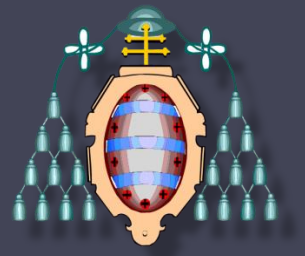
ES
Español



University of Oviedo

School of Computer Science

SOFTWARE ARCHITECTURE

Software Architecture

Lab. 08
TDD: Test-driven development
Cobertura de código(SonarCloud)
Integración continua (GitHub Actions)
Herramientas para el análisis estático (SonarCloud)

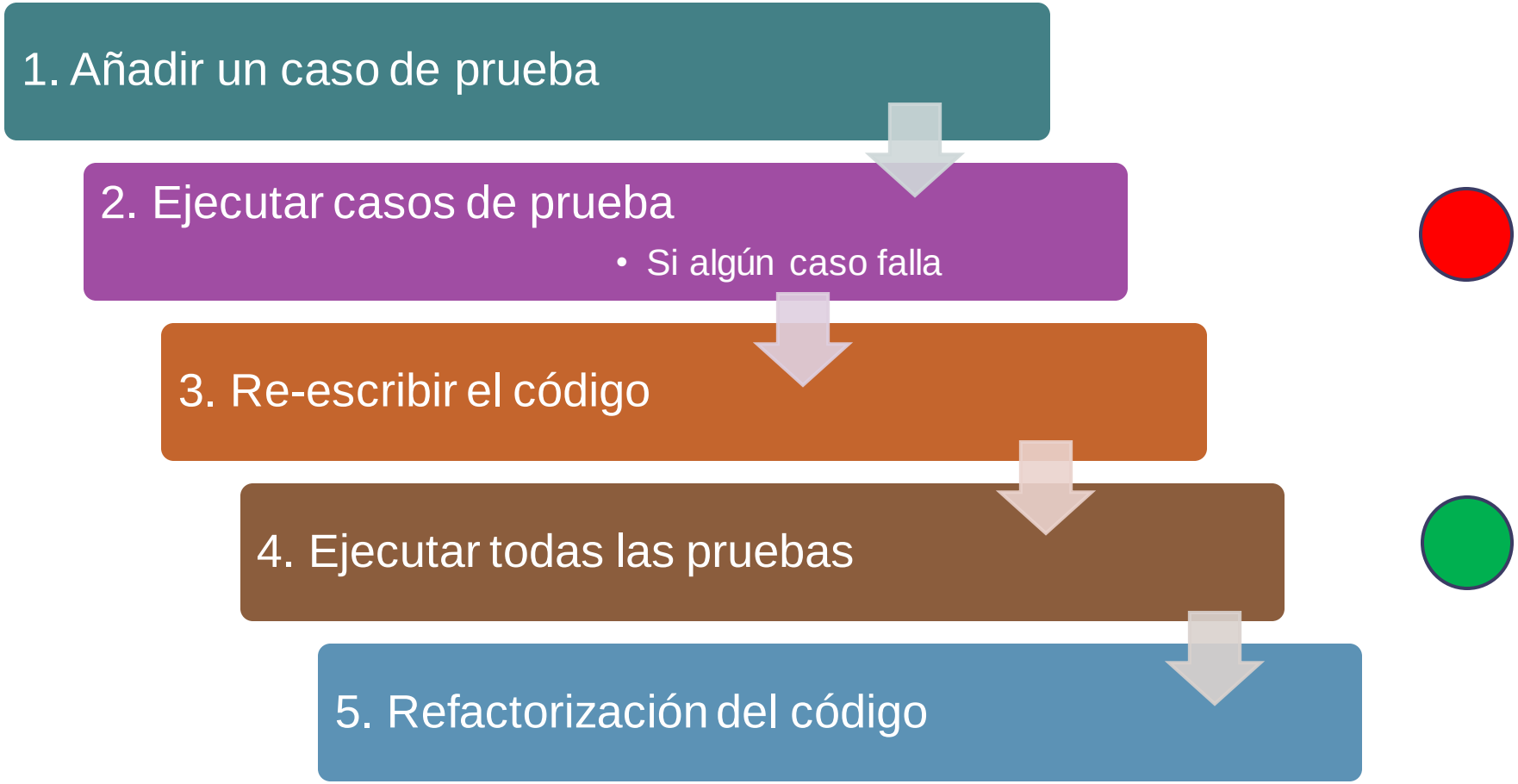
2022-23

Jose Emilio Labra Gayo
Pablo González
Irene Cid
Cristian Augusto

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
• Si algún caso falla
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

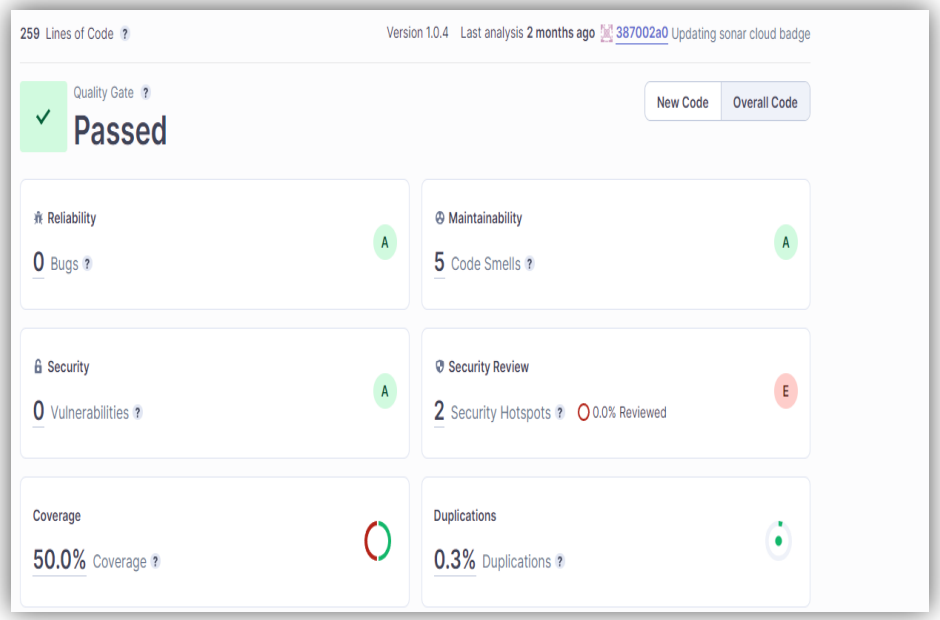
SonarCloud - Coverage

- Herramienta que incluye la cobertura de código como una métrica más en la evaluación del código
- Cobertura de código: Medida que nos indica la proporción de líneas de código que son probadas en alguno de nuestros test
- Sonar Cloud recoge los datos de los resultados del lanzamiento de los test, recogiendo los siguientes valores:
  - LC = Líneas cubiertas (lines_to_cover - uncovered_lines)
  - EL = Número total de líneas ejecutables (lines_to_cover)

SonarCloud

- La ratio de cobertura es calculado con la siguiente fórmula:
  - LC/EL
- Tras la ejecución de test, nos genera un fichero para su posterior análisis

https://sonarcloud.io/summary/overall?id=Arquisoft_lomap_???



TDD – Test de ejemplo

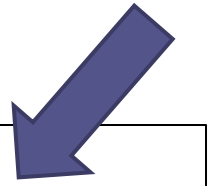
- Comprobación del funcionamiento del componente UserList:
  - Creamos una lista de usuarios
  - Se la añadimos al componente

  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/shareddtypes";
5
6   test('check that the list of users renders propertly', 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 componenete 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 promueve la integración del código varias veces al día.
- El lanzamiento del proceso de integración contínua 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 Continuun
- 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

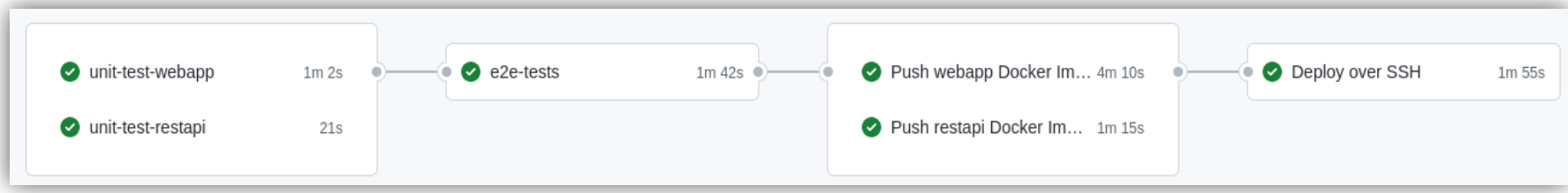
GitHub Actions

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

```yaml
name: CI for LOMAP_0

on:
  release:
    types: [published]

jobs:
  unit-test-webapp:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: webapp
    steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-node@v3
      with:
        node-version: 18
    - run: npm ci
    - run: npm test --coverage --watchAll
    - name: Analyze with SonarCloud
      uses: sonarsource/sonarcloud-github-action@master
      env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```



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 del Proyecto vía npm

- *run: npm test*

  Ejecuta las pruebas unitarias. Si alguna falla, la integración continua fallará

GitHub Actions

- Tambien tenemos jobs para crear imágenes de docker y publicarlas
- Comprueba la documentation para más configuraciones

```yaml
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

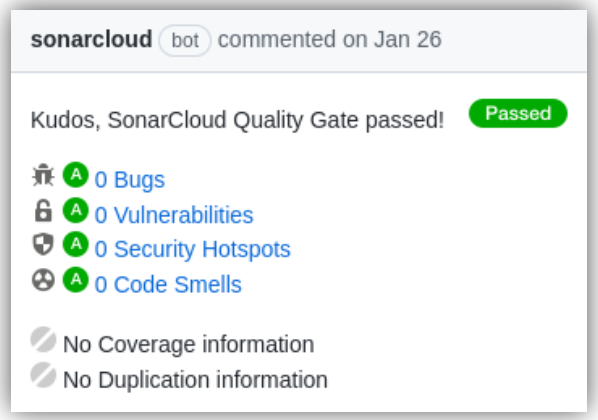
- Analiza el código sin compilarlo
- Detecta bugs, code smells, vulnerabilidades del sistema, etc
- Util para medir la calidad del código.
- Se puede bloquear la subida de código que no cumpla con ciertas caracetrísticas de calidad

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.

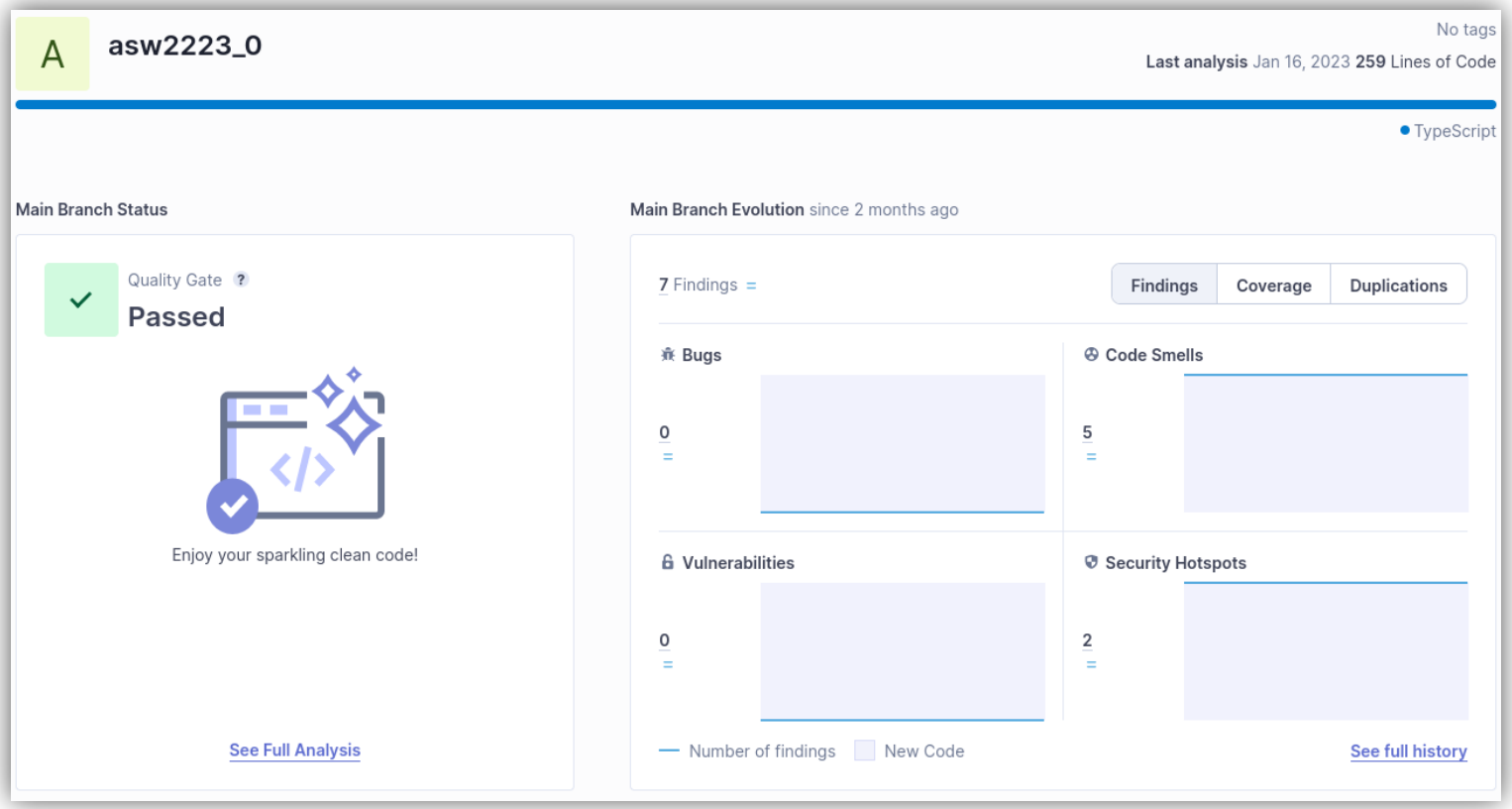
SonarCloud - lomap_0 configuration

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

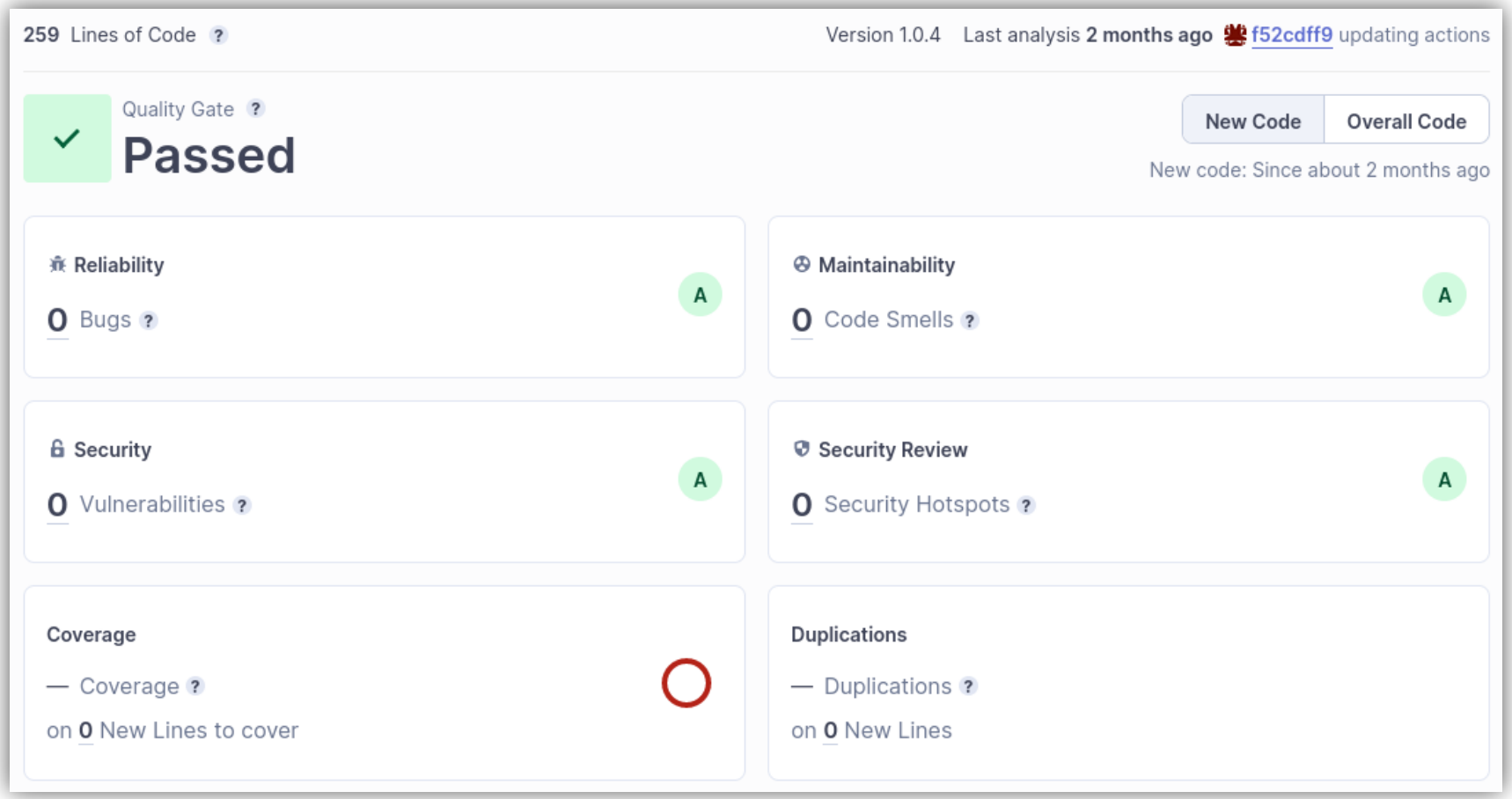


SonarCloud

- En: Project Dashboard podemos chequedar el ultimo análisis de la rama principal, las pull-requests y las ramas específicas

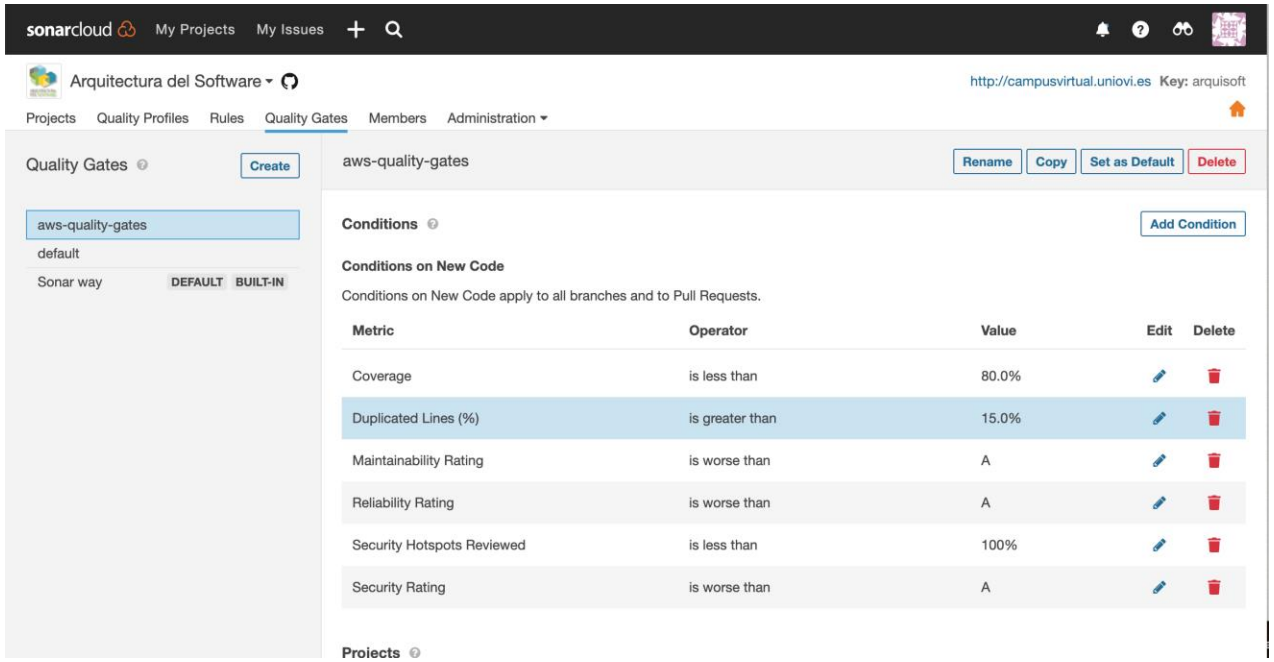


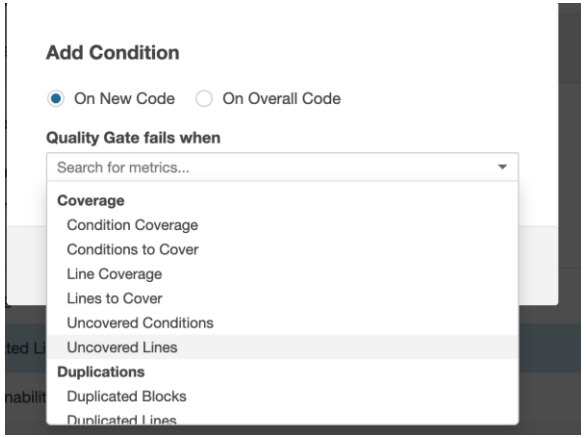
SonarCloud: Evolución de la calidad del proyecto



SonarCloud: Umbral de calidad

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

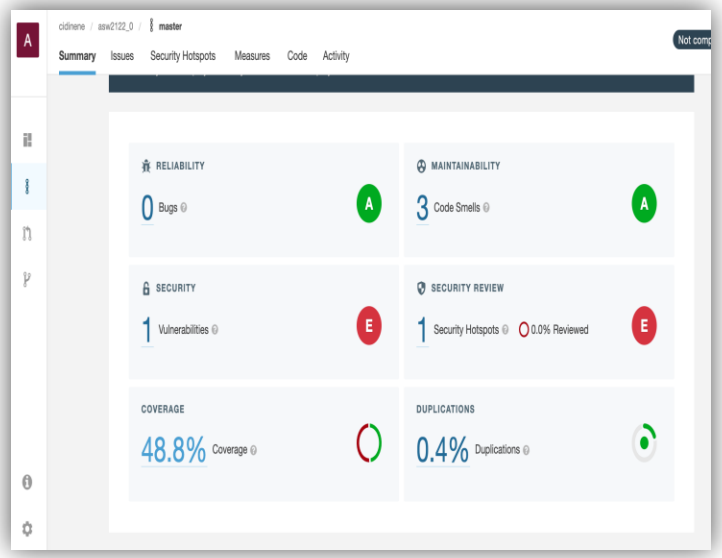




Ejemplo AWS-Quality-Gates , se puede incrementar el porcentaje de líneas duplicadas que se pueden encontrar antes de lanzar una excepción

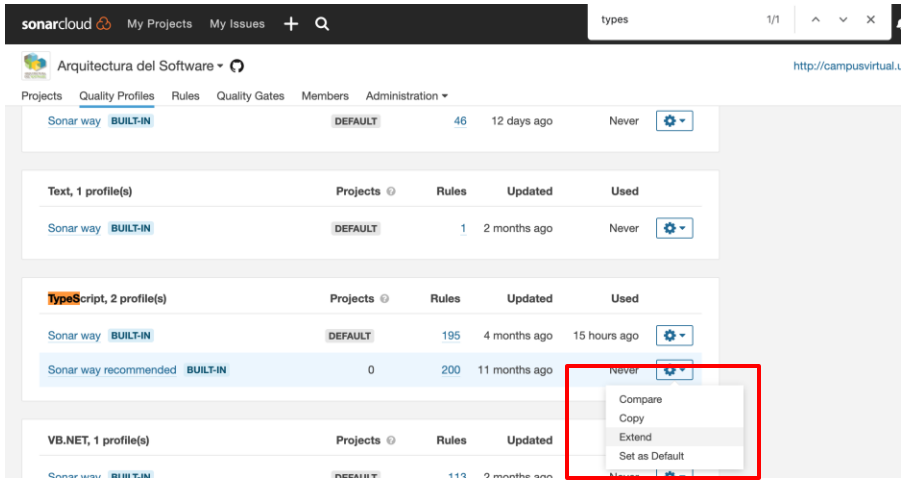
SonarCloud: Umbral de calidad

- 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, ..
- **lomap_0** tiene configurada la calidad de código con sonarcloud.

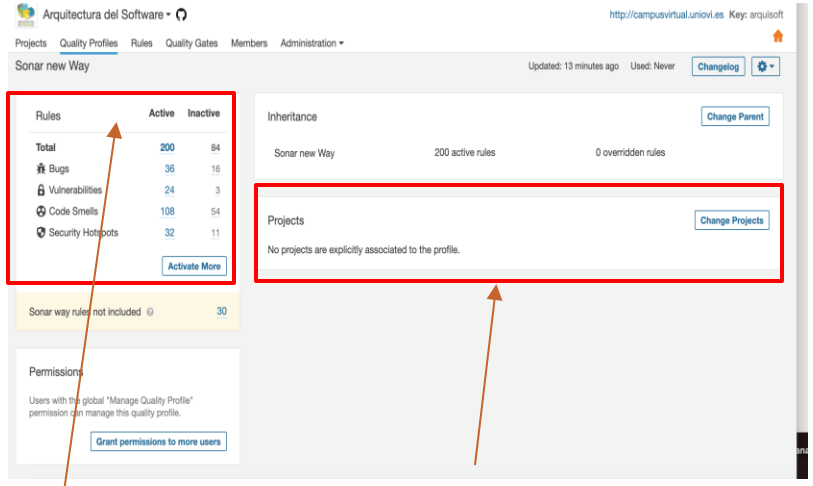


SonarCloud: Perfiles y reglas

- 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



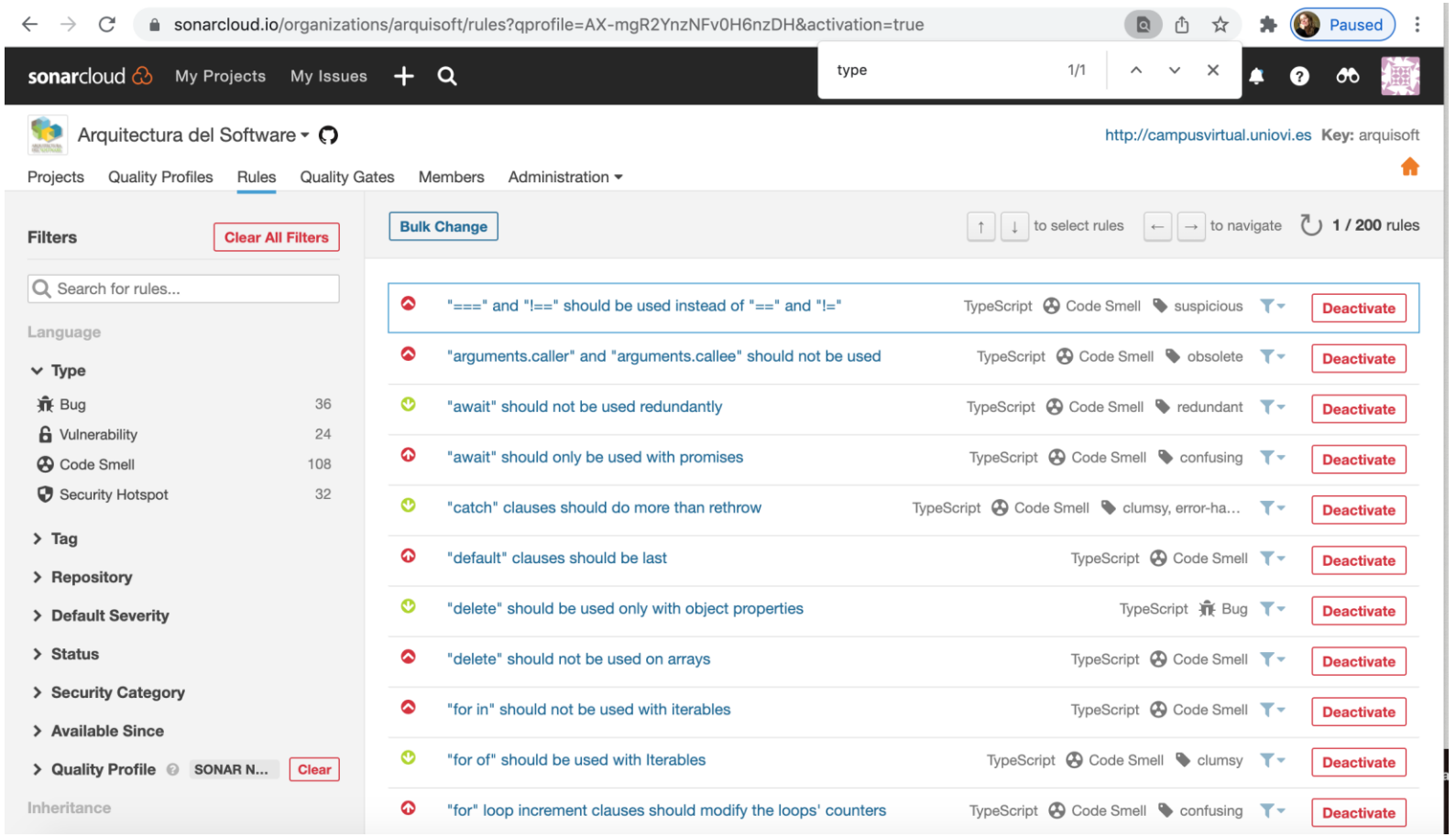
Create a new profile



Set the profile rules

Associate the profile to the project

Configuración de reglas



Ver las alertas mientras programamos

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