MANEJO DE ERRORES

CONSTRUYENDO RESILENCIA

Idea principal

El manejo adecuado de errores es crucial para la estabilidad y la mantenibilidad de cualquier sistema. No se trata solo de "atrapar" excepciones, sino de diseñar un flujo de control que anticipe y gestione fallas de manera elegante.

Errores vs Excepciones

Errores: Situaciones irrecuperables (p.ej., falta de memoria). Deben terminar el programa.

Excepciones: Condiciones anómalas pero recuperables (p.ej., archivo no encontrado). Deben ser manejadas.

Principios clave

Usa excepciones, no códigos de error: Simplifica el código y separa la lógica de negocio del manejo de errores.

Provee contexto significativo: Los mensajes de error deben ser informativos y útiles para la depuración

Lanza excepciones enelnivel de abstracción correcto: No expongas detalles de implementación de bajo nivel.

Tecnicas y ejemplos

Clases de excepciones personalizadas

Crea jerarquías de excepciones para clasificar y manejar diferentes tipos de errores de manera específica.

```
public class MiExcepcionArchivoNoEncontrado extends MiExcepcionDeAplicacion {}
```

No devolver nulos

No permitas que se pasen nulos a tus métodos. Valida los argumentos al principio de la función para evitar NullPointerExceptions.

```
public void procesar(Objeto o) {if (o == null) throw new IllegalArgumentException("El objeto no puede ser nulo");// ... lógica ...}
```

FRONTERAS

CONVIVENCIA CON EL CÓDIGO EXTERNO

Idea principal

Las fronteras son los puntos de interacción de nuestro código con el mundo exterior: APIs de terceros, frameworks, sistemas operativos, bases de datos. Robert C. Martin nos enseña a gestionar estas interacciones para evitar que la inestabilidad externa contamine nuestro código.

Mapas o Envolturas

Envuelve las APIs externas en tus propias clases o interfaces. Esto te da control y te permite cambiar la implementación subyacente sin afectar el resto de tu código.

Pruebas de Aprendizaje

Antes de integrar una nueva API, escribe pequeñas pruebas para entender su comportamiento. Esto documenta tu entendimiento y protege tu código de futuros cambios en la API externa.

Controlar las Dependencias

Define claramente qué partes de tu sistema interactúan con las fronteras. Esto minimiza el acoplamiento y facilita la refactorización.

En resumen

- Se trata de interacciones con APIs, librerías externas, frameworks, BD.
- Usa envolturas/adaptadores para proteger tu código de cambios externos.
- Haz pruebas de aprendizaje antes de integrar nuevas APIs.
- Controla dependencias → menos acoplamiento, más flexibilidad.
- 👉 Enfoque: proteger tu sistema de la inestabilidad externa.

PRUEBAS UNITARIAS

LA CALIDAD ES EL CIMIENTO

Idea principal

Para Robert C. Martin, las pruebas unitarias no son solo una herramienta de verificación, sino un pilar fundamental del buen diseño de software. Son la primera línea de defensa contra los errores y un excelente mecanismo para documentar el comportamiento esperado del código.

Rapidas

Deben ejecutarse rápidamente para poder ser corridas con frecuencia.

Independientes

Cada prueba debe ser independiente de las demás y su orden de ejecución no debe importar.

Repetibles

Deben producir el mismo resultado cada vez que se ejecuten.

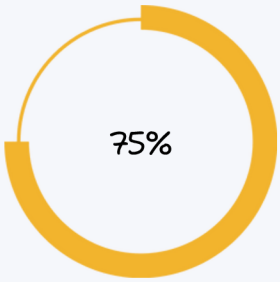
Auto-verificables

Deben ser capaces de determinar si pasaron o fallaron por sí mismas.

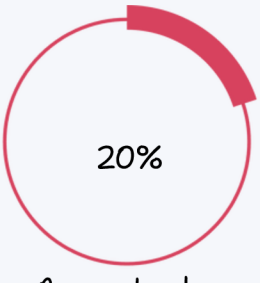
Oportunas

Deben escribirse antes del código de producción (TDD) para influir en el diseño.

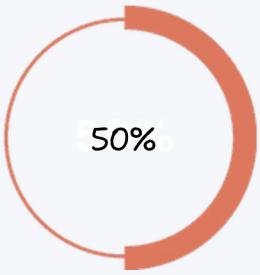
Conclusiones y Siguientes Pasos



Reducción de errores en producción al aplicar TDD.



Aumento de la velocidad de desarrollo a largo plazo.



Mejora en la legibilidad y mantenibilidad del código.

Los principios de código limpio de Robert C. Martin son atemporales y esenciales para cualquier desarrollador que busque construir sistemas de software robustos, mantenibles y escalables.