# Bài tập 06: Chỉnh sửa lại các thiết kế dựa trên kiến thức đã học. Phân tích cách chỉnh sửa và ý nghĩa đem lại.

A diagram of a computer program

Description automatically generated with medium confidence

1. **Single Responsibility Priciple:**

- Nguyên lý này nói rằng không nên có hơn 1 lý do để một lớp (class) thay đổi. Nghĩa là, nên thiết kế class của bạn sao cho mỗi class chỉ phục vụ cho một mục đích.

=> Khi áp dụng nguyên tắc này, em đã tách các class thành các lớp đơn lẻ phục vụ mục địch riêng hơn.

+ Ví dụ : Phân thành các model con như ChamCongInterface, StaffInterface và HRInterface và để chỉ ra rằng, các model này chỉ có chức năng như một biến, chỉ cho một mục đích là get và set dữ liệu vào một đối tượng Object.

1. **Open/Closed Priciple:**

- Nguyên lý này nói rằng các thực thể phần mềm như: classs, modules, functions,… thì có thể thoải mái viết code để mở rộng nhưng hạn chế cho việc sửa đổi.

- Ví dụ: một lớp cơ sở ChamCong chứa tất cả các thuộc tính và phương thức lấy dữ liệu chấm công cơ bản. Lớp này có thể được mở rộng bởi các lớp ChamCongRepository khác nhau cho các người sử dụng khác nhau(nhân viên or manager) để phục vụ các chức năng của họ. Do đó, nó được mở rộng dễ dàng và thoải mái nhưng hạn chế việc thay đổi.

1. **Liscov Substitution Priciple:**

- Nguyên ly này nói rằng các đối tượng của lớp con có thể thay thế được các đối tượng của lớp cha mà vẫn đảm bảo tính đúng đắn của chương trình.

=> Em đã áp dụng nguyên lý này cho các phương thức xem chấm công bù, xin nghỉ, đơn OT, đơn đi muộn về sớm.

+ Ví dụ: Model đơn từ có thể được kế thừa từ các model con như chấm công bù, xin nghỉ, đơn OT, đơn đi muộn về sớm. Vì các phương thức này cũng khá giống nhau nên việc mở rộng và thay thế phương thức lớp cha để phù hợp với lớp con là hoàn toàn hợp lý.

1. **Interface Segregation Priciple**

- Nguyên lý này nói rằng thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ với nhiều mục đích cụ thể.

=> Ví dụ ở cách thiết kế class trên, em đã phân thành các interface nhỏ hơn như ChamCongInterface, StaffInterface và DonTuInterface, rồi ném vào interface lớn như ChamCongRepositoryInterface.

1. **Dependency Inversion Priciple:**

- Nguyên lý này nói rằng, các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả hai nên phụ thuộc vào abstractions (interface).

=> Em đã phân tách thành các model con để implement các interface. Khi thực thi bất kỳ một action nào thì đều phải qua interface, tránh thực thi trực tiếp các model.

1. **DRY – Don’t Repeat Yourself**

- Nguyên lý này nói rằng mỗi mẩu kiến thức nhỏ (code) chỉ có thể xảy ra chính xác một lần trong toàn bộ hệ thống. Điều này giúp chúng ta viết mã mở rộng, bảo trì và có thể sử dụng lại được.  
=> Em sẽ nghiên cứu và chỉnh sửa cách viết code sao cho 1 đoạn code không phải thực thi nhiều lần

=> Solution: viết thành các mothod riêng lẻ hoặc các method để gộp chung vào 1 class service.

1. **Keep it simple, Stupid!**

- Nguyên lý này nói rằng cố gắng giữ cho từng phần nhỏ của phần mềm đơn giản và nên tránh phức tạp chúng không cần thiết. Điều này giúp chúng ta viết code dễ bảo trì.

=> Trong ví dụ trên, em thiết kế theo đơn giản dễ đọc, đối với những bạn mới join project cũng có khả năng đọc hiểu nhanh.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ý nghĩa đem lại :**

* Dễ bảo trì, dễ dàng trace code đồng thời dễ tìm bug vì áp dụng nguyên lý Single Responsibility Priciple, nguyên lý thiết kế giúp tạo ra mã nguồn dễ đọc, hiểu và bảo trì. Điều này giúp nhóm phát triển phần mềm dễ dàng thực hiện các cải tiến, sửa lỗi và nâng cấp hệ thống mà không gặp nhiều khó khăn.
* Có khả năng mở rộng vì áp dụng nguyên lý Open/Closed Priciple.
* Có khả năng tái sử dụng vì áp dụng nguyên lý Liscov Substitution Priciple, nguyên lý thiết kế giúp tạo ra các lớp con, có khả năng tái sử dụng lại lớp cha.
* Hiệu suất (Performance): Thiết kế chặt chẽ và tuân thủ nguyên lý giúp tối ưu hóa hiệu suất của phần mềm. Các thuật toán và cấu trúc dữ liệu có thể được tối ưu hóa để đảm bảo phần mềm chạy mượt mà và đáp ứng nhanh chóng.
* Dễ dàng quản lý, dễ dàng đọc đồng thời tuân theo một quy chuẩn nhất định của Framework vì áp dụng nguyên lý Interface Segregation Priciple
* **Tóm lại**, việc áp dụng các nguyên lý thiết kế trong phát triển phần mềm không chỉ giúp tạo ra sản phẩm chất lượng cao mà còn giảm thiểu rủi ro, chi phí và thời gian liên quan đến quá trình phát triển và duy trì.