Sesión 14 Code Smells y Análisis Estático de Código con SonarQube y ESLint

Hasta ahora hemos aprendido **a estructurar bien el código** (patrones, SOLID, cohesión y acoplamiento).

En esta sesión damos un paso clave:
Aprender a detectar código que "huele mal" antes de que se convierta en un problema real.

Esta sesión conecta directamente con:

- SOLID
- Refactorización
- Buenas prácticas profesionales
- Trabajo en equipo
- Calidad del software en empresas reales

1. Objetivos

- Explicar qué es un **code smell** y por qué no es un error de compilación.
- Diferenciar **bug**, **code smell** y **deuda técnica**.
- Identificar manualmente code smells comunes.
- Comprender qué es el **análisis estático de código**.
- Interpretar informes reales de **SonarQube**.
- Utilizar **ESLint** para mejorar código JavaScript.
- Refactorizar código siguiendo indicaciones de herramientas.
- Entender cómo trabajan los **equipos profesionales**.

2. ¿Qué es un Code Smell?

Un **code smell** es un **síntoma** de que el código tiene un problema de diseño, estructura o mantenibilidad, aunque funcione correctamente.

Un code smell **no rompe el programa hoy**, pero **lo rompe mañana**.

Analogía real

- Un coche sin mantenimiento: hoy arranca
- Un edificio con grietas: hoy no se cae
- Código sin refactorizar: hoy funciona

El problema no es el presente, es el **futuro del software**.

3. Bug vs Code Smell vs Deuda Técnica

Concepto	¿Falla el programa?	Impacto
Bug	Sí	Error visible inmediato
Code Smell	No	Dificulta mantenimiento
Deuda técnica	No (al inicio)	Aumenta el coste del proyecto

- Bug: división entre cero
- Code smell: método de 300 líneas
- Deuda técnica: "ya lo arreglaremos luego"

4. Por qué los Code Smells son peligrosos

Un proyecto con muchos code smells:

- Es difícil de entender
- Es caro de modificar
- Genera errores colaterales
- Provoca miedo a tocar el código

Código que da miedo = proyecto muerto

5. Catálogo completo de Code Smells (con ejemplos)

5.1 Long Method (Métodos largos)

Síntomas:

- Más de 30–40 líneas
- Muchos comentarios explicativos

```
public void processOrder() {
  validateUser();
  calculatePrices();
  applyDiscounts();
```

```
  saveOrder();
  sendEmail();
}
```

Problemas:

- No es reutilizable
- No es testeable

Refactorización:

- Extract Method

5.2 God Class (Clases Dios)

Una clase que:

- Controla lógica
- Accede a la BD
- Gestiona usuarios
- Genera informes

```
class ApplicationManager { ... }
```

Violaciones:

- SRP (SOLID)
- Cohesión baja

5.3 Duplicated Code

```
if (password.length > 8 && password.includes("@")) { }
```

Copiado en 5 archivos distintos.

Problema real:

- Cambiar reglas = 5 cambios

Solución:

- Función común

5.4 Large Parameter List

```
createUser(name, surname, email, phone, address, city, zip)
```

Solución:

- Objeto DTO

5.5 Feature Envy

Una clase usa más datos de otra que los suyos.
Método mal ubicado.

5.6 Primitive Obsession

String status = "P";
Solución:

- Enum

5.7 Switch / If complejos
if(type == 1) {...}
else if(type == 2) {...}
Solución:

- Polimorfismo
- Strategy

5.8 Nombres pobres
data, x, tmp
totalInvoiceAmount

6. Qué es el Análisis Estático de Código

Definición
Proceso automático que analiza el código **sin ejecutarlo** para detectar:

- Bugs potenciales
- Vulnerabilidades
- Code smells
- Malas prácticas

Diferencia con análisis dinámico

Estático	Dinámico
Sin ejecutar	En ejecución
Prevención	Detección tardía
Más barato	Más costoso

7. SonarQube en profundidad

Qué es SonarQube
Plataforma profesional de **calidad de código** usada por empresas.
Analiza:

- Bugs

- Vulnerabilidades
- Code smells
- Duplicaciones
- Complejidad
- Deuda técnica

Conceptos clave

- **Quality Gate**: reglas mínimas para aprobar código
- **Technical Debt**: tiempo estimado para corregir problemas
- **Maintainability Rating**

Ejemplo real de informe

- 12 code smells
- Deuda técnica: 1h 30min
- Severidad: minor / major / critical

SonarQube **no solo señala**, explica.

8. ESLint en profundidad

Qué es ESLint

Analizador estático para JavaScript.
Detecta:

- Variables no usadas
- Errores de alcance
- Código peligroso
- Estilo inconsistente

Ejemplo

```
let count = 0;
function test() {
  let count = 5;
}
```

Variable sombreada

ESLint en tiempo real

- Avisos mientras escribes
- Prevención inmediata

9. SonarQube vs ESLint

SonarQube	ESLint
Multilenguaje	JS/TS
Análisis global	Editor
Ideal empresas	Ideal aprendizaje

10. Flujo profesional completo

1. Programador escribe código
2. ESLint detecta errores
3. Código se sube al repositorio
4. SonarQube analiza
5. Quality Gate decide
6. Refactorización

Así se trabaja en empresa.