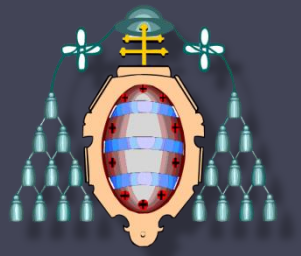
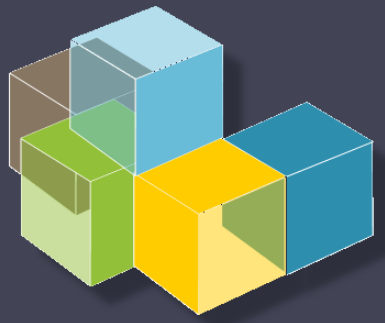
ES
Español



Universidad de Oviedo

Escuela de Ingeniería Informática



ARQUITECTURA DEL SOFTWARE

Software Architecture

Lab. 06
TDD: Test-driven development
Code coverage(Codecov)
Continous integration (Travis)
Tools to static analyze the code (Codacy)

2019-20

Jose Emilio Labra Gayo
Hugo Lebredo
Pablo González
Irene Cid

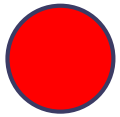
TDD

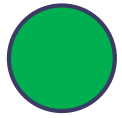
- Software development process where requirements are converted to specific test cases
- The opposite to software development that allows not tested software to be deployed
- Technique proposed by Kent Beck

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
- Codecov uses the following terminology for each line:
  - Hit: The line was executed
  - Partial: The line was partially executed. For example conditional structures
  - Miss: The lines 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/viade_???

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 equal

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');

});
```

Continous integration

- Development practice that requires developers to **integrate** code into a shared repository several times a day
- Every task to build the software is executed with each addition of code

Continous integration

- Detect and solve problems continously
- Always available
- Inmeadiate execution of unit test cases.
- Quality Project monitorization.

Continous integration

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

Continous integration

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

Travis

- Continous integration service for projects stored in GitHub
- Free for free software projects
- Configuration is in a YAML file named .travis.yml that is localized in the root directory of the project

Travis

- .travis.yml specifies:
  - Programming language
  - Building environment
  - Test execution environment
  - How to deploy
  - Other parameters

```yaml
language: node_js
node_js:
  - 12.14.0
cache:
  directories:
  - node_modules
before_install:
  - sudo apt-get update
  - sudo apt-get -y install ruby openjdk-8-jre
  - sudo gem install asciidoctor asciidoctor-diagram
script:
  - npm install -g codecov
  - npm test && codecov
  - npm run build
  - npm run docs
deploy:
  provider: pages
  skip_cleanup: true
  github_token: $github_token
  local_dir: build
  on:
    branch: master
```

Travis

- dist: Distro used for Linux. In macOS and Windows it has the tag os
- sudo: If sudo is activated or not
- language: Programming language
- node_js: Version of nodejs.
- addons: Used extensions
- Directories: Where are the dependencies stored
- Install: To deploy the software

Travis

- Script: Commands executed after the project is deployed
- after_success: If no error has ocurred, then these statements are executed

Static analysis of the code

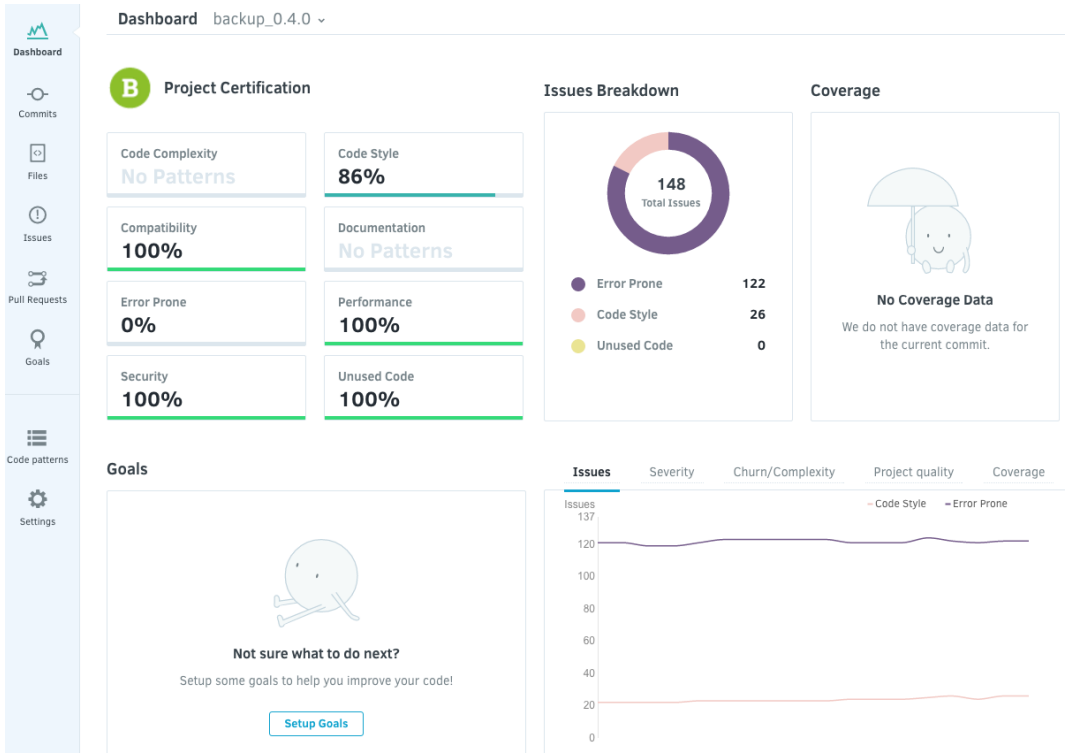
- Analyzed the code without compiling it
- Detects bugs, code smells, system vulnerabilities, etc
- Useful to control the code quality.
- If the code does not meet the quality requirements then the commit can be blocked

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

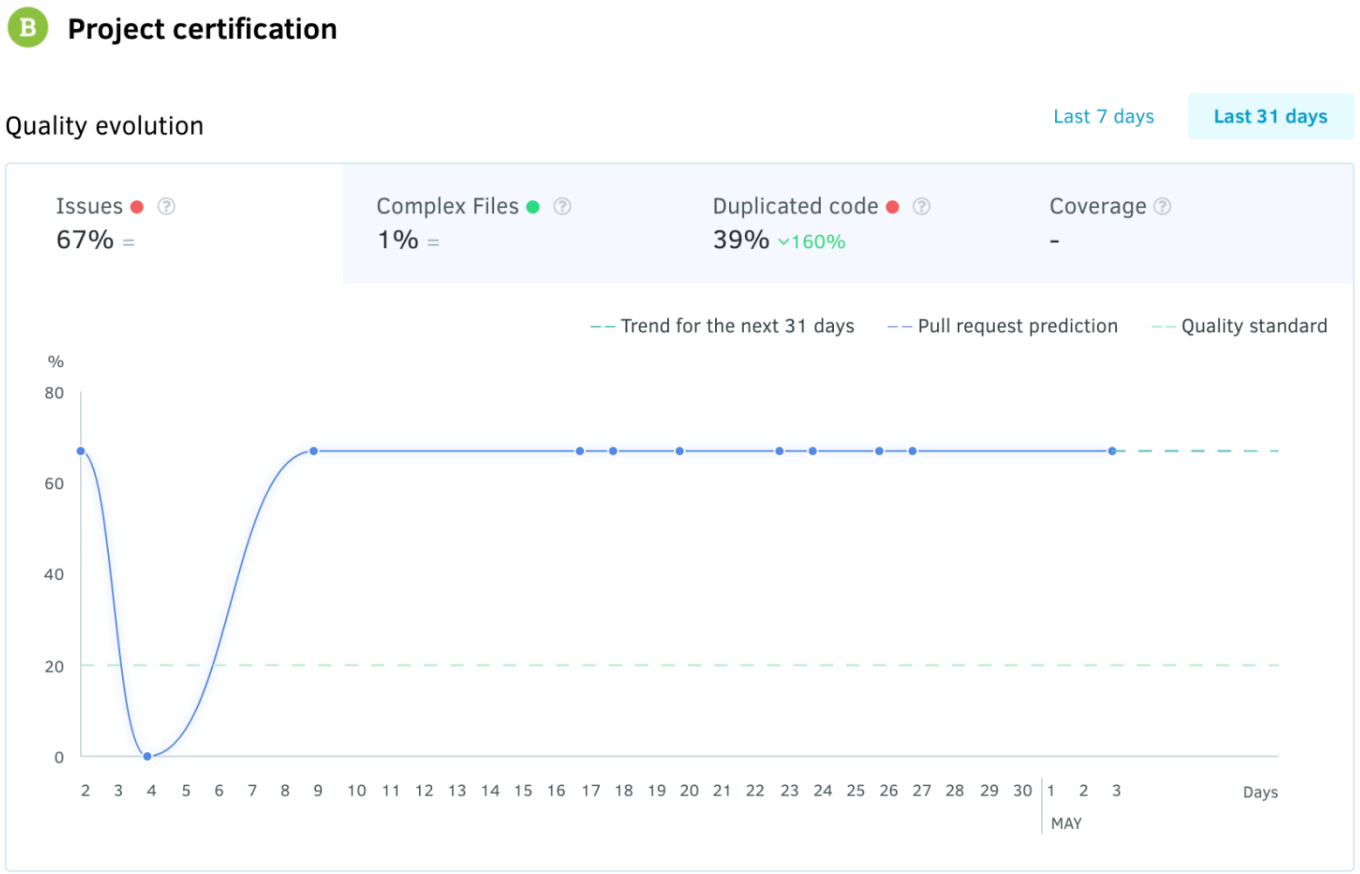
- After the analysis Codacy sends an email



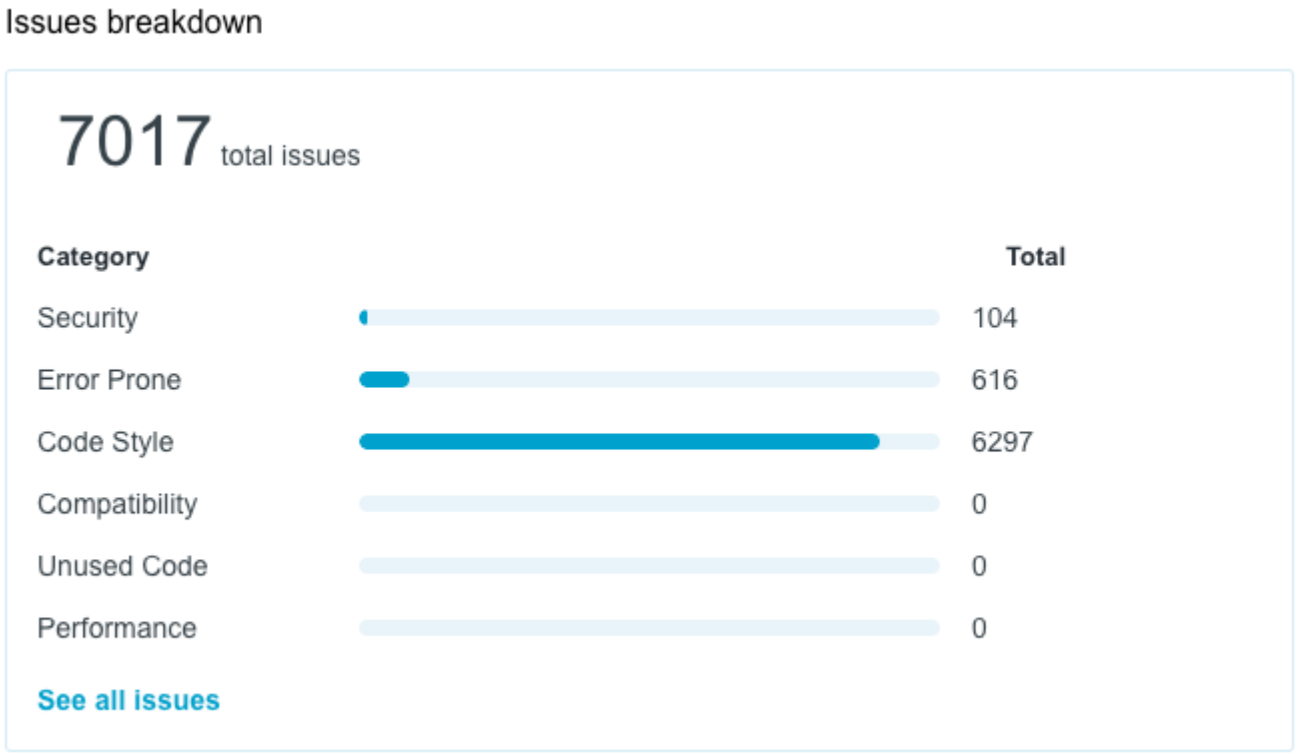
Codacy

- 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**

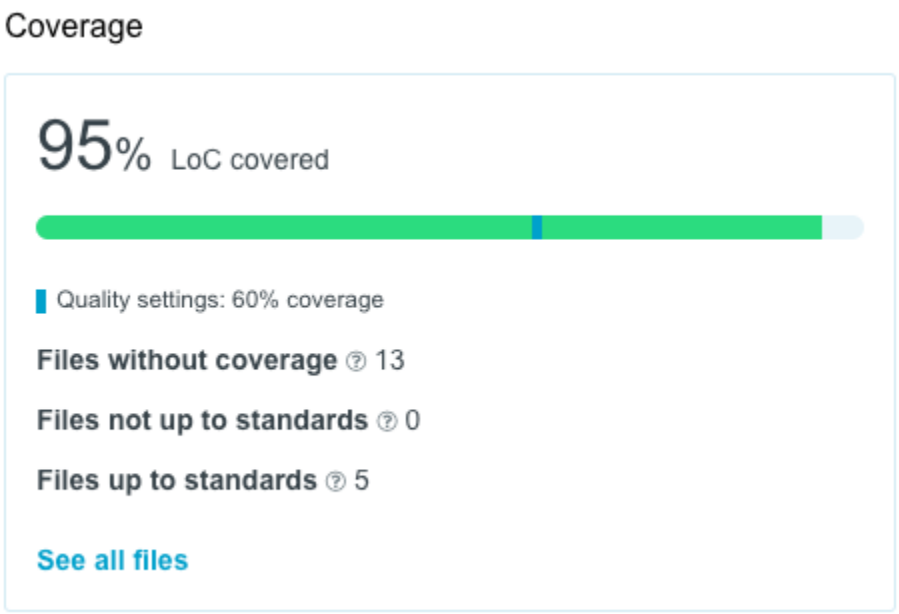
Codacy: Project certification and Quality evolution



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

E squbs-unicomplex/src/main/scala/org/squbs/unicomplex/streaming/ServiceRegistry.scala

Ignore File

TIME TO FIX: 1 hour View on GitHub

Size		Structure		Complexity		Duplication	
Lines of code:	273	Number of Classes:	8	Complexity:	26	Number of Clones:	13
Source lines of code:	194	sLoC / Class:	24.25	Complexity / Class:	3.25	Duplicated lines of code:	134
Commented lines of code:	26	Number of Methods:	31	Complexity / Method:	0.84		
		sLoC / Method:	6.26	Churn:	19		

End