**Single Responsibility Principle (SRP)**

El Principio de Responsabilidad Única (Single Responsibility Principle, SRP) es uno de los principios fundamentales de SOLID. Este principio establece que una clase debe tener una única razón para cambiar, lo que significa que debe cumplir con una sola responsabilidad dentro del sistema.

Este principio fue formulado por Robert C. Martin ("Uncle Bob") y ayuda a mejorar la organización y mantenibilidad del código, promoviendo un diseño más modular y desacoplado.

* **Concepto de SRP**

SRP se basa en la idea de que cada módulo, clase o función debe encargarse únicamente de una parte específica del sistema. Si una clase tiene múltiples responsabilidades, puede volverse difícil de entender, modificar y probar.

Una buena práctica es dividir grandes clases en varias clases más pequeñas y especializadas, de modo que cada una maneje una sola función.

# **Beneficios del SRP**

Aplicar correctamente el principio de responsabilidad única proporciona múltiples ventajas en el desarrollo de software:

1. Código más mantenible: Al separar responsabilidades, los cambios en una parte del código no afectan otras áreas del sistema.
2. Menor acoplamiento: Cada clase se enfoca en una tarea específica, reduciendo la dependencia entre módulos.
3. Mayor reutilización: Componentes más pequeños y especializados pueden ser reutilizados en diferentes partes del sistema.
4. Facilita las pruebas unitarias: Al tener clases con responsabilidades bien definidas, las pruebas se pueden escribir y ejecutar de manera más eficiente.
5. Mejora la claridad del código: Facilita la comprensión del código, reduciendo la complejidad en el mantenimiento y escalabilidad.

# **Relación con Clean Architecture y TDD**

* Clean Architecture: SRP es clave en la separación de capas. Por ejemplo, una capa de negocio no debería encargarse de detalles de infraestructura como la persistencia de datos.
* TDD: La implementación de SRP facilita el desarrollo basado en pruebas, ya que las clases pequeñas y con una sola responsabilidad son más fáciles de probar de manera aislada.

# **Ejemplo Implementado en el Proyecto**

En este proyecto, aplicamos SRP al separar la lógica de gestión de archivos en dos clases:

### **Antes (Violación de SRP)**

public class FileService

{

public void SaveFile(string fileName, string content)

{

// Guardar archivo

File.WriteAllText(fileName, content);

// Registrar log

Console.WriteLine("Archivo guardado: " + fileName);

}

}

Aquí FileService maneja tanto la escritura de archivos como el logging, lo cual viola SRP.

### **Después (Aplicando SRP)**

public class FileManager

{

public void WriteToFile(string fileName, string content)

{

File.WriteAllText(fileName, content);

}

}

public class FileService

{

private readonly FileManager \_fileManager;

private readonly ILogger<FileService> \_logger;

public FileService(FileManager fileManager, ILogger<FileService> logger)

{

\_fileManager = fileManager;

\_logger = logger;

}

public void SaveFile(string fileName, string content)

{

\_fileManager.WriteToFile(fileName, content);

\_logger.LogInformation("Archivo guardado: {FileName}", fileName);

}

}

Ahora FileManager se encarga solo de la escritura de archivos y FileService solo de la lógica de servicio.

* **Cómo Aplicar SRP en la Práctica**

1. Identificar las responsabilidades: Revisar si una clase está manejando más de una tarea.
2. Dividir en clases más específicas: Extraer funcionalidades independientes en nuevas clases.
3. Utilizar inyección de dependencias: En lugar de que una clase cree sus propias dependencias, se le deben proporcionar desde el exterior.
4. Evaluar periódicamente: A medida que la aplicación crece, verificar que cada clase mantenga su única responsabilidad.