

# Software Architecture

Lab. 08

TDD: Test-driven development

Cobertura de código(Codecov)

Integración continua (GitHub Actions)

Herramientas para el análisis estático (SonarCloud)

2021-22

Jose Emilio Labra Gayo  
Pablo González  
Irene Cid  
Hugo Lebreo

# TDD - Introducción

- Proceso de desarrollo de código donde los requisitos se convierten en casos de test que se pueden probar
- Surge como respuesta al desarrollo de código donde los test se dejaban en la fase final tras el desarrollo.
- Técnica propuesta por Kent Beck

# TDD - Fases

1. Añadir un caso de prueba



2. Ejecutar casos de prueba

- En caso de que uno falle



3. Re-escribir el código



4. Ejecutar todas las pruebas



5. Refactorización del código

# TDD - Características

- Código sencillo que satisface las necesidades del cliente
- Obtenemos código sencillo
- ....Y nuestra batería de pruebas
- Nos ayuda a centrarnos en lo que queremos desarrollar

# Codecov

- Herramienta de cobertura de código
- Cobertura de código: Medida que nos indica la proporción de líneas de código que son probadas en alguno de nuestros test
- Codecov usa la siguiente :
  - Hit: La línea fue ejecutada
  - Partial: La línea fue parcialmente ejecutada, por ej.estructuras condicionales
  - Miss:La línea no fue ejecutada

# Codecov

- La ratio de cobertura es calculado con la siguiente fórmula:
- $\text{hits} / (\text{hits} + \text{misses} + \text{partials})$
- Tras la ejecución de test, nos genera un fichero para su posterior análisis

[https://codecov.io/gh/arquisoft/dede\\_???](https://codecov.io/gh/arquisoft/dede_???)

# TDD - Test de ejemplo

- Comprobación del funcionamiento del componente  
UserList:

- Creamos una lista de usuarios
- Se la añadimos al componente

Comprobamos que el nombre es renderizado en el componente

```
1  import React from 'react'
2  import { render } from "@testing-library/react";
3  import UserList from "../UserList";
4  import {User} from "../shared/sharedddtypes";
5
6  test('check that the list of users renders properly', async () => {
7      const userList:User[] = [{name: 'Pablo', email: 'gonzalezgpablo@uniovi.es' }];
8      const {getByText} = render(<UserList users={userList}/>);
9      expect(getByText(userList[0].name)).toBeInTheDocument();
10     expect(getByText(userList[0].email)).toBeInTheDocument();
11 });
```

# TDD - Test de ejemplo

- Comprobamos que el componente EmailForm funciona bien:
  - Algunas veces tenemos que moquear parte de la prueba
  - Si no mockeamos en este caso, dependemos de los resultados de la restapi
  - Como se trata de test unitarios debemos eliminar esta dependencia



```
6  jest.mock('../api/api');
7
8  test('check register fail', async () => {
9    jest.spyOn(api, 'addUser').mockImplementation((user:User):Promise<boolean> => Promise.resolve(false))
10   await act(async () => {
11     const {container, getByText} = render(<EmailForm OnUserListChange={()=>{}}/>)
12     const inputName = container.querySelector('input[name="username"]')!;
13     const inputEmail = container.querySelector('input[name="email"]')!;
14     fireEvent.change(inputName, { target: { value: "Pablo" } });
15     fireEvent.change(inputEmail, { target: { value: "gonzalezgpablo@uniovi.es" } });
16     const button = getByText("Accept");
17     fireEvent.click(button);
18   });
19 })
```



# Integración Continua - Definición

- Práctica de desarrollo que exige a los desarrolladores integrar el código varias veces al día.
- Cada tarea para generar el software con las nuevas modificaciones es ejecutado cuando se cumple alguna condición (Cada vez que se genera una instancia, un push o un pull en el repositorio)

# Integración Continua - Mejoras

- Detecta y resuelve problemas de una manera continua
- Siempre una versión disponible
- Ejecución automática de los casos de test
- Monitorización de la calidad de código.
- Despliegue automático
- Monitorización de la calidad de código

# integración Continua - ejemplos

- Jenkins
- Pipeline
- Hudson
- Apache Continuum
- Travis
- GitHub Actions

# Integración Continua - Usos

- Mantenimiento del código en el repositorio.
- Construcción automática
- Despliegue
- Ejecutar los test en un entorno clonado en los entornos de producción
- Mantener el histórico de las construcciones.

# GitHub Actions

- Permite gestionar la integración continua sobre los proyectos de los repositorios en GitHub
- Gratis para proyectos gratuitos
- La configuración se mantiene en uno o varios ficheros yaml dentro del directorio **.github/workflows** , que podemos localizar en la raíz del directorio

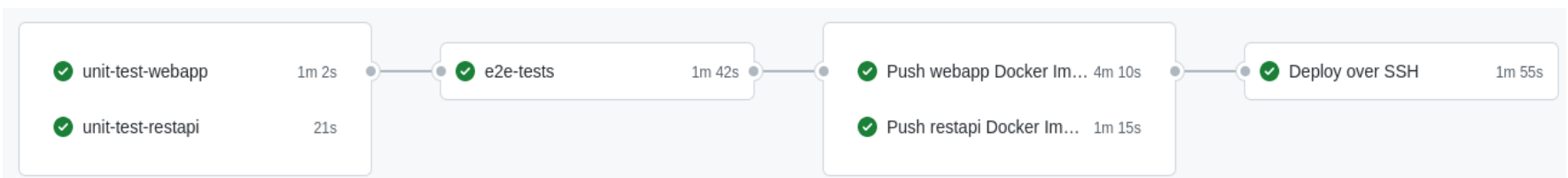
# GitHub Actions

- Contenido .yaml :
  - Condiciones que lanzan el proceso (On )
  - Lista de tareas (Jobs)
    - Cada tarea ejecutada en su propio entorno
  - Una especificacion para cada tarea (checkout, install dependencies, build and test)

```
name: CI for ASW2122

on:
  release:
    types: [published]

jobs:
  unit-test-webapp:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: webapp
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: 16
      - run: npm ci
      - run: npm test
      - uses: codecov/codecov-action@v2
```



# GitHub Actions

- Cada tarea debe tener un propósito específico. (probar una parte de la app, desplegar, .. etc).
- Se puede usar para automatizar otras partes del repositorio. Ejemplo: responder automáticamente cuando un nuevo issue es creado.

# GitHub Actions

- -
- - *uses: actions/checkout@v2.*
  - Uso de una acción ya creada por la comunidad.
  - In this case, realiza un checkout de la rama especificada y se la pasa al Runner
- - *uses: actions/setup-node@v1*  
*with:*
  - node-version: 12.14.1*
  - Instala node en el Runner
- - *run: npm ci*
  - Ejecuta un commando, en este caso instalamos las dependencias de l Proyecto vía npm



# GitHub Actions

- Tambien tenemos jobs para crear imágenes de docker y publicarlas
- Comprueba la [documentation](#) para más configuraciones

```
docker-push-webapp:
  name: Push webapp Docker Image to GitHub Packages
  runs-on: ubuntu-latest
  needs: [e2e-tests]
  steps:
    - uses: actions/checkout@v2
    - name: Publish to Registry
      uses: elgohr/Publish-Docker-Github-Action@3.04
      env:
        API_URI: http://${{ secrets.DEPLOY_HOST }}:5000/api
      with:
        name: pglez82/asw2122_0/webapp
        username: ${{ github.actor }}
        password: ${{ secrets.DOCKER_PUSH_TOKEN }}
        registry: ghcr.io
        workdir: webapp
        buildargs: API_URI
```

# Análisis estático del código

- Analiza el código sin compilarlo
- Detecta bugs, code smells, vulnerabilidades del sistema, etc
- Util para medir la calidad del código.
- Se puede bloquear la subida de código que no cumpla con ciertas características de calidad

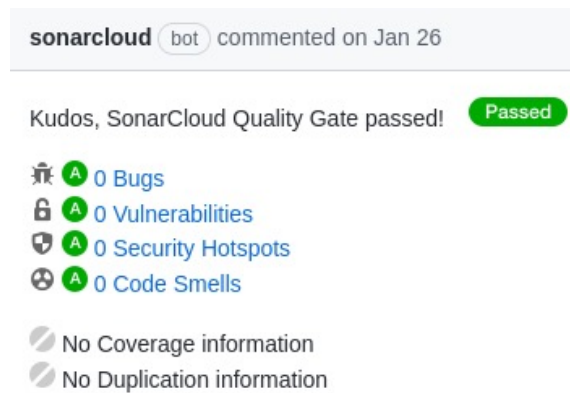
# SonarCloud



- Herramienta para el análisis estático del código
- Necesita:
  - Git server como GitHub
  - Acceso al repositorio
  - Un lenguaje aceptado
- Dos clases de configuración de los análisis:
  - **Automated Analysis** (Default). Cobertura de código no disponible. Scanner del código en servidor sonar.
  - **CI-based analysis**. Sonar scanner ejecutado externamente. Los report son enviados a SonarCloud.

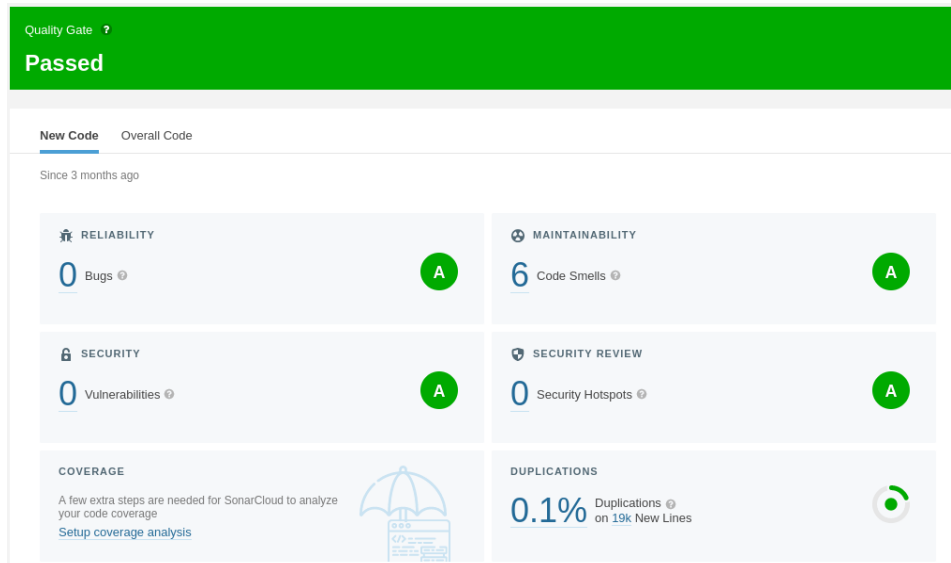
# SonarCloud - dede\_0 configuration

- Cuando los cambios son enviados al repositorio (example, a new pull request)
- Recuperamos información del análisis de código



# SonarCloud

- Servicio de Sonarqube en la nube.
- En el DashBoard podemos comprobar el último análisis de la rama principal pull request and specific branches



asw2122\_0

PUBLIC

---

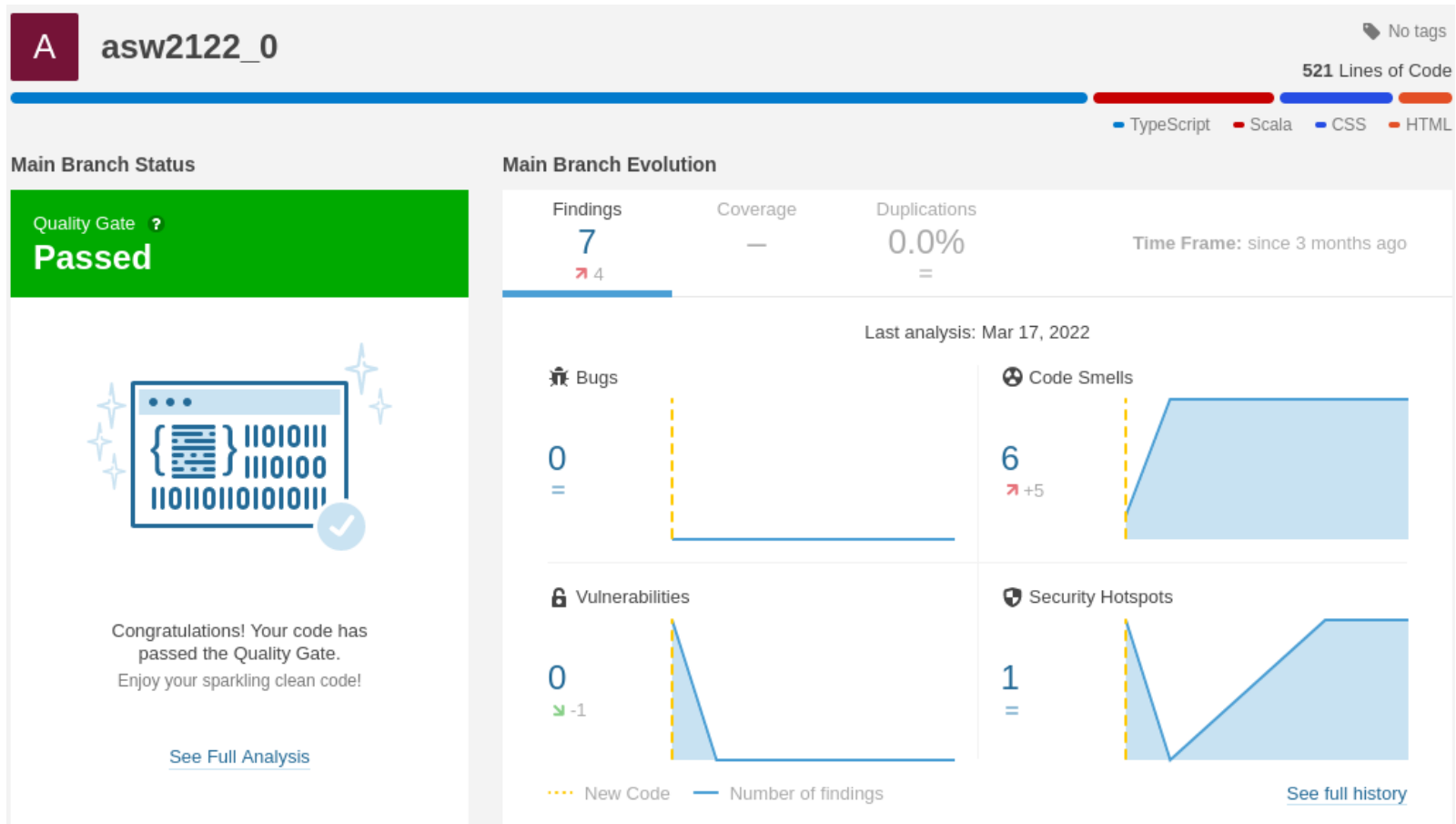
Overview

Main Branch

Pull Requests
0

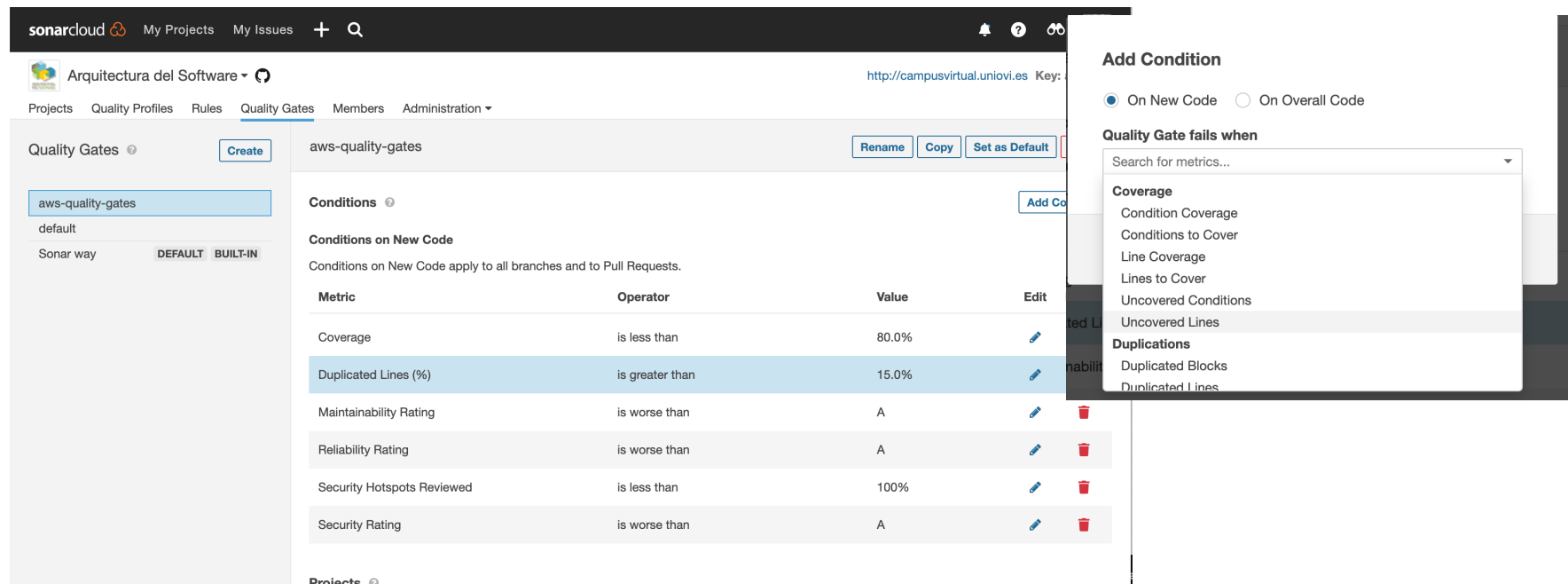
Branches
1

# SonarCloud: Evolución de la calidad de código



# SonarCloud: Umbral de calidad

- En el nivel de la organización definimos distintos umbrales de calidad para asignarlos a los proyectos.



The screenshot shows the SonarCloud interface for configuring quality gates. The main table lists conditions for the 'aws-quality-gates' profile:

Metric	Operator	Value	Edit
Coverage	is less than	80.0%	
Duplicated Lines (%)	is greater than	15.0%	
Maintainability Rating	is worse than	A	
Reliability Rating	is worse than	A	
Security Hotspots Reviewed	is less than	100%	
Security Rating	is worse than	A	

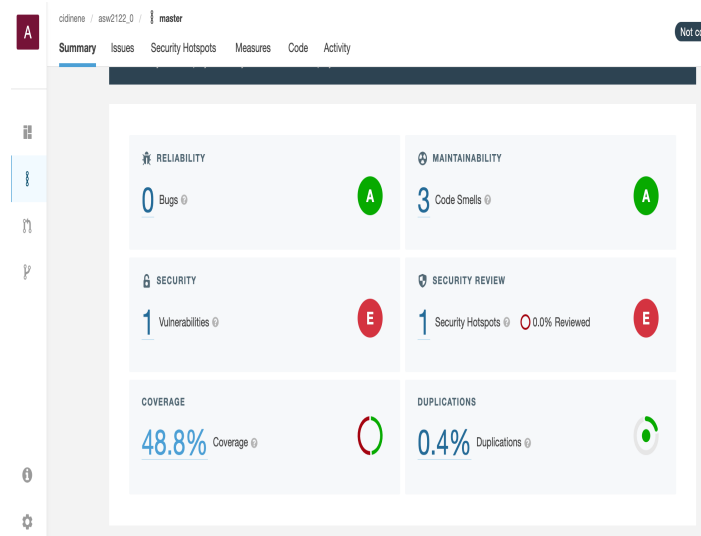
The 'Add Condition' modal is open, showing a search for metrics and a list of options including 'Duplicated Lines'.

Example AWS-Quality-Gates , we increase the procentage of duplicate lines that can be found before launch exception

# SonarCloud: Umbral de calidad

- Lo que se conoce como **Quality Gate** es la definición de condiciones que nuestro proyecto debe alcanzar. Estas condiciones requieren distintos aspectos: cobertura de código, análisis estático del código, líneas duplicadas, ..
- **Dede\_o** tiene configurada la calidad de código con codecov. Podemos incluir esta cobertura de código con SonarCloud:

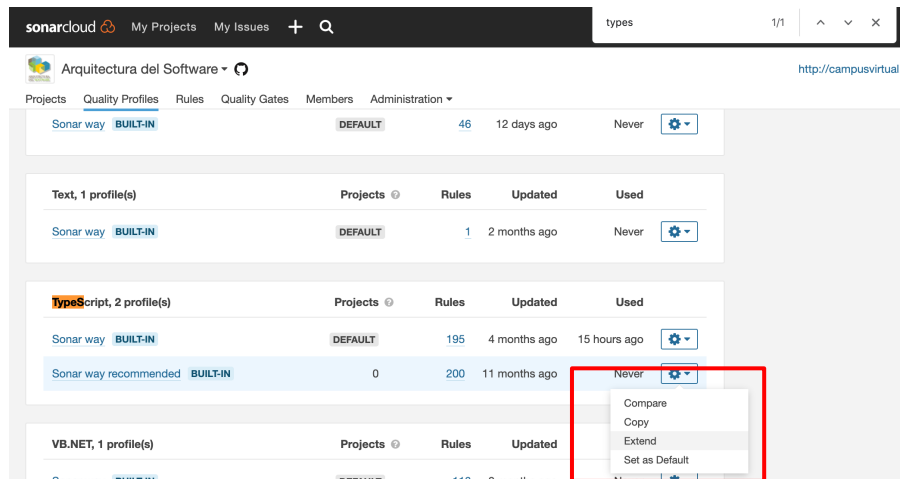
[SONAR COVERAGE SETUP.md](#)



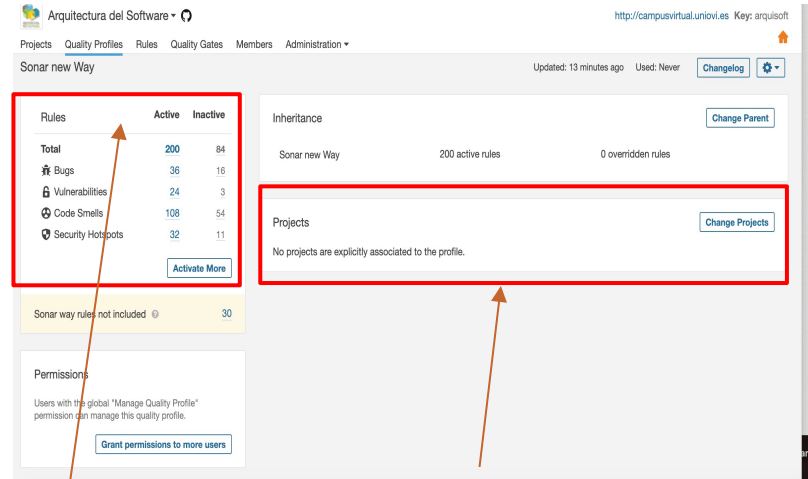


# SonarCloud: Perfiles y reglas

- Las reglas están definidas en los perfiles
- Podemos añadir, desactivar y actualizar reglas creando un nuevo perfil : Copiar un perfil padre – Cambiarlo – asociarlo al proyecto



Create a new profile



Set the profile rules

Associate the profile to the project

# Configuración de reglas

← → ↺


sonarcloud.io/organizations/arquisoft/rules?qprofile=AX-mgR2YnzNFv0H6nzDH&activation=true

🔍 📄 ⭐ ⚙️ 👤 Paused ⋮

sonarcloud

My Projects My Issues + 🔍

type 1/1 ^ v x 🔔 ? 👁 🖨

 Arquitectura del Software

[http://campusvirtual.uniovi.es](#) Key: arquisoft

Projects Quality Profiles Rules Quality Gates Members Administration

Filters

Clear All Filters

Bulk Change

↑ ↓ to select rules ← → to navigate ↺ 1 / 200 rules

🔍 Search for rules...

Language

Type

- Bug 36
- Vulnerability 24
- Code Smell 108
- Security Hotspot 32

Tag

Repository

Default Severity

Status

Security Category

Available Since

Quality Profile SONAR N... Clear

Inheritance

⬆

"===" and "!==" should be used instead of "==" and "!="

TypeScript

Code Smell

suspicious

⬇

Deactivate

⬆

"arguments.caller" and "arguments.callee" should not be used

TypeScript

Code Smell

obsolete

⬇

Deactivate

⬆

"await" should not be used redundantly

TypeScript

Code Smell

redundant

⬇

Deactivate

⬆

"await" should only be used with promises

TypeScript

Code Smell

confusing

⬇

Deactivate

⬆

"catch" clauses should do more than rethrow

TypeScript

Code Smell

clumsy, error-ha...

⬇

Deactivate

⬆

"default" clauses should be last

TypeScript

Code Smell

⬇

Deactivate

⬆

"delete" should be used only with object properties

TypeScript

Bug

⬇

Deactivate

⬆

"delete" should not be used on arrays

TypeScript

Code Smell

⬇

Deactivate

⬆

"for in" should not be used with iterables

TypeScript

Code Smell

⬇

Deactivate

⬆

"for of" should be used with Iterables

TypeScript

Code Smell

clumsy

⬇

Deactivate

⬆

"for" loop increment clauses should modify the loops' counters

TypeScript

Code Smell

confusing

⬇

Deactivate

School of Computer Science, University of Oviedo

# Ver las alertas mientras programamos

- <https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarlint-vscode>

