởng tới *framework* (Document/View, 3-tiers...)
- Mỗi *package* chứa một hoặc một vài lớp, trong đó có thể tham chiếu đến một số lớp thuộc các *package* khác



TỔNG KẾT

- Mô hình thiết kế bao trùm các khía cạnh tính năng của hệ thống phần mềm cần xây dựng
- UML hỗ trợ một số lược đồ giúp mô tả khía cạnh năng: công tác, tuân thủ, trạng thái, hành động
- Miêu tả chính xác thuộc tính và tác vụ, bổ sung một số lớp thiết kế \Rightarrow hoàn thiện khía cạnh tính
- Thiết lập các *package* tạo thành kiến trúc phần mềm



Chương 8

HIỆN THỨC VÀ TRIỂN KHAI

- ❖ Các thành phần
- ❖ Các thiết bị

NOI DUNG

8.1. Hien thoc

8.1.1. Thanh phan (*component*)

8.1.2. Looi ñoà thanh phan (*component diagram*)

8.1.3. Gan cac lop vao cac thanh phan ma nguon

8.1.4. Sinh ma nguon

8.2. Trien khai

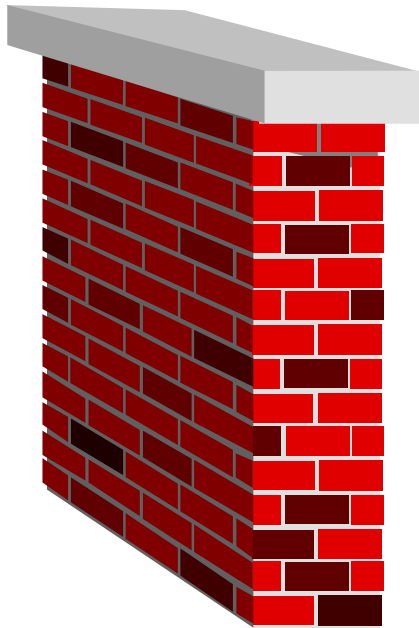
8.2.1. *Node* trien khai

8.2.2. Ket noi cac *node*

8.2.3. Looi ñoà trien khai (*deployment diagram*)



GIỚI THIỆU



- Cần phải xây dựng chương trình chạy thử để kiểm tra kết quả của giai đoạn thiết kế
- Các lớp sẽ được sử dụng để phân tích phần mềm nhỏ để xác định các thành phần và các mối quan hệ giữa chúng
- Chương trình sẽ được cài đặt và chạy thử để kiểm tra nguyên tắc tính toán ?

THÀNH PHẦN

- Thành phần (*component*) biểu diễn một phần hiển thức nào đó của hệ thống
- Một số *stereotype* quy ước thông thường:
 - ◆ <<file>>: mã nguồn hay dữ liệu
 - ◆ <<executable>>: chương trình chạy được
 - ◆ <<library>>: thư viện liên kết tĩnh hay động
 - ◆ <<document>>: tài liệu được thiết lập trong quá trình phát triển
 - ◆ <<table>>: bảng cơ sở dữ liệu



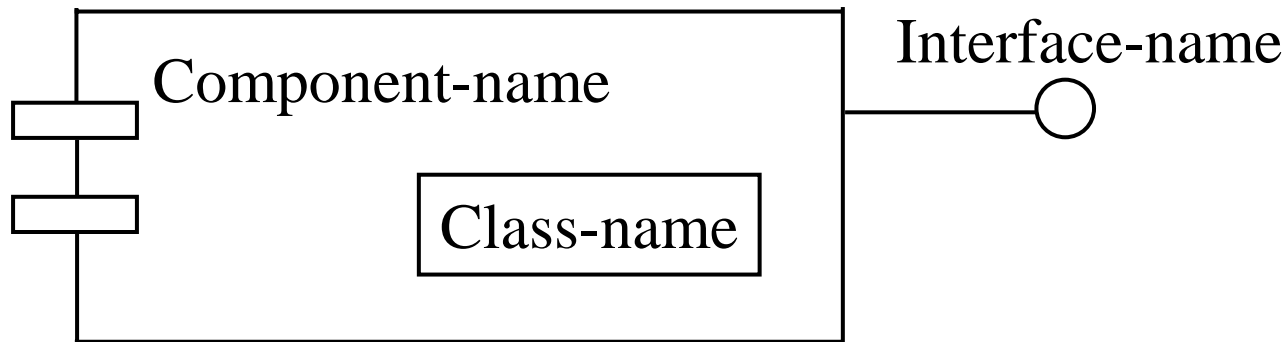
THÀNH PHẦN (t.t)

- Thành phần phần mềm (*software component*) bao gồm
 - ◆ Mã nguồn: *.cpp, *.c, *.pas, *.java, *.bas
 - ◆ Mã nối tổng: *.obj
 - ◆ Mã nhúng: *.class
 - ◆ Chương trình thực thi: *.dll, *.exe
- Thành phần phần mềm có thể tồn tại trong thời gian biên dịch, thời gian liên kết chương trình hoặc thời gian thực thi



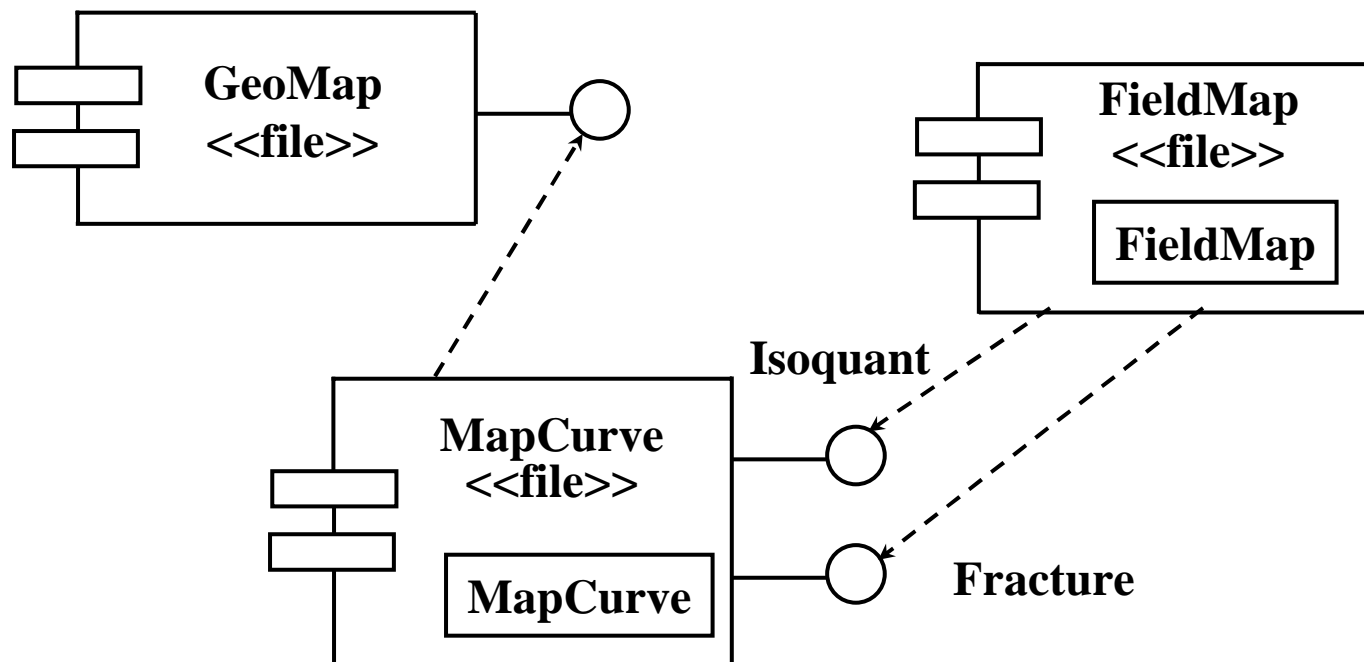
LÖÖC ÑOÀ THÀNH PHẦN

- Lööc ñoà thành phần là một ñoà thò gồm các thành phần kết nối với nhau bởi quan hệ phụ thuộc
- Ký hiệu của thành phần có thể bao gồm một số hình trong biểu diễn các giao tiếp và chứa các lớp mà nó sử dụng



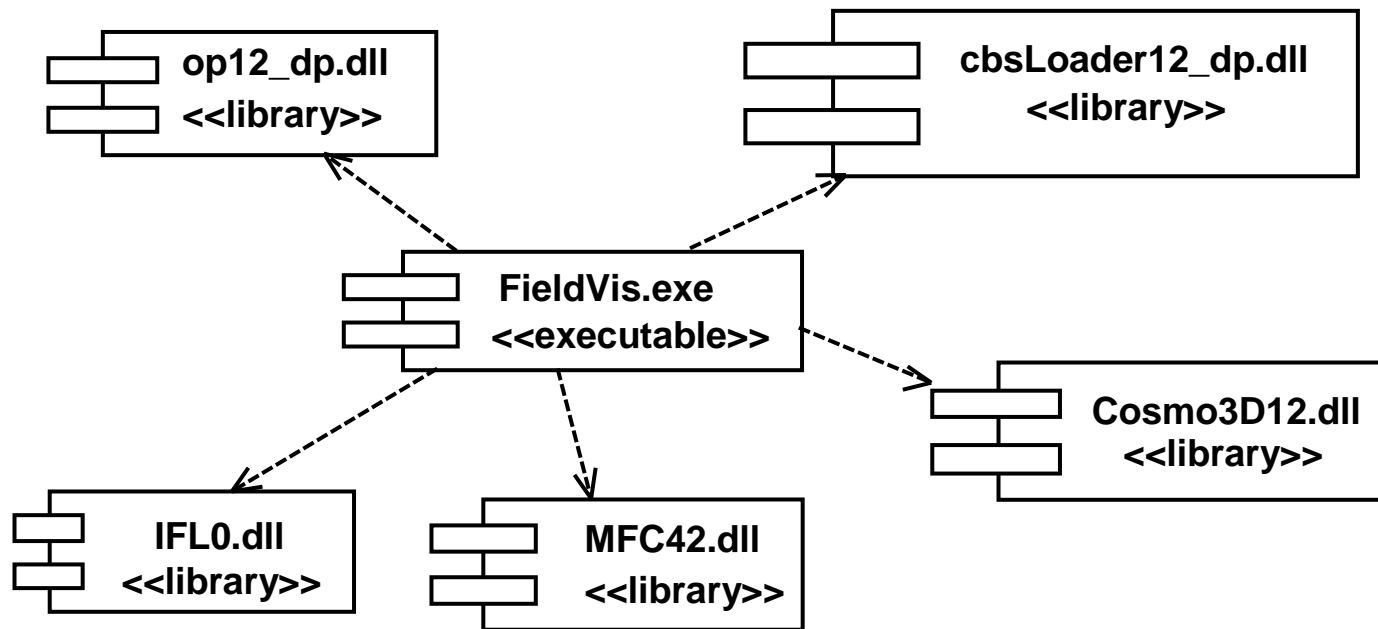
LÖÖC ÑOÀ THÀNH PHẦN (t.t)

- Ví dụ: l  c ñ   th  nh ph  n th   hi  n m  t s   *module* m  ng u  n c  a ch   ng tr  nh hi  n th   b   m  t ñ  a h  nh



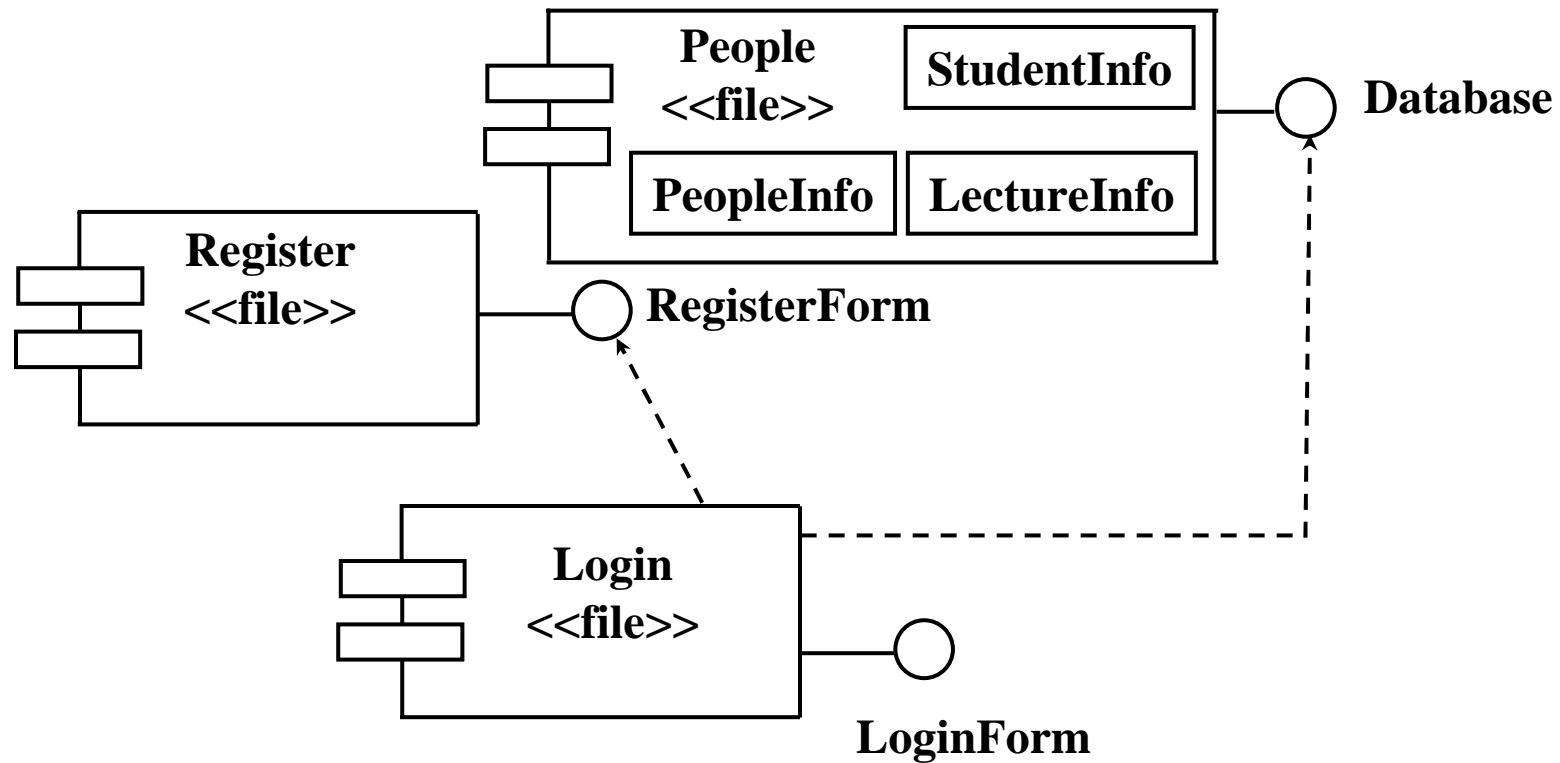
LÖÖC ÑOÀTHANH PHẦN (t.t)

- Ví dụ: lööc ñoàthanh phần thểhiện thời gian thời thi của chöông trình hiện thờ beàmat ñòà hình



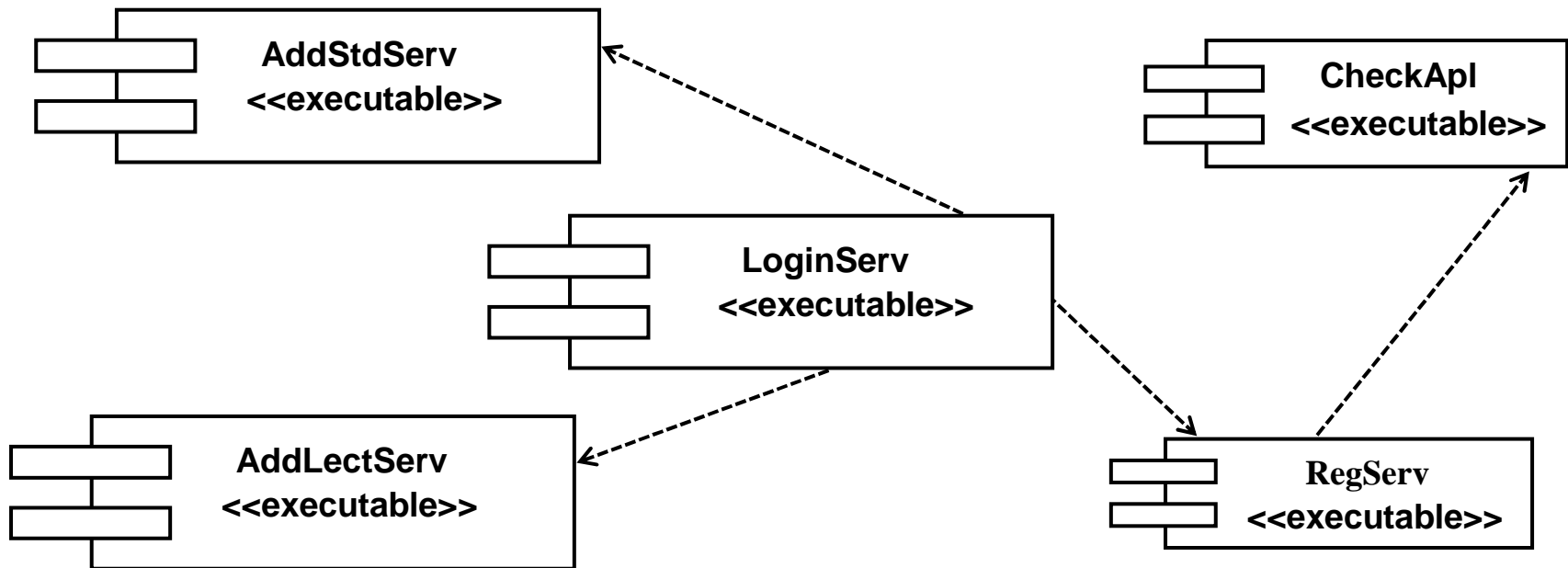
LÖÖC ÑOÀTHANH PHẦN (t.t)

- Ví dụ: löôc ñoàthanh phần của hệthống ñang kyừmôn học



LÖÖC ÑOÀTHANH PHẦN (t.t)

- Ví dụ: l  c ñ  thanh phần thể hi  n thời gian thực thi của hệ thống năng kỹ m  n học qua WEB



GAÌN CÁC LỚP VÀO CÁC THÀNH PHẦN

- Khi thiết lập các thành phần mã nguồn, chúng ta cần (bind) các lớp thiết kế vào các nguồn gốc lập trình
 - ◆ Gắn lớp FieldMap vào thành phần FieldMap (C++)
 - ◆ Gắn lớp MapCurve, Isoquant và Fracture vào thành phần MapCurve
 - ◆ Gắn lớp PeopleInfo, StudentInfo, LectureInfo và Database vào thành phần People (Java)
 - ◆ Gắn lớp và LoginForm vào thành phần Login (Java)
- Ký hiệu của thành phần chứa ký hiệu của lớp nó cần gắn
- Chúng ta *component* ≠ *package*



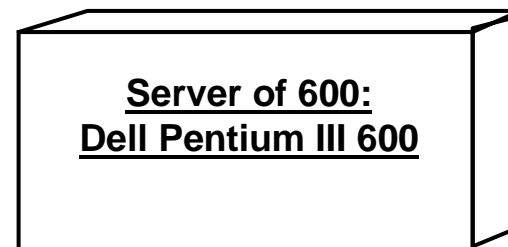
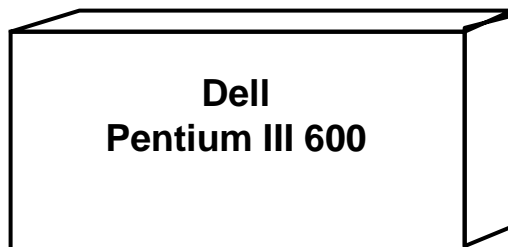
SINH MÃ NGUỒN

- Dựa vào các tài liệu để viết mã cho từng thành phần nguồn theo ngôn ngữ lập trình nào đó
- Viết mã nguồn là công việc hơi nhàm chán \Rightarrow có thể nhờ tới những hoạt động các công cụ CASE



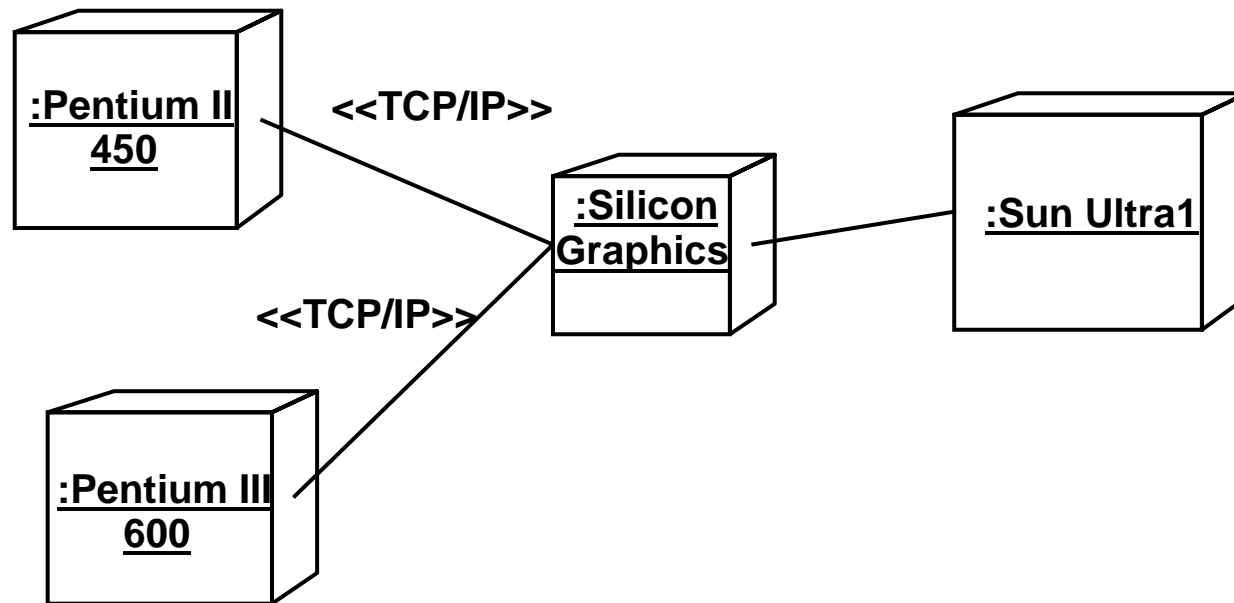
NODE TRIỂN KHAI

- Node là một thiết bị vật lý có khả năng tính toán, bao gồm: máy tính, máy in, thiết bị quét *card*, *router*...
- Node có mô tả cấu trúc 2 dạng: dạng lớp và dạng *instance*
- Node có ký hiệu nhỏ hình hộp ba chiều
- Các minh đồ của thành phần có thể sống trong một minh đồ node



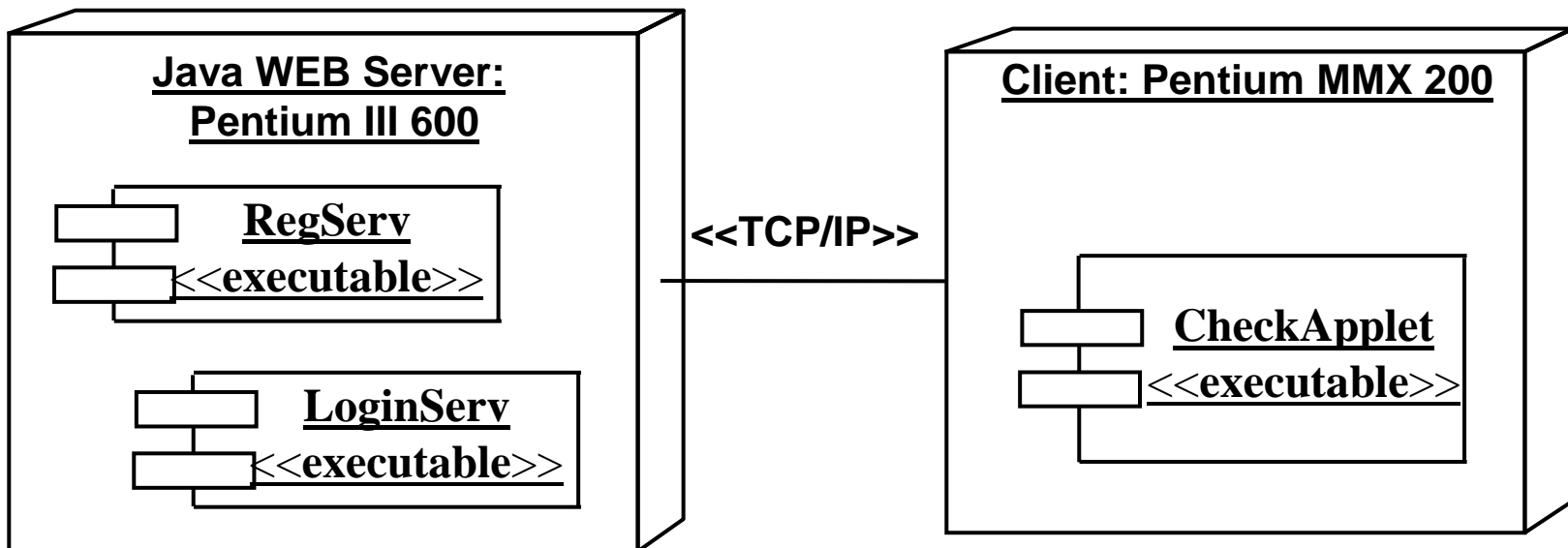
KEÁT NỐI CÁC NODE

- Có thể chia ra quan hệ liên kết giữa các *node* trong mô tả cấu trúc kết nối (*connection*)



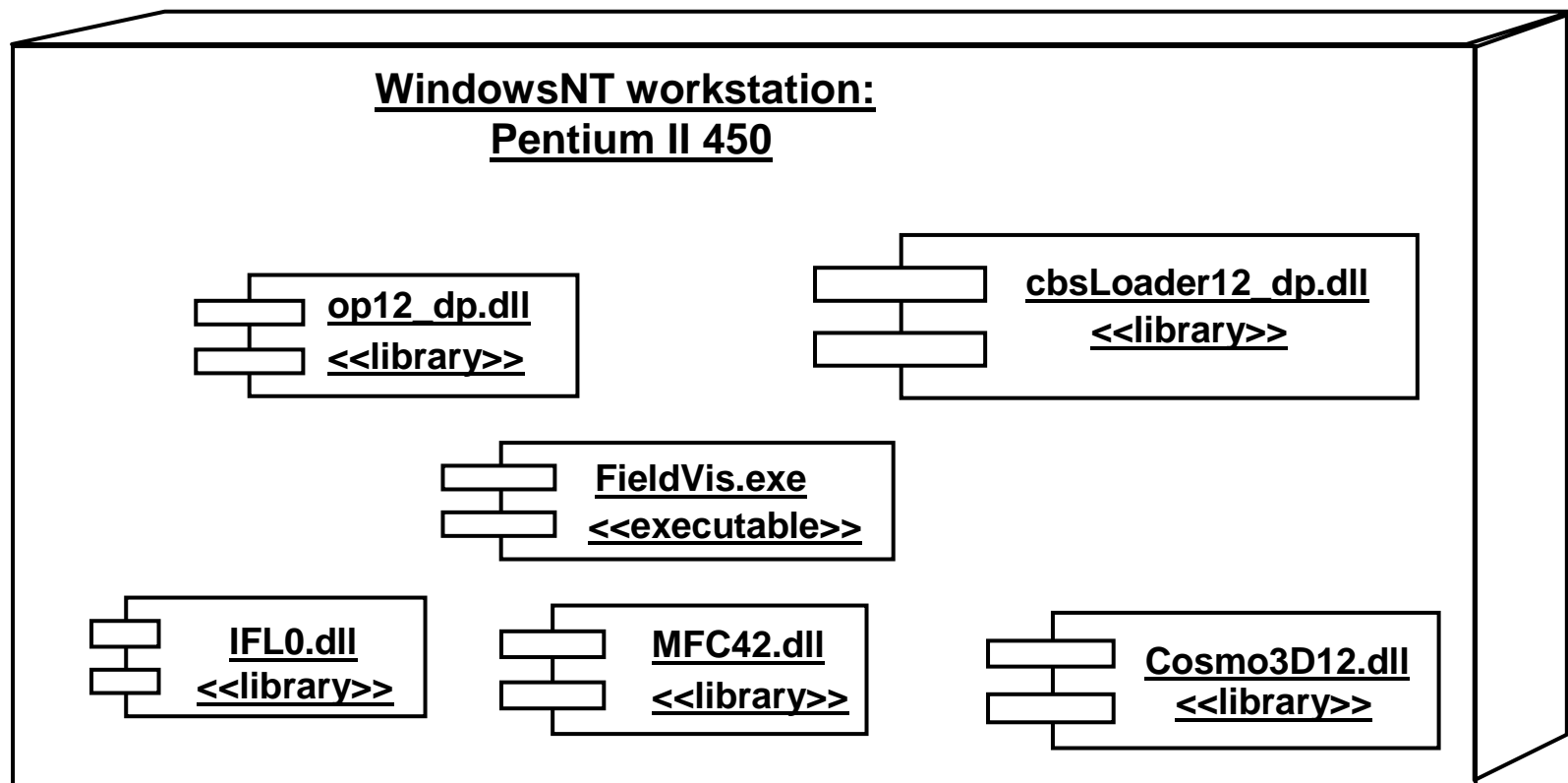
LÖÖC ÑOÀ TRIỂN KHAI

- Löôc ñoà triễn khai cho phép miêu tả cách cài ñặt các thành phần thộc thi trên các *node*
- Ví dụ: hệ thống ñăng ký môn học qua WEB



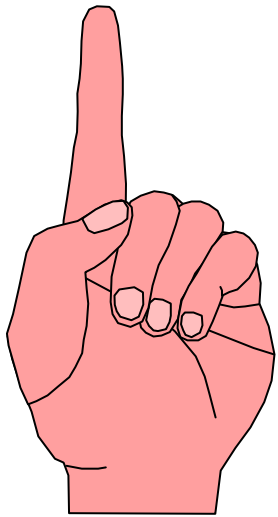
LÖÖIC ÑOÀTRIEÑ KHAI (t.t)

- Ví dụ: chương trình hiển thị bề mặt nhôa hình



TỔNG KẾT

- Hiện thực và triển khai tập trung vào xây dựng các thành phần chạy nền hoặc các thư viện, *module* mã nguồn, trang HTML, dạng nhô phần...
- Các thành phần mã nguồn cuối cùng của một số lập thiết kế và cấu trúc nền viết bằng các ngôn ngữ lập trình khác nhau
- Cuối cùng triển khai các thành phần chạy nền trên các thiết bị tính toán



Chương 9

KỸ THUẬT KIỂM NGHIỆM PHẦN MỀM

◆ *Test-case*

◆ Kiểm tra các công việc lập cơ bản

NOI DUNG

9.1 Một số khái niệm

9.2. Các bước kiểm nghiệm phần mềm

9.1.1. Mục tiêu của kiểm nghiệm phần mềm

9.1.2. Các nguyên lý của kiểm nghiệm phần mềm

9.1.3. Thiết lập các *test-case*

9.3. Kiểm nghiệm các lỗi logic lập trình

9.2.1. Xây dựng đồ thị dòng chảy (*flow graph*)

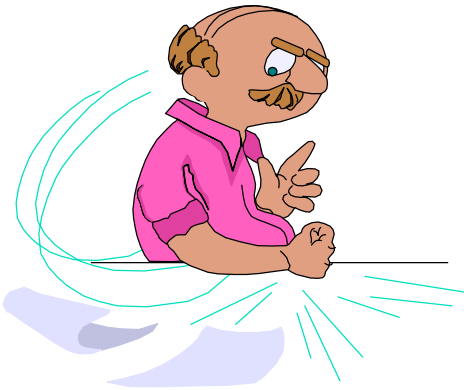
9.2.2. Liệt kê các lỗi logic lập trình

9.2.3. Thiết lập các *test-case*



GIỚI THIỆU

- Mặc dù những tài năng của một phần lớn các công cụ CASE, rất nhiều công việc trong quá trình sản xuất phần mềm vẫn cần thực hiện bởi con người
- Loài lỗi xảy ra trong tất cả các giai đoạn: phân tích yêu cầu, thiết kế mã nguồn
- Do đó phải kiểm nghiệm chương trình trước khi chính thức sử dụng



MOẬT SỐ KINH NGHIỆM

- Kiểm nghiệm phần mềm là hoạt động thi công trình với mục đích tìm ra lỗi \Rightarrow phê phán, không mang tính xây dựng
- Phân loại:
 - ◆ Kiểm nghiệm *black-box*: kiểm tra các chức năng của phần mềm, không quan tâm cấu trúc bên trong, thông dụng cho những *module* lớn.
 - ◆ Kiểm nghiệm *white-box*: kiểm tra cấu trúc nội bộ bên trong chương trình, thông dụng cho những *module* nhỏ
- Mỗi loại kiểm nghiệm có khả năng tìm ra những nhóm lỗi khác nhau \Rightarrow nên kết hợp cả hai

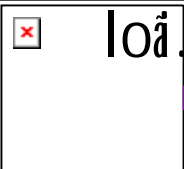
MỨC TIÊU CỦA KIỂM NGHIỆM PHẦN MỀM

- Mục tiêu của kiểm nghiệm phần mềm là tìm ra lỗi (nếu có) với chi phí thấp nhất.
 - Kiểm nghiệm phần mềm giúp
 - ◆ Phát hiện lỗi trong chương trình (nếu có).
 - ◆ Chứng minh được phần mềm hoạt động đúng nhờ thiết kế
 - ◆ Chứng minh được phần mềm đáp ứng yêu cầu của *user*
- ⇒ Góp phần chứng minh chất lượng của phần mềm.



MỤC TIÊU CỦA KIỂM NGHIỆM PHẦN MỀM (t.t)

- Quá trình kiểm nghiệm phần mềm là tốt khi
 - ◆ Có khả năng tìm ra lỗi cao.
 - ◆ Không dễ thỏa.
 - ◆ Biết chọn lọc: chỉ kiểm nghiệm những phần nào có khả năng tìm ra lỗi nặng trọng.
 - ◆ Không quá phức tạp cũng không quá nhàn nhã.
- Chuyên Kiểm nghiệm phần mềm không phải là một công việc phân mềm không còn khiếm khuyết, chỉ là công việc phân mềm có



lỗi.

CÁC NGUYÊN LÝ KIỂM NGHIỆM PHẦN MỀM

- Việc kiểm nghiệm nên hướng vào yêu cầu của khách hàng
- Nên có kế hoạch nên trước một thời gian dài.
- Áp dụng nguyên lý Pareto: 80% lỗi có nguyên nhân từ 20% các *module* \Rightarrow có thể tập trung kiểm tra những *module* quan trọng nhất.
- Nên tiến hành từng bước liên tục: bắt đầu từ những *module* riêng biệt rồi sau đó tích hợp các *module* lại.
- Không thể kiểm nghiệm triệt để một phần mềm.
- Nên có thể hiện bởi những nhà tài trợ KHÔNG tham gia vào quá trình phát triển phần mềm.



THIẾT LẬP CÁC TEST-CASE

- Khái niệm *test-case*
 - ◆ Dữ liệu *input*
 - ◆ Thao tác kiểm nghiệm
 - ◆ Dữ liệu *output* hay kết quả mong đợi của chương trình
- *Test-case* cho kiểm nghiệm *black-box*: chủ yếu dựa vào các yêu cầu chức năng của chức năng phần mềm.
- *Test-case* cho kiểm nghiệm *white-box*: chủ yếu dựa vào cấu trúc nội bộ của phần mềm \Rightarrow vấn đề đặt ra: số lượng *test-case* cần thiết là bao nhiêu



KIỂM NGHIỆM CÁC NÖÔNG NÖC LẠP CÔ BAN

- Kiểm nghiệm *white-box* dựa vào cấu trúc nhiều khiên của thiết kế thuật toán để sinh các *test-case* với tiêu chí
 - ◆ Tất cả các công thức thi công lập công thức ít nhất một lần
 - ◆ Thời gian nhiều khiên rõ ràng ở hai nhánh **true** và **false**
 - ◆ Thời gian vòng lặp tại biến cũng nhớ bên trong
 - ◆ Thời gian cấu trúc dữ liệu để đảm bảo tính toán về của nó
- Kiểm nghiệm các công thức lập công bản là một trong những phương pháp kiểm nghiệm *white-box*

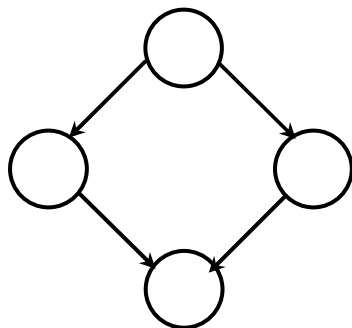
XÂY DỰNG NƠI ĐÓNG CHÁY

- Mỗi *node* hình tròn biểu diễn một hoặc một vài tác vụ (hội khai so với lỗi thuật giải)
- Cạnh có hướng miêu tả mối quan hệ
- Nơi đóng cháy được xây dựng từ lỗi thuật giải

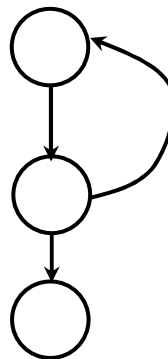
sequence



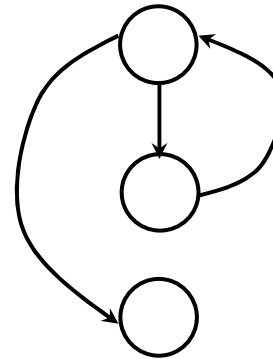
if



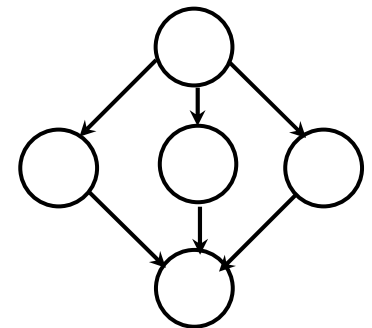
until



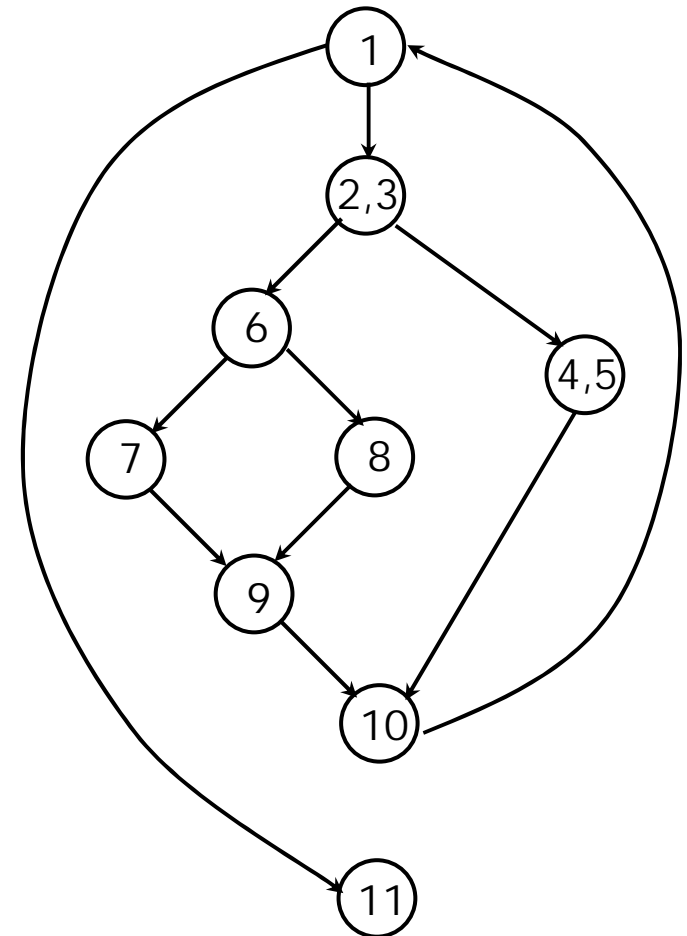
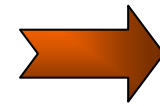
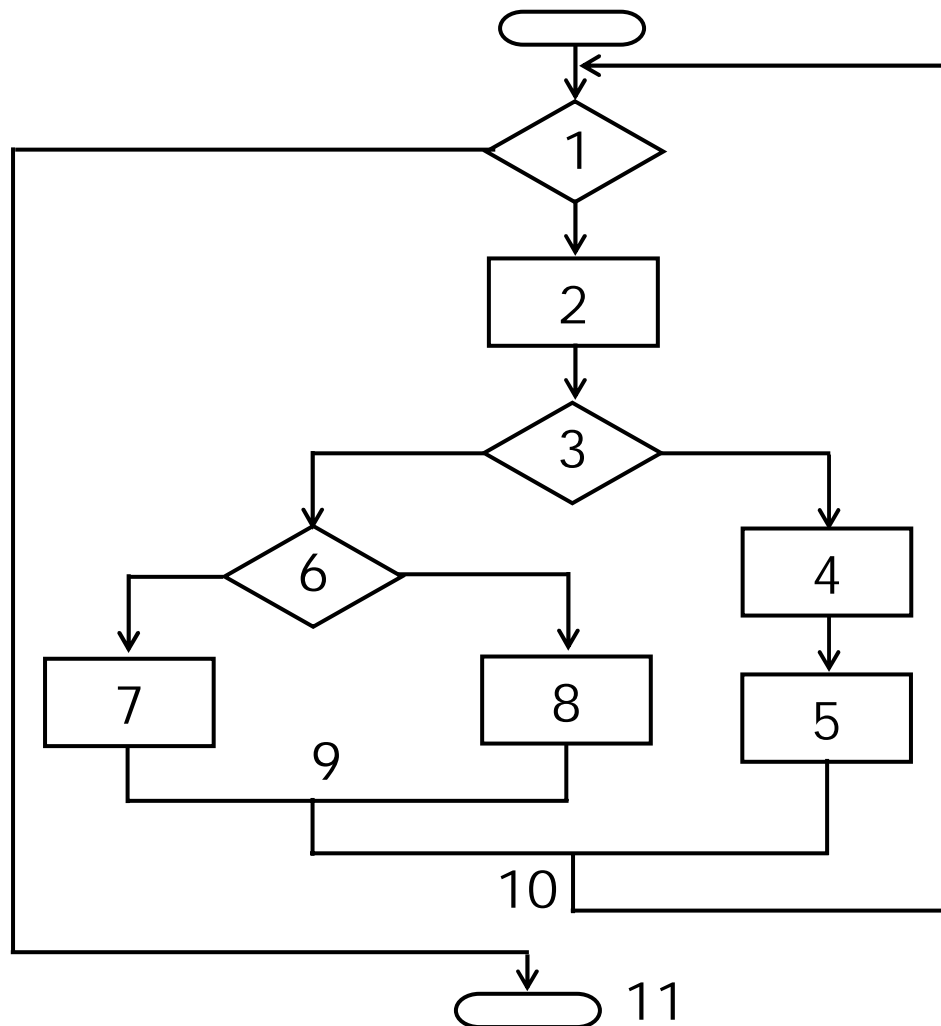
while



case

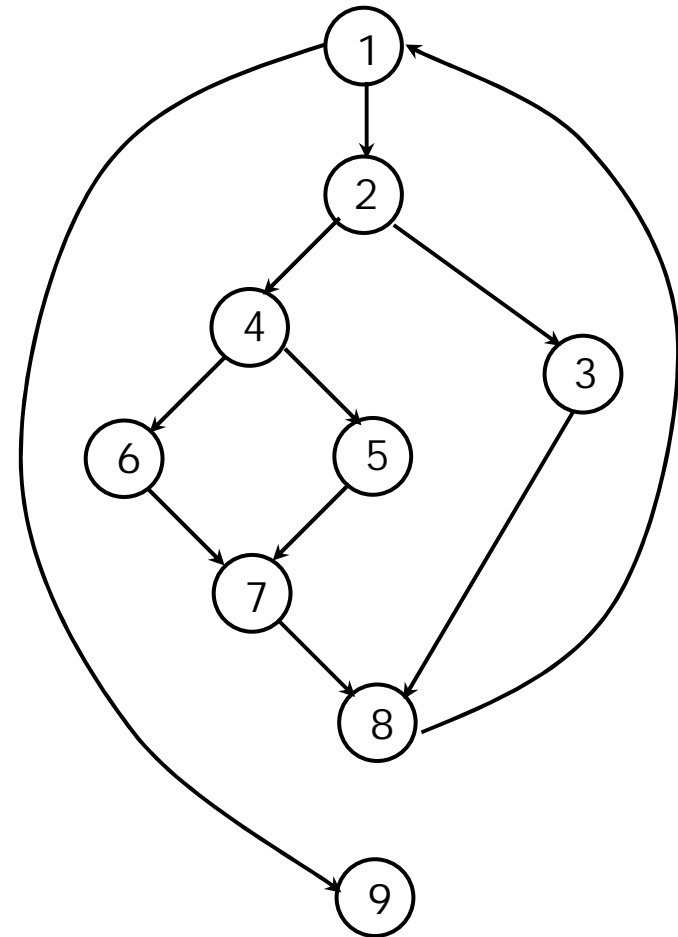
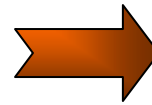


XÂY DỰNG MÃ THÌ DÒNG CHAY (t.t)



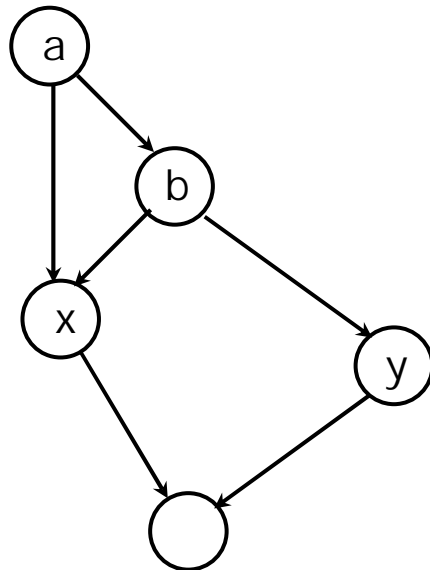
XÂY DỰNG ÑỒÀ THỜ DÒNG CHẢY (t.t)

```
procedure: DoSomething
1:      do while x=0
2:          if y=0 then
3:              z=0;
4:          elseif k=0 then
5:              z=1;
6:          else x=1;
7:          endif;
8:      enddo
9: end
```

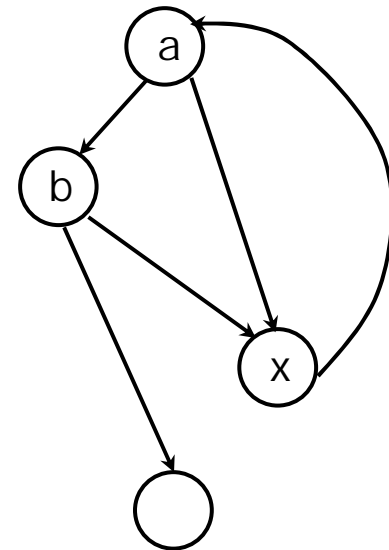


XÂY DỰNG NÀO THÌ DÒNG CHẢY (t.t)

- Phải phân rã các điều kiện phức tạp thành các điều kiện đơn
- Mỗi *node* mô tả một điều kiện đơn nên nó gọi là *predicate*



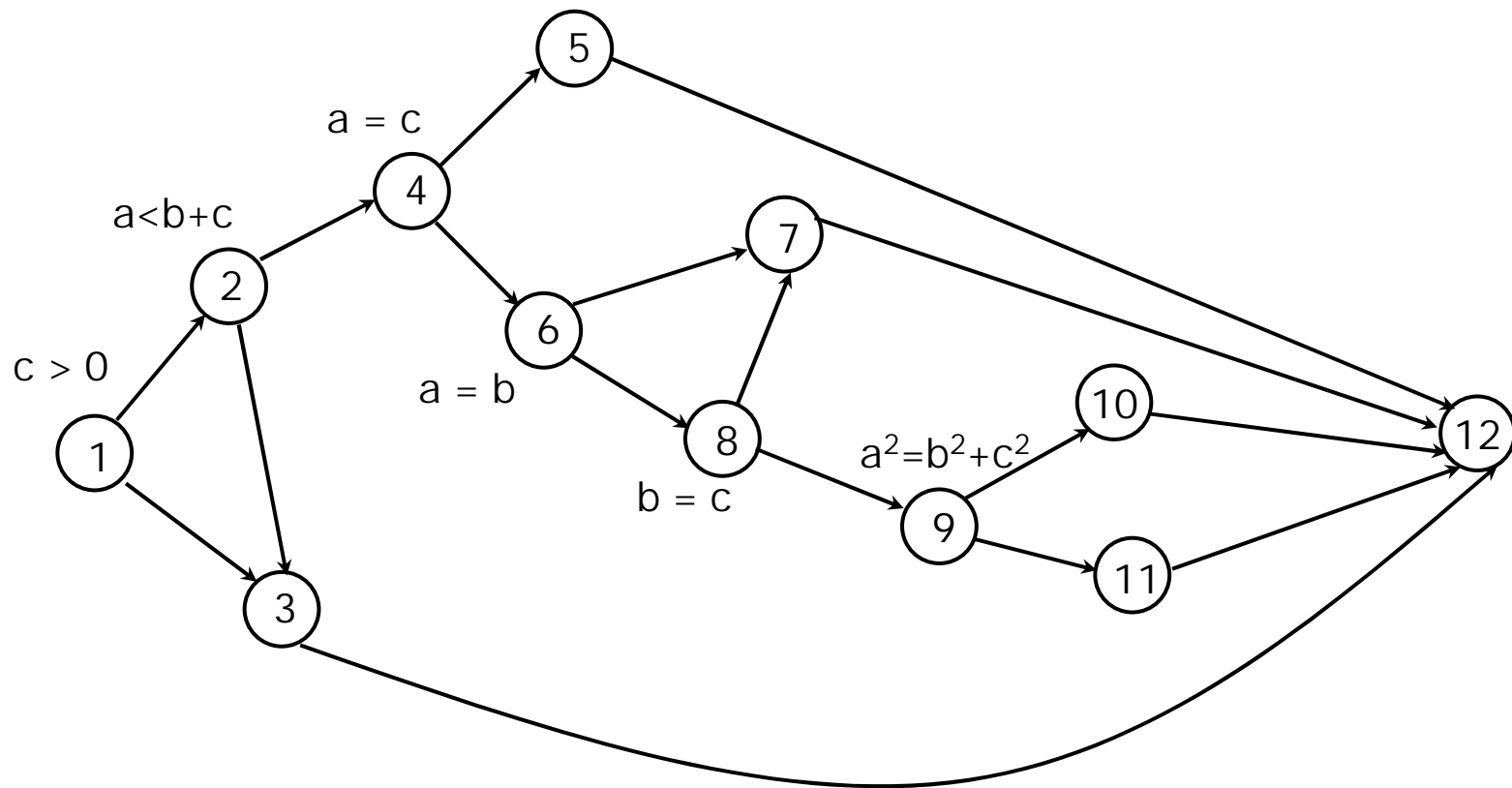
if a and b then y else x



while a or b do x

XÂY DỰNG MÀNG DÒNG CHẢY (t.t)

procedure AnalyzeTriangle



LIEA KEA CAI NOONG NOC LAP CO BAN

- Tõ node bat nãu nẽn node kea thuẽ, cai nõong thõc thi cõ ban nõõc liea kea theo mã thõ tõi nã nõi nẽ nãim bã rang: nõong nang liea kea ít nhã nĩ qua mã canh chõ nõõc duyẽ qua bõ cai nõong nã liea kea trõõc nõi
- Tõng soã nõong thõc thi cõ ban nõc lap nhã nõõc tĩnh bẽng

$V = P + 1$; trong nõi P la soã node phãn nhãn (*predicate*)

LIỆT KÊ CÁC NỖNG NƠI LẬP CÔ BẢN (t.t)

- Nói với chương trình con `DoSomething`

- ◆ Tổng số công : $V = 3 + 1 = 4$

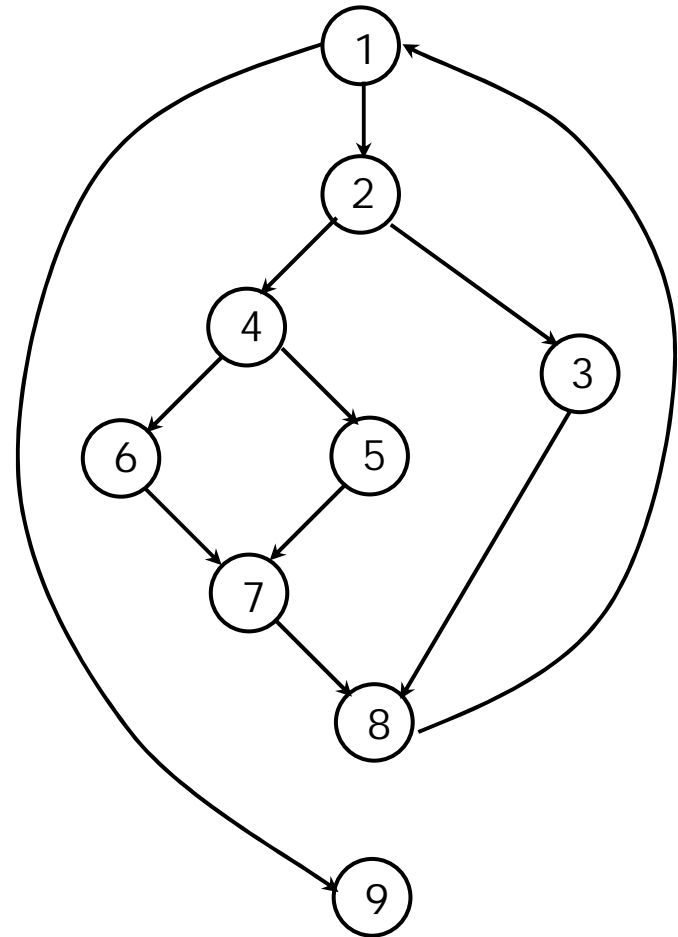
- ◆ Công 1: 1-9

- ◆ Công 2: 1-2-3-8-1...

- ◆ Công 3: 1-2-4-5-7-8-1...

- ◆ Công 4: 1-2-4-6-7-8-1...

- Chuỗi dài 3 chấm (...) mang ý nghĩa "không quan tâm", tức là có thể đi theo bất kỳ cách nào bởi vì các cạnh sau đó nào cũng duyệt qua rồi



LIỆT KÊ CÁC NỖÔNG NƠI LẬP CÔ BẢN (t.t)

- Noá vôi chöông trình con **AnalyzeTriangle**

- ◆ Tổng số nööông : $V = 6 + 1 = 7$

- ◆ Nööông 1: 1-3-12

- ◆ Nööông 2: 1-2-3-12

- ◆ Nööông 3: 1-2-4-5-12

- ◆ Nööông 4: 1-2-4-6-7-12

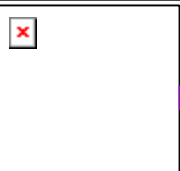
- ◆ Nööông 5: 1-2-4-6-8-7-12

- ◆ Nööông 6: 1-2-4-6-8-9-10-12

- ◆ Nööông 7: 1-2-4-6-8-9-11-12

THIẾT LẬP CÁC *TEST-CASE*

- Thiết lập một *test-case* cho mỗi nhóm thời gian
- Dựa vào thuật giải để tìm ra một dữ liệu input, sau đó tính ra dữ liệu *output* hay đáp ứng mong đợi của thuật giải
- Chú ý có thể không tạo ra một *test-case* cho mỗi nhóm thời gian nào



THIẾT LẬP CÁC *TEST-CASE* (t.t)

- Sinh *test-case* cho chương trình con AnalyzeTriangle

- *Test-case* cho nhóm 1:

- ◆ Input: $a = 3, b = 2, c = 0$

- ◆ Output mong đợi: $\text{type} = \text{"Error"}$

- *Test-case* cho nhóm 2:

- ◆ Input: $a = 17, b = 5, c = 4$

- ◆ Output mong đợi: $\text{type} = \text{"Error"}$

- *Test-case* cho nhóm 3:

- ◆ Input: $a = 6, b = 6, c = 6$

- ◆ Output mong đợi: $\text{type} = \text{"Equilateral"}$



THIẾT LẬP CÁC *TEST-CASE* (t.t)

- *Test-case* cho trường 4:

- ◆ Input:

$a = 7, b = 7, c = 4$

- ◆ Output mong đợi:

type = "Isosceles"

- *Test-case* cho trường 5:

- ◆ Input:

$a = 12, b = 9, c = 9$

- ◆ Output mong đợi:

type = "Isosceles"

- *Test-case* cho trường 6:

- ◆ Input:

$a = 5, b = 4, c = 3$

- ◆ Output mong đợi:

type = "Right"

- *Test-case* cho trường 7:

- ◆ Input:

$a = 13, b = 11, c = 6$

- ◆ Output mong đợi:

type = "Scalene"

TỔNG KẾT

- Mục tiêu của kiểm nghiệm phần mềm là tìm ra lỗi
- Hai loại kiểm nghiệm: *white-box* và *black-box*.
- Kiểm nghiệm các lỗi thường gặp lập cơ bản dùng trong kiểm nghiệm *white-box*, bao gồm các bước
 - ◆ Thiết lập môi trường đang chạy
 - ◆ Liệt kê các lỗi thường thấy khi lập cơ bản
 - ◆ Sinh các *test-case* cho các lỗi thường thấy



Chương 10

CHIẾN THUẬT KIỂM NGHIỆM PHẦN MỀM

- ❖ *Verification & Validation*
- ❖ *Unit test & Integration test*
- ❖ Kiểm nghiệm hệ thống nội tổng
- ❖ Thuật toán gỡ lỗi

NOI DUNG

10.1. Một số khái niệm

10.1.1. *Verification và validation*

10.1.2. Một chiến thuật kiểm nghiệm phổ biến

10.2. Kiểm nghiệm từng *module*

10.3. Kiểm nghiệm tích hợp

10.3.1. Tích hợp từ trên xuống (*top-down*)

10.3.2. Tích hợp từ dưới lên (*bottom-up*)

10.3.3. Kiểm nghiệm hồi quy (*regression*)

10.4. Kiểm nghiệm tính năng (*validation*)



NOI DUNG (t.t)

10.5. Kiểm nghiệm phòng nhiễu tổng

10.5.1. Kiểm nghiệm nhiễu và phòng nhiễu tổng

10.5.2. Kiểm nghiệm tích hợp phòng nhiễu tổng

10.5.3. Kiểm nghiệm theo kịch bản

10.6. Thuật toán gỡ rối (*debug*)

10.6.1. *Brute force*

10.6.2. Loại trừ nguyên nhân

10.6.3. Theo vết



MOẬT SỐ KẾT QUẢ KIỂM

- Chiến thuật kiểm tra phần mềm tích hợp các phương pháp tạo ra *test-case* trở thành một chuỗi các bước có thể lặp lại để kiểm tra nghiệm phần mềm thành công.
- Bao gồm các công việc
 - ◆ Lập kế hoạch kiểm nghiệm
 - ◆ Sinh *test-case*
 - ◆ Thực hiện kiểm nghiệm, thu thập kết quả và đánh giá

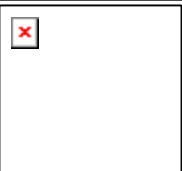
VERIFICATION và VALIDATION

● *Verification*: các hành động nhằm bảo đảm cho phần mềm được

hiện thực đúng theo một chức năng cụ thể nào đó \Rightarrow "*Are we building the product right?*"

● *Validation*: các hành động nhằm bảo đảm cho phần mềm được xây

dựng theo đúng yêu cầu của khách hàng \Rightarrow "*Are we building the right product?*"



MO' CHIEN THUA' KIEM NGHIEM PHO' BIEN

Phân tích toán b' h' th' ng

Kiểm nghiệm toán b' h' th' ng

Phân tích yêu cầu

Kiểm nghiệm tính năng

Thiết kế

Kiểm nghiệm tích hợp

Mã ho' i

Kiểm nghiệm ñ' n' v' i



MO' CHIEN THUA'T KIEM NGHIEM PHO'BIEN (t.t)

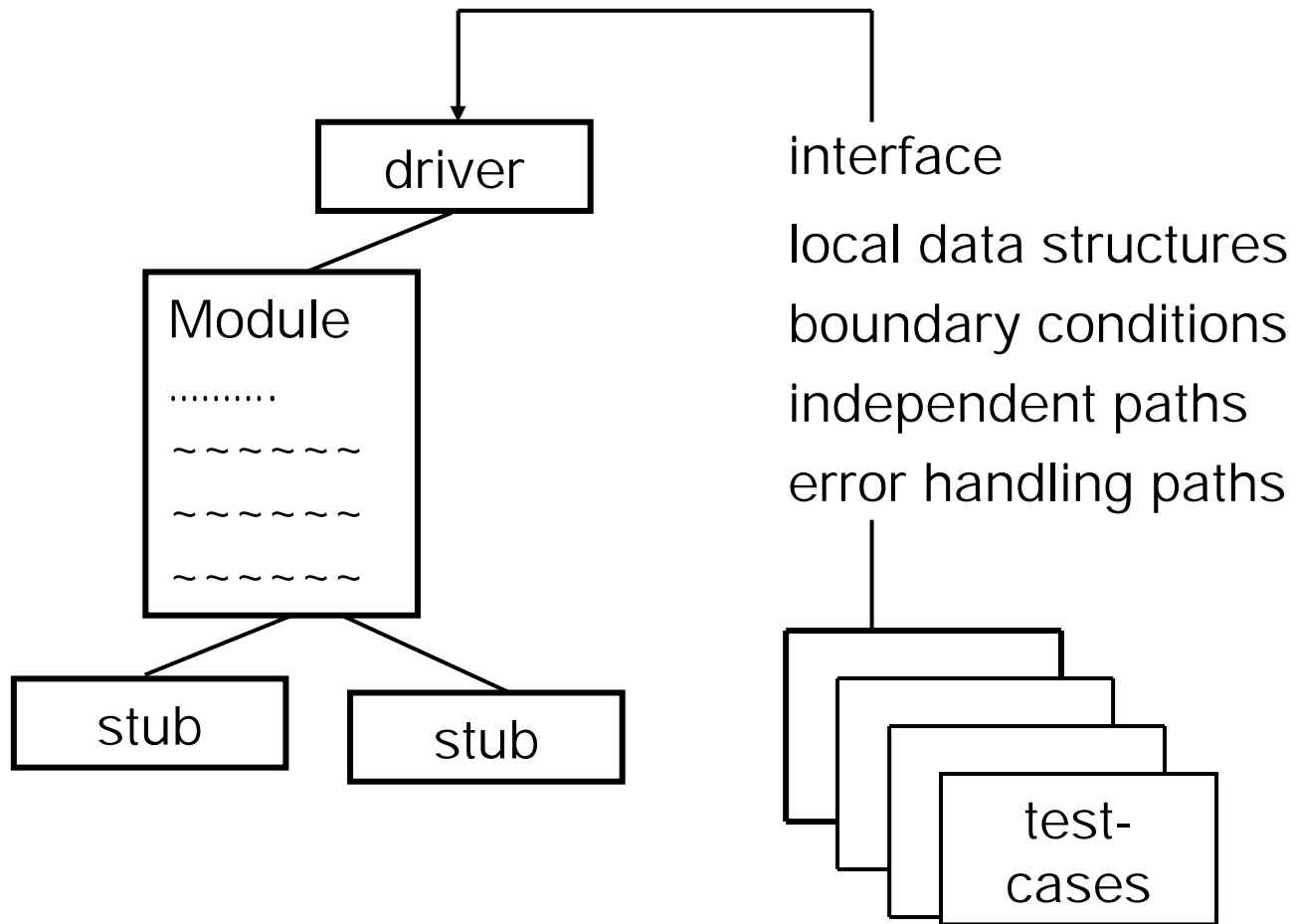
- Bat n'au tai tong *module* ro' tích h'p lon dan nen toan bo'he' thong.
- Cac ky' thua't khac nhau tích h'p tai cac giai n'oa' khac nhau.
- Kiem nghiem co' the' n'oo'c tien hanh bo' ngoo'i phat trien phan mem, nhong no'a' voi cac doi an lon thi viec kiem nghiem phai n'oo'c tien hanh bo' mo' nhom no'c lap.
- Kiem nghiem va' so'a lo'i la' cac hoat n'ong no'c lap nhong viec so'a lo'i phai phu' h'p voi cac chie'n thua't kiem nghiem.



KIỂM NGHIỆM TỔNG *MODULE*

- Tiến hành kiểm nghiệm trên tổng nền và nhân vật của phần mềm, *không là module* mã nguồn, sau khi đã thiết kế mã hóa và biến dịch thành công
- Thông dụng kỹ thuật kiểm nghiệm *white-box*
- Có thể tiến hành kiểm nghiệm cùng lúc nhiều *module*.

KIỂM NGHIỆM TỔNG *MODULE* (t.t)



KIỂM NGHIỆM TỔNG *MODULE* (t.t)

- Mỗi *module* mã nguồn không phải là một chương trình hoàn chỉnh và nó khi phải gọi các *module* chứa nội dung kiểm nghiệm khác \Rightarrow có thể phải thiết lập *driver* và hoặc *stub*: phí tổn khá lớn (70%)
- *Driver* là một chương trình chính còn hiển thị nhận dữ liệu kiểm nghiệm, chuyển dữ liệu đó xuống cho *module* để kiểm tra và in ra các kết quả kiểm tra tổng cộng.
- *Stub* thay thế các *module* nội dung gọi bởi *module* đang kiểm tra.



KIỂM NGHIỆM TÍCH HỢP

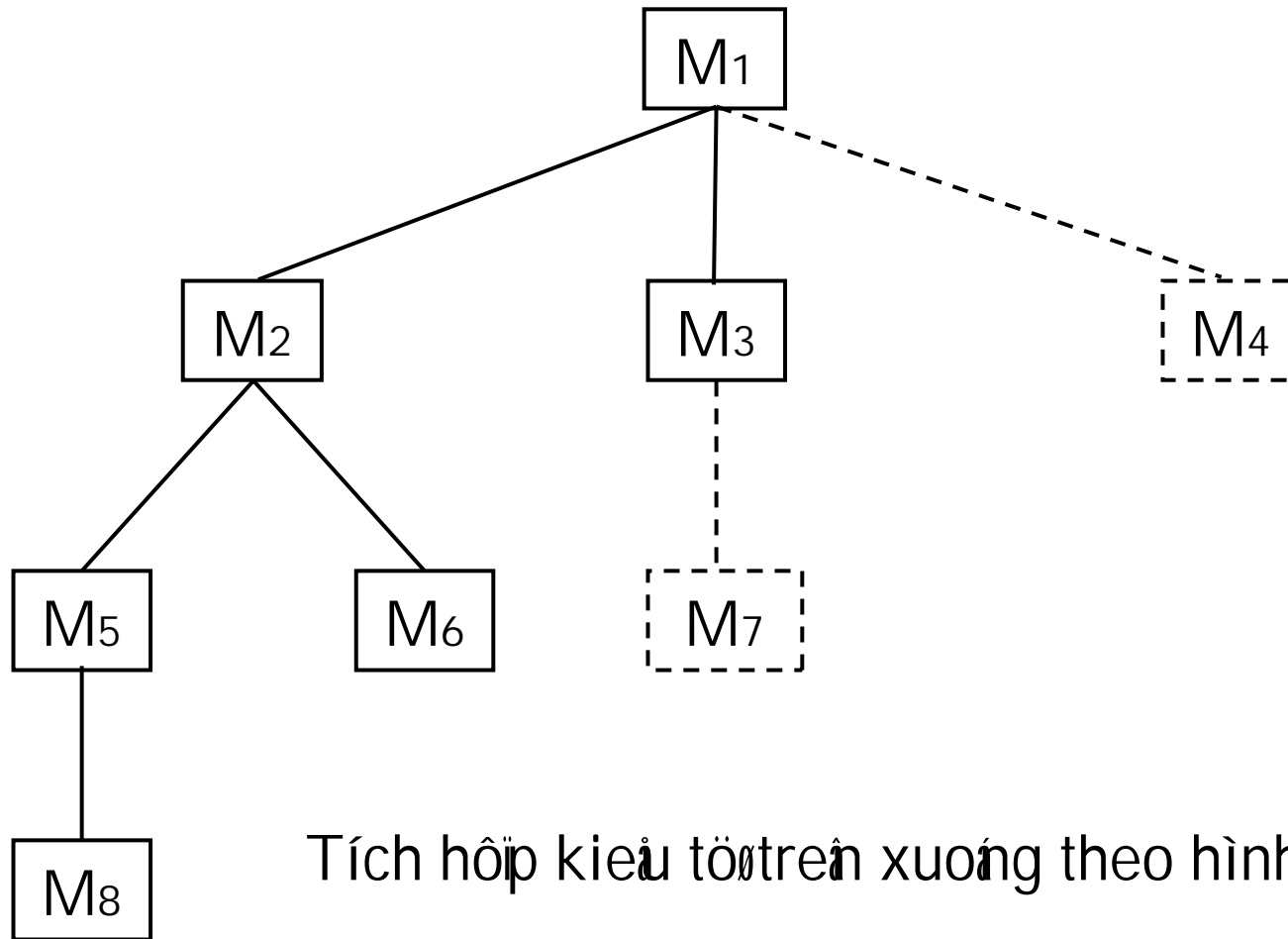
- Tổng *module* mã nguồn nào hoạt động không. Liệu khi kết hợp chúng lại thành một nhóm lớn chúng có hoạt động không ?
- Phải tiến hành kiểm nghiệm tích hợp nếu phát hiện lỗi liên quan đến giao tiếp giữa các *module*.
- Trình tích hợp kiểu *big-bang*: tất cả các *module* đều kết hợp lại, và toàn bộ chương trình sẽ được kiểm nghiệm một lúc
- Nên tích hợp tăng dần: từ trên xuống hoặc từ dưới lên



TÍCH HỘP TỪ TRÊN XUỐNG

- *Module* chính sẽ dùng *driver*, và *stub* sẽ thay thế bởi các *module* con trực tiếp của *module* chính này.
- Tùy thuộc vào cách tích hợp theo chiều sâu (*depth-first*) hoặc chiều ngang (*breadth-first*), mỗi *stub* con sẽ thay thế một lần bởi *module* tổng cộng cần kiểm nghiệm.
- Tiến hành kiểm nghiệm khi có sự thay thế mới
- Tiến hành kiểm nghiệm hồi quy để phát hiện các lỗi khác trong tổng *module*

TÍCH HỘP TÖTTRÊN XUÖNG (t.t)

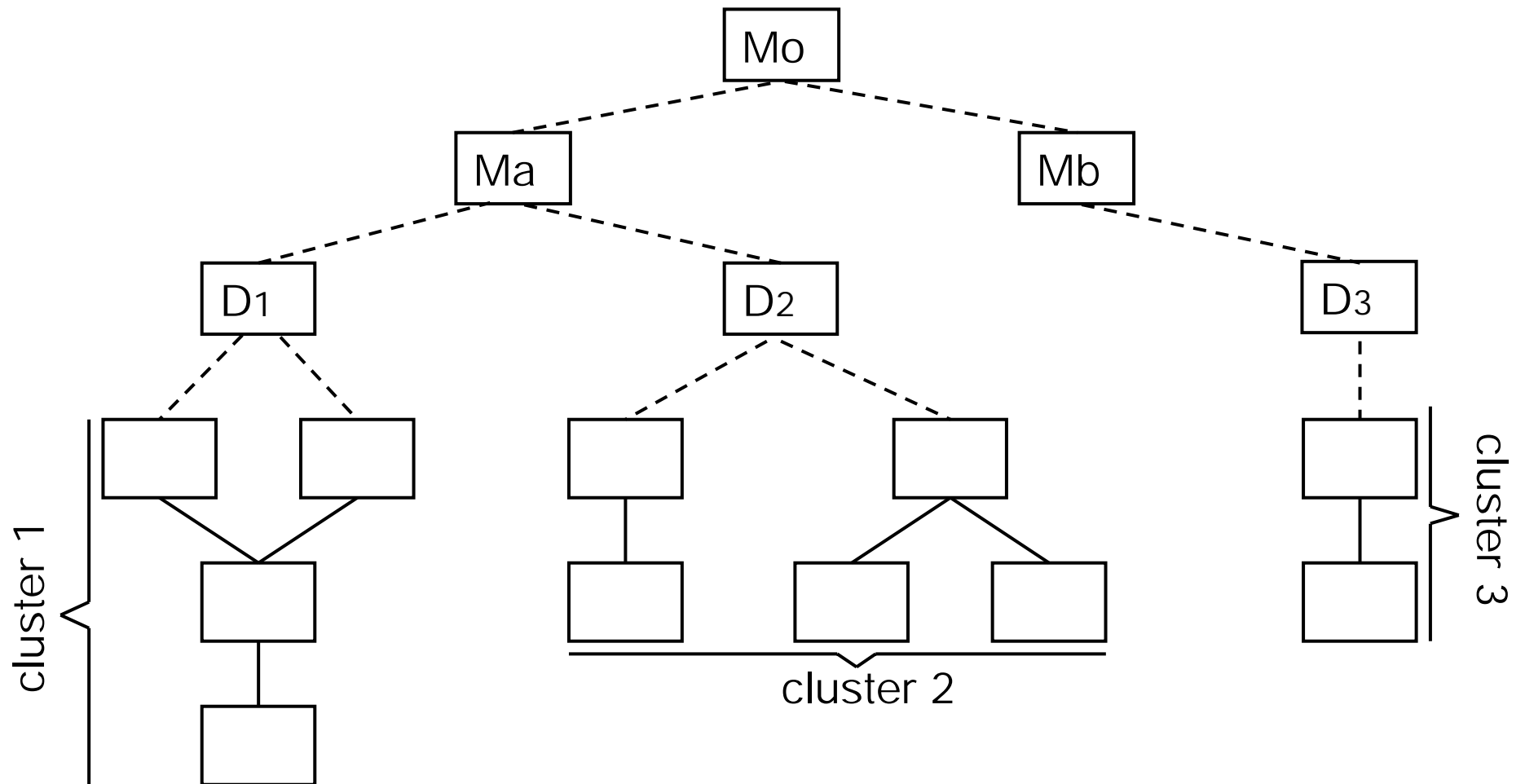


Tích hợp kiểu tốttrên xuống theo hình thức *depth-first*

TÍCH HỢP TỒN ĐỒ LÊN

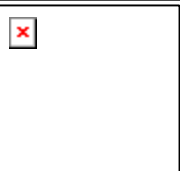
- Các *module* mới tập nhát ñöôc kết hợp thành các nhóm thể hiện một chức năng con ñặc biệt của phần mềm.
- Một *driver* ñöôc tạo ra ñể thao tác các *test-case*
- Nhóm module ñöôc kiểm nghiệm.
- *Driver* ñöôc bổ ñi vào các nhóm *module* ñöôc kết hợp dần lên phía trên trong sơ ñồ phân cấp của chương trình.

TÍCH HỢP TỔ ĐỒ LÊN (t.t)



KIỂM NGHIỆM HOÀI QUY

- Việc kết hợp các *module* lại với nhau có thể ảnh hưởng đến vòng lặp điều khiển, cấu trúc dữ liệu hay I/O chia sẻ trong một số *module*
- Nếu không làm rõ ra một số lỗi không thể phát hiện nếu không tiến hành kiểm nghiệm theo nhu cầu
- Kiểm nghiệm hồi quy có thể được tiến hành thủ công bằng cách thực hiện lại các *test-case* đã tạo ra. Hoặc có thể dùng một công cụ *capture-playback* nếu thực hiện tối ưu



KIỂM NGHIỆM TÍNH NĂNG

- Kiểm nghiệm tính năng hiệu theo cách nôn giận nhất là các chức năng của phần mềm đáp ứng được nhu cầu của khách hàng vốn dĩ được xác định trong văn bản các yêu cầu của phần mềm
- Áp dụng kỹ thuật *black-box*
- Kiểm nghiệm tính năng bao gồm
 - ◆ Xem xét lại cấu hình phần mềm
 - ◆ Kiểm nghiệm *alpha*
 - ◆ Kiểm nghiệm *beta*



KIỂM NGHIỆM TÍNH NĂNG (t.t)

- Kiểm nghiệm *alpha*

- ◆ Nỗ lực tiến hành ngay tại nơi sản xuất phần mềm.
- ◆ Nhà phát triển phần mềm sẽ quan sát người sử dụng sản phẩm và ghi nhận lại những lỗi phát sinh nếu có.

- Kiểm nghiệm *beta*

- ◆ Phần mềm sẽ được kiểm tra bên ngoài phạm vi của nhóm sản xuất.
- ◆ Khách hàng trực tiếp sử dụng và ghi nhận lỗi nếu có để báo lại cho nhà phát triển sửa chữa.



KIỂM NGHIỆM HỒNG NƠI TỒNG

● Về cô ban chiến thuật kiểm nghiệm hồng nơi tổng cũng theo

thời giờ giống như kiểm nghiệm của bạn:

kiểm nghiệm đơn vị - kiểm nghiệm tích hợp - kiểm nghiệm chức

năng - kiểm nghiệm toàn bộ hệ thống



KIỂM NGHIỆM NÔN VÀ HỒÔNG NỮ

- Không thể tách rời tổng tài vụ của nó tổng/lớp nếu kiểm nghiệm
 - ◆ Tài vụ nó ở trong bao trong lớp
 - ◆ Các lớp con có thể *override* một tài vụ nào đó
- Kiểm nghiệm nên vì tổng nó tổng tập trung vào các lớp ⇒ kiểm nghiệm hành vi của lớp



KIỂM NGHIỆM TÍCH HỢP HỒNG ỨT

- Khái niệm sơ đồ phân cấp không còn nhiều ý nghĩa trong chương trình hòng hoá tổng \Rightarrow kiểm nghiệm tích hợp theo cách khác
- Hai hình thức kiểm nghiệm tích hợp hòng hoá tổng
 - ◆ Kiểm nghiệm trên cơ sở *thread*: tích hợp các lớp tạo thành một thread để phục vụ cho một input nào đó của chương trình
 - ◆ Kiểm nghiệm trên cơ sở sử dụng: các lớp *client* sẽ được tích hợp để sử dụng dịch vụ nào đó cung cấp bởi các lớp *server*



KIỂM NGHIỆM THEO KỊCH BẢN

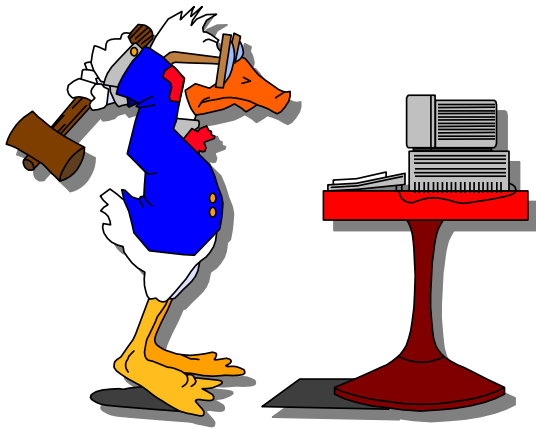
- Đưa vào các *use-case* để soạn ra các kịch bản
- Ví dụ: một kịch bản cho hệ thống nâng cấp môn học qua WEB
 1. Login với *username* = "e59306547", *password* = "6547"
 2. Chọn chức năng nâng cấp môn học
 3. Chọn 5 nhóm môn học của 5 môn: CNPM, AI, XLTHS, PTTK, XLSS trong đó có 2 nhóm trong thời khóa biểu
 4. Nhấn nút SubmitChương trình phải báo lỗi và liệt kê 2 nhóm bỏ trong thời khóa biểu

NGHEATHUAT GOROI

- Goroí la moá qua trính nam loai boi caic loí nööc phát hién trong qua trính kiém tra.
- Goroí nööc thöc hién nhö la moá ket qua cuä viec kiém tra: loí phát hién nööc \Rightarrow tìm kiém nguyên nhân \Rightarrow söa loí
- Coü 3 hình thöc goroí: *brute force*, loai trö nguyên nhân va theo veä. Neän dung ket höp caü 3 hình thöc nay.

NGHỆ THUẬT GỖ ROÁ (t.t)

- Gỗ roá là công việc khoét khoen vào để gây tâm lý chán nản bởi nguyên nhân gây ra lỗi nhiều khi lại mô phỏng do *time-out*, do lỗi chính xác, do chu kỳ quan lập trình...



- Khả năng gỗ roá gần như là bản năng của mọi người

BRUTE FORCE

- Là phương pháp phổ biến nhất nhưng lại ít hiệu quả nhất cho việc phát hiện nguyên nhân gây lỗi phần mềm.
- Triết lý của phương pháp này là “Hãy thử mọi tính tìm ra lỗi”.
- Có 3 cách thực hiện:
 - ◆ Lấy dữ liệu trong bộ nhớ để xem xét.
 - ◆ Dùng *run-time trace* để tìm lỗi.
 - ◆ Dùng lệnh WRITE để xuất dữ liệu cần kiểm tra ra màn hình.
- Áp dụng phương pháp này khi tất cả các phương pháp khác đều thất bại.

LOẠI TRỒNG NGUYÊN NHÂN

- Phương pháp này dựa trên nguyên tắc phân chia nhỏ phân.
- Cách thực hiện:
 - ◆ Khi một lỗi nào đó phát hiện, có gắng nêu ra một danh sách các nguyên nhân có thể gây ra lỗi.
 - ◆ Danh sách này nên được nghiệm lại nếu loại bỏ được các nguyên nhân không xứng cho nên khi tìm thấy một nguyên nhân khả nghi nhất.
 - ◆ Khi nguồn dữ liệu kiểm nghiệm sẽ được tinh chế lại để tiếp tục tìm lỗi.

THEO VEÁT

- Là một phương pháp gỡ lỗi khai phá biến có thể dùng thành công trong các chương trình như những khai phá dùng cho nó với các chương trình rất lớn.
- Cách thực hiện: bắt đầu tại dòng mã nguồn có triệu chứng lỗi thực hiện lần ngược trở lại tổng dòng mã nguồn cho đến khi tìm thấy dòng gây ra lỗi.

KEÁT THÚC MÔN HỌC

Thi cuối kỳ?

Phân tích - Thiết kế -
Hiện thực/triển khai -
Kiểm nghiệm -UML
⇒ Tất cả nội dung

Chúc mừng bạn đã hoàn tất môn học
Công Nghệ Phần Mềm !

