

The Single-Responsibility Principle (SRP)

“A class should have only one reason to change.

Đó là cách mà Robert C. Martin nói về SRP. Điểm quan trọng ở đây là **"reason to change"**. Trong ngữ cảnh của SRP, một responsibility được định nghĩa là một lý do khiến class phải thay đổi. Nếu chúng ta có thể nghĩ ra được nhiều hơn 1 lý do độc lập khiến class phải thay đổi tức là class đó đang đảm nhận nhiều hơn một trách nhiệm.

Có nghĩa là SRP không đơn giản là mỗi class chỉ làm một việc, càng không có nghĩa là một class chỉ nên có 1 method. Câu hỏi chính xác nên là: **"Có những yêu cầu nào từ hệ thống khiến tôi phải sửa class này?"**.

Responsibility là gì?

Với SRP, Responsibility có thể hiểu là một lý do để class đó phải thay đổi.

Ví dụ 1

```
public class Player
{
    public int CalculateDamage()
    {
        // Gameplay logic
    }

    public void Save()
    {
        // Save player
    }
}
```

Trong ví dụ này, 2 method đều liên quan đến player nhưng chúng thay đổi vì những lý do khác nhau.

- Khi gameplay thay đổi, công thức tính sát thương thay đổi thì cần sửa lại `CalculateDamage()`
- Khi logic save thay đổi. Save local sang cloud, save json sang binary... thì cần sửa lại `Save()`

Như vậy, Player đang có 2 responsibility.

Ví dụ 2

```
public class Rectangle
{
    public double GetArea()
    {
        // Calculate area
    }

    public void Draw()
    {
        // Draw on screen
    }
}
```

Với ví dụ này chúng ta cũng có thể thấy có 2 phần là logic tính toán và render. Khi sử dụng Rectangle trong tính toán chúng ta mặc nhiên bị kéo theo cả GUI một cách không cần thiết và ngược lại.

SRP được sinh ra để tránh các coupling không cần thiết.

Khi nào nên tách class

Trong thực tế không phải lúc nào việc tách class và đảm bảo SPR cũng đem lại lợi ích. Tùy vào dự án thực tế để lựa chọn tách hoặc không tách class cho phù hợp với yêu cầu dự án.

Ví dụ 1

```
public interface Modem
{
    void Dial(string phoneNumber);
    void Hangup();

    void Send(char data);
    char Receive();
}
```

Nhìn vào ví dụ này chúng ta có thể thấy được 2 lý do để Modem phải thay đổi.

- Quản lý kết nối: Dial(), Hangup()
- Truyền dữ liệu: Send(), Receive()

Mặc dù 2 nhóm này có thể thay đổi độc lập nhưng điều đó **không có nghĩa là cần phải tách chúng ngay lập tức.**

Nếu trong hệ thống thực tế 2 nhóm này luôn thay đổi cùng nhau. Ví dụ như chỉ cần cập nhật khi hạ tầng phần cứng thay đổi. Khi đó cả 2 nhóm sẽ thay đổi cùng nhau. Khi này việc tạo thêm interface và class chỉ để đảm bảo đúng SPR khiến cho dự án trở nên phức tạp một cách không cần thiết.

SRP nên được áp dụng khi vấn đề thay đổi thực sự xuất hiện, không phải để dự đoán mọi khả năng có thể xảy ra trong tương lai.

Ví dụ 2

```
public class Employee
{
    public decimal CalculateSalary()
    {
        // Business rules
    }

    public void Save()
    {
        // Database
    }
}
```

Ví dụ này cũng tương tự như ví dụ về player ở phần trên. Quy tắc tính lương và cách lưu dữ liệu thay đổi vì những lý do hoàn toàn khác nhau.

Quy tắc tính lương có thể thay đổi theo nhân sự, theo chính sách công ty... còn save sẽ chỉ cần thay đổi khi thay đổi hạ tầng. Vậy nên chúng có thể tách ra như sau:

```
public class Employee
{
    public decimal CalculateSalary()
    {
        // Business rules
    }
}
```

```
public class EmployeeRepository
{
    public void Save(Employee employee)
    {
        // Persistence
    }
}
```

Việc tách ra này giúp cho việc thay đổi cách tính lương không kéo theo cả việc cập nhật lại cách lưu dữ liệu.

Cách nhận biết SRP

SRP không nói về độ lớn file code, số dòng code, số method hay kích thước class.

Nó nói về **lý do mà class cần thay đổi**. Khi đọc một class hãy thử hoàn thành câu: "Class này sẽ phải thay đổi nếu..."

Nếu bạn có câu là lời kiểu như *Player sẽ cần thay đổi nếu UI thay đổi, gameplay thay đổi, save format thay đổi...* thì rất có khả năng Player đang chứa quá nhiều responsibility.

Khi đó chúng ta có thể cân nhắc tách Player thành:

- Player → Gameplay

- PlayerRepository → Persistence
- PlayerView → Presentation
- ...

Mục tiêu cuối cùng không phải là càng nhiều class càng tốt, class càng ngắn càng tốt mà là không để thay đổi phải kéo theo những phần không liên quan.

Nhớ rằng, **một class chỉ nên có một lý do để thay đổi**. Nhưng cái khó khi áp dụng SRP là làm sao nhận biết được đâu mới thực sự là responsibility.

Revision #2

Created 2026-08-20 02:51:36 UTC by ThanhDV

Updated 2026-08-20 09:28:58 UTC by ThanhDV