Reflexión General sobre los Principios SOLID

¿Cuál fue el principio más desafiante de aplicar? ¿Por qué?

Andres: El Principio de Inversión de Dependencias (DIP) fue el más desafiante, porque identificar todas las dependencias ocultas e introducir abstracciones adecuadas nos obligó a replantear la arquitectura e implementar inyección de dependencias.

Juan: El Principio Abierto/Cerrado (OCP) resultó algo complicado diseñar la capa de notificaciones para crecer sin modificar código existente nos olbigo a crear interfaces correctas antes de escribir cualquier lógica.

Leonardo: El Principio de Sustitución de Liskov (LSP) supuso el mayor reto; garantizar que cada subclase respetara por completo el contrato de su superclase nos dio herencias forzadas y métodos que no encajaban en el dominio.

Sammy: El Principio de Segregación de Interfaces (ISP) fue el más complicado para mí, ya que dividir una interfaz monolítica en contratos mínimos es difícil de comprender a fondo qué necesitaba realmente cada dispositivo.

¿Cómo crees que SOLID mejora el diseño de software?

Andres: Convierte el proyecto en un conjunto modular donde las piezas cambian sin afectar a las demás, aumentando la mantenibilidad y la velocidad de las pruebas.

Juan: Reduce la tasa de regresiones, al extender en lugar de modificar, el código probado permanece intacto y el despliegue continuo se vuelve más confiable.

Leonardo: Aporta coherencia semántica. Si un objeto cumple lo que promete su tipo, se eliminan validaciones redundantes y parches temporales.

Sammy: Hace que las APIs sean autoexplicativas; cada clase ve solo lo que necesita, disminuyendo el acoplamiento y facilitando la incorporación de nuevos desarrolladores.

¿Específicamente qué principio SOLID aplicarías de lo aprendido, en futuros o actuales proyectos?

Andres: SRP porque asegura que cada clase tenga una sola razón de cambio simplifica la integración del resto de principios.

Juan: OCP ya que pienso asegurar funcionalidades detrás de interfaces para que añadir variantes requiera solo nuevas clases, no editar las existentes.

Leonardo: LSP antes que todo para heredar, verificaré que la subclase cumpla exactamente el mismo contrato, prefiriendo composición cuando no sea así.

Sammy: DIP con inyección de dependencias puedo intercambiar implementaciones — bases de datos, pasarelas de pago, en poco tiempo, crucial para proyectos que escalan rápido.