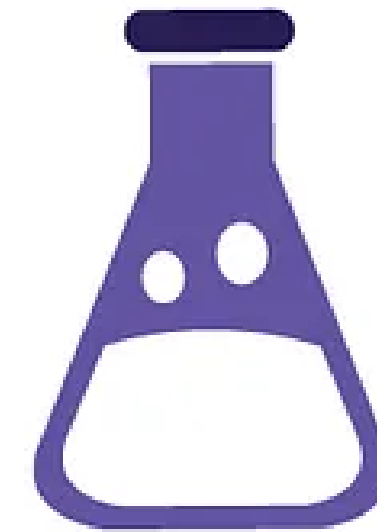




UNIT TESTING



unittest

by
Amanuel Shiferaw

April 2024

TABLE OF CONTENT

UNIT TESTING PRESENTATION

01 Types OF Testing

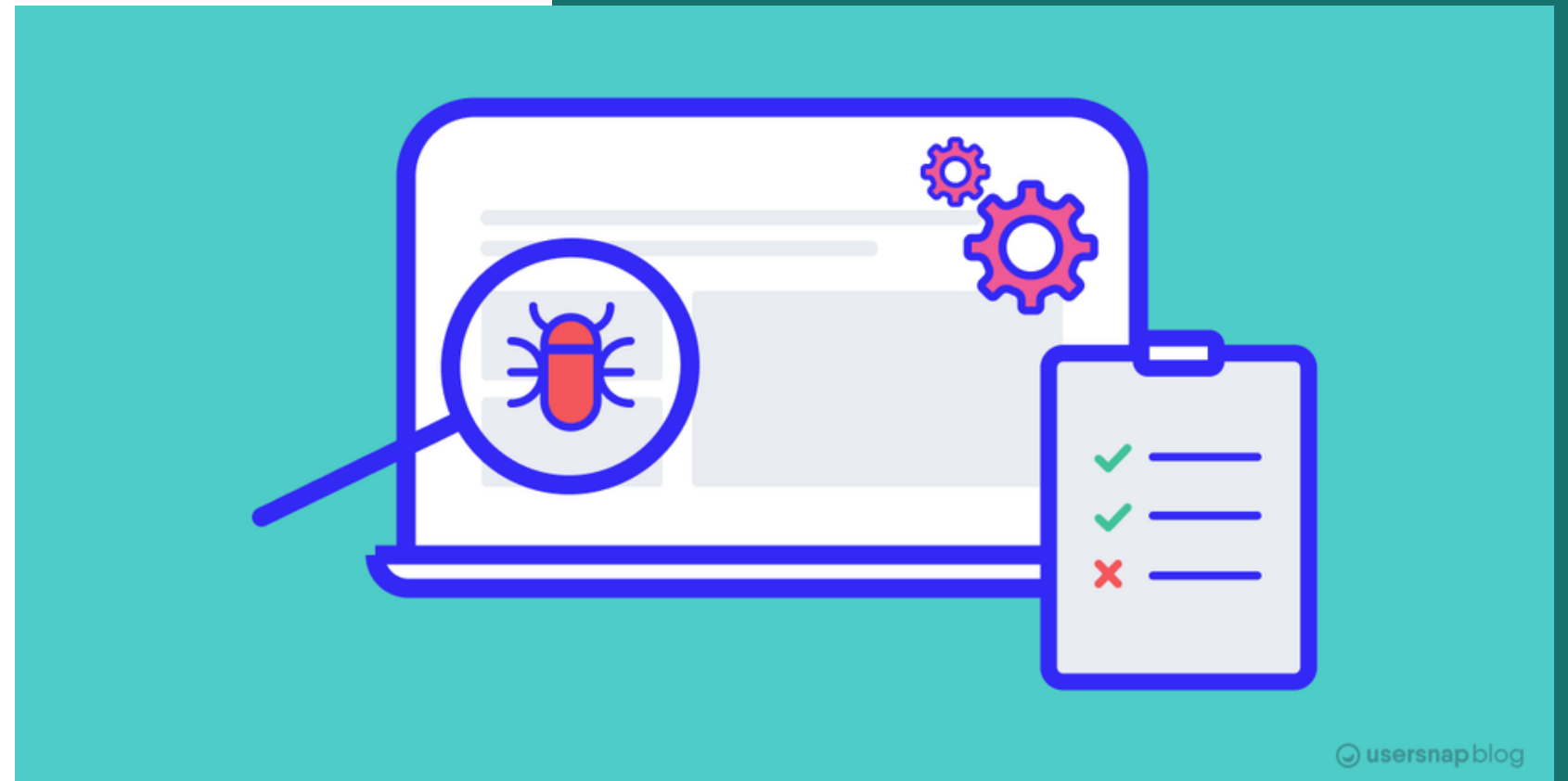
02 Unit Test

03 Principles of Unit Testing


04 Testing Frameworks


05 REST API Testing

06 A Few Points about TDD





TYPES OF TESTING

 **Unit Testing:** Testing individual code units in isolation to verify their functionality and behavior.

 **Integration Testing:** Testing interactions and integration between modules/components to ensure correct functionality.

 **Functional Testing:** Testing the functionality of a software system to ensure it meets specified requirements.

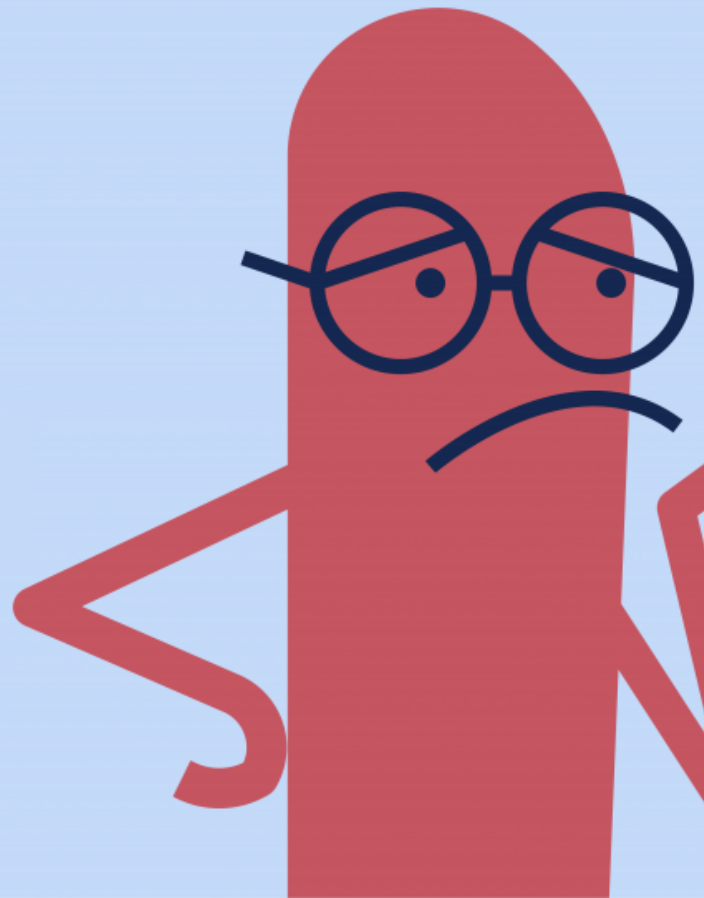
 **Regression Testing:** Testing previously validated functionality to catch any new defects introduced by changes or enhancements.

 **Acceptance Testing:** Testing to determine if a system meets acceptance criteria and stakeholder needs.

Unit Testing

(TODAY'S TOPIC 😊)

SO, WHAT IS UNIT TESTING



UNIT TEST

It is a self-contained, piece of code that evaluates and verifies the correctness and expected behavior of another piece of code, typically at the level of individual functions or methods.

WHY UNIT TESTING?

Increases confidence in refactoring

Detect and prevent bugs early in the development process

Ensure individual units of code function correctly in isolation

Support continuous integration and delivery practices

Focused and Comprehensive



UNIT TESTING PRINCIPLES

- ▶ **Isolatable**
- ▶ **Repeatable**
- ▶ **Independent**
- ▶ **Automatable**
- ▶ **Focused and Comprehensive**



Testing Framework



TESTING FRAMEWORK CONSISTS

Test
Runner


Assertion
Libraries

Mocking
Tools


Coverage
Tools

Introduction To Mocha, Chai & Sinon

MOCHA



A test runner for JavaScript in Node.js and browsers, perfect for organizing and executing tests



Define test cases effortlessly with hooks for setup and teardown, ensuring well-structured and organized test environments.




provides a straightforward and expressive syntax for writing tests


```
describe("My Test suite", () => {  
  before(() => {  
    | // Code to run once before all tests in the describe block  
  });  
  beforeEach(() => {  
    | // Code to run before each test case in the describe block  
  });  
  afterEach(() => {  
    | // Code to run after each test case in the describe block  
  });  
  after(() => {  
    | // Code to run once after all tests in the describe block  
  });  
  it("should do something", () => {  
    | // Test case code  
  });  
});
```

You, 1 second ago • Uncommitted changes


CHAI



Expressive and flexible JavaScript assertion library for test-driven development.



Rich set of assertion styles (should, expect, assert) for clear and concise assertions, tailored to developers' preferred syntax.




Extensible plugin system for additional matchers, framework integrations, and customized assertion behaviors.

Chai's different assertion styles

```
it("should do something", () => {  
  
  const actual = add(5,2);  
  const expected = 7;  
  
  expect(actual).to.be.equal(expected);  
  
  assert.isTrue(actual);  
  
  actual.should.equal(expected);  
  
});
```

SINON

(PRONOUNCED "SIGH-NON")



JavaScript mocking library used for creating test doubles such as spies, stubs, and mocks.



enable for creating test doubles, enabling easy spying on function calls, stubbing method behavior, and mocking objects.

You, 3 hours ago | 1 author (You)

```
class Calculator {  
  static add(a, b) {  
    return a + b;  
  }  
}
```

```
// Create a stub for the method add and override it to return 10  
const addStub = sinon.stub(Calculator, 'add');  
addStub.returns(10);
```

```
// Use the stub  
const result = Calculator.add(3, 5); // Output: 10
```

```
// Restore the stub  
addStub.restore();
```

```
// Create a mock for the static method  
const addMock = sinon.mock(Calculator);  
addMock.expects('add').once().withExactArgs(2, 3).returns(5);
```

```
// Use the mock  
const result2 = Calculator.add(2, 3);  
console.log(result2); // Output: 5
```

```
// Verify the mock  
addMock.verify();
```

You, 3 hours ago • Uncommitted changes

REST API TESTING

- ▶ Automated verification of functionality of a RESTful API using HTTP requests and response validation.
- ▶ Verify endpoints respond correctly to requests (GET, POST, PUT, DELETE).
- ▶ Check status codes, data format (JSON, XML), and content accuracy.
- ▶ Tests that the API responds appropriately to invalid requests.

TEST COVERAGE REPORTING WITH "NYC"

- ▶ provide insights into which parts of your code are covered by tests and their respective coverage percentages
- ▶ can specify the output format, such as text, HTML, or XML, using the configuration options.
- ▶ can be integrated into your continuous integration and deployment (CI/CD) pipelines

9 passing (62ms)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	89.65	100	66.66	89.65	
common.js	85.71	100	60	85.71	3-7
index.js	93.33	100	100	93.33	19

Done in 2.32s.

FEW POINTS ABOUT TDD

- ▶ a development approach where tests are written before the code implementation.
- ▶ promotes better code quality by encouraging developers to write modular and testable code
- ▶ helps in creating clear and specific requirements by writing test cases that define the expected behavior of the code.

THANK YOU

