# Las responsabilidades ocultas del software

Imaginá que estás trabajando en una aplicación para una librería. Al principio, el sistema era sencillo: cargar libros, buscar por autor o título, y registrar ventas. Una compañera del equipo diseñó una clase principal llamada Libreria, que en sus primeras versiones funcionaba perfectamente.

Con el tiempo, el sistema fue creciendo. Se agregaron nuevos requerimientos: enviar correos promocionales a los clientes, generar estadísticas mensuales de ventas, realizar backups automáticos de la base de datos y sincronizar los precios con un proveedor online. Todos estos cambios se fueron incorporando a la misma clase Libreria.

A simple vista, la idea parecía práctica: tener “todo en un solo lugar”. Sin embargo, empezaron a aparecer problemas. Cada vez que alguien tocaba el código de Libreria, algo se rompía. Era común que una mejora en el cálculo de estadísticas afectara por error la lógica de ventas. Algunos métodos crecieron tanto que ocupaban más de 100 líneas, con condicionales anidados y variables difíciles de seguir. Otros métodos tenían nombres tan generales como procesar() o manejarDatos(), que no dejaban claro qué hacían realmente.

Los nuevos integrantes del equipo tardaban horas en entender qué hacía la clase. Incluso desarrolladores con experiencia preferían no modificarla por miedo a causar errores colaterales. Las pruebas automáticas eran difíciles de mantener, y cualquier ajuste requería revisar varias partes del sistema.

En una reunión, alguien propuso reorganizar el diseño: dividir esa gran clase en componentes más pequeños y específicos. Así nacieron clases como GestorDeLibros, GestorDeClientes, AnalizadorDeVentas y ServicioDeCorreo. Cada una tenía una única función clara, era más fácil de testear y de comprender.

Al principio, la tarea pareció engorrosa. Requirió mover código, ajustar métodos y redefinir responsabilidades. Pero con el tiempo, los beneficios fueron evidentes: los errores disminuyeron, el desarrollo se volvió más ágil, y cada clase se pudo mejorar de forma independiente sin comprometer todo el sistema.

Diseñar software no es solo escribir código que funcione. También implica pensar en cómo ese código va a vivir, crecer y ser modificado con el tiempo. Las responsabilidades mal definidas o mezcladas no siempre generan errores inmediatos, pero casi siempre se convierten en problemas silenciosos que complican el futuro.

A veces, las verdaderas responsabilidades del software no están a la vista desde el primer día. Se revelan con el uso, con los cambios y con la experiencia. Por eso, aprender a identificarlas —y a distribuirlas con criterio— es una habilidad clave para cualquier desarrollador.

**❓ Preguntas a responder:**

1. ¿Qué problemas empezaron a aparecer en la clase **Libreria** a medida que fue creciendo?
2. ¿Qué ventajas trajo el rediseño en clases más pequeñas y específicas?
3. ¿Qué dificultades enfrentaron inicialmente al dividir la clase? ¿Valió la pena ese esfuerzo? Justificá.
4. Explicá con tus palabras qué significa que “las responsabilidades del software no siempre están a la vista desde el primer día”.
5. ¿Por qué creés que una clase con muchos métodos diferentes puede dificultar el trabajo en equipo?