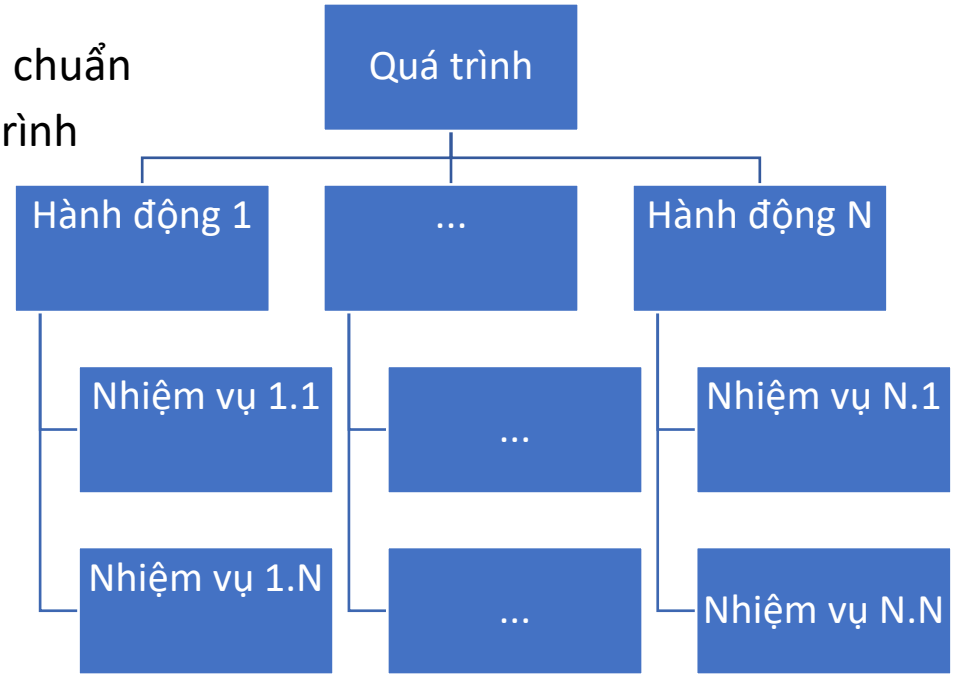
Phát triển phần mềm ITSS

Tổng kết các nội dung môn học

Quy trình vòng đời phần mềm

- Theo "ISO / IEC 12207: 2008, Hệ thống và kỹ thuật phần mềm – Quy trình vòng đời phần mềm"
  - Quy trình phát triển phần mềm tiêu chuẩn quốc tế và mới nhất
- "Vòng đời bắt đầu với một ý tưởng hoặc một nhu cầu có thể được phần mềm đáp ứng toàn bộ hoặc một phần và kết thúc bằng việc phần mềm ngừng hoạt động".
- Các công việc thực hiện tiêu chuẩn
  - Phân cấp như các quy trình



Quy trình vòng đời

- "Tiêu chuẩn này nhóm các hoạt động có thể được thực hiện trong vòng đời của hệ thống phần mềm thành 7 nhóm quá trình" [1]*:
  1. Các quy trình thỏa thuận: 2 quy trình
  2. Các quy trình hỗ trợ dự án của tổ chức: 5 quy trinh
  3. Quy trình dự án: 7 quy trình
  4. Quy trình kỹ thuật: 11 quy trình
  5. Quy trình triển khai: 6 quy trình

     Mục đích: "để tạo ra một phần tử hệ thống cụ thể được triển khai như một sản phẩm hoặc dịch vụ phần mềm" [1**.
  6. Quy trình hỗ trợ phần mềm: 8 quy trình
  7. Quy trình tái sử dụng phần mềm: 3 quy trình

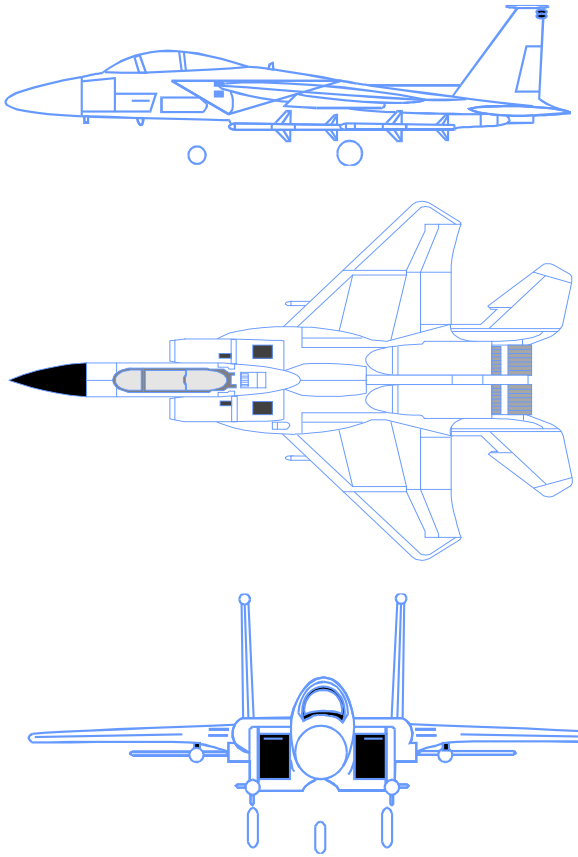
[1]*: clause 5.2.1; pp. 13, [1]**: clause 7.1.1.1; pp. 57,

Tổng kết

- "Quy trình Vòng đời Phần mềm – SLCP" là các quy trình tiêu chuẩn quốc tế tập trung vào việc phát triển và hỗ trợ Phần mềm Ứng dụng.
- SLCP có thể được sử dụng như một ngôn ngữ chung giữa các bên liên quan như bên mua và nhà cung cấp. Họ có thể giao tiếp hoặc ra lệnh phát triển phần mềm bằng SLCP. Ví dụ, chúng ta có thể nói "Để đặt hàng quy trình thiết kế chi tiết phần mềm hoặc các quy trình triển khai phần mềm sau này của hệ thống thư viện mới".

Mô hình hóa

- Mô hình là sự đơn giản hóa hệ thống.



Ngôn ngữ mô hình thống nhất (UML)

- "UML là ngôn ngữ dành cho
  - Hình dung
  - Xác định
  - Cấu tạo
- Lập hồ sơ tạo tác của một hệ thống sử dụng nhiều phần mềm "[1].

[1]: Chapter 2, Section 2.1

Lịch sử của UML

UML 2.0 (2004)

UML 1.5 (March, '03)

UML Partners' Expertise

UML 1.1 (Sept. '97)

UML 1.0 (Jan. '97)

UML 0.9 and UML 0.91 (June '96) (Oct. '96)

Unified Method 0.8 (OOPSLA ' 95)

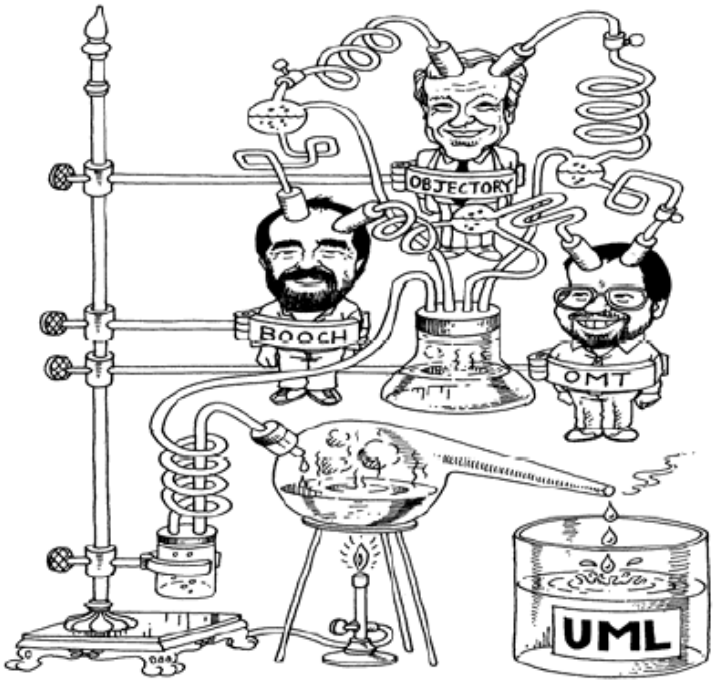
Booch ' 93

Booch '91

OMT - 1

OOSE

Other Methods

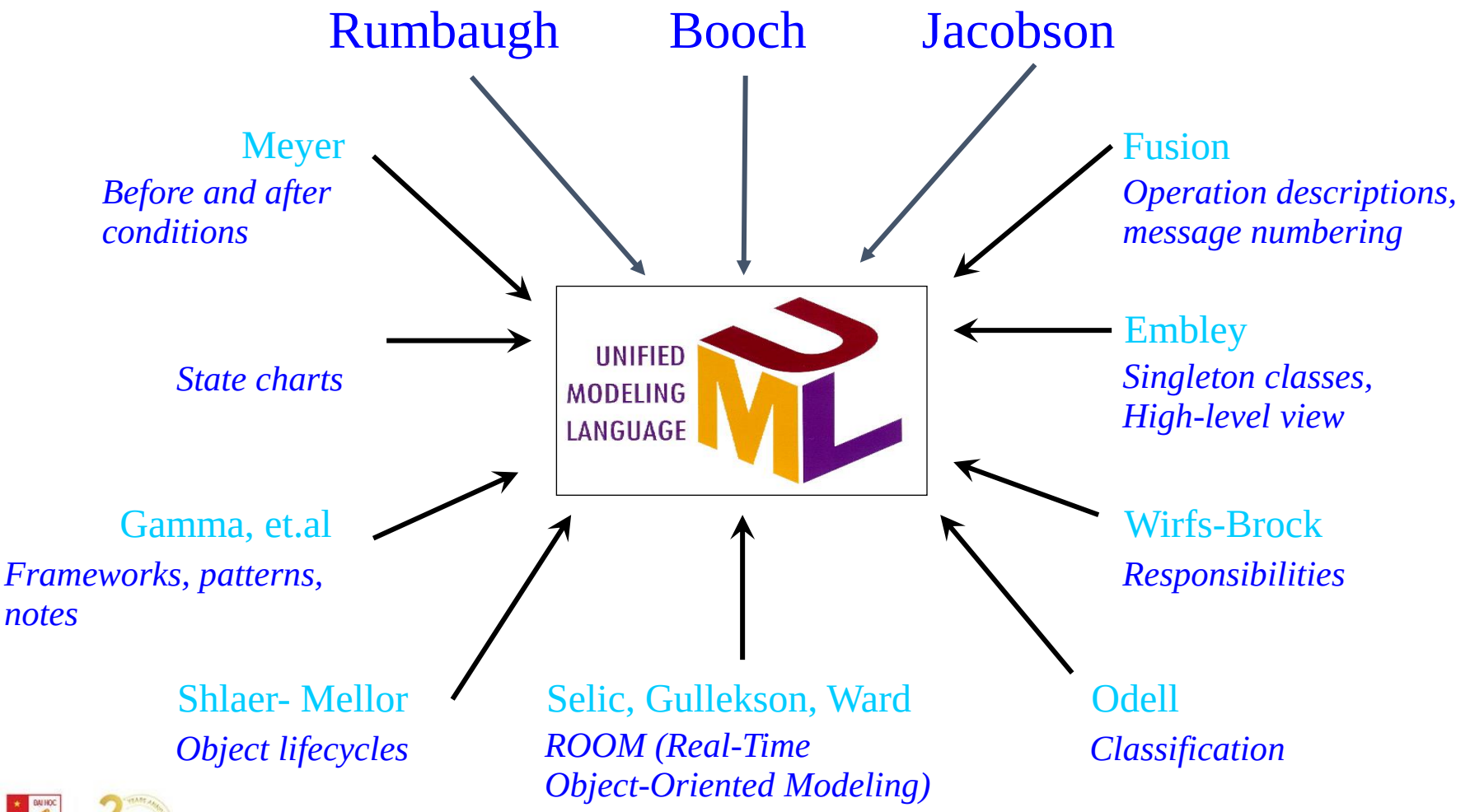


Booch91 (Grady Booch): Conception, Architecture

OOSE (Ivar Jacobson): Use cases

OMT (Jim Rumbaugh): Analysis

Đầu vào cho UML



Chế độ xem tĩnh so với chế độ xem động

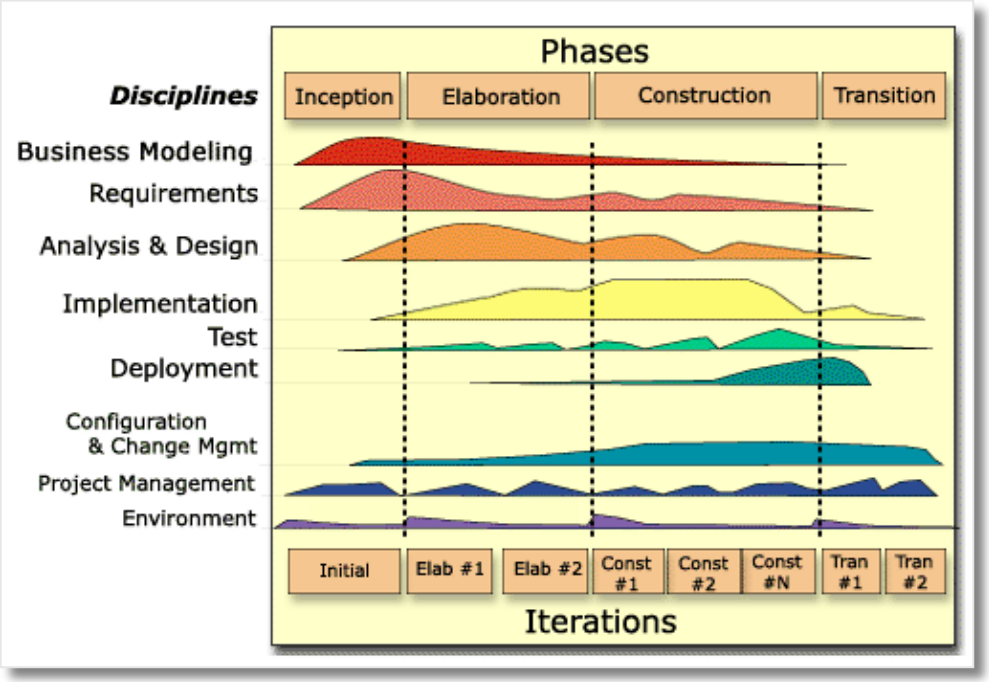
Vai trò của các bộ phận thành phần và cách chúng liên quan với nhau

Cách các thành phần tương tác với nhau và / hoặc thay đổi trạng thái nội bộ theo thời gian

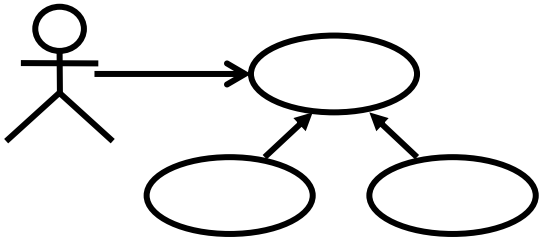
Phân tích và thiết kế hệ thống

Mục đích của Phân tích và Thiết kế là:

- Chuyển đổi các yêu cầu thành một thiết kế của hệ thống tương lai.
- Phát triển một kiến trúc mạnh mẽ cho hệ thống.
- Điều chỉnh thiết kế để phù hợp với môi trường thực hiện, thiết kế để thực hiện.

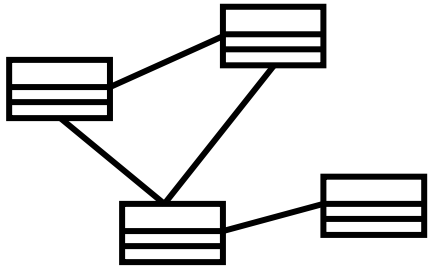


Phân tích và thiết kế hướng đối tượng

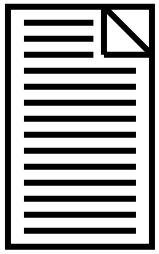


Use-Case Model

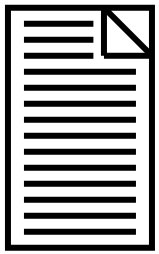
Analysis and Design



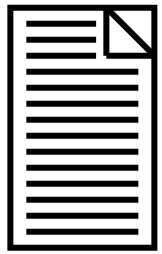
Design Model



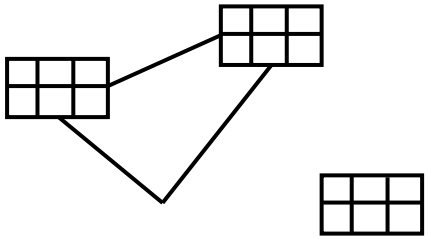
Glossary



Supplementary Specification

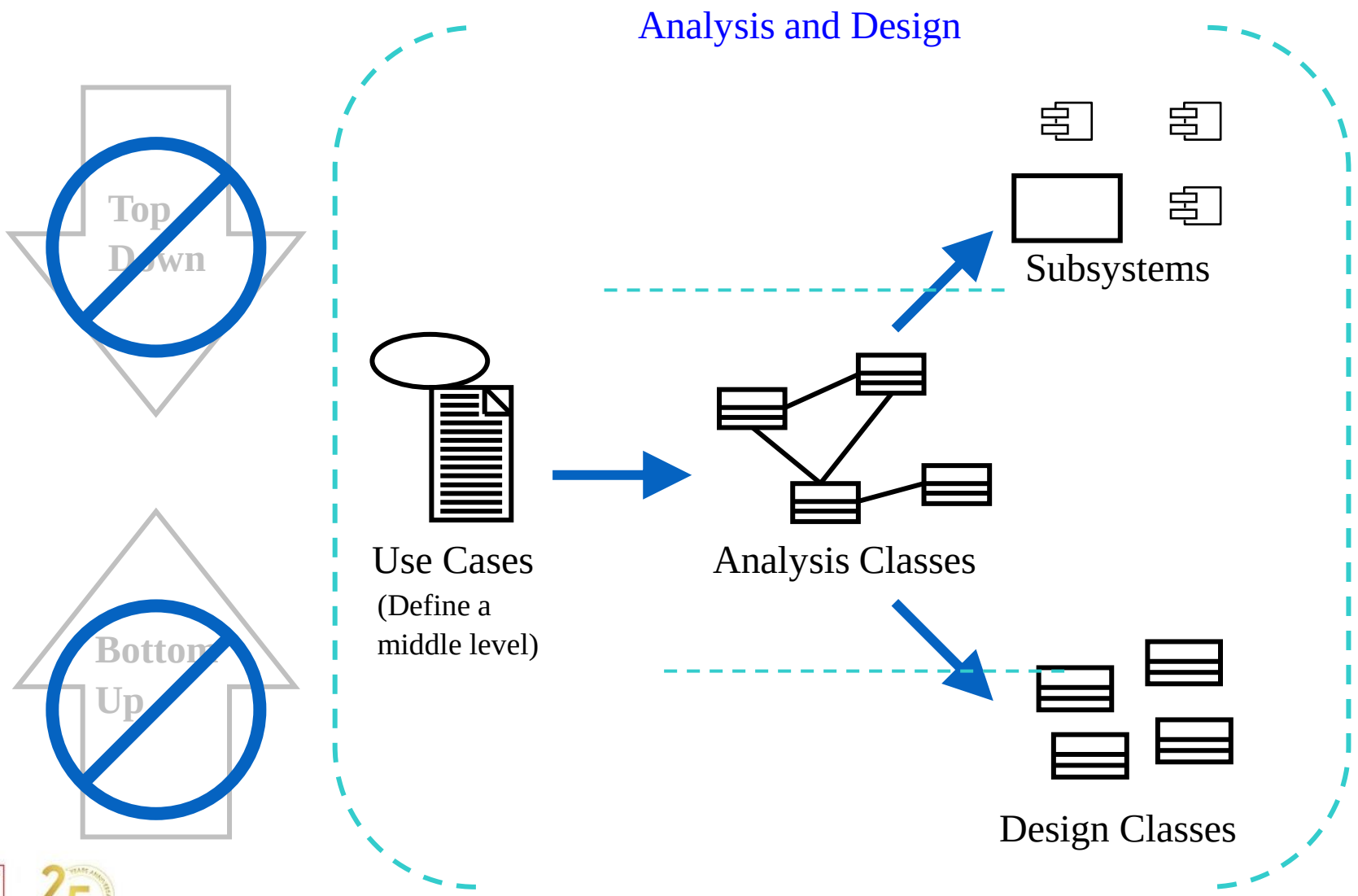


Architecture Document



Data Model

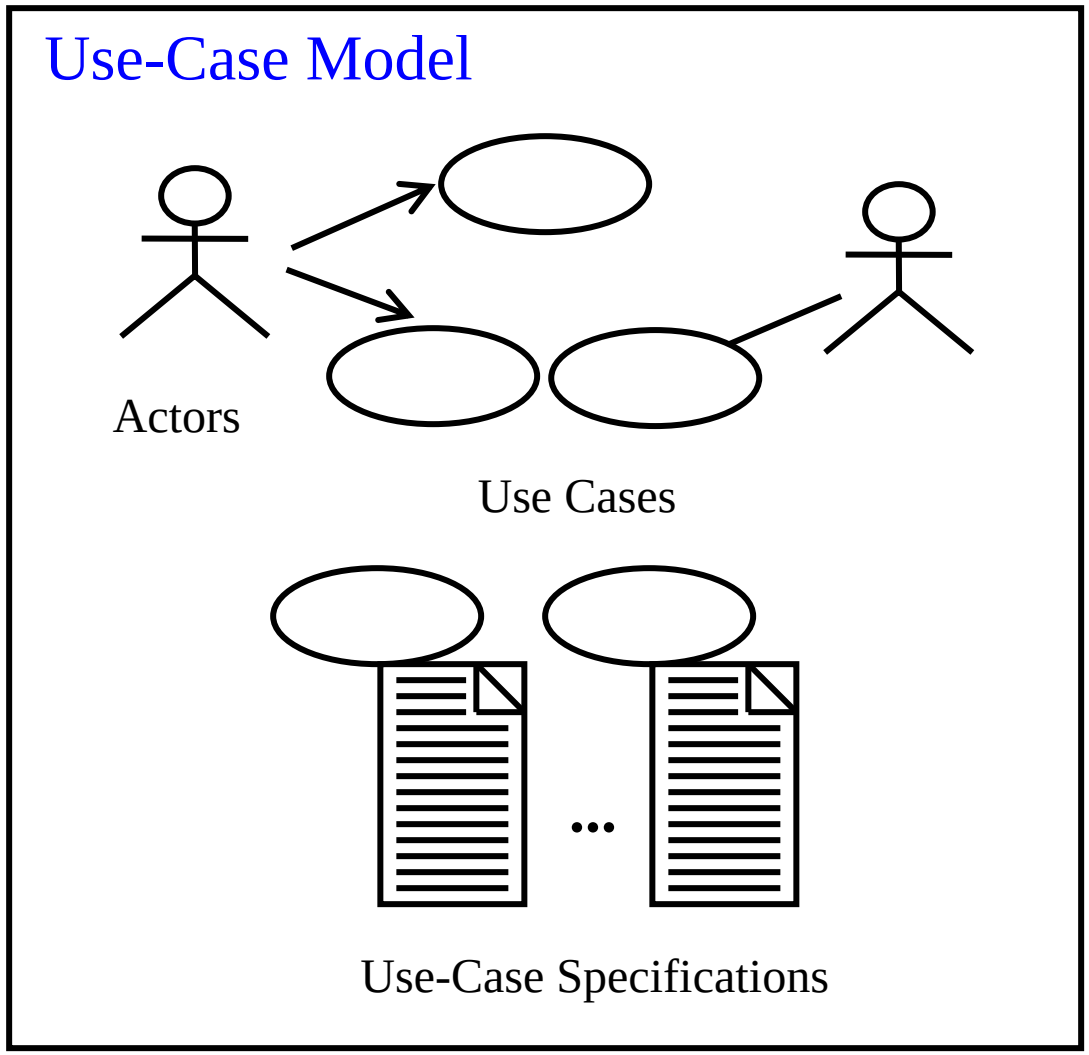
Phân tích và thiết kế không từ trên xuống hoặc từ dưới lên

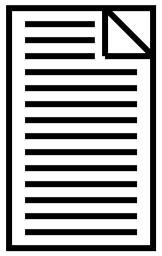


Các bước phân tích và thiết kế

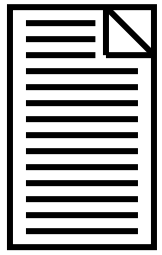
Hoạt động	Bước	Mô tả	Người thực hiện
Xác định kiến trúc ứng viên	1. Phân tích kiến trúc	• Một lần ở giai đoạn đầu • Bỏ qua nếu rủi ro kiến trúc thấp	Kiến trúc sư
Phân tích hành vi	**2. Phân tích ca sử dụng**	• **Mỗi trường hợp ca sử dụng**	**Nhà thiết kế**
Tinh chỉnh kiến trúc	**3. Xác định các yếu tố thiết kế**	• **Khớp nối và gắn kết** • **Khả năng tái sử dụng**	**Kiến trúc sư**
	4. Xác định cơ chế thiết kế	• Mẫu thiết kế	
	5. Mô tả kiến trúc thời gian chạy	• Bỏ qua nếu không phải đa luồng • Chế độ xem	
	6. Mô tả phân phối	• Kiến trúc vật lý	
Thành phần thiết kế	**7. Thiết kế ca sử dụng**	• **Mỗi trường hợp sử dụng**	**Nhà thiết kế**
	8. Thiết kế hệ thống con		
	9. Thiết kế lớp		
Thiết kế DB	**10. Thiết kế CSDL**		

Mô hình hoá Yêu cầu



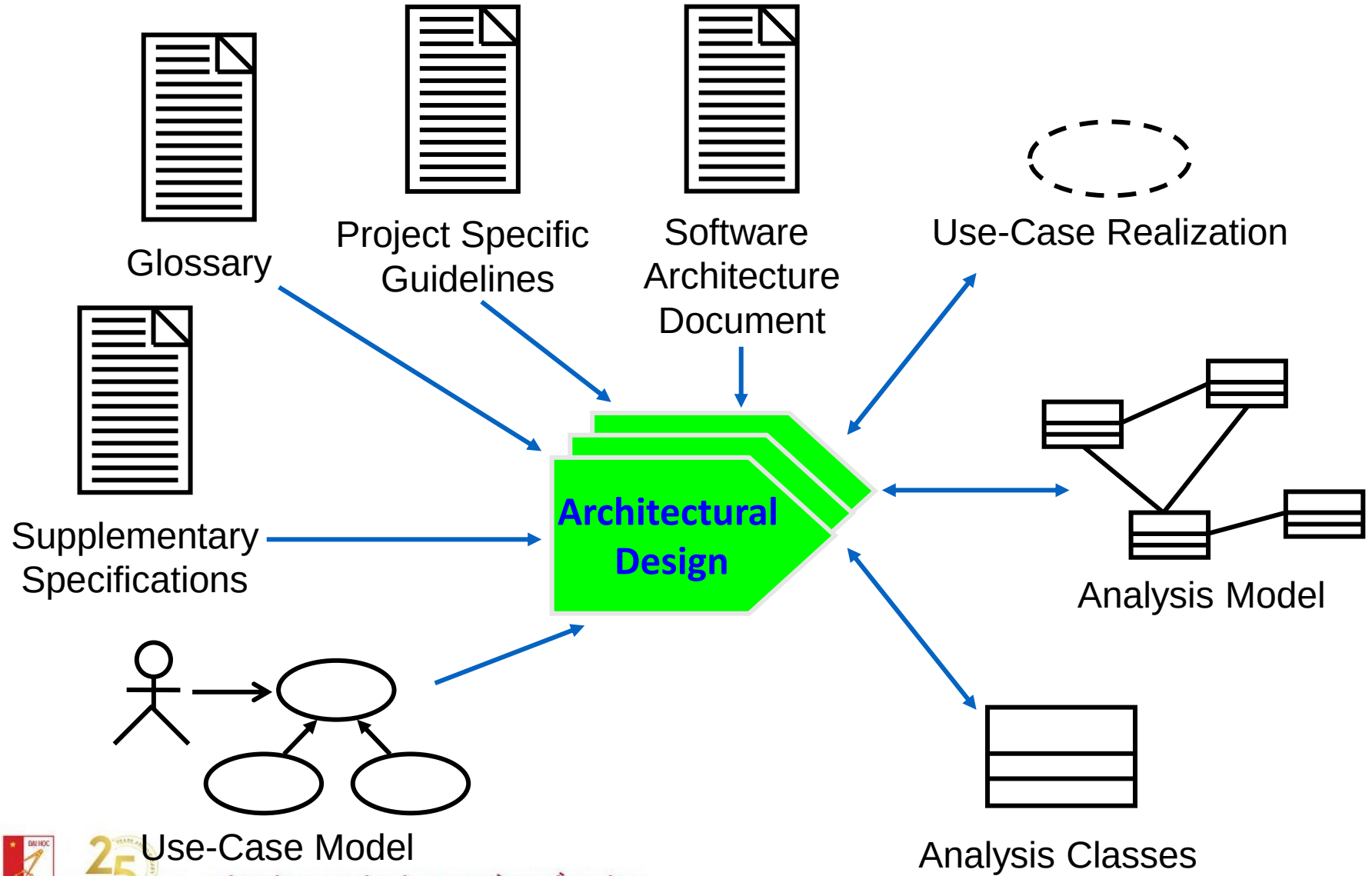


Glossary

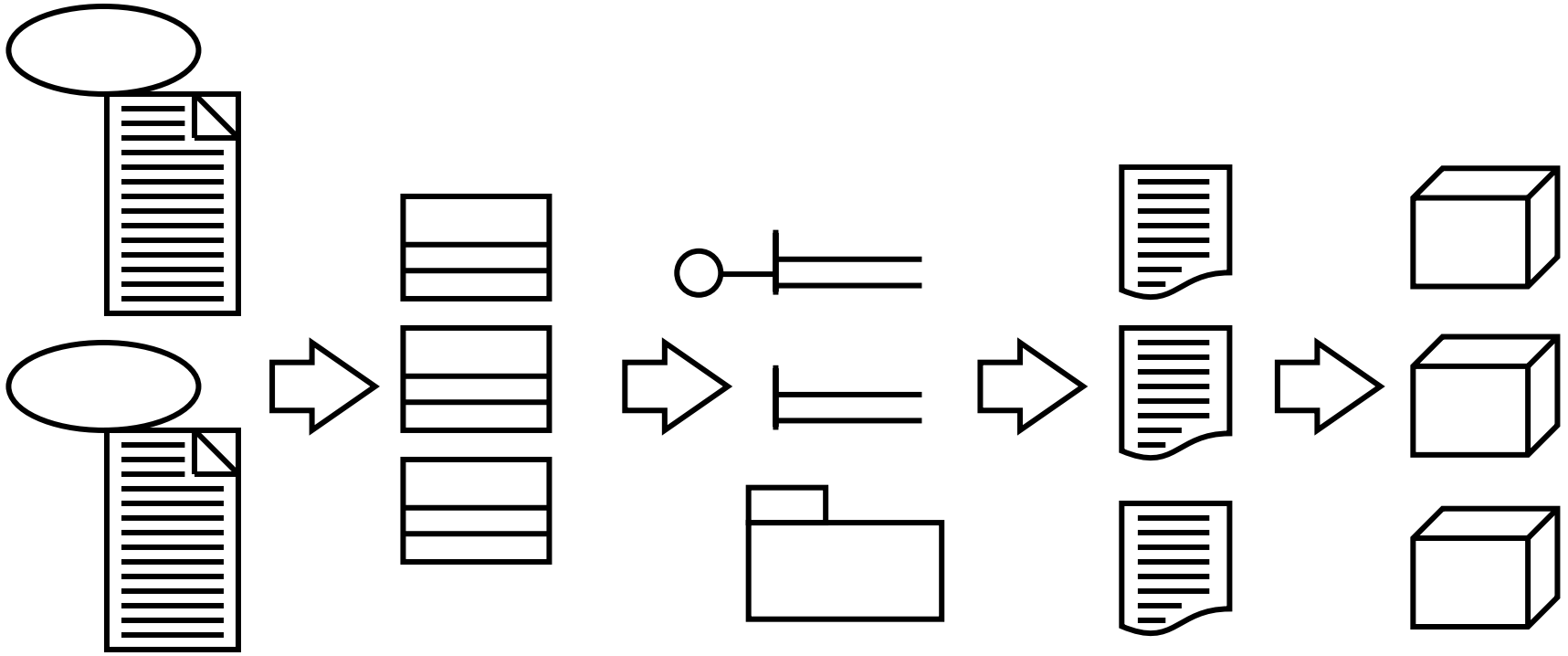


Supplementary Specification

Thiết kế kiến trúc

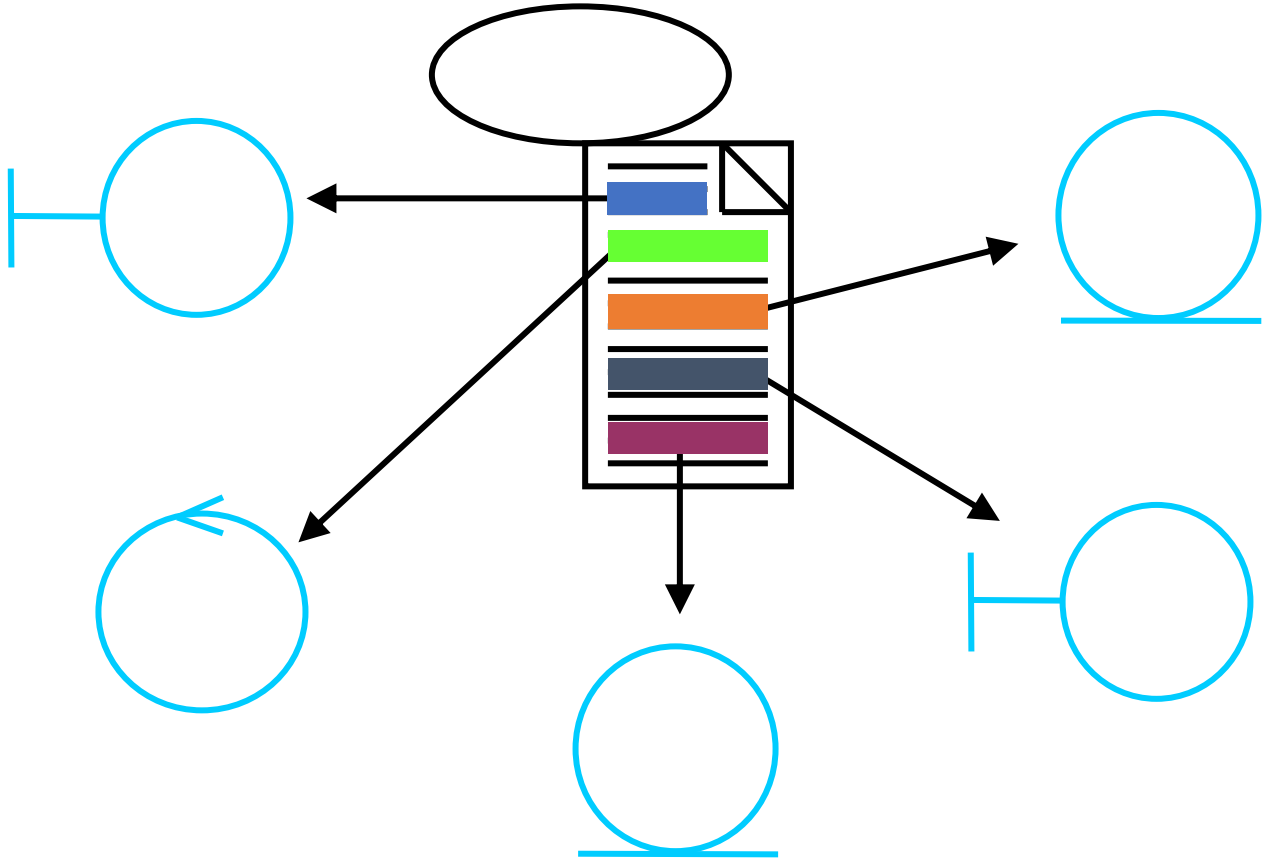


Các lớp phân tích: Bước đầu tiên hướng tới thực thi

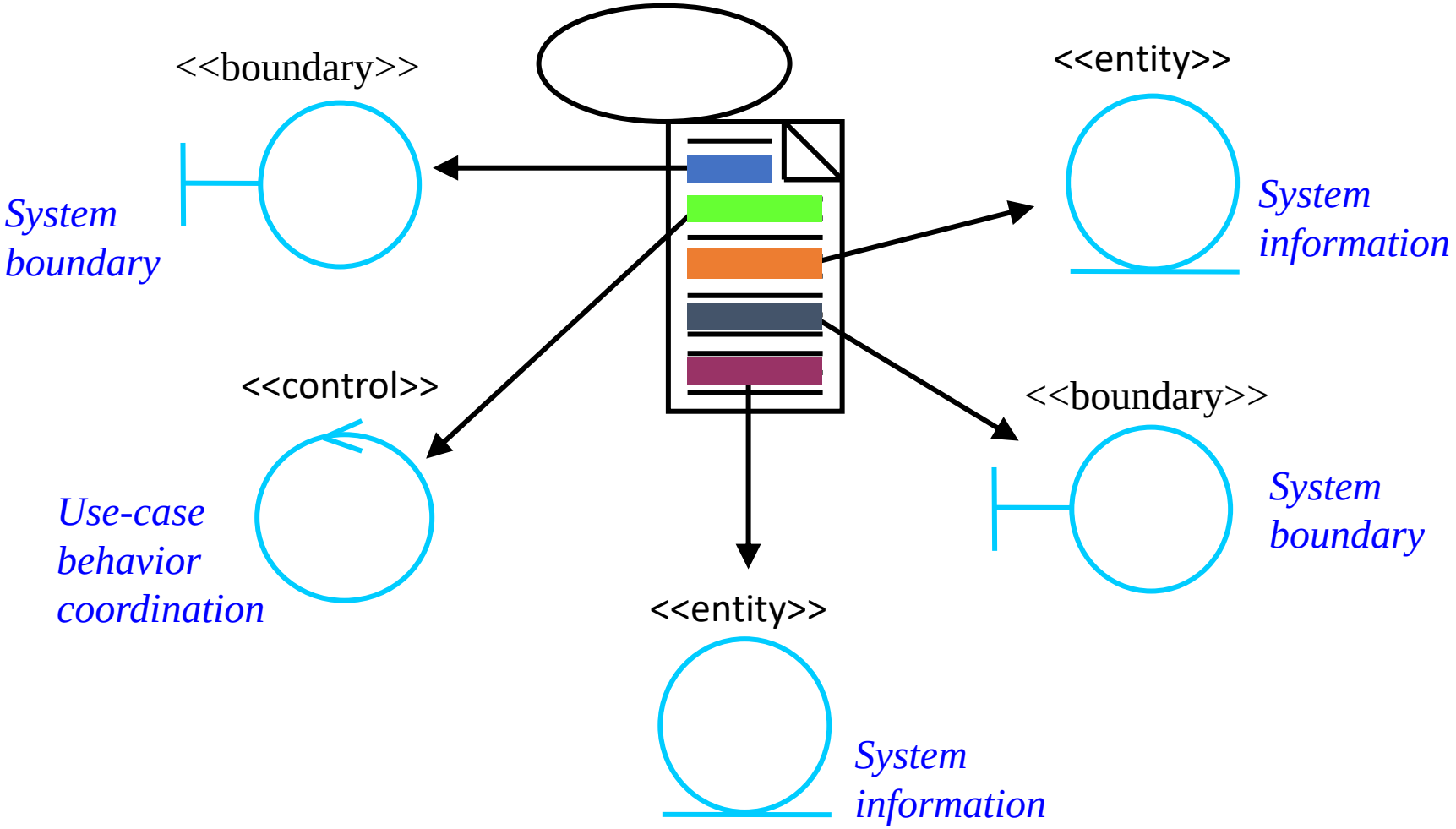


Tìm lớp từ hành vi sử dụng trường hợp

- Hành vi đầy đủ của một trường hợp sử dụng đã được phân phối cho các lớp học phân tích

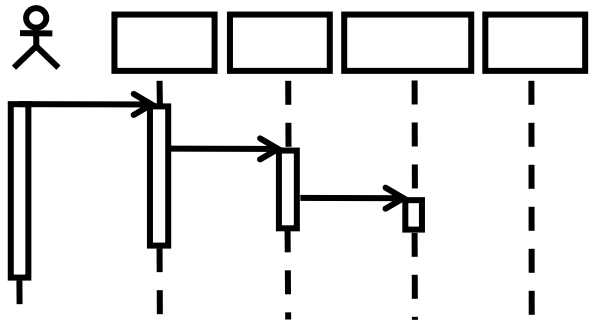


Các loại lớp phân tích



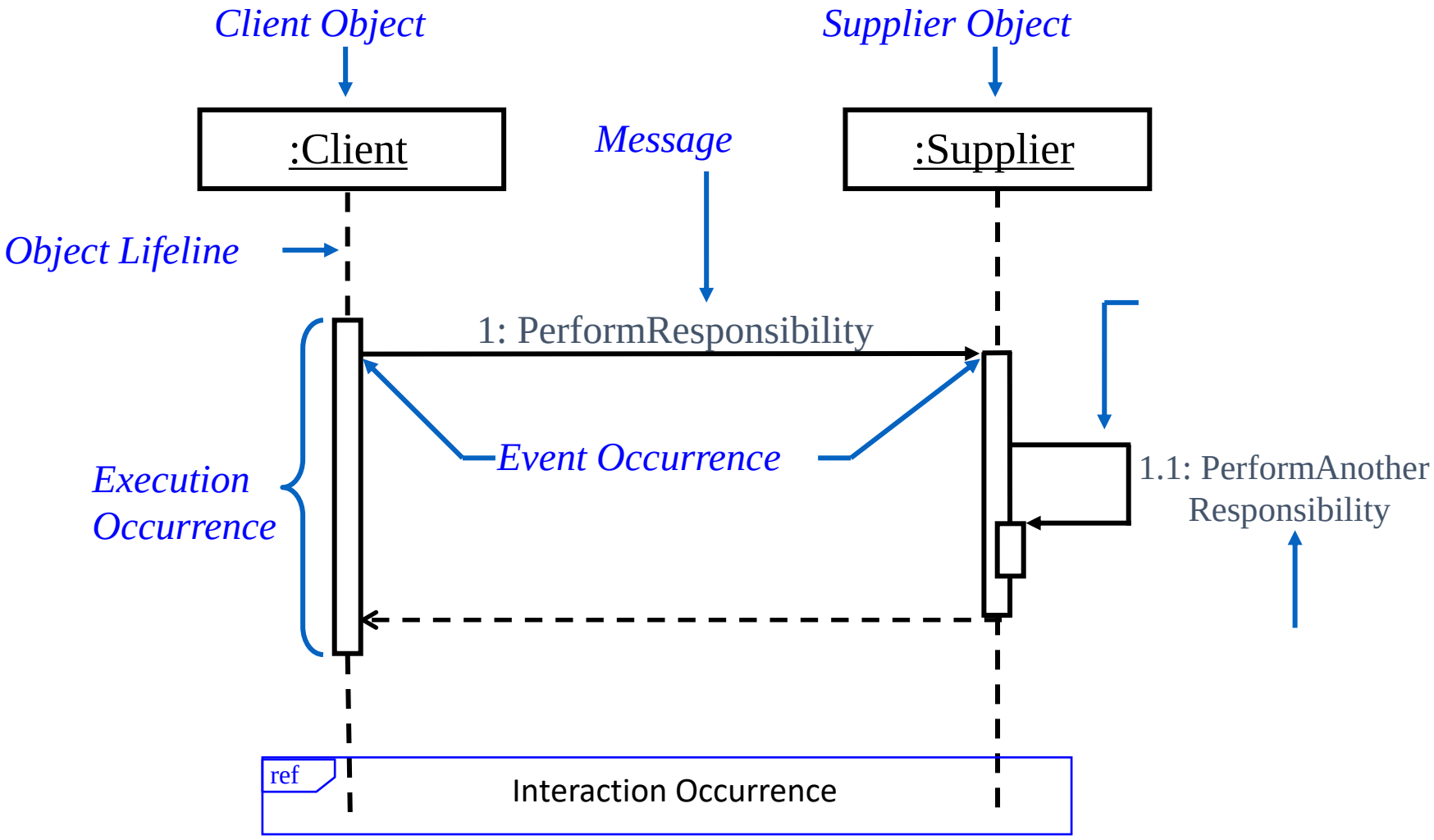
Sơ đồ trình tự

- Sơ đồ trình tự là một sơ đồ tương tác nhấn mạnh thứ tự thời gian của thư.
- Sơ đồ hiển thị:
- Các đối tượng tham gia vào sự tương tác.
- Chuỗi tin nhắn được trao đổi.



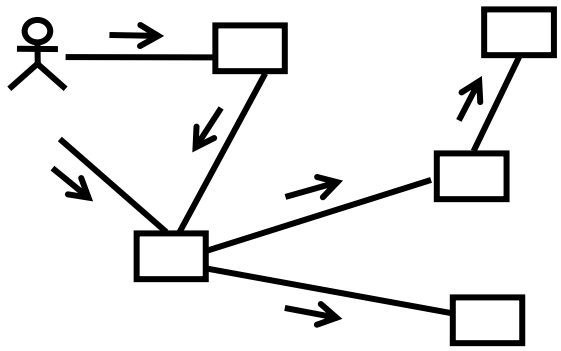
Sequence Diagram

Giải thích sơ đồ trình tự



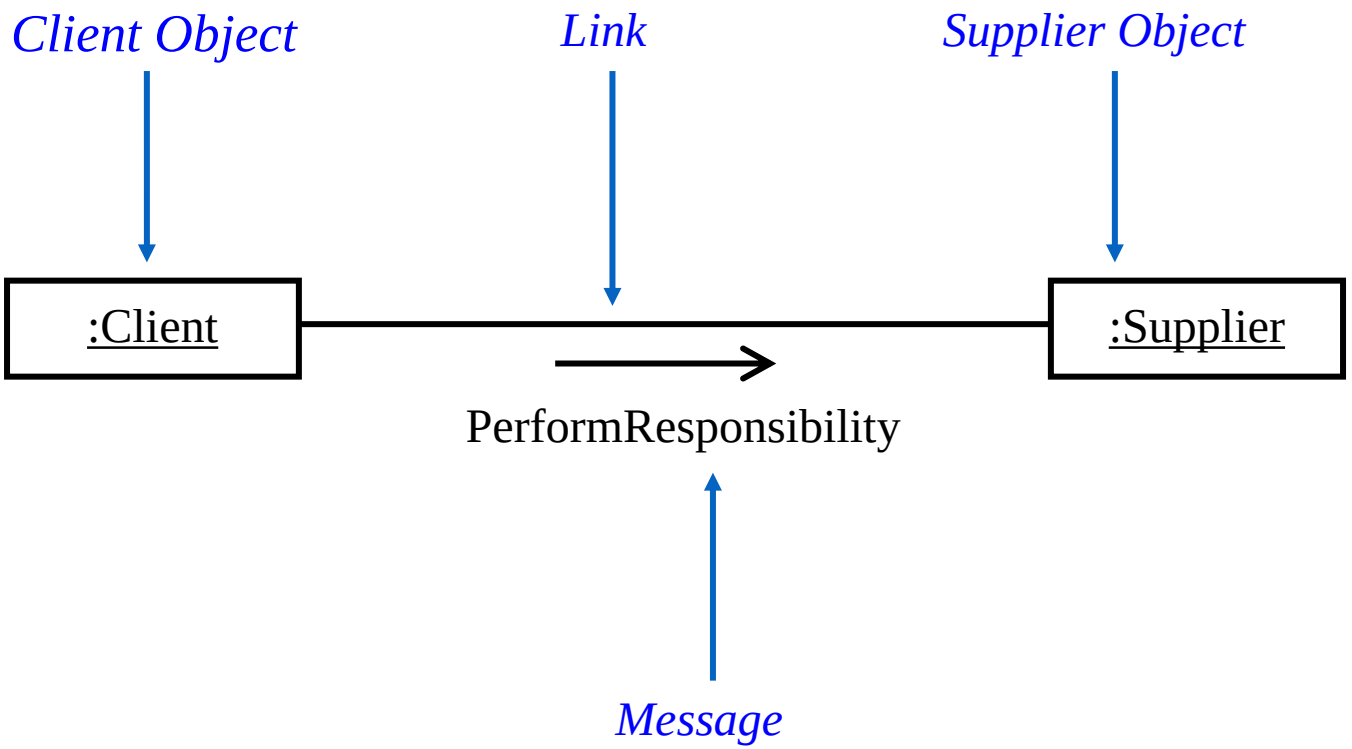
Sơ đồ giao tiếp

- Sơ đồ giao tiếp nhấn mạnh tổ chức của các đối tượng tham gia vào một tương tác.
- Sơ đồ giao tiếp cho thấy:
- Các đối tượng tham gia vào sự tương tác.
- Liên kết giữa các đối tượng.
- Thư được chuyển giữa các đối tượng.

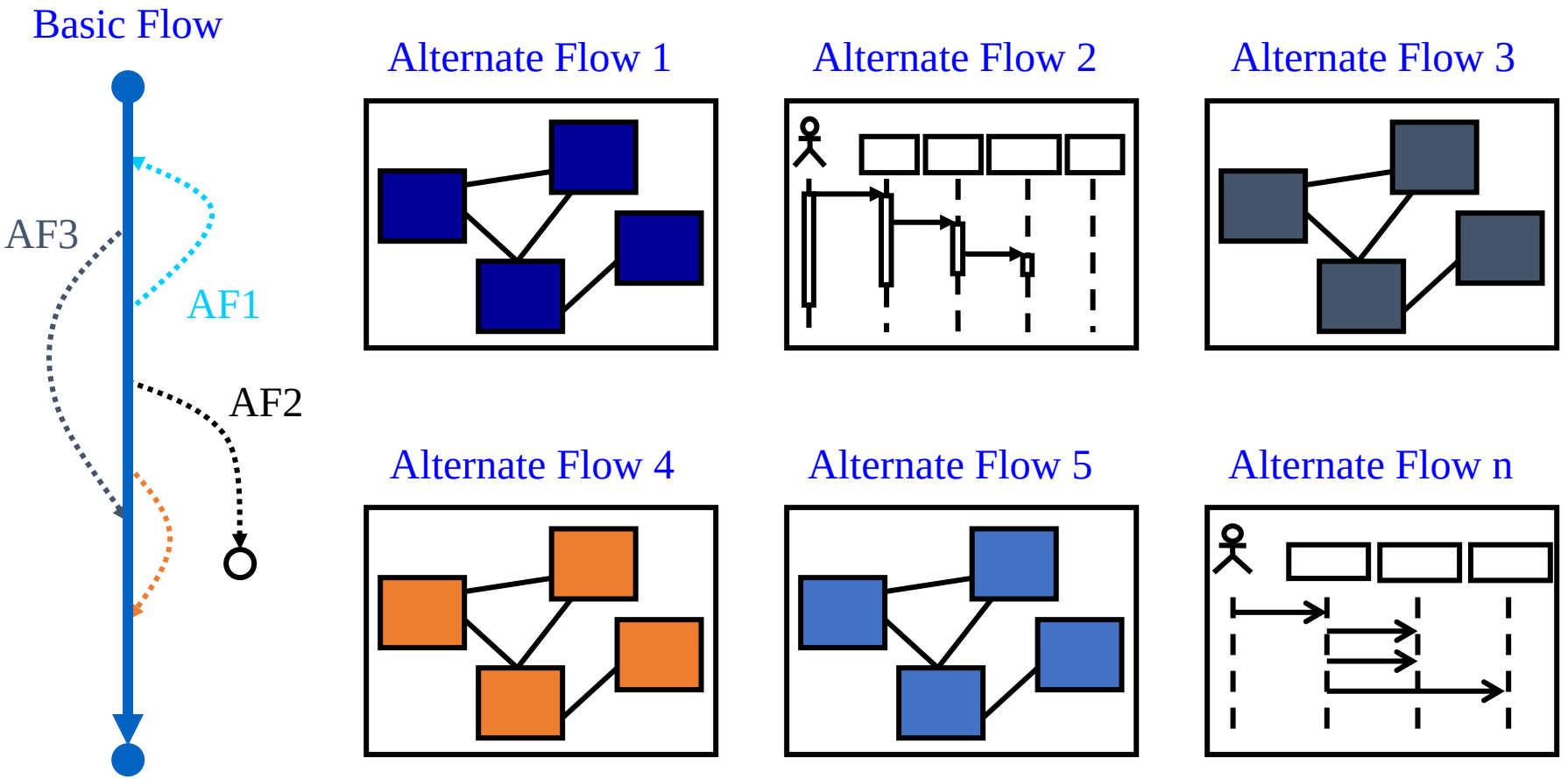


Communication Diagrams

Giải thích sơ đồ giao tiếp

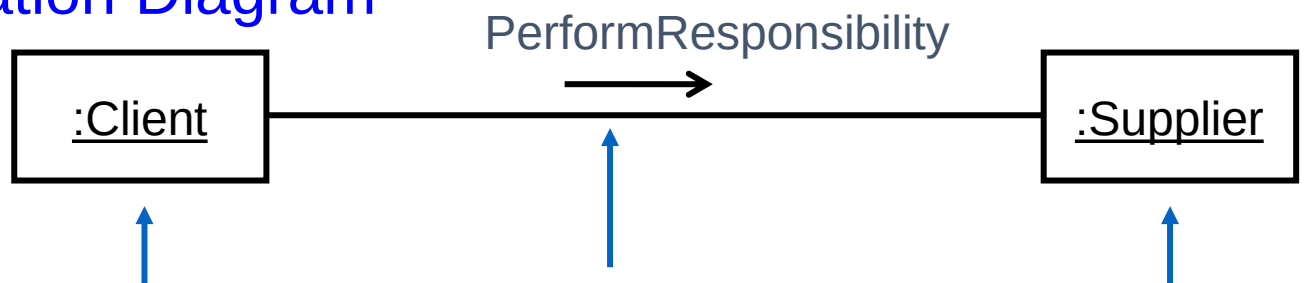


Một sơ đồ tương tác có thể không đủ tốt

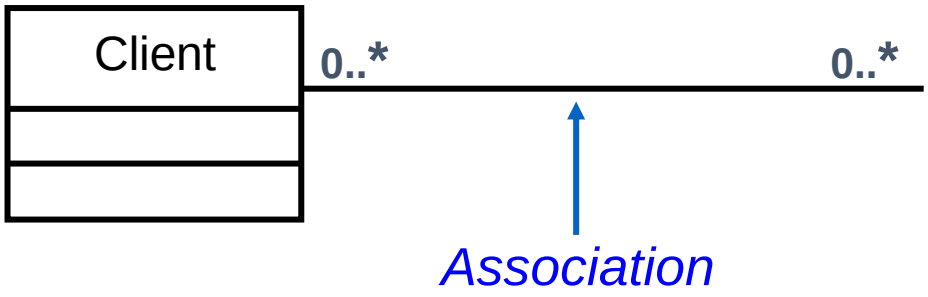


Tìm mối quan hệ

Communication Diagram



Class Diagram

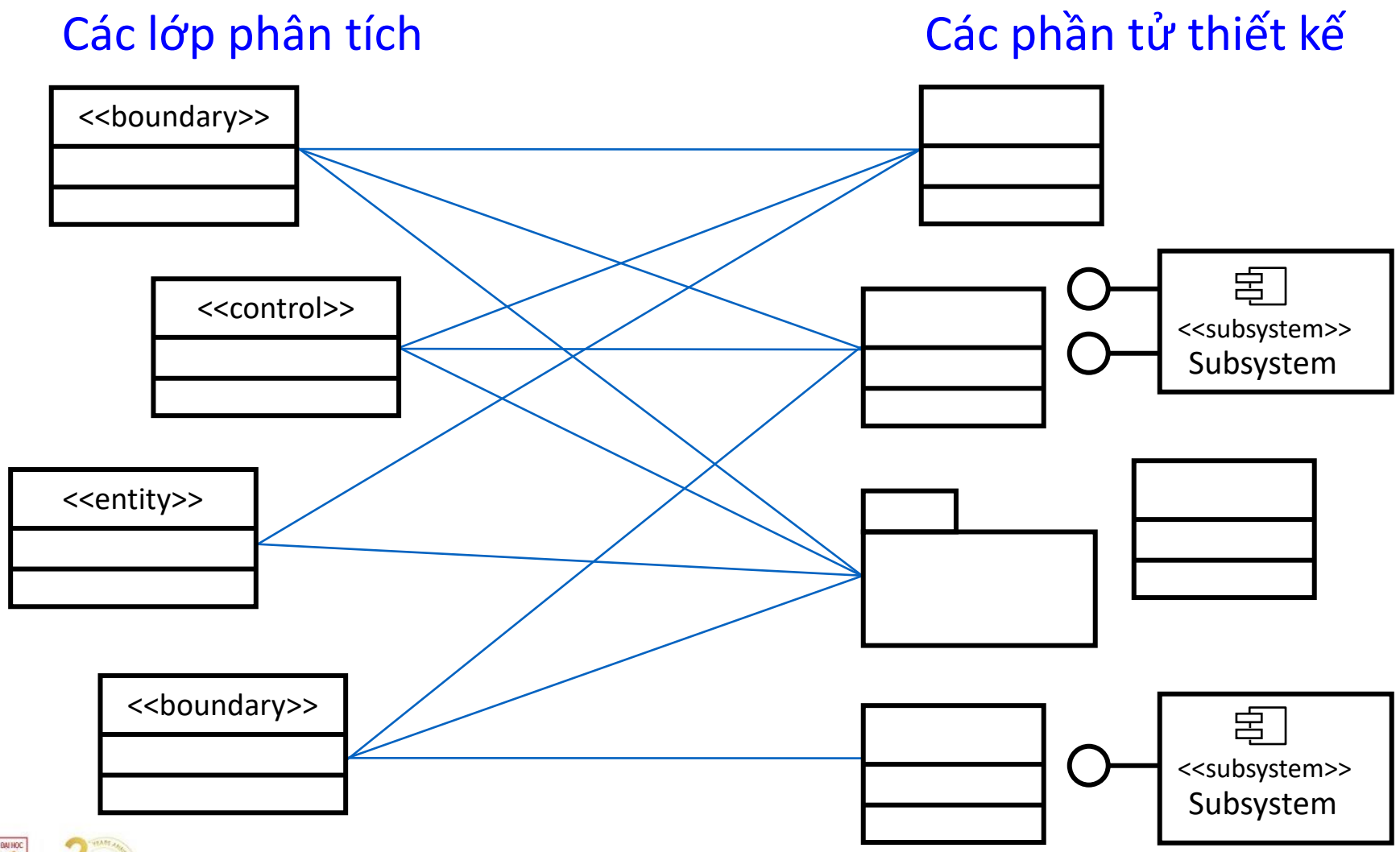


Relationship for every link!

Mục tiêu: Xác định các phần tử thiết kế

- Định nghĩa mục đích của các phần tử thiết kế và làm rõ vòng đời của chúng được thực hiện khi nào
- Phân tích sự tương tác của các lớp phân tích và xác định các phần tử mô hình thiết kế => Thiết kế lớp

Từ Lớp phân tích tới Các phần tử thiết kế



Ánh xạ nhiều-nhiều

Thiết kế giao diện đồ họa người dùng

1.1. Chuẩn hóa cấu hình màn hình
1.2. Tạo hình ảnh màn hình
1.3. Tạo biểu đồ chuyển tiếp màn hình
1.4. Tạo đặc tả màn hình

Thiết kế giao diện hệ thống/thiết bị

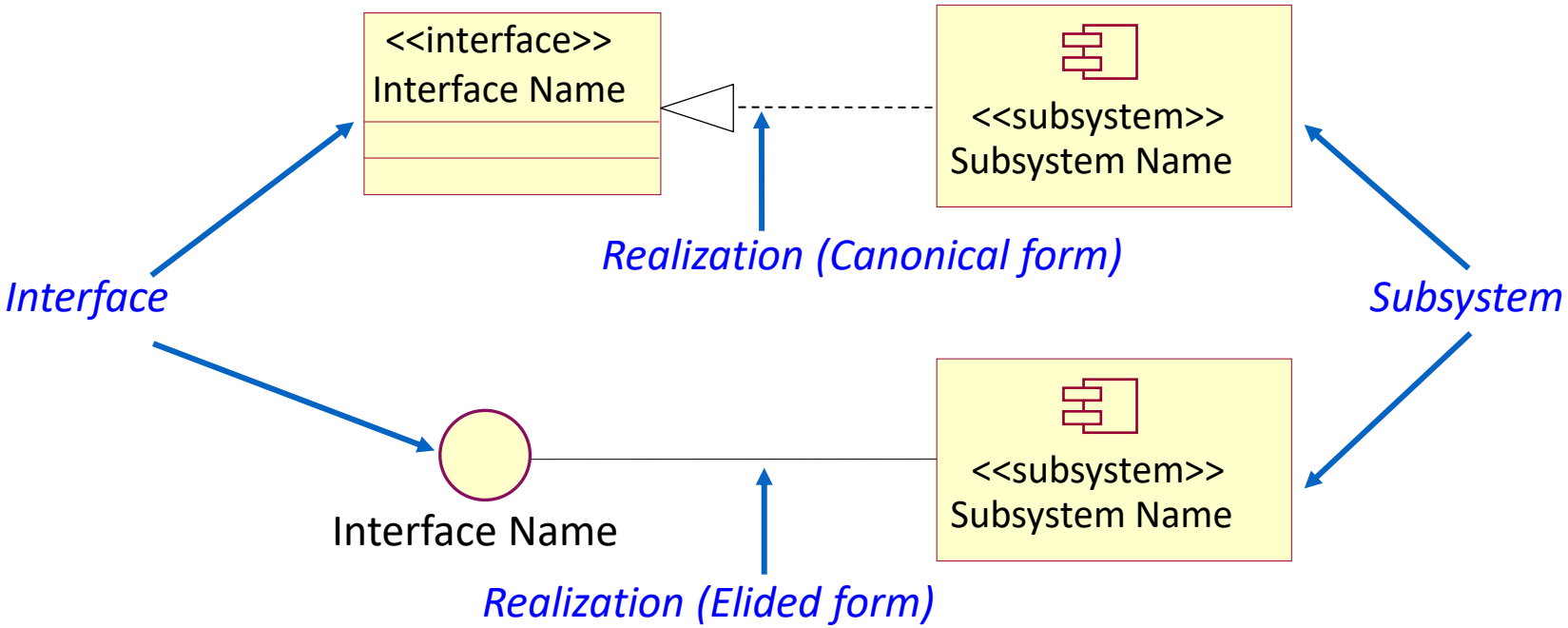
2.1. Xác định hệ thống con

2.2. Xác định giao diện hệ thống con

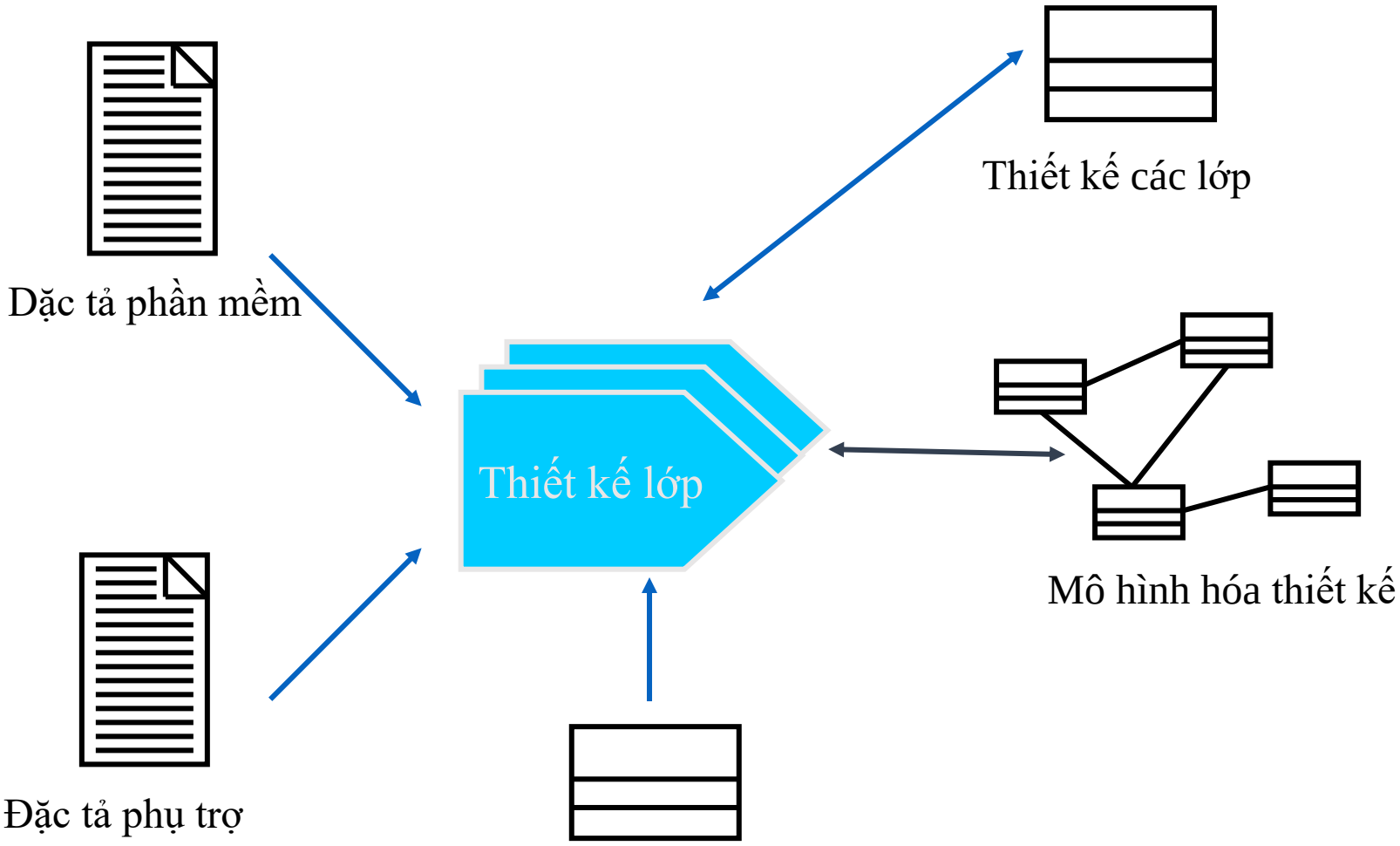
2.3. Thiết kế hệ thống con

Hệ thống con và giao diện

- Tạo một hoặc nhiều giao diện mà định nghĩa các hành vi của hệ thống con (subsystem)



Thiết kế lớp


Thiết kế các lớp
Dặc tả phần mềm
Thiết kế lớp
Mô hình hóa thiết kế
Đặc tả phụ trợ
Các lớp phân tích

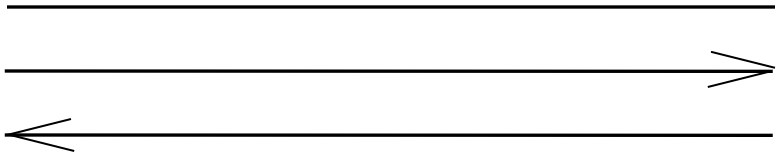
Thiết kế lớp

1. Tạo các lớp khởi tạo
2. Định nghĩa ra các thao tác/phương thức
3. Định nghĩa ra mối quan hệ giữa các lớp
4. Định nghĩa ra các trạng thái
5. Định nghĩa ra các thuộc tính
6. Sơ đồ lớp

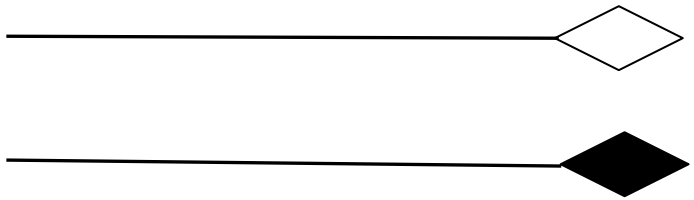
Mỗi quan hệ giữa các class

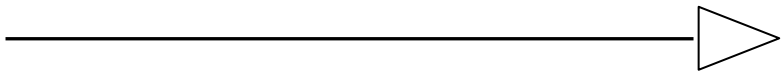
- Association

(kết hợp)

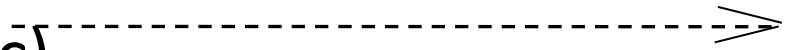


- Aggregation (kết tập)
- Composition (hợp thành)

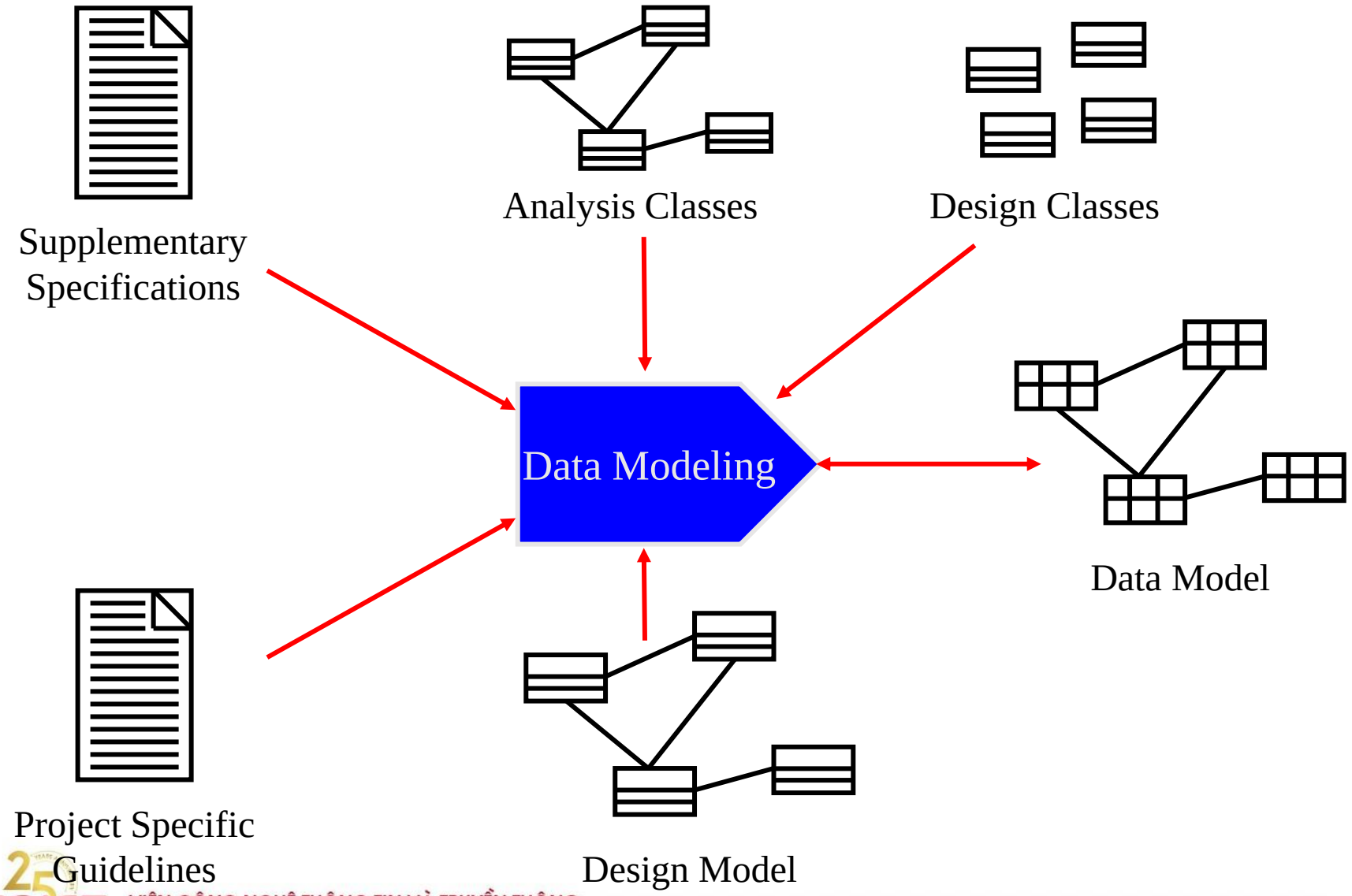




- Inheritance (Kế thừa)
- Dependence (Phụ thuộc)

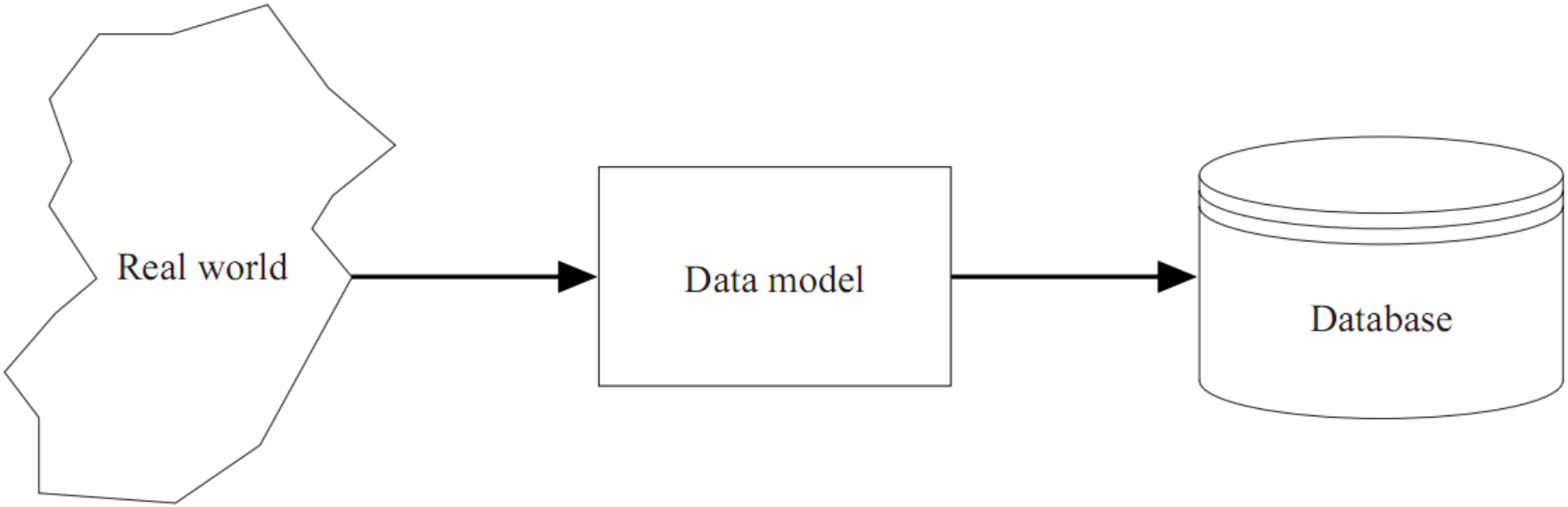


Mô hình hoá dữ liệu



Mô hình dữ liệu

- Mô hình hoá dữ liệu:
  - Quá trình trừu tượng hoá và tổ chức cấu trúc của thông tin trong thế giới thực, là đối tượng tạo thành cơ sở dữ liệu và sau đó thể hiện nó



Chuẩn hoá

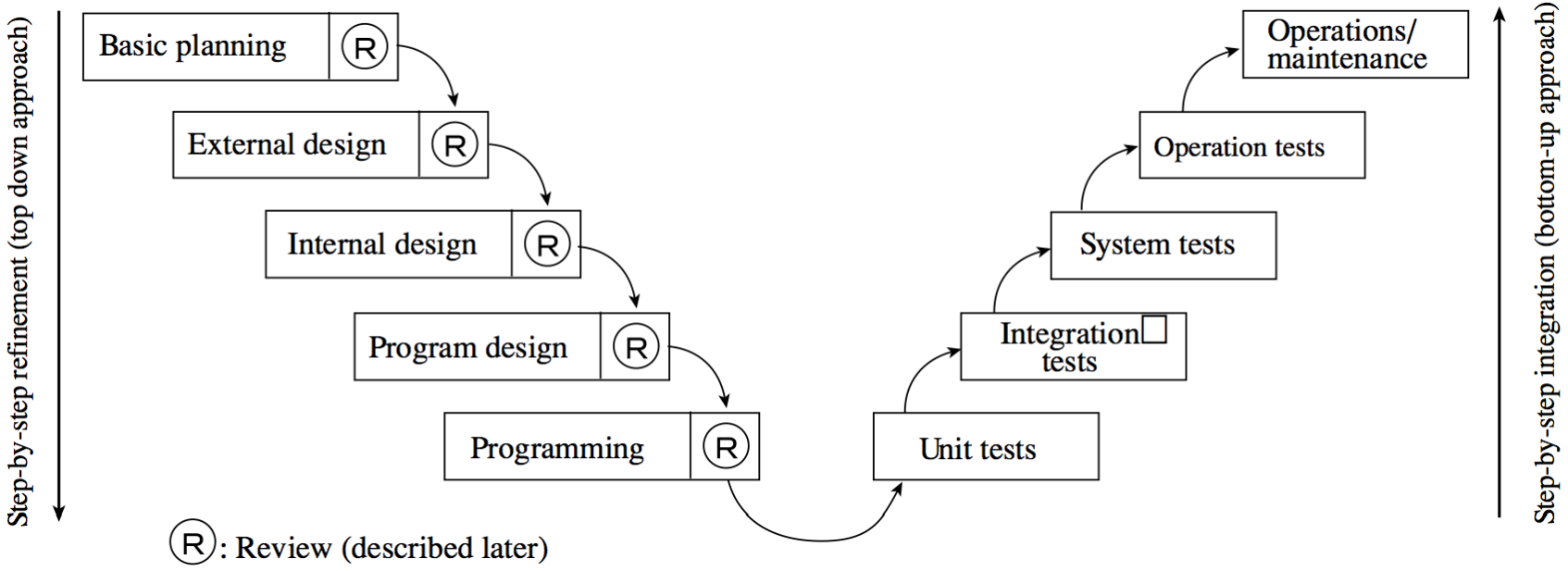
- Chuẩn hoá (Normalization): quy trình các bước sẽ xác định và loại bỏ các dư thừa trong thiết kế cơ sở dữ liệu.
- Mục đích của chuẩn hoá: để cải thiện
  - hiệu quả lưu trữ
  - toàn vẹn dữ liệu
  - và khả năng mở rộng

Chuẩn hoá (2)

- Trong mô hình quan hệ, tồn tại các phương pháp để định lượng mức độ hiệu quả của cơ sở dữ liệu.
- Các phân loại này được gọi là **các dạng chuẩn (normal forms hoặc NF)**, và có các thuật toán để chuyển đổi một cơ sở dữ liệu đã cho giữa chúng.
- Chuẩn hóa thường liên quan đến việc tách các bảng hiện có thành nhiều bảng, các bảng này phải được nối lại hoặc liên kết mỗi khi truy vấn được đưa ra.

Mô hình chữ V (V Model) – Các mức kiểm thử khác nhau

- Kiểm thử đơn vị (Unit test): riêng từng module một (ONE module at a time)
- Kiểm thử tích hợp (Integration test): liên kết các modules
- Kiểm thử hệ thống (System test): tổng thể (toàn bộ) hệ thống



Các kỹ thuật kiểm thử đơn vị

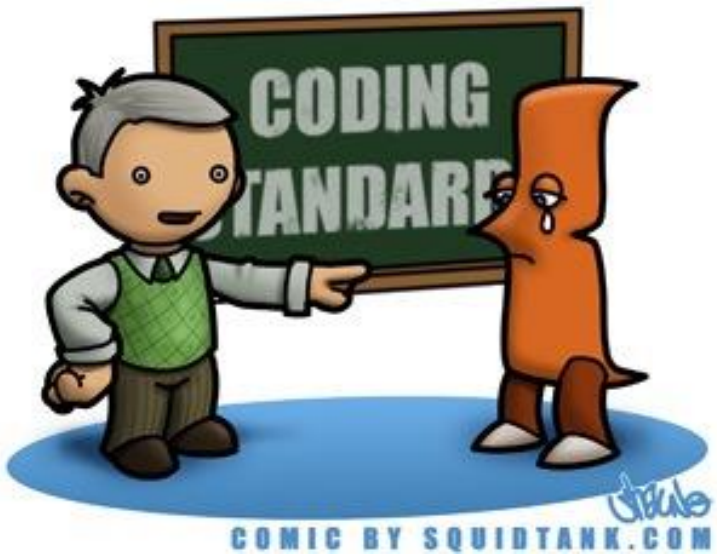
- Dành cho thiết kế trường hợp thử
- (2.2) Các kỹ thuật kiểm thử cho kiểm thử hộp đen (Black Box Test)
  - Phân tích phân vùng tương đương (Equivalence Partitioning Analysis)
  - Phân tích giá trị biên (Boundary-value Analysis)
  - Bảng quyết định (Decision Table)
  - Kiểm thử dựa trên ca sử dựng (Use Case-based Test)
- (2.3) Các kỹ thuật kiểm thử cho kiểm thử hộp trắng (White Box Test)
  - Kiểm thử luồng điều khiển với phủ C0, C1 (Control Flow Test with C0, C1 coverage)
  - Kiểm thử phủ biểu đồ tuần tự (Sequence chart coverage test)

Xây dựng chương trình

1. Phong cách lập trình
2. Tinh chỉnh / tối ưu mã (tuning / optimization)
3. Tái cấu trúc mã (refactoring)
4. Debugging

Programming style

- Coding convention/standard



Java Coding Conventions on One Page

William C. Wake (William.Wake@acm.org), 2-17-2000

General conventions: see http://java.sun.com/docs/codeconv/index.html

```
package com.some.display;

import java.util.*;

public class NounPhrase implements Interfaceable {

    public static final int MIN_SIZE = 22;

    static int height;

    String lastName;

    /** @deprecated Use getLastName(), not getName() */
    public String getName() {return lastName;}

    public int getScore() {
        if (height < MIN_SIZE) {
            return MIN_SIZE;
        }
        return height;
    }

    protected void paste(String filename, Vector v) throws IOException {
        Writer out;
        try {
            out = new FileWriter(filename);
            super.save(v);
            out.println(v.size());
        } catch (FileNotFoundException fnf) {
            Log.log(fnf);
            throw fnf;
        } finally {
            if (out != null) try {out.close();} catch(Exception ignored) {}
        }
    }
}
```

Specify a package (not default)

May use ".*"

Each word capitalized

Each word capitalized: noun phrase + optional "able" or "ing"

All caps with "_" separator

Constant

No special prefix for "static" variables

Noun phrase. Fields not "public".

Javadoc conventions: see http://java.sun.com/products/jdk/javadoc/writingdoccomments/index.html

One-liners are OK

JavaBeans naming conventions: see http://java.sun.com/beans/docs/beans.101.pdf

Braces here and here

4-space indent throughout

Exceptions either re-thrown or handled

"Finally" clause to release resources

Catch any exceptions in "finally" clause so original exception is reported

Code tuning

- Modifying correct code to make it run more efficiently
- Not the most effective/cheapest way to improve performance
- 20% of a program's methods consume 80% of its execution time.

What is Refactoring?

Refactoring means "to improve the design and quality of existing source code without changing its external behavior".

Martin Fowler

- A step by step process that turns the bad code into good code
  - Based on "refactoring patterns" → well-known recipes for improving the code

Refactoring: The Typical Process

1. Save the code you start with
   - Check-in or backup the current code
2. Prepare tests to assure the behavior after the code is refactored
   - Unit tests / characterization tests
3. Do refactoring one at a time
   - Keep refactoring small
   - Don't underestimate small changes
4. Run the tests and they should pass / else revert
5. Check-in (into the source control system)

Code Smells

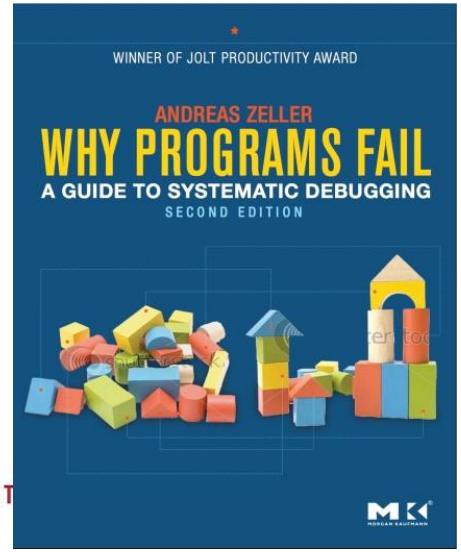
- Code smells == certain structures in the code that suggest the possibility of refactoring
- Types of code smells:
  - The bloaters
  - The obfuscators
  - Object-oriented abusers
  - Change preventers
  - Dispensables
  - The couplers

Rafactoring Patterns

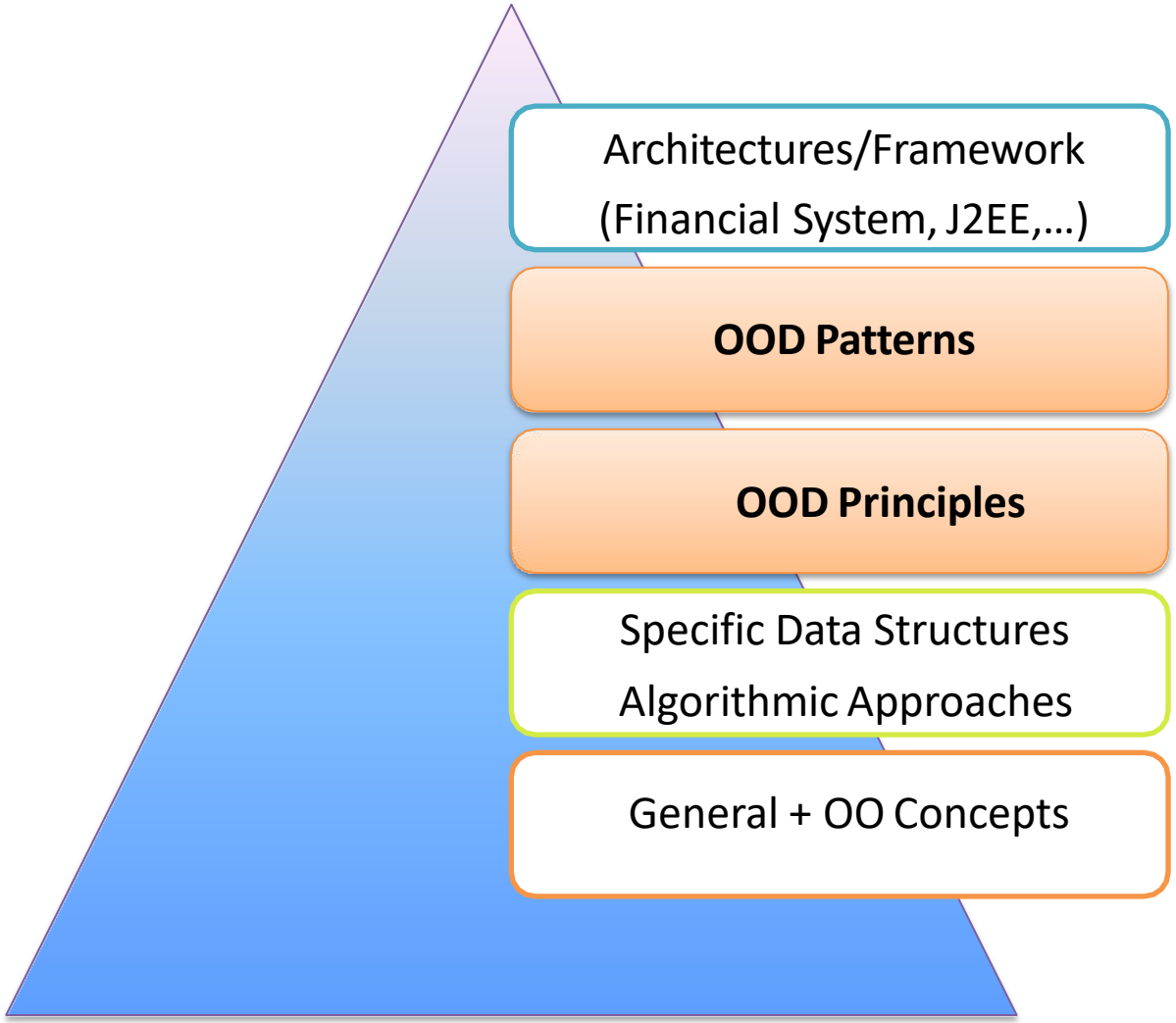
- When should we perform refactoring of the code?
  - Bad smells in the code indicate need of refactoring
- Unit tests guarantee that refactoring preserves the behavior
- Rafactoring patterns
  - Large repeating code fragments → extract duplicated code in separate method
  - Large methods → split them logically
  - Large loop body or deep nesting → extract method

Debugging and the scientific method

- Debugging should be systematic
  - Carefully decide what to do – flailing can be an instance of an epic fail
  - Keep a record of everything that you do
  - Don't get sucked into fruitless avenues
- Formulate a hypothesis
- Design an experiment
- Perform the experiment
- Adjust your hypothesis and continue



Các nguyên lý thiết kế



Key design concepts

General

- Cohesion
- Coupling
- Information hiding
  - Encapsulation
  - Creation
- Binding time

OO Specific

- Behaviors follow data
- Class vs. Interface Inheritance
  - Class = implementation
  - Interface = type
- Inheritance / composition / delegation

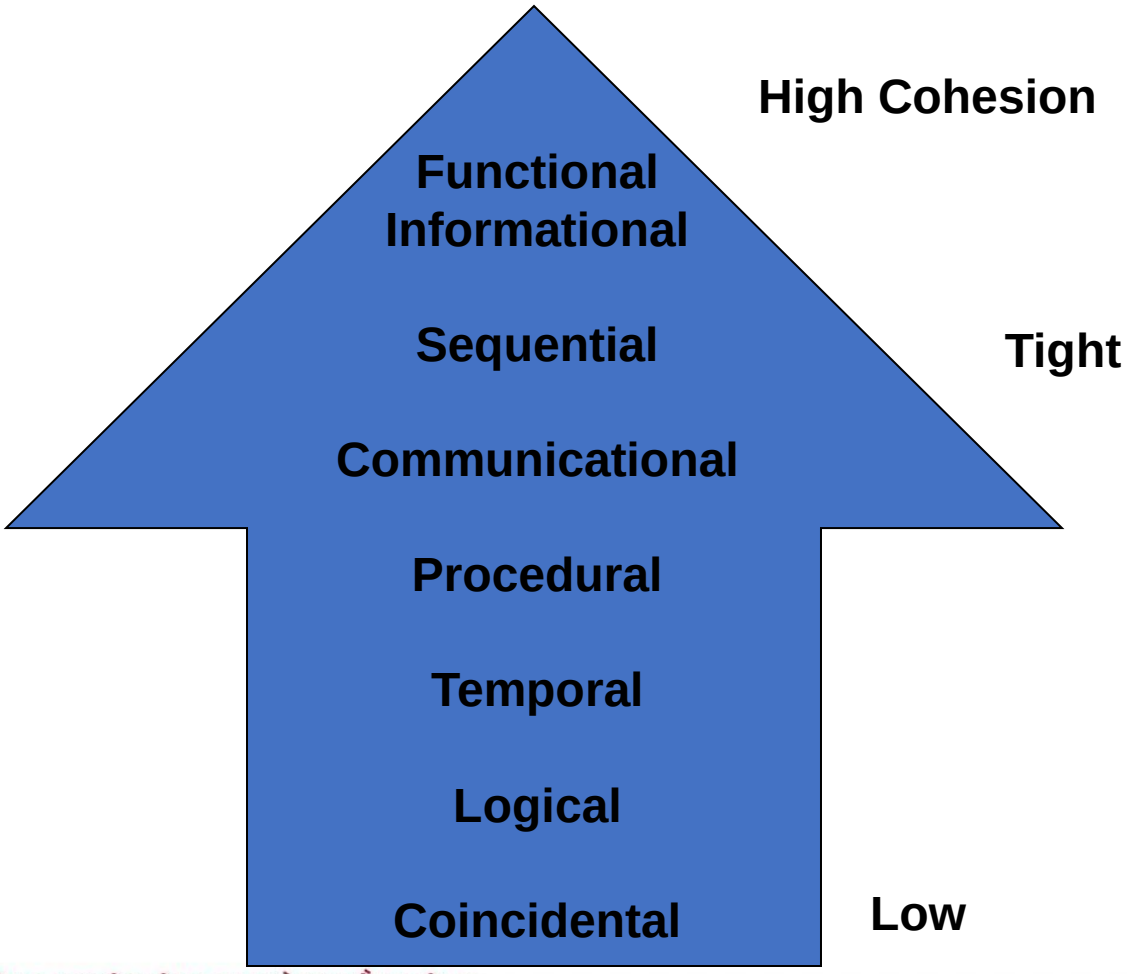
Good Design

- What's a design?
  - Express a idea to resolve a problem
  - Use for communications in the team members
- What's a good design?
  - Easy for Developing, Reading & Understanding
  - Easy for Communication
  - Easy for Extending (add new features)
  - Easy for Maintenance

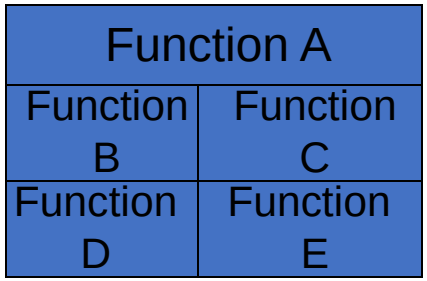
Two general design issues

- *Cohesion* – why are sub-modules (like methods) placed in the same module? Usually to collectively form an ADT
- *Coupling* – what is the dependence between modules? Reducing the dependences (which come in many forms) is desirable

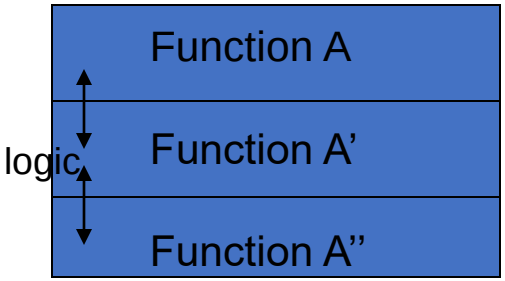
Range of Cohesion



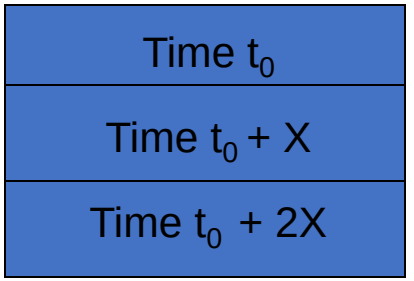
Examples of Cohesion



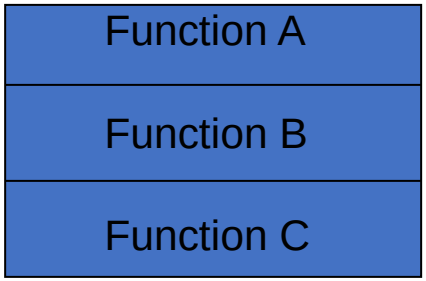
Coincidental
Parts unrelated



Logical
Similar functions

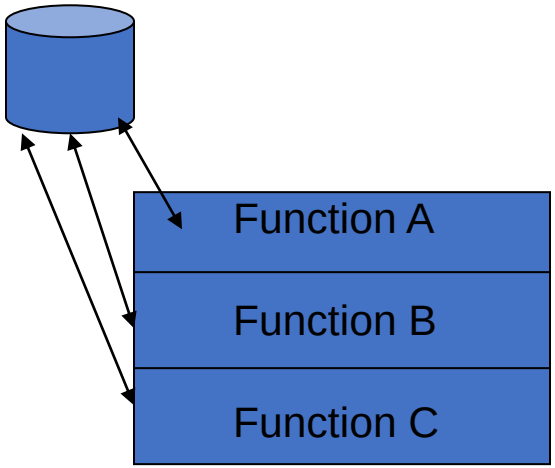


Temporal
Related by time

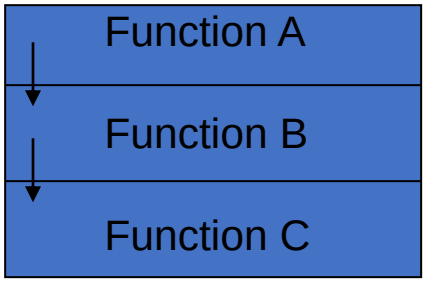


Procedural
Related by order of functions

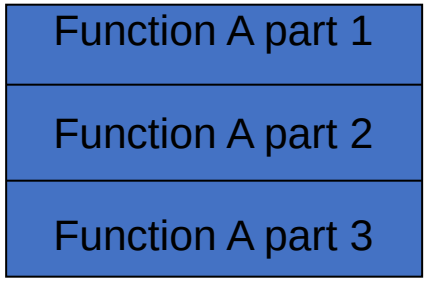
Examples of Cohesion-2



Communicational
Access same data

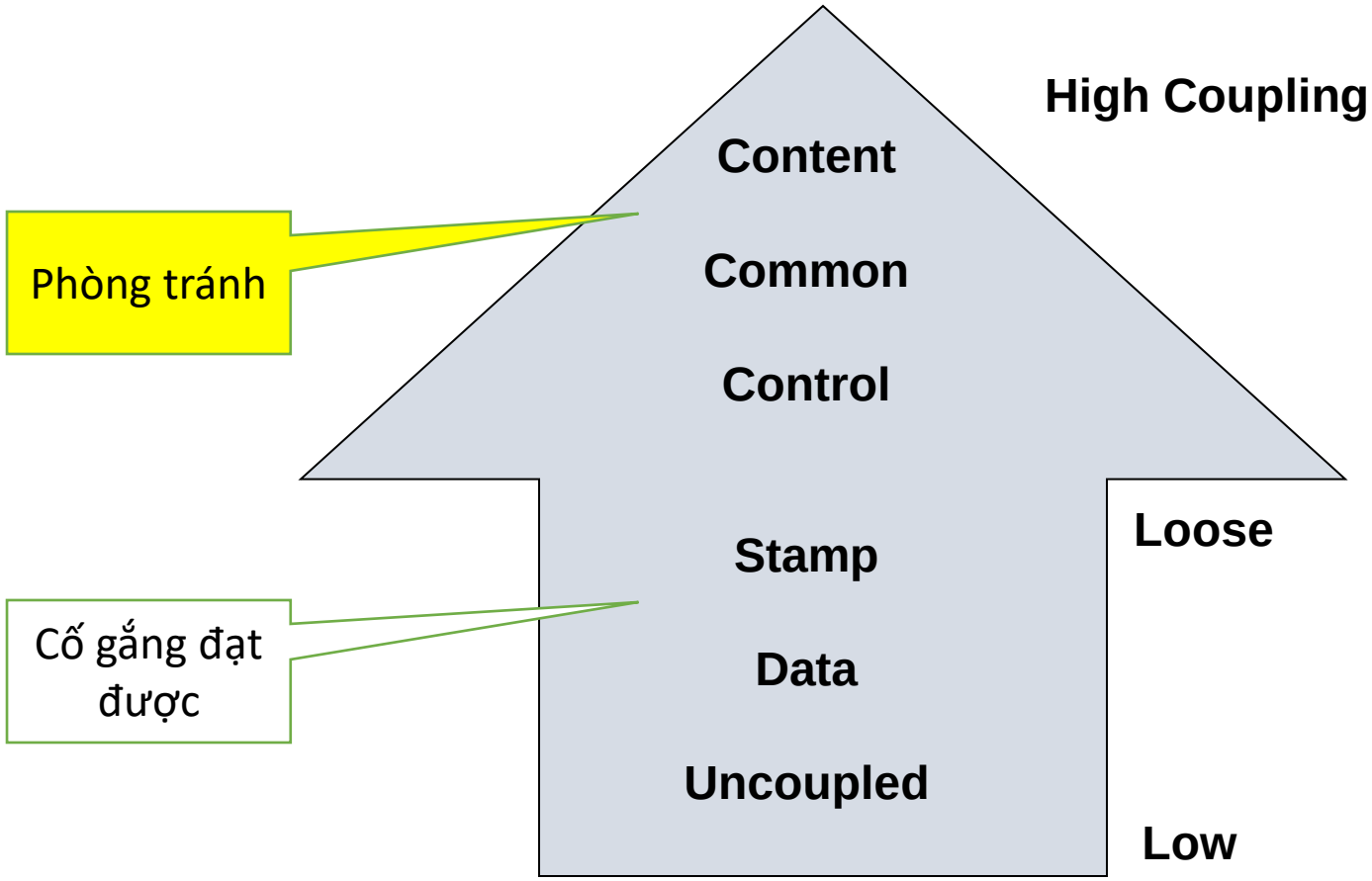


Sequential
Output of one is input to another



Functional
Sequential with complete, related functions

Range of Coupling



GRASP

Có 9 mẫu (nguyên lý) sử dụng trong GRASP

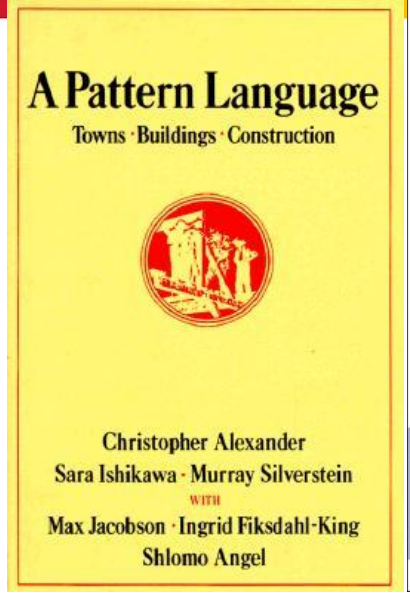
1. Information expert
2. Creator
3. Controller
4. Low coupling
5. High cohesion
6. Indirection
7. Polymorphism
8. Pure fabrication
9. Protected variations- Don’t Talk to Strangers

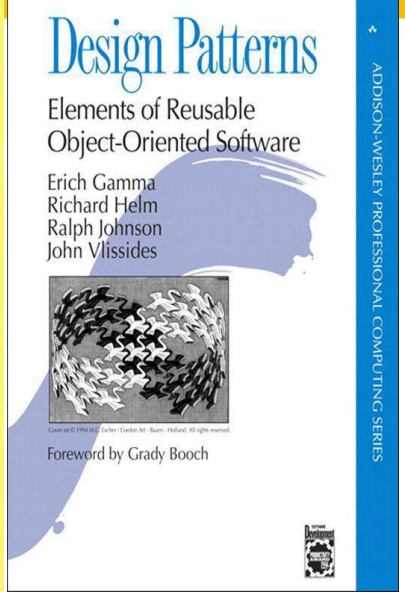
Craig Larman: *Apply UML and Patterns – An Introduction to Object-Oriented Analysis and Design*

S.O.L.I.D Principles of OOD

- SRP: The Single Responsibility Principle
- OCP: The Open Closed Principle
- LSP: The Liskov Substitution Principle
- ISP: The Interface Segregation Principle
- DIP: The Dependency Inversion Principle

Design Patterns





- Published in 1994
- "Each pattern describes a **problem** which occurs over and over **again** in our environment, and then describes the **core of the solution** to that problem, in such a way that you can use this solution **a million times over**, without ever doing it the same way twice"
  - Christopher Alexander
- Today's amazon.com stats

Amazon Best Sellers Rank: #2,069 in Books (See Top 100 in Books)
#1 in Books > Computers & Internet > Computer Science > Software Engineering > **Design Tools & Techniques**
#1 in Books > Computers & Internet > Programming > Software Design, Testing & Engineering > **Software Reuse**
#3 in Books > Nonfiction > Foreign Language Nonfiction > **French**

Đánh giá môn học

- Câu hỏi về các giai đoạn trong quy trình phát triển phần mềm, các mô hình và biểu đồ UML (biểu đồ uscase, trình tự / giao tiếp, hoạt động, trạng thái, biểu đồ lớp,…)
- Câu hỏi về nguyên lý thiết kế hướng đối tượng (mã nguồn hoặc thiết kế và phân tích các vấn đề theo các nguyên lý thiết kế hướng đối tượng)
- Câu hỏi case study: đề bài cho mô tả về case study sau đó thực hiện các bước phân tích, thiết kế,… theo các chủ đề trong môn học:
  - Xác định yêu cầu phần mềm (xác định tác nhân, usecase, xây dựng biểu đồ usecase, đặc tả usecase, ràng buộc đầu vào,…)
  - Phân tích và thiết kế phần mềm (các sơ đồ UML: biểu đồ trình tự, giao tiếp, hoạt động), xây dựng và mô tả biểu đồ lớp thiết kế, thiết kế dữ liệu, thiết kế giao diện màn hình.
  - Kiểm thử (hộp đen và hộp trắng)
  - Nguyên lý thiết kế hướng đối tượng đảm bảo cho các yêu cầu về khả năng sửa đổi, mở rộng thêm chức năng,…
  - …

Bài tập 1

- Cho 3 lớp Bird, KingFisher, và Ostrich như sau:

```
public class Bird {
   public void fly() {
      System.out.println("I'm flying");
   }
   public void walk() {
      System.out.println("I'm
walking");
   }
}
```

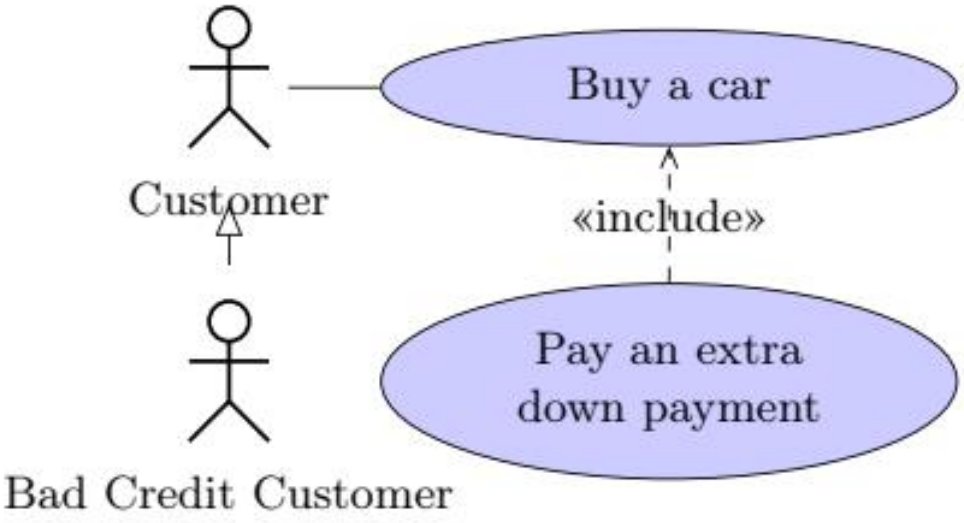
- Theo bạn mã nguồn như trên có vấn đề gì không, giải thích ngắn gọn và đưa ra giải pháp khắc phục?

```
public class KingFisher extends Bird {
   @Override
   public void fly() {
      System.out.println("I'm flying as a KingFisher");
   }
   @Override
   public void walk() {
      System.out.println("I'm walking as a KingFisher");
   }
}

public class Ostrich extends Bird {
   @Override
   public void fly() {
      System.out.println("Error!");
   }
   @Override
   public void walk() {
      System.out.println("I'm walking as a Ostrich");
      ;
   }
}
```

Bài tập 2

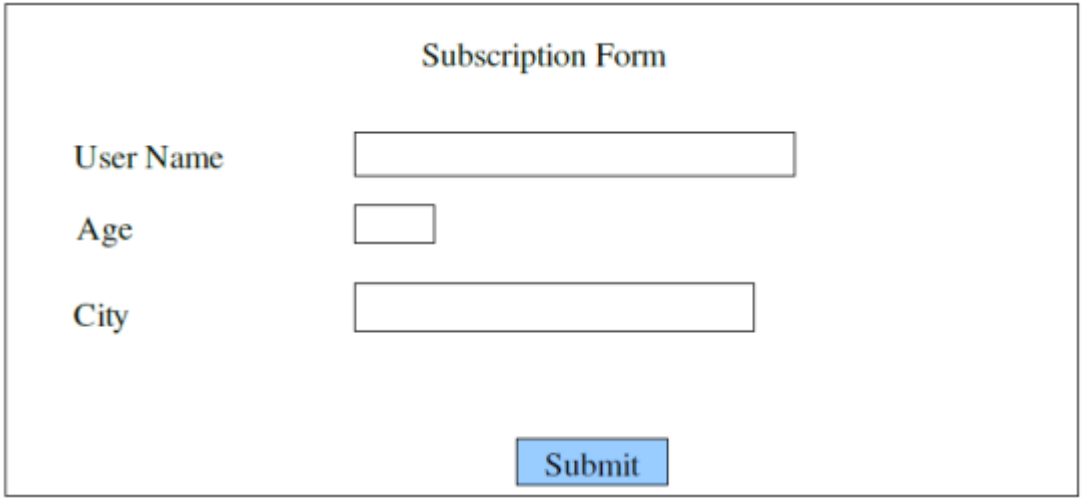
- Mô tả về một tình huống trong ứng dụng như sau: “Customers of the garage can buy cars. Customers with a bad credit should pay an extra down payment”.
- Một nhà phát triển đã mô hình hóa tình huống này bằng biểu đồ Usecase như sau:



- Hãy cho biết biểu đồ Usecase này có phù hợp với tình huống đã cho không, cần điều chỉnh lại như thế nào?

Bài tập 3

❖ Cho giao diện một chức năng như hình sau:



❖ Các ràng buộc đầu vào:

- Tên người dùng phải dài từ 6 đến 12 ký tự.
- Tuổi phải là một số nguyên lớn hơn hoặc bằng 18 và nhỏ hơn hoặc bằng 65.
- Thành phố phải là một trong “Ha Noi”, “Da Nang”, “TP Ho Chi Minh” hoặc “Khác”

❖ Hệ thống sẽ hiển thị thông báo thành công nếu tất cả các trường được nhập chính xác hoặc thông báo lỗi nếu các ràng buộc đầu vào không được thỏa mãn.

a) Liệt kê các lớp tương đương (hợp lệ và không hợp lệ) cho mỗi điều kiện đầu vào.

b) Đề xuất các giá trị biên cho mỗi điều kiện đầu vào.

c) Thiết kế các trường hợp kiểm thử (test cases) bao gồm tất cả các lớp tương đương. Sử dụng các giá trị biên càng nhiều càng tốt.

Bài tập 4

- Vẽ một sơ đồ lớp biểu diễn cho phát biểu sau: Một cuốn sách (book) bao gồm một số phần (part), các phần này lại bao gồm một số chương (chapter), các chương lại bao gồm các phần nhỏ (section).
  - Một cuốn sách (book) bao gồm thông tin về publisher, publication date, và ISBN
  - Một part có thông tin về title và number
  - Một chapter bao gồm title, number, và abstract
  - Một section có thông tin về title và number

Bài tập 5

- Thực hiện refactor đoạn mã nguồn dưới đây:

```
public void checkoutBook(Book b) {
    checkout.add(b);
    Library lib = b.getLibrary();
    lib.notifyChange(b);
}

public void returnBook(Book b) {
    checkout.remove(b);
    Library lib = b.getLibrary();
    lib.notifyChange(b);
}
```

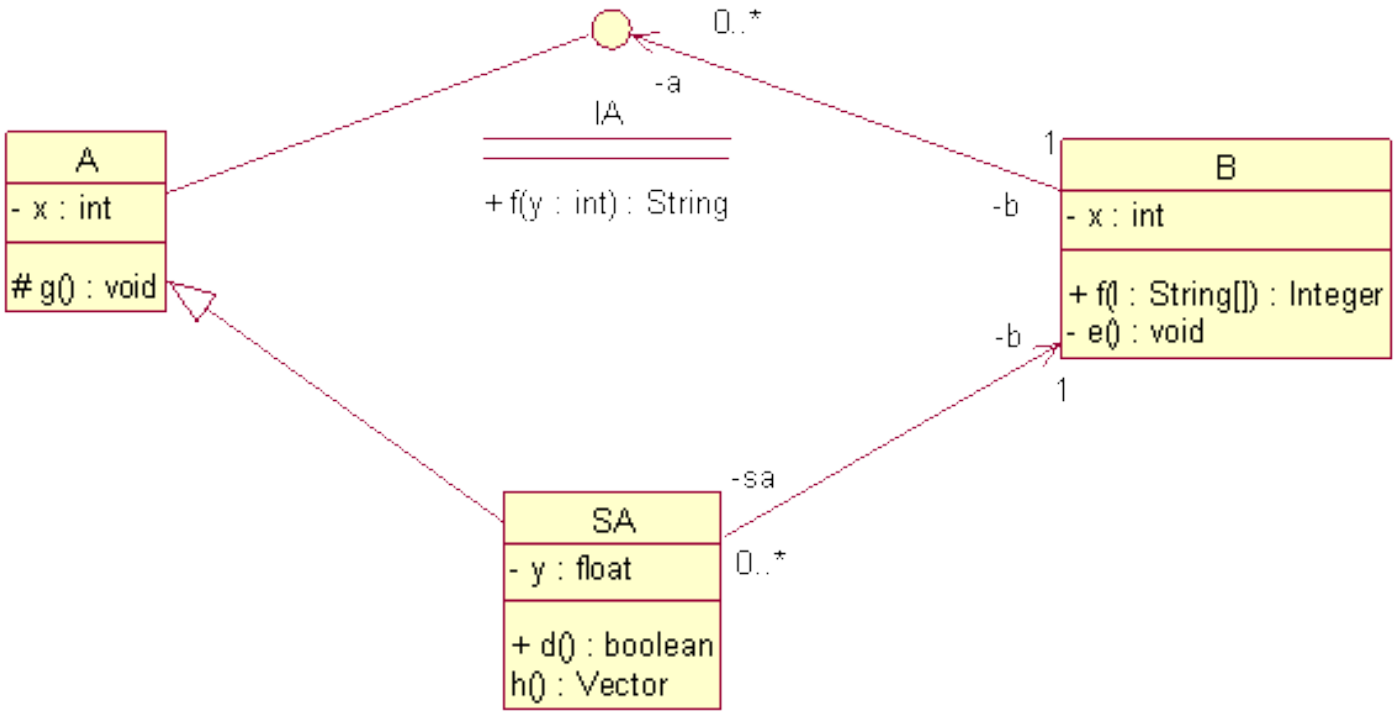
Bài tập 6

- Xác định các phát biểu dưới đây là yêu cầu chức năng (functional requirements) hay yêu cầu phi chức năng (nonfunctional requirements):

1. “The ticket distributor must enable a traveler to buy weekly passes.”
2. “The ticket distributor must be written in Java.”
3. “The ticket distributor must be easy to use.”

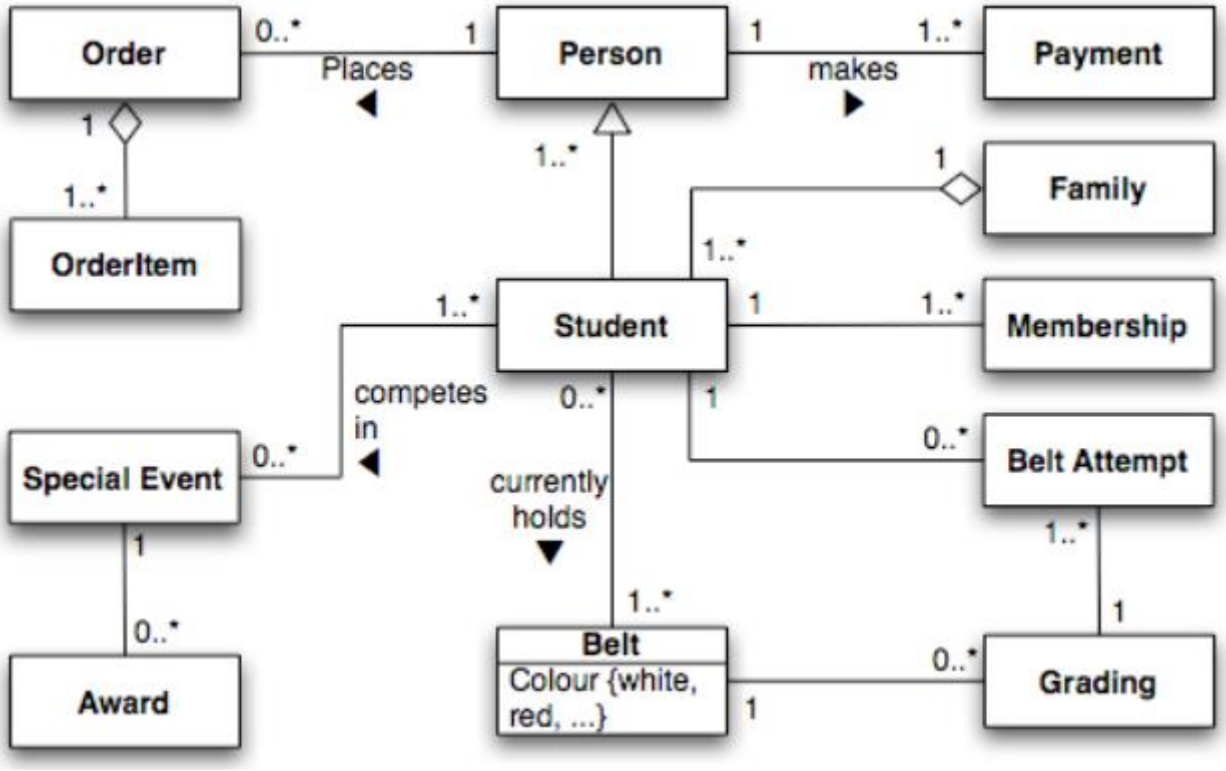
Bài tập 7

- Viết khung mã nguồn Java tương ứng với biểu đồ giao tiếp sau:



Bài tập 8

- Mô hình miền nghiệp vụ (domain model) sau sau nắm bắt một số khái niệm cơ bản về Hệ thống quản lý câu lạc bộ karate dành cho trẻ em. Mô hình gồm các class và quan hệ nhằm mô hình hóa cho các thông tin chính của hệ thống.



- Chủ câu lạc bộ muốn giảm giá theo gia đình, để nếu có nhiều hơn một học sinh trong cùng một gia đình là thành viên của câu lạc bộ, thì mỗi người sẽ được giảm 10%. Làm thế nào bạn sẽ thay đổi mô hình để đáp ứng yêu cầu này?

Bài tập 9

- Một trình phát nhạc kỹ thuật số được mô tả gồm các thông tin sau: Một nghệ sĩ là một ban nhạc hoặc một nhạc sĩ, trong đó một ban nhạc bao gồm hai hoặc nhiều nhạc sĩ. Mỗi bài hát có một nghệ sĩ đã viết nó, một nghệ sĩ đã biểu diễn nó và một tiêu đề. Giả sử "bài hát" có nghĩa là bản ghi âm của một bản nhạc, vì vậy nếu một bản nhạc được ghi lại nhiều lần (chẳng hạn, bởi các nghệ sĩ khác nhau), thì chúng ta coi chúng là các bài hát khác nhau. Do đó, mỗi bài hát được thể hiện bởi một nghệ sĩ và được viết bởi một nghệ sĩ. Một album bao gồm một số bản nhạc, mỗi bản tương ứng với một bài hát. Một bài hát có thể được sử dụng trong bất kỳ số lượng bản nhạc nào, bởi vì nó có thể xuất hiện trên nhiều album (hoặc thậm chí nhiều hơn một lần trong cùng một album!). Bản nhạc có tốc độ bit và thời lượng. Bởi vì thứ tự của các bản nhạc trong một album rất quan trọng nên hệ thống sẽ cần biết, đối với bất kỳ bản nhạc cụ thể nào, bản nhạc tiếp theo là gì và bản nhạc trước đó là gì (nếu có).
- Vẽ một biểu đồ lớp mô hình hóa các thông tin này và đảm bảo gắn nhãn tất cả các liên kết với các bội số thích hợp.

Bài tập 10

- Phương thức sau đây được xây dựng nhằm hoán đổi các cặp phần tử trong mảng số nguyên là đối số của nó. Bắt đầu bằng arr[0] và arr[1], nếu số phần tử của mảng là số lẻ thì phần tử cuối cùng không bị thay đổi. Thật không may, có ít nhất một bug trong đoạn mã này.
- Xây dựng hai testcase khác nhau cho phương thức này. Một testcase sẽ tiết lộ lỗi. Testcase khác sẽ thực hiện thành công.

```
public void swapPairs(int[] arr) {
    int i = 0;
    while (i != arr.length) {
            int temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;
            i = i + 2;
    }
}
```

Bonus

- Với mỗi phát biểu dưới đây hãy xác định mức cohesion và coupling?

	coupling		cohesion		type of cohesion or coupling
	low	high	low	high	
Module A contains two functions that often follow each other.					
Module A always executes after module B.					
Module A updates the same data as module B, but for different reasons.					
Only module A contains the statements that change the entity status.					