

UD 4: Análisis de Código

Contornos de desenvolvimento (CODE)

Eduardo Mosqueira (UDC) & Víctor Blanco

Curso 2023-2024



XUNTA
DE GALICIA

IES Muralla Romana
Dpto. de Informática

Índice

1 Herramientas

2 Instalación

Índice

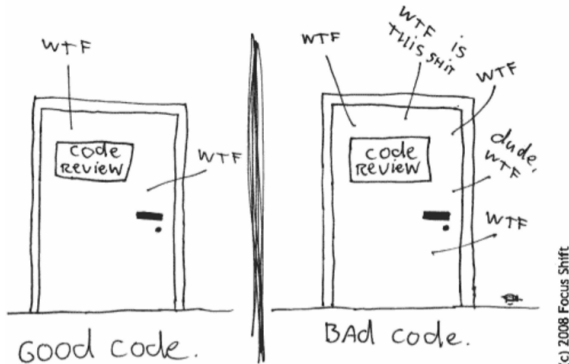
1 Herramientas

- Calidad y Predictibilidad del Software
- Herramientas

2 Instalación

WTFs / Minute

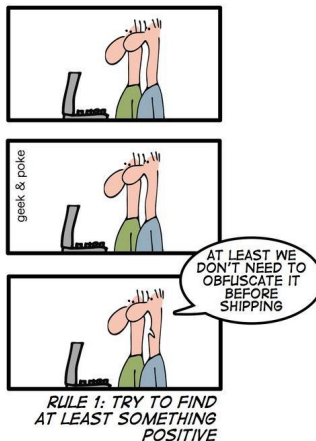
The ONLY valid measurement
of code quality: WTFs/minute



(c) 2008 Focus Shift

Revisión del código

HOW TO MAKE A GOOD CODE REVIEW



Calidad del software

Calidad del Software

Se determina a través de la frecuencia de errores (*bugs*) críticos o bloqueantes descubiertos después del lanzamiento del software

Calidad del software

Calidad del Software

Se determina a través de la frecuencia de errores (*bugs*) críticos o bloqueantes descubiertos después del lanzamiento del software

- Se considera que una calidad es del 100 % si no existe ningún error crítico o bloqueante después del lanzamiento del software

Do you find critical or blocker bugs after a release?

- A.** No - 100%
- B.** Almost never - 75%
- C.** Sometimes - 50%
- D.** Often - 25%
- E.** Always - 0%

Análisis de ZeroTurnaround RebelLabs: <https://zeroturnaround.com/rebellabs/>



Predictibilidad del software

Predictibilidad del Software

Se determina a partir de los retrasos en los lanzamientos, la ejecución de los requisitos planeados y los cambios dentro del proceso de desarrollo (*scope creep*)

Predictibilidad del software

Predictibilidad del Software

Se determina a partir de los retrasos en los lanzamientos, la ejecución de los requisitos planeados y los cambios dentro del proceso de desarrollo (*scope creep*)

- Se considera que una predictibilidad es del 100 % si los lanzamientos son realizados a tiempo tal y como estaba planeado

How late are your releases (vs initial planned time)?

- A. On time or early 0%
- B. A bit 10%
- C. Moderately 25%
- D. A lot 50%

How much of the original plans get done?

- A. Everything
- B. All but 10%
- C. All but 25%
- D. Only 50%
- E. Only 25%

How much do plans change/expand during development? (scope creep)

- A. None
- B. 10% creep
- C. 25% creep
- D. 50% creep
- E. 75% or more creep

Predictibilidad del software

$$\text{PREDICTABILITY} = \left(\frac{1}{(1 + \% \text{ late})} \right) \times (\% \text{ of plans delivered}) \times (1 - \% \text{ scope creep})$$

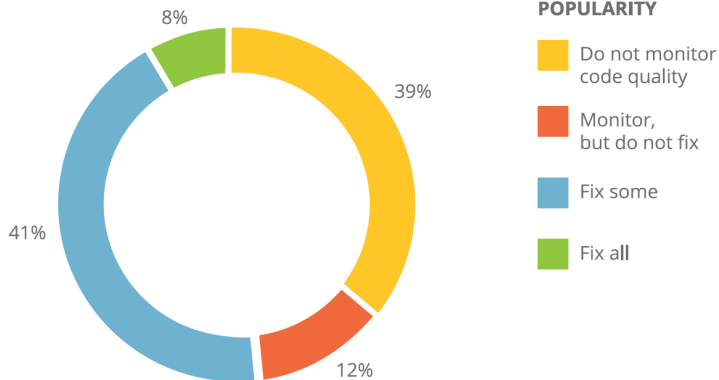
Example: Simon's team is pretty good at delivering on time--they only release late 10% of the time. On average, they get 75% of their development plans done, and they've been able to limit scope creep to just 10% as well. Based on that, we can calculate that Simon's team releases software with 61% predictability.

$$\text{MATH!} \left(\frac{1}{(1 + 0.10)} \right) \times (0.75) \times (1 - 0.10) = 0.61 = 61\%$$



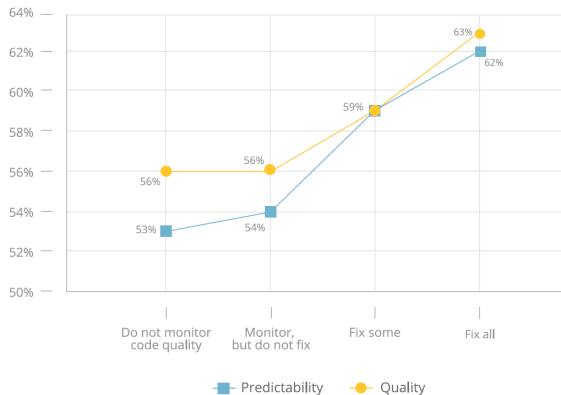
Revisión del código (2013)

DO YOU MONITOR AND FIX CODE QUALITY PROBLEMS? (e.g. with Sonar)

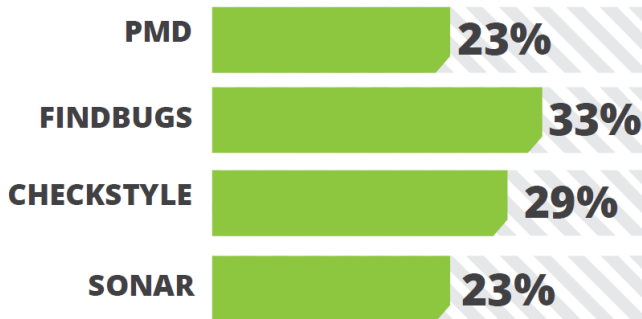


Revisión del código (2013)

The effects of fixing code quality issues on predictability and quality



Herramientas (2012)



Herramientas (2024)

Herramientas de pago de referencia

Hoy en día hay dos herramientas muy populares en lo que respecta a la calidad del código.

- SonarQube: 95 % de cuota de mercado.
- Atlassian Crucible: Cuota de mercado significativamente inferior.

Herramientas (2024)

Herramientas gratuitas y de código abierto

La rápida evolución tecnológica ha transformado completamente el escenario de este tipo de sistemas en este corto periodo de tiempo.

- FindBugs: discontinuado.
- PMD y Checkstyle: siguen en funcionamiento. Nos centraremos en PMD.
- SonarLint: es una extensión IDE que identifica y ayuda a solucionar los problemas de calidad y seguridad del código mientras el programador lo escribe.

PMD



- Herramienta de análisis de código fuente
- Se centra más en analizar malas prácticas que pueden o no llevar a errores en el código
- Página web: <https://github.com/pmd>
- Aunque el acrónimo oficialmente no significa nada, se le han buscado significados como *Programming Mistake Detector*

Aspectos típicos a analizar en PMD

- **Posibles bugs**

- P.ej. sentencias `try/catch/finally/switch` vacías

- **Código muerto**

- P.ej. variables locales o parámetros no usados

- **Código subóptimo**

- P.ej. uso incorrecto de las clases `String/StringBuffer`

- **Código duplicado**

- Copiar y pegar código es copiar y pegar errores

- **etc.**



Checkstyle



- Herramienta que analiza si el código desarrollado se adhiere a unos estándares de codificación apropiados
- Por ejemplo los Sun Code Conventions o el Google Java Style.
- Distribuido a través de una licencia LGPL
- Página web: <http://checkstyle.sourceforge.net/>.
- Repositorio:
<https://github.com/checkstyle/checkstyle>

Checkstyle

■ Algunos aspectos analizados por Checkstyle

- Comentarios Javadoc para clases, atributos y métodos
- Convenciones de nombres en clases, atributos y métodos
- Número de parámetros en una función
- Longitud de las líneas
- Uso de los imports
- Espacios entre los caracteres
- Código duplicado
- Múltiples medidas de complejidad
- etc.

■ Se puede ver una lista de los mismos en

<http://checkstyle.sourceforge.net/checks.html>

SonarLint



- Extensión gratuita y open-source que identifica y ayuda a solucionar los problemas de calidad y seguridad mientras el programador escribe el código.
- Podría hacerse un símil con un corrector ortográfico, que proporciona comentarios en tiempo real y una guía de corrección para construir un código limpio desde el inicio del proyecto.
- Unifica en una única herramienta el control de calidad y seguridad del código.
- Página web: <https://www.excentia.es/sonarlint>.

SonarLint

Disponibilidad

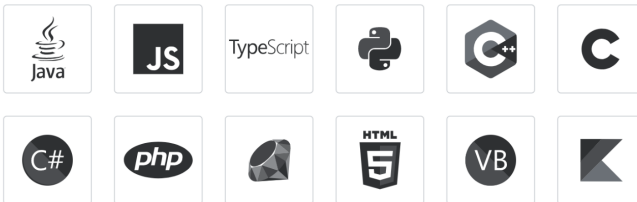
Todos los IDE, menos NetBeans.



SonarLint

Lenguajes

Soporte para múltiples lenguajes, entre ellos, Java.

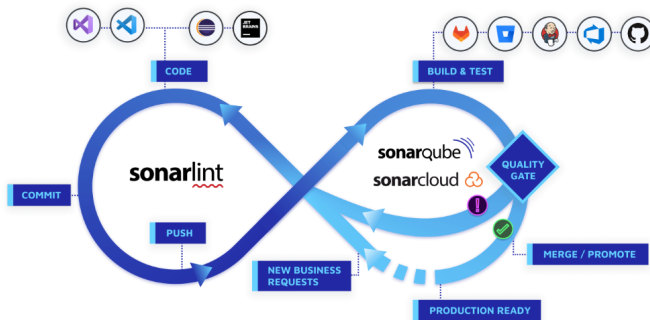


SonarLint

Integración con otros sistemas

Puede integrarse con herramientas en la nube como SonarQube o SonarCloud.

También es compatible con sistemas de gestión de proyectos, como GitHub, GitLab, Bitbucket, etc.



Comparativa

■ SonarLint

- Es la herramienta gratuita más moderna y completa
- Trabaja con muchos lenguajes de programación

■ PMD

- Especialmente buena en la detección de problemas relacionados con malas prácticas en la programación
- No solo trabaja con Java, sino con otros lenguajes como JavaScript
- Los análisis pueden llegar a ser lentos, pudiendo causar la sensación de que no funciona cuando se configura inicialmente

■ Checkstyle

- Se centra en la detección de violaciones de las convenciones de codificación
- No trata de encontrar *bugs*
- Es recomendable activar sólo aquellas reglas aceptadas por el equipo para evitar numerosos falsos positivos

Comparativa

Conclusión

Con el uso combinado de las tres herramientas podemos cubrir todo el rango necesario de aspectos a analizar en nuestro código



Índice

1 Herramientas

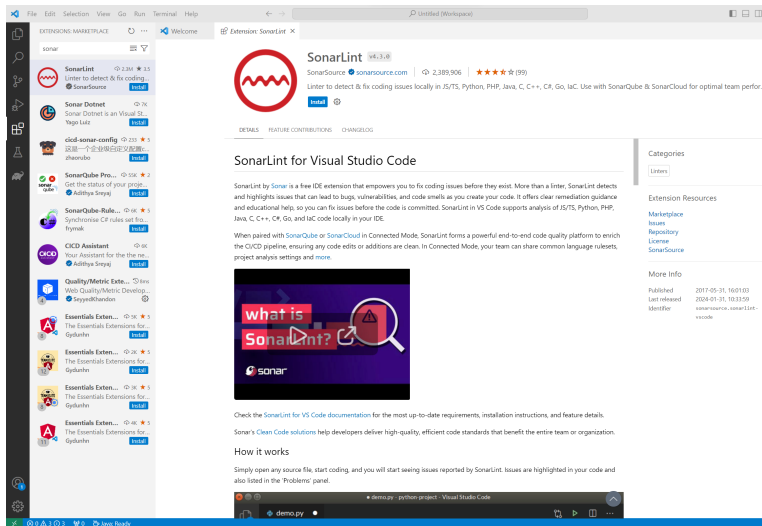
2 Instalación

- SonarLint

SonarLint

- Dentro del VSCode, nos vamos a *extensions*, buscamos *sonarlint*
- Nos aparece una ventana en la que se describe la herramienta, se especifican detalles, requisitos y su funcionamiento
- Seleccionamos la opción *instalar*

SonarLint



SonarLint v4.3.0

SonarSource sonarsource.com 2,389,906 ★★★★★ (99)

Linter to detect & fix coding issues locally in JS/TS, Python, PHP, Java, C, C++, C#, Go, IaC. Use with SonarQube & SonarCloud for optimal team perfor...

[Install](#)

[Details](#) [Feature Contributions](#) [ChangeLog](#)

SonarLint for Visual Studio Code

SonarLint by Sonar is a free IDE extension that empowers you to fix coding issues before they exist. More than a linter, SonarLint detects and highlights issues that can lead to bugs, vulnerabilities, and code smells as you create your code. It offers clear remediation guidance and educational help, so you can fix issues before the code is committed. SonarLint in VS Code supports analysis of JS/TS, Python, PHP, Java, C, C++, C#, Go, and IaC code locally in your IDE.

When paired with [SonarQube](#) or [SonarCloud](#) in Connected Mode, SonarLint forms a powerful end-to-end code quality platform to enrich the CI/CD pipeline, ensuring any code edits or additions are clean. In Connected Mode, your team can share common language rulesets, project analysis settings and [more](#).

Categories

- [Linters](#)

Extension Resources

- [Marketplace](#)
- [Issues](#)
- [Repository](#)
- [License](#)
- [SonarSource](#)

More Info

Published	2017-05-31, 16@103
Last released	2024-01-31, 10:33:59
Identifier	sonarsource.sonarlint-vscode

Check the [SonarLint for VS Code documentation](#) for the most up-to-date requirements, installation instructions, and feature details.

Sonar's [Clean Code solutions](#) help developers deliver high-quality, efficient code standards that benefit the entire team or organization.

How it works

Simply open any source file, start coding, and you will start seeing issues reported by SonarLint. Issues are highlighted in your code and also listed in the "Problems" panel.

what is SonarLint?

demo.py - python project - Visual Studio Code

SonarLint

- Posteriormente, abrimos nuestro proyecto Java (en este caso creado con Gradle)
- Nos situamos en una clase cualquiera
- En la parte inferior de la ventana, en la sección *problems*, aparecen indicadas las clases que presentan algún tipo de error o advertencia
- Pulsando sobre la clase, se despliega la lista de líneas en las que se enumeran dichas indicaciones, así como la línea en la que se encuentran y la recomendación de qué hacer
- Pulsando sobre cada advertencia, la herramienta nos lleva a la línea de código correspondiente, para poder actuar
- En la parte derecha, se puede filtrar por nombre o tipo de advertencia

SonarLint

Terminal Help ← → Analizador

Welcome J App.java 9+ X

app > src > main > java > analizador > J App.java > App > main(String[])

```
1  /*
2  * This Java source file was generated by the Gradle 'init' task.
3  */
4  package analizador;
5
6  import java.util.Scanner;
7
8  public class App {
9      public String getGreeting() {
10         return "Hello World!";
11     }
12
13     /**
14     * @param args
15     */
16     public static void main(String[] args) {
17         System.out.println(new App().getGreeting());
18         //TODO
19         //FIXME
20         //XXX
21         int opcion = 0;
22         do {
23             Scanner teclado = new Scanner(System.in);
24             opcion = 0;
25             System.out.println("Bienvenido al examen de la UD2 - estructuras de control. Seleccione la opción deseada");
26             System.out.println("1- Opcion 1");
27             System.out.println("2- Opcion 2");
28             System.out.println("3- Opcion 3");
29             /*Esperamos a que se introduzca que opcion selecciona*/
30             opcion = teclado.nextInt();
31             Integer Entero = null;
32             switch (opcion) {
33                 case 1:
34                     int numero1 = 0; //numero 1 a multiplicar
35                     int numero2 = 0; //numero 2 a multiplicar
36                     int suma = 0; //Variable para sumar los el numero 1 tantas veces como indique el numero 2
```

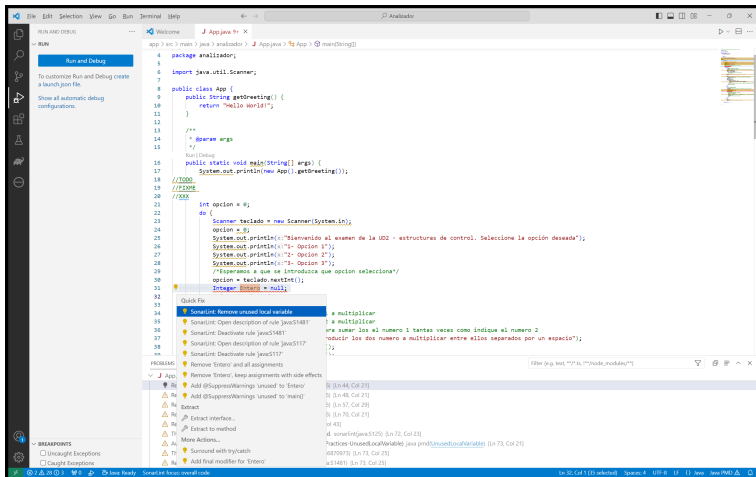
PROBLEMS 33 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, **/*.ts, !...)

▼ J App.java app/src/main/java/analizador 33

- Replace this use of System.out by a logger. sonarlint(java:S106) [Ln 44, Col 21]
- ▲ Replace this use of System.out by a logger. sonarlint(java:S106) [Ln 48, Col 21]
- ▲ Replace this use of System.out by a logger. sonarlint(java:S106) [Ln 57, Col 29]
- ▲ Replace this use of System.out by a logger. sonarlint(java:S106) [Ln 70, Col 21]
- ▲ Remove this empty statement. sonarlint(java:S1116) [Ln 71, Col 43]
- ▲ This block of commented-out lines of code should be removed. sonarlint(java:S125) [Ln 72, Col 23]
- ▲ Avoid unused local variables such as 'Variablelee' (rule: Best Practices-UnusedLocalVariable) java:prmd(UnusedLocalVariable) [Ln 73, Col 21]
- ▲ The use of the local variable Variablelee is not used. java:S3670073 [Ln 73, Col 25]

SonarLint

- Si situamos el cursor y pulsamos en el icono amarillo de la línea en la que se encuentra la advertencia, nos aparecen una serie de recomendaciones. El primer bloque de ellas es de SonarLint



Java-PMD



- Seguimos el mismo proceso de instalación que con SonarLint, pero buscando *Java PMD*
- En la parte inferior de la pantalla podemos ver las advertencias y errores que nos marca
- En la columna de la derecha, podemos diferenciar la herramienta que ha generado cada una de las advertencias

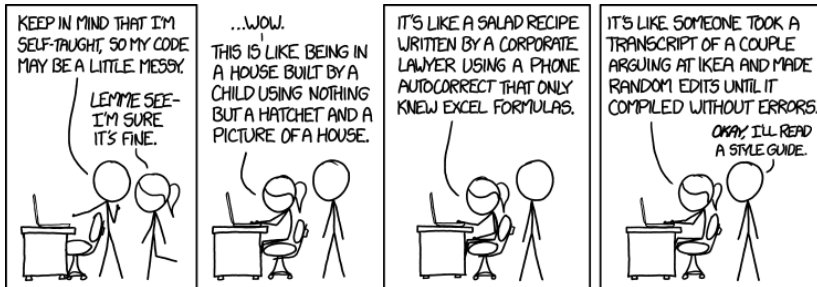
Java-PMD

The screenshot shows an IDE window with a Java file named `App.java`. The code is a simple application with a `main` method. There are several SonarLint annotations (yellow squiggly lines) on the code, including `@param args`, `Run | Debug`, `//TODO`, `//FIXME`, `//XXX`, `Integer entero = null;`, `switch (opcion) {`, `case 1:`, `int numero1 = 0;`, `int numero2 = 0;`, `int suma = 0;`, `for (int i = 0; i < numero2; i++) {`, `suma += numero1;`, and `break;`.

The **PROBLEMS** panel at the bottom shows two errors from PMD:

Code	Message	File	Source
java pmd/LocalVariableNaming...	The local variable name 'Entero' doesn't match '[a-z][a-zA-Z0-9]*' (rule: Code Style-LocalVariableNamingConventions)	app/src/main/java/analizador/App.java [Ln 31, Col 13]	java-pmd
java pmd/LocalVariableNaming...	The local variable name 'Variablees' doesn't match '[a-z][a-zA-Z0-9]*' (rule: Code Style-LocalVariableNamingConventions)	app/src/main/java/analizador/App.java [Ln 73, Col 21]	java-pmd

Léete las guías de estilo...



UD 4: Análisis de Código

Contornos de desenvolvimento (CODE)

Eduardo Mosqueira (UDC) & Víctor Blanco

Curso 2023-2024



XUNTA
DE GALICIA

IES Muralla Romana
Dpto. de Informática