Universidad de La Laguna

Programación de Aplicaciones Interactivas

Curso 2025-2026

Guillermo López Concepción
Raúl González Acosta
Paula Díaz Jorge

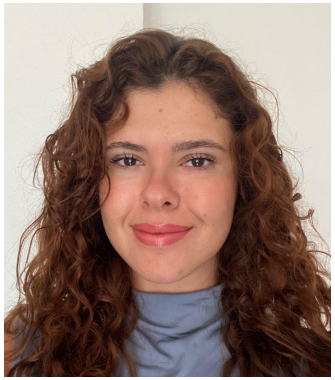
Testing in Javascript/Typescript and debug

Link to code examples in these slides

Components

Raúl González Acosta
(raul.gonzalez.acosta.33@ull.edu.es)

Guillermo López Concepción
(guillermo.lopez.37@ull.edu.es)



Paula Díaz Jorge
(paula.diaz.31@ull.edu.es)

Topics

- Introduction to testing. Unit Test (UT).
- BDD (Behaviour Driven Development).
- Jest.
- Testing in Javascript / Typescript.
- Testing in TypeScript with OOP.
- Examples from Exercism / Jutge.
- Code debugging.

What is the meaning of testing?

- Act of **checking** whether it meets the **requirements**

What is the meaning of testing?

- Act of **checking** whether it meets the **requirements**
- Divided into **Verification** and **Validation**

What is the meaning of testing?

- Act of **checking** whether it meets the **requirements**
- Divided into **Verification** and **Validation**
- **Detects mistakes** as soon as possible

What is the meaning of testing?

- Act of **checking** whether it meets the **requirements**
- Divided into **Verification** and **Validation**
- **Detects mistakes** as soon as possible
- **Documents** the expected behaviour

What is the meaning of testing?

- Act of **checking** whether it meets the **requirements**
- Divided into **Verification** and **Validation**
- **Detects mistakes** as soon as possible
- **Documents** the expected behaviour
- Nowadays, is the **key tool for CI/CD**

What is Unit Testing?

- Testing the **smallest "unit" of code** (functions/method/classes).

What is Unit Testing?

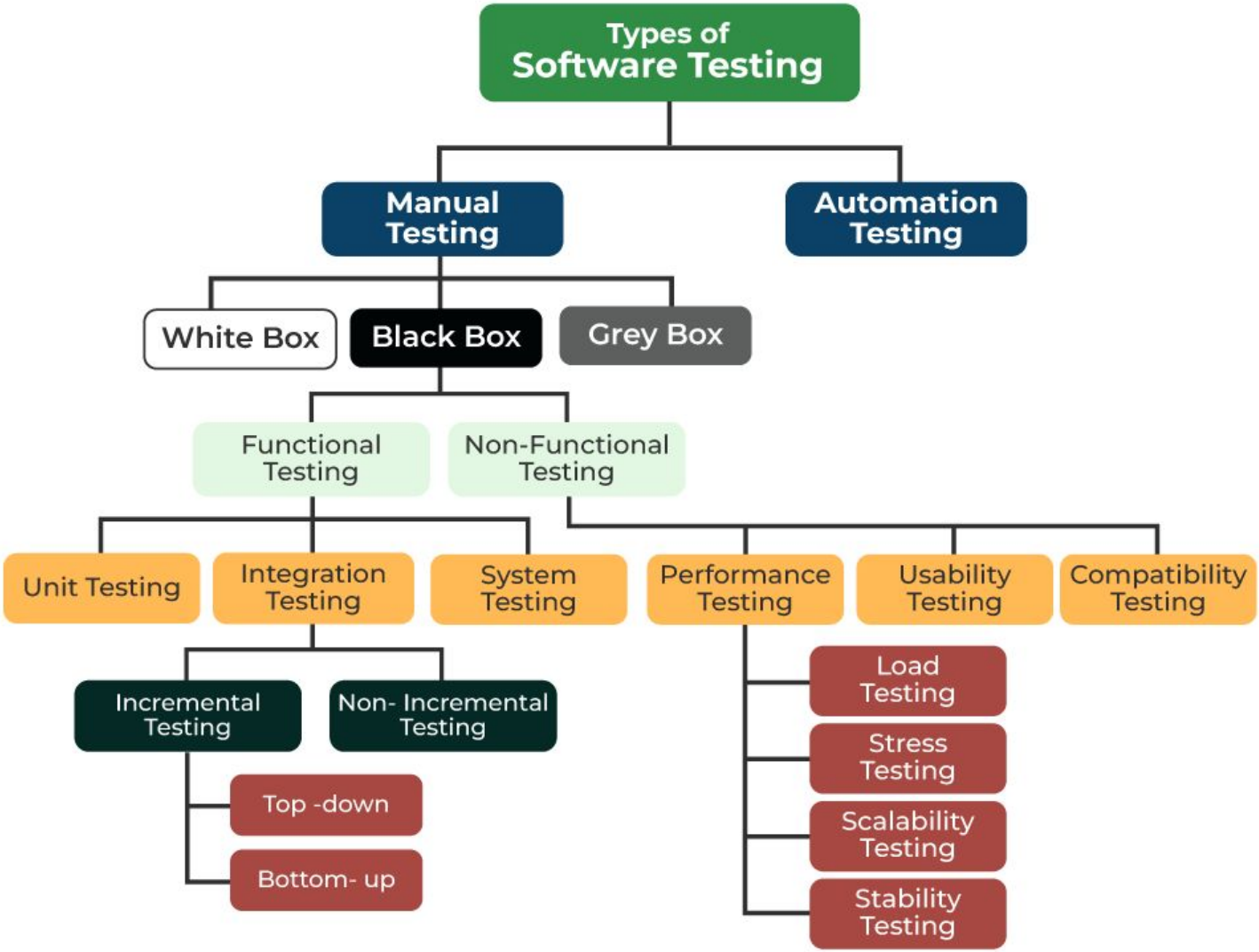
- Testing the **smallest "unit" of code** (functions/method/classes).

- **Isolation** is key.

What is Unit Testing?

- Testing the **smallest "unit" of code** (functions/method/classes).
- **Isolation** is key.
- Fast, automated, and repeatable.

Types of Testing



BDD (Behavior Driven Development)

- Focus on **behaviour** rather than implementation.

BDD (Behavior Driven Development)

- Focus on **behaviour** rather than implementation.
- **Human-readable** syntax.

BDD (Behavior Driven Development)

- Focus on **behaviour** rather than implementation.
- **Human-readable** syntax.
- Pattern: **Given / When / Then.**

BDD (Behavior Driven Development)

```
Feature: Login
  As a user
  I want to be able to log in to the application
  So that I can access my account information

  Scenario: Successful login
    Given I am on the login page
    When I enter my username and password
    And I click the login button
    Then I should be taken to the dashboard page
```

Example code

UT Testing Platforms

UT Testing Platforms

UT Testing Platforms

UT Testing Platforms

What is Jest?

What is Jest?

- A delightful JavaScript Testing Framework with a focus on simplicity.

What is Jest?

- A delightful JavaScript Testing Framework with a focus on simplicity.
- Developed and maintained by Meta (Facebook).

What is Jest?

- A delightful JavaScript Testing Framework with a focus on simplicity.
- Developed and maintained by Meta (Facebook).
- **All-in-one tool:** Includes a test runner, assertion library, and mocking support.

What is Jest?

- A delightful JavaScript Testing Framework with a focus on simplicity.
- Developed and maintained by Meta (Facebook).
- **All-in-one tool:** Includes a test runner, assertion library, and mocking support.
- Works out of the box for most JavaScript projects.

How Jest Works

- **Discovery:** Automatically finds files ending in .test.js or .spec.js.

How Jest Works

- **Discovery:** Automatically finds files ending in .test.js or .spec.js.
- **Isolation:** Runs each test file in its own sandbox for consistent results.

How Jest Works

- **Discovery:** Automatically finds files ending in .test.js or .spec.js.
- **Isolation:** Runs each test file in its own sandbox for consistent results.
- **Snapshot Testing:** Captures large data structures to track changes over time.

How Jest Works

```
describe('Mathematics Module', () => {

  test('should verify that the environment is ready', () => {
    const systemReady = true;

    expect(systemReady).toBe(true);
  });
});
```

How Jest Works

```
describe('Mathematics Module', () => {

  test('should verify that the environment is ready', () => {
    const systemReady = true;

    expect(systemReady).toBe(true);
  });
});
```

- 'describe' creates a block that groups together several related tests.

How Jest Works

```
describe('Mathematics Module', () => {

  test('should verify that the environment is ready', () => {
    const systemReady = true;

    expect(systemReady).toBe(true);
  });
});
```

- 'describe' creates a block that groups together several related tests.
- 'test' (or 'it') is the actual unit test.

How Jest Works

```
describe('Mathematics Module', () => {

  test('should verify that the environment is ready', () => {
    const systemReady = true;

    expect(systemReady).toBe(true);
  });
});
```

- 'describe' creates a block that groups together several related tests.
- 'test' (or 'it') is the actual unit test.
- 'expect' and 'toBe' are the assertion part.

Testing Matchers: Equality

Testing Matchers: Equality

- **toBe(value)**: Exact identity (**===**).

Testing Matchers: Equality

- **toBe(value)**: Exact identity (**===**).
- **toEqual(object)**: Deep equality (content check).

Testing Matchers: Equality

- **toBe(value)**: Exact identity (**===**).
- **toEqual(object)**: Deep equality (content check).

```
function createUser(name) {
  return { name, active: true };
}
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
    expect(result).toEqual({name: 'Guillermo', active: true});
  });
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
    expect(result).toEqual({name: 'Guillermo', active: true});
  });

  it('should distinguish between different instances (toBe)', () => {
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
    expect(result).toEqual({name: 'Guillermo', active: true});
  });

  it('should distinguish between different instances (toBe)', () => {
    const result = createUser('Guillermo');
    const expected = {name: 'Guillermo', active: true};
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
    expect(result).toEqual({name: 'Guillermo', active: true});
  });

  it('should distinguish between different instances (toBe)', () => {
    const result = createUser('Guillermo');
    const expected = {name: 'Guillermo', active: true};
    expect(result).not.toBe(expected);
  });
});
```

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
    expect(result).toEqual({name: 'Guillermo', active: true});
  });

  it('should distinguish between different instances (toBe)', () => {
    const result = createUser('Guillermo');
    const expected = {name: 'Guillermo', active: true};
    expect(result).not.toBe(expected);
  });
});
```

Do we need to redefine everything?

Testing Matchers: Equality

```
describe('User Factory Matchers', () => {
  it('should create a user with the correct content (toEqual)', () => {
    const result = createUser('Guillermo');
    expect(result).toEqual({name: 'Guillermo', active: true});
  });

  it('should distinguish between different instances (toBe)', () => {
    const result = createUser('Guillermo');
    const expected = {name: 'Guillermo', active: true};
    expect(result).not.toBe(expected);
  });
});
```

Do we need to redefine everything? **NO**

beforeEach sentence

beforeEach sentence

```
describe('User Management Optimization', () => {
  let sharedUser: User;
```

beforeEach sentence

```
describe('User Management Optimization', () => {
  let sharedUser: User;

  beforeEach(() => {
    sharedUser = createUser('Guillermo');
  });
```

beforeEach sentence

```
describe('User Management Optimization', () => {
  let sharedUser: User;

  beforeEach(() => {
    sharedUser = createUser('Guillermo');
  });

  test('should have the correct initial name', () => {
    // No need to call createUser() here
    expect(sharedUser.name).toBe('Guillermo');
  });
```

beforeEach sentence

```
describe('User Management Optimization', () => {
  let sharedUser: User;

  beforeEach(() => {
    sharedUser = createUser('Guillermo');
  });

  test('should have the correct initial name', () => {
    // No need to call createUser() here
    expect(sharedUser.name).toBe('Guillermo');
  });

  test('should initialize as an active user', () => {
    expect(sharedUser.active).toBe(true);
  });
});
```

Testing Matchers: Arrays & Errors

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.
- **toThrow(error):** Verifies that a function crashes correctly.

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.
- **toThrow(error):** Verifies that a function crashes correctly.

```
export function validateRole(role: string): string[] {
  if (!role) throw new Error('Empty role');
  return ['admin', 'guest', role];
}
```

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.

```
test('should include the new role in the list', () => {
  const roles = validateRole('editor');
  expect(roles).toContain('editor');
});
```

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.

```
test('should include the new role in the list', () => {
  const roles = validateRole('editor');
  expect(roles).toContain('editor');
});
```

- **toThrow(error):** Verifies that a function crashes correctly.

Testing Matchers: Arrays & Errors

- **toContain(item):** Checks if an array has an element.

```
  test('should include the new role in the list', () => {
    const roles = validateRole('editor');
    expect(roles).toContain('editor');
  });
```

- **toThrow(error):** Verifies that a function crashes correctly.

```
  test('should throw error on empty input', () => {
    expect(() => validateRole('')).toThrow('Empty role');
  });
});
```

Advanced Matchers: Truthiness & Numbers

Advanced Matchers: Truthiness & Numbers

- **Truthiness:** toBeNull, toBeUndefined, toBeDefined, toBeTruthy, toBeFalsy.

Advanced Matchers: Truthiness & Numbers

- **Truthiness:** toBeNull, toBeUndefined, toBeDefined, toBeTruthy, toBeFalsy.

- **Numbers:** toBeGreaterThan, toBeLessThan, toBeCloseTo...

Advanced Matchers: Truthiness & Numbers

- **Truthiness:** toBeNull, toBeUndefined, toBeDefined, toBeTruthy, toBeFalsy.

- **Numbers:** toBeGreaterThan, toBeLessThan, toBeCloseTo...

Check the example

List of Jest Matchers (non exhaustive list)

- **Truthiness**
  toBeNull()
  toBeUndefined()
  toBeDefined()
  toBeTruthy()
  toBeFalsy()

- **Numbers**
  toBeGreaterThan(number)
  toBeGreaterThanOrEqual(number)
  toBeLessThan(number)
  toBeLessThanOrEqual(number)
  toBeCloseTo(number, numberOfDecimals)

- **Equality Matchers**
  toBe(value)
  toEqual(value)
  toStrictEqual(value)

List of Jest Matchers (non exhaustive list)

- **Strings**
  toMatch(regularExpressionOrString)
  toContain(subString)
- **Arrays**
  toContain(item)
  toHaveLength(number)
- **Exceptions**
  toThrow(error?)
- **Negation**
  not.toBe(value)
  not.toContain(item)

Every matcher can be negated

Execution time testing

Execution time testing

Jest Timeout

```
/**
* Tests an asynchronous function with a time limit of 5000ms.
* If the promise does not resolve in that period, the test fails.
*/
test('the function must be completed in less than 5 seconds', async () => {
  await myHeavyFunction();
}, 5000); // Time limit in milliseconds
```

Execution time testing

Jest Timeout

```
/**
* Tests an asynchronous function with a time limit of 5000ms.
* If the promise does not resolve in that period, the test fails.
*/
test('the function must be completed in less than 5 seconds', async () => {
  await myHeavyFunction();
}, 5000); // Time limit in milliseconds
```

- Does not work to test time limits on synchronous functions.

Execution time testing

Jest Timeout

```
/**
* Tests an asynchronous function with a time limit of 5000ms.
* If the promise does not resolve in that period, the test fails.
*/
test('the function must be completed in less than 5 seconds', async () => {
  await myHeavyFunction();
}, 5000); // Time limit in milliseconds
```

- Does not work to test time limits on synchronous functions.
- With asynchronous tests, Jest's timeout fails the test if the **async** operation (**Promise**) doesn't finish within the given time limit.

Execution time testing

Date.now() / performance.now()

```
describe('slowFunction function tests', () => {
  test('slowFunction must not exceed 5ms of execution time', () => {
    const start = Date.now();
    slowFunction(5);
    const end = Date.now()

    expect(end - start).toBeLessThanOrEqual(5);
  });
```

Code example

Execution time testing

Date.now() / performance.now()

```
describe('slowFunction function tests', () => {
  test('slowFunction must not exceed 5ms of execution time', () => {
    const start = Date.now();
    slowFunction(5);
    const end = Date.now()

    expect(end - start).toBeLessThanOrEqual(5);
  });
```

- Use **Date.now()** for simple, coarse timing.

Execution time testing

Date.now() / performance.now()

```
describe('slowFunction function tests', () => {
  test('slowFunction must not exceed 5ms of execution time', () => {
    const start = Date.now();
    slowFunction(5);
    const end = Date.now()

    expect(end - start).toBeLessThanOrEqual(5);
  });
```

- Use **Date.now()** for simple, coarse timing.
- Use **performance.now()** for more precise performance measurements.

Testing with Jest in OOP

```
test("subclass instances are also instances of the base class", () => {
   const circle = new Circle("red", 2);
   expect(circle).toBeInstanceOf(Circle);
   expect(circle).toBeInstanceOf(Shape);
 });
```

Code example

Testing with Jest in OOP

```
describe("ShoppingCart tests", () => {
 let cart: ShoppingCart;

 beforeEach(() => {
   cart = new ShoppingCart();
 });

...
```

Code example

How to Run Your Tests

How to Run Your Tests

- **Installation:** *npm install --save-dev jest ts-jest*

How to Run Your Tests

- **Installation:** *npm install --save-dev jest ts-jest*
- **Init:** *npx ts-jest config:init*

How to Run Your Tests

- **Installation:** *npm install --save-dev jest ts-jest*
- **Init:** *npx ts-jest config:init*
- **Run:** *npm test* or *npx jest*

Jest Configuration File

- Must be named “**jest.config.ts**”

Jest Configuration File

- Must be named “**jest.config.ts**”
- Describes **Jest behaviour**

Jest Configuration File

- Must be named “**jest.config.ts**”
- Describes **Jest behaviour**
- Describes which **test environment** to use

Jest Configuration File

```ts
import type { Config } from 'jest';

const config: Config = {
  preset: 'ts-jest',
  testEnvironment: 'node',
};

export default config;
```

Convert Jutge test into Jest test

- **Automated validation** before submission

Convert Jutge test into Jest test

- **Automated validation** before submission
- Easy coverage of edge cases

Convert Jutge test into Jest test

- **Automated validation** before submission
- Easy coverage of edge cases
- **Exact output** verification

Convert Jutge test into Jest test

- **Automated validation** before submission
- Easy coverage of edge cases
- **Exact output** verification
- Fast feedback loop

Convert Jutge test into Jest test

Numbers in an interval

P97156

Statement

Write a program that reads two numbers a and b, and prints all numbers between a and b.

Input

Input consists of two natural numbers a and b.

Output

Print a line with $a, a+1, ..., b-1, b$. Separate the numbers with commas.

Public test cases

Input	Output
15 21	15,16,17,18,19,20,21
20 10	
7 7	7

[Example code]

Convert Jutge test into Jest test

```
describe('numbersInInterval', () => {
  test('first jutge public test case', () => {
    expect(numbersInInterval(15, 21)).toEqual([15, 16, 17, 18, 19, 20, 21]);
  });

  test('second jutge public test case', () => {
    expect(numbersInInterval(20, 10)).toEqual([]);
  });

  test('third jutge public test case', () => {
    expect(numbersInInterval(7, 7)).toEqual([7]);
  })
});
```

Exercism tests

- Source code
- Testing in Exercism
- Remember that Exercism **by default skips tests**

Summary & Best Practices

Summary & Best Practices

- ***F.I.R.S.T.***

Summary & Best Practices

- ***F.I.R.S.T.***

- ***Fast:*** *Tests should run quickly so you can run them often.*

Summary & Best Practices

- ***F.I.R.S.T.***
- ***Fast:*** *Tests should run quickly so you can run them often.*
- ***Independent:*** *Tests should not depend on each other; no test should set up the state for the next.*

Summary & Best Practices

- ***F.I.R.S.T.***
- ***Fast:*** *Tests should run quickly so you can run them often.*
- ***Independent:*** *Tests should not depend on each other; no test should set up the state for the next.*
- ***Repeatable:*** *Tests should be able to run in any environment (your laptop, a server, etc.).*

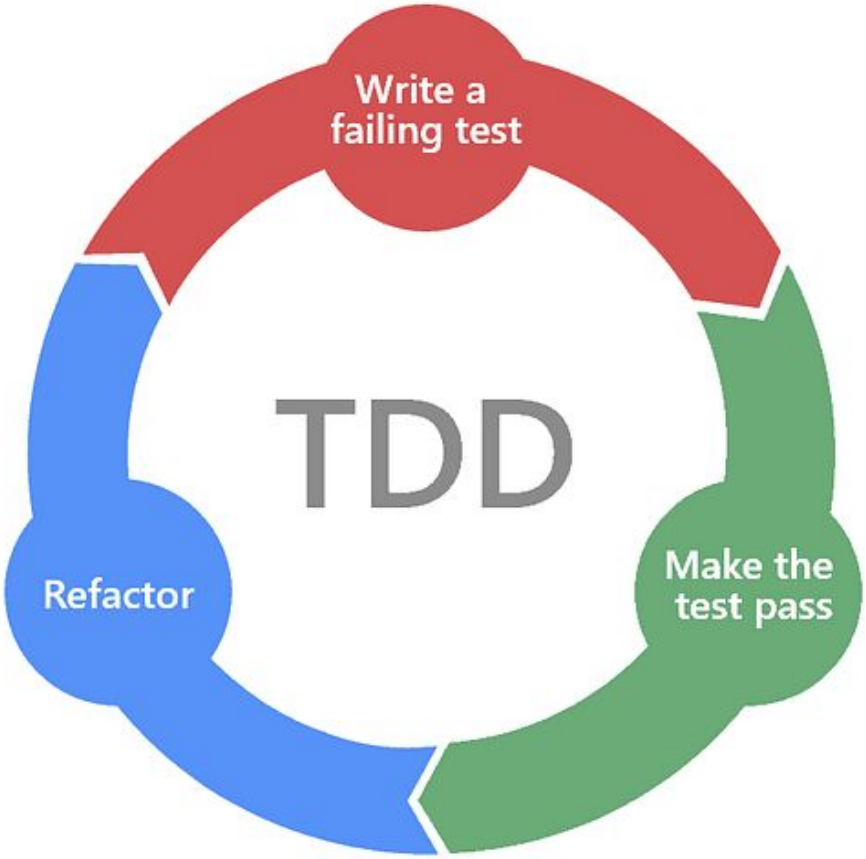
Summary & Best Practices

- ***F.I.R.S.T.***
- ***Fast:*** *Tests should run quickly so you can run them often.*
- ***Independent:*** *Tests should not depend on each other; no test should set up the state for the next.*
- ***Repeatable:*** *Tests should be able to run in any environment (your laptop, a server, etc.).*
- ***Self-Validating:*** *Tests should have a clear boolean output (pass or fail).*

Summary & Best Practices

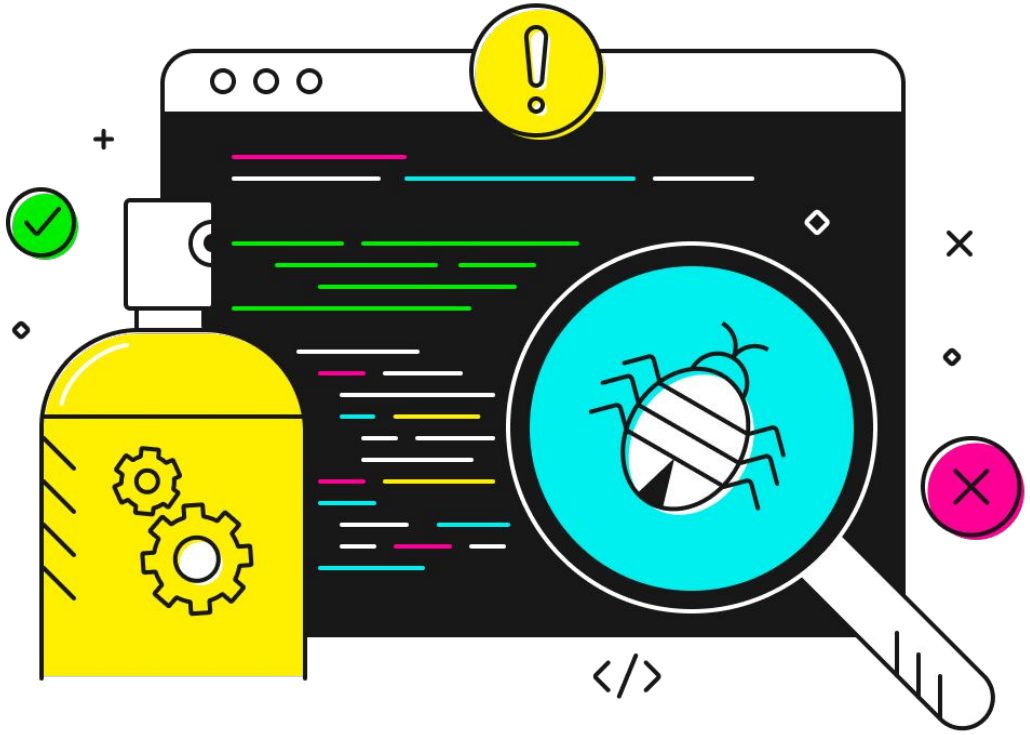
- ***F.I.R.S.T.***
- ***Fast:*** *Tests should run quickly so you can run them often.*
- ***Independent:*** *Tests should not depend on each other; no test should set up the state for the next.*
- ***Repeatable:*** *Tests should be able to run in any environment (your laptop, a server, etc.).*
- ***Self-Validating:*** *Tests should have a clear boolean output (pass or fail).*
- ***Timely:*** *Tests should be written just before the production code that makes them pass (TDD).*

Summary & Best Practices



Lectura recomendada: Clean code

Debugging



What is debugging?

- **Goal:** Find the root cause of a bug by pausing the program and inspecting its state.

What is debugging?

- **Goal:** Find the root cause of a bug by pausing the program and inspecting its state.

- Debugging helps you:

What is debugging?

- **Goal:** Find the root cause of a bug by pausing the program and inspecting its state.

- Debugging helps you:
  - See variable values at a specific moment.

What is debugging?

- **Goal:** Find the root cause of a bug by pausing the program and inspecting its state.

- Debugging helps you:
  - See variable values at a specific moment.
  - Follow the execution path step by step.

What is debugging?

- **Goal:** Find the root cause of a bug by pausing the program and inspecting its state.

- Debugging helps you:
  - See variable values at a specific moment.
  - Follow the execution path step by step.
  - Understand why something fails (not just that it fails)

Debugging VS Logging

Debugging

- **Pauses execution**.

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.
- Step through code and evaluate expressions.

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.
- Step through code and evaluate expressions.

Logging (console.log)

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.
- Step through code and evaluate expressions.

Logging (console.log)

- Quick visibility.

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.
- Step through code and evaluate expressions.

Logging (console.log)

- Quick visibility.
- Does not stop the program.

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.
- Step through code and evaluate expressions.

Logging (console.log)

- Quick visibility.
- Does not stop the program.
- Can get noisy.

Debugging VS Logging

Debugging

- **Pauses execution**.
- **Inspects state** and variables in real-time.
- Step through code and evaluate expressions.

Logging (console.log)

- Quick visibility.
- Does not stop the program.
- Can get noisy.

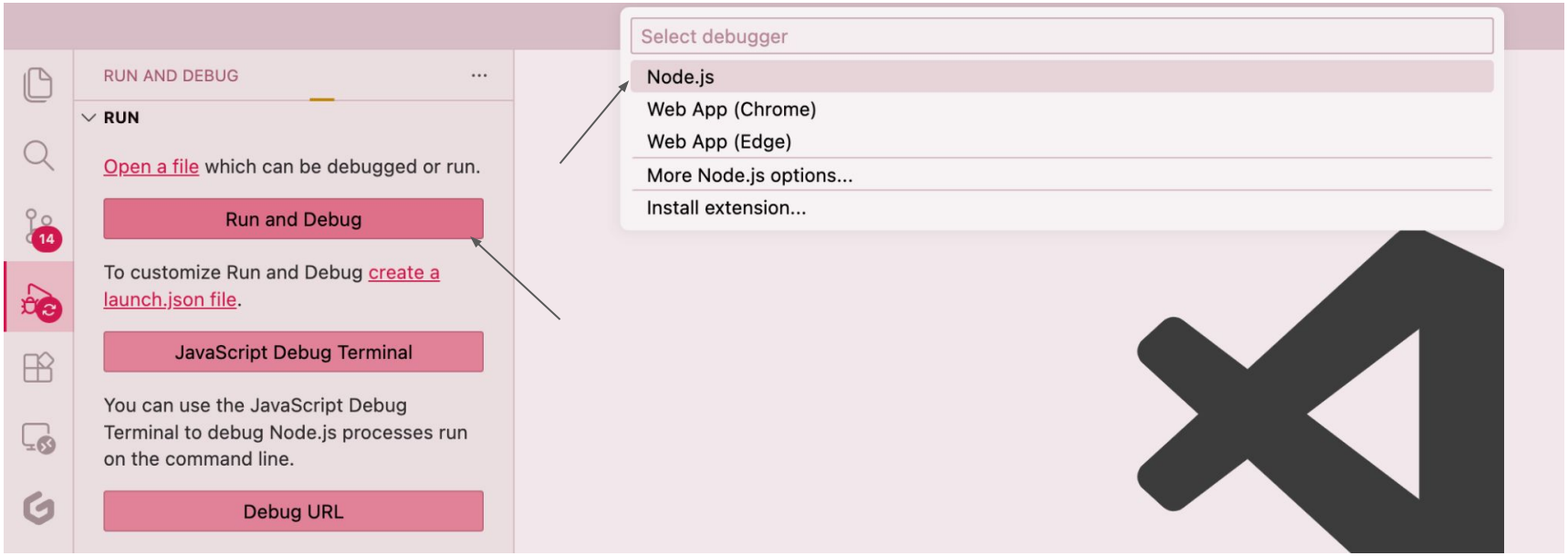
Logging is **passive observation**, while debugging is **active investigation**.

Debugging in Visual Studio Code

- VS Code comes with an integrated debugger for Node.js

Debugging in Visual Studio Code

- VS Code comes with an integrated debugger for Node.js



Debugging in Visual Studio Code

Debugging tools

- **Breakpoints**: pause at a line.

Debugging in Visual Studio Code

Debugging tools

- **Breakpoints**: pause at a line.
- **Variables** panel: current values.

Debugging in Visual Studio Code

Debugging tools

- **Breakpoints**: pause at a line.
- **Variables** panel: current values.
- **Watch** panel: custom expressions.

Debugging in Visual Studio Code

Debugging tools

- **Breakpoints**: pause at a line.
- **Variables** panel: current values.
- **Watch** panel: custom expressions.
- **Call Stack**

Debugging in Visual Studio Code

Debugging tools

- **Breakpoints**: pause at a line.
- **Variables** panel: current values.
- **Watch** panel: custom expressions.
- **Call Stack**
- **Debug Console**: evaluate code while paused.

Debugging in Visual Studio Code

The basics

- **Add a breakpoint:** click left on the line number.

Debugging in Visual Studio Code

The basics

- **Add a breakpoint:** click left on the line number.
- **Controls:**
  - Step Over.

Debugging in Visual Studio Code

The basics

- **Add a breakpoint:** click left on the line number.
- **Controls:**
  - Step Over.
  - Continue.

Debugging in Visual Studio Code

The basics

- **Add a breakpoint:** click left on the line number.
- **Controls:**
  - Step Over.
  - Continue.
  - Step Into.

Debugging in Visual Studio Code

The basics

- **Add a breakpoint:** click left on the line number.
- **Controls:**
  - Step Over.
  - Continue.
  - Step Into.
  - Step out.

Debugging in Visual Studio Code

Useful breakpoint types

- **Conditional breakpoint:** pause only if a condition is true.

Debugging in Visual Studio Code

Useful breakpoint types

- **Conditional breakpoint:** pause only if a condition is true.
- **Logpoint:** prints a message without changing code.

Debugging in Visual Studio Code

Useful breakpoint types

- **Conditional breakpoint:** pause only if a condition is true.
- **Logpoint:** prints a message without changing code.
- **Break on exceptions**: pause when an error is thrown.

References

- Clean Code: A Handbook of Agile Software Craftsmanship
- Clean Code TypeScript Repository
- Official Jest Documentation
- Google JavaScript Style Guide
- Google TypeScript Style Guide
- Professional Best Practices (F.I.R.S.T.)

Universidad de La Laguna

Doubts?
Questions?