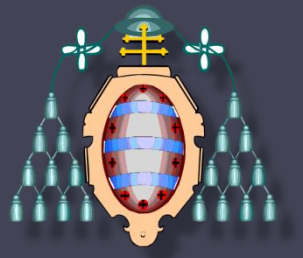
EN
English



University of Oviedo

School of Computer Science

SOFTWARE ARCHITECTURE

Software Architecture

Lab. 08
TDD: Test-driven development
Code coverage(Codecov)
Continuous integration (GitHub Actions)
Tools to static analyze the code (Codacy)

2020-21

Jose Emilio Labra Gayo
Pablo González
Irene Cid
Paulino Álvarez

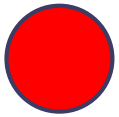
TDD

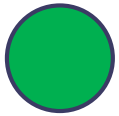
- Software development process where requirements are converted to specific test cases
- The opposite to software development that allows not tested software to be deployed
- Technique proposed by Kent Beck

TDD

Phases:

1. Add a test case
2. Execute test cases -> new one fails
3. Write the code
4. Execute all test cases
5. Code refactor





TDD

- Simple code created to satisfy the test case
- We get clean code as a result
- And a test-suite
- Helps focus to know what we want to implement

Codecov

- Coverage code tool
- Code coverage: Measure to show what code lines has been executed by a test suite
- Some terminology about CodeCov:
  - Hit: Line was executed
  - Partial: Line was not tested fully. Example: an if sentence with only one path tested.
  - Miss: Line was not executed

Codecov

- Coverage ratio is calculated with the following formula

  hits / (hits + misses + partials)

- After the tests, it generates a file that allows to do the analysis

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

TDD – Example test

```
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>
 )
}
```

We have a form with two email inputs (email and remail).
It should be disabled until both inputs are equals

TDD – Example test

```
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');

});
```

Continuous Integration (CI)

- Development practice that requires developers to **integrate** code into a shared repository several times a day
- Every task to build the software is executed when some condition is met (for instance, a push or pull request to master)

Continuous Integration (CI)

- Detect and solve problems continuously
- Always available
- Immediate execution of unit test cases.
- Project quality monitorization.

Continuous Integration (CI)

- Examples:
  - Jenkins
  - Pipeline
  - Hudson
  - Apache Continuun
  - Travis
  - GitHub Actions

Continuous Integration (CI)

- Common usages:
  - Maintenance of the code in a repository
  - Building automation
  - Quick building
  - Execute test cases in a cloned production environment
  - Show results of last build.

GitHub Actions

- Continuous integration service for projects stored in GitHub
- Free for free software projects
- Configuration is in one or multiple YAML files inside the .github/workflows directory that is localized in the root directory of the project

GitHub Actions

- .yml specifies:
  - Conditions for firing the process
  - List of jobs
    - Each executed in a specific environment
  - Steps to carry out the job (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
```

GitHub Actions

- Each job can have a specific purpose (test a part of the app, deploy, etc.)
- GitHub actions can be used to automate other parts of the repository. Example: autoreply to new issues created in the repository

GitHub Actions

- *- uses: actions/checkout@v2.*
  - Uses an action created by the community.
  - In this case, it checks out the project to the runner
- *- uses: actions/setup-node@v1*
  with:
  node-version: 12.14.1
  - Installs node in the runner
- *- run: npm ci*
  - Runs a command (install the dependencies)

Static analysis of the code

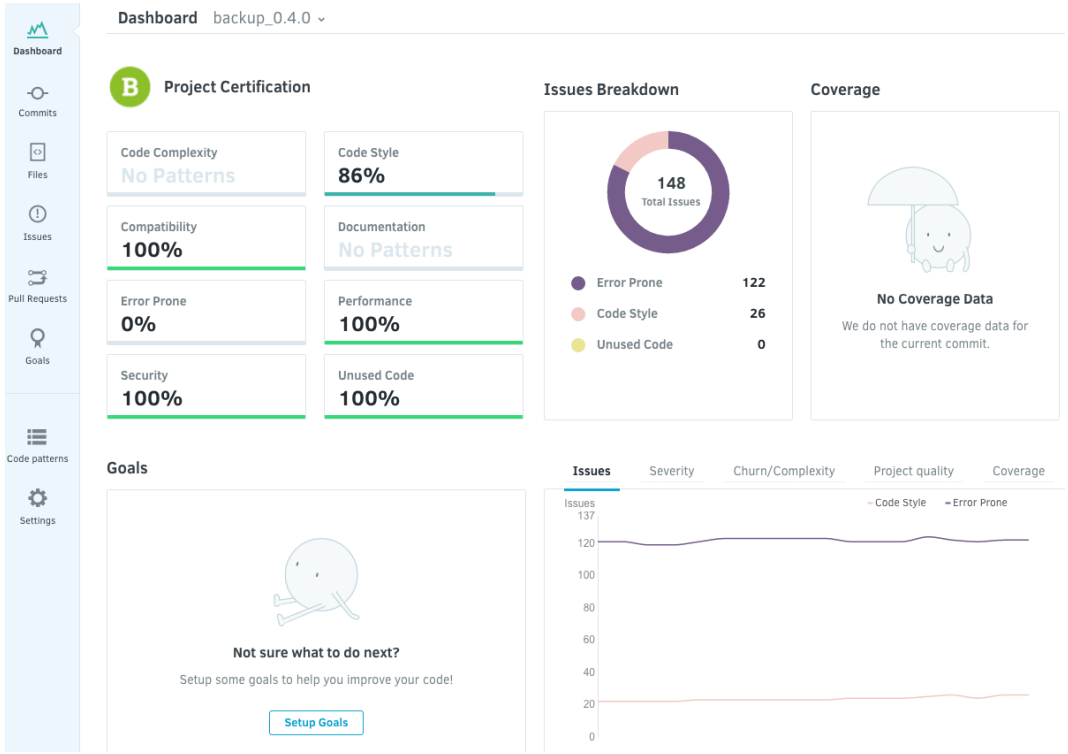
- Analyzed the code without compiling it
- Detects bugs, code smells, system vulnerabilities, etc.
- Useful to control the code quality.
- If the code does not meet the quality requirements, then the commit can be blocked

Codacy

- Static code analysis tool
- It needs:
  - Git server like GitHub
  - Repository access
  - An accepted language
- The Project is imported to Codacy so it can be analyzed

Codacy

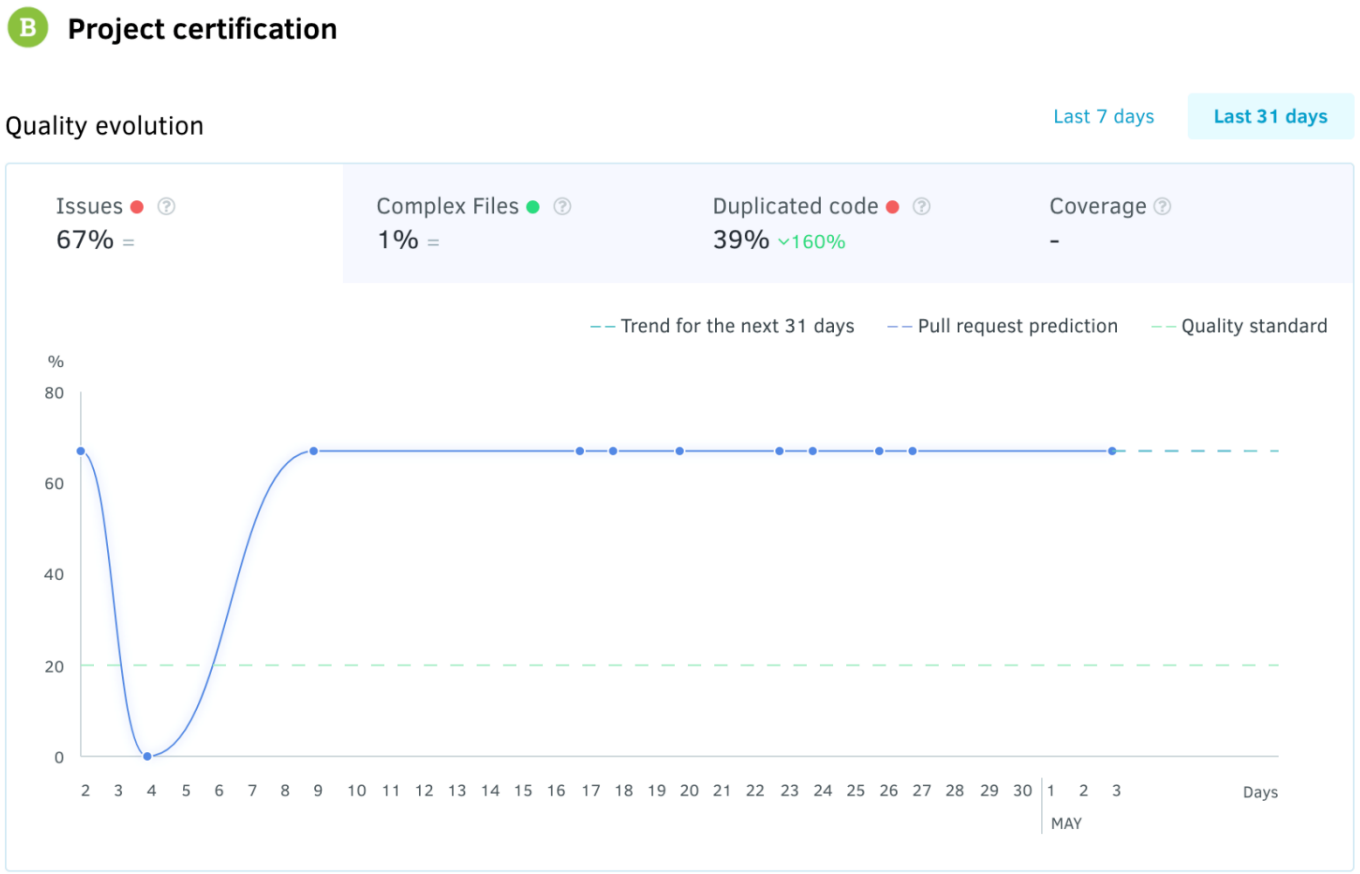
- After the analysis Codacy sends an email



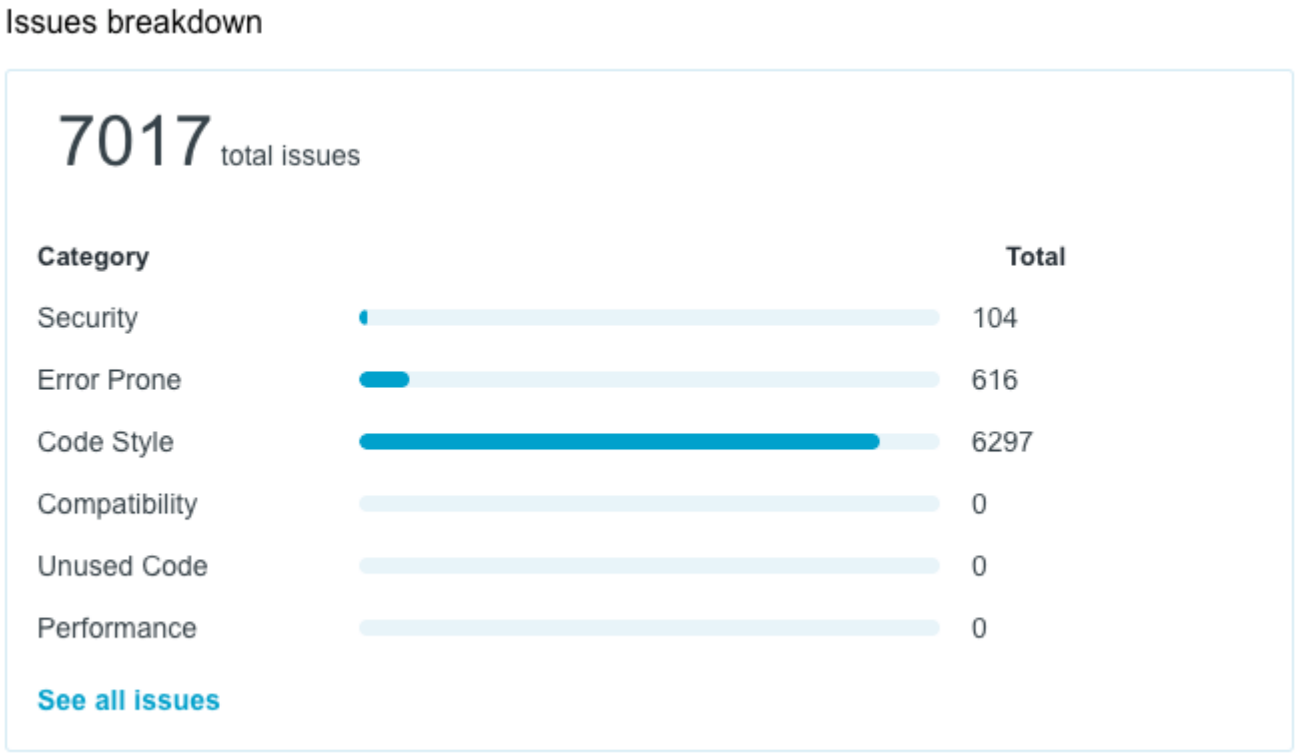
Codacy

- In the Project Dashboard we see two main sections: specific branches and the main one
- For each branch there are the following sections:
  - **Quality evolution**
  - **Issues breakdown**
  - **Coverage status**
  - **Hotspots**
  - **Logs**
  - **Pull requests status**

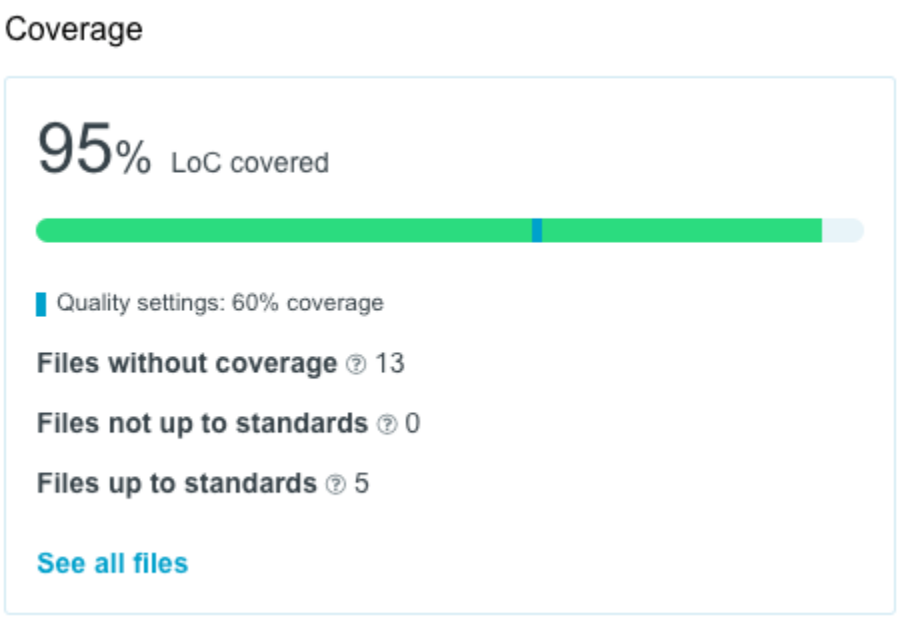
Codacy: Project certification and Quality evolution



Codacy: Issues breakdown



Codacy: Coverage status



Codacy

- **Security:** security issues, potential vulnerabilities, unsafe dependencies.
- **Error Prone:** bad practices/patterns that cause code to fail/prone to bugs.
- **Code Style:** related to the style of the code, line length, tabs vs spaces.
- **Compatibility:** identifies code that has problems with older systems or cross platform support.
- **Unused Code:** unnecessary code not being used.
- **Performance:** inefficiently written code.

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