Trường Đại Học Bách Khoa TP.HCM
Khoa CÔNG NGHỆ THÔNG TIN

Bài Giảng

Công Nghệ Phần Mềm

Software Engineering

Giáo viên & Giao tiếp giảng dạy

ThS Nguyễn Cao Trí – ngacotri@gmail.com
http://www.cse.hcmut.edu.vn/~caotri

Room 109 A5 – Trung tâm Kỹ thuật Điện toán
Tel: 8647256 – 5370 Mobile: 091 391 6290
Hobbies: Automation , Flying Model
http://www.rc-easy.net

- Tài liệu download trên website file: TailieudientuCNPM-PrintableVersion.ppt
- Học thế nào? ☐ Hỏi ngay trên lớp
- Bảng mã sử dụng là Unicode dựng sẵn
- Các bài tập nộp bằng email, dạng file *.ZIP
- Email phải ghi rõ nội dung file đính kèm là gì bằng tiếng Việt

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Giới thiệu môn học

- Nội dung môn học
  - Giới thiệu các khái niệm cơ bản về công nghệ phần mềm
  - Mục tiêu của sản xuất phần mềm và công nghệ phần mềm
  - Các mô hình sản xuất phần mềm
  - Quy trình sản xuất và quản lý dự án phần mềm
- Tài liệu tham khảo
  - Introduction to Software Engineering – Ronald J. Leach – CRC Press (Thư viện A2 MS: 9075802004)
  - Software Engineering – Ian Sommerville – Fifth edition (Thư viện A3 MS: 200032)
- Hình thức kiểm tra
  - Giữa kỳ + Cuối kỳ + Bài tập
  - Hình thức kiểm tra: trắc nghiệm khách quan – open book
  - Đánh giá kêt quả: tương đối - phi tuyến

???????? & !!!!!!!!

- Công Nghiệp & Công Nghệ
- Công Nghiệp Phần Mềm (CNpPM)
- Công Nghệ Phần Mềm (CNPM)
- Công nghiệp phần mềm & các công nghiệp khác
  - **Giống**
  - **Khác**
- Có hay không (những) công nghệ cho sản xuất phần mềm?
- Có cần thiết phải có công nghệ cho sản xuất phần mềm không, khi sản xuất phần mềm là hoạt động sản xuất "*đặc biệt*" vì không thể nói làm một phần mềm như sản xuất một lon coca.

Đặc tính của sản phẩm phần mềm

- Software = Program
- Software product = Program + Document + Support
- Loại sản phẩm phần mềm
  - Generic Product: là sản phẩm đóng gói và bán rộng rãi trên thị trường.
  - Bespoke Product: là sản phẩm được phát triển theo yêu cầu đặc thù của từng khách hàng.
- Các đặc tính quan trọng của sản phẩm phần mềm
  - Maintainability: phần mềm có thể thay đổi thuận tiện theo yêu cầu của người dùng
  - Dependability: tính ổn định, bảo mật và an toàn của phần mềm. Không gây tổn hại về vật chất hay kinh thế cho hệ thống.
  - Efficiency: Sử dụng hiệu quả tài nguyên của hệ thống cho công việc
  - Usability: giao diện và phương thức phải phù hợp với người dùng đồng thời đáp ứng đúng yêu cầu của người dùng

Software - Đủ hay Thiếu?

- Phần mềm được viết ngay từ khi có những máy tính programable đầu tiên. Được quan tâm và phát triển từ rất sớm
- Có rất nhiều phần mềm đã được viết

  ☐ **Không thiếu phần mềm**
- Thực tế việc sản xuất phần mềm không đáp ứng kịp yêu cầu của người sử dụng:
  - **Không đủ về số lượng**
  - **Thiếu về chất lượng**
  - **Không kịp về thời gian**

☐ **Phần mềm không đáp ứng đủ cho người dùng**

Nguyên nhân khách quan

- Số lượng phần mềm phải được hiểu là số đầu/loại phần mềm được sử dụng cho từng mục tiêu ứng dụng.
- Nhu cầu sử dụng phần mềm là rất lớn
  - **Nhiều ngành nghề cần dùng phần mềm máy tính**
  - **Mỗi ngành nghề cần nhiều loại phần mềm khác nhau**
  - **Mội loại phần mềm cần nhiều cấp độ khác nhau theo trình độ người dùng**
- Chất lượng phần mềm cũng chưa đáp ứng tốt hoàn toàn người sử dụng:
  - **Tính customize rất cao của sản phẩm phần mềm.**
  - **Trình độ sử dụng khác nhau và điều kiện hạ tầng ứng dụng khác nhau**
- Nhu cầu phần mềm thường rất cấp bách
  - **Tầm nhìn và chiến lược chưa đầy đủ của người sử dụng**
  - **Không có kế hoạch lâu dài**
  - **Phải thay đổi theo từng đối tượng người dùng**

Nguyên nhân chủ quan

- Tính chuyên nghiệp trong sản xuất phần mềm chưa cao

Các dữ liệu quan sát được

- **Cứ 6 đề án triển khai thì có 2 bị huỷ bỏ**
- **Trung bình thời gian thực hiện thực tế bị kéo dài 50 % (cá biệt 200-300%)**
- **Các đề án lớn dễ thất bại**
- **3/4 các hệ thống lớn có lỗi khi thực thi**
- **Quá trình phân tích yêu cầu (5 % công sức): để lại 55 % lỗi, có 18 % phát hiện được**
- **Quá trình thiết kế (25 % công sức): để lại 30 % lỗi, có 10 % phát hiện được**
- **Quá trình mã hoá, kiểm tra và bảo trì: để lại 15 % lỗi, có 72 % phát hiện được**

Nguyên nhân chủ quan (tt)

Lý do của những hệ quả trên

- Phát triển phần mềm giống như một *"nghệ thuật"*, chưa được xem như một ngành khoa học
- Quy trình phát triển phần mềm chưa được thống nhất
- Phải viết lại s/w mỗi khi có sự thay đổi về ngôn ngữ, h/w hoặc o/s
- Chưa đạt được 1 chuẩn cho việc đo lường hiệu suất và sản phẩm
- Độ phức tạp của phần mềm quá cao đối với 1 "kiến trúc sư"
- Kỹ thuật đặc tả để lại sự nhập nhằng trong các yêu cầu phần mềm
- Làm việc nhóm không đúng kỷ luật gây ra các lỗi

- **CẦN PHẢI CÓ MỘT/NHIỀU CHUẨN QUY TRÌNH TRONG SẢN XUẤT PHẦN MỀM ĐỂ NÂNG CAO TÍNH CHUYÊN NGHIỆP CỦA NỀN SẢN XUẤT ĐẶC BIỆT NÀY**
- **CẦN CÔNG NGHỆ CHO CÔNG NGHIỆP PHẦN MỀM**

Định nghĩa "Công nghệ phần mềm"

- **Công Nghệ Phần Mềm** là sự thiết lập và sử dụng các nguyên tắc khoa học nhằm mục đích tạo ra các phần mềm một cách kinh tế mà các phần mềm đó hoạt động hiệu quả và tin cậy trên các máy tính.

- **Công nghệ phần mềm** là một quy trình có hệ thống được sử dụng trong quá trình phân tích, thiết kế, hiện thực, kiểm tra và bảo trì để bảo đảm các sản phẩm phần mềm được sản xuất và hoạt động: hiệu quả, tin cậy, hữu dụng, nâng cấp dễ dàng (modificable), khả chuyển (portable), khả kiểm tra (testable), cộng tác được với các hệ thống khác (interoperable) và vận hành đúng (correct).

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Cụ thể

- Efficiency: Phần mềm được sản xuất trong thời gian và điều kiện vừa phải. Phần mềm vận hành đúng mức độ yêu cầu về công việc và thời gian.
- Reliablity: Phần mềm vận hành ổn định và tương tác được với các hệ thống ứng dụng
- Usability: Phần mềm có thể dùng được bởi người sử dụng và với môi trường mà người sử dụng đang có. Chú ý tới giao diện, điều kiện hệ thống,…
- Modifiability: Phần mềm có thể được thay đổi dể dàng, nhanh chóng khi yêu cầu của người sử dụng thay đổi.
- Portability: Phần mềm có thể chuyển đổi dễ dàng sang các hệ thống khác mà không cần phải điều chỉnh lớn. Chỉ cần recompile nều cần thiết là tốt nhất.
- Testability: Phần mềmcó thể d0ược kiểm tra dễ dàng. Tốt nhất là được modul hóa.
- Reusability: Phần mềm hay một phần có thể được tái sử dụng cho các ứng dụng khác. Các modul có thiết kế tốt, độc lập và giao tiến đơn giản, cả về tình tương thích công nghệ phát triển

Cụ thể (tt)

- Maintainability: thiết kế của phần mềm có thể được hiểu dễ dàng cũng như chuyển giao thuận tiện cho người khác trong quá trình điều chỉnh, nâng cấp hay thay đổi theo yêu cầu.
- Interoperability: Phần mềm vận hành ổn định và đúng như mong đợi. Trên hệ thống nhiều người dùng (multi users) phần mềm vẫn hoạt động được với các vận hành khác của hệ thống.
- Correctness: Phần mềm phải tính toán đúng và tạo ra kết quả đúng và đúng với mục tiêu ứng dụng của người dùng.

- Các yêu cầu khác:
  - **Đúng tiến độ**
  - **Sử dụng hợp lý nguồn nhân lực phát triển**
  - **Chi phí phát triển thấp nhất**

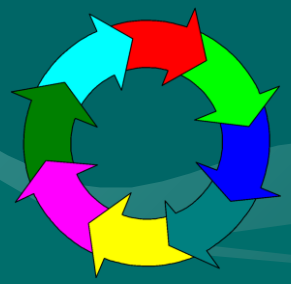
Nội dung công việc của Software Engineering

Công việc của software engineering bao gồm:

- Phân tích hệ thống/vấn đề
- Xác định các yêu cầu
- Thiết kế phần mềm
- Viết phần mềm (coding)
- Kiểm tra và tích hợp hệ thống
- Cài đặt và chuyển giao phần mềm
- Lập tài liệu
- Bảo trì
- Quản lý chất lượng
- Huấn luyện
- Dự đoán tài nguyên
- Quản trị dự án

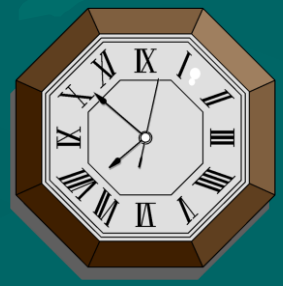
Một định nghĩa khác của CNPM

- CNPM là các quy trình đúng kỷ luật và có định lượng được áp dụng cho sự phát triển, thực thi và bảo trì các hệ thống thiên về phần mềm
- Tập trung vào quy trình, sự đo lường, sản phẩm, tính đúng thời gian và chất lượng



Qui trình

Đo lường
Tiêu chuẩn



Thời gian
Quản lý

Chất lượng
Dịch vụ

Mô hình phát triển phần mềm

Các công đoạn chính tổng quát bao gồm 4 giai đoạn

- Giai đoạn đặc tả: xác định các tính năng và điều kiện hoạt động của hệ thống. (thu thập yêu cầu và phân tích)
- Giai đoạn phát triển: Thiết kế phần mềm (software design), viết code (code generation
- Giai đoạn kiểm tra: kiểm tra phần mềm (software testing), kiểm tra tính hợp lý của phần mềm.
- Giai đoạn bảo trì: Sửa lỗi (correction), thay đổi môi trường thực thi (adaptation), tăng cường (enhancement)

Các mô hình sản xuất phần mềm

Tùy theo quy mô và công nghệ phát triển, có các mô hình sản xuất khác nhau.

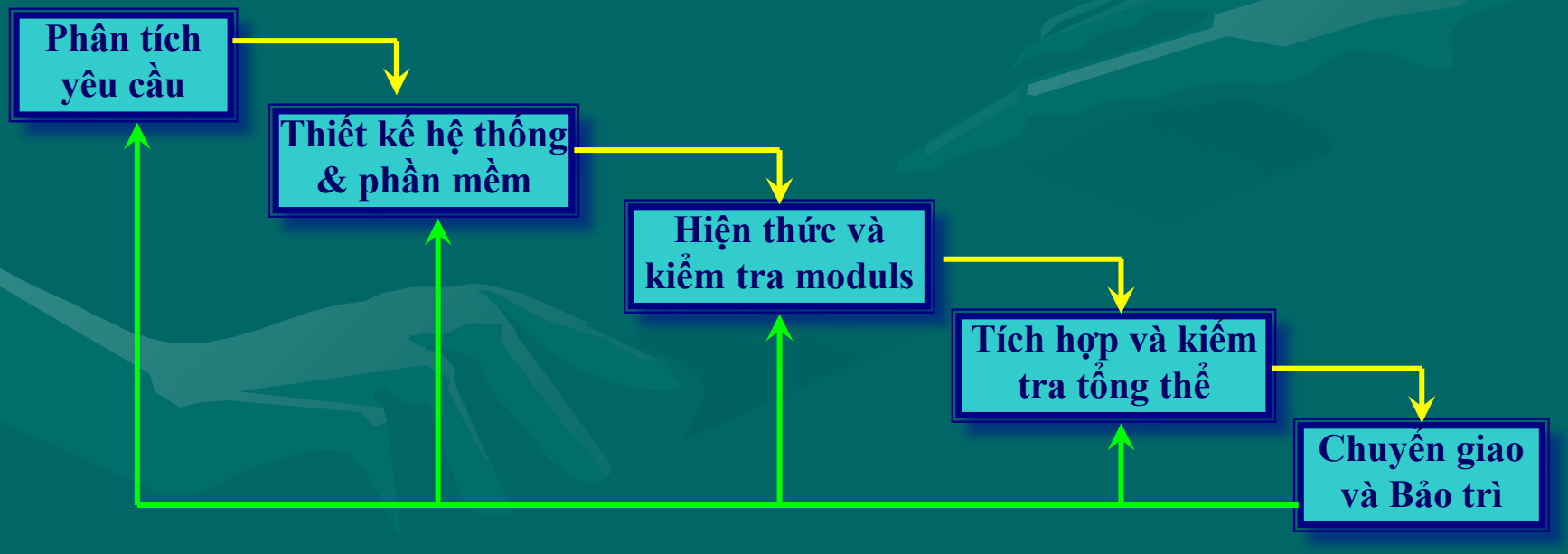
- Mô hình tuần tự tuyến tính- waterfall
- Mô hình Prototyping - Evolutionary Development
- Mô hình xoắn ốc – Boehm's Spiral Model
- Mô hình RAD – Rapid Application Development

MÔ HÌNH NÀO TỐT HƠN

Mỗi mô hình phù hợp với trình độ phát triển, quy mô sản phẩm và yêu cầu ràng buộc cụ thể về thời gian và tính chất của hệ thống.

Mô hình WaterFall – Sequency model

- Mô hình phát triển phần mềm đầu tiên
- Các công việc tiếp nối nhau một cách tuần tự
- Đặt nền móng cho các phương pháp phân tích, thiết kế, kiểm tra…

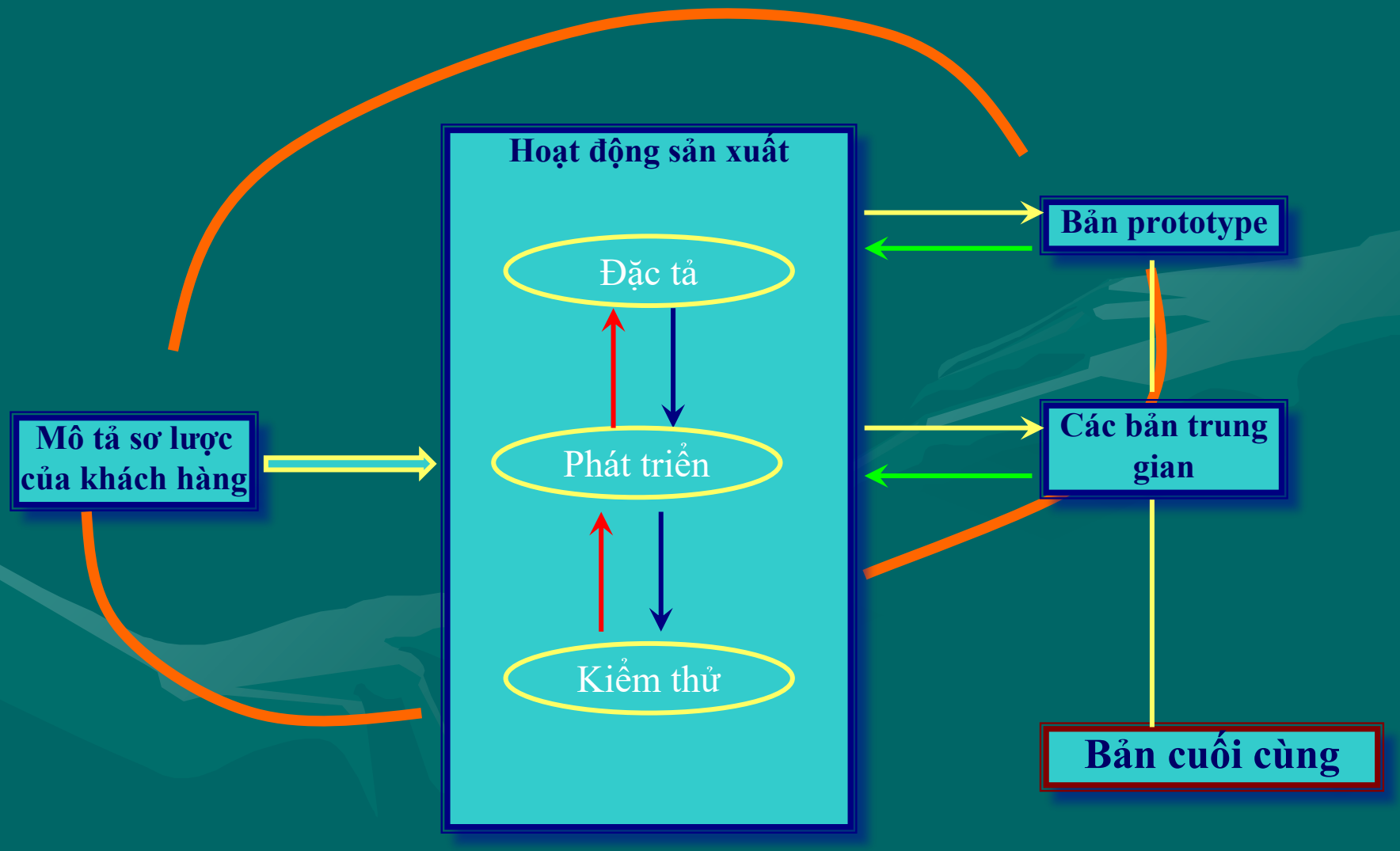


Mô hình WaterFall – Sequency model (tt)

Bộc lộ một số khuyết điểm

- Bản chất của phát triển phần mềm là quá trình lặp đi lặp lại chứ không phải tuần tự
- Các bước thực chất không tách biệt hoàn toàn mà có chồng lấn và tham khảo lại
- Bắt buộc khách hàng đặc tả tất cả yêu cầu một cách chính xác và đầy đủ ngay từ ban đầu
- Khách hàng thường phải chờ đợi rất lâu để thấy được phiên bản đầu tiên của sản phẩm
- Tồn tại "delay" tích lũy trong nhóm làm việc -> dự án thường bị trễ.
- Chỉ phù hợp cho dự án nhỏ, đơn giản.

Mô Hình Prototype



Mô Hình Prototype – ưu & khuyết

- Prototype như là một cơ chế để nhận diện chính xác yêu cầu của khách hàng
  - **Bản thân khách hàng chưa hiểu rõ yêu cầu của mình, cũng như các quy trình chưa được xác lập rõ ràng.**
  - **Khách hàng chưa hiểu rõ khả năng hỗ trợ của hệ thống máy tính**
- Kích thích sự thích thú của người dùng với dự án

- Prototype có thể bị "throw-away" -> Lãng phí
- Các process không được phân định rõ ràng
- Hệ thống thông thường có cấu trúc lỏng lẻo
- Cần có những kỹ năng đặc biệt trong quản lý và phát triển
- Khách hàng hối thúc nhà phát triển hoàn thành sản phẩm một khi thấy được các prototype đầu tiên

Mô Hình Prototype – Ứng dụng

- Dùng cho các hệ thống nhỏ. Các chi phí khi thay đổi hệ thống là không quá lớn khia cần phải thay đổi sau khi thực hiệ prototype
- Cần sự cấp bách về thời gian triển khai ngắn. Hệ thống cần được đưa vào ứng dụng từng phần trong khoảng thời gian nhất định.
- Trong trường hợp những hệ thống mà việc đặc tả các yêu cầu là rất khó và không rõ ràng ngay từ đầu.

Mô hình Xoắn Ốc - Boehm's Spiral Model

- Được thực hiện theo một chuỗi lặp kiểu xoắn ốc, mỗi lần lặp cải thiện sản phẩm
- Có phương pháp đánh giá rủi ro
- Có thể áp dụng prototype
- Mỗi lần lặp được cải thiện cho thích nghi với bản chất của đề án

Mô hình RAD

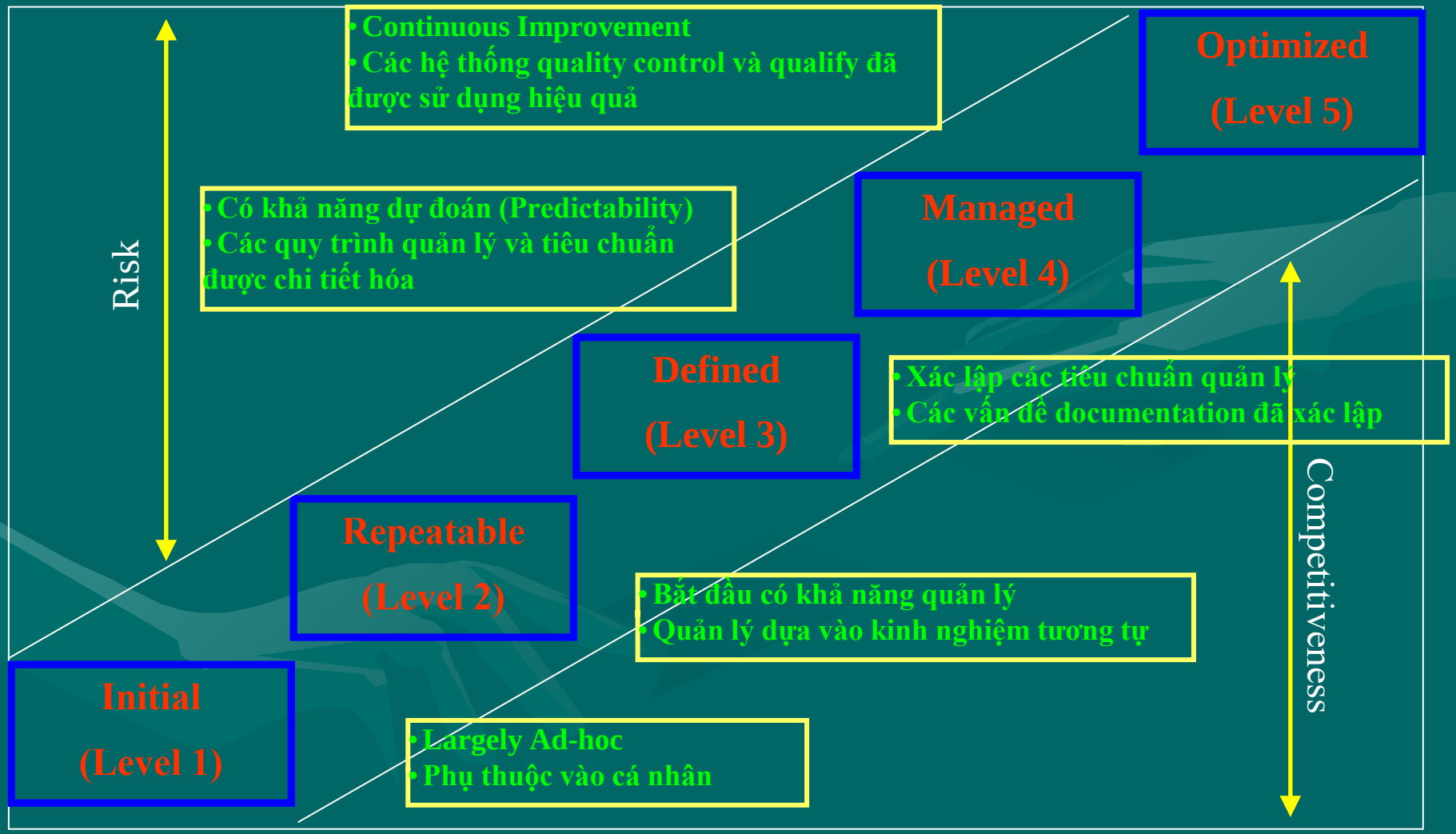
- Rapid Application Development là mô hình tuần tự tuyến tính có thời gian phát triển rất ngắn
- Sử dụng các thành phần có sẵn càng nhiều càng tốt
- Sử dụng công cụ lập trình ở dạng tự động sinh mã chứ không phải các ngôn ngữ truyền thống

- Phụ thuộc vào công nghệ phát triển có tính reusable cao.
- Partten system development

Các tiêu chuẩn dùng trong CNpPM

- The capability Maturity Model (CMM) của Software Engineering Institue (SEI) - Đại học Carnegie Mellon.
  - **Chú trọng đến tính hệ thống và khả năng quản trị của các công ty phần mềm hơn là một quy trình (process) cụ thể.**
- The process Improvement Paradigm (PIP) của Software Engineering Laboratory (SEL) – NASA's Goddard Space Flight Center
  - **Tương tự như CMM, chú trọng đến tính hệ thống và những hướng dẫn để tăng cường tính năng của các quá trình quản lý.**
- Các chuẩn khác của Department of Defense
  - **MIL – STD 2167A ; MIL-STD 1574A ; MIL-STD 882C**
- The electronic Industries Association (EIA) chuẩn SEB-6-A
- The European ESPRIT project
- International Standards Organisation - ISO 9001
- United Kingdom MOD 0055

Chuẩn CMM


Continuous Improvement
Các hệ thống quality control và qualify đã được sử dụng hiệu quả
Optimized
(Level 5)
Có khả năng dự đoán (Predictability)
Các quy trình quản lý và tiêu chuẩn được chi tiết hóa
Managed
(Level 4)
Risk
Defined
(Level 3)
Xác lập các tiêu chuẩn quản lý
Các vấn đề documentation đã xác lập
Competitiveness
Repeatable
(Level 2)
Bắt đầu có khả năng quản lý
Quản lý dựa vào kinh nghiệm tương tự
Initial
(Level 1)
Largely Ad-hoc
Phụ thuộc vào cá nhân

Chương 2

Project Management

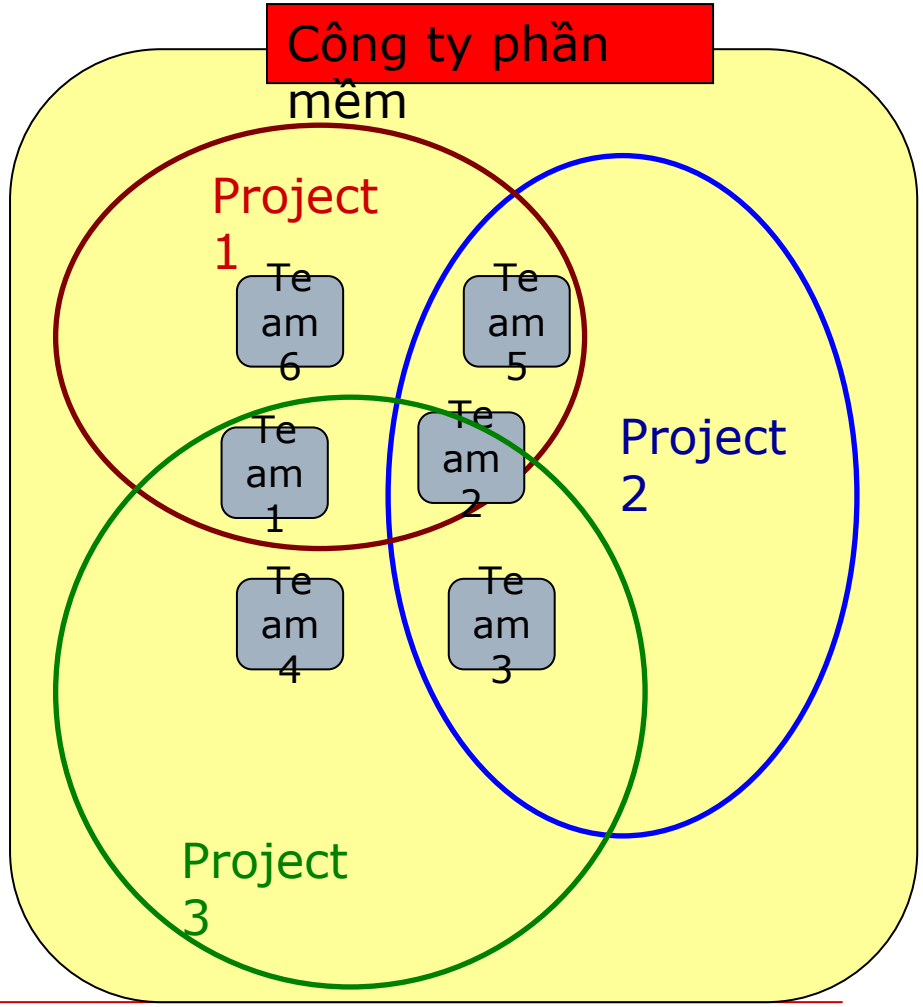
- Sub-Team trong Software Projects
- Project Estimation
- Project Scheduling
- Project Management Tools

Tại sao cần Project management

- Phát triển phần mềm hiện đại làm theo teamworks
- Cần quản lý và kiểm soát được rủi ro (Risk) trong quá trình sản xuất
- Các dự án phần mềm đòi hỏi nhiều nguồn nhân lực với chuyên môn khác nhau
- Tính tích hợp công nghệ cao và sự thay đổi nhanh chóng của công nghệ
- Phải bảo đảm tính chuyên nghiệp trong phát triển dự án phần mềm:
  - Bảo đảm lịch trình của dự án
  - Điều phối và khai thác tối đa nguồn nhân lực hiện có
  - Bảo đảm chất lượng của sản phẩm
  - Khả năng khắc phục các sự cố xảy ra khách quan
- Các dự án càng lớn càng cần có sự quản lý chặt chẻ và đồng bộ

SUB-Team trong software engineering

- Teamwork là mô hình hiện tại cho hầu hết các dự án phần mềm:
  - Khả năng chuyên nghiệp hóa cao
  - Hiệu quả trong quản lý, giao tiếp và điều hành
- Một software project team được tạo ra từ nhiều sub-teams
  - Các sub-team không nhất thiết là một nhóm người mà có thể là 1 người
  - Các sub-team không nhất thiết tồn tại suốt quá trình của một dự án phần mềm



Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Vai trò nhiệm vụ của các SUB Team

Vai trò & nhiệm vụ các Sub-Team

- **System analysis**
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

System Analysis

Xác định tính khả thi của dự án

- Phân tích chi phí (Cost analysis)
- Dự đoán lợi nhận (Estimate revenues)
- Tiền liệu các khó khăn về kỹ thuật và công nghệ

•Sau khi nghiên cứu khả thi, nhóm này sẽ làm việc với Requirement Team để nhận feedbacks

•Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác.

Các Sub-Team

- System analysis
- **Planning Team**
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Planning Team

Nhóm này có nhiệm vụ xây dựng tổng thể tất cả các kế hoạch quản trị dự án và bao đảm các tiến trình diễn ra đúng tiến độ đã định

- Xây dựng các kế hoạch thực hiện
- Lập các time frame cho các tiến trình
- Kế hoạch sử dụng tài nguyên của hệ thống bao gồm cả nhân lực
- Các kế hoạch dự phòng và điều chỉnh khi có sự cố

Các Sub-Team

- System analysis
- Planning Team
- **Requirements Team**
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Requirement Team

Tiếp xúc khách hàng và xác định đầy đủ, hoàn chỉnh và chính xác các yêu cầu cho dự án

•Dùng các phương thức gặp gỡ chính thức và bền lề để xác định các yêu cầu của hệ thống

•Nếu không có khách hàng, có thể tiếp xúc với các user tiềm năng

Sau khi xác định các yêu cầu, nhóm này sẽ làm việc với System Design Team để nhận các feedback.

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- **System Design Team**
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

System Design Team

Xây dựng thiết kế chi tiết của hệ thống sau khi các yêu cầu đã được xác định.

Nếu sử dụng mô hình Waterfall, nhóm này phải feedback cho nhóm Requirement những khó khăn nếu có.

Sau khi hoàn chỉnh thiết kế, nhóm này phải cộng tác với Implementation Team để nhận feedback.

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- **Implementation Team**
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Implementation Team

Phát triển hệ thống theo thiết kế đã có.

- Coding
- Kiểm tra cặp Module

Sau khi hoàn tất chương trình, nhóm này sẽ cộng tác với nhóm Tesing & Integration để kiểm tra các module

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- **Tesing & Intergration Team**
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Testing & Integration Team

Xây dựng thiết kế chi tiết của hệ thống sau khi các yêu cầu đã được xác định.
Nếu sử dụng mô hình Waterfall, nhóm này phải feedback cho nhóm Requirement những khó khăn nếu có.
Sau khi hoàn chỉnh thiết kế, nhóm này phải cộng tác với Implementation Team để nhận feedback.
Nhóm này có thể tiếp nhận các module rời rạc và kiểm tra sau đó tích hợp thành hệ thống hoàn chỉnh.
Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác
Nhóm này cũng có vai trò trong Interface Control Document để đặc tả các giao diện và giao tiếp giữa các thành phần trong hệ thống

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- **Training Team**
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Training Team

Chuẩn bị các công cụ và tài liệu cho việc training cho người dùng

- Kế hoạch training
- Các tài liệu giảng dạy

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- **Delivery & Installation Team**
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Delivery & Installation Team

Nhiệm vụ là cài đặt hệ thống cho khách hàng và các hỗ trợ kỹ thuật trong cài đặt vận hành hệ thống.

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- **Maintenance Team**
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Maintenance Team

Bảo trì hệ thống sau khi chuyển giao và cài đặt

- Cập nhật sửa chữa
- Nâng cấp mở rộng

Cộng tác chặt chẽ với nhóm implementation để thực hiện việc maintenance

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- **Quality Assurance Team**
- Metrics Team
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Quality Assurance Team

Nhóm này có 2 nhiệm vụ

1. Thiết lập các tiêu chuẩn cho các quá trình sản xuất cũng như tiêu chuẩn thực hiện của sản phẩm phần mềm
2. Cung cấp các cơ chế kiểm tra, kiểm soát nhằm đánh giá khả năng thỏa mãn các tiêu chuẩn tương ứng của các nhóm làm việc.

Các tiêu chuẩn này dùng trong nội bộ và không chia sẻ với khách hàng. Các tiêu chuẩn có thể được công bố khi cần thiết, vì vậy cần được lưu trữ và báo cáo cho project manager để hoạt động với bộ phận QBA

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- **Metrics Team**
- Documentation Team
- System Administration Team
- Reuse & Reengineering Team

Metrics Team

Lưu trữ các thông tin thống kê về các hoạt động của các TEAm trong dự án.

- Số lượng các yêu cầu maintenance
- Số lượng thực hiện dịch vụ maintenance
- Số dòng code được viết
- Thời gian thực hiện từng công việc

Nhóm này làm việc với hầu hết các nhóm để cung cấp báo cáo về chất lượng, hiệu quả, đồng thời feedback cho các nhóm đó về hiệu quả công việc.

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- **Documentation Team**
- System Administration Team
- Reuse & Reengineering Team

Documentation Team

Nhóm này thực hiện các hoạt động thiết lập các tài liệu cho hệ thống

- Tài liệu về phân tích, thiết kế, hiện thực, source code,..
- Tài liệu hỗ trợ : userguide, manual, support document

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- **System Administration Team**
- Reuse & Reengineering Team

System Administration Team

Nhóm này có nhiệm vụ cung cấp và bảo đảm các hoạt động của các hệ thống hạ tầng kỹ thuật cần thiết cho dự án

Nhóm này thông thường bao gồm cả Network Administration Team

Các Sub-Team

- System analysis
- Planning Team
- Requirements Team
- System Design Team
- Implementation Team
- Tesing & Intergration Team
- Training Team
- Delivery & Installation Team
- Maintenance Team
- Quality Assurance Team
- Metrics Team
- Documentation Team
- System Administration Team
- **Reuse & Reengineering Team**

Reuse & Reengineering Team

- Chọn lựa và quyết định việc reuse các module đã có
- Việc reengineering cũng cần thiết khi mà việc phát triển đòi hỏi phải dùng đến các code cũ khi công nghệ đã thay đổi.

Sub-Team (tt)

- Có rất nhiều việc quản lý trong từng nhóm và từng công đoạn.
- Có khá nhiều nhóm -> nhiều manager, tuy nhiên nếu các nhóm này nhỏ thì thường là một người trong nhóm sẽ là manager của team.
- Việc scheduling giữa các team phụ thuộc vào mô hình phát triển cụ thể là gì.
- Ví dụ nếu dùng mô hình Waterfall thì các công đoạn có thể được tích hợp thành các bước lớn hơn, các nhóm tham gia vào từng bước theo chức năng của mình.

Các nhóm trong mô hình Waterfall

SPECIFICATION: Planning, System Analysis, QA, Metrics, Documentation, System Administration, Reuse & Reengineering

DESIGN: QA, Metrics, Documentation, System Administration, Reuse & Reengineering

CODE:QA, Metrics, Documentation, System Administration, Reuse & Reengineering

TESTING & INTEGRATION: QA, Metrics, Documentation, System Administration, Reuse & Reengineering

MAINTENANCE: Trainning, Delivery & Instalation,QA, Metrics, Documentation, System Administration, Reuse & Reengineering

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Các nhân sự khác trong dự án

Bên cạnh còn có một số nhân sự khác tham gia vào quá trình phát triển dự án nhưng có thể không được nêu tên một cách chính qui

- **Human-Computer Interface Evaluation**: Đánh giá khả năng thích hợp của giao diện cả như người dùng cấp thấp và cấp cao
- **Tools Support Person**: Người cung cấp và bảo đảo các công cụ cần thiết như tools, software, network vận hành theo yêu cầu của quá trình phát triển
- **Software Economist**: Sử dụng các mô hình đánh giá cần thiết để ước lượng chi phí phần mềm, phần cứng, resource và thời gian cần cho dự án hoàn tất
- **Project Librarian**: Có trách nhiệm lưu trữ và sắp xếp hệ thống tất cả các tài liệu của dự án
- **Chuyên gia hỗ trợ**: Một số dự án cần có những chuyên gia trong lĩnh vực tương ứng hổ trợ, tư vấn về mặt chuyên môn hay kỹ thuật

NHÂN SỰ CỦA CÁC TEAM CÓ THỂ THAY ĐỔI THƯỜNG XUYÊN TRONG QUÁ TRÌNH HOẠT ĐỘNG DO NHIỀU YẾU TỐ

Các yếu tố ảnh hưởng đến các team

- Nhân sự cần thay đổi theo từng công đoạn: các công đoạn cần nhiều nhân sự và cần thời gian dài như coding, testing & integration.
- Các nguyên nhân khách quan khác:
  - Nhân sự thay đổi công việc: chuyên môn thay đổi, công nghệ mới cập nhật
  - Nhân sự nghĩ do thay đổi việc, bệnh, về hưu
  - Nhân sự mới: mang lại tư duy mới và công nghệ mới tuy nhiên phải cần thời gian tiếp cập
- Việc xây dựng các team là linh động theo từng dự án và cần có điều phối giữa các dự án theo từng tiến độ công việc.

Project management

- Dự đoán quy mô và độ phức tạp của dự án
- Xác định các team cần thiết cho hiện thực dự án
- Xác định kế hoạch dự đoán thời gian hoàn thành dự án
- Xác định các tài nguyên cần thiết cho dự án bao gồm phần mềm, hệ thống, ….
- Tính toán chi phí xây dựng dự án
- Xây dựng lô trình thực hiện dự án (smiletone)
- Thực hiện các công việc quản lý trong thời gian thực hiện dự án để bảo đảm đúng kế hoạch đã đề ra. ☐

Các công việc của Project management trong thời gian thực hiện dự án

- Quản trị nhân sự: điều phối, quản lý công việc,..
- Phân bổ các tài nguyên của hệ thống theo kế hoạch
- Điều phối nhân sự: trong công ty và bên ngoài
- Xử lý các phát sinh về thời gian biểu
- Quản lý các thay đổi yêu cầu của dự án
- Giải quyết các sự cố ngoài kế hoạch: máy móc hư hỏng, nhân sự thay đổi,..
- Báo cáo cho lãnh đạo về dự án
- Giao tiếp với khách hàng
- Staffs trainning

Software Project Estimation

- Dự đoán điều gì?
  - Quy mô của sản phẩm sẽ được phát triển
  - Các yêu cầu, resource cần thiết để có thể phát triển sản phẩm thành công
- Để phát triển sản phẩm cần biết nhiều thông tin

- **Máy tính cần cho phát triển**
- **Các phần mềm cơ bản như compiler**
- **Phương thức giao tiếp giữa các bộ phận**
- **CASE Tools**
- **Các gói phần mềm mà hệ thống cần liên kết, tương tác.**
- **Máy tính cho tesing**
- **Máy tính cho trainning**
- **Các công cụ cho Documentation**
- **Thiết bị copy, lưu trữ**
- **Công nghệ sử dụng**
- **Lập trình viên**
- **Kiểm tra viên**
- **Quản lý**
- **Nhà thiết kế**
- **Các nhân sự khác**
- **.....**

Size of Project

Project Size

- Kích thước được đo bằng "lines of code"
- Làm thế nào để dự đoán được kích thước của một đề án phần mềm?
  - Kính thước đo bằng số dòng code được viết để hoàn thành dự án
  - Dự án chưa thực hiện làm sao biết số dòng code sẽ được viết
- Phương pháp Analogy và Reasoning-by-Analogy
- Phân bố nhân lực cho dự án phần mềm trên cơ sở lines of code đã dự đoán.
  - Đánh giá năng xuất của mỗi người trong dự án
- Metric data cho vấn đề estimated và phân bổ resource.

Phương pháp Analogy

- Analogy là gì ? Suy luận theo analogy?
- Điều kiện để sử dụng phương pháp analogy:
  - Có **bài toán đã biết** có bản chất tương tự
  - Đã có giải pháp của **bài toán đã biết**
- Cách áp dụng analogy
  - Xác định được các thông số tương ứng giữa 2 bài toán
  - Áp dụng giải pháp của bài toán đã biết cho bài toán mới
- Với dự án phần mềm
  - Xác định các dữ liệu của những vấn đề đã làm nhờ vào metric
  - Phân tích dự án cần dự đoán thành các thành phần
  - Xác định kích thước tương ứng từng phần với thông tin metric đã có loại trừ các kích thước các phần được reuse
  - Tổng kích thước các phần **kích thước dự án**

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Dự đoán nhân lực cho dự án

Lines of new code	Approximate Number of Software Enginneers
5,000	1
10,000	1
20,000	2
50,000	5
100,000	10
200,000	20
500,000	50
1,000,000	100
2,000,000	200
5,000,000	500
10,000,000	1,000
100,000,000	10,000

- Thực tế không thể tăng tuyến tính một cách đơn giản như thế
- Không phải tất cả nhân lực đều đồng đều.
- Khi dự án càng lớn thì các vấn đề quản lý giao tiếp và liên kết sẽ phức tạp hơn làm giảm năng xuất.
- Dự án lớn, các thay đổi ngoài dự kiến sẽ nhiều hơn nên năng xuất bị ảnh hưởng.
- Còn nhiều khâu khác ảnh hưởng đến dự án chứ không phải chỉ có coding.

Tính dựa trên thông số nào?

Dự đoán nhân lực theo Metric

Lines of new code	Approximate Number of Software Enginneers
5,000	7
10,000	14
20,000	27
50,000	77
100,000	144
200,000	288
500,000	790
1,000,000	1,480
2,000,000	3,220
5,000,000	8,000
10,000,000	15,960
100,000,000	160,027

- Thiết lập bảng dự liệu theo số liệu thống kê mà metric team đã thực hiện từ trước.
- Số liệu này chứa đựng các yếu tố khác về quản lý và các công đoạn khác nhau của một đề án tổng thể.

- Xây dựng một hàm dự đoán nhân lực cho dự án (person- month)dạng

$$y = mx + b$$

với m, b được xác định bằng phương pháp bình phương tối thiểu
(Xem công thức tính m, b ở trang 71)
Dựa vào đó ta tính được số nhân lực cần thiết cho dự án.
Y thời gian thực hiện (person month)
X kích thước dự án

Ví dụ

Dựa vào bảng trên ta tính được $y = 0.002x + 1.84$.
Ví dụ với dự án kích thước 15,000 lines of code ta sẽ cần 32 person-month để giải quyết

- Xây dựng bảng metric của các dự án tương tự ta tính được là cần phân bổ bao nhiêu người làm trong bao lâu.
- Ví dụ bảng tính toán cho các dự án database

Projects	Domain	Months	Effort (person-month)	Size
Application 1	Graphics Utility	12	30	5000
Application 2	Graphics Utility	10	40	8000
Application 3	Graphics Utility	24	30	10000
Application 4	Graphics Utility	36	100	20000
Application 5	Graphics Utility	12	30	5000
Application 6	Graphics Utility	24	30	10000
Application 7	Graphics Utility	48	90	25000

HẠN CHẾ CỦA PHƯƠNG PHÁP NÀY

COCOMO Model

- Xác định cả 2 thông số là số nhân lực và thời gian phát triển
- Trình tự thực hiện
  - Phân tích các yêu cầu của dự án
  - Xác định line of code của từng yêu cầu dự vào metric data
  - Trừ các phần đã được xác định là reuse code
  - Tổng các phần còn lại tính được **K** line of code
  - Áp dụng công thức tính
  - $E = a_b * K * \exp(b_b)$
  - $D = c_b * E * \exp(d_b)$

E là số nhân lực tham gia vào dự án

D là thời gian thực hiện dự án

COCOMO Model (tt)

- Các hệ số của công thức COCOMO

Software Project type	Ab	Bb	Cb	Db
Small project, experienced team, flexible requirement	2.4	1.05	2.5	0.38
Hard real-time rew\quirements and strict interoperability	3.6	1.2	2.5	0.32
A mixture of the other two type of projects	3.0	1.12	2.5	0.35

- Các giá trị E và D tính được phụ thuộc vào khả năng estimate giá trị line of code (K) của dự án.

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

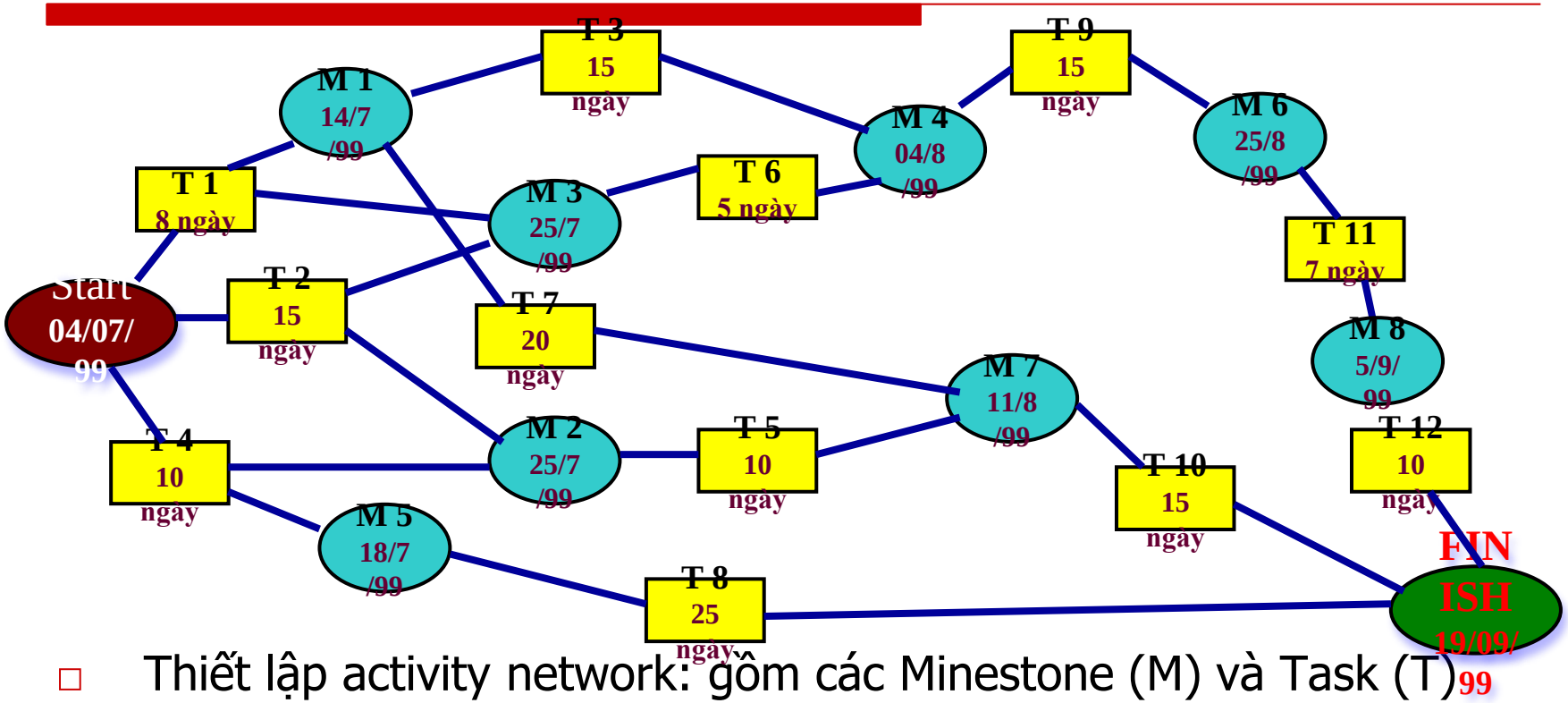
Project Scheduling

- Scheduling bao gồm:
  - Xác định các mốc tiến trình thực hiện dự án
  - Phân bổ resource cho các tiến trình của dự án
- Quản trị và thực hiện các điều chỉnh cần thiết cho các tiến trình khi có sự thay đổi ngoài kế hoạch
  - Phân bổ lại nguồn resource (thêm người)
  - Phân bổ lại milestone của từng tiến trình
  - **Mục tiêu là bảo đảm deadline không bị thay đổi**
- Phương pháp quản trị và điều chỉnh:

FUNCTION POINT

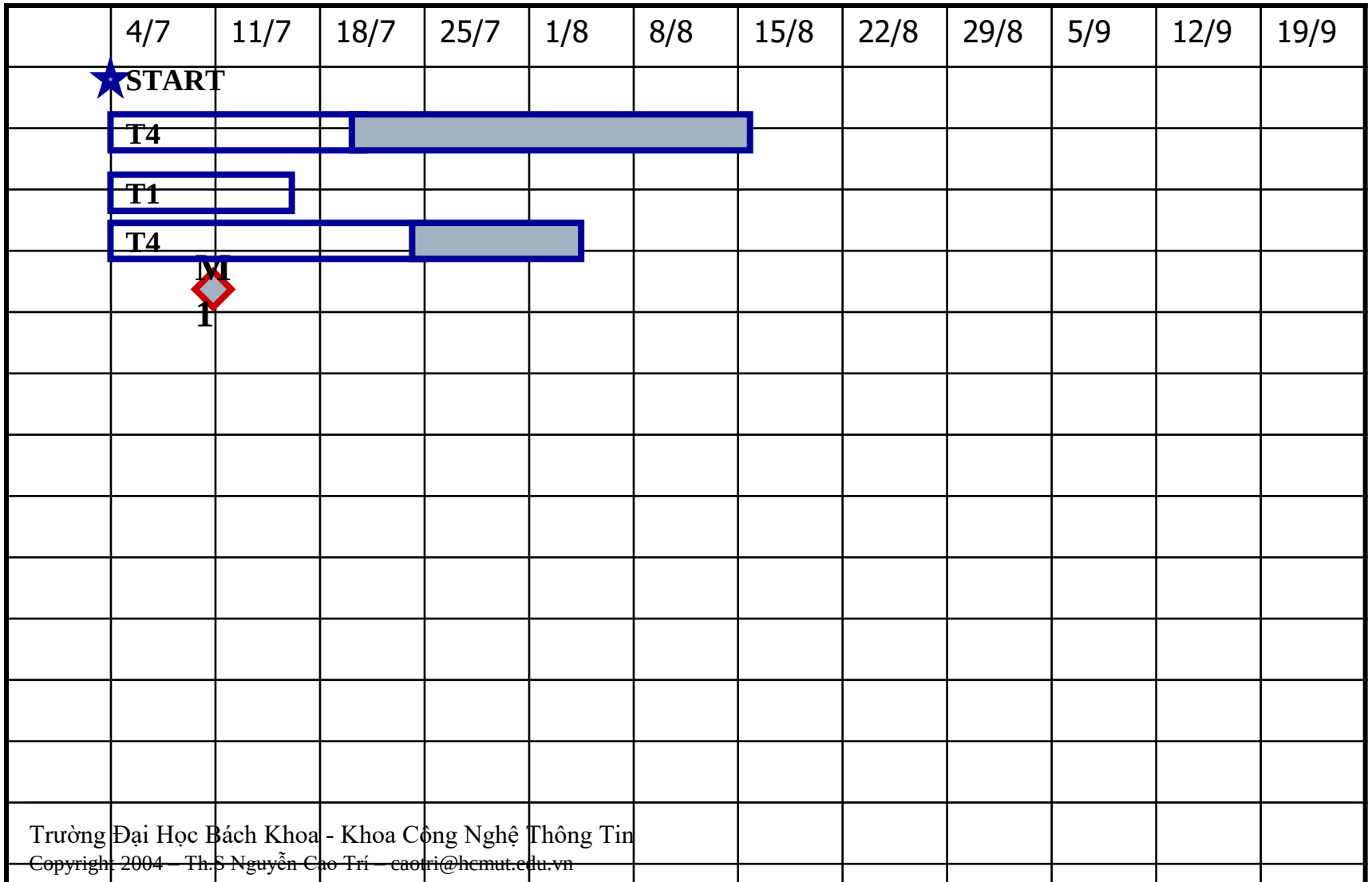
- Các công cụ dùng trong scheduling : MS project, Cocomo2

Activity network model



- Thiết lập activity network: gồm các Minestone (M) và Task (T)
- Xác định critical path: có tổng thời gian thực hiện dài nhất
- Điều chỉnh các M-i để bảo đảm deadline
- Điều chỉnh các T-i bằng cách thay đổi/bổ sung nhân sự (Team)

Activities Bar Chart –PERL chart



Chương 3

Phân tích hệ thống

(system analysis)

- Những vấn đề trong phân tích hệ thống
- Thu thập yêu cầu từ người sử dụng
- Phân tích yêu cầu
- Xác định tính năng hệ thống

Mục tiêu của phân tích hệ thống

- 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 tìm hiểu, phân tích và kiểm chứng lại (validate) yêu cầu và biểu diễn nó bằng mô hình phân tích
- Mô hình phân tích đặc tả toàn bộ nội dung : chức năng, dữ liệu nhập/xuất, các hoạt động của hệ thống cần phát triển
- Xây dựng các từ điển dữ liệu định nghĩa các khái niệm đặc thù của hệ thống, ý nghĩa, cấu trúc,…
- Thống nhất với khách hàng về mô hình và tính năng của hệ thống

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Phân tích hệ thống

- Phân tích hệ thống là bước đầu tiên rất quan trọng cho dự án phát triển phần mềm
- Công việc phân tích hệ thống bao gồm
  - **Thu thập yêu cầu và quy trình nghiệp vụ hiện tại**
  - **Phân tích và xác lập các quy trình sẽ được phát triển/thay thế bằng máy tính**
  - **Xác thực các yêu cầu/tính năng của hệ thống**
- Kết quả của việc phân tích hệ thống là các tài liệu đặc tả tính năng hệ thống. Các tài liệu này thông thường ở dạng các sơ đồ, biểu đồ,..
- Kết quả này dùng cho việc xác thực các tính năng của hệ thống với khách hàng
- Kết quả này là đầu vào của quá trình tiếp theo là thiết kế hệ thống.
- Tùy thuộc vào công nghệ phát triển mà sử dụng các phương pháp phân tích phù hợp : cấu trúc hay OOP

Những vấn đề trong phân tích hệ thống

- Cách biệt về chuyên môn của lĩnh vực cần phân tích
- Sự hiểu biết của những người end user về quy trình làm việc và khả năng ứng dụng phần mềm cho công việc của họ
- Những vấn đề về điều kiện hạ tầng hổ trợ hoạt động của hệ thống
- Tính sẳn sàng thông tin của các hệ thống đang có sẽ tương tác với hệ thống cần xây dựng
- Định hướng ứng dụng lâu dài chưa có/ chưa rõ ràng
- Công cụ/ngôn ngữ sử dụng để đặc tả hệ thống / kết quả phân tích

Quy trình phân tích hệ thống

- Các bước chính
  - **Thu thập thông tin hệ thống hiện tại**
  - **Thu thập yêu cầu**
  - **Phân tích yêu cầu**
  - **Xác lập tính năng hệ thống**
  - **Xác thực tính năng hệ thống**

Tìm hiểu và xây dựng lại hiện trạng của hệ thống

- Các quy trình hoạt động/nghiệp vụ
- Phương thức và ý nghĩa của các quá trình xử lý
- Dữ liệu của hệ thống
- Điều kiện hạ tầng: thiết bị, con người

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Quy trình phân tích hệ thống

- Các bước chính
  - **Thu thập thông tin hệ thống hiện tại**
  - **Thu thập yêu cầu**
  - **Phân tích yêu cầu**
  - **Xác lập tính năng hệ thống**
  - **Xác thực tính năng hệ thống**

Xác định các yêu cầu

- Các yêu cầu về chức năng của hệ thống
- Các yêu cầu về môi trường vận hành: thiết bị, con người

Quy trình phân tích hệ thống

- Các bước chính
  - **Thu thập thông tin hệ thống hiện tại**
  - **Thu thập yêu cầu**
  - **Phân tích yêu cầu**
  - **Xác lập tính năng hệ thống**
  - **Xác thực tính năng hệ thống**

Phân tích các yêu cầu

- Phân tích các yêu cầu theo quy trình sử lý
- Bổ sung các quy trình cho phù hợp với máy tính
- Yều cầu bổ sung các thông tin

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Quy trình phân tích hệ thống

- Các bước chính
  - **Thu thập thông tin hệ thống hiện tại**
  - **Thu thập yêu cầu**
  - **Phân tích yêu cầu**
  - **Xác lập tính năng hệ thống**
  - **Xác thực tính năng hệ thống**

Xác lập tính năng của hệ thống

- Xác lập các chức năng mà hệ thống sẽ bao gồm
- Xác lập các điều kiện và môi trường hoạt động

Quy trình phân tích hệ thống

- Các bước chính
  - **Thu thập thông tin hệ thống hiện tại**
  - **Thu thập yêu cầu**
  - **Phân tích yêu cầu**
  - **Xác lập tính năng hệ thống**
  - **Xác thực tính năng hệ thống**

Xác thực tính năng hệ thống

- Xác thực với người dùng về tính hợp lý và đầy đủ của các tính năng
- Xác thực các quy trình nghiệp vụ
- Xác thực các ràng buộc

Quy trình phân tích hệ thống

- Các bước chính
  - Thu thập thông tin hệ thống hiện tại
  - Thu thập yêu cầu
  - Phân tích yêu cầu
  - Xác lập tính năng hệ thống
  - Xác thực tính năng hệ thống

Phương pháp & Công Cụ

Phương pháp cấu trúc

Các bước được thực hiện đồng thời và sen kẽ nhau
Thường sử dụng lược đồ: DFD, ERD, STD

Phương pháp OOP

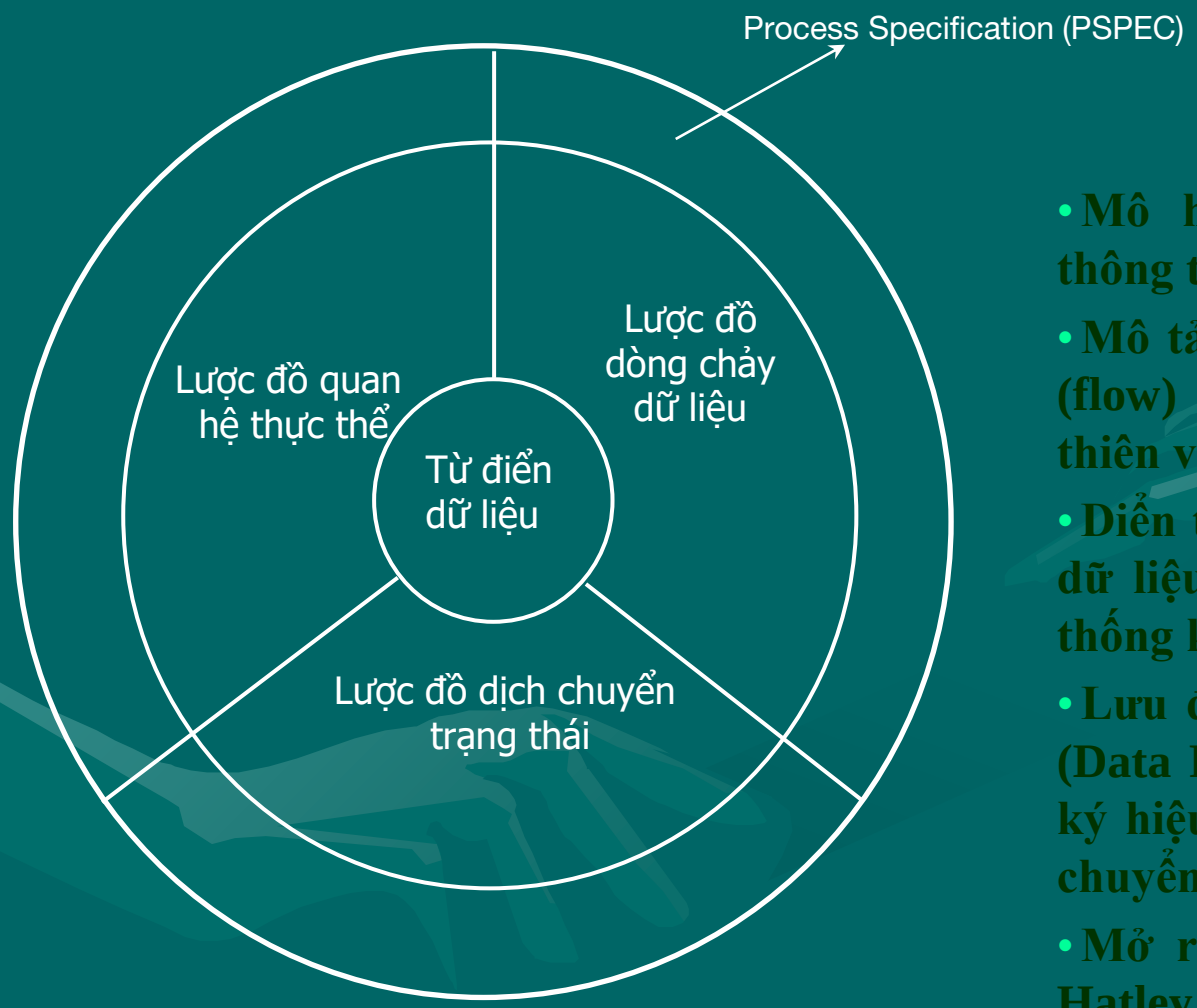
Sử dụng UML: lược đồ Use case, Class

Phân tích hệ thống theo hướng phát triển kỹ thuật lập trình cấu trúc

Tiếp cận của phương pháp phát triển cổ điển cho bước phân tích hệ thống

Các lược đồ DFD, STD, ERD

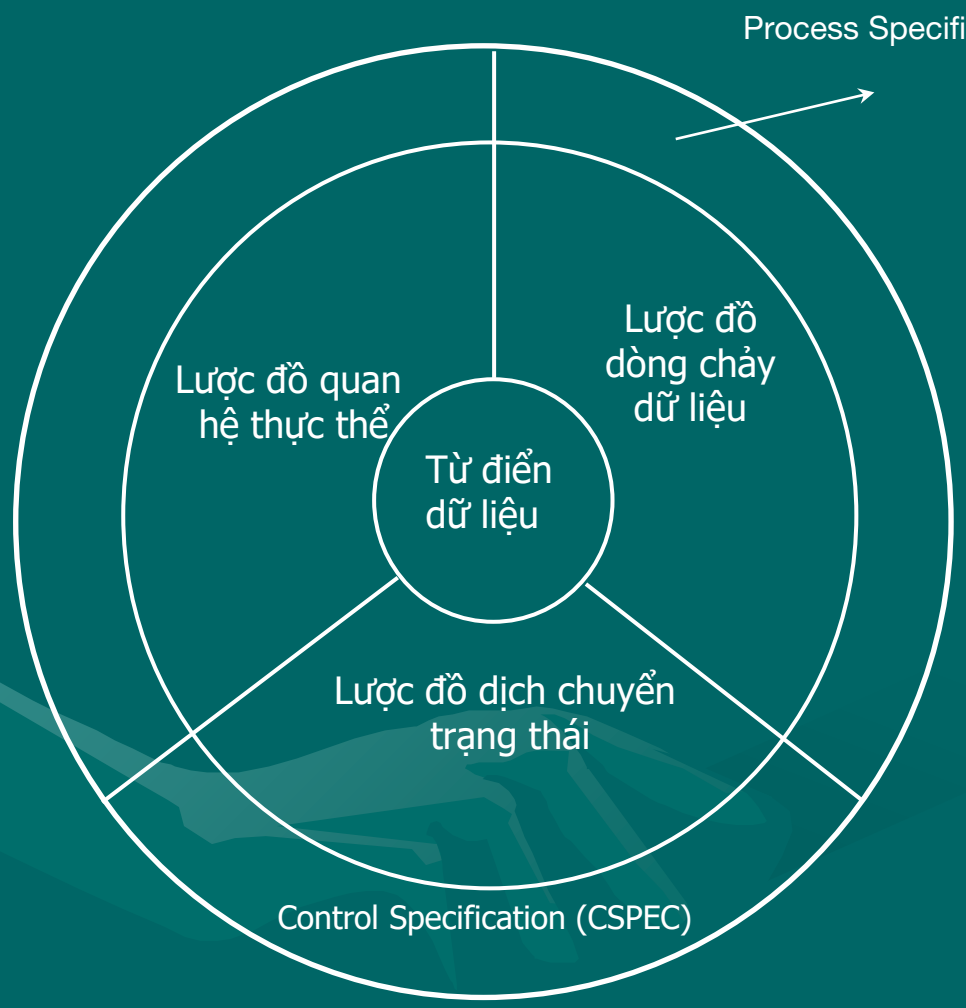
CÁC YẾU TỐ CĂN BẢN CỦA MÔ HÌNH



Lược đồ DFD

- Mô hình chức năng và dòng thông tin: DFD, PSPEC
- Mô tả dòng thông tin di chuyển (flow) xuyên qua các hệ thống thiên về phần mềm.
- Diễn tả các tương tác xuất nhập dữ liệu với con người và các hệ thống khác
- Lưu đồ dòng chảy dữ liệu DFD (Data Flow Diagram) cung cấp 4 ký hiệu cơ bản để mô hình sự di chuyển của dòng thông tin
- Mở rộng của Ward & Mellor; Hatley & Pirbhai cho realtime

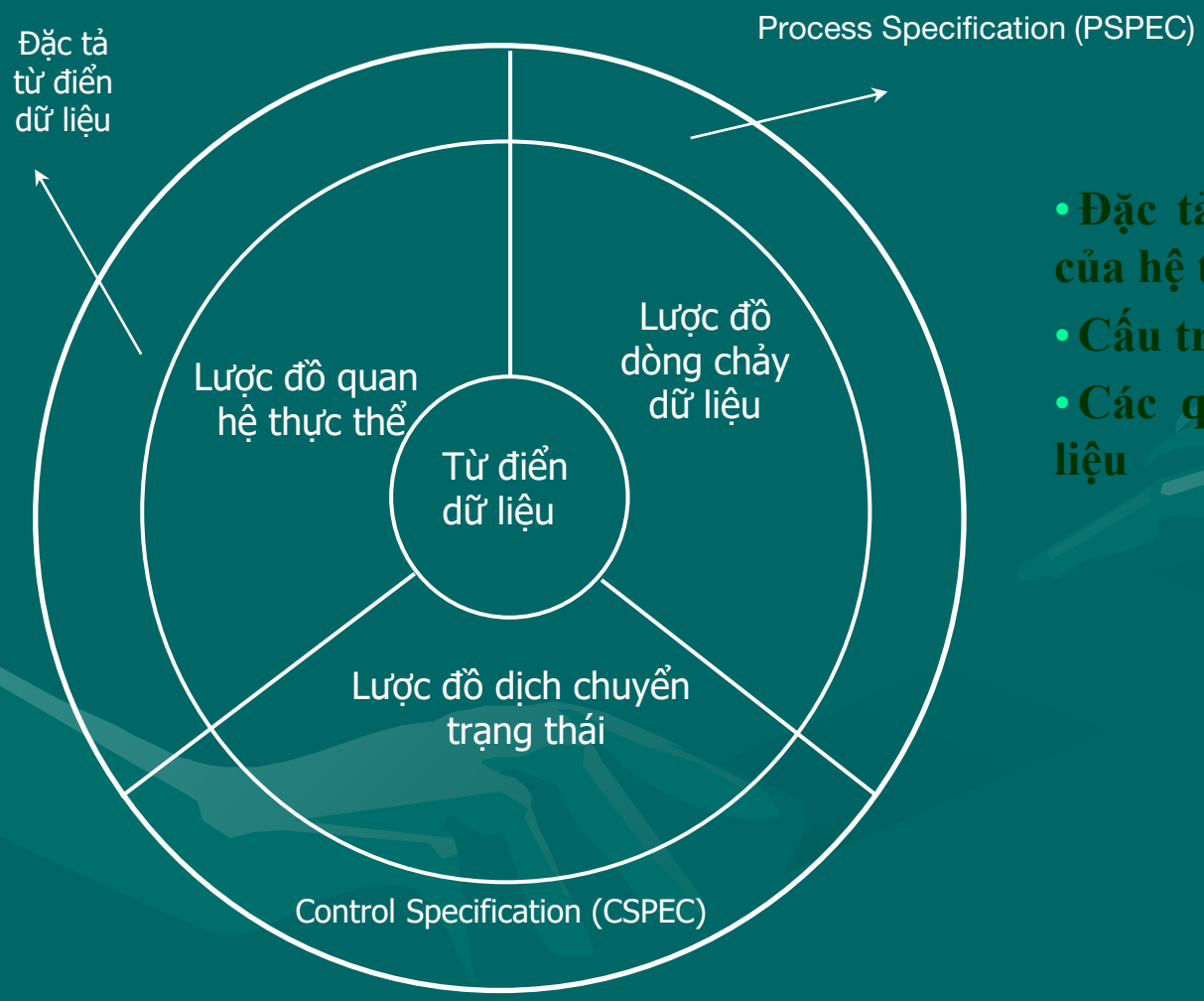
CÁC YẾU TỐ CĂN BẢN CỦA MÔ HÌNH



Lược đồ STD

- Mô hình hành vi của hệ thống
- Lược đồ dịch chuyển trạng thái (STD) thể hiện
  - Các trạng thái khác nhau của hệ thống
  - Sự dịch chuyển giữa các trạng thái đó
- Mô tả chi tiết hơn điều kiện xảy ra của các hành vi
- Cung cấp một hình ảnh động về hệ thống

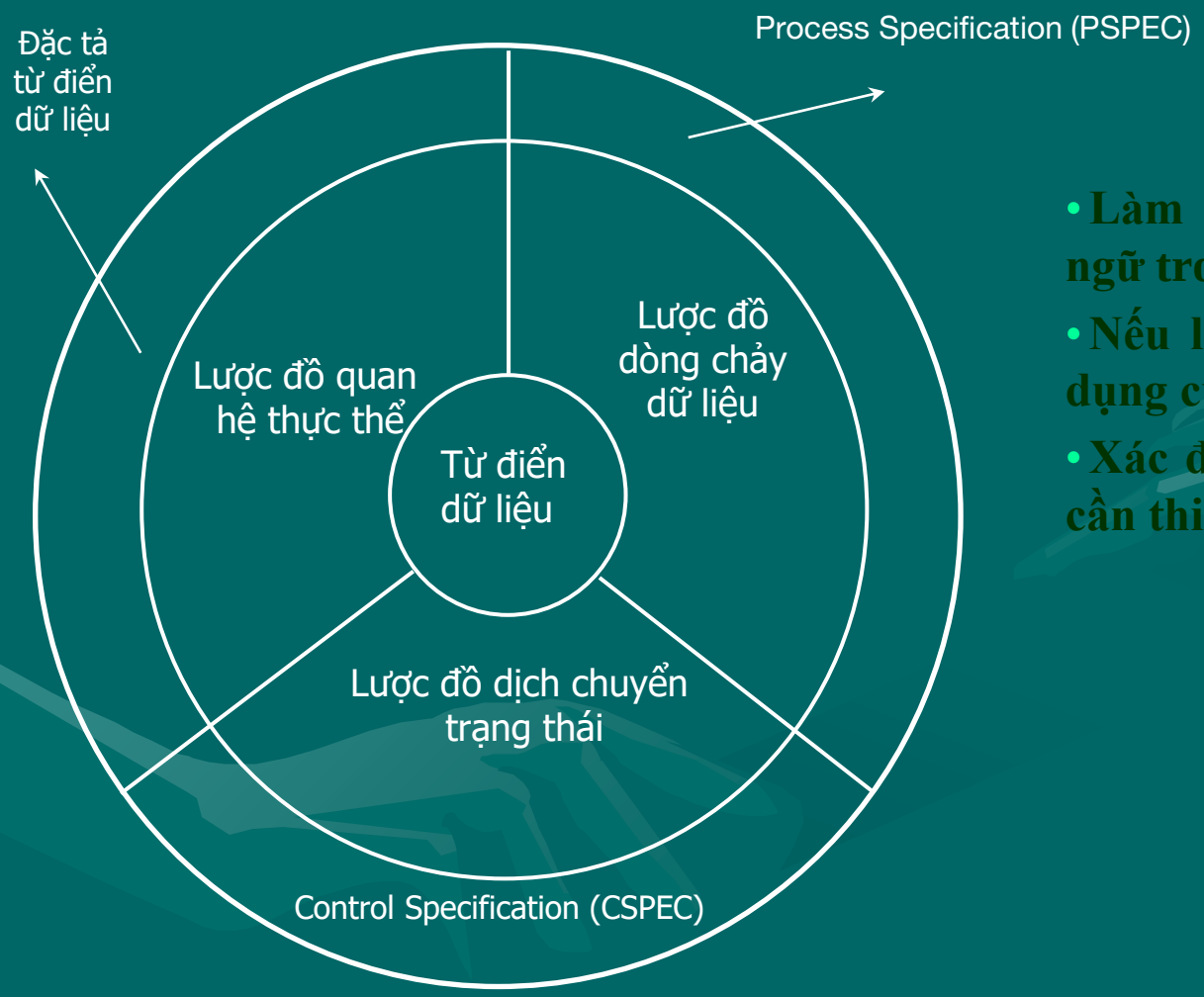
CÁC YẾU TỐ CĂN BẢN CỦA MÔ HÌNH



Lược đồ ERD

- **Đặc tả các thông tin về dữ liệu của hệ thống**
- **Cấu trúc dữ liệu**
- **Các quan hệ và ràng buộc dữ liệu**

CÁC YẾU TỐ CĂN BẢN CỦA MÔ HÌNH



Từ điển dữ liệu

- Làm rõ các khái niệm và thuật ngữ trong hệ thống
- Nếu lên ý nghĩa và phạm vi sử dụng của các khái niệm này
- Xác định các cấu trúc thông tin cần thiết

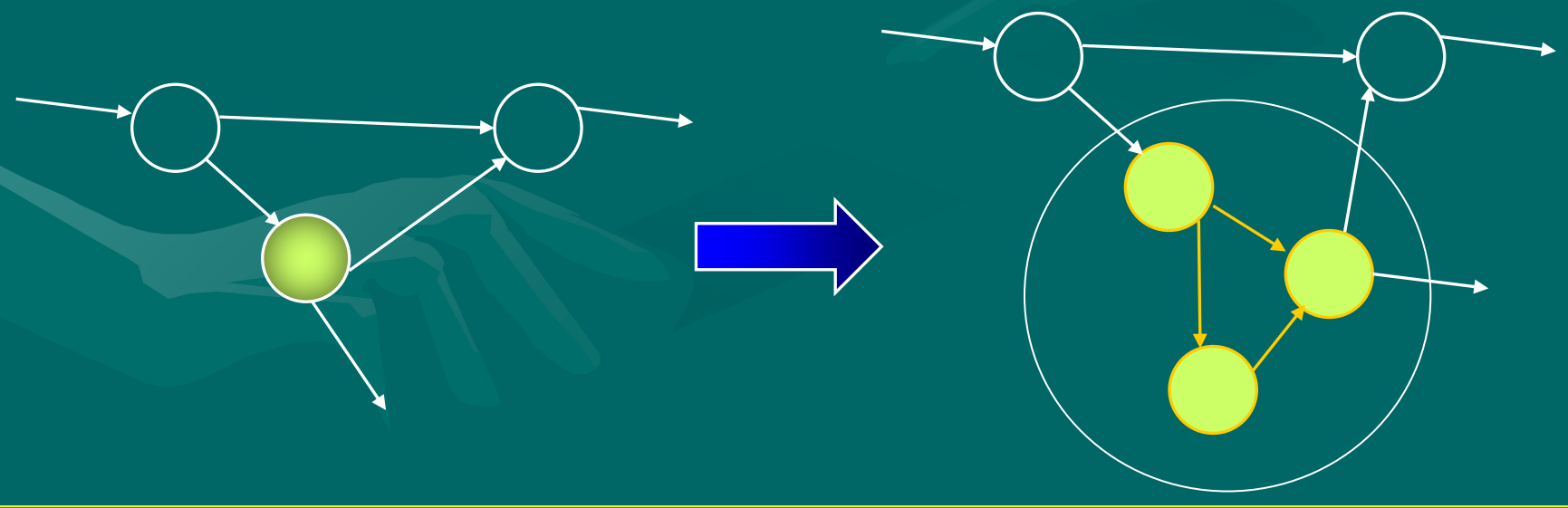
Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

LƯỢC ĐỒ DÒNG CHẢY DỮ LIỆU (DFD)

- Được xây dựng từ 4 phần tử chính
  - Thực thể**: tạo ra hoặc tiêu thụ thông tin, nằm bên ngoài biên giới của phạm vi thông tin hệ thống**
  - Chức năng xử lý**: thực hiện chức năng nào đó, tiêu thụ và tạo ra thông tin, nằm bên trong phạm vi thông tin hệ thống**
  - Thông tin hay dữ liệu
  - Kho dữ liệu**: lưu trữ dữ liệu mà được sử dụng bởi nhiều chức năng xử lý**

LƯỢC ĐỒ DÒNG CHẢY DỮ LIỆU (t.t)

- DFD được xây dựng qua nhiều mức khác nhau: mức 0, 1, 2…
- DFD mức sau chi tiết hơn mức trước
- Process Specification (PSPEC) bổ sung cho DFD
- Tính liên tục của dòng dữ liệu

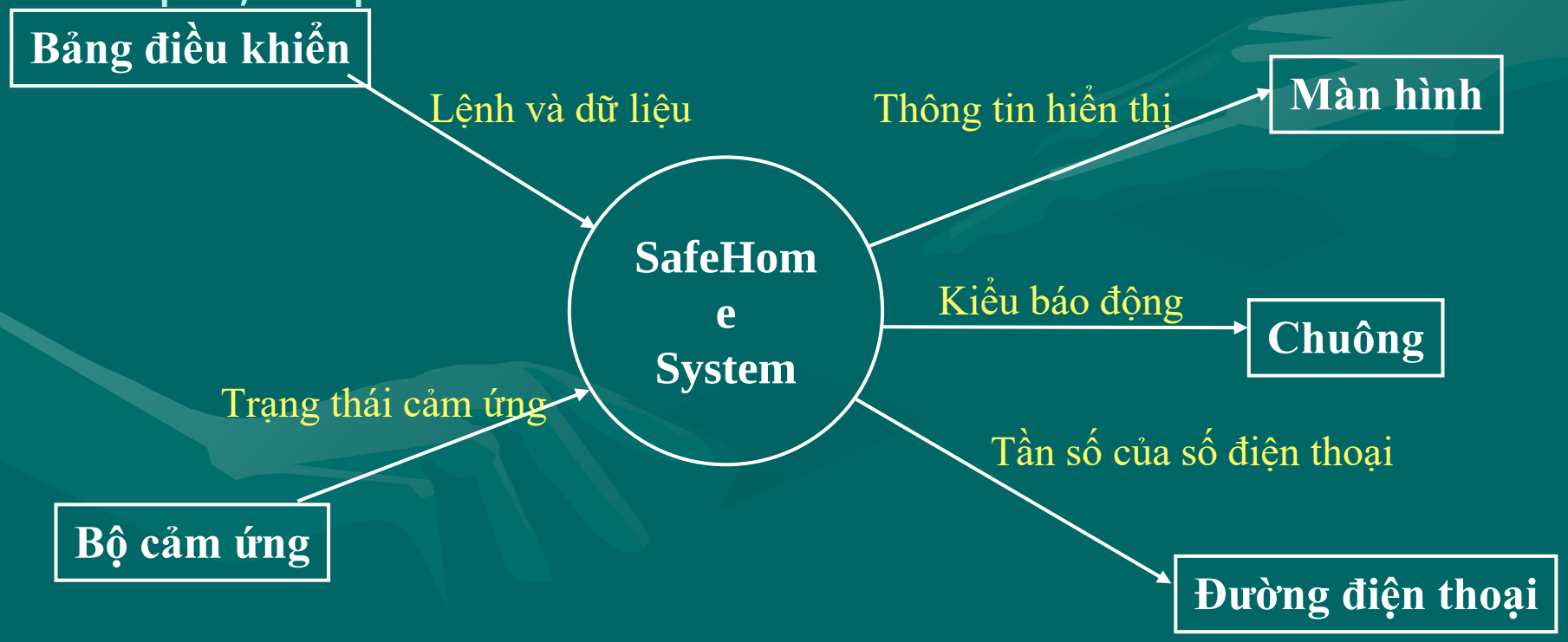


Kỹ thuật phân tích hệ thống

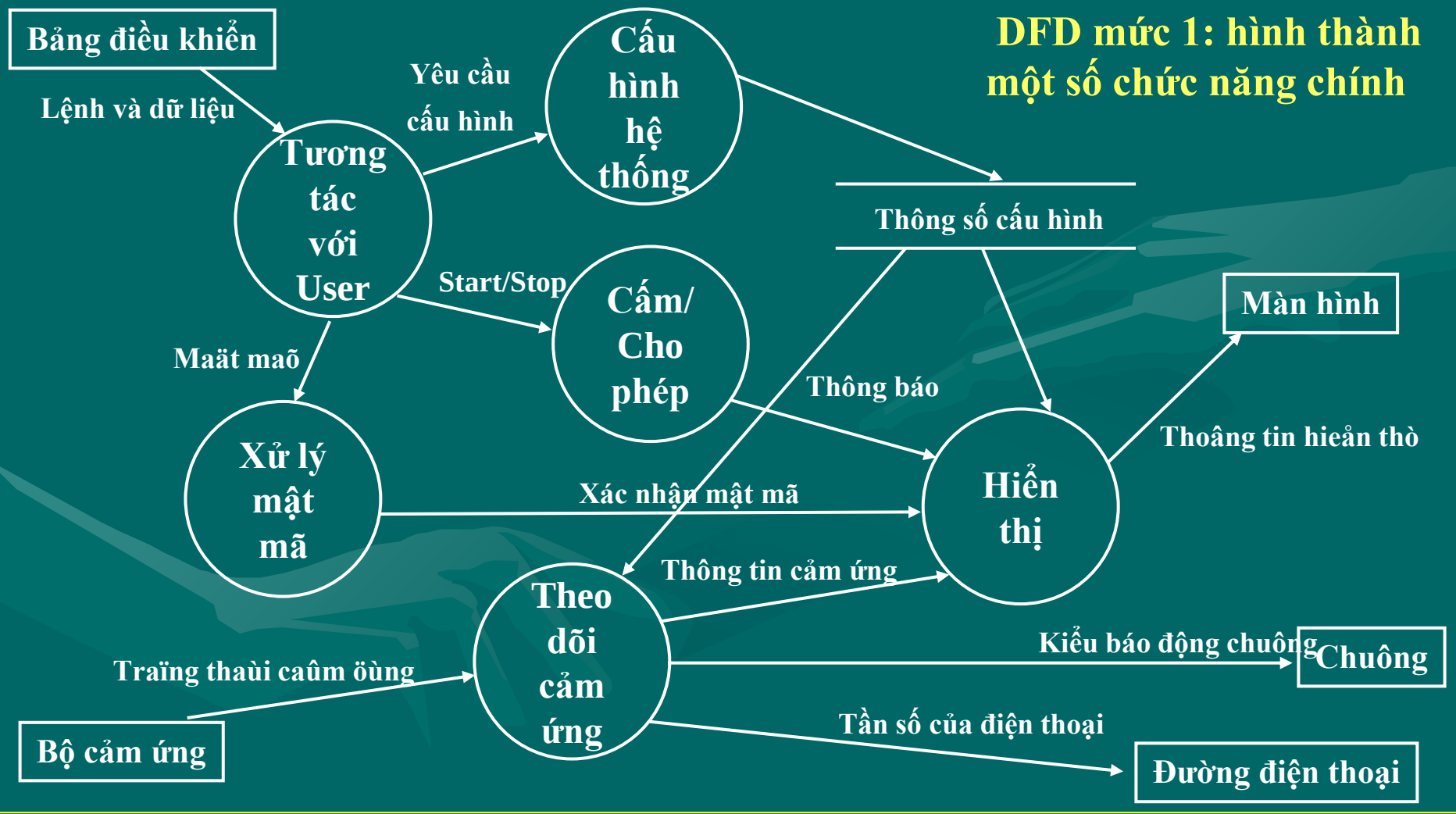
- Tiếp xúc, phỏng vấn các người dùng trong hệ thống thu thập các thông tin về nghiệp vụ của người dùng
- Thiết lập đoạn văn miêu tả chức năng (processing narrative) cho hệ thống cần xây dựng
- Xây dựng DFD ở các mức khác nhau
  - **Thiết lập sơ đồ ngữ cảnh (DFD mức 0)**
  - **Phân hoạch DFD vào các mức cao hơn**
  - **Sử dụng phương pháp duyệt văn phạm.**
  - **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 – Ví dụ

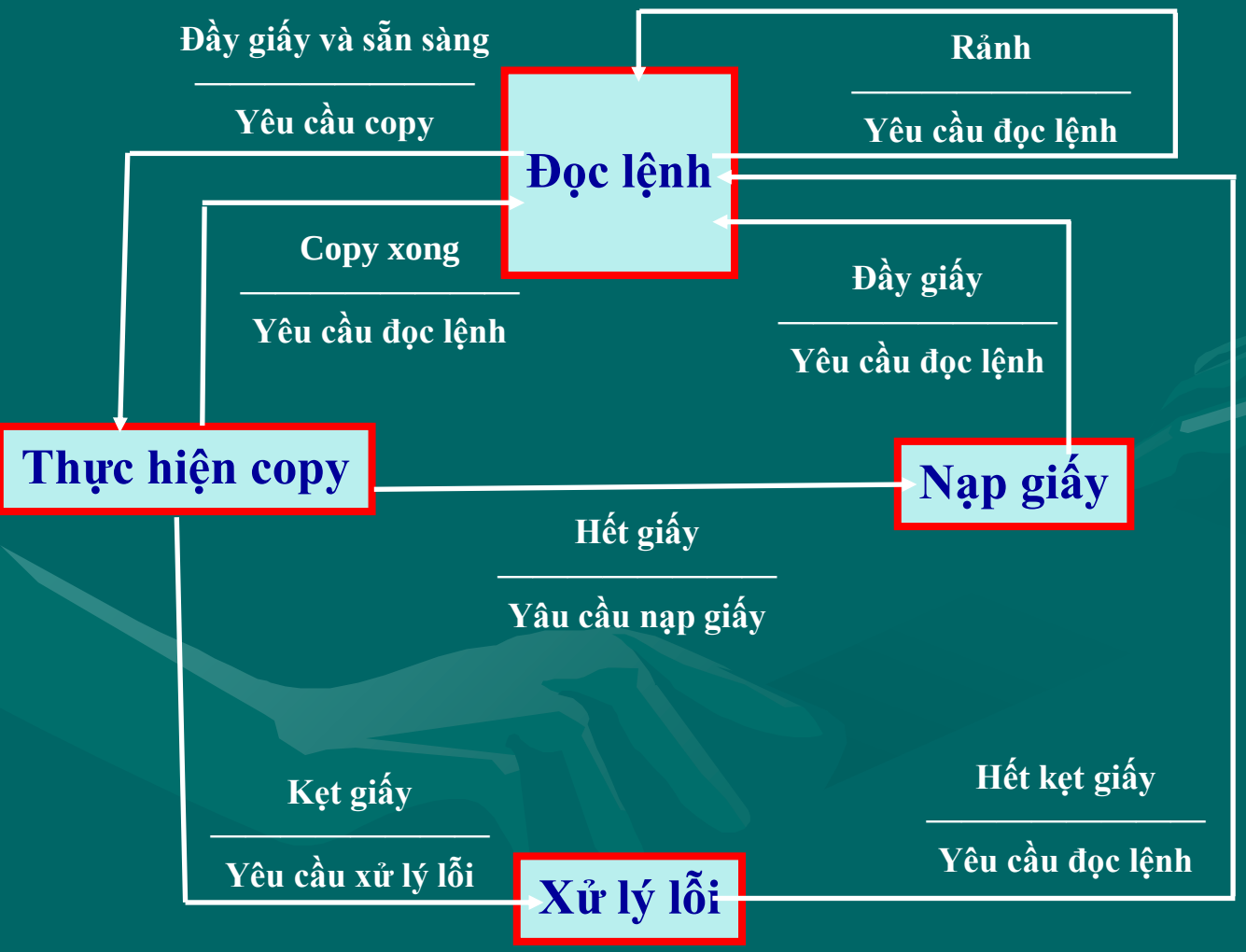
- Phần mềm SafeHome: Thiết lập đoạn văn miêu tả xử lý
- DFD mức ngữ cảnh: nhận diện các thực thể và dữ liệu input, output



Xây dựng DFD – Ví dụ



Mô hình hành vi STD – Ví dụ



Mô hình hành vi hệ thống máy photocopy

Tự điển dữ liệu

- Nhiều phần tử được tạo ra trong mô hình phân tích: dữ liệu, chức năng, điều khiển …
- Phải có một cách thức quản lý các phần tử đó sao cho hiệu quả ☐ **Từ điển dữ liệu**
- Định nghĩa: Từ điển dữ liệu là một danh sách có tổ chức của tất cả các phần tử dữ liệu cần thiết cho hệ thống. Các phần tử được định nghĩa chính xác và chặt chẽ sao cho cả phân tích viên và khách hàng cùng chia sẻ một suy nghĩ về chúng.
- Từ điển dữ liệu thường được hiện thực như là một phần của công cụ CASE.
- Mỗi phần tử bao gồm những thông tin: tên, bí danh, được dùng ở đâu/như thế nào, đặc tả nội dung và thông tin phụ trợ

Tự điển dữ liệu – Ví dụ

- Ví dụ phần tử dữ liệu số điện thoại

  Tên: **Số điện thoại**

  Bí danh: **Không**

  Được dùng ở đâu/như thế nào:

  output của Thiết lập điều kiện báo động

  input của Quay số

  Đặc tả nội dung:

  số điện thoại = [mở rộng địa phương | số bên ngoài]

  mở rộng địa phương = [2001 | 2002 … | 2009]

  số bên ngoài = 9 + [số địa phương | số đường dài]

  số địa phương = tiền tố + <chuỗi 4 ký số>

  số đường dài = (1) + mã vùng + số địa phương

  tiền tố = [795 | 799 | 874 | 877]

Review – Phân tích hệ thống theo cấu trúc

- Phân tích yêu cầu theo pp cổ điển 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 được mở rộng để biểu diễn được các hệ thống thời gian thực
- Xây dựng DFD mức 0 rồi đến các mức cao hơn; chú ý bảo toàn tính liên tục của dòng dữ liệu
- Từ điển dữ liệu giúp quản lý và tra cứu các phần tử dữ liệu

Phân tích hệ thống theo hướng phát triển kỹ thuật lập trình OOP

Tiếp cận của phương pháp phát triển OOP cho bước phân tích hệ thống

AI / Vị trí như thế nào / Làm Gì / Khi nào

Các lược đồ

- **Lược đồ Use-case: thu thập yêu cầu – mô hình nghiệp vụ**
- **Lược đồ lớp: phân tích yêu cầu – mô hình phân tích**

Mô hình nghiệp vụ - Thu thập yêu cầu

- Quan điểm thu thập/phân tích yêu cầu của mô hình nghiệp vụ: hệ thống gồm có AI/Làm những gì/Khi nào
- Lược đồ Use-case :
  - **Actor & Use-case**
  - **Các mối quan hệ : Actor – Actor ; Actor-Use-case, - Use-case-Usecase**
- Actor xác định một bộ 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**
  - **Chỉ quan tâm các thông điệp mà actor gửi hay nhận**
  - **Không quan tâm cấu trúc bên trong của actor**
- Phân loại actor
  - **Chủ yếu / Thứ yếu**
  - **Tích cực / Thụ động**

Nhận diện các ACTOR

- Trả lời một số câu hỏi như
  - **Ai là người sử dụng chức năng chính của hệ thống ?**
  - **Ai cần sự hỗ trợ từ hệ thống để thực hiện công việc thường nhật của họ ?**
  - **Ai phải thực hiện công việc bảo dưỡng, quản trị và giữ cho hệ thống hoạt động ?**
  - **Hệ thống sẽ kiểm soát thiết bị phần cứng nào ?**
  - **Hệ thống đang 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 ?**

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

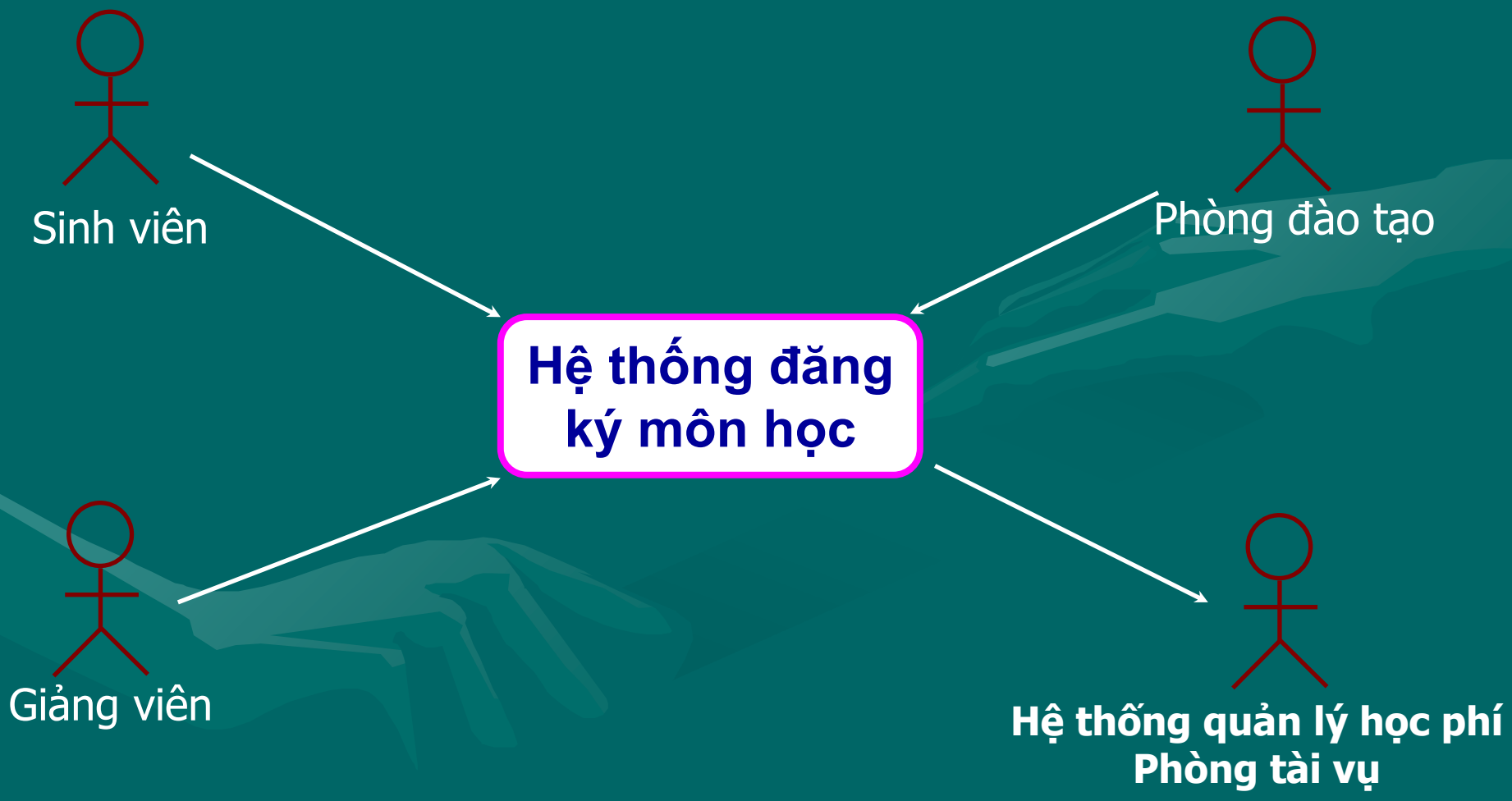
Biểu diễn ACTOR trong UML

- Actor được biểu diễn bằng ký hiệu hình người

Tên Actor

- Actor được xem là một lớp (class) có stereotype là <<actor>>
- Giữa các actor có thể có quan hệ tổng quá hoá
  - **Ví dụ: Sinh viên, giảng viên và khách đều là độc giả của hệ thống quản lý thư viện: độc giả là actor tổng quát hóa của 2 actor sinh viên và giảng viên**
- Ví dụ: một hệ thống đăng ký môn học trong trường đại học

Các Actor trong hệ thống đăng ký môn học

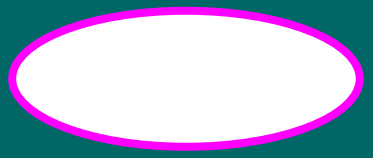


Khái niệm Use-case

- Use-case biểu diễn một chức năng của hệ thống phần mềm
- Use-case được 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 được bắt đầu bằng thông điệp đến từ actor**
  - **Use-case phải hoàn tất: chuỗi thông điệp phải kết thúc bằng kết quả cụ thể.**
- Lỗi thường gặp: chia nhỏ use-case trở thành những chức năng vụn vặt

Khái niệm Use-case

- Use-case được biểu diễn bằng hình ellipse:



Tên Use-case

- Điểm mở rộng là một vị trí trong use-case mà tại đó có thể chèn chuỗi sự kiện của một use-case khác
- Use-case có thể chứa điều kiện rẽ nhánh, xử lý lỗi, ngoại lệ...
- Minh dụ của use-case là kịch bản (scenario): miêu tả cụ thể trình tự các sự kiện xảy ra khi Use-case được thực hiện.

Tìm kiếm Use-case

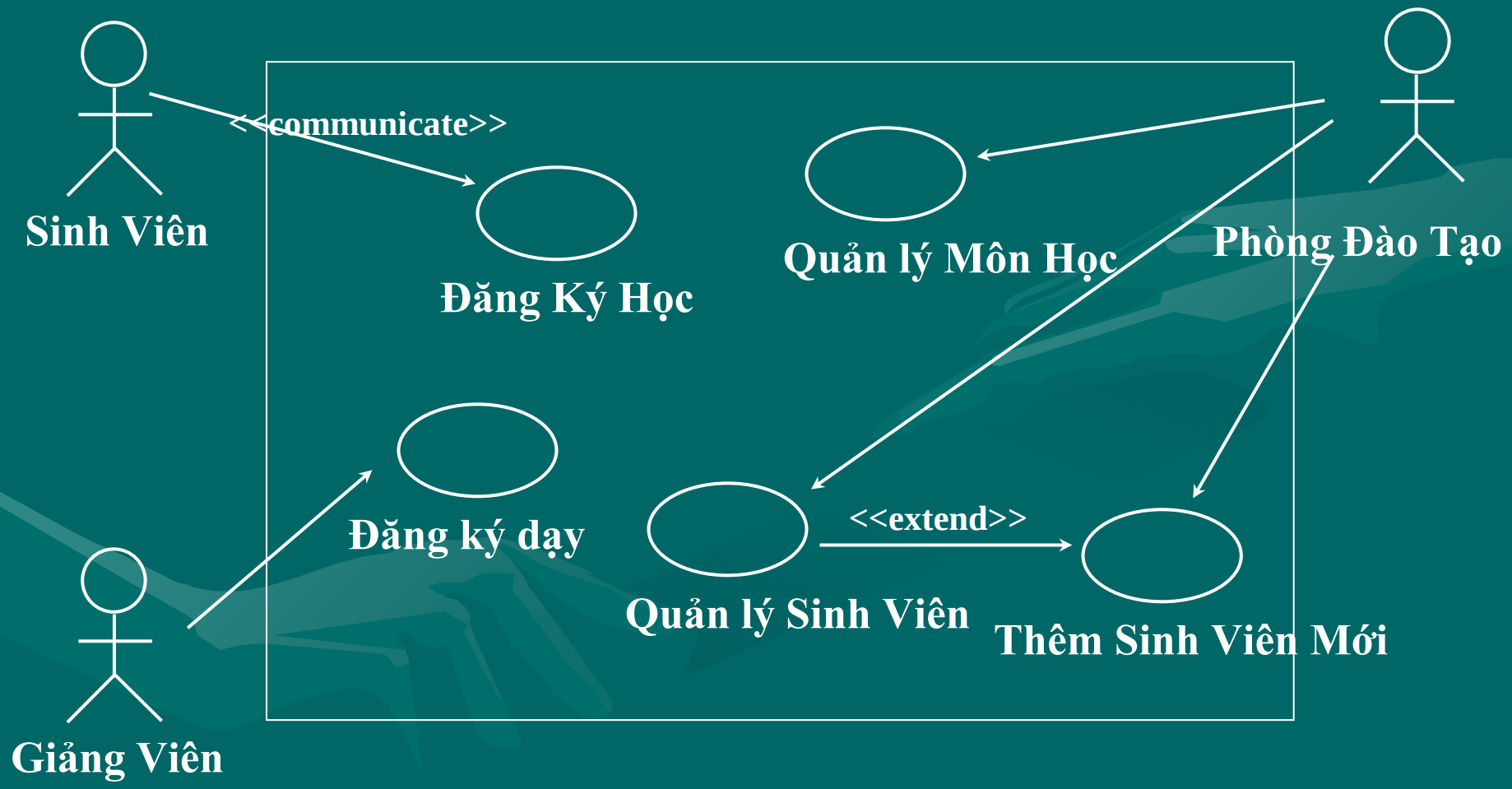
- Trả lời một số câu hỏi như
  - **Actor yêu cầu chức năng gì của hệ thống ?**
  - **Actor cần phải đọc, tạo, xoá, sửa đổi hoặc lưu trữ thông tin nào đó của hệ thống không ?**
  - **Actor cần thiết phải được cảnh báo về những sự kiện trong hệ thống, hay actor cần phải báo hiệu cho hệ thống về vấn đề nào đó không ?**
  - **Hệ thống có thể hỗ trợ một số công việc thường nhật của actor nào đó hay không ?**
- Một số câu hỏi khác cần chú ý
  - **Hệ thống cần dữ liệu input/ouput nào ? Dữ liệu đó đến từ đâu ?**
  - **Những khó khăn 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) ?**

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Các quan hệ của Use-case

- Sau khi xác định các Actor và Use-case thì các quan hệ sẽ được thiết lập để hoàn chỉnh lược đồ Use-case
- Giữa use-case và actor thường có quan hệ liên kết
  - **Use-case được Actor nào kích hoạt**
- Giữa các use-case cũng có quan hệ liên kết hoặc tổng quát hoá
  - **Quan hệ liên kết: extent , incluse**
  - **Quan hệ thổng quát hóa**
- Ví dụ: một hệ thống đăng ký môn học theo tín chỉ trong trường đại học

Ví dụ một Use-case đơn giản



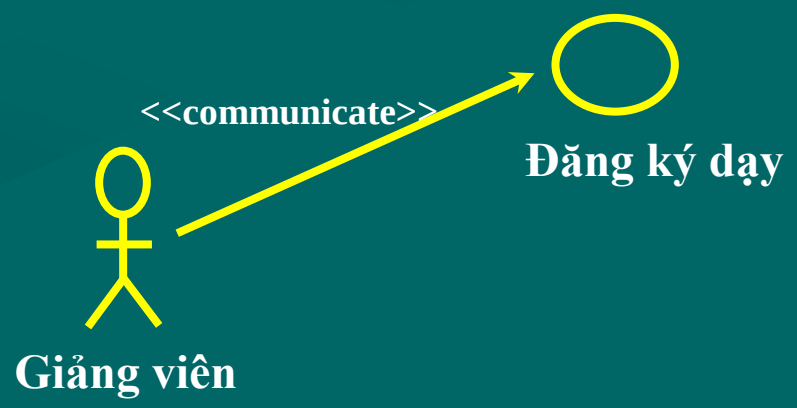
Quan hệ liên kết

- Quan hệ liên kết chỉ ra một quan hệ có ý nghĩa 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**
- UML biểu diễn liên kết như là một đoạn thẳng (hai chiều) hoặc mũi tên (một chiều)

- Có thể áp dụng stereotype:
  - **<<include>>**
  - **<<extend>>**
  - **<<communicate>>**
  - **…**

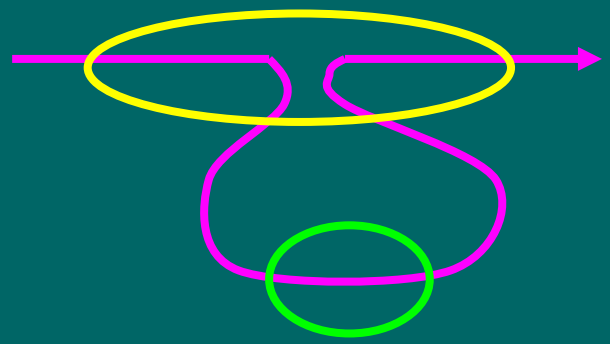
Quan hệ liên kết giữa Actor và Use-case

- Liên kết là quan hệ duy hất giữa actor & Use-case
- Liên kế có thể là 2 chiều hay 1 chiều
  - **actor kích hoạt use-case và nhận kết quả về: liên kết 2 chiều**
  - **actor kích hoạt use-case, không quan tâm kết quả về: liên kết 1 chiều**
- Quan hệ liên kết phổ biến giữa Actor & Use-case là giao tiếp: *stereotype* là <<communicate>> dùng liên kết giữa actor và use case mà nó kích hoạt



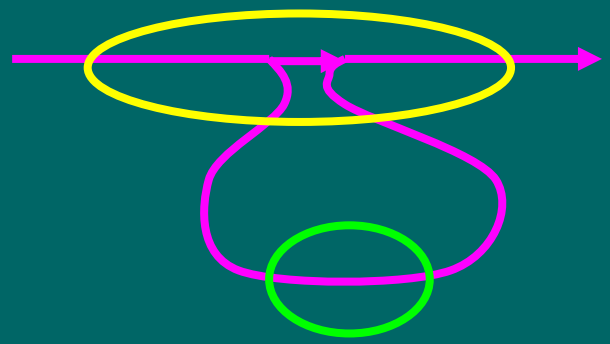
Quan hệ gộp giữa 2 Use-case

- Dùng để liên kết 2 Use-case: có stereotype là <<include>>
- Trong Use-case nguồn có điểm mở rộng mà tại đó bắt buộc phải chèn Use-case đích vào.
  - **Tại điểm mở rộng, diễn tiến của use-case nguồn tạm thời ngừng lại để 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 mở rộng giữa 2 Use-case

- Dùng để liên kết 2 Use-case: có stereotype là <<extend>>
- Trong use-case nguồn có một điểm mở rộng mà tại đó có thể (hoặc không) chèn use-case đích vào tùy thuộc vào điều kiện rẽ nhánh hoặc tương tác từ actor
  - Tại điểm mở rộng, nếu được mở rộng thì diễn tiến của use-case nguồn tạm thời ngừng lại để chuyển sang diễn tiến của use-case đích
  - Khi kết thúc use-case đích, diễn tiến của use-case nguồn lại tiếp tục



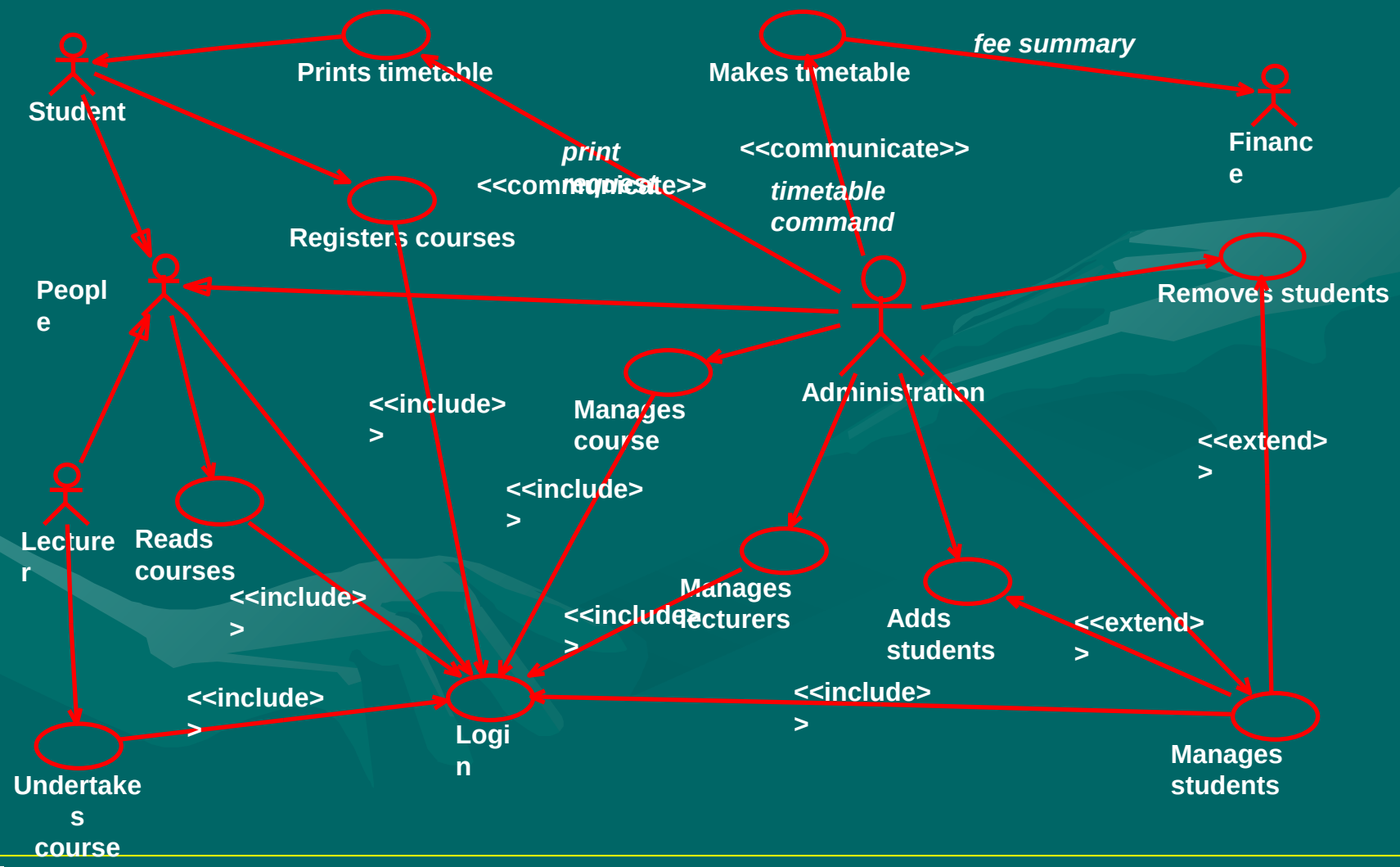
Quan hệ tổng quát hóa

- Là mối quan hệ giữa các đối tượng cùng nhóm tạo nên một đối tượng mang những tính chất chung của các đối tượng kia
- Quan hệ thống quát hóa giữa các Actor: nhiều actor có chung một số vai trò -> hình than2h actor tổng quát hóa mang vai trò chung đó.
  - **Ví dụ: sinh viên, giáo viên đều có chung use-case login và đều là user của hệ thống -> tạo nên actor user là tổng quát hóa của actor sinh viên và actor giáo viên**
- Quan hệ thổng quát hóa giữa các Use-case: khi có nhiều use-case là trường hợp cụ thể một use-case trừu tượng
  - **Vì dụ: Use-case login của sinh viên , giáo viên có thể được thực hiện theo cơ chế khác nhau nhưng cùng mang chung ý nghĩa là đăng nhập**
    - **là các trường hợp cụ thể của Use-case trừu trương LOGIN**

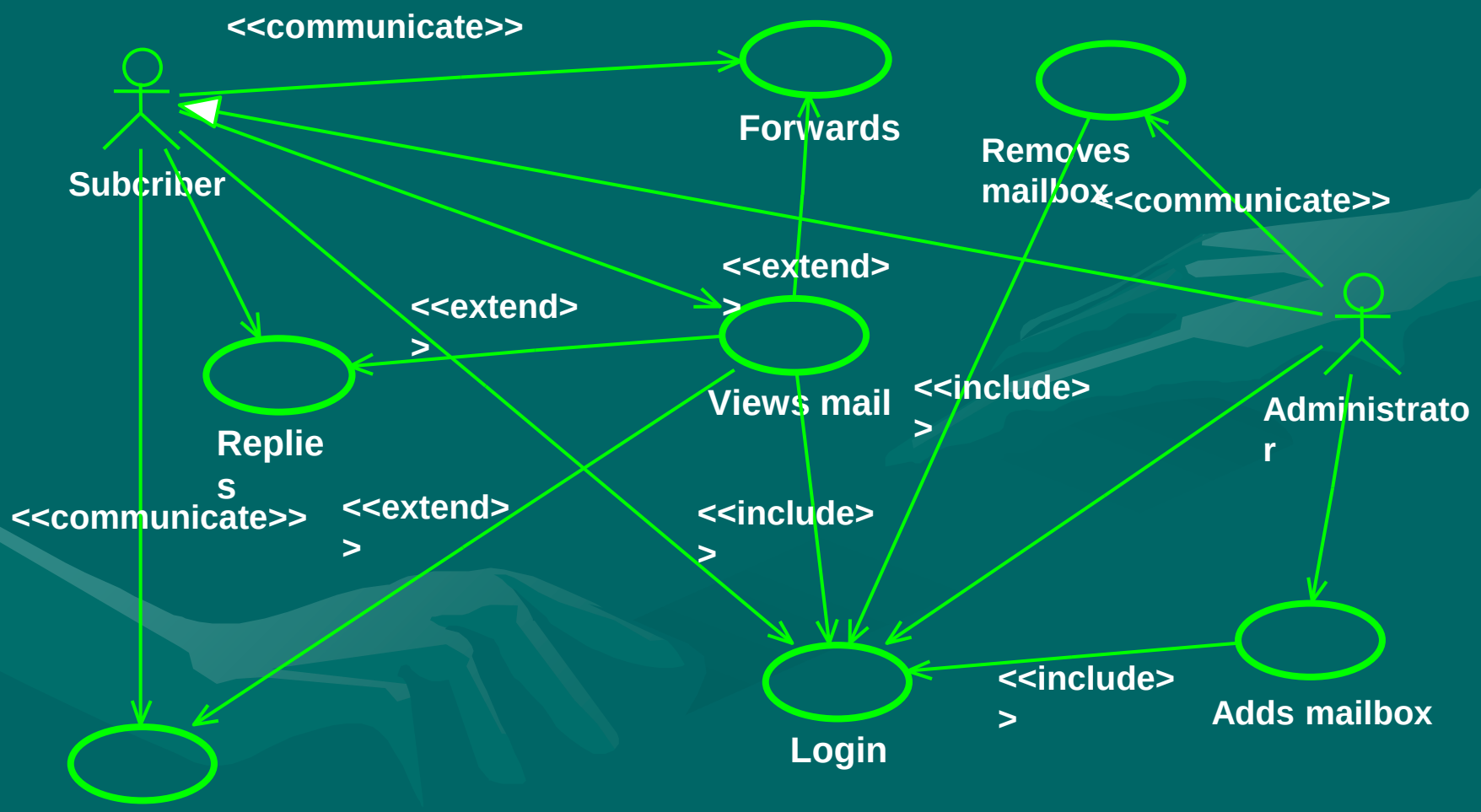
Xây dựng mô hình Use-case

- Các yêu cầu của phần mềm được mô tả trong mô hình use-case
- Mô hình use-case bao gồm lược đồ use-case và (có thể) một số packet (gom một số use-case thành một bộ chức năng con của hệ thống)
- Phương pháp thực hiện:
  - **Xác định các actor và use-case của hệ thống**
  - **Xác lập các quan hệ giữa các đối tượng này**
    - Quan hệ liên kết giữa actor và use-case: một chiều hoặc hai chiều, thường có stereotype là <<communicate>>
    - Quan hệ mở rộng 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 trừu tượng
    - Quan hệ tổng quát hoá giữa các use-case: nhiều use-case là trường hợp cụ thể của một use-case trừu tượng
  - **Trình bày thành lược đồ use-case theo chuẩn UML**
  - **Có thể xác định các packet**

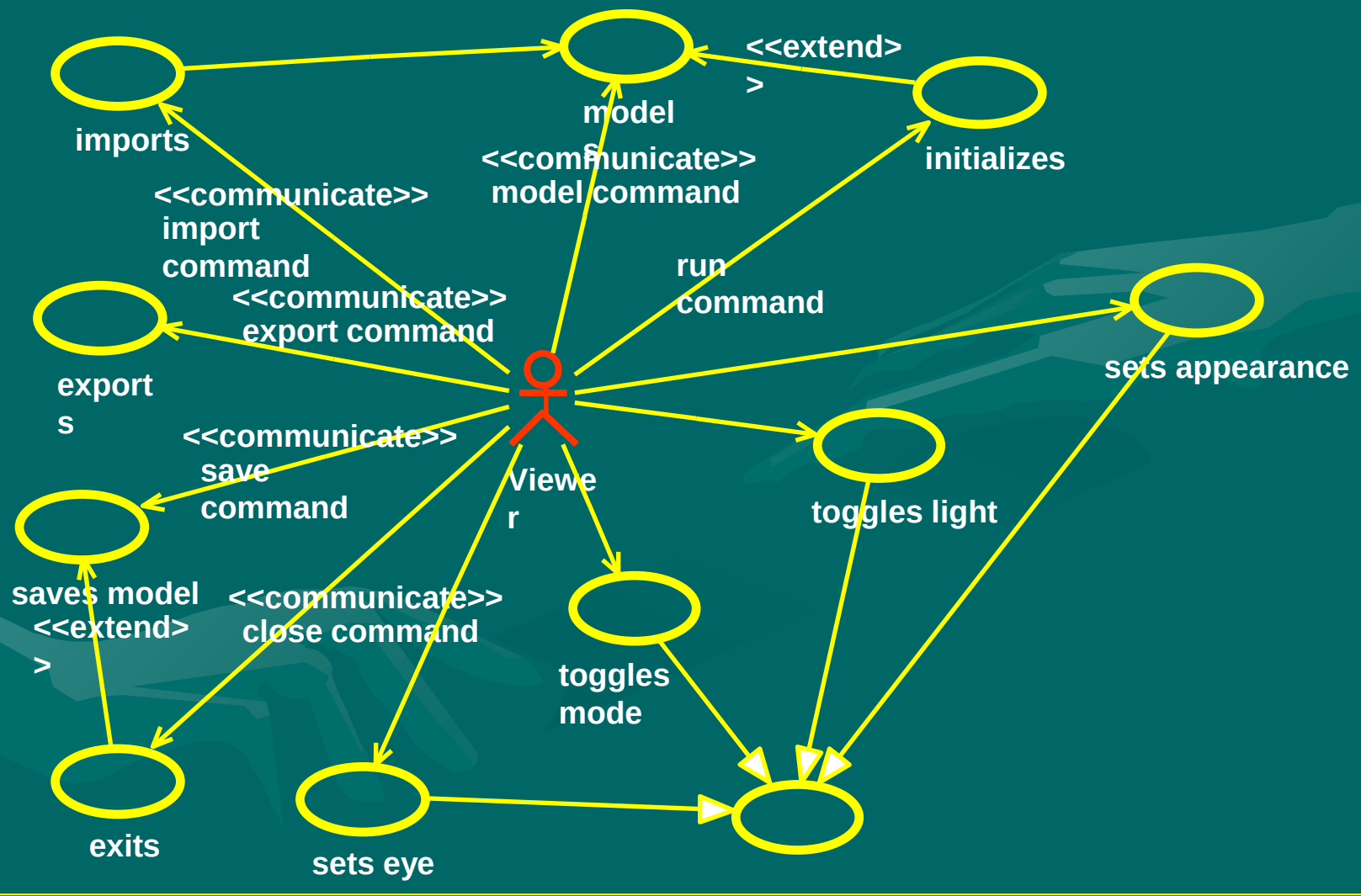
Xây dựng mô hình Use-case – VÍ DỤ



Xây dựng mô hình Use-case – VÍ DỤ



Xây dựng mô hình Use-case – VÍ DỤ



Mô hình phân tích – Phân tích yêu cầu

Đối tượng/lớp - quan hệ

- **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 đối tượng "sống" trong ngữ cảnh của phần mềm
- Các đối tượng sẽ tương tác với nhau để tạo nên các chức năng mô tả bởi use-case
- Lược đồ Class phân tích diễn tả cấu trúc, mối quan hệ giữa các đối tượng/lớp trong hệ thống
- Chưa quan tâm đến hành vi cụ 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

Xây dựng mô hình phân tích

- Mô hình phân tích được diễn đạt trong UML bằng lược đồ lớp phân tích (Class diagrame)
- Các công việc xây dựng lược đồ phân tích bao gồm
  - **Tìm kiếm các đối tượng / lớp trong hệ thống**
    - Đối tượng / lớp thực thể
    - Đối tượng / lớp biên
    - Đối tượng / lớp control
  - **Xác định các thuộc tính của đối tượng / lớp**
  - **Xác định các tác vụ của đối tượng / lớp**
  - **Nhận diện các lớp trừu tượng qua mối quan hệ thổng quát hóa**
  - **Xác lập các mối quan hệ giữa các lớp:**
    - Tổng quát hoá (generalization)
    - Liên kết (association)
    - Bao gộp (aggregation)
- Biểu diễn thành lược đồ lớp phân tích

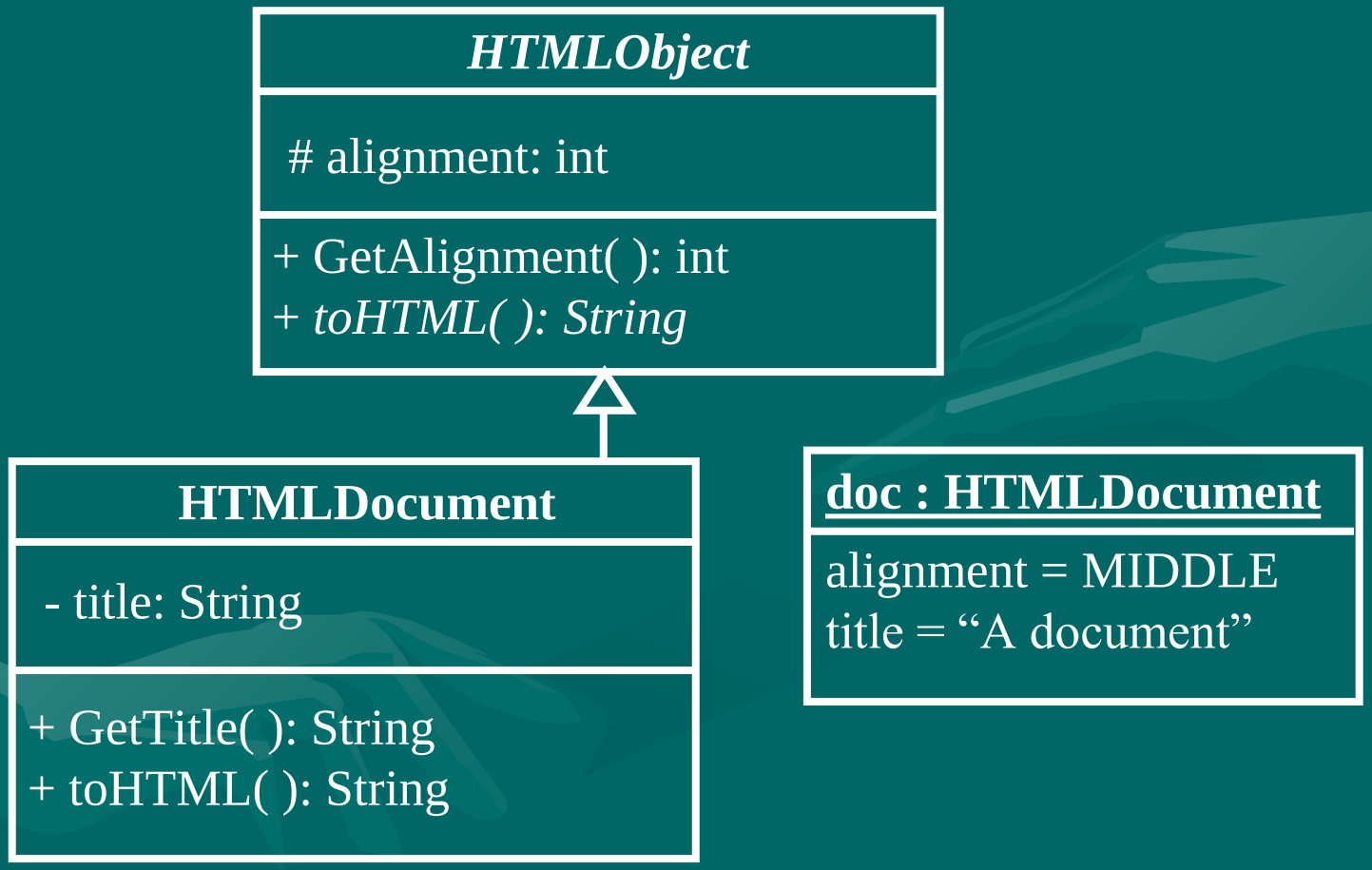
Nhập diện đối tượng / lớp

- Dựa vào đặc tả của từng use-case để tìm kiếm các đối tượng
- Các đối tượng thường xuất hiện trong các danh từ hay nhóm danh từ
- Một số lưu ý
  - **Không nên dùng đối tượng để biểu diễn một dữ liệu đơn (nên xem là thuộc tính của đối tượng khác)**
  - **Đối tượng/lớp phải thực sự cần thiết cho sự hoạt động của hệ thống**
  - **Đối tượng/lớp >< bảng cơ sở dữ liệu**
  - **Đối tượng/lớp >< actor**

Nhận diện và biểu diễn đối tượng / lớp

- Phân loại đối tượng/lớp
  - **Đối tượng thực thể (entity): biểu diễn các thông tin thiết yếu của hệ thống, có thể được lưu trong cơ sở dữ liệu**
  - **Đối tượng biên (boundary): thực hiện chức năng giao tiếp với actor**
  - **Đối tượng điều khiển (control): điều khiển các đối tượng khác**
- Trong UML, lớp được biểu diễn bằng một hình chữ nhật gồm 3 phần: tên, các thuộc tính và các tác vụ
- Có thể áp dụng stereotype cho lớp: <<entity>>, <<boundary>>, <<control>>...
- Đối tượng cũng được biểu diễn bằng hình chữ nhật, thông thường gồm 2 phần: tên đối tượng + tên lớp (được gạch chân), giá trị các thuộc tính (trạng thái của đối tượng)

Biểu diễn lớp / đối tượng

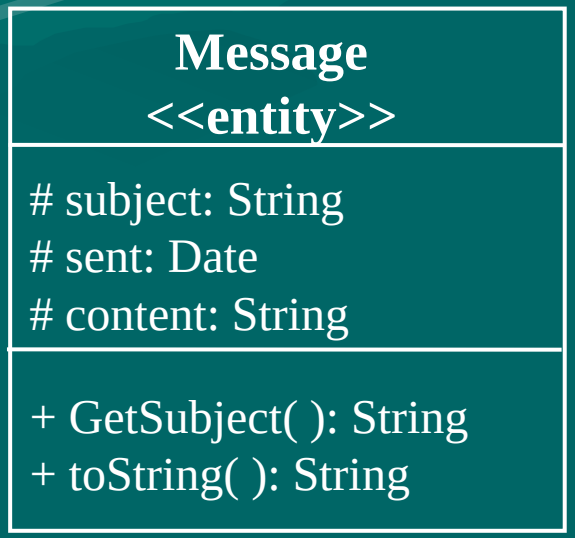


Đối tượng / lớp thực thể

- Biểu diễn cho các thực thể xuất hiện một cách tự nhiên trong hệ thống
- Thông tin về các đối tượng thực thể có thể phải được lưu trữ lâu dài (database, file...)
- Trong UML, được gán stereotype <<entity>>
- Dễ nhận diện các thuộc tính của chúng

Ví dụ:

- Đối với hệ thống đăng ký môn học hệ tín chỉ qua WEB, nhận diện các đối tượng thực thể như: thông tin SV, thông tin GV, nhóm lớp học, đăng ký nhóm, sổ tay sinh viên …
- Đối với hệ thống mail, nhận diện các đối tượng thực thể như: hộp thư, thông điệp mail…

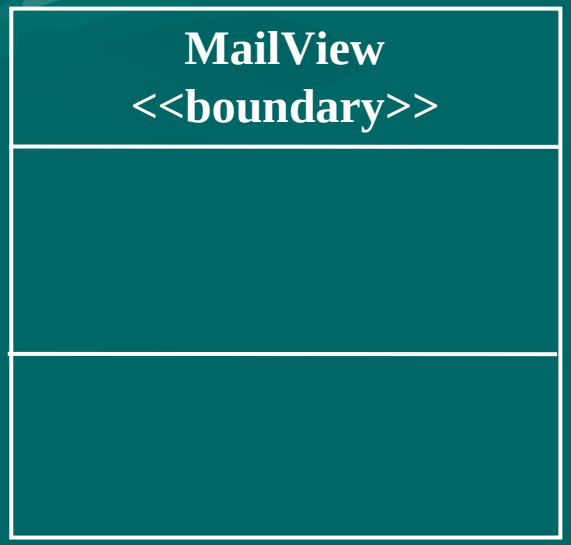


Đối tượng / lớp biên

- Thực hiện chức năng giao tiếp với actor
- Thường chứa các phần tử hoặc điều khiển giao diện người dùng (nút nhấn, hộp danh sách, tuỳ chọn, menu...)
- Trong UML, được gán stereotype <<boundary>>
- Khó nhận biết các thuộc tính và tác vụ trong mô hình phân tích

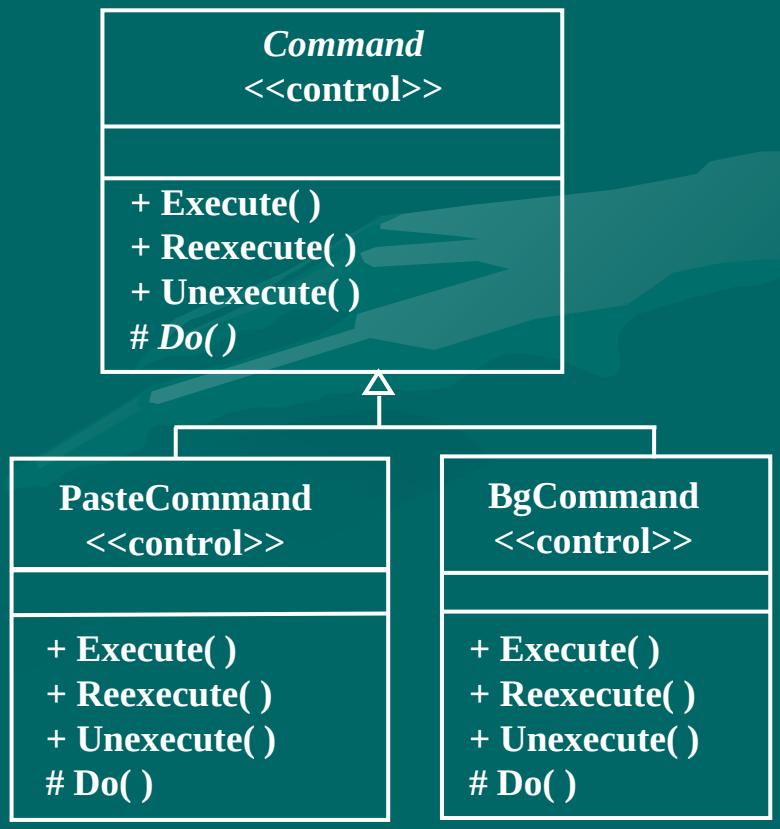
Ví dụ:

- Đối với hệ thống đăng ký môn học hệ tín chỉ qua WEB, nhận diện các đối tượng biên như: RegisterForm, StudentForm…
- Đối với hệ thống mail, nhận diện các đối tượng biên như: MailView, MailCompose...



Đối tượng / lớp điều khiển

- Có nhiệm vụ điều khiển các lớp khác hoặc
- Những lớp không phải là lớp thực thể và lớp biên
- Trong UML, được gán stereotype <<control>>
- Lớp biên thường có quan hệ liên kết hoặc phụ thuộc với các lớp khác
- Ví dụ:
- Đối tượng biểu diễn một số lệnh thông thường như cắt, dán, thay đổi thông số nhìn trong hiển thị đồ hoạ …



Nhận điện các thuộc tính

- Dựa vào đặc tả của từng use-case, tìm kiếm các danh từ hoặc nhóm danh từ liên quan đến đối tượng đang xét
- Trả lời câu hỏi: những thành phần nào cấu thành đối tượng đang xét ?
  - **Lưu ý: cùng một đối tượng trong các ngữ cảnh khác nhau chúng ta có thể tìm được các thuộc tính khác nhau**
- Nên xác định (tuy nhiên không bắt buộc) trong mô hình phân tích
  - **Kiểu của thuộc tính: một số kiểu cơ bản**
  - **Bậc của thuộc tính: số ít hoặc số nhiều**
  - **Visibility của thuộc tính: mức độ cho phép truy xuất thuộc tính từ bên ngoài**
- UML: thuộc tính được miêu tả tường minh hoặc thông qua quan hệ với các lớp khác

Xác định mức độ truy cập của thuộc tính

- Mức độ truy cập và phạm vi mà thuộc tính đó có thể được tham khảo đến trực tiếp
- UML định nghĩa 3 mức độ truy xuất thuộc tính (visibility)
  - **public (+): có thể truy xuất thuộc tính từ tất cả các vị trí khác nhau**
  - **protected (#): bản thân lớp đang xét và các lớp con của nó có thể truy xuất thuộc tính**
  - **private (-): chỉ có lớp đang xét có thể truy xuất thuộc tính**
- Thông thường nên đặt mức độ truy xuất thuộc tính là **private** hoặc **protected** (cho các lớp cơ sở), không nên là public.
- Thuộc tính nên được truy xuất thông qua tác vụ **get/set**

Ví dụ về nhận diện các thuộc tính

- Hệ thống đăng ký môn học hệ tín chỉ qua WEB - Nhận diện các thuộc tính cho các đối tượng: StudentInfo, LecturerInfo
- Chú ý các mức độ truy cập của các thuộc tính
- Các tác vụ phát sinh trong khi nhận diện các thuộc tính □ như Get/Set

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

Ví dụ về nhận diện các thuộc tính

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

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

- Hệ thống đăng ký môn học hệ tín chỉ qua WEB
- Nhận diện các thuộc tính cho các đối tượng:

  CourseOffering,

  Catalog

Nhận diện các tác vụ

- Dựa vào đặc tả của từng use-case, tìm kiếm các *động từ* hoặc *nhóm động từ* liên quan đến đối tượng đang xét
- Chú ý xem đối tượng được tạo ra và bị huỷ bỏ đi như thế nào ? Trong thời gian đó nó gửi/nhận thông điệp ra sao ?
- Các đối tượng biên có các tác vụ nhận lệnh từ actor.
- Xem xét mức độ truy xuất của tác vụ tương tự như đối với các thuộc tính; các tác vụ thường có visibility là **+** hoặc **#**
- Một số tác vụ không xuất hiện một cách tự nhiên trong mô hình phân tích □ mô hình thiết kế sẽ nghiên cứu kỹ trách nhiệm và hành vi của từng đối tượng

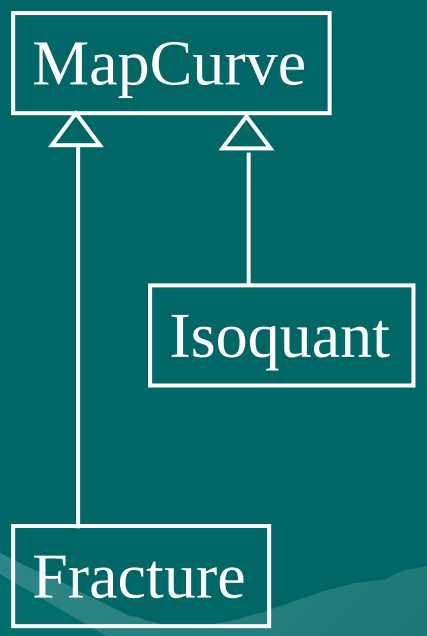
Nhận diện lớp cơ sở

- Lớp cơ sở (base class) được nhận diện sau khi đã nhận diện các lớp cụ thể
- Sự xuất hiện của lớp cơ sở làm cho mô hình phân tích có tính dùng lại cao (reusability) và dễ mở rộng (scalability)
- UML hỗ trợ quan hệ tổng quát hoá (generalization)
- Lớp cơ sở trừu tượng (không thể cụ thể hoá tạo ra đối tượng) có tên in nghiêng
- Lớp cơ sở được hình thành bằng cách xác lập các quan hệ tổng quát hóa của các lớp cụ thể có chung một số thuộc tính và/hay một số tác vụ

Nhận diện lớp cơ sở (tt)

- Đối với các đối tượng/lớp thực thể, tìm các thuộc tính chung để hình thành lớp cơ sở
- Ví dụ
  - **Trong hệ thống quản lý thư viện qua WEB: các đối tượng Book, Magazine có một số thuộc tính chung -> hình thành lớp LibraryItem**
  - **Đối với hệ thống đăng ký môn học tín chỉ qua WEB: lớp PeopleInfo là lớp cơ sở của StudentInfo và LecturerInfo**
  - **Chương trình vẽ bề mặt địa hình: lớp MapCurve là lớp cơ sở của đường đồng mức Isoquant và đứt gãy Fracture**
- Giữa lớp cơ sở và các lớp cụ thể có mối quan hệ thổng quát hóa có thể biểu diễn được trong UML

Biểu diễn lớp cơ sở và quan hệ tổng quát hóa

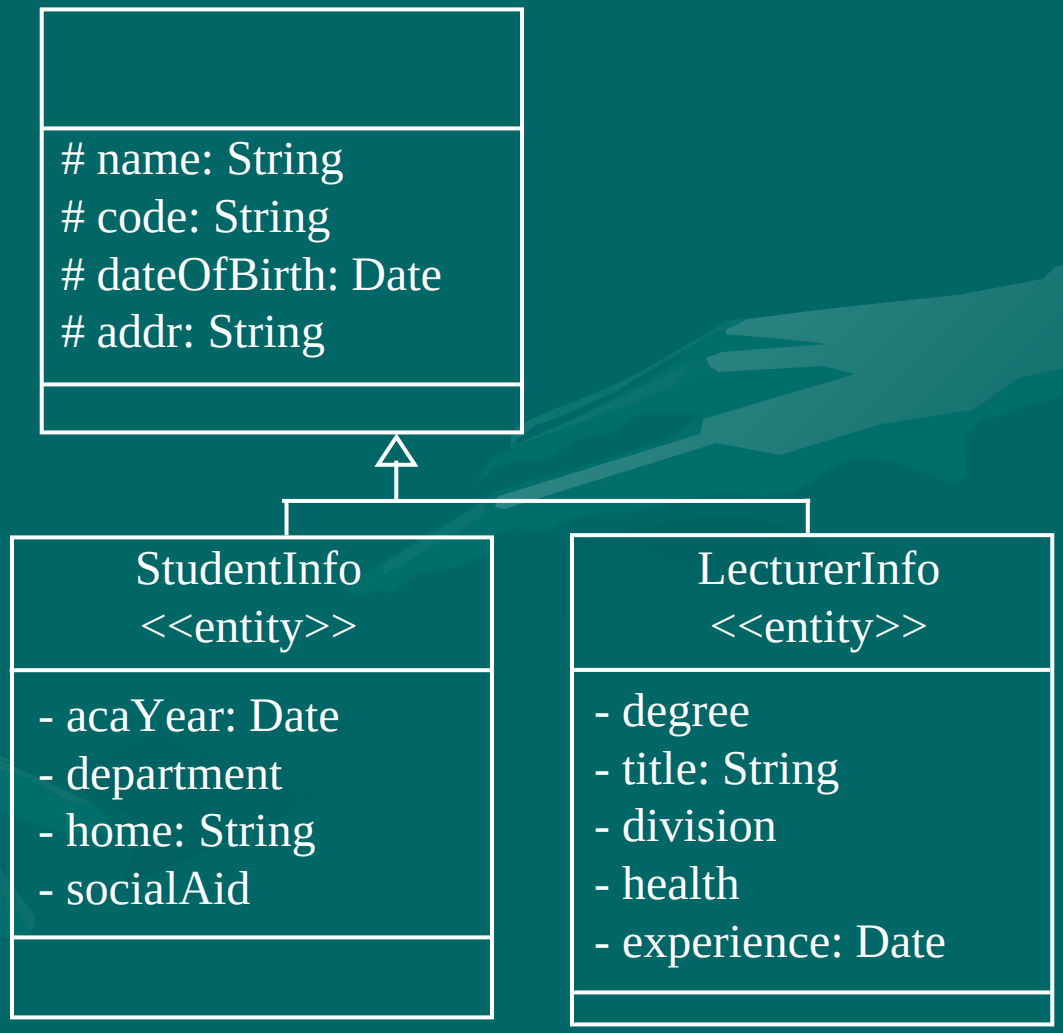


- UML định nghĩa quan hệ tổng quát hoá giữa một lớp tổng quá hơn với một lớp cụ thể hơn: lớp cụ thể hơn có tất cả thuộc tính, tác vụ và quan hệ của lớp kia + những thuộc tính/tác vụ riêng của nó
- Ký hiệu: mũi tên có đầu là một tam giác nhỏ
- Lớp tổng quát hơn nằm về phía mũi tên

Ví dụ về nhận diện lớp cơ sở

Ví dụ:

- Trong hệ thống đăng ký môn học tín chỉ qua WEB, lớp PeopleInfo là tổng quát hoá của StudentInfo và LecturerInfo



Nhận diện các mối quan hệ

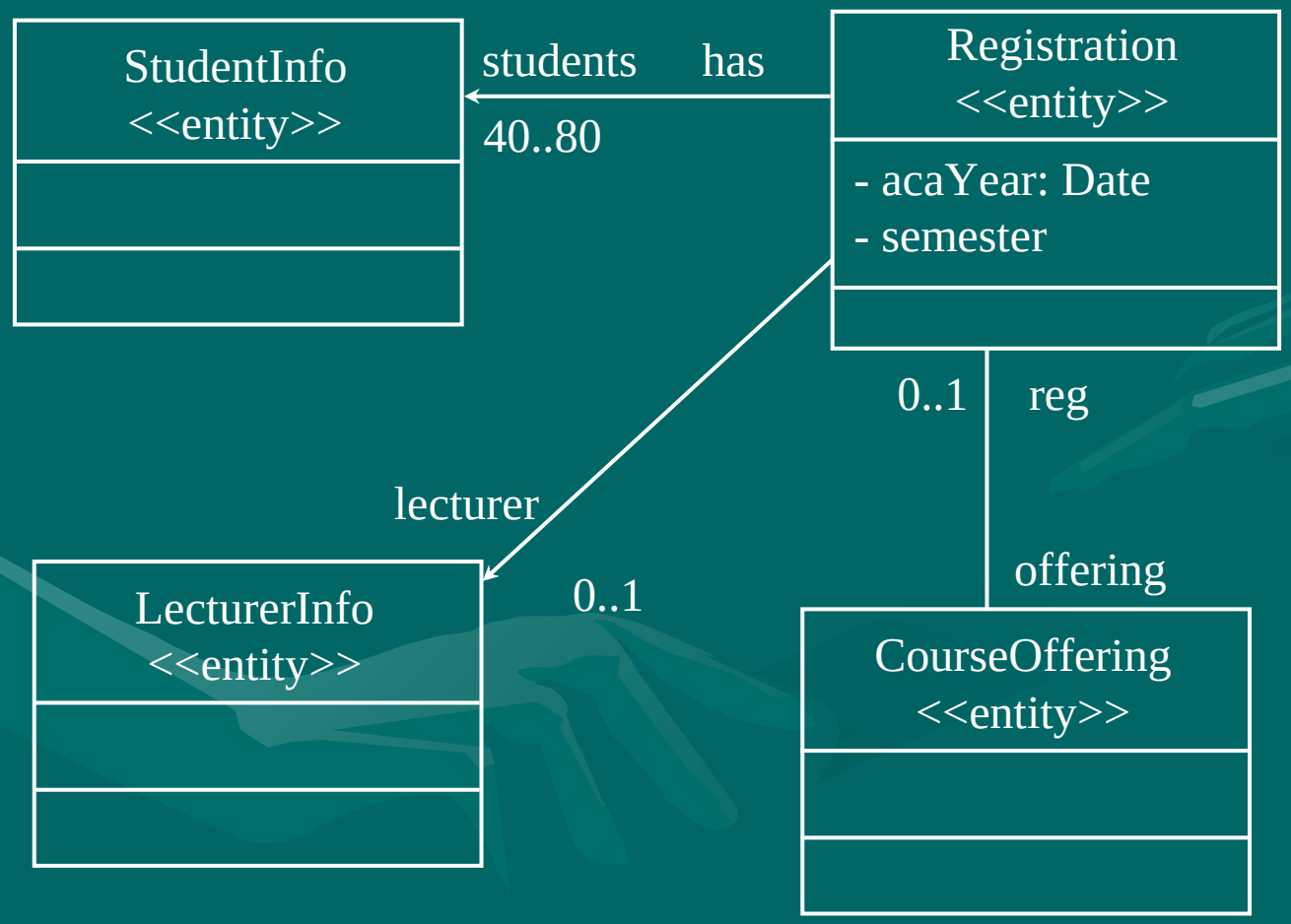
- Sau khi xác định các lớp/đối tương kể cả các đối tượng cơ sở, các quan hệ giữa các lớp cần được xác lập
- Trong mô hình phân tích các đối tượng/lớp có quan hệ với nhau
- Một số quan hệ mà UML hỗ trợ
  - **Tổng quát hoá (generalization)**
  - **Liên kết (association)**
  - **Bao gộp (aggregation)**
- Các quan hệ khác được áp dụng cho mô hình thiết kế
  - **Phụ thuộc (dependency)**
  - **Cụ thể hoá (realization)**

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Quan hệ liên kết

- Quan hệ liên kết là mối quan hệ giữa 2 đối tượng/lớp
- Về ý nghĩa và ký hiệu giống như quan hệ liên kết trong mô hình nghiệp vụ
- Áp dụng cho 2 lớp có mối tương quan mang ý nghĩa nhất định
- Chú ý ghi rõ (nếu có thể được)
  - **Bậc và tên vai trò của mỗi lớp trong quan hệ**
  - **Tên của chính quan hệ liên kết**
- Dựa vào mô hình nhgiệp vụ xác định các mối quan hệ liên kết

Ví dụ - mối quan hệ liên kết



Ví dụ:

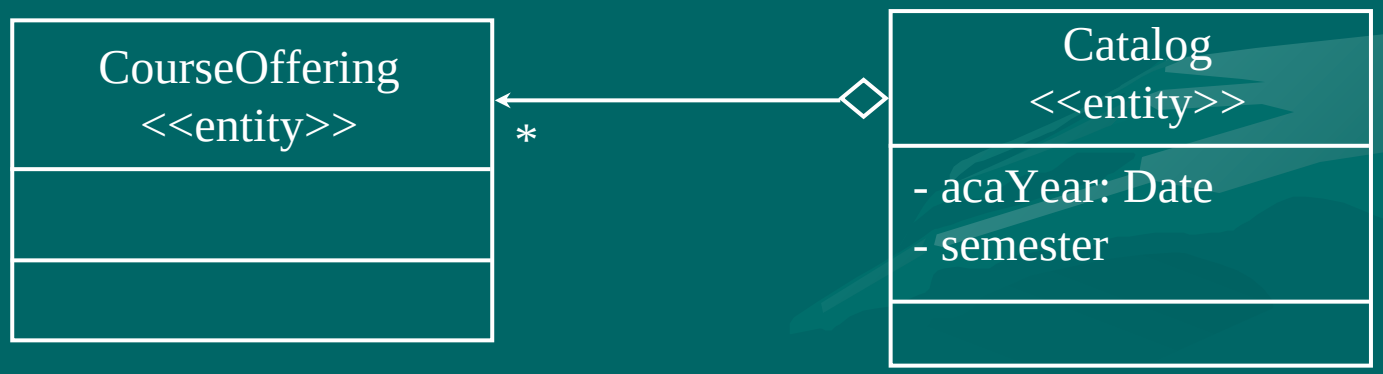
Lớp Registration liên kết với lớp StudentInfo LecturerInfo và CourseOffering

Quan hệ bao gộp

- UML định nghĩa quan hệ bao gộp là trường hợp đặc biệt của quan hệ liên kết, khi mà một đầu nối liên kết trở thành đầu nối bao gộp (aggregation)
- Lớp ở đầu nối bao gộp sẽ bao hàm lớp kia
- Ký hiệu của đầu nối bao gộp là một hình thoi tô hoặc không tô đen
- Có hai dạng bao gộp
  - **Chia xẻ (shared): chia xẻ giữa các bao gộp khác nhau**
  - **Hoàn toàn (composite): sở hữu đầy đủ**
- Xác lập các mối quan hệ bao gộm và biểu diễn chúng lên lược đồ lớp

Quan hệ bao gộp – ví dụ

- Đối với hệ thống đăng ký môn học tín chỉ qua WEB, lớp Catalog bao gộp lớp CourseOffering



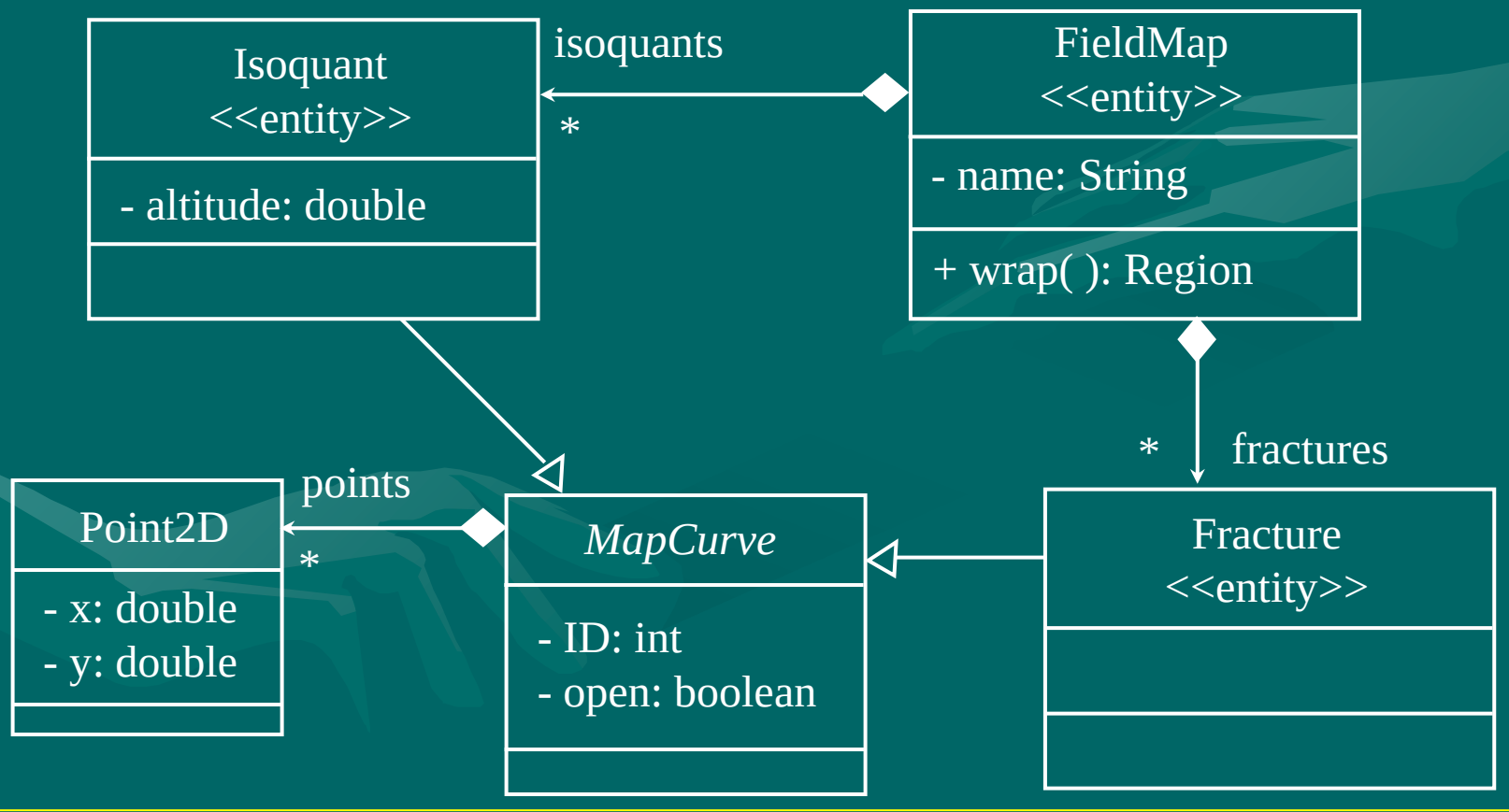
- Cửa sổ giao diện bao gộp hoàn toàn thanh cuộn và menu

Xây dựng lược đồ lớp

- Lược đồ lớp (class diagram) biểu diễn cấu trúc của một số lớp và quan hệ giữa chúng □ mô tả khía cạnh tĩnh (static) của hệ thống
- Hệ thống phức tạp có nhiều lớp □ cần xây dựng nhiều lược đồ lớp, mỗi lược đồ mô tả một phần của hệ thống
- Lược đồ lớp được bổ sung và hoàn thiện trong mô hình thiết kế (thêm một số lớp, chi tiết các thuộc tính và tác vụ, làm rõ các quan hệ)
- Lược đồ lớp được xây dựng qua các bước
  - **Xác định các lớp**
  - **Xác định thuộc tính và tác vụ của các lớp**
  - **Xác định các lớp cơ sở và quan hệ tổng quát hoá**
  - **Xác định các quan hệ liên kết và bao gộp**

Ví dụ một lược đồ lớp phân tích

- một lược đồ lớp của chương trình hiển thị bề mặt địa hình

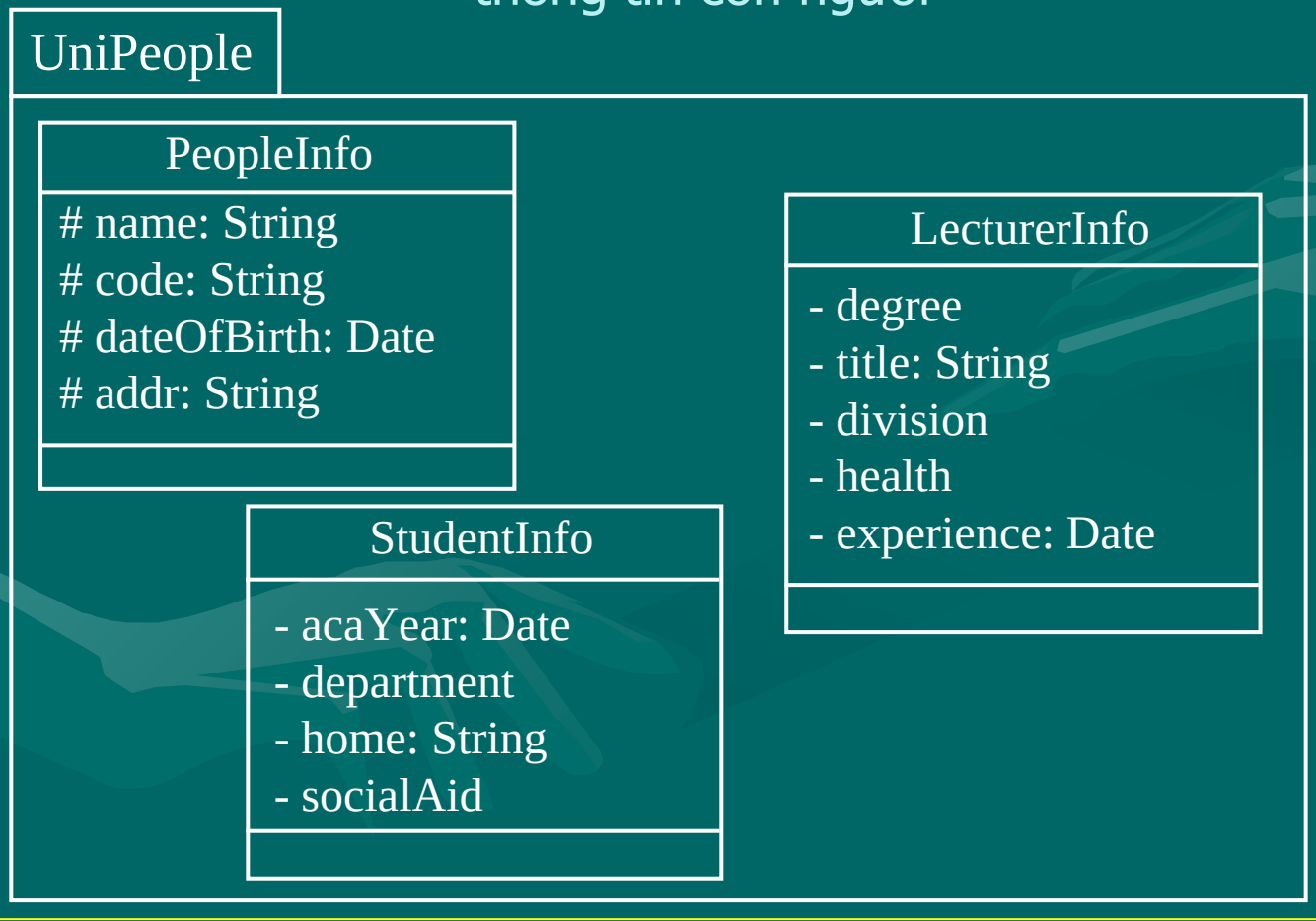


Thiết lập PACKAGE

- Package là một cơ chế để tổ chức các phần tử vào các nhóm có liên hệ về ngữ nghĩa với nhau
- Package có thể import các phần tử từ một package khác
- Có thể chỉ ra quan hệ giữa các package
  - **Phụ thuộc**
  - **Tổng quát hoá**
- Mức độ truy xuất của package
  - **Private: chỉ nó và các package import nó có thể 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**
- UML cho phép biểu diễn các PACKAGE và các mối quan hệ PACKAGE

Ví dụ về một PACKAGE

- package UniPeople chứa các lớp liên quan đến thông tin con người



Tổng kết

- Phân tích hệ thống cho OOP theo UML chia làm 2 bước:
  - **Thu thập yêu cầu bằng mô hình nhgiệp vụ**
  - **Phân tích và xác định tính năng hệ thống bằng mô hình phân tích**
- Mô hình phân tích nhận diện các đối tượng/lớp: thực thể, biên, điều khiển
- Nhận diện các thuộc tính và một số tác vụ, tuy nhiên chưa làm rõ hành vi của chúng (□ mô hình thiết kế)
- UML hỗ trợ một số phần tử: lớp, đối tượng, lược đồ lớp, package

Thiết kế phần mềm
(Software Design)

- CƠ SỞ CỦA THIẾT KẾ PHẦN MỀM
- GIAO DIỆN & TƯƠNG TÁC NGƯỜI DÙNG
- PHƯƠNG PHÁP THIẾT KẾ CỔ ĐIỂN
- PHƯƠNG PHÁP THIẾT KẾ OOP

Thiết kế phần mềm

- Thiết kế phần mềm là công việc đầu tiên của giai đoạn phát triển
- Thiết kế tạo ra các biểu diễn và dữ kiện của hệ thống phần mềm cần xây dựng từ kết quả phân tích yêu cầu để có thể dễ dàng hiện thực sau đó
- Thiết kế tạo ra phương thức và quy trình tương tác giữa người dùng với hệ thống phần mềm cũng như tương tác giữa các hệ thống khác với phần mềm.

Trừu tượng hóa

Quá trình thiết kế trải qua nhiều mức trừu tượng hoá khác nhau

- Mức cao nhất: vấn đề cần thiết kế được mô tả một cách tổng quát sử dụng thuật ngữ hướng vấn đề
- Các mức thấp hơn: hướng đến thủ tục xử lý chi tiết; kết hợp các thuật ngữ hướng đến hiện thực
- Mức thấp nhất: vấn đề được mô tả theo cách có thể hiện thực trực tiếp
- Phân loại trừu tượng hoá: thủ tục và dữ liệu

Trừu tượng hóa

- Trừu tượng hoá thủ tục: là chuỗi các lệnh liên tiếp thực hiện chức năng nào đó.

Ví dụ: **mở cửa** (bao gồm đi đến cửa, cầm lấy tay nắm, xoay tay nắm, kéo cánh cửa, đi vào…); thêm một phần tử vào danh sách có thứ tự (xác định vị trí, chèn phần tử mới)

- Trừu tượng hoá dữ liệu: là tổ hợp dữ liệu mô tả một đối tượng dữ liệu (liên hệ tới đối tượng thực thể trong UML). Ví dụ: hàng, chồng, cánh cửa...
- Một số ngôn ngữ lập trình hỗ trợ kiểu ADT và template

Tinh chế

- 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
- Thiết kế phần mềm: trừu tượng hóa rồi tinh chế hoá.
- Tại sao?

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 đáp ứng của hệ thống: giá trị trung bình và độ lệch
  - Phương tiện trợ giúp người sử dụng: tích hợp + add-on
  - Kiểm soát thông tin lỗi: hiện thị cả nguyên nhân lỗi và cách khắc phục
  - Đặt tên nhãn: ngắn gọn và gợi nhớ
- Thường dùng các công cụ thiết kế giao diện

Công cụ thiết kế giao diện

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 như tự động hiển thị ký tự đánh vào)
- Kiểm soát cửa sổ và vùng, khả năng cuộn
- Thiết lập giao tiếp giữa chương trình với giao diện (vd: hàm đáp ứng)
- Cách ly chương trình với các hàm quản lý giao diện
- Cho phép tuỳ biến giao diện: màu sắc, kích thước,..

Một số định hướng về thiết kế giao diện

Một số hướng dẫn chung

- Nên đồng nhất (menu, lệnh, hiển thị...)
- Nên cung cấp feedback cho người dùng
- Yêu cầu xác nhận những tác vụ mang tính phá hoại (xoá file, account)
- Nên hỗ trợ UNDO, REDO
- Hạn chế lượng thông tin phải ghi nhớ giữa 2 tác vụ liên tiếp
- Tối ưu trong trình bày hộp thoại và di chuyển mouse
- Chấp nhận lỗi từ phía người sử dụng
- Cung cấp trợ giúp trực tuyến
- Dùng động từ đơn giản và ngắn gọn để đặt tên các lệnh

Một số định hướng về thiết kế giao diện

Đối với thông tin hiển thị

- Chỉ hiển thị những thông tin phù hợp với ngữ cảnh hiện tại
- Dùng tên, từ viết tắt và màu gợi nhớ
- Cho phép tương tác trực quan
- Tạo thông báo lỗi có ý nghĩa
- Hiển thị dữ liệu ở nhiều dạng khác nhau trong cửa sổ
- Thiết lập biểu diễn tương tự
- Sử dụng không gian màn hình một cách tối ưu

Một số định hướng về thiết kế giao diện

Đối 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 tuỳ biến input
- Cấm các chức năng không thích hợp trong ngữ cảnh hiện tại
- Cho phép input ở nhiều dạng khác nhau
- Để cho người sử dụng kiểm soát dòng sự kiện tương tác
- Tự động tính các giá trị input cho người sử dụng nếu có thể

Thiết kế phần mềm
Phương pháp lập trình cấu trúc

Module hoá chương trình
Phân chia module
Kiến trúc phần mềm
Thiết kế dữ liệu

Thiết kế phần mềm cổ điển

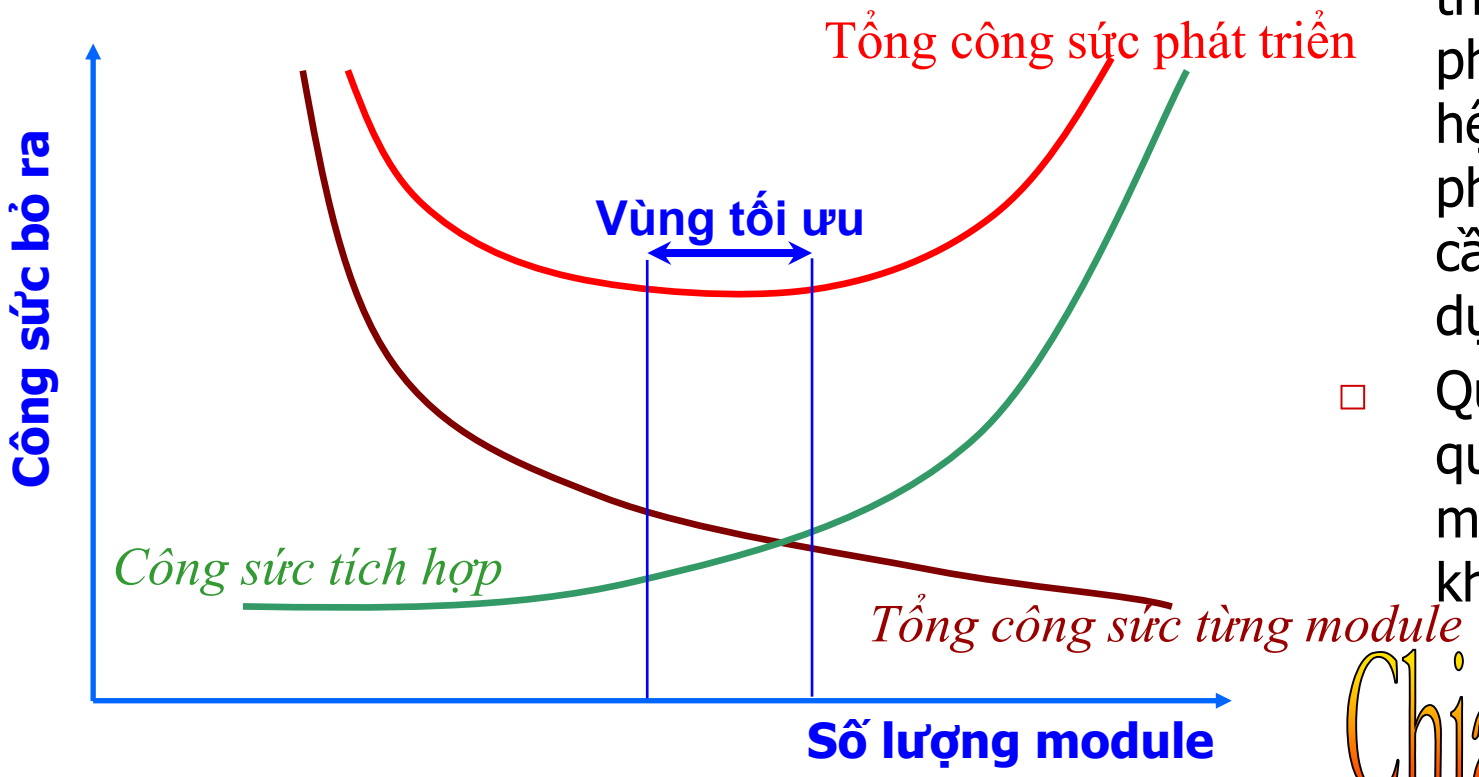
- Các công đoạn trong thiết kế phần mềm cổ điển
  - Phân chia module
  - Thiết kế dữ liệu
  - Thiết kế kiến trúc
  - Thiết kế thủ tục
  - Thiết kế giao diện
- Phần mềm phát triển theo mô hình cổ điển: quan tâm đến cấu trúc và giải thuật của các module

PHÂN CHIA *MODULE*

- Module là gì?
- Khái niệm module đã xuất hiện khoảng 4 thập niên trở lại 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 để 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) □ E(p1 + p2) > E(p1) + E(p2)

Phân chia module như thế nào?



- Số lượng module phụ thuộc vào độ phức tạp của hệ thống phần mềm cần xây dựng
- Quá ít hoặc quá nhiều module đều không tốt

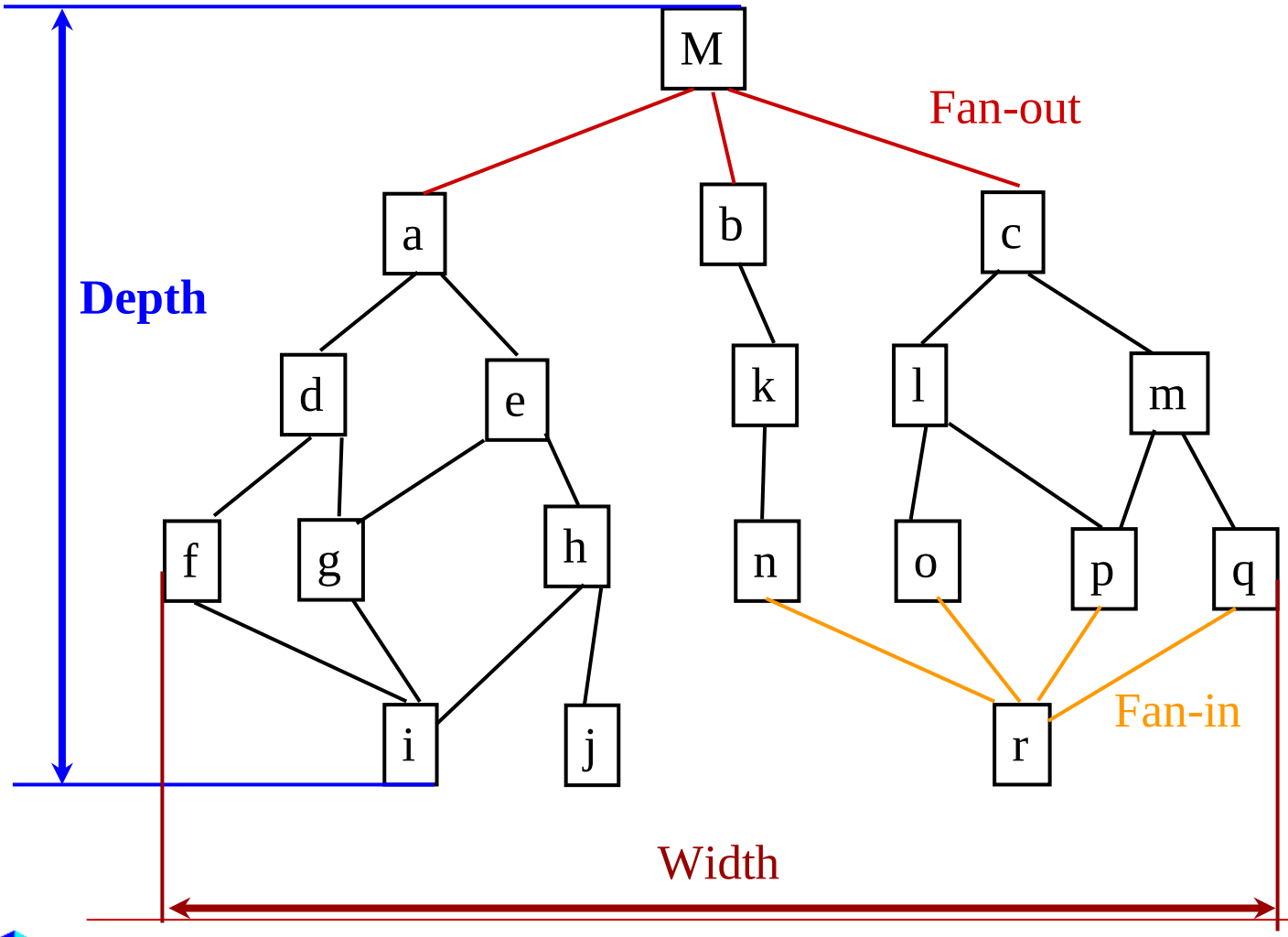
Chia thế nào?

Kiến trúc phần mềm

- Kiến trúc phần mềm mô tả các thành phần (component) kiến tạo nên hệ thống phần mềm và sự 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- Sơ đồ phân cấp



- Sơ đồ phân cấp được dùng để miêu tả sự phân rã các module.

Phân chia module hiệu quả

- Phân chia module là bắt buộc trong giai đoạn thiết kế
- Tuy nhiên: phân chia kiến trúc phần mềm thành một bộ các module như thế nào là tốt nhất ?

  Đạt được vùng tối ưu về tổng chi phí quản lý và phát triển từng module
- Tiêu chí quan trọng nhất: tính độc lập chức năng của các module
- Tính độc lập chức năng được đo bằng 2 tiêu chuẩn:
  - Độ kết dính (cohesion)
  - Sự liên kết (coupling)

Độ kết dính

- Độ kết dính dùng để đo sự phụ thuộc lẫn nhau giữa những tác vụ (task) của một module
- Module có độ kết dính cao nhất khi nó chỉ đảm nhận đúng một tác vụ □ kết dính chức năng
- Thiết kế kiến trúc phần mềm: cố gắng tăng độ kết dính

Các mức độ kết dính

Có nhiều mức độ kết dính (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 được thực thi trong một khoảng thời gian
- Giao tiếp: các tác vụ có sử dụng chung một dữ liệu nào đó
- Thủ tục: các tác vụ phải được thực hiện theo một trật tự nhất định
- Chức năng: chỉ có một tác vụ

Sự liên kết

- Sự liên kết dùng để đo đạc quá trình giao tiếp giữa các module: giao tiếp của module chứa nhiều tác vụ và nhiều thông số gọi thì sự liên kết càng cao

- Thiết kế kiến trúc phần mềm: cố gắng giảm sự liên kết

Các mức độ liên kết

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

Nguyên lý che dấu 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 sự cần thiết
- Những thông tin về thủ tục 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

Các Heuristic trong phân chia module

- Sửa lại thiết kế ban đầu để tăng độ kết dính và giảm sự liên kết
- Khi chiều sâu tăng, hạn chế fan-out trong khi sử dụng fan-in
- Giữ cho tầm ảnh hưởng của một module nằm bên trong tầm điều khiển của nó
- Loại bỏ dư thừa trong giao tiếp của các module
- Ưu tiên các module tất định, hạn chế các module nhiều ràng buộc
- Đóng gói các module để đạt được tính khả chuyển (portability)

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Thiết kế dữ liệu

- Tìm kiếm biểu diễn luận lý cho các phần tử dữ liệu đã được nhận diện trong giai đoạn phân tích yêu cầu
- Thiết kế các cấu trúc dữ liệu của chương trình và cơ sở dữ liệu
- Thực hiện tinh chế từng bước
- Các tiêu chí thiết kế hệ cơ sở dữ liệu
  - Không dư thừa dữ liệu
  - Tối ưu hóa không gian lưu trữ
- Chuẩn hóa cơ sở dữ liệu bằng lược đồ ERD và dạng chuẩn 3

Cấu trúc dữ liệu

Cấu trúc dữ liệu là thiết kế cơ sở dữ liệu động tron hệ thống

- Cấu trúc dữ liệu mô tả sự tổ chức, phương thức truy xuất, mức độ liên kết và các xử lý khác của thông tin
- Dữ liệu đơn là dạng cấu trúc dữ liệu đơn giản nhất chỉ bao gồm một phần tử thông tin mà có thể được truy xuất bằng một danh định
- Một số dạng phức tạp hơn: vector, ma trận, mảng nhiều chiều, danh sách liên kết, hàng, chồng, cây nhị phân…
- Được biểu diễn ở các mức trừu tượng hoá khác nhau

Mộ số nguyên tắc thiết kế dữ liệu

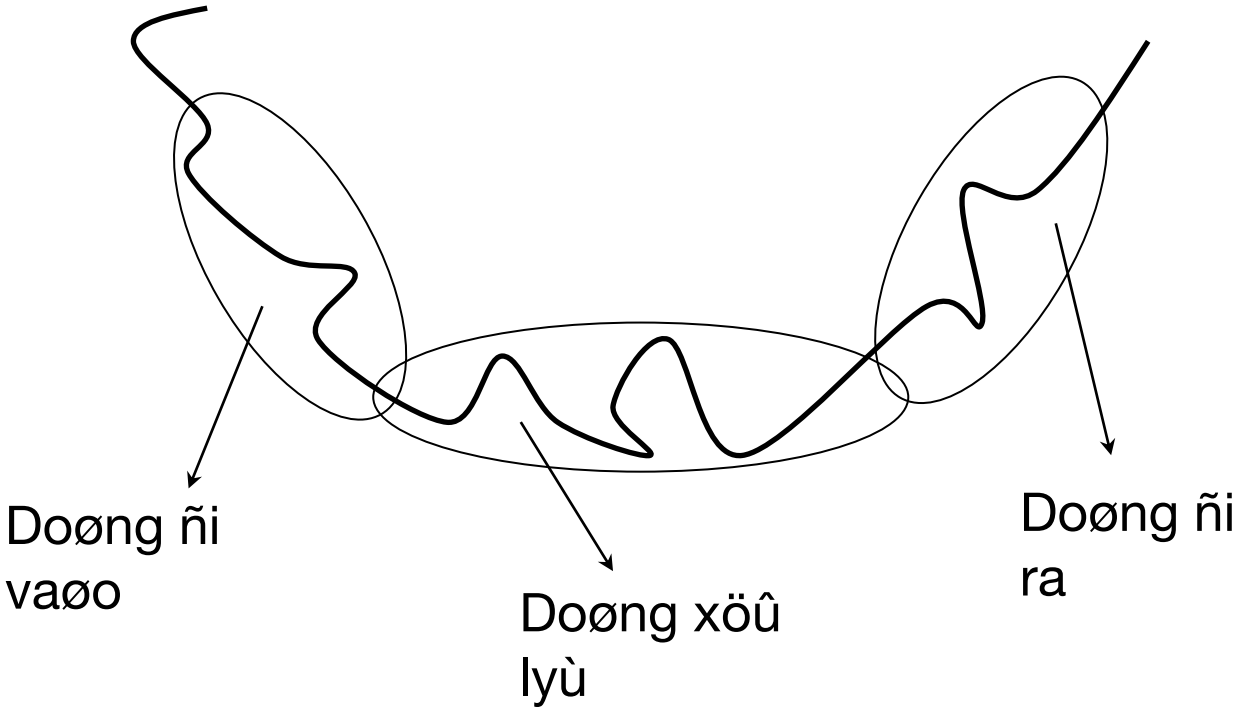
- Một số nguyên tắc
- Nhận diện cả cấu trúc dữ liệu và tác vụ truy xuất
- Chú ý sử dụng từ điển dữ liệu
- Trì hoãn thiết kế dữ liệu mức thấp cho đến cuối giai đoạn này
- Che dấu biểu diễn bên trong của cấu trúc dữ liệu
- Phát triển một thư viện các cấu trúc dữ liệu + tác vụ thường gặp
- Nên áp dung kiểu ADT trong thiết kế cũng như trong lập trình

Thiết kế kiến trúc

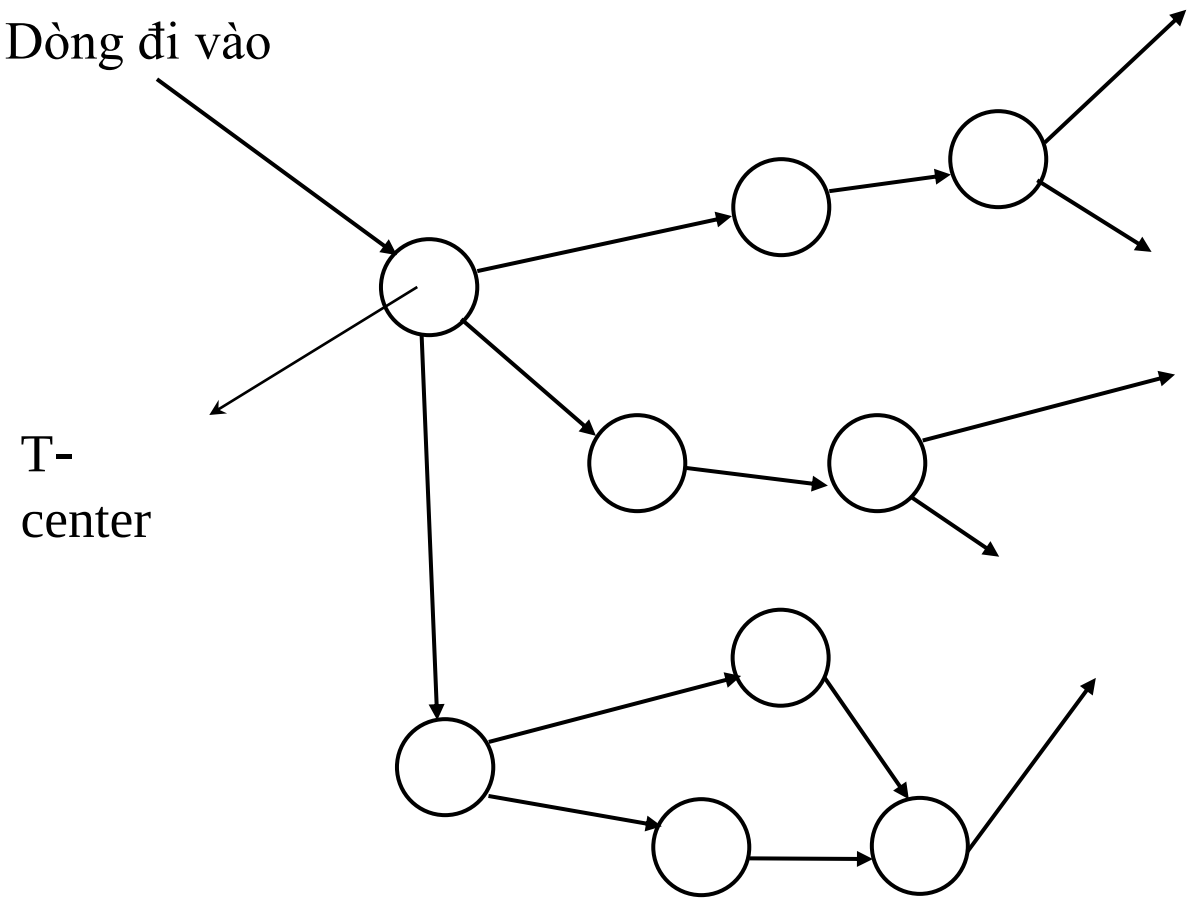
- Mục tiêu là xây dựng sơ đồ phân cấp module từ DFD
- Đặt nền móng để thiết kế chi tiết thủ tục và dữ liệu
- Phân biệt dòng transform và dòng transaction trong DFD
- Thực hiện ánh xạ cho từng vùng của DFD tuỳ theo nó là dòng transform hay transaction
  - Xác định các module và các mối liên hệ theo DFD quá trình sử lý

Dòng Transform

- Dòng transform bao gồm 3 phần: dòng đi vào, dòng xử lý và dòng đi ra



Dòng Transaction



- Dòng transaction bao gồm: dòng đi vào, T-center và các đường xử lý đầu ra
- T-center: Chỉ có một đường ra được kích hoạt tại một thời điểm

Thiết kế thủ tục

- Thiết lập thuật giải cho các module đã kiến tạo sao cho có thể dễ dàng mã hoá bằng ngôn ngữ lập trình có cấu trúc

- Có thể biểu diễn thuât giải bằng
  - Lưu đồ thuật giải: Flowchart
  - Ký hiệu dạng bảng
  - 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 có cấu trúc. Nó có các tính chất sau:
- Cú pháp chặt chẽ của các từ khoá hỗ trợ đặc tả cấu trúc, khai báo dữ liệu, phân chia module
- Cú pháp tự do của ngôn ngữ tự nhiên giúp miêu tả xử lý
- Phương tiện mô tả dữ liệu đơn cũng như dữ liệu tổ hợp
- Cơ chế định nghĩa chương trình con và phương cách gọi

Ngôn ngữ PDL – Ví dụ

```
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"
    else
        type := "Error"
end
```


Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Thiết kế phần mềm UML

Lược đồ cộng tác
Lược đồ tuần tự
Lược đồ trạng thái các lớp/đối tượng
Lược đồ hành động
Hoàn chỉnh lược đồ lớp

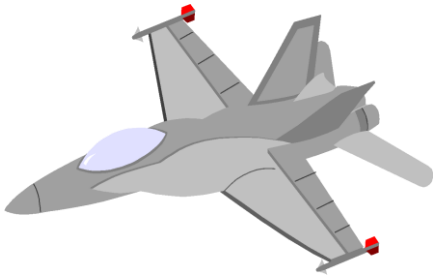
Giới thiệu

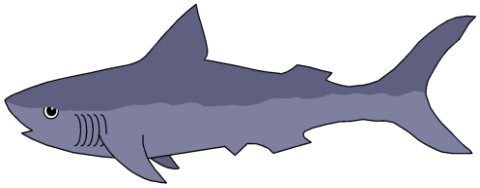
- Giai đoạn thiết kế quan tâm đến "HOW":
  - Thứ tự các thông điệp trao đổi, thông số của thông điệp
  - Thuật giải của tác vụ đáp ứng
  - Cấu trúc dữ liệu cho các thuộc tính
  - Framework (console, document/view, 3-tier...)

- Thiết kế cũng chịu ảnh hưởng từ:
  - Ngôn ngữ lập trình và thư viện lập trình (Hỗ trợ Vector, List, Map... hay không ? Hỗ trợ template hay không ?...)
  - Kiến trúc hệ thống (COM, CORBA hay EJB)
- Thiết lập mô hình động (dynamic modeling) và chi tiết hoá mô hình tĩnh

Khái niệm mô hình động

- Lược đồ lớp chỉ mô tả khía cạnh tĩnh của hệ thống
- Hành vi của hệ thống được mô tả bằng mô hình động bao gồm
  - Tương tác giữa các đối tượng: cộng tác hay trình tự
  - Trạng thái của đối tượng/lớp
  - Quá trình hoạt động của lớp/đối tượng





Tương tác giữa các đối tượng

- Đối tượng tương tác với nhau (*interaction*) bằng cách gửi/nhận kích thích (*stimulus*)
- *Actor* cũng có thể gửi kích thích đến đối tượng
- Kích thích khiến một tác vụ thực thi, một đối tượng được tạo ra hay huỷ đi, hoặc gây ra một tín hiệu
- Thông điệp (*message*) là đặc tả của kích thích
- Các loại thông điệp
  - Đơn giản
  - Đồng bộ
  - Bất đồng bộ
  - Trả về của gọi hàm

Sự cộng tác – Lược đồ cộng tác

- Cộng tác (*collaboration*) định nghĩa tập hợp các thành phần tham gia và quan hệ giữa chúng
- Các thành phần tham gia là vai trò mà đối tượng/lớp đóng vai khi tương tác với nhau
- Các vai trò của đối tượng thường chỉ có nghĩa đối với một mục đích nào đó
- Lược đồ cộng tác (*collaboration diagram*) được thiết lập để cụ thể hoá một *use-case* hoặc một tác vụ

Lược đồ cộng tác

- Lược đồ cộng tác là một đồ thị liên kết các vai trò của các đối tượng hình thành nên các chức năng của hệ thống (các usecase)
- Mỗi cạnh liên kết 2 vai trò được biểu diễn bằng một đoạn thẳng
- Tương tác được thể hiện bằng gửi/nhận thông điệp
- Hai vai trò liên kết với nhau khi có trao đổi thông điệp
- Mỗi thông điệp được thể hiện bằng mũi tên (như đã miêu tả) cộng với phần đặc tả

Lược đồ cộng tác (tt)

- Các thông điệp được đánh số theo kiểu phân cấp
  - 3.4.2 xảy ra sau 3.4.1 và cả hai được lồng (*nested*) trong 3.4
  - 3.4.3a và 3.4.3b xảy ra đồng thời và được lồng trong 3.4
- Cú pháp tổng quát của thông điệp

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

- Ví dụ: 2/ 1.3.1: p := find(specs)

Lược đồ cộng tác (tt)

- Lược đồ cộng tác có thể được thiết lập ở một trong 2 dạng:

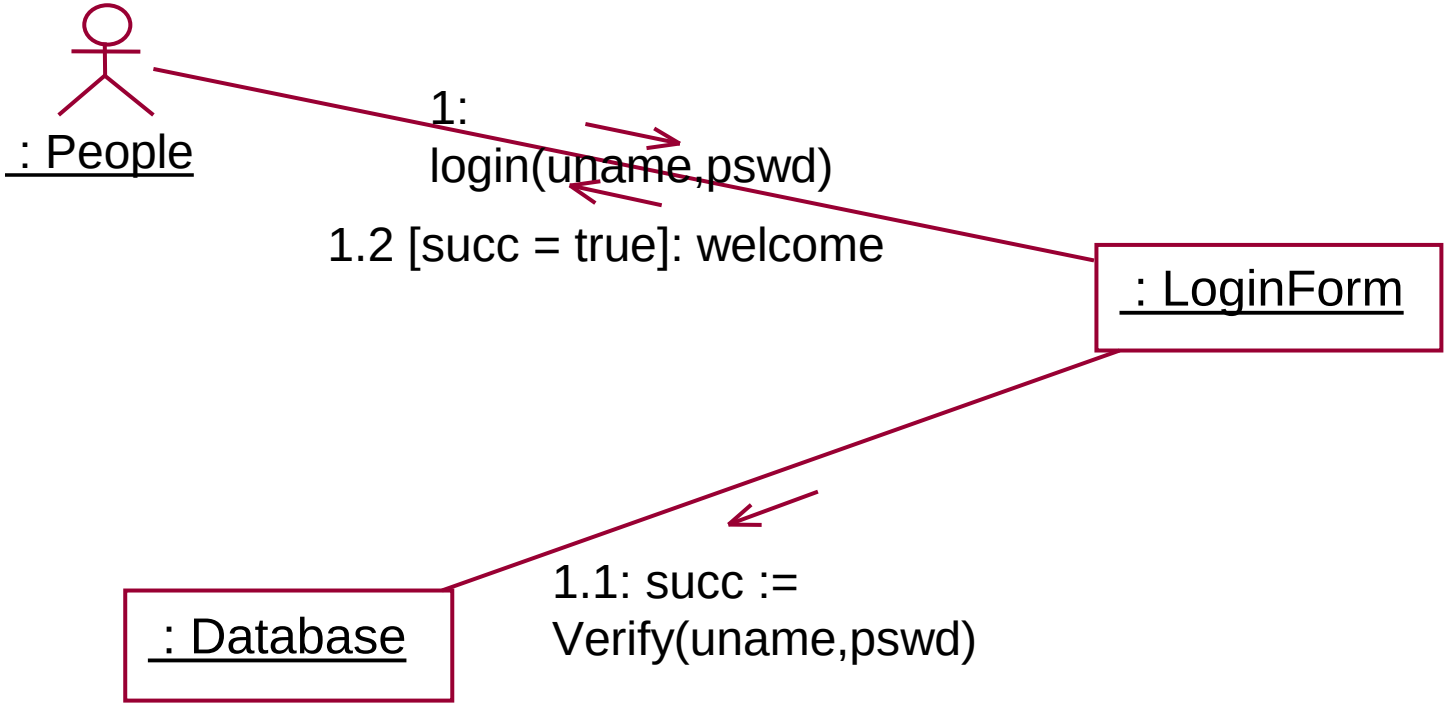
Dạng cụ thể: mỗi vai trò được biểu diễn bằng một ký hiệu của đối tượng cụ thể, các thông điệp được trao đổi trên các đường liên kết

Dạng đặc tả: mô tả các lớp; các đường liên kết được ánh xạ vào các thông điệp

- Thiết lập lược đồ cộng tác giúp cụ thể hoá (realize) các use-case và nhận diện thêm một số tác vụ của các đối tượng/lớp phân tích

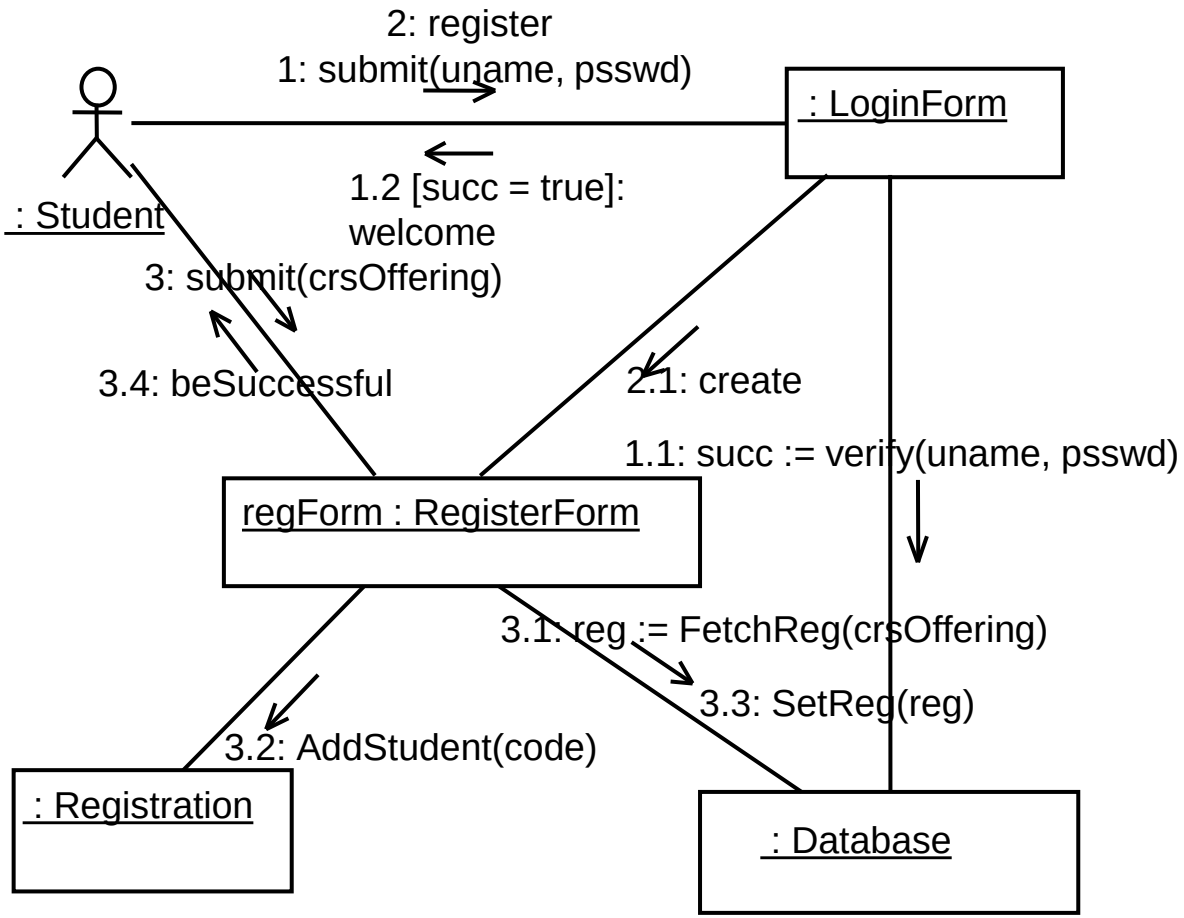
Lược đồ cộng tác – Ví dụ

- Ví dụ: lược đồ cộng tác mức cụ thể cho *use-case* Login của hệ thống đăng ký môn học tín chỉ qua WEB



Lược đồ cộng tác – ví dụ 2

Ví dụ: lược đồ cộng tác mức cụ thể cho use-case Registers course của hệ thống đăng ký môn học tín chỉ qua WEB



Miêu tả trình tự

- Lược đồ cộng tác miêu tả sự tương tác theo khía cạnh không gian
- Để nhấn mạnh trình tự của tương tác ▪ dùng lược đồ tuần tự (sequence diagram)
- Lược đồ tuần tự miêu tả các đối tượng tương tác với nhau theo thời gian sống của nó
- Các thông điệp được trao đổi theo trình tự thời gian
- Các mối liên kết không được thể hiện trong lược đồ

Lược đồ tuần tự

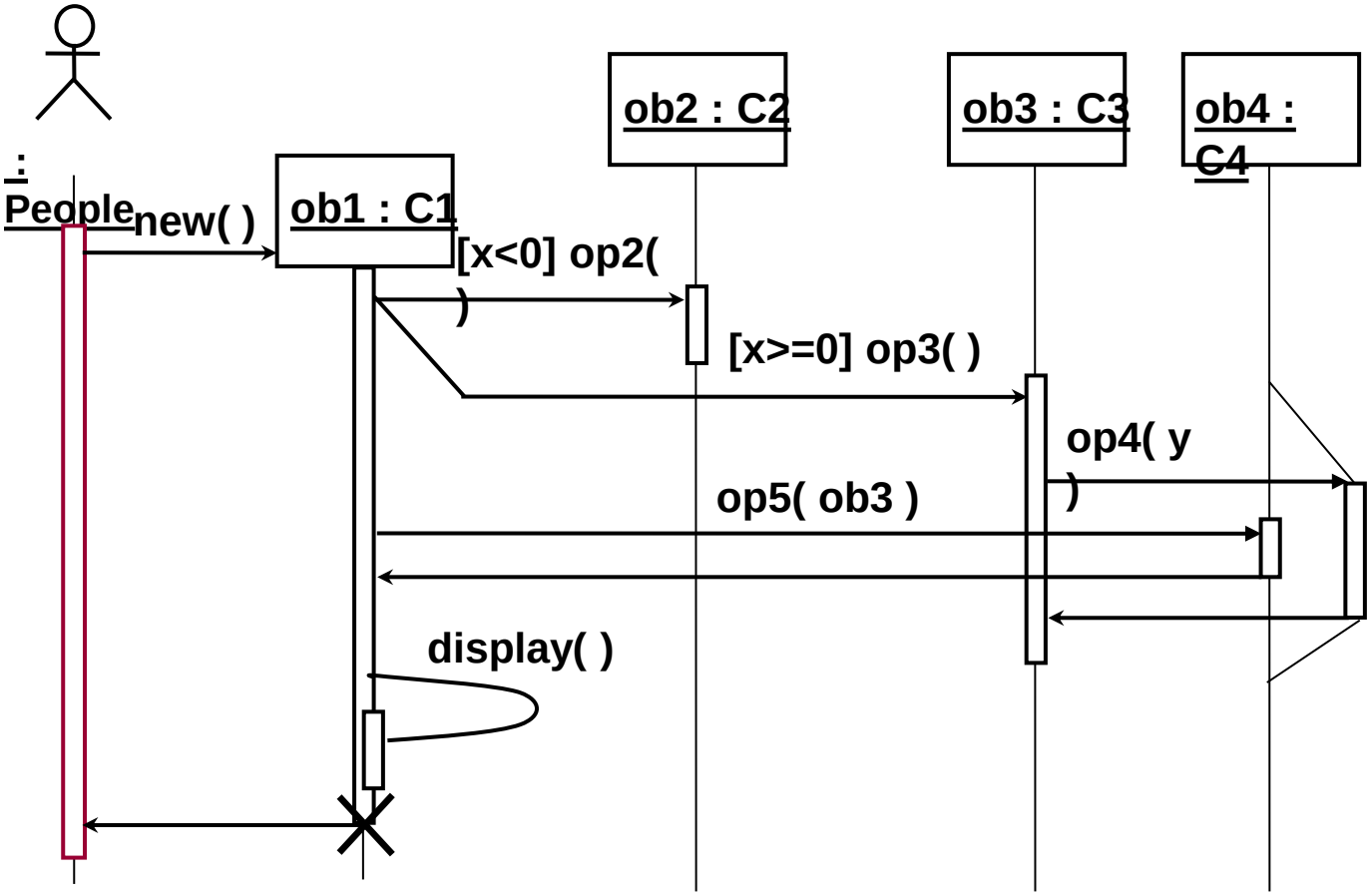
- Lược đồ tuần tự có 2 dạng

  Dạng tổng quát: thể hiện cả vòng lặp và rẽ nhánh

  Dạng cụ thể: miêu tả một kịch bản cụ thể
- Thời gian sống của mỗi đối tượng được mô tả theo một đường thẳng đứng
- Thông thường thời gian trôi theo chiều từ trên xuống dưới
- Ít khi quan tâm đến khoảng thời gian, thường chỉ quan tâm đến trình tự mà thôi
- Thanh hình chữ nhật mô tả sự thực thi của một tác vụ để đáp ứng lại thông điệp gửi đến. Độ dài của thanh chữ nhật phản ánh thời gian thực thi của tác vụ và tính chất lồng nhau (nested) giữa chúng
- Các dòng text phụ trợ (mô tả tác vụ, ràng buộc thời gian...) được viết ở lề trái

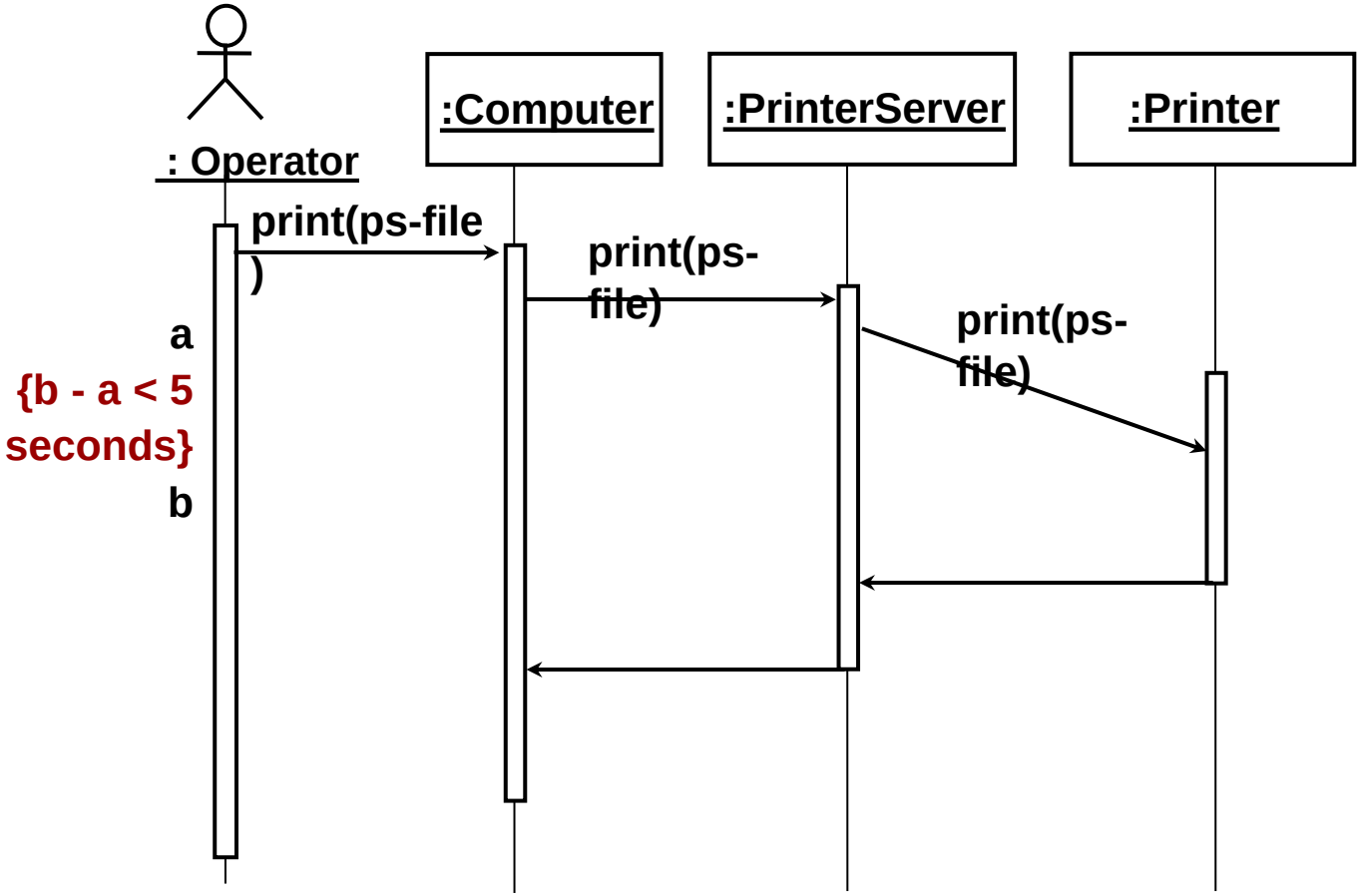
Lược đồ tuần tự - Ví dụ

- Ví dụ: lược đồ tuần tự dạng tổng quát



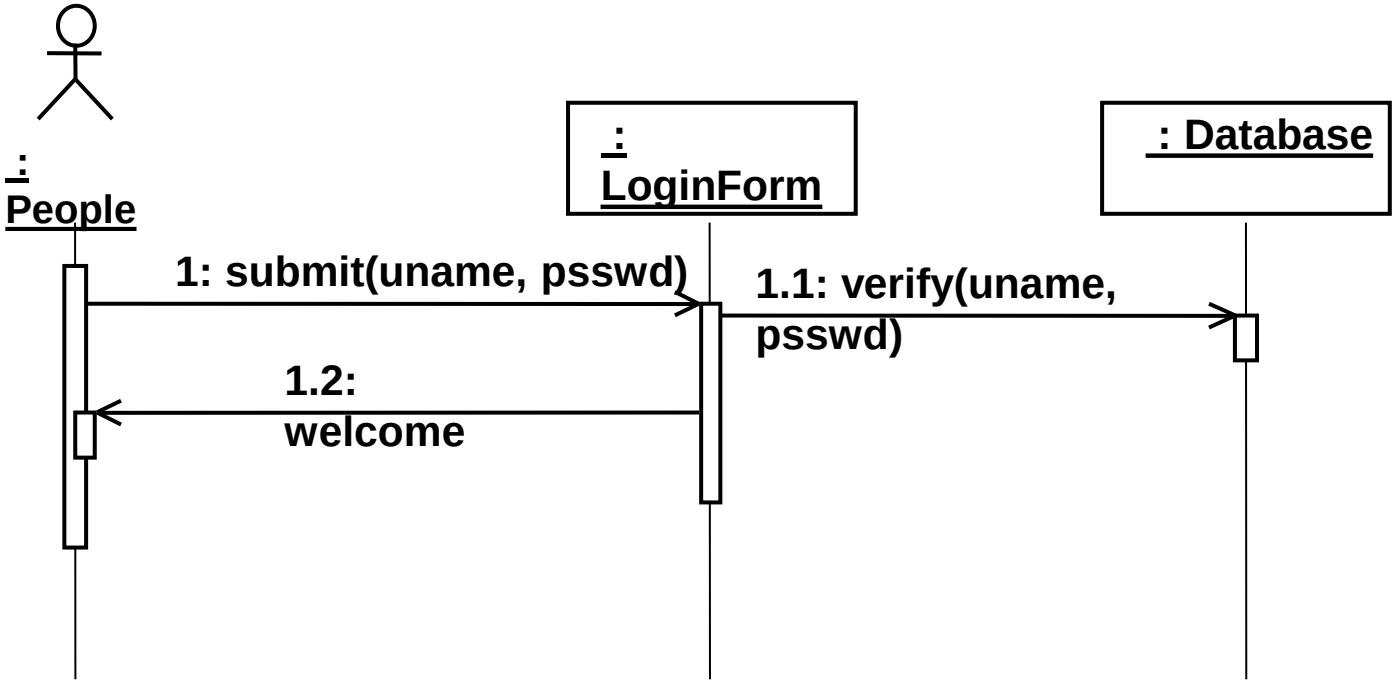
Lược đồ tuần tự - Ví dụ

- **Lược đồ tuần tự với các ghi chú ràng buộc thời gian**

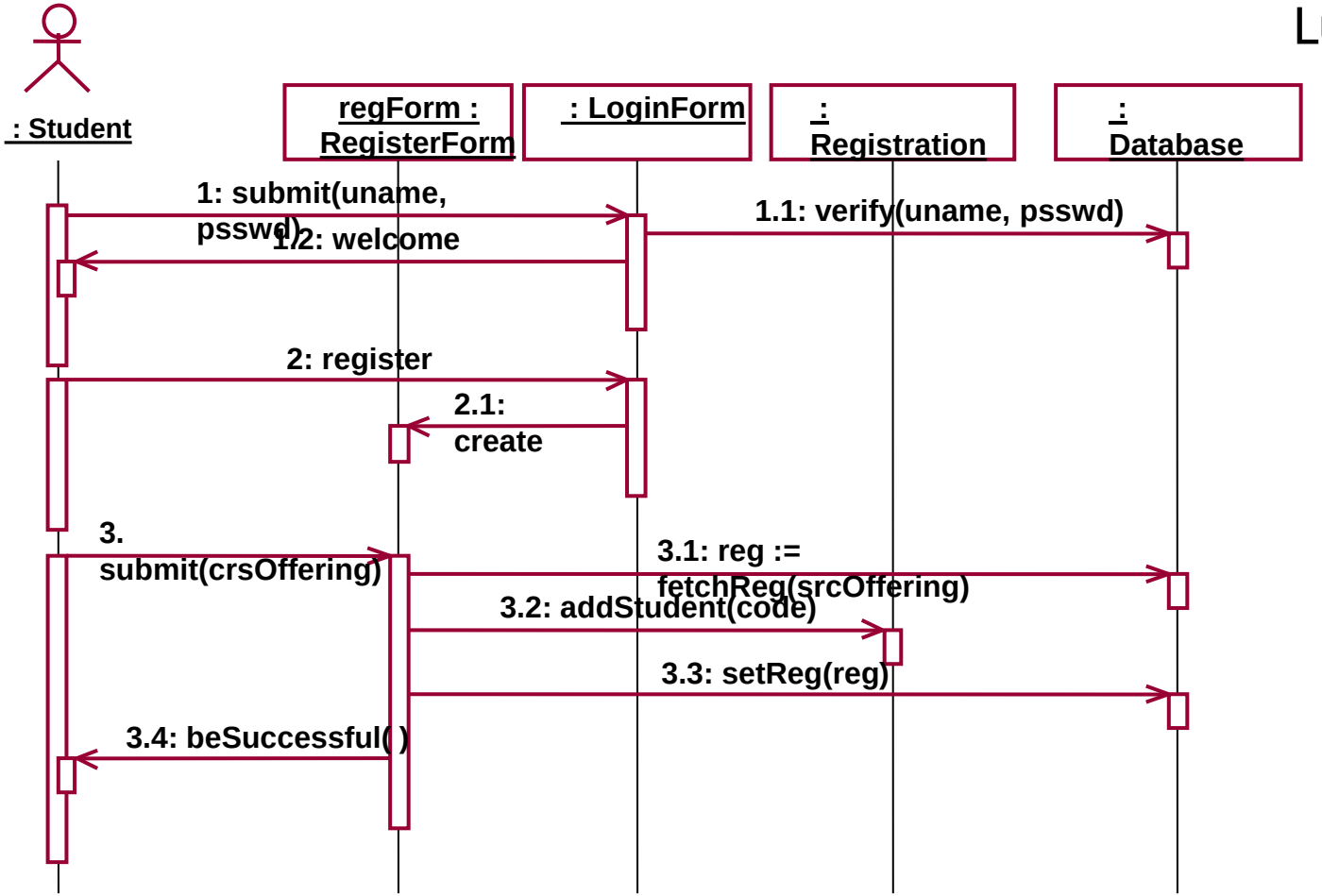


Lược đồ tuần tự - Ví dụ

- Ví dụ: lược đồ tuần tự dạng cụ thể cho use-case Login của hệ thống đăng ký môn học tín chỉ qua WEB



Lược đồ tuần tự - Ví dụ



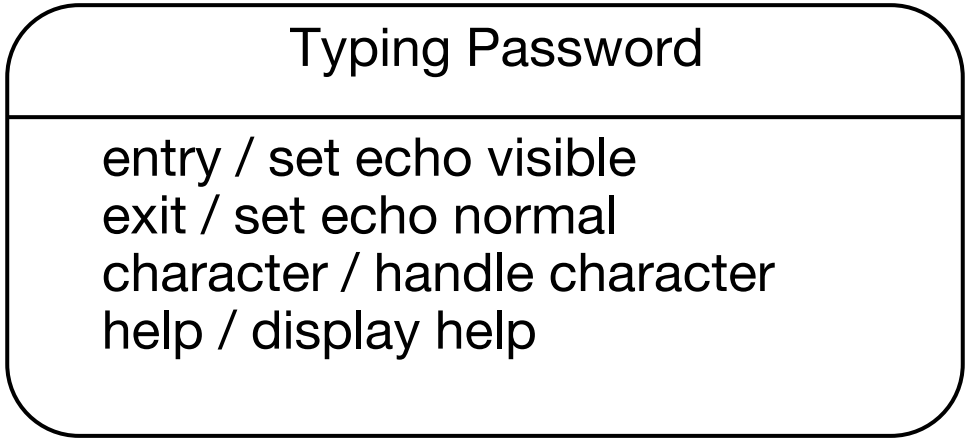
Lược đồ tuần tự dạng cụ thể cho *use-case* Register courses

Trạng thái của đối tượng

- Chuẩn UML đưa ra lược đồ trạng thái để biểu diễn hành vi của một phần tử bất kỳ bằng cách chỉ ra đáp ứng của nó đối với các sự kiện bên ngoài
- Thông thường lược đồ trạng thái được áp dụng cho đối tượng/lớp □ biểu diễn hành vi của lớp
- Trạng thái của mỗi đối tượng (định nghĩa gốc ?) ít nhiều sẽ bị thay đổi trong suốt chu kỳ sống của đối tượng

Trạng thái của đối tượng

- Trong UML ký hiệu của trạng thái là một hình chữ nhật tròn góc và được chia làm nhiều phần phân cách nhau bằng các đoạn thẳng nằm ngang:
  - Phần tên
  - Phần miêu tả các hành động bên trong



Trạng thái của đối tượng

- Tên trạng thái là duy nhất trong lược đồ; có thể không có tên (trạng thái vô danh)
- Các hành động bên trong: các hành động hoặc tác vụ được thực hiện khi đối tượng nằm ở trạng thái đang xét; có cú pháp như sau

  action-label '/' action-expression
- Một số nhãn hành động (action-label) được quy ước trước:
  - entry: thực hiện hành động tại thời điểm bắt đầu trạng thái
  - exit: thực hiện hành động tại thời điểm kết thúc trạng thái
  - do: thực hiện hành động suốt trạng thái hoặc cho đến khi kết thúc nó
  - include: triệu gọi một máy trạng thái con khác

Trạng thái của đối tượng

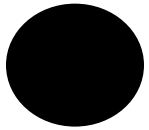
- Các nhãn hành động khác chỉ ra sự kiện kích hoạt hành động tương ứng trong biểu thức hành động (action-expression)

- Cú pháp của biểu thức hành động

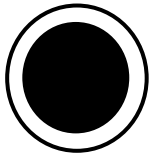
event-name '(' parameter-list ')' '['guard-condition']' '/' action-expression

Lược đồ trạng thái

- Trạng thái bắt đầu: khi đối tượng được tạo ra hoặc trạng thái tổng hợp được xác định; ký hiệu



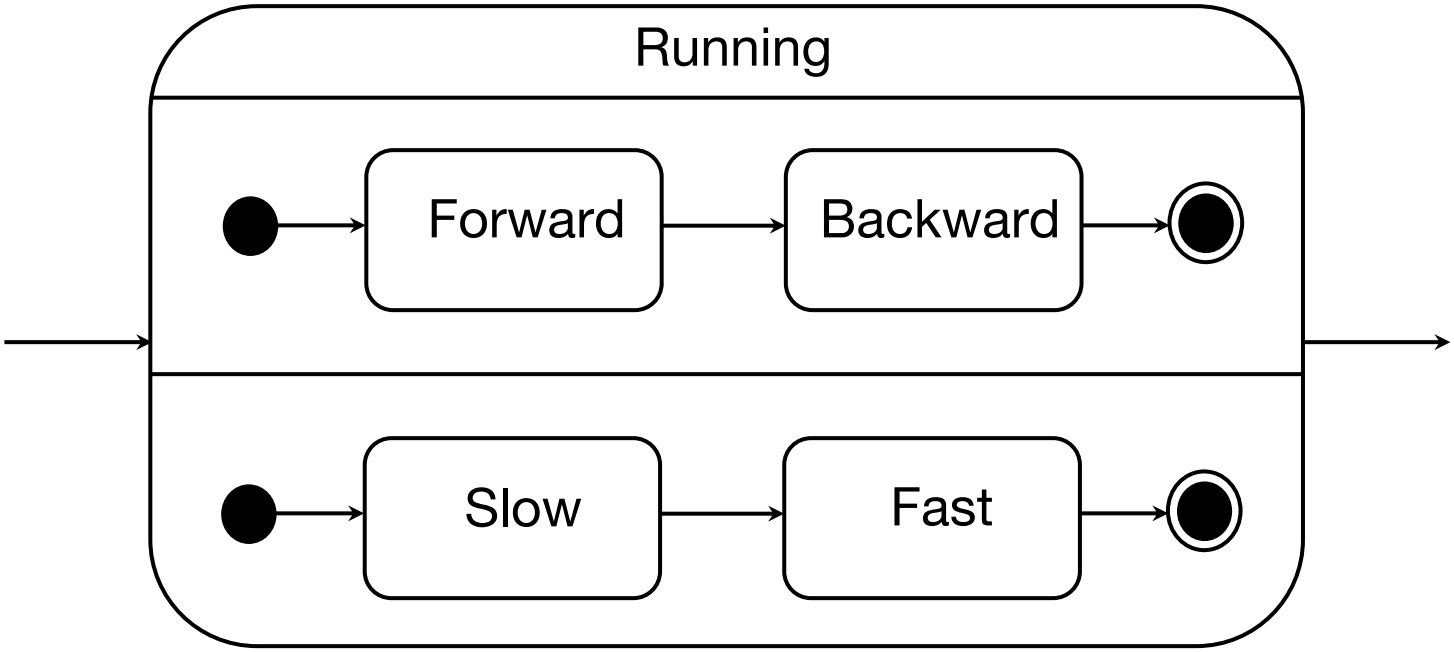
- Trạng thái kết thúc: khi đối tượng bị huỷ bỏ hoặc trạng thái tổng hợp trở nên không xác định; ký hiệu



- Trạng thái tổng hợp (composite) được phân rã thành nhiều trạng thái con đồng thời hoặc các trạng thái con loại trừ nhau

Lược đồ trạng thái – trạng thái tổng hợp

- Phân rã trạng thái tổng hợp Running



Lược đồ trạng thái

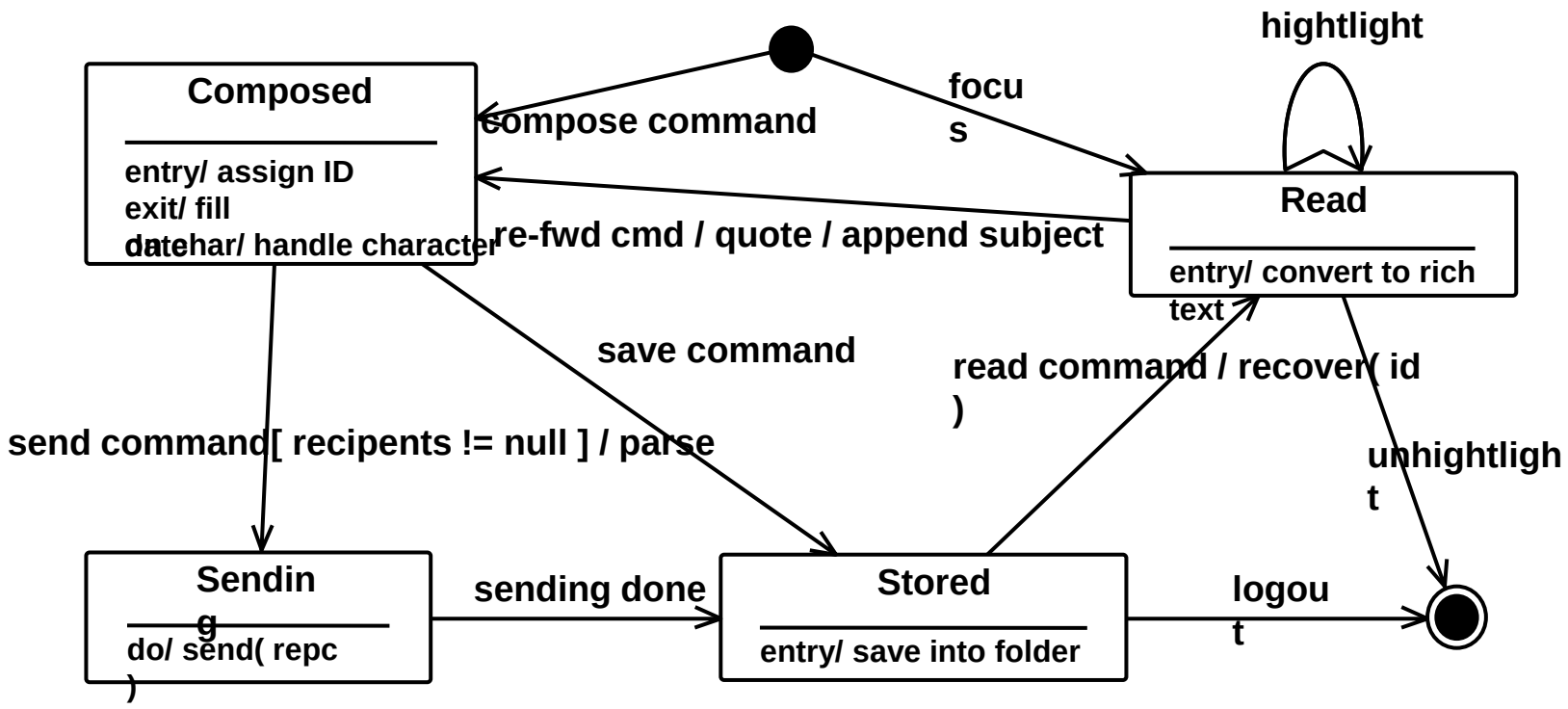
- Sự kiện (event) kích hoạt dịch chuyển trạng thái, có thể là
  - Một điều kiện trở nên đúng (chú ý khác với guard-condition)
  - Một đối tượng nhận tín hiệu từ đối tượng khác
  - Một phép gọi tác vụ
  - Một khoảng thời gian đã trôi qua kể từ một sự kiện nào đó
- Cú pháp của sự kiện: event-name '(' parameter-list ')'
- Sự kiện có tầm vực thuộc về package chứa lớp đang mô tả lược đồ trạng thái, chứ không chỉ thuộc về riêng lớp đó

Lược đồ trạng thái – Dịch chuyển

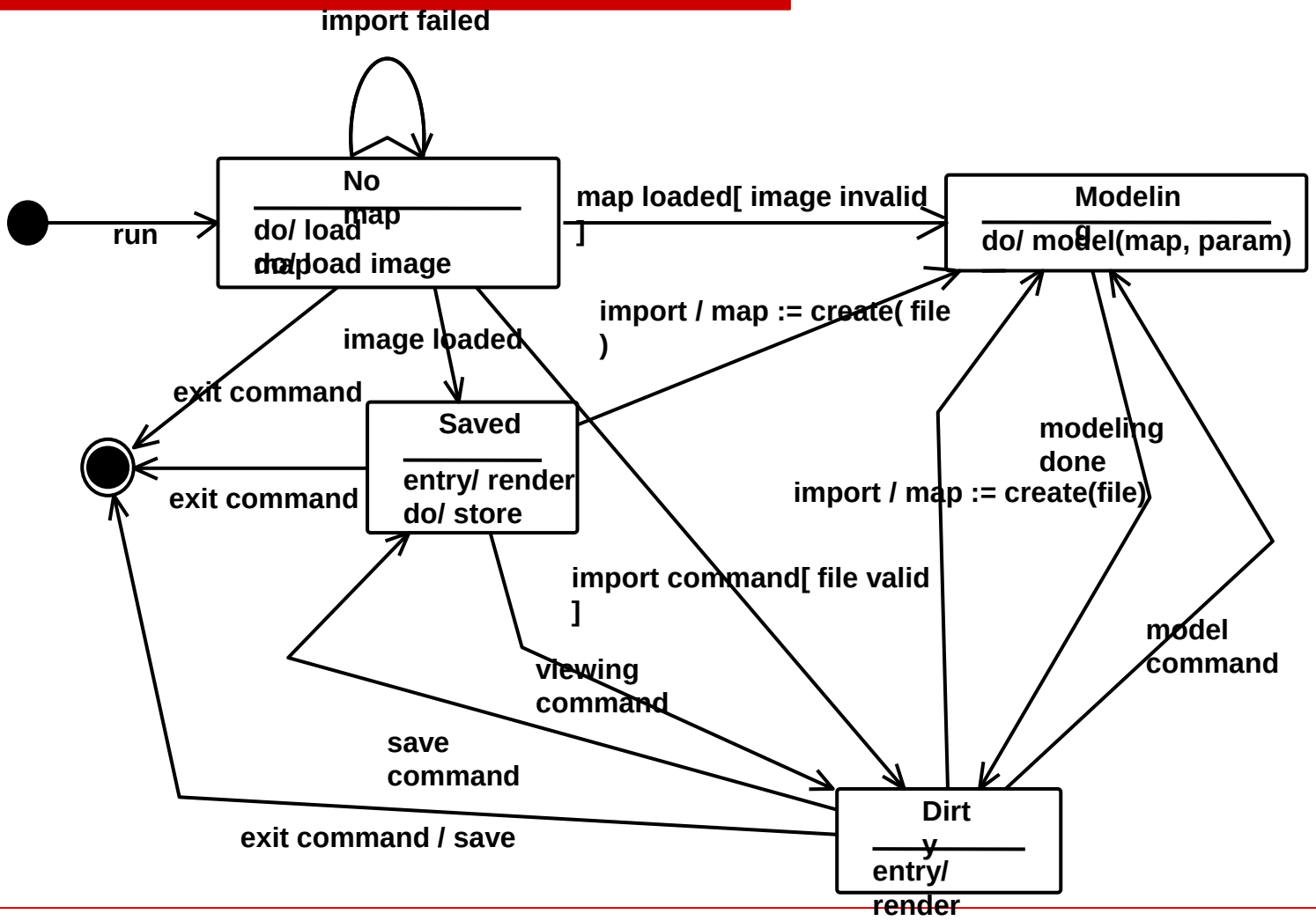
- Dịch chuyển trạng thái là quan hệ giữa hai trạng thái theo đó đối tượng đang ở trạng thái thứ nhất sẽ chuyển sang trạng thái thứ hai đồng thời sẽ thực hiện một số hành động khi sự kiện tương ứng xảy ra và thoả mãn một số điều kiện nhất định
- Được ký hiệu như một mũi tên hướng từ trạng thái nguồn đến trạng thái đích và được gán nhãn
- Nhãn có cú pháp: event-signature '[' guard-condition ']' '/' action-expression

Lược đồ trạng thái – Ví dụ

- lược đồ trạng thái của lớp Message



Lược đồ trạng thái – ví dụ 2

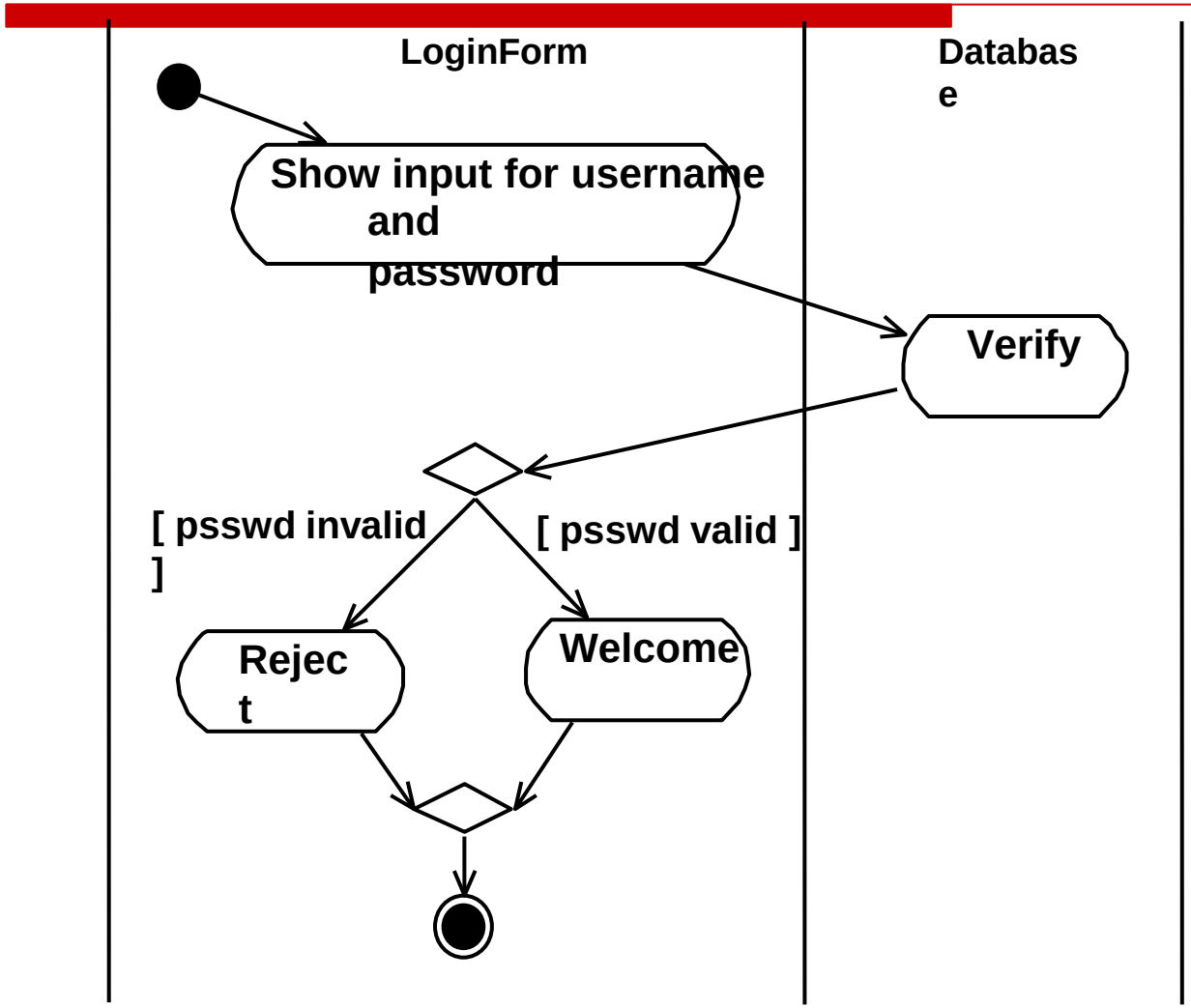


Lược đồ hoạt động

- Lược đồ hoạt động (activity diagram) là một biến thể của lược đồ trạng thái trong đó trạng thái là sự thực thi một hành động và sự dịch chuyển được kích hoạt khi hành động hoàn tất
- Được dùng để mô tả một thủ tục hay thuật giải ▪ tập trung vào các hành động
- Mỗi hành động được ký hiệu bằng hình vẽ như sau

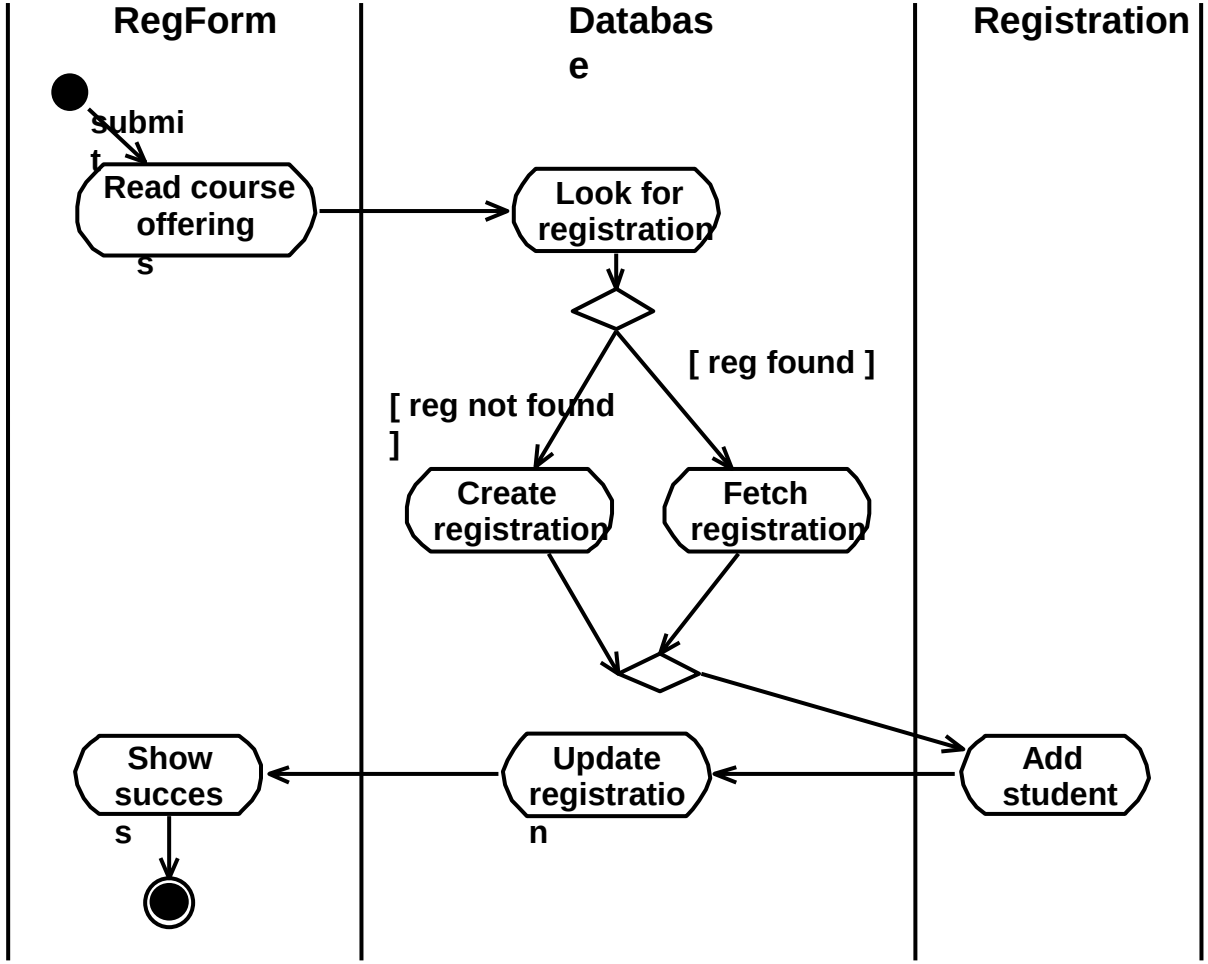
- Quyết định rẽ nhánh: hình thoi có một đường vào và nhiều nhánh ra, mỗi nhánh được gán một guard-condition
- Các nhánh ra được nhập lại bằng một hình thoi khác
- Mỗi "đường bơi" (swimlane) đại diện một lớp hoặc một actor

Lược đồ hoạt động – ví dụ



lược đồ hoạt động cho tác vụ submit của LoginForm

Lược đồ hoạt động – ví dụ 2



Lược đồ hoạt động cho tác vụ submit của RegisterForm

Nhận diện 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, mở cầu nối network, các phần tử giao diện….

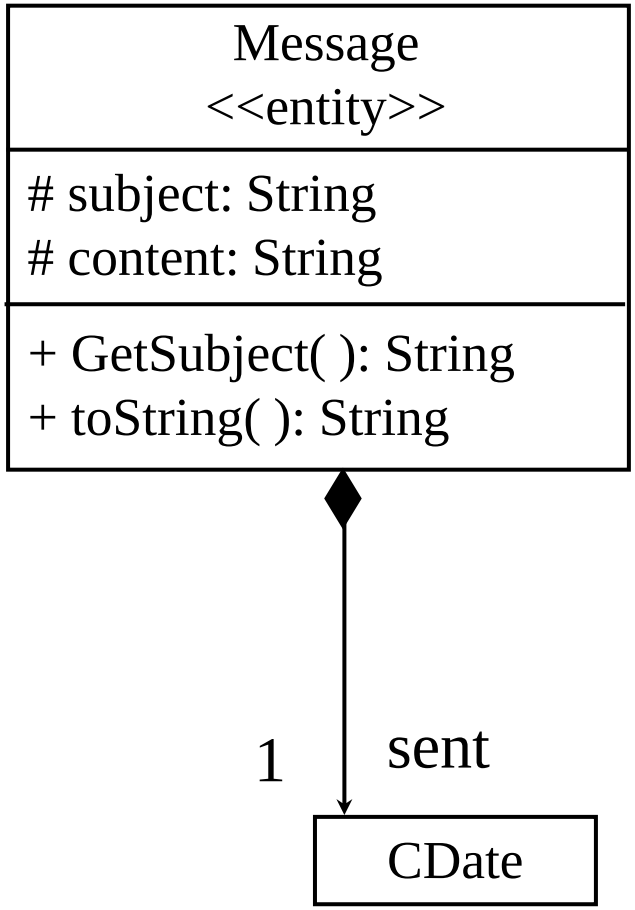
Nhận diện một số lớp thiết kế

- Một số lớp khác xuất hiện làm chức năng duyệt (iterate) một lớp khác hay thực hiện các tính toán phức tạp...
- Sử dụng trực tiếp các lớp do thư viện hay ngôn ngữ cung cấp, hoặc
- Tạo ra lớp mới bằng cách thừa kế hay tích hợp các lớp có sẵn,

  ví dụ CArray<Isoquant, Isoquant&>
- Bổ sung các lớp mới vào lược đồ lớp đồng thời cập nhật các mối quan hệ mới (bao gộp, phụ thuộc)

Đặc tả chi tiết các thuộc tính

- Trong mô hình phân tích cần phải chỉ rõ kiểu (hoặc cấu trúc dữ liệu) và mức độ truy xuất của các thuộc tính
- Có thể chọn một lớp cung cấp bởi thư viện lập trình để cụ thể hoá kiểu hay cấu trúc dữ liệu
  - bổ sung lớp của thư viện và quan hệ bao gộp vào lược đồ lớp



Nhận diện chính xác các tác vụ

- Các lược đồ mô tả hành vi (cộng tác, tuần tự, trạng thái, hành động) giúp nhận diện chính xác các tác vụ của các lớp
- Dựa vào các thông điệp hay hành động để xác định signature của các tác vụ

- Ví dụ: nhận diện một tác vụ của lớp Database

fetchStudent(code: Long): StudentInfo;

setReg(reg: Registration);

Nhận diện chính xác các tác vụ (tt)

- Ví dụ: nhận diện một tác vụ của lớp ChildView

  render( );

  store( );

  load( );

  model(map: FieldMap, param);

- Ví dụ: nhận diện một tác vụ của lớp LoginForm

  submit(uname: String; psswd: String);

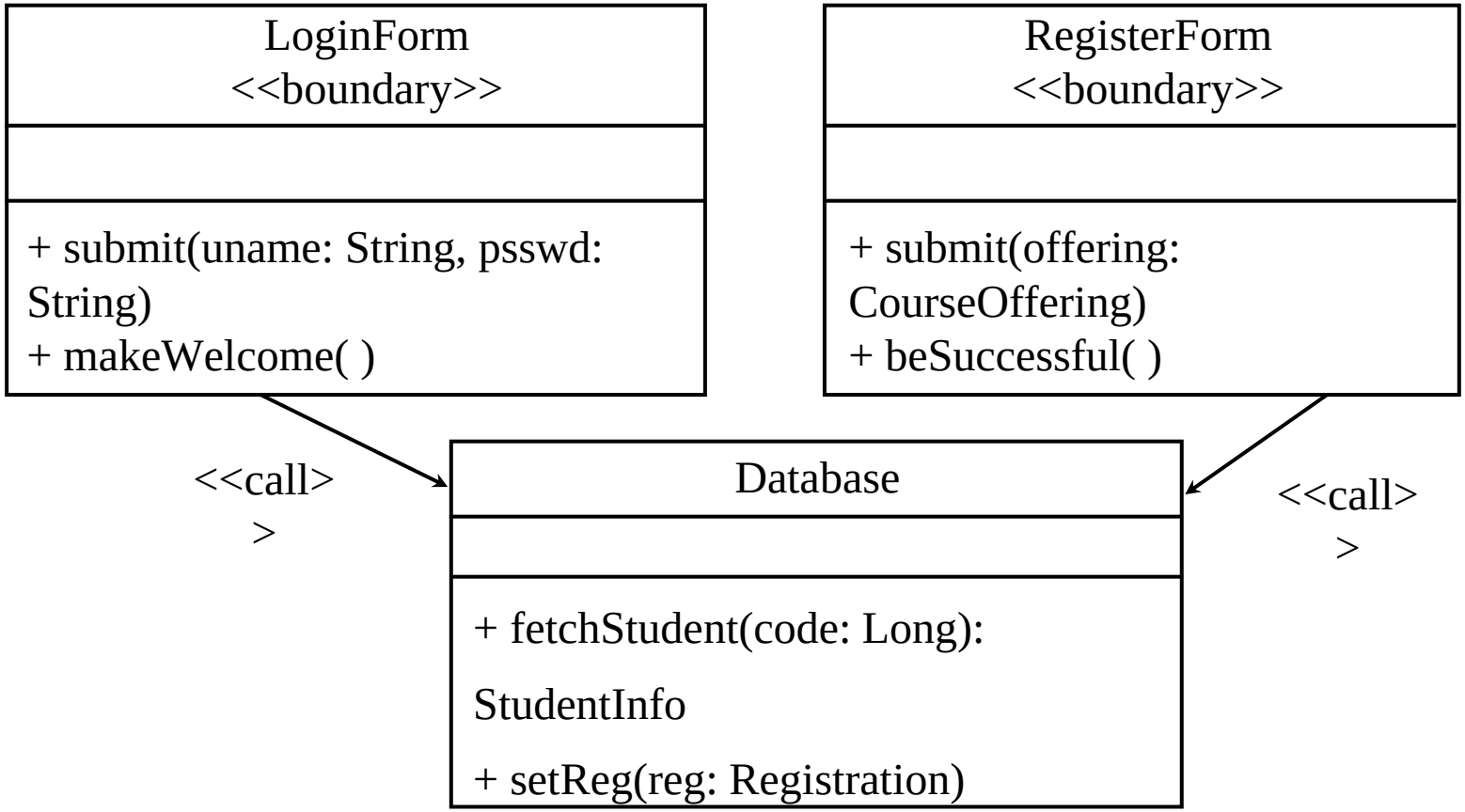
  makeWelcome( );

Hoàn chỉnh lược đồ lớp

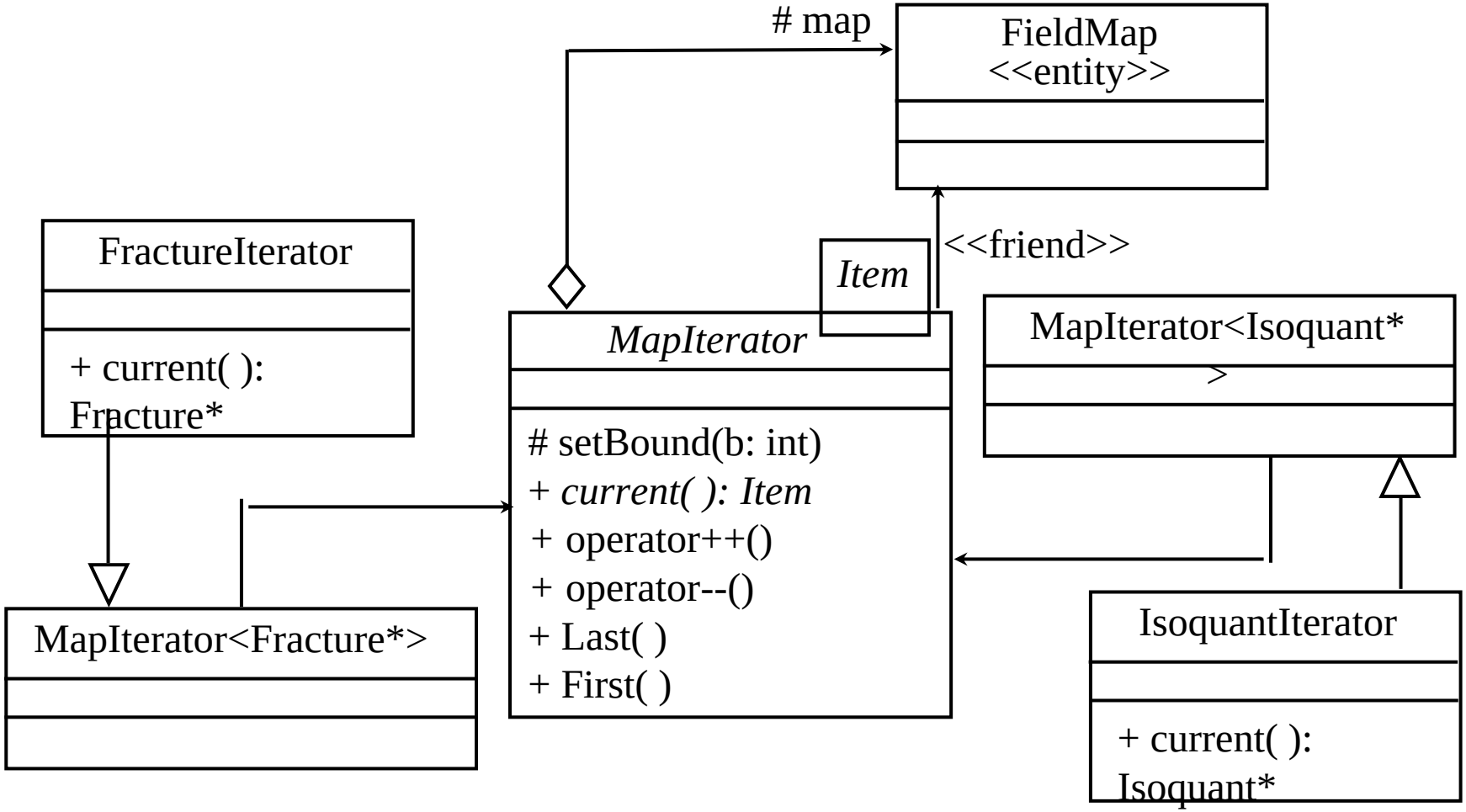
- Cập nhật các lớp mới, thuộc tính, tác vụ và các mối quan hệ mới
- UML định nghĩa quan hệ phụ thuộc (dependency) giữa 2 lớp hoặc package: thay đổi ở một lớp, package kéo theo thay đổi ở lớp, package kia
- Ký hiệu của quan hệ phụ thuộc là mũi tên đứt nét: lớp, package ở phía đuôi mũi tên phụ thuộc vào lớp, package phía đầu mũi tên
- Một số stereotype quy ước trước: <<call>>, <<instantiate>>, <<import>>, <<refine>>, <<realize>>, <<derive>>, <<trace>>

Hoàn chỉnh lược đồ lớp – Ví dụ

- thêm lược đồ lớp cho hệ thống đăng ký môn học



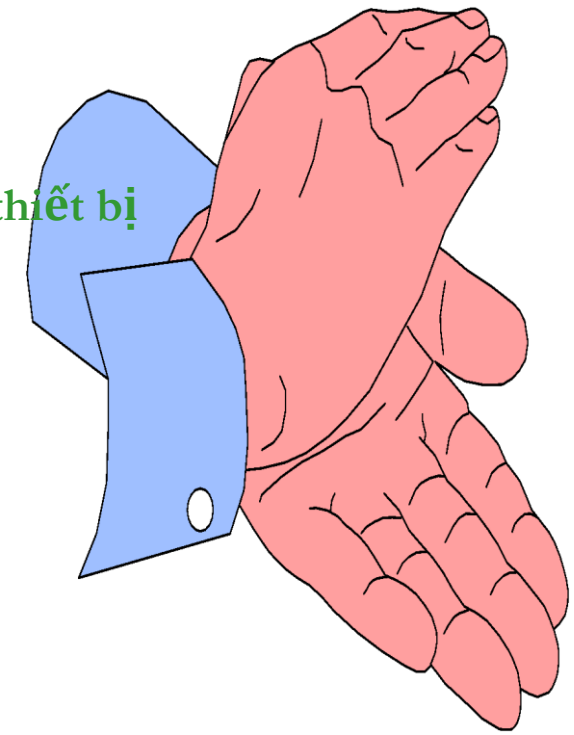
Hoàn chỉnh lược đồ lớp – ví dụ 2



Hoàn chỉnh lược đồ lớp - package

- Chú ý sử dụng package để tổ chức các phần tử liên quan với nhau: các lớp về bản đồ địa 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 chịu ảnh hưởng từ framework (Document/View, 3-tiers...)
- Mỗi package chứa một hoặc một vài lược đồ lớp, trong đó có thể tham chiếu đến một số lớp thuộc các package khác

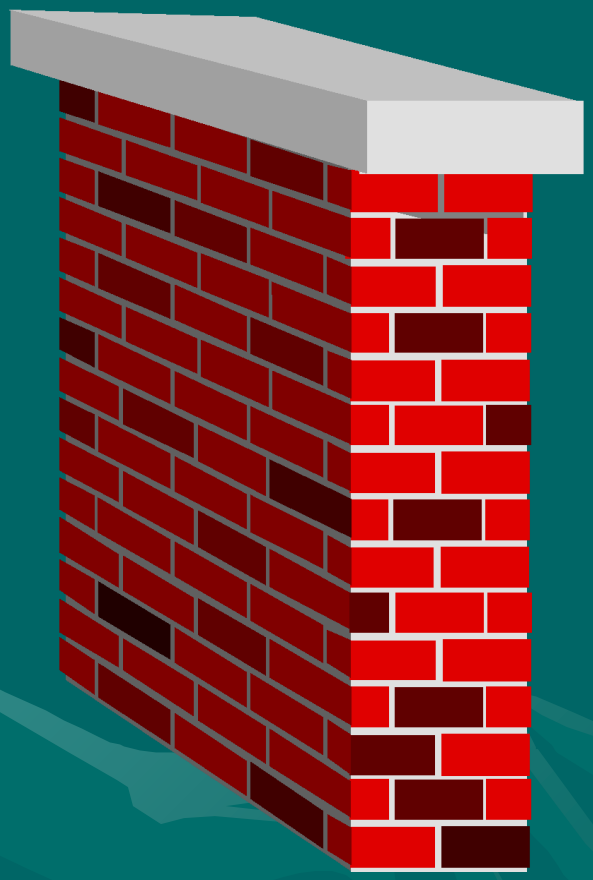
- Mô hình thiết kế bao trùm cả khía cạnh tĩnh và động của hệ thống phần mềm cần xây dựng
- UML hỗ trợ một số lược đồ giúp mô tả khía cạnh động: cộng tác, tuần tự, trạng thái, hành động
- Miêu tả chính xác thuộc tính và tác vụ, bổ sung một số lớp thiết kế □ hoàn thiện khía cạnh tĩnh
- Thiết lập các package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



HIỆN THỰC VÀ TRIỂN KHAI

- Các thành phần
- Các thiết bị

Giới thiệu



- Cần phải xây dựng chương trình chạy được từ kết qủa của giai đoạn thiết kế
- Các lớp sẽ được cụ thể hoá vào các thành phần phần mềm như thế nào và bằng ngôn ngữ lập trình gì ?
- Chương trình sẽ được cài đặt ra sao trên tài nguyên tính toán ?

Thành phần (Component)

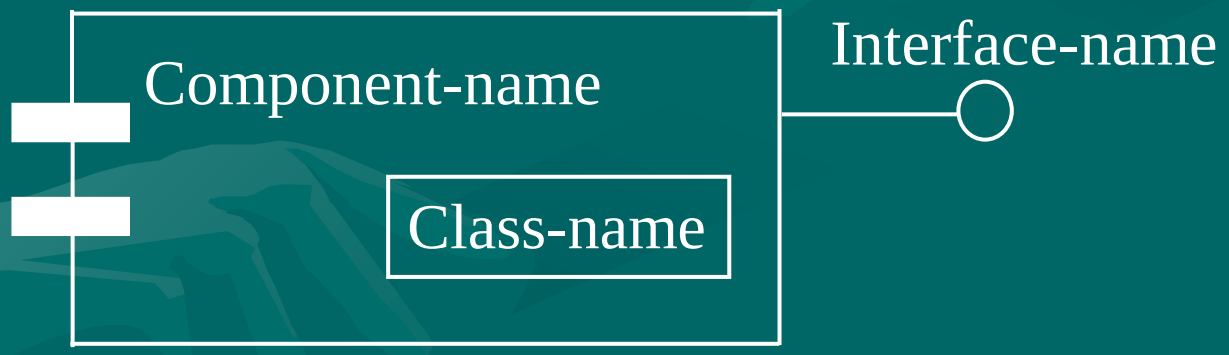
- Thành phần (*component*) biểu diễn một phần hiện thực nào đó của hệ thống
- Một số *stereotype* quy ước trước:
  - **<<file>>: mã nguồn hay dữ liệu**
  - **<<executable>>: chương trình chạy được**
  - **<<library>>: thư viện liên kết tĩnh hay động**
  - **<<document>>: tài liệu được thiết lập trong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

Thành phần (Component)

- Thành phần phần mềm (*software component*) bao gồm
  - **Mã nguồn: *.cpp, *.c, *.pas, *.java, *.bas**
  - **Mã đối tượng: *.obj**
  - **Mã nhị phân: *.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

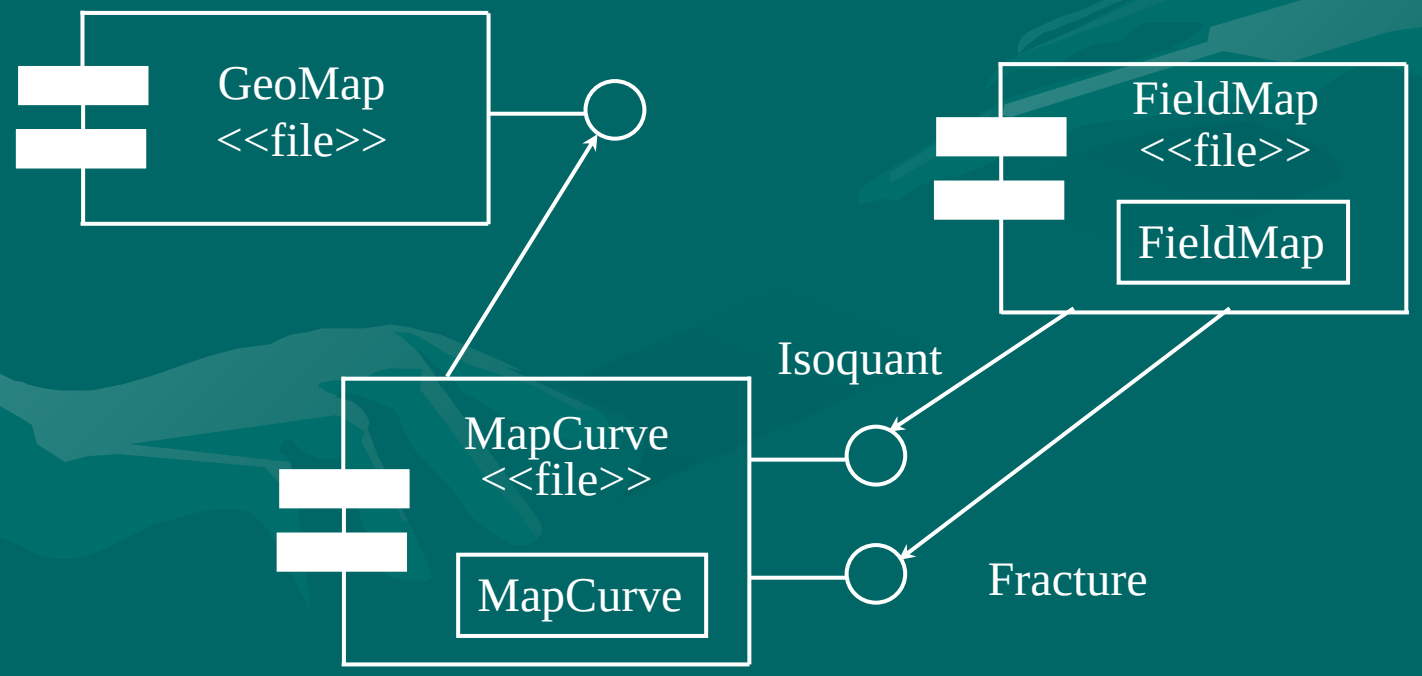
Lược đồ thành phần

- Lược đồ thành phần là một đồ thị 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 tròn biểu diễn các giao tiếp và chứa các lớp mà nó cụ thể hoá



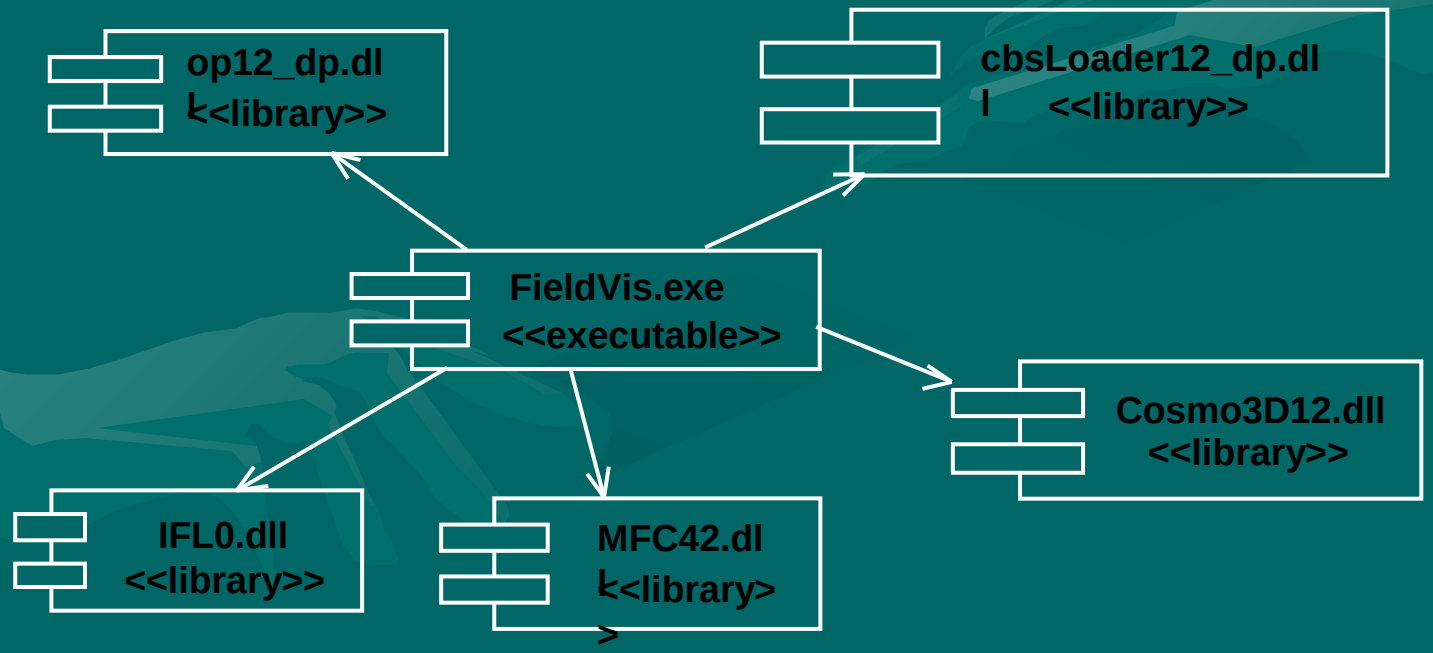
Lược đồ thành phần – Ví dụ

- Ví dụ: lược đồ thành phần thể hiện một số *module* mã nguồn của chương trình hiển thị bề mặt địa hình



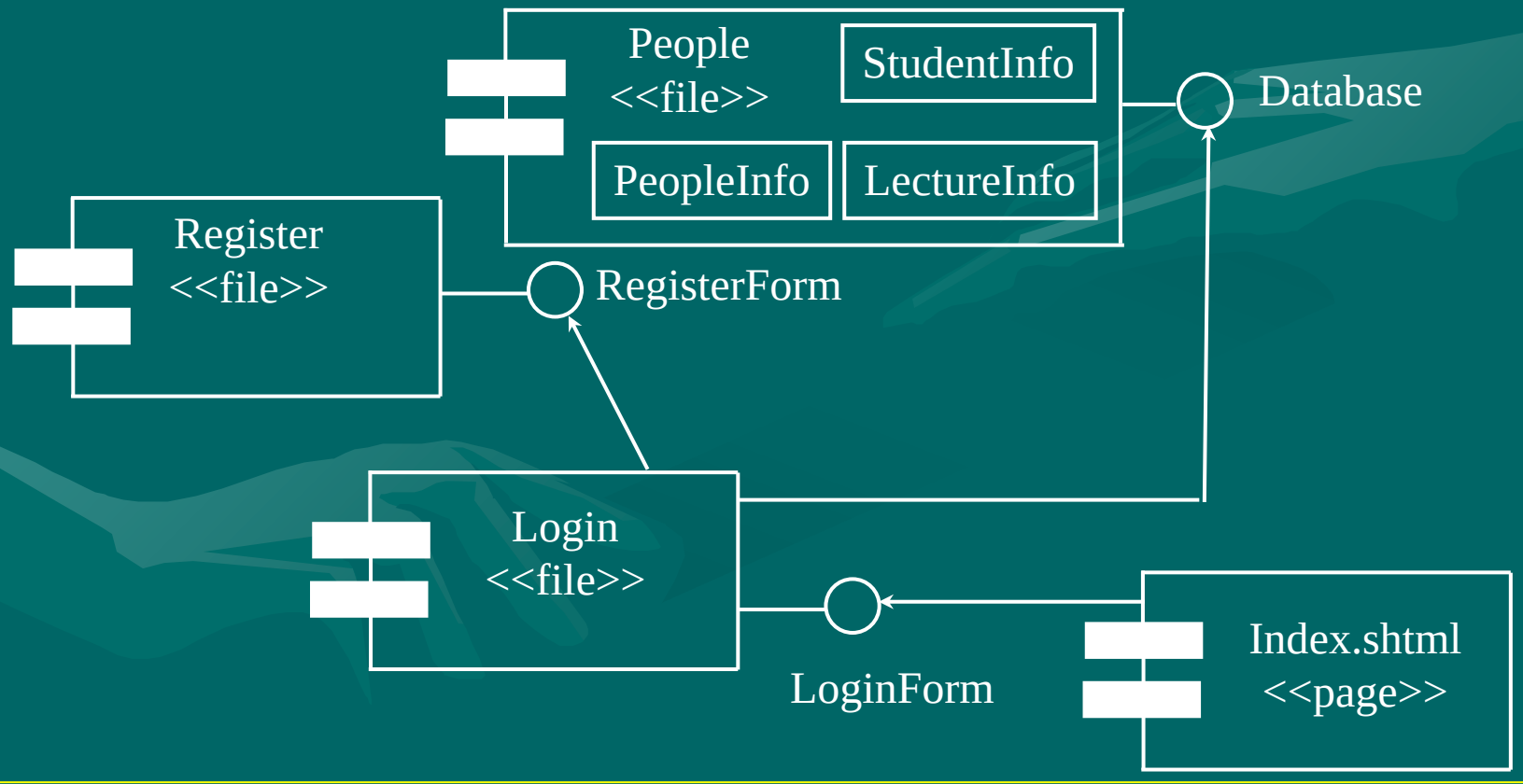
Lược đồ thành phần – Ví dụ

- Ví dụ: lược đồ thành phần thể hiện thời gian thực thi của chương trình hiển thị bề mặt địa hình



Lược đồ thành phần – Ví dụ

- Ví dụ: lược đồ thành phần của hệ thống đăng ký môn học

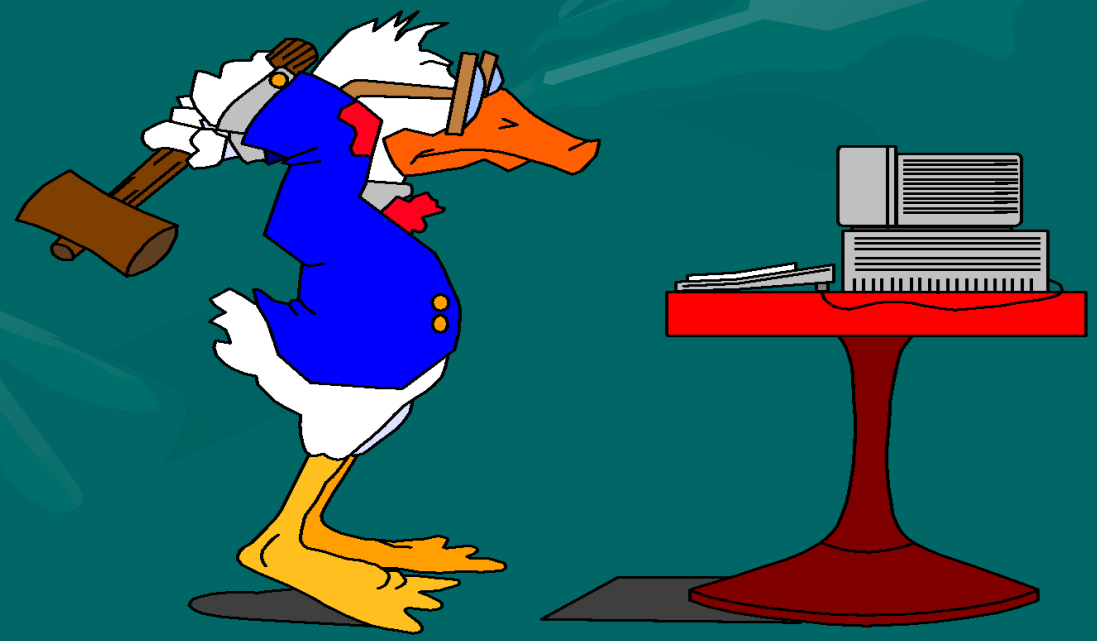


Gán các lớp vào các thành phần

- Khi thiết lập các thành phần mã nguồn, chú ý gán (*bind*) các lớp thiết kế và chọn ngôn ngữ lập trình
  - **Gán lớp FieldMap vào thành phần FieldMap (C++)**
  - **Gán lớp MapCurve, Isoquant và Fracture vào thành phần MapCurve**
  - **Gán lớp PeopleInfo, StudentInfo, LectureInfo và Database vào thành phần People (Java)**
  - **Gán lớp và LoginForm vào thành phần Login (Java)**
- Ký hiệu của thành phần chứa ký hiệu của lớp được gán
- Chú ý: *component* ≠ *package*

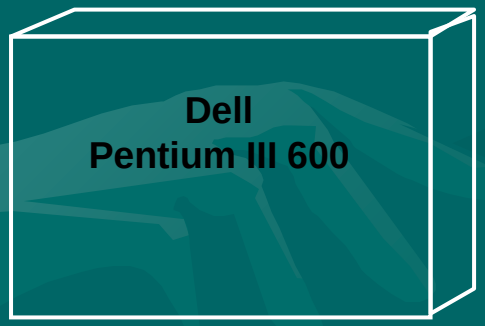
Sinh mã nguồn – Sourse code generation

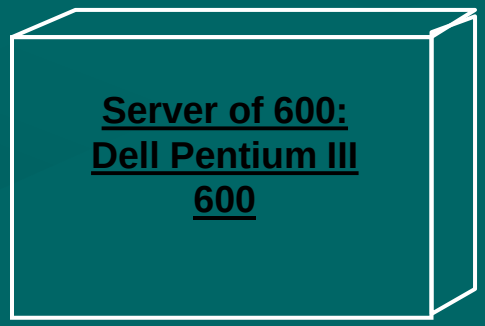
- Dựa vào đặc tả lớp để viết mã cho từng thành phần mã nguồn theo ngôn ngữ lập trình đã chọn
- Viết mã sườn là công việc hơi nhàm chán ⇒ có thể được tự động hoá bởi các công cụ CASE



Node triển khai

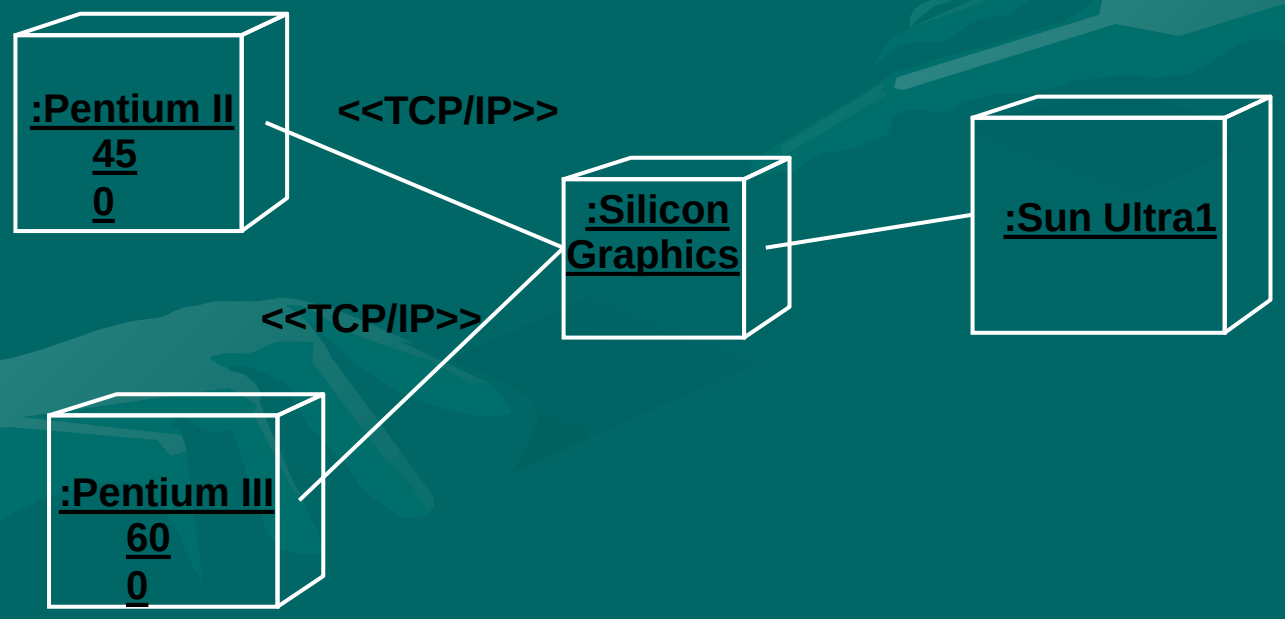
- Node là một thiết bị vật lý có khả năng tính toán, bao gồm: máy tính, máy in, thiết bị quét *card*, *router…*
- Node được mô tả ở cả 2 dạng: dạng lớp và dạng *instance*
- Node được ký hiệu như hình hộp ba chiều
- Các minh dụ của thành phần có thể sống trong một minh dụ node





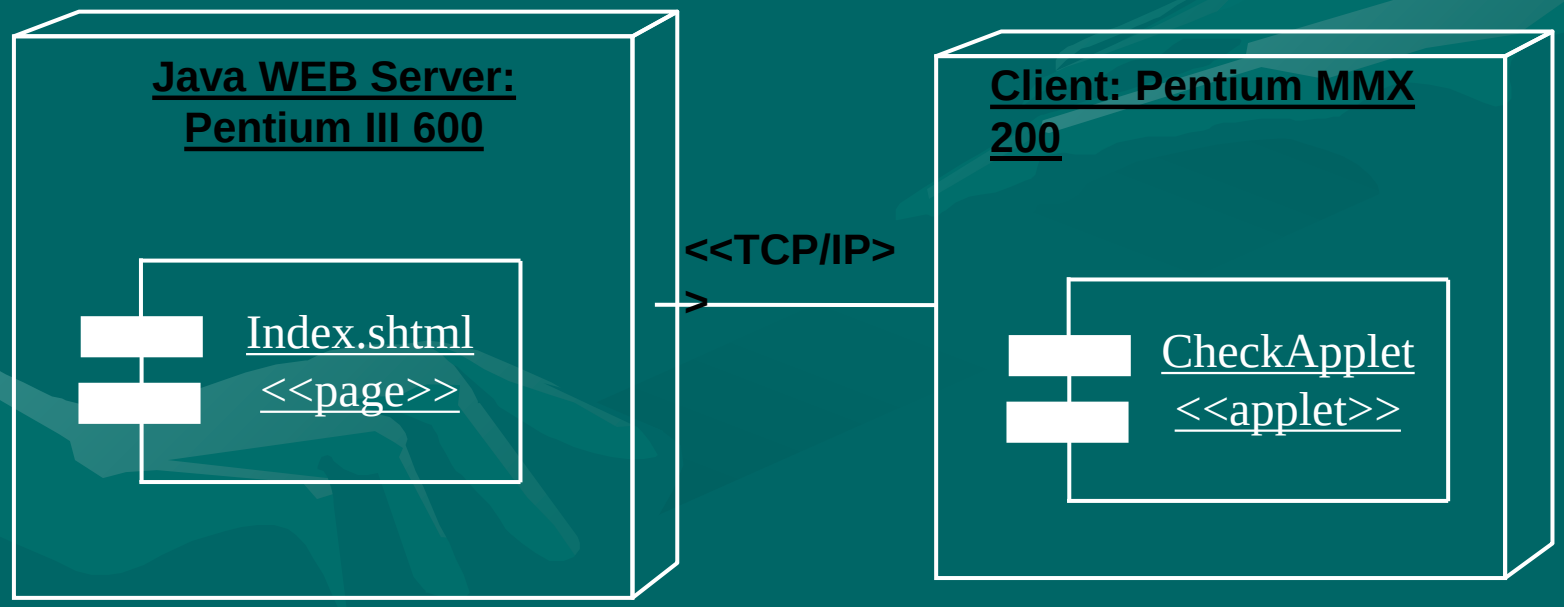
Kết nối giữa các node

- Có thể chỉ ra quan hệ liên kết giữa các *node* để mô tả cấu hình kết nối (*connection*)



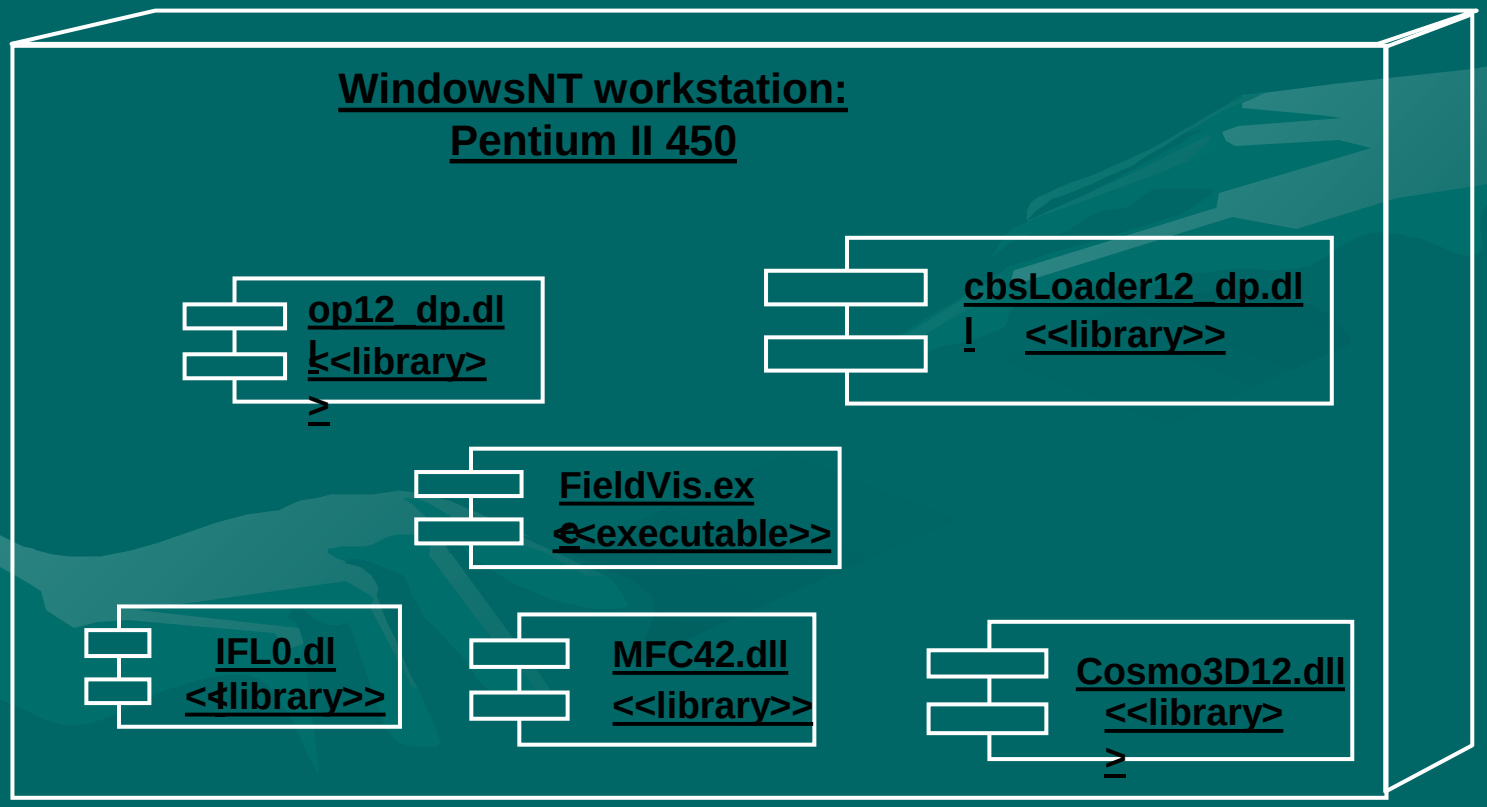
Lược đồ triển khai

- Lược đồ triển khai cho phép miêu tả cách cài đặt các thành phần thực thi trên các *node*
- Ví dụ: hệ thống đăng ký môn học qua WEB

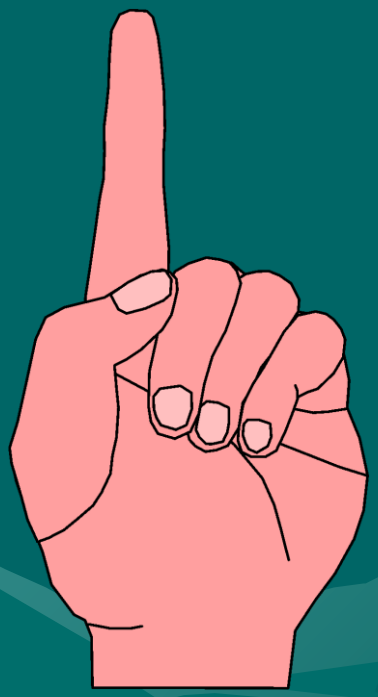


Lược đồ triển khai

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



Tổng kết



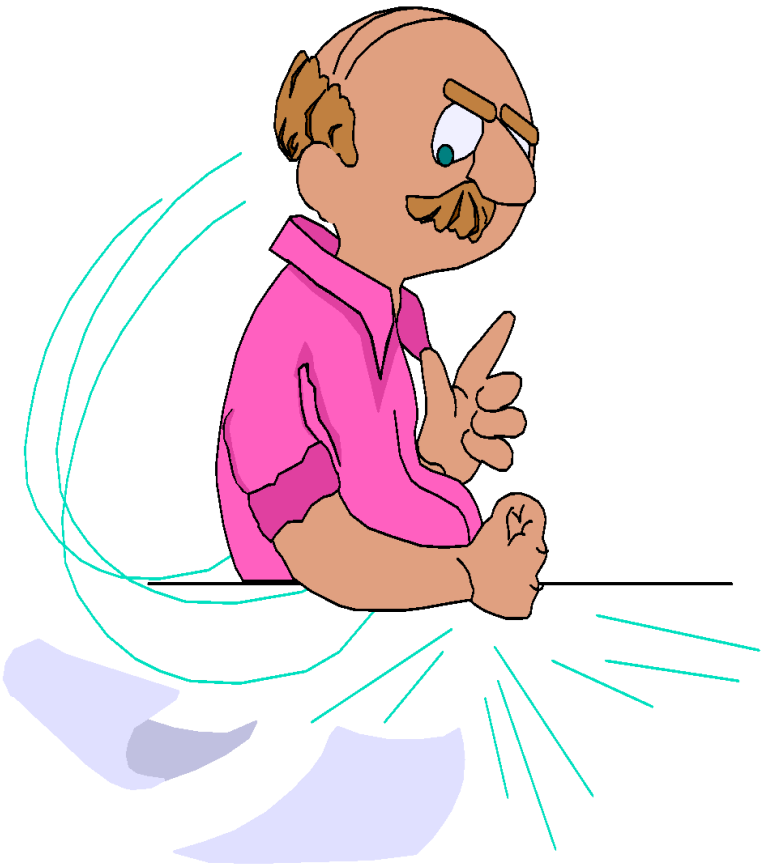
- Hiện thực và triển khai tập trung vào xây dựng các thành phần chạy được hoặc các thư viện, *module* mã nguồn, trang HTML, dạng nhị phân...
- Các thành phần mã nguồn cụ thể hoá một số lớp thiết kế và có thể được viết bằng các ngôn ngữ lập trình khác nhau
- Cuối cùng triển khai các thành phần chạy được trên các thiết bị tính toán

Kiểm nghiệm phần mềm

Kỹ thuật kiểm tra phần mềm
Kiểm tra các đường thực thi độc lập
Chiến thuật kiểm tra phần mềm
Alpha, Beta testing

Tại sao phải kiểm tra phần mềm

- Mặc dù được tự động hoá một phần bởi các công cụ CASE, rất nhiều công đoạn trong quá trình sản xuất phần mềm vẫn được thực hiện bởi con người
- vẫn có khả năng xảy ra lỗi.
- Lỗi có thể xảy ra trong tất cả các giai đoạn: phân tích yêu cầu, thiết kế, mã hoá
- Do đó phải kiểm nghiệm chương trình trước khi chính thức sử dụng



Một số quan điểm – khái niệm

- **Kiểm nghiệm phần mềm** là hoạt động thực thi chương trình với mục đích tìm ra lỗi □ phê phán, không phải để mang tính xây dựng
- Phân loại:
  - Kiểm nghiệm **black-box**: kiểm tra các chức năng cụ thể của phần mềm, không quan tâm cấu trúc bên trong, thường áp dụng cho những module lớn.
  - Kiểm nghiệm **white-box**: kiểm tra cấu trúc điều khiển bên trong chương trình, thường dùng cho những những module nhỏ.
- Mỗi loại kiểm nghiệm có khả năng tìm ra những nhóm lỗi khác nhau □ nên kết hợp cả hai

Mục tiêu của kiệm 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 được lỗi trong chương trình (nếu có).
  - Chứng minh được phần mềm hoạt động đúng như đã thiết kế.
- Chứng minh phần mềm được viết đúng
  - Chứng minh được phần mềm đáp ứng yêu cầu của user
- Góp phần chứng minh chất lượng của phần mềm.

Mục tiêu của kiệm nghiệm phần mềm

- Quá trình kiểm nghiệm phần mềm là tốt khi
  - Có khả năng tìm ra lỗi cao.
  - Không dư thừa.
  - Biết chọn lọc: chỉ kiểm nghiệm những phần nào có khả năng tìm ra lỗi đặc trưng.
  - Không quá phức tạp cũng không quá đơn giản.

- Chú ý: Kiểm nghiệm phần mềm không khẳng định được phần mềm không còn khiếm khuyết, chỉ khẳng định được phần mềm có lỗi và giúp giảm thiểu lỗi

Các nguyên lý kiểm nghiệm phần mềm

- Việc kiểm nghiệm nên hướng về yêu cầu của khách hàng
- Nên được hoạch định trước một thời gian dài.
- Áp dụng nguyên lý Pareto (nguyên tắc 80-20):

80% lỗi có nguyên nhân từ 20% các module ☐ cô lập và kiểm tra những module khả nghi nhất.

- Nên tiến hành từ nhỏ đến lớn: 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 được thực hiện bởi những đối tượng KHÔNG tham gia vào quá trình phát triển phần mềm.

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Phương pháp kiểm nghiệm – Test case

- Thiết lập các test case – vận hành thử - so sánh kết quả
- Khái niệm test-case
  - Dữ liệu input
  - Thao tác kiểm nghiệm
  - Dữ liệu output hay đáp ứng mong đợi của chương trình
- Test-case cho kiểm nghiệm black-box: chủ yếu dựa vào các yêu cầu cụ thể của chức năng phần mềm.
- Test-case cho kiểm nghiệm white-box: chủ yếu dựa vào cấu trúc điều khiển của phần mềm ☐ vấn đề đặt ra: số lượng test-case cần thiết là quá lớn

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Kiểm nghiệm các đường độc lập cơ bản

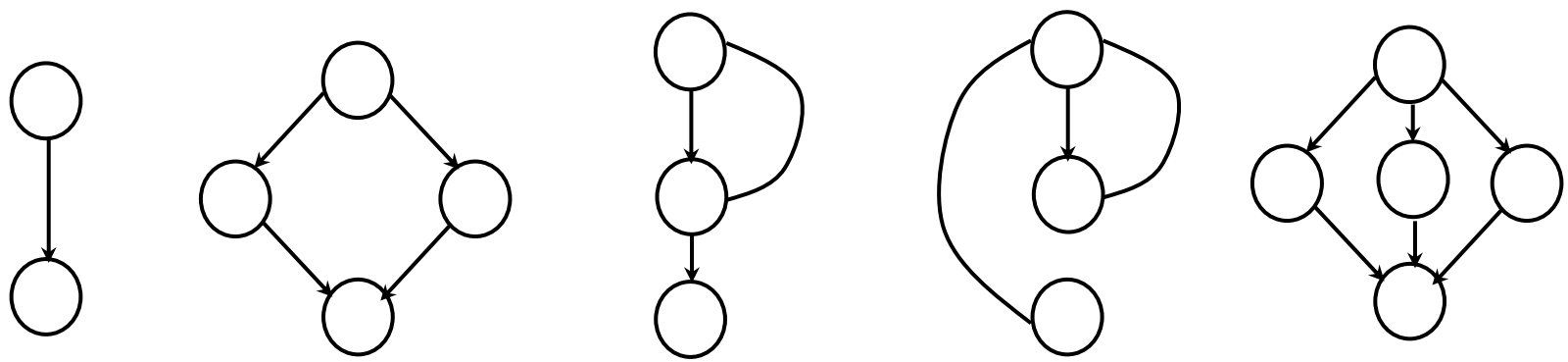
- Kiểm nghiệm white-box dựa vào cấu trúc điều khiển của thiết kế thủ tục để sinh các test-case với tiêu chí
  - Kiểm nghiệm các đường độc lập cơ bản là một trong những phương cách kiểm nghiệm white-box
  - Bảo đảm số phép thử là ít nhất đủ để phát hiện các lỗi
  - Tất cả các đường thực thi độc lập được thử qua ít nhất một lần
  - Thử các điều kiện rẽ nhánh ở cả 2 nhánh true và false
  - Thử qua vòng lặp tại biên cũng như bên trong
  - Thử qua cấu trúc dữ liệu để đảm bảo tính toàn vẹn của nó

Đồ thị dòng chảy

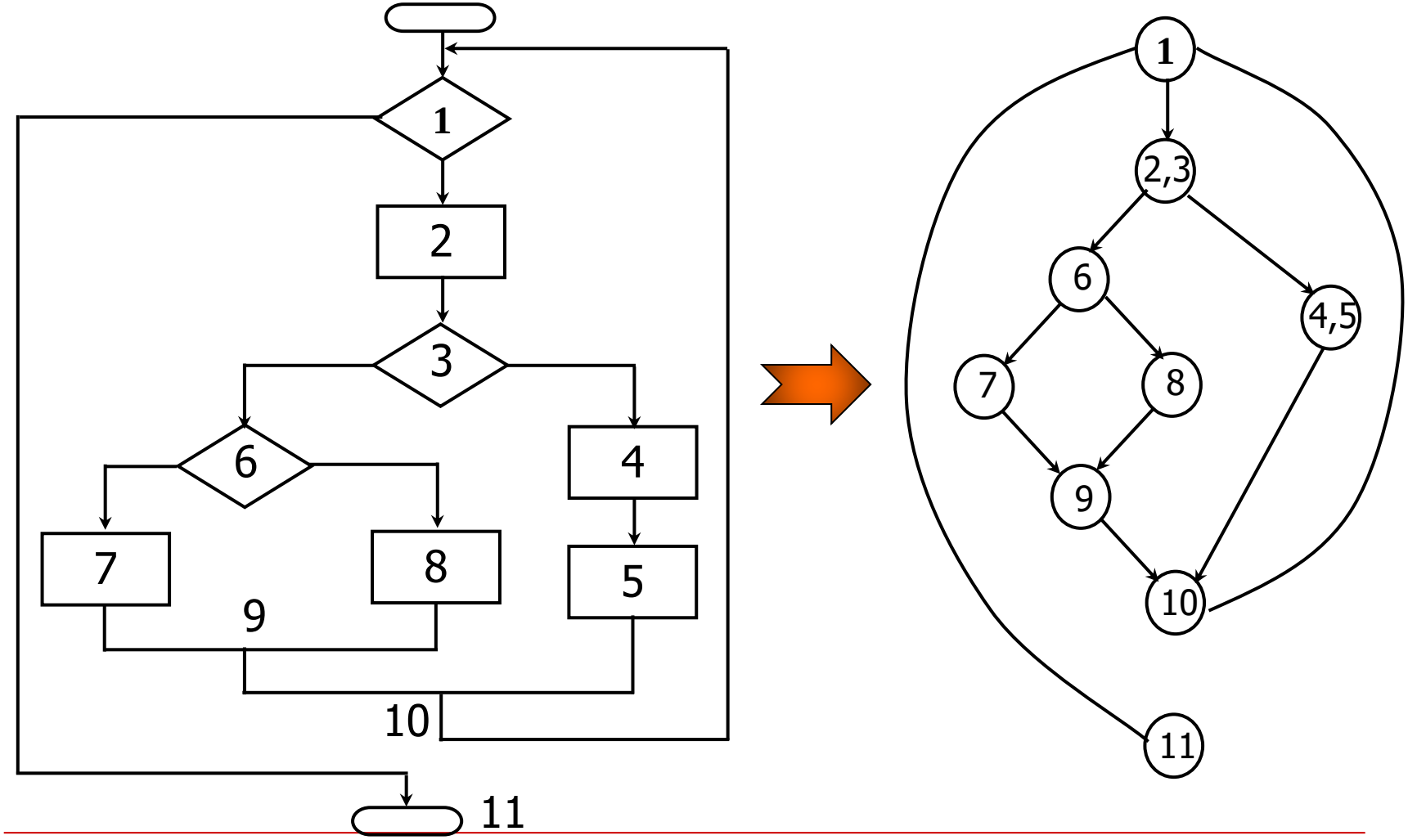
- Mỗi node hình tròn biểu diễn một hoặc một vài tác vụ (hơi khác so với lưu đồ thuật giải)
- Cạnh có hướng miêu tả đường thực thi

- Đồ thị dòng chảy được xây dựng từ lưu đồ thuật giải

sequence if until while case

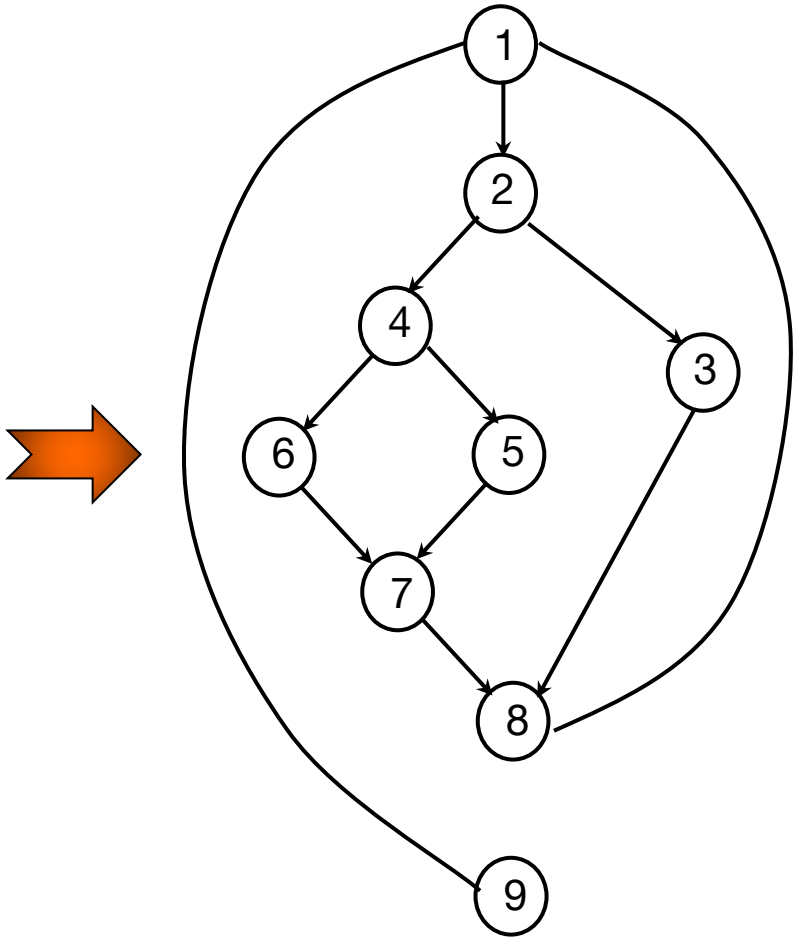


Xây dựng đồ thị dòng chảy – Ví dụ



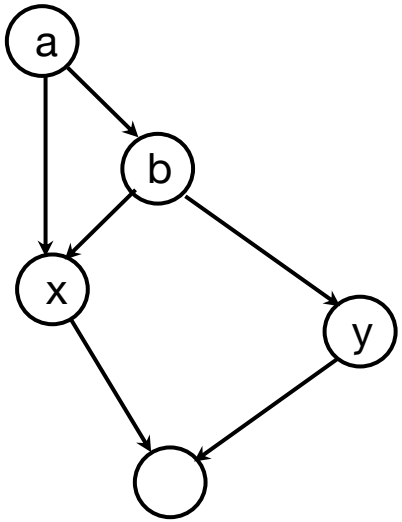
Xây dựng đồ thị dòng chảy – Ví dụ

```
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;
         endif;
8:    enddo
9: end
```

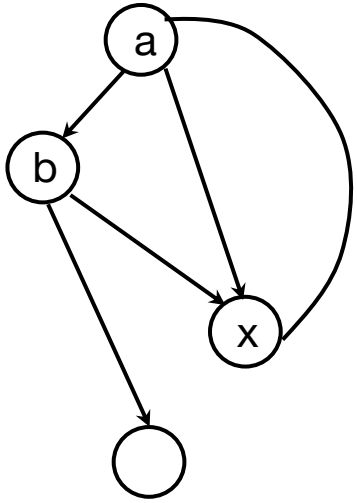


Xây dựng đồ thị dòng chảy

- Phải phân rã tất cả các điều kiện phức trở thành các điều kiện đơn
- Mỗi node mô tả một điều kiện đơn được gọi là predicate



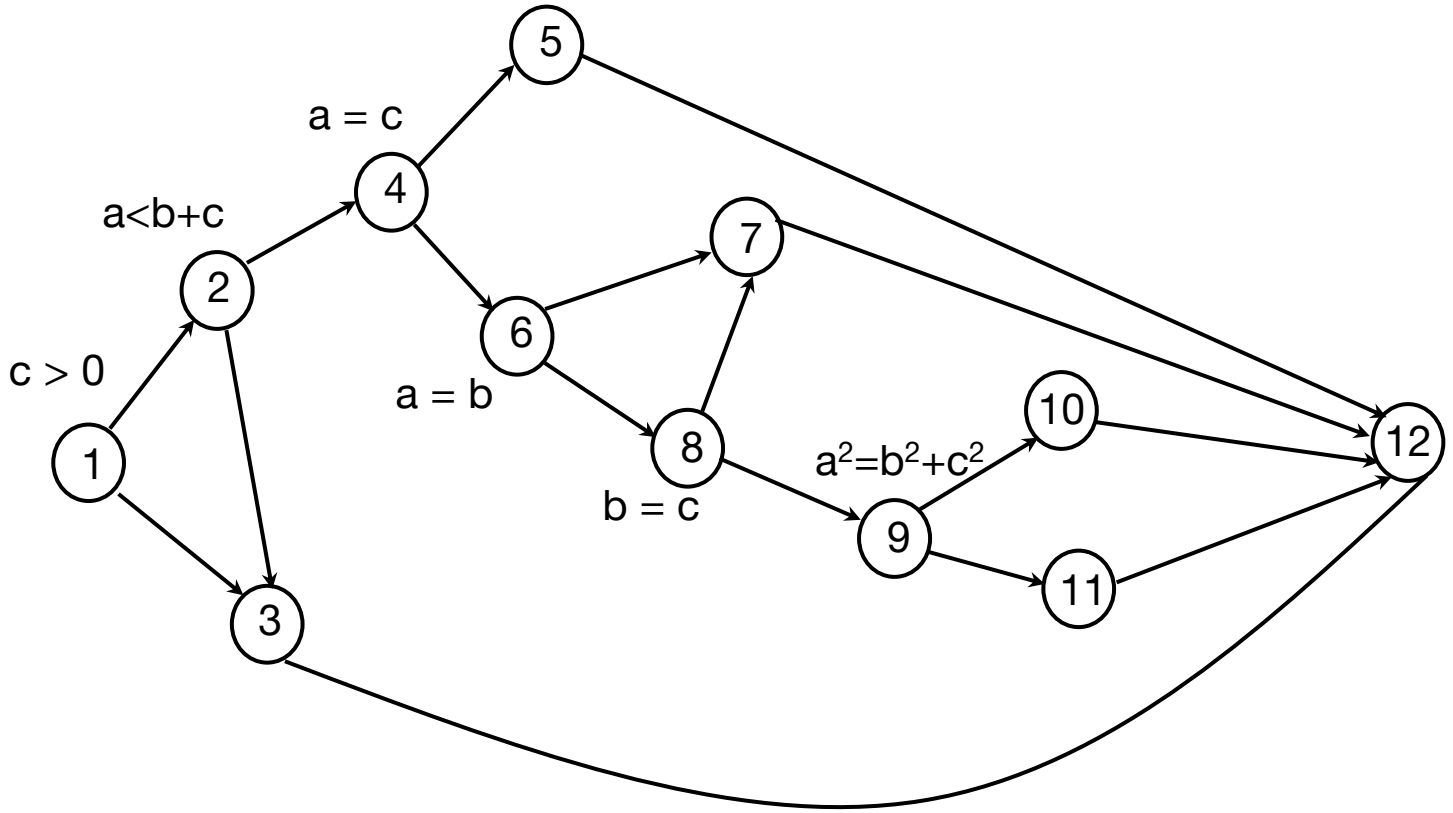
if a **and** b **then** y **else** x



while a **or** b **do** x

Xây dựng đồ thị dòng chảy – ví dụ

- procedure AnalyzeTriangle



Các đường độc lập cơ bản

- Đường thực thi?
- Đường thực thi cơ bản
- Các đường thực thi độc lập cơ bản

  Từ node bắt đầu đến node kết thúc, các đường thực thi cơ bản được liệt kê theo một thứ tự nào đó để đảm bảo rằng: đường đang liệt kê ít nhất đi qua một cạnh chưa được duyệt qua bởi các đường đã liệt kê trước đó
- Tổng số đường thực thi cơ bản độc lập nhau được tính bằng

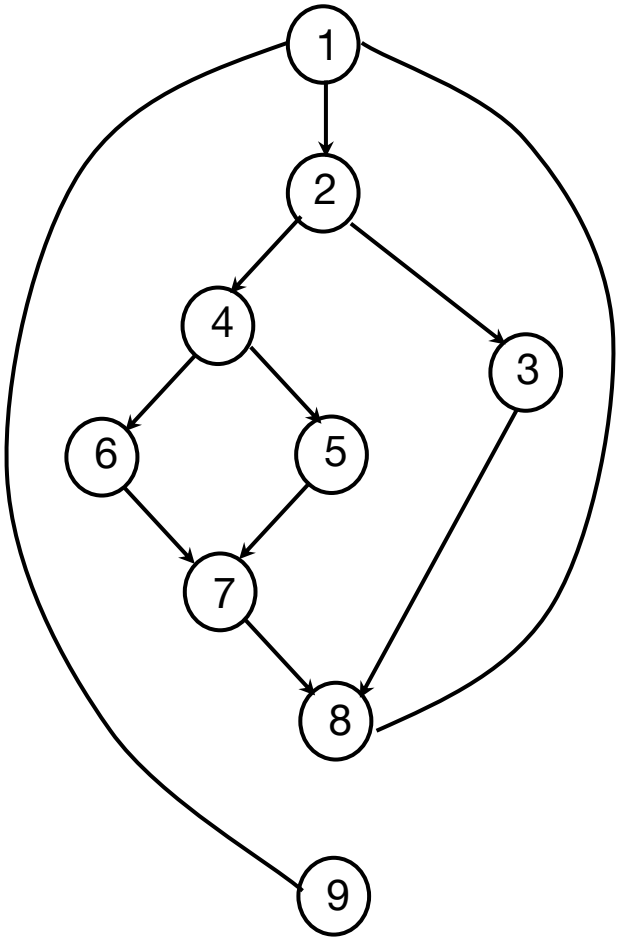
  V = P + 1; trong đó **P** là số node phân nhánh (predicate)

Các đường độc lập cơ bản – ví dụ

- Đối với chương trình con DoSomething
- Tổng số đường :

  V = 3 + 1 = 4
- Đường 1: 1-9
- Đường 2: 1-2-3-8-1…
- Đường 3: 1-2-4-5-7-8-1…
- Đường 4: 1-2-4-6-7-8-1…

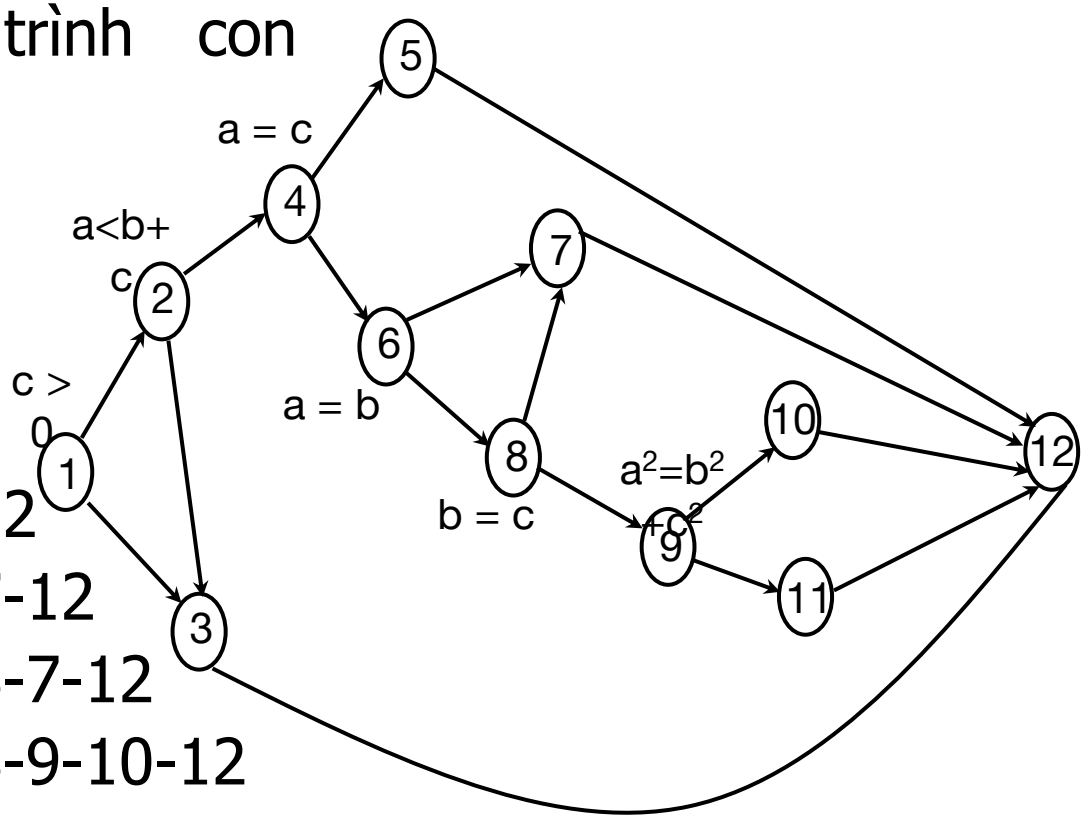
Chú ý: dấu 3 chấm (…) mang ý nghĩa "không quan tâm", từ đó có thể đi theo bất kỳ cạnh nào bởi vì các cạnh sau đó đã được duyệt qua rồi



Các đường độc lập cơ bản – ví dụ

- Đối với chương trình con AnalyzeTriangle
- Tổng số đường :

  V = 6 + 1 = 7
- Đường 1: 1-3-12
- Đường 2: 1-2-3-12
- Đường 3: 1-2-4-5-12
- Đường 4: 1-2-4-6-7-12
- Đường 5: 1-2-4-6-8-7-12
- Đường 6: 1-2-4-6-8-9-10-12
- Đường 7: 1-2-4-6-8-9-11-12



Thiết lập các test case

- Thiết lập một test-case cho mỗi đường thực thi cơ bản
- 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 được test-case cho một đường thực thi nào đó
- Ví dụ Sinh test-case cho chương trình con AnalyzeTriangle
- Test-case cho đường 1:

  Input: a = 3, b = 2, c = 0

  Output mong đợi: type = "Error"
- Test-case cho đường 2:

  Input: a = 17, b = 5, c = 4

  Output mong đợi: type = "Error"
- Test-case cho đường 3:

  Input: a = 6, b = 6, c = 6

  Output mong đợi: type = "Equilateral"

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 đường độc lập cơ bản dùng trong kiểm nghiệm white-box, bao gồm các bước
  - Thiết lập đồ thị dòng chảy
  - Liệt kê các đường thực thi độc lập cơ bản
  - Sinh các test-case cho các đường thực thi đó
  - Thực hiện các test case và kiểm tra kết quả
- Kiểm nghiệm Black-box thường dùng trong bước kiểm nghiệm tính năng của phần mềm

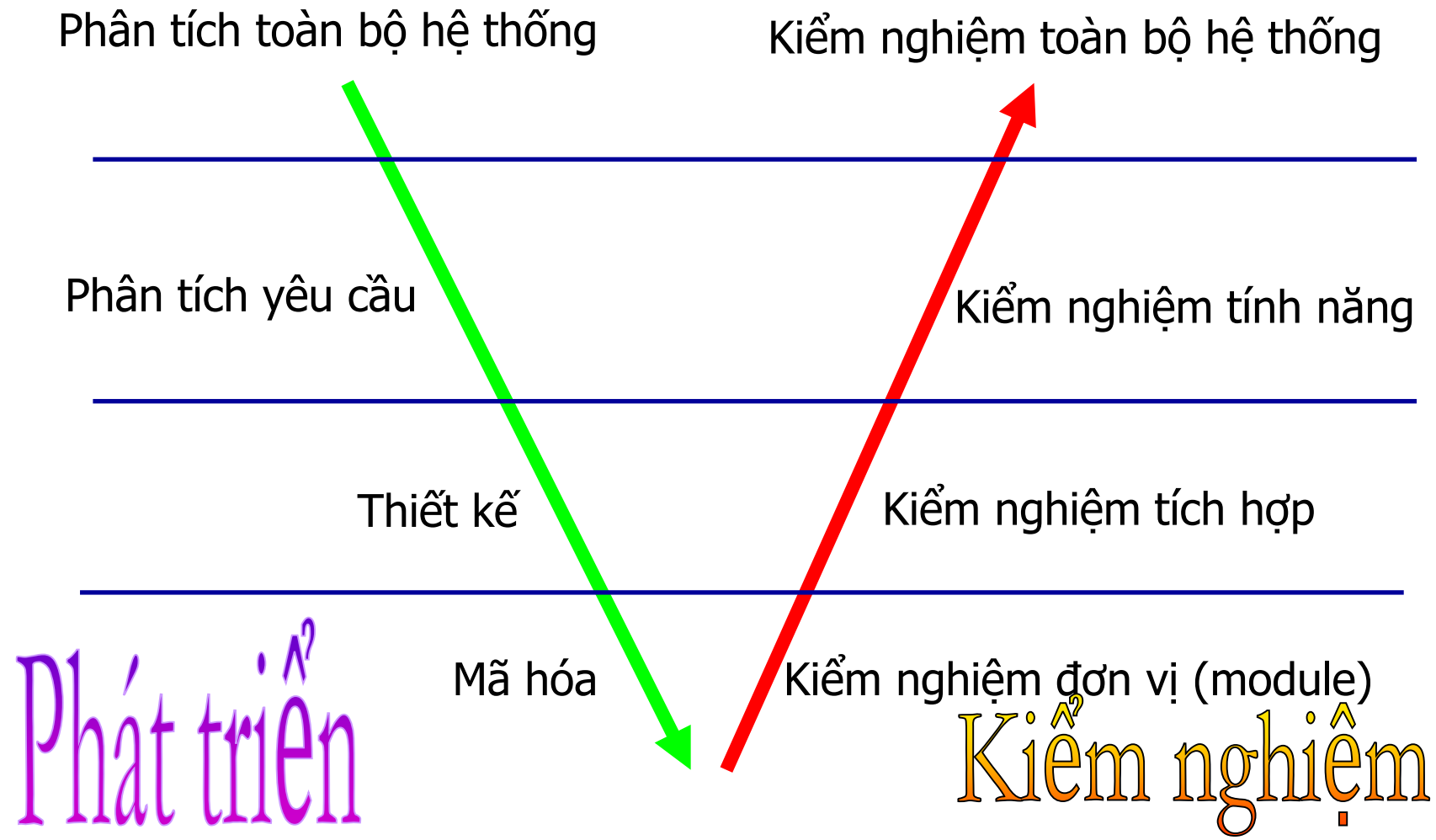
Chiến thuật kiểm nghiệm phần mềm

- Verification & Validation
- Unit test & Integration test
- Kiểm nghiệm hướng đối tượng
- Nghệ thuật gỡ rối

Khái niệm

- **Chiến thuật kiểm tra phần mềm** tích hợp các phương pháp tạo ra test-case trở thành một chuỗi các bước có thứ tự để có thể kiểm nghiệm phần mềm thành công với chi phí thấp.
- Bao gồm các công việc
  - Lập kế hoạch kiểm nghiệm
  - Sinh test-case
  - Thực hiện kiểm nghiệm, thu thập kết qủa và đánh giá
- Verification: các hành động để đảm bảo cho phần mềm được hiện thực đúng theo một chức năng cụ thể nào đó **"Are we building the product right ?"**
- Validation: các hành động để đảm bảo cho phần mềm được xây dựng theo đúng yêu cầu của khách hàng **"Are we building the right product ?"**

Một chiến thuật kiểm nghiệm phổ biến



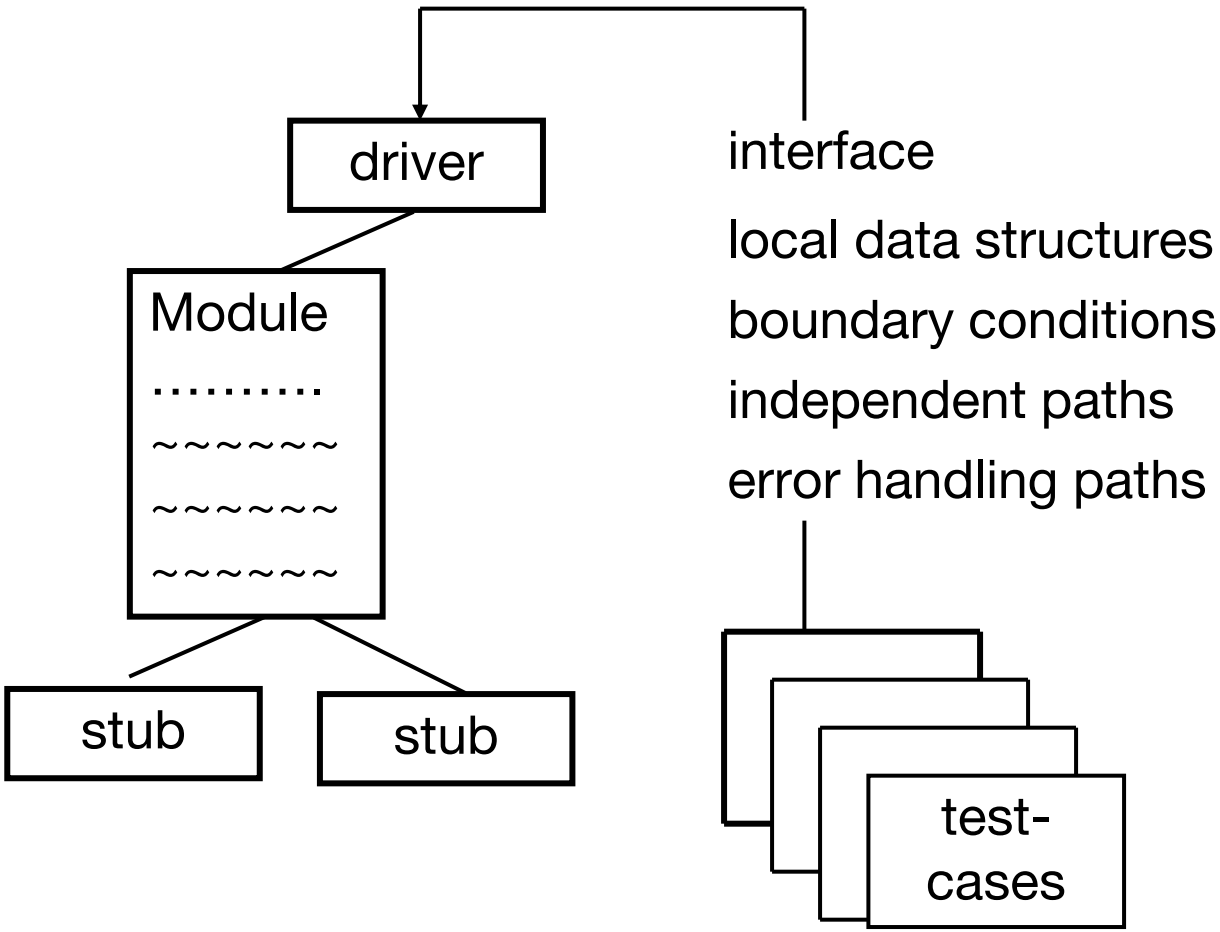
Một chiến thuật kiểm nghiệm phổ biến

- Bắt đầu tại từng module rồi tích hợp lớn dần đến toàn bộ hệ thống.
- Các kỹ thuật khác nhau được sử dụng thích hợp tại các giai đoạn khác nhau.
- Kiểm nghiệm có thể được tiến hành bởi người phát triển phần mềm, nhưng đối với các dự án lớn thì việc kiểm nghiệm phải được tiến hành bởi một nhóm độc lập.
- Kiểm nghiệm và sửa lỗi là các hoạt động độc lập nhưng việc sửa lỗi phải phù hợp với các chiến thuật kiểm nghiệm.

Kiểm nghiệm từng module

- Tiến hành kiểm nghiệm trên từng đơn vị nhỏ nhất của phần mềm, đó là module mã nguồn, sau khi đã thiết kế, mã hoá và biên dịch thành công
- Thường dùng kỹ thuật kiểm nghiệm white-box
- Có thể tiến hành kiểm nghiệm cùng lúc nhiều module.
- Một số vấn đề trong việc xây dựng các test case
  - Test case nào?
  - Dữ liệu đầu vào và đầu ra có tử đâu?
  - Tính độc lập/phụ thuộc hoạt động của các module

Kiểm nghiệm module



Kiểm nghiệm module

- Mỗi module mã nguồn không phải là một chương trình hoàn chỉnh và đôi khi phải gọi các module chưa được kiểm nghiệm khác □ có thể phải thiết lập **driver** và/hoặc **stub**: phí tổn khá lớn (70%)
- **Driver** là một chương trình chính có nhiệm vụ nhận dữ liệu kiểm nghiệm, chuyển dữ liệu đó xuống cho module để kiểm tra và in ra các kết quả kiểm tra tương ứng.
- **Stub** thay thế các module được gọi bởi module đang kiểm tra.

Làm thế nào để giảm các chi phí tạo driver hay stub

Kiểm nghiệm tích hợp

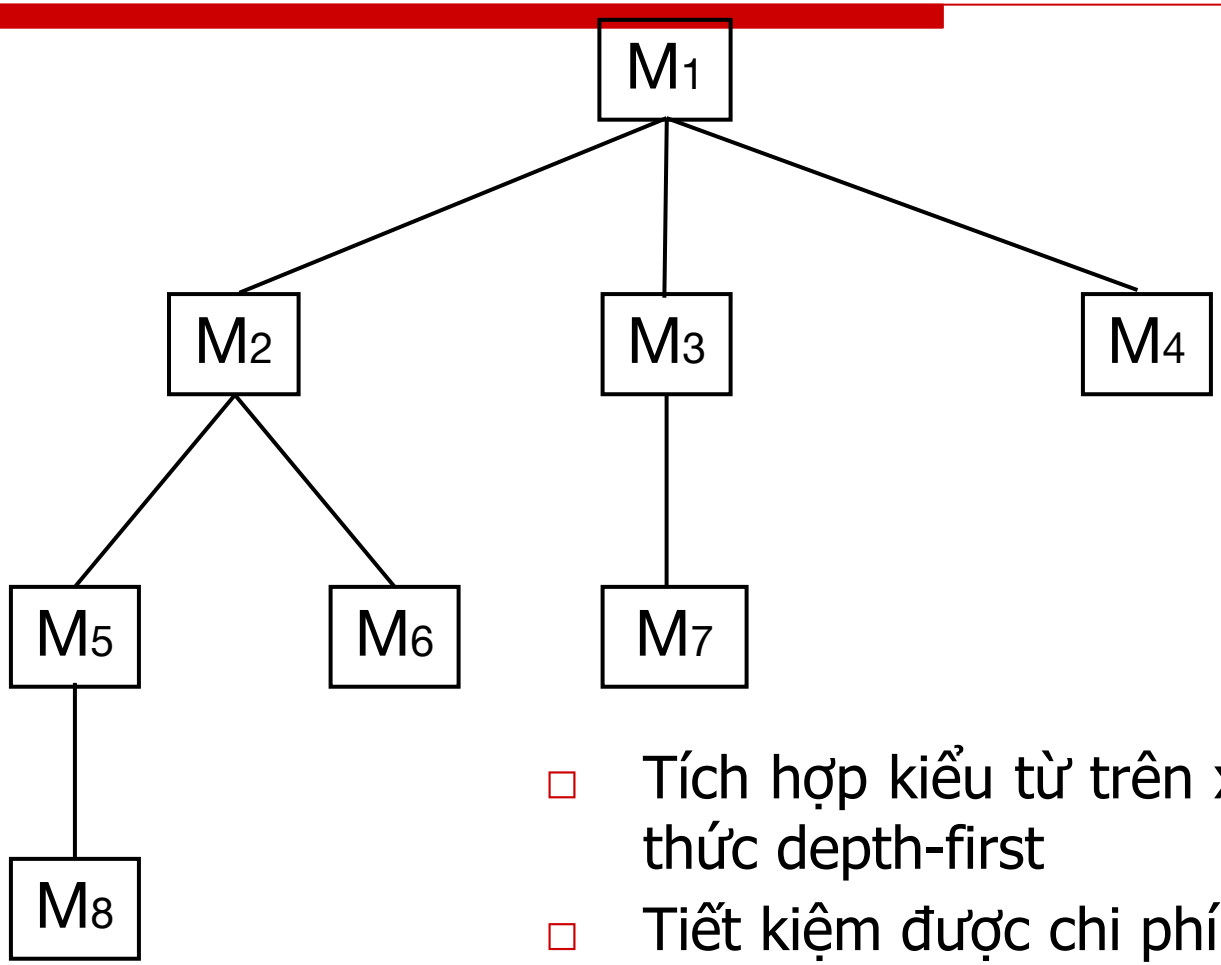
- Từng module mã nguồn đã hoạt động đúng. Liệu khi kết hợp chúng lại thành một nhóm lớn chúng có hoạt động đúng không ?
- Phải tiến hành kiểm nghiệm tích hợp để phát hiện lỗi liên quan đến giao tiếp giữa các module.

- Tránh tích hợp kiểu big-bang: tất cả các module được 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 hợp từ trên xuống

- Module chính được dùng như là driver, và stub được thay thế bởi các module con trực tiếp của của module chính này.
- Tuỳ thuộc vào cách tích hợp theo chiều sâu (depth-first) hoặc chiều ngang(breath-first), mỗi stub con được thay thế một lần bởi module tương ứng đã 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 từng module

Tích hợp từ trên xuống

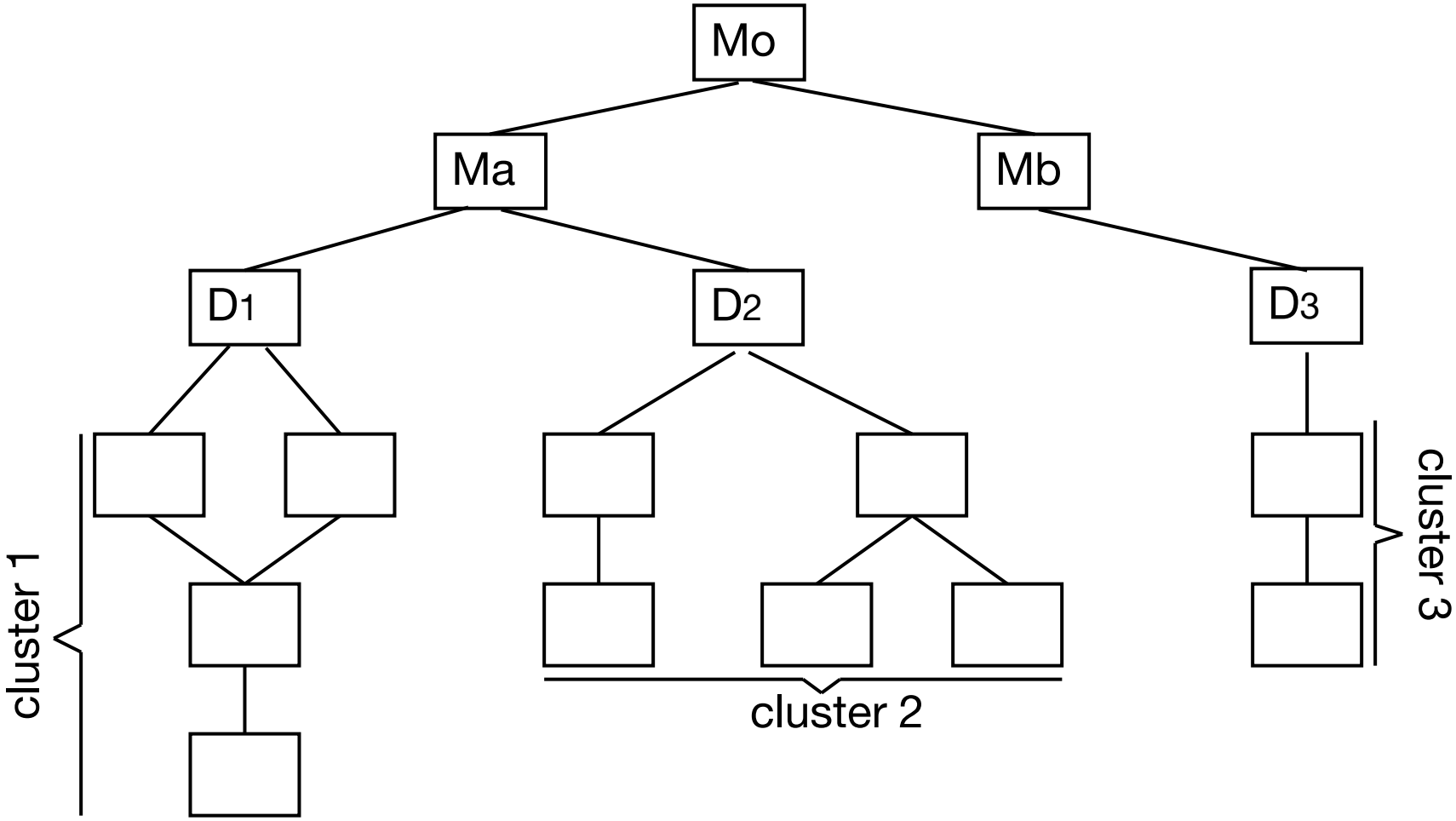


- Tích hợp kiểu từ trên xuống theo hình thức depth-first
- Tiết kiệm được chi phí tạo các driver

Tích hợp từ dưới lên

- Các module mức thấp nhất được kết hợp thành các nhóm thể hiện một chức năng con đặc biệt của phần mềm.
- Một driver được tạo ra để thao tác các test-case
- Các module được kiểm nghiệm theo từng nhóm (Cluster): là nhóm các module mà module phía trên cần đến khi kiểm nghiệm
- Driver được bỏ đi và các nhóm module được kết hợp dần lên phía trên trong sơ đồ phân cấp của chương trình.
- Tiết kiệm được chi phí tạo các stub

Tích hợp từ dưới lên



Kiểm nghiệm hồi 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
- Điều đó làm lộ ra một số lỗi không thể phát hiện được khi tiến hành kiểm nghiệm theo đơn vị
- **Phải kiểm nghiệm hồi quy khi tích hợp**
- 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 để thực hiện tự động

Kiểm nghiệm tính năng

- **Kiểm nghiệm tính năng** hiểu theo cách đơn giản nhất là: kiểm tra các chức năng của phần mềm đáp ứng được nhu cầu của khách hàng đã được xác định trong văn bản đặc tả yêu cầu của phần mềm

- Áp dụng kỹ thuật black-box

- Kiểm nghiệm tính năng bao gồm
  - Xem xét lại cấu hình phần mềm theo lược đồ triển khai
  - Kiểm nghiệm alpha
  - Kiểm nghiệm beta

Kiểm nghiệm tính năng

- Kiểm nghiệm alpha
  - Được tiến hành ngay tại nơi sản xuất phần mềm.
  - Nhà phát triển phần mềm sẽ quan sát người sử dụng dùng sản phẩm và ghi nhận lại những lỗi phát sinh để sửa chữa.
- Kiểm nghiệm beta
  - Phần mềm được kiểm tra bên ngoài phạm vi của đơn vị sản xuất.
  - Khách hành trực tiếp sử dụng và ghi nhận lỗi để báo lại cho nhà phát triển sửa chữa.

Kiểm nghiệm hướng đối tượng

- Về cơ bản chiến thuật kiểm nghiệm hướng đối tượng cũng theo thứ tự giống như kiểm nghiệm cổ điển:

Kiểm nghiệm đơn vị

→ **Kiểm nghiệm tích hợp**

→ **Kiểm nghiệm chức năng**

→ **Kiểm nghiệm toàn bộ hệ thống**

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Kiểm nghiệm hướng đối tượng

- Không thể tách rời từng tác vụ của đối tượng/lớp để kiểm nghiệm
  - Tác vụ được đóng bao trong lớp
  - Các lớp con có thể override một tác vụ nào đó

- Kiểm nghiệm đơn vị hướng đối tượng tập trung vào các lớp ☐ kiểm nghiệm hành vi của lớp

Kiểm nghiệm tích hợp hướng đối tượng

- Khái niệm sơ đồ phân cấp không còn nhiều ý nghĩa trong chương trình hướng đối tượng ☐ kiểm nghiệm tích hợp theo cách khác

- Hai hình thức kiểm nghiệm tích hợp hướng đối tượng
  - 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

- Dựa vào các use-case để soạn ra các kịch bản
- Ví dụ: một kịch bản cho hệ thống đăng ký môn học qua WEB

1. Login với username = "e59306547", password = "6547"
2. Chọn chức năng đăng ký môn học
3. Chọn 5 nhóm môn học của 5 môn: CNPM, AI, XLTHS, PTTK, XLSS trong đó có 2 nhóm trùng thời khoá biểu
4. Nhấn nút Submit

Chương trình phải báo lỗi và liệt kê 2 nhóm bị trùng thời khoá biểu

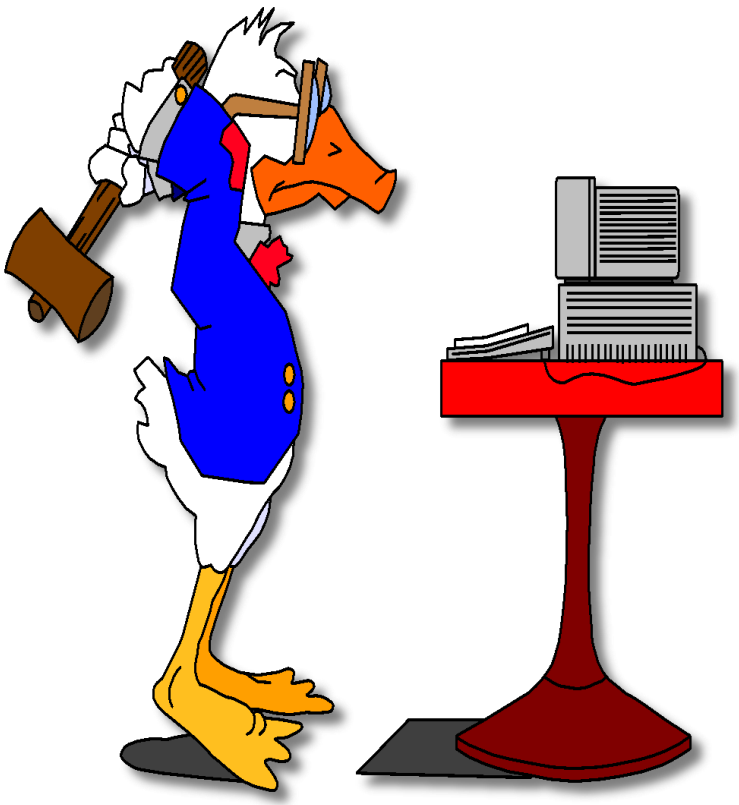
Nghệ thuật gỡ rối - DEBUG

- Gỡ rối là một quá trình nhằm loại bỏ các lỗi được phát hiện trong quá trình kiểm tra.

- Gỡ rối được thực hiện như là một kết quả của việc kiểm tra: lỗi phát hiện được ☐ tìm kiếm nguyên nhân ☐ sửa lỗi

- Có 3 hình thức gỡ rối: brute force, loại trừ nguyên nhân và theo vết. Nên dùng kết hợp cả 3 hình thức này.

Nghệ thuật gỡ rối



- Gỡ rối là công việc khó khăn và dễ gây tâm lý chán nản bởi nguyên nhân gây ra lỗi nhiều khi lại mơ hồ: do timeout, do độ chính xác, do chủ quan lập trình...
- Khả năng gỡ rối gần như là bẩm sinh của mỗi 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 để máy 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.

Copyright 2004 – Th.S Nguyễn Cao Trí – caotri@hcmut.edu.vn

Loại trừ nguyên nhân

- Phương pháp này dựa trên nguyên tắc phân chia nhị phân.
- Cách thực hiện:
  - Khi một lỗi được phát hiện, cố gắng đưa ra một danh sách các nguyên nhân có thể gây ra lỗi.
  - Danh sách này được nghiệm lại để loại bỏ dần các nguyên nhân không đúng cho đến khi tìm thấy một nguyên nhân khả nghi nhất.
  - Khi đó dữ liệu kiểm nghiệm sẽ được tinh chế lại để tiếp tục tìm lỗi.

Theo vết

- Là một phương pháp gỡ lỗi khá phổ biến có thể dùng thành công trong các chương trình nhỏ nhưng khó áp dụng cho đối 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 từng dòng mã nguồn cho đến khi tìm thấy dòng gây ra lỗi.

Kết thúc môn học

Thi cuối kỳ ?

Giới thiệu-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 !