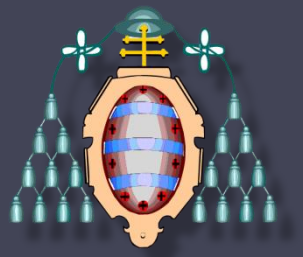
ES
Español



University of Oviedo

School of Computer Science

SOFTWARE ARCHITECTURE

Arquitectura del Software

Lab. 08

- TDD: Desarrollo guiado por pruebas (Test Driven Development)
- Cobertura de Código (Codecov)
- Integración Continua(GitHub Actions)
- Herramientas de análisis estático del código(Codacy)

Jose Emilio Labra Gayo
Pablo González
Irene Cid
Paulino Álvarez

2020-21

TDD - Definición

- Proceso de software en el que los requisitos son trasladados a pruebas.
- Lo opuesto al desarrollo de software donde se permite que código no testeado se despliegue
- 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 - Particularidades

- 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

TDD -Codecov

- Herramienta de cobertura de código
- Cobetura 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: The line was executed
  - Partial: The line was partially executed. For example conditional structures
  - Miss: The lines was not executed

TDD - Codecov

- La ratio de cobertura es calculado con la siguiente fórmula:

  hits / (hits + misses + partials)

- Tras la ejecución de test, nos genera un fichero para su posterior análisis

https://codecov.io/gh/arquisoft/radarin_ ???

TDD- Test de ejemplo

```
export default function EmailForm(props) {
 const [state, setState] = useState({email: '', remail: '', enabled: false});

 function changeEmail(e) {
     const email = e.target.value ;
     setState({...state, email: email, enabled: email === state.remail});
 }

 function changeRemail(e) {
     const remail = e.target.value ;
     setState({...state, remail: remail, enabled: remail === state.email});
  }

 return (
   <Form>
       <Form.Control type="text" name="email" placeholder="Input email" aria-label="email-input"
                     onChange={changeEmail} value={state.email}/>
       <Form.Control type="text" name="remail" placeholder="Input remail" aria-label="remail-input"
                     onChange={changeRemail} value={state.remail}/>
       <Button variant="primary" type="submit" disabled={!state.enabled}>Submit</Button>
   </Form>
 )
}
```

Tenemos un formulario con dos cajas de texto: email y remail
El botón no se activa hasta que se introduzca el mismo texto en ambas entrada

TDD -Test de ejemplo

```
import React from 'react'
import { render, fireEvent } from "@testing-library/react";
import EmailForm from "./EmailForm";

test('check email button activated when 2 emails are equal', async () => {
  const correctValues = { email: 'test@example.org', remail: 'test@example.org' };

  const { getByLabelText, getByText, container } = render(<EmailForm/>);

  const inputEmail = getByLabelText('email-input');
  const inputRemail = getByLabelText('remail-input');

  fireEvent.change(inputEmail, { target: { value: correctValues.email }});
  expect(getByText(/Submit/i).closest('button')).toHaveAttribute('disabled');

  fireEvent.change(inputRemail, { target: { value: correctValues.remail }});
  expect(getByText(/Submit/i).closest('button')).not.toHaveAttribute('disabled');

});
```

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
- Ejecución automática de los casos de test
- 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.

Integración Continua - GitHub Actions

- Permite gestionar la integración continua sobre los proyectos de los repositorios en GitHub
- Ratis 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

CI-GitHub Actions

- Contenido .yml :
  - 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)

```yaml
name: CI for radarin

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build-test-webapp:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: webapp
    steps:
    - uses: actions/checkout@v2
    - uses: actions/setup-node@v1
      with:
        node-version: 12.14.1
    - run: npm ci
    - run: npm run build
    - run: npm test
    - uses: codecov/codecov-action@v1
```

Integración Continua -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.

Integración Continua -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

Análisis estático del código - Definición

- 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

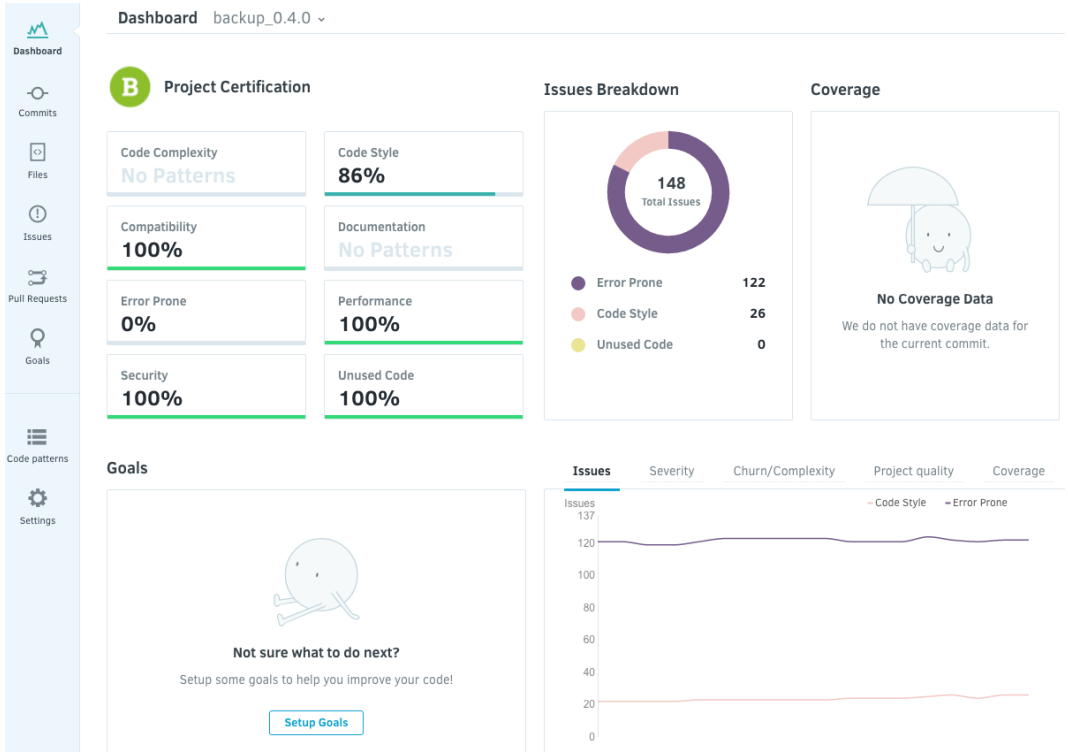
Análisis estático del código -Codacy

- Herramienta de análisis estático del código
- Requisitos:
  - Servidor Git como GitHub
  - Acceso al repositorio
  - Lenguaje de aceptación
- El proyecto es importado y analizado

https://app.codacy.com/gh/Arquisoft/radarin_??)

Análisis estático del código -Codacy

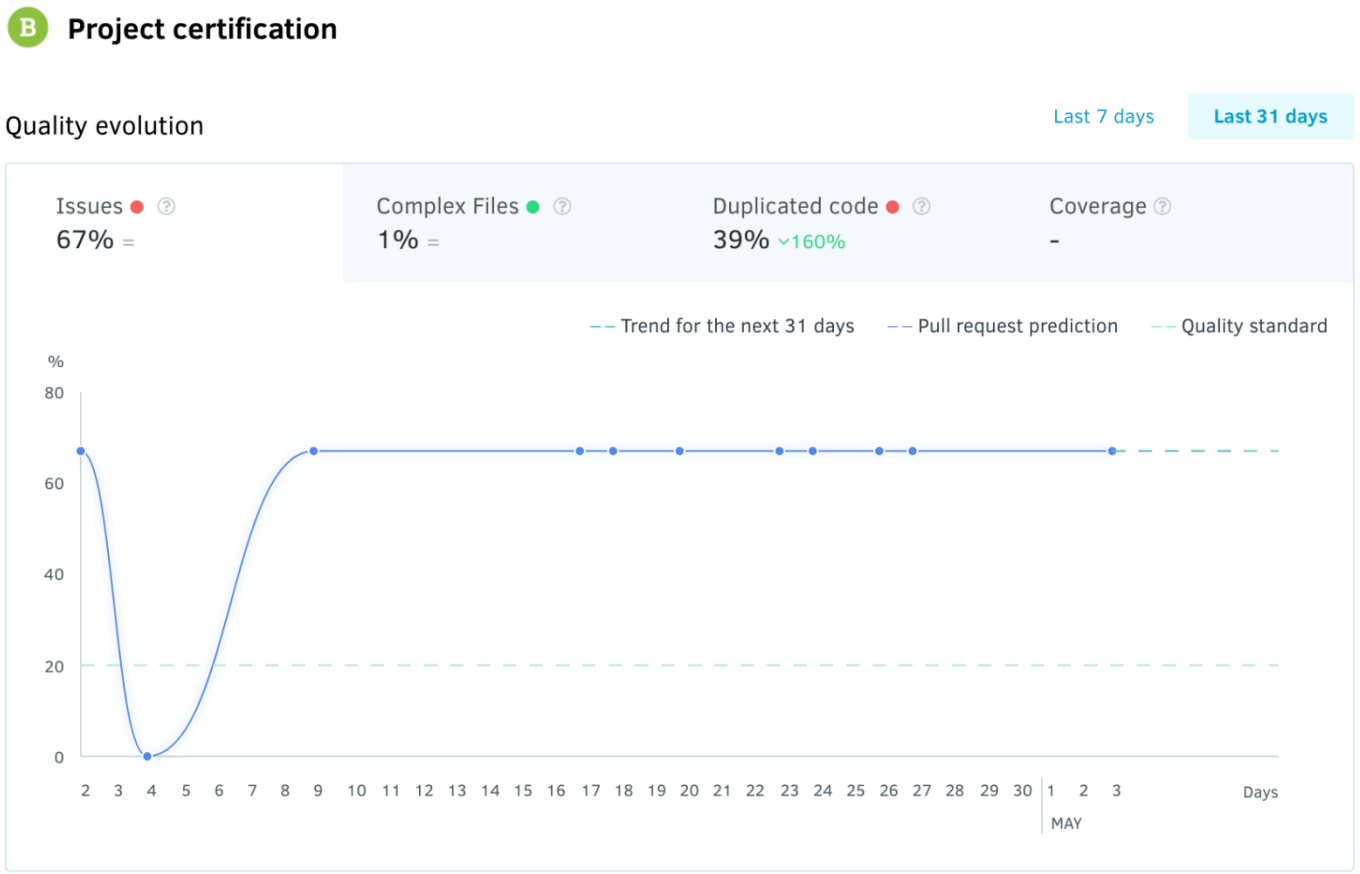
- Tras el análisis, codacy envía el mail



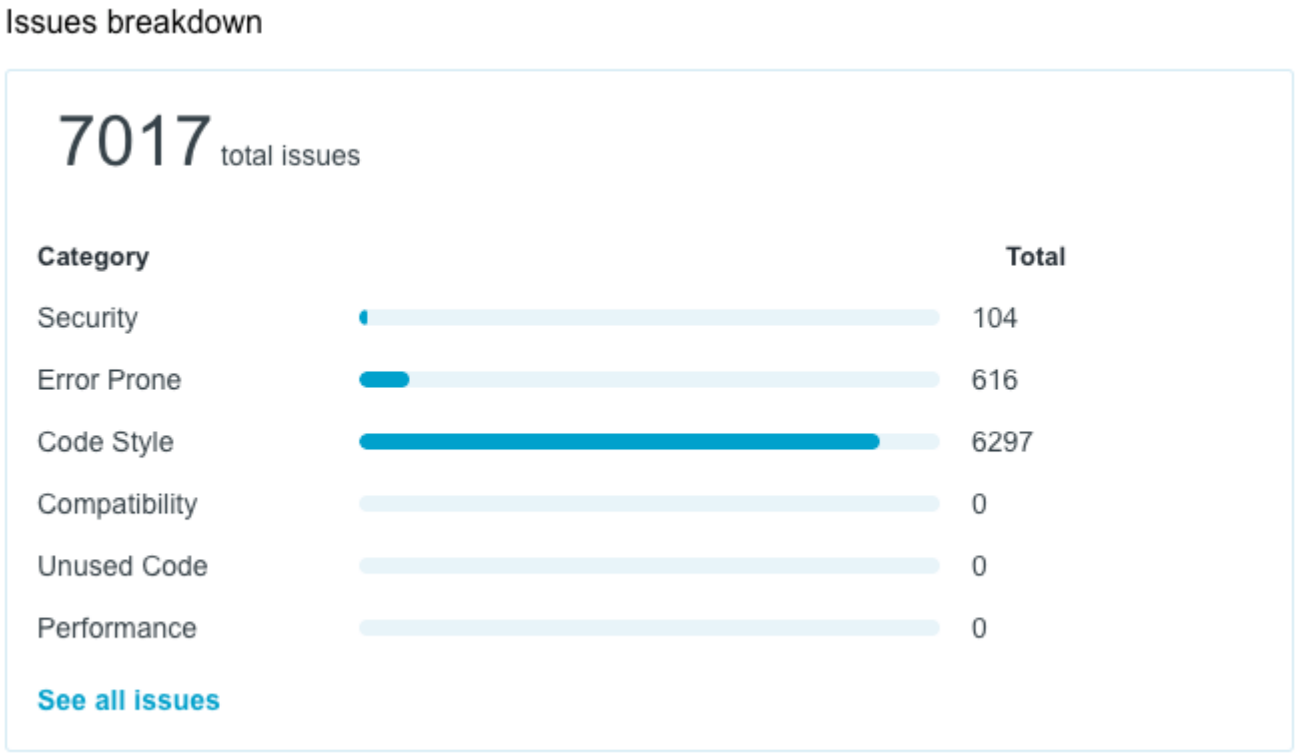
Codacy

- En la página principal del proyecto podemos elegir la rama
- Para cada rama podemos visualizar las siguientes secciones:
  - **Quality evolution**
  - **Issues breakdown**
  - **Coverage status**
  - **Hotspots**
  - **Logs**
  - **Pull requests status**

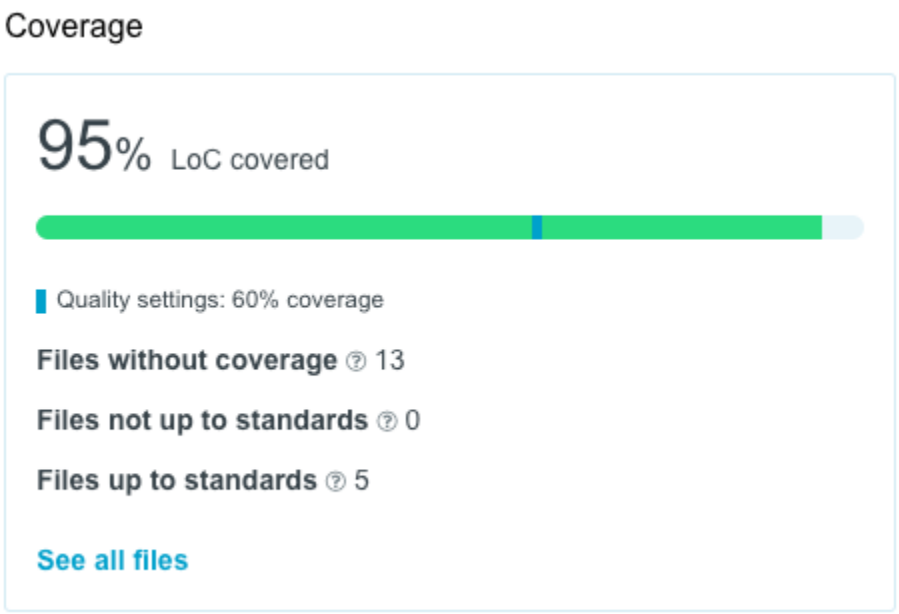
Codacy: Project certification and Quality evolution



Codacy: Issues breakdown



Codacy: Coverage status



Codacy

- **Security:** recomendaciones de seguridad , potenciales vulnerabilidades, dependencias de seguridad.
- **Error Prone:** Malas prácticas, código conducente a fallos.
- **Code Style:** mejores practices en el formato de las líneas de código: líneas demasiada largas, tabulaciones.
- **Compatibility:** identifica código que puede presentar problemas con antiguos sistemas u otras plataformas.
- **Unused Code:** Código no accesible o usado.
- **Performance:** errores de optimización .

Codacy: Files

Files master

Search file

GRADE	FILENAME	ISSUES	DUPLICATION	COMPLEXITY	COVERAGE
A	tests/Codacy/Coverage/Parser/CloverParserTest.php	1	0	4	-
A	src/Codacy/Coverage/Parser/CloverParser.php	1	0	16	94%
B	src/Codacy/Coverage/Application.php	0	0	1	0%
A	tests/Codacy/Coverage/Parser/ParserTest.php	0	0	1	-
A	tests/Codacy/Coverage/Util/GitClientTest.php	0	0	1	-
A	tests/Codacy/Coverage/Parser/PhpUnitXmlParserTest.php	0	0	2	-
B	src/Codacy/Coverage/Command/Phpunit.php	0	0	3	0%
A	src/Codacy/Coverage/Util/GitClient.php	0	0	3	67%
A	src/Codacy/Coverage/Util/CodacyApiClient.php	0	0	4	-

Codacy: File detail