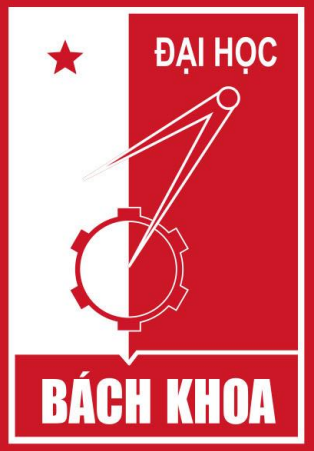


ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Nhập môn Công nghệ Phần mềm

(Introduction to Software Engineering)

CHƯƠNG 8

Xây dựng phần mềm
Software Construction

Nội dung

1. Khái niệm
2. Quy trình xây dựng phần mềm
3. Quy ước viết mã nguồn
4. Tái cấu trúc mã nguồn
5. Rà soát mã nguồn

1. Khái niệm

- **Coding**: viết mã nguồn
  - Chuyển đổi **thiết kế chi tiết** của một hệ thống thành mã nguồn
  - Việc chuyển đổi mã này sẽ phải được thực hiện theo các cấp độ từng unit/modul/component riêng lẻ, sau đó các thành phần riêng lẻ này phải được tích hợp lại (unit/modul/component integration)
- **Documenting**: biên soạn tài liệu đi kèm với mã nguồn
  - giúp xác minh sự phù hợp giữa mã nguồn với bản đặc tả của unit/modul/component, đảm bảo chất lượng của mã nguồn
  - giúp chia sẻ các công việc có thể bị lặp lại giữa các project
- **Unit Testing**: kiểm thử đơn vị phải được thực hiện ở giai đoạn này cho từng unit/module/component một cách độc lập

Mục tiêu của việc xây dựng phần mềm

1. Thực hiện các tác vụ theo chỉ định của thiết kế.

2. Để giảm chi phí của các giai đoạn: kiểm tra và bảo trì (mã hóa hiệu quả).

3. Làm cho mã nguồn dễ đọc, dễ hiểu hơn: Việc mã hóa cần đảm bảo mục tiêu làm tăng khả năng hiểu mã và đọc mã trong quá trình tạo ra phần mềm dễ bảo trì.

Từ chương trình tới hệ thống
From programs to systems

- Chương trình = (cấu trúc dữ liệu + thuật toán)
- Hệ thống / Phần mềm:
  - Cấu trúc dữ liệu, giải thuật và tài liệu đặc tả
- Hệ thống có thể bao gồm nhiều chương trình, module, thành phần có kết nối hoặc tương tác với nhau
- Quy mô của hệ thống lớn hơn chương trình
- Thường được xây dựng sử dụng các thư viện, các framework có sẵn để tận dụng được nhiều thành phần xây dựng sẵn

Mô thức lập trình
Programming paradigm

- Imperative paradigm
- Logical paradigm
- Functional paradigm
- Object-oriented paradigm
- Visual paradigm
- Parallel paradigm
- Concurrent paradigm
- Distributed paradigm
- Service-oriented paradigm

Mô thức LT hướng chức năng Functional Programming

- Nguồn gốc: lý thuyết hàm số
- Ngôn ngữ lập trình hướng chức năng miêu tả
  - Tập hợp các kiểu dữ liệu có cấu trúc
  - Tập hợp các hàm định nghĩa trên các kiểu dữ liệu đó
- Thành phần:
  - Tập hợp các cấu trúc dữ liệu và các hàm có liên quan
  - Tập hợp các hàm cơ sở
  - Tập hợp các toán tử

Mô thức LT hướng chức năng Functional Programming

- Đặc trưng cơ bản: tính module hoá chương trình - Modularity
  - Chương trình là tập hợp của các hàm (function)
  - Tính toán trong chương trình được thực hiện bằng các lời gọi hàm
  - Các chức năng khác nhau trong chương trình có thể gọi chéo hàm lẫn nhau (function sharing)
  - Khi xây dựng chương trình theo mô thức lập trình hướng chức năng, thông thường lập trình viên sẽ tiếp cận vấn đề theo hướng top-down

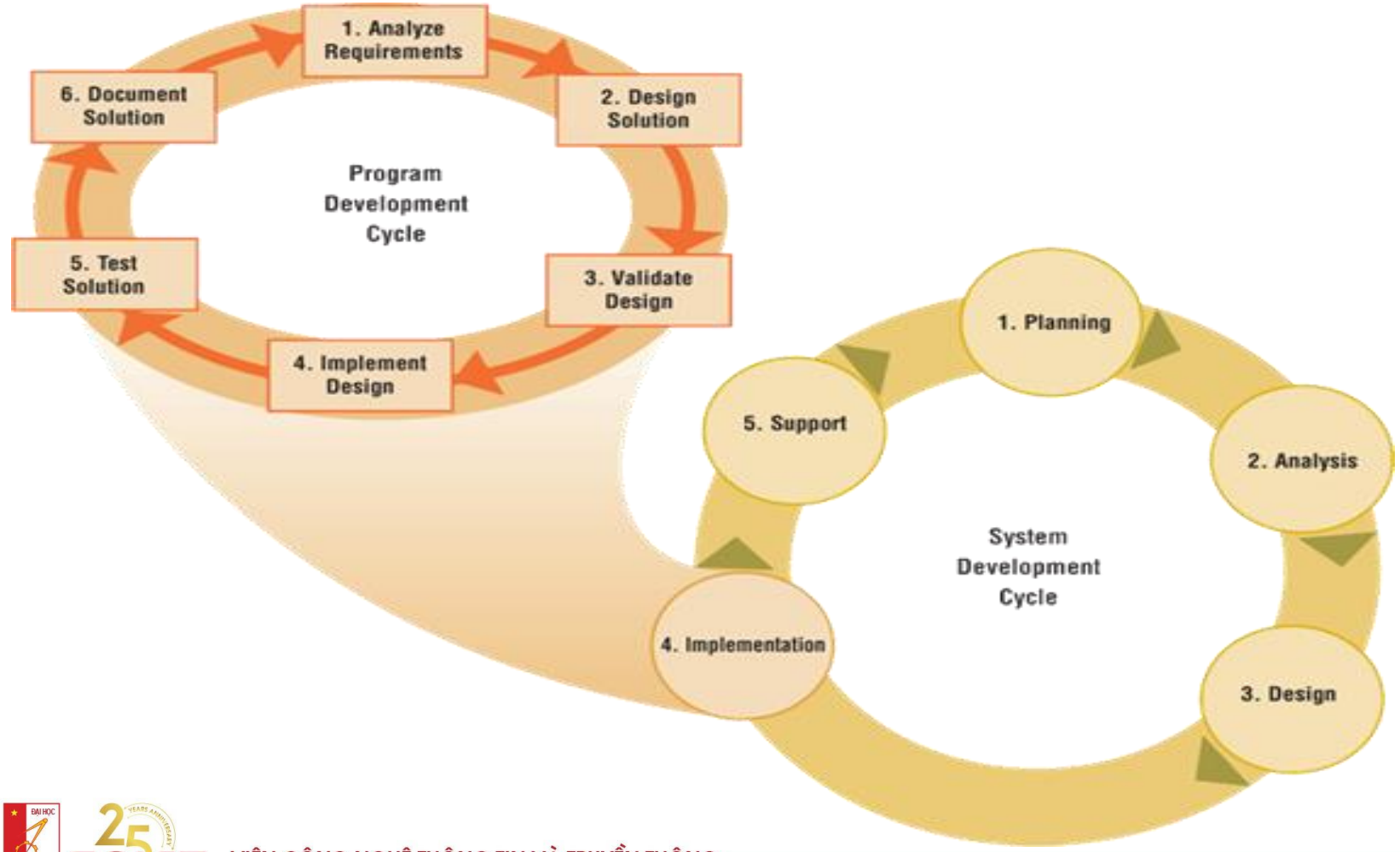
Mô thức LT hướng đối tượng
Object Oriented Programming

- History:
  - Simulation – Simula 67 là ngôn ngữ lập trình OOP đầu tiên
  - Interactive graphics – Smalltalk-76, lấy ý tưởng từ Simula
- Đặc trưng của mô thức lập trình HĐT:
  - Tập trung vào các khái niệm (concepts) thay vì các thao tác, hàm chức năng(operations/functions)
  - Mỗi khái niệm có thể biểu diễn một đối tượng dữ liệu trong chương trình (Book, Category, Product etc.)

Mô thức LT hướng đối tượng Object Oriented Programming

- Một số ưu điểm của OOP so với các ngôn ngữ lập trình cấu trúc
  - Dễ dàng tái sử dụng các khái niệm trong các chương trình khác
    - Ví dụ Book trong chương trình quản lý thư viện và Book trong BookStore có thể được tái sử dụng dễ dàng
  - Dễ dàng mở rộng các khái niệm cũ (old concepts) thành các khái niệm mới hơn cho các chương trình khác
    - Ví dụ: MusicPlayer → MP3Player, WinampPlayer, DVDPlayer etc.
  - Dễ dàng hơn trong việc định vị các khu vực thay đổi khi các đối tượng dữ liệu liên quan có thay đổi (changes are localized in some code base of changed concepts)

2. Quy trình xây dựng phần mềm



Bước 1: Phân tích yêu cầu
Analyse requirements

Phân tích hệ thống

- *Dựa trên các hệ thống có thực (do con người vận hành hoặc hệ thống tự động)*
- *Do các nhà phân tích hệ thống tiến hành, sẽ hiệu quả hơn nếu phỏng vấn người dùng*

Mục tiêu

- Xác định xem hệ thống hiện tại đã làm được những gì, làm như thế nào, còn tồn tại các vấn đề gì

→ Quyết định xem có nên thực hiện bước tiếp theo hay không (Return-on-Investment – ROI estimation)

Bước 1: Phân tích yêu cầu

Analyse requirements

Các công việc chính

- Thiết lập các requirements
- Gặp các nhà phân tích hệ thống và users
- Xác định input, output, processing, và các thành phần dữ liệu

IPO CHART

Input	Processing	Output
Regular Time Hours Worked	Read regular time hours worked, overtime hours worked, hourly pay rate.	Gross Pay
Overtime Hours Worked	Calculate regular time pay.	
Hourly Pay Rate	If employee worked overtime, calculate overtime pay.	
	Calculate gross pay.	
	Print gross pay.	

Ví dụ
IPO chart module Register for Course

- Register for Course:
  - Sinh viên mỗi đầu kì học sẽ phải đăng kí các khoá học
  - Module sẽ căn cứ vào học kỳ mấy, mã số sinh viên, lịch sử học tập của sinh viên và danh sách các khoá học của học kỳ mới để có thể output ra một danh sách các môn học sinh viên có thể đăng ký
- Yêu cầu: xây dựng IPO chart cho module

Bước 2: Thiết kế giải pháp
Design solution

Object-oriented design, thường thực hiện theo bottom-up

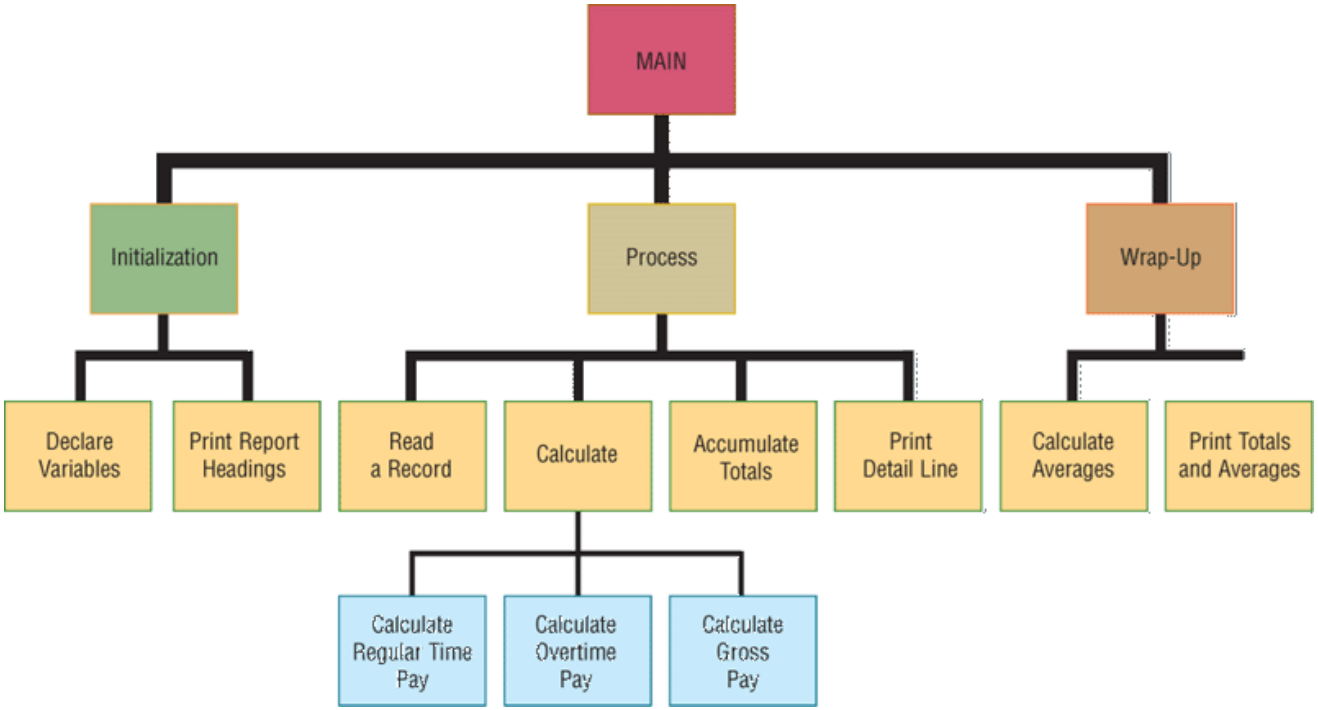
Structured design, còn gọi là top-down design

LTV bắt đầu với thiết kế tổng thể rồi đi đến thiết kế chi tiết

Bước 2: Thiết kế giải pháp
Design solution

Thiết kế Sơ đồ phân cấp chức năng (hierarchy chart)

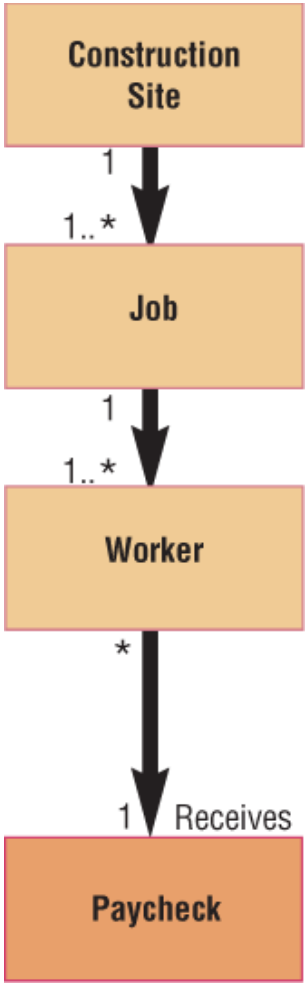
- Còn gọi là sơ đồ cấu trúc
- Trực quan hóa các module chương trình



Bước 2: Thiết kế giải pháp
Design solution

Thiết kế hướng đối tượng

- LTV đóng gói dữ liệu và các thủ tục xử lý dữ liệu trong đối tượng (object)
- Các đối tượng được phân loại thành các lớp (classes)
- Thiết kế các biểu đồ lớp thể hiện trực quan các quan hệ phân cấp quan hệ của các lớp



Bước 2: Thiết kế giải pháp
Design solution

Thiết kế giải thuật

- Máy tính không thể tự nghĩ ra hay tự quyết định một sơ đồ hoạt động
- Máy tính chỉ có thể làm chính xác những gì được yêu cầu, theo cách được yêu cầu, chứ không phải làm những gì con người muốn máy tính làm
- Giải thuật là một tập các chỉ thị miêu tả cho máy tính nhiệm vụ cần làm và thứ tự thực hiện các nhiệm vụ đó.

Bước 3: Chứng thực thiết kế
Validate design

Kiểm tra độ chính xác của chương trình

LTV kiểm tra tính đúng đắn bằng cách tìm các lỗi logic (Logic Error)

Desk check
LTV dùng các dữ liệu thử nghiệm (Test data) để kiểm tra chương trình

Logic error
các sai sót khi thiết kế gây ra những kết quả không chính xác

Test data
các dữ liệu thử nghiệm giống như số liệu thực mà chương trình sẽ thực hiện

Structured Walkthrough
LTV mô tả logic của thuật toán trong khi đội lập trình duyệt theo logic chương trình

Bước 4: Cài đặt thiết kế

Implement design

- Viết mã nguồn chương trình (coding): dịch từ thiết kế thành chương trình
  - Cú pháp (Syntax): Quy tắc xác định cách viết các lệnh
  - Chú thích (Comments): tài liệu chương trình (tài liệu trong)
- Lập trình nhanh (Extreme programming - XP): viết mã nguồn và kiểm thử ngay sau khi các yêu cầu được xác định

Bước 5: Kiểm thử giải pháp
Test solution

Đảm bảo chương trình chạy thông và cho kết quả chính xác

Debugging - Tìm và sửa các lỗi syntax và logic errors

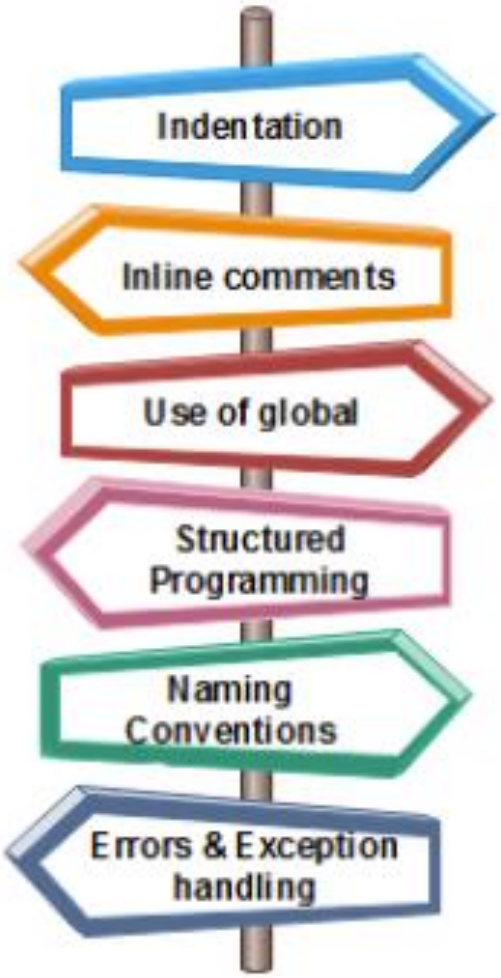
Kiểm tra phiên bản beta, giao cho Users dùng thử và thu thập phản hồi

Bước 6: Viết tài liệu cho giải pháp
Document solution

Rà soát lại program code - loại bỏ các **dead code**, tức các lệnh mà chương trình không bao giờ gọi đến

Rà soát, hoàn thiện documentation

3. Quy ước viết mã nguồn



Tại sao cần có phong cách lập trình tốt

- Lỗi logic đôi khi đến từ việc nhầm lẫn tên biến, nhầm lẫn mục đích của hàm do cách đặt tên...
- Mã nguồn tốt là mã nguồn dễ đọc
  - ***Cấu trúc chương trình rõ ràng, dễ hiểu, khúc triết***
  - ***Sử dụng thành ngữ phổ biến***
  - ***Chọn tên phù hợp, gợi nhớ***
  - ***Viết chú thích rõ ràng***

Một số quy tắc cơ bản

Nhất quán

- Tuân thủ quy tắc đặt tên trong toàn bộ chương trình
- Nhất quán trong việc dùng các biến cục bộ.

Khúc triết

- Mỗi function/method phải có một nhiệm vụ rõ ràng.
- Đủ ngắn để có thể nắm bắt được
- Số tham số của chương trình con là tối thiểu (dưới 6)

Một số quy tắc cơ bản

Rõ ràng

- Chú thích rõ ràng, vd. đầu mỗi chương trình con

Bao đóng

- Hàm chỉ nên tác động tới duy nhất 1 giá trị - giá trị trả về của hàm
- Không nên thay đổi giá trị của biến chạy trong thân của vòng lặp, ví dụ

```
for(i=1;i<=10;i++) i++;
```

Khoảng trắng – Cách lề
Spacing - Indentation

- Sử dụng khoảng trắng dễ đọc và nhất quán
- Cách lề hợp lý để tránh nhầm lẫn cấu trúc các khối lệnh
- Cách đoạn (paragraph) để tạo các khối lệnh thực hiện các mục đích khác nhau giúp dễ đọc dễ theo dõi

Đặt tên
Naming

- Dùng tên gợi nhớ, có tính miêu tả cho các biến và hàm
  - VD : hovaten , **`CONTROL`**, **`CAPACITY`**
- Dùng tên nhất quán cho các biến cục bộ
  - VD : **`i`** (not **`arrayIndex`**) cho biến chạy vòng lặp
- Dùng chữ hoa, chữ thường nhất quán
  - VD : Buffer_Insert (Tên hàm)
    CAPACITY (hằng số)
    buf (biến cục bộ)
- Dùng phong cách nhất quánkhi ghép từ
  - VD : **`frontsize`**, **`frontSize`**, **`front_size`**

Đặt tên
Naming

- Tên của các biến, các class thường là danh từ
- Tên class viết hoa chữ cái đầu
- Tên các phương thức, hàm thường là các động từ, chữ cái đầu viết thường

Chú thích
Comments

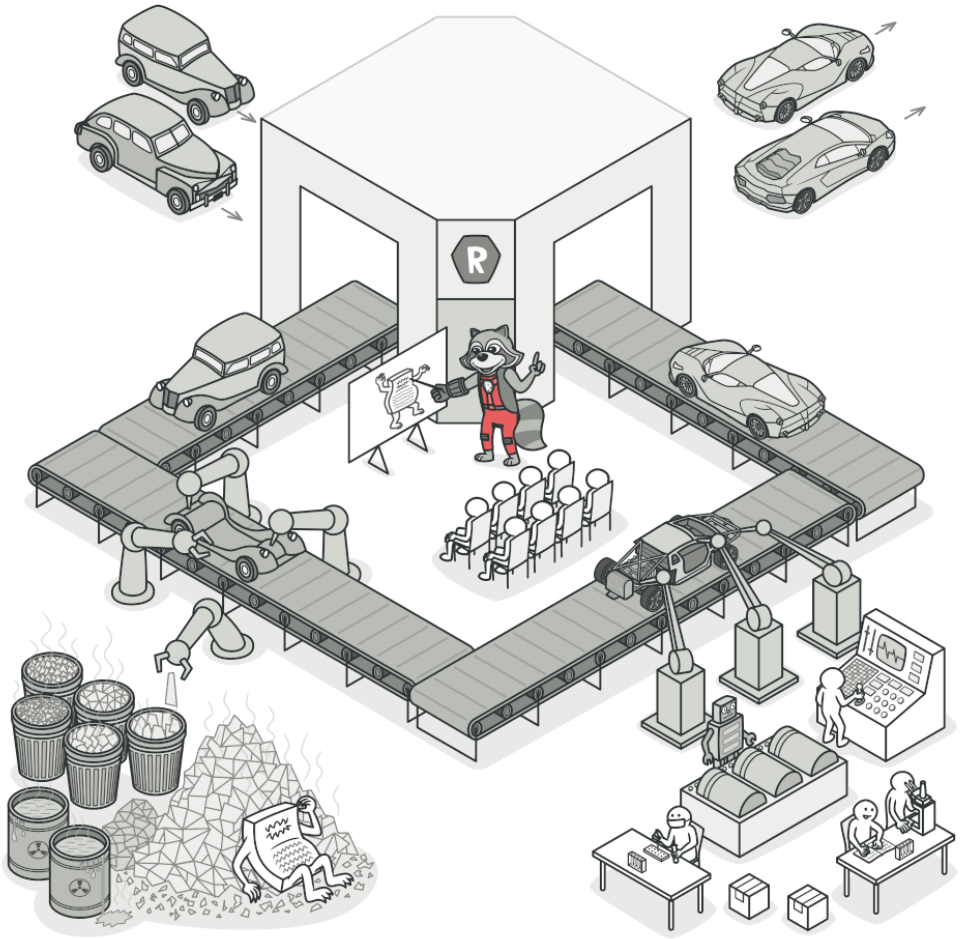
- Chú thích cho từng hàm, mô tả mục đích của hàm, inputs, outputs
- Chú thích cho từng khối lệnh, không nên chú thích cho từng dòng lệnh
- Một số framework hỗ trợ generate tự động tài liệu mã nguồn (e.g., JAVADoc)

4. Tái cấu trúc mã nguồn
Code Restructuring / Refactoring

Refactoring Code: là một quá trình cải thiện mã nguồn mà không cần viết thêm các chức năng mới, làm cho mã nguồn sạch hơn (clean code) và chuyển đổi thiết kế phức tạp khó hiểu thành đơn giản

Source: https://refactoring.guru/refactoring

4. Tái cấu trúc mã nguồn
Code Restructuring / Refactoring

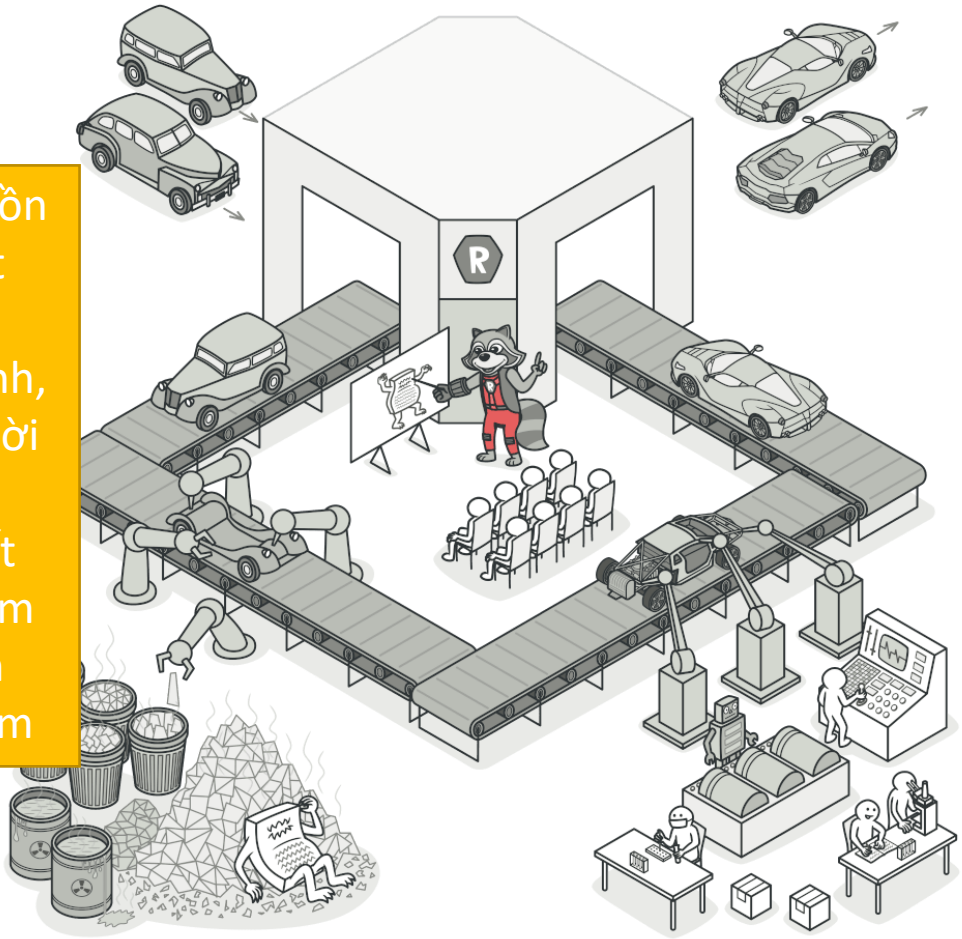


Clean Code: Mã sạch là mã dễ đọc, dễ hiểu, dễ bảo trì. Làm sạch mã giúp quá trình phát triển phần mềm dễ dự đoán được và làm tăng chất lượng sản phẩm phần mềm

Source: https://refactoring.guru/refactoring

4. Tái cấu trúc mã nguồn
Code Restructuring / Refactoring

Dirty Code: Mã nguồn "bẩn" thường là kết quả của việc thiếu kinh nghiệm lập trình, quản lý yếu kém, thời gian thực hiện quá ngắn hoặc các quyết định có tính chất tạm thời trong quá trình phát triển phần mềm



Source: https://refactoring.guru/refactoring

4. Tái cấu trúc mã nguồn
Code Restructuring / Refactoring

Quy trình tái cấu trúc – Refactoring Process: Thực hiện từng bước và luôn chạy lại các tests mỗi lần thực hiện một thay đổi

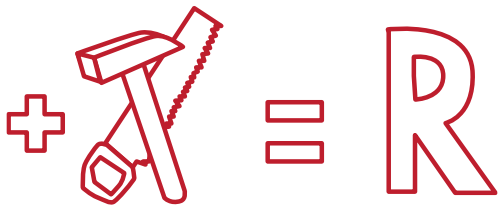
Source: https://refactoring.guru/refactoring

Mục tiêu của tái cấu trúc
Refactoring's goal

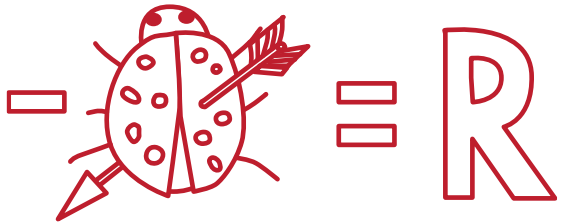
- Mục tiêu của tái cấu trúc mã nguồn là để tạo ra mã nguồn sạch (clean code), loại bỏ các lỗi tiềm ẩn, và làm cho thiết kế đơn giản hơn
- Mã nguồn trước khi đem tái cấu trúc là mã nguồn đã chạy, đã được test và đảm bảo các chức năng của chương trình
- Việc tái cấu trúc mã nguồn không làm thay đổi các chức năng của chương trình mà chỉ giúp cho mã nguồn sạch hơn và dễ đọc, dễ hiểu, dễ bảo trì

Khi nào cần thực hiện?
When to refactor?

- When adding a feature
  - Khi cần thêm 1 tính năng mới vào một code sẵn có, có thể do người khác phát triển
  - Chúng ta cần hiểu code đó ➔ Hãy thử refactoring lại code để giúp chúng ta dễ hiểu nó hơn
  - “It’s much easier to make changes in clean code”



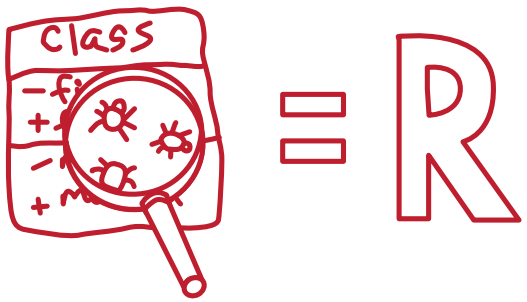
Khi nào cần thực hiện?
When to refactor?



- When fixing a bug
  - Bug trong code thường tập trung ở những chỗ rối rắm khó hiểu của mã nguồn
  - Đôi khi chỉ cần làm sạch mã là các bug sẽ "**tự xuất hiện**"

Khi nào cần thực hiện?
When to refactor?

- During a code review
  - Rà soát mã nguồn là cơ hội cuối để làm sạch mã trước khi nó được public cho toàn bộ đội phát triển
  - Việc rà soát thường được thực hiện bởi 2 người trong đó có tác giả của mã nguồn



Refactoring checklist

- ✓Kiểm tra xem code đã dễ đọc, dễ hiểu hơn chưa, các method quá dài đã được tách nhỏ hơn để dễ hiểu cấu trúc code?
- ✓Trong quá trình refactoring, kiểm tra để đảm bảo không có một chức năng mới nào được thêm vào
- ✓Tất cả các tests đều passed sau khi refactoring
- **Các kỹ thuật refactoring code [đọc thêm]**
  - https://refactoring.guru/refactoring/techniques

5. Rà soát mã nguồn
Code Review / Inspection

- Là quá trình mà mã nguồn (và cả thiết kế) được xem xét lại bởi một người (hoặc 1 nhóm người) khác thay vì chính tác giả
- Lợi ích - Benefits:
  - Góc nhìn khác với tác giả
    - Lỗi đôi khi dễ dàng được phát hiện bởi người khác hơn là tác giả
  - Chia sẻ tri thức
    - Xem xét các thiết kế và ý tưởng
  - Phát hiện lỗi sớm
  - Giảm thiểu công sức phải làm lại

Inspection vs Test

- Một số vấn đề chỉ được phát hiện trong quá trình review code thay vì kiểm thử
  - Xem xét một số tính chất của mã nguồn
    - Tính bảo trì được - Maintainability
    - Tính tái sử dụng được - Reusability
    - Tính dễ dàng mở rộng được – Extensibility
  - Xem xét độ tin cậy và tính tối ưu của thiết kế, tài liệu

Một số kĩ thuật rà soát
Different kinds of inspection

- Inspection / Formal technical review – Rà soát theo quy trình chính thống
  - Người tham gia: QA leaders, QA testers, developpers, project manager, team leaders etc.
  - Cần được chuẩn bị trước
    - Review checklist
  - Formal meeting:
    - Được chủ trì bởi một moderator không phải tác giả
    - Quá trình thực hiện phải được ghi chép lại đầy đủ
    - Theo danh sách checklist đã được chuẩn bị sẵn
  - Formal follow-up
    - Kết quả review sẽ được gửi cho tất cả các bên liên quan

Một số kĩ thuật rà soát
Different kinds of inspection

- Walkthroughs:
  - Không cần chuẩn bị trước
  - Tác giả sẽ là người dẫn dắt cuộc họp
  - Chi phí tốn ít hơn so với Formal Technical Review
  - Có tính chất đào tạo hơn


Q&A

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict