TOPIC :Unit Testing
Test Suite Developme

PRESENTED TO –

BARGA DEORI SIR

PRESENTED BY-

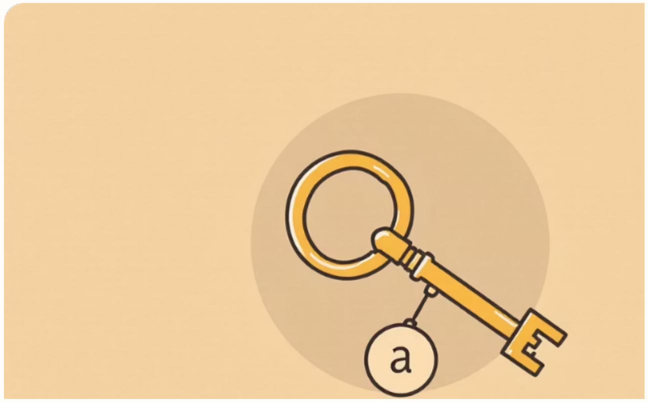
1. DEBOPRIYO PURKAYASTH
2. DHRUPAD KASHYAP - 24
3. DEBANKUR PAUL - 2481
4. SARANGA PANI DUTTA -
5. KANCHANDEEP GOHAIN

g the "Unit" in Unit Testing

nallest testable part of a program, designed to function independently. Its definition can vary base
anguage and architecture:

n

`eTotal(price)`

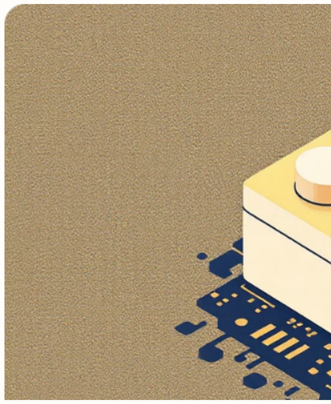


A Method

E.g., `user.login()`

A Class

Such as a `BankAccount` class



A Module

Like a Python m
package

nas a clear purpose, defined input, predictable output, and minimal dependencies, making it easier

nerstone of Software Quality: Unit Testing

Unit Testing?

g is a fundamental software testing technique
 verifying individual components in isolation.
each unit of code—be it a function, method,
odule—behaves precisely as expected.

The Purpose

- Catch bugs early in the development cyc
- Enhance software reliability and maintain
- Promote cleaner, modular code design.
- Validate individual component behavior.

testing, a well-structured test suite is crucial for comprehensive validation, acting as the backbor
ty assurance.

it Testing is Indispensable

ves as the first line of defense against software defects, offering numerous critical benefits throughout the

tection

at the earliest stages, significantly reducing fix costs and

Improved Code Quality

Encourages developers to write cleaner, more modular, and well-
code.

factoring

cation and optimization of code, ensuring existing functionality

Reduces Debugging Time

Pinpoints issues rapidly by localizing failures to the unit level, not
integrated systems.

/CD

run on every commit, ensuring code consistency and stability in

Living Documentation

Well-written tests clarify how code components are designed to
behave.

High-Quality Unit Tests

sts are not just about finding bugs; they are about building a robust and reliable codebase. They possess ke

ution

d run in milliseconds to support frequent execution without hindering developer workflow.

nistic Results

ame input, tests must consistently produce the same outcome, free from environmental or timing inconsistencies.

Environment

d not depend on external systems (databases, APIs, network). Mocks and stubs are vital for maintaining isolation.

& Maintainable

be clear, well-structured, and adequately commented to explain what is being tested and why.

le Across Environments

d execute reliably on any machine or operating system without complex setup requirements.

ensive Coverage

d validate both typical use cases and critical edge cases, including error conditions and boundary values.

t Testing Workflow

approach to unit testing ensures thorough validation and efficient bug resolution. Follow this ge

its

h functions, classes, or
e testing.

02

Design Test Cases

Create scenarios covering normal, boundary, invalid, and exception cases.

03

Implement Tests

Write tests using an appropri
framework (e.g., JUnit, pytest

sts

natically or manually to verify

05

Debug & Fix

Trace failures, identify root causes, and resolve defects.

06

Re-run Tests

Confirm fixes without introdu
regressions.

overage

eate new tests for new features to ensure high test coverage.

c Test Case Design

case design is paramount for comprehensive evaluation, ensuring all possible behaviors are thoroughly ass

est Cases

cted behavior with valid inputs.

Negative Test Cases

Ensure graceful failure with invalid or unexpected i

Value Tests

at the extremes of allowable ranges (min/max).

Edge Case Tests

Address unusual conditions (e.g., empty lists, null i

Handling Tests

exception raising and handling mechanisms.

State & Behavior Tests

Validate internal state changes and external syste

Robust Test Suites

gically grouped collection of test cases that validate specific features or the entire system. A well-structured suite is critic

Key Components

lown: Prepares and cleans the test environment.

s/Files: Focuses on specific components.

ubbing Layer: Isolates tests from external dependencies.

elpers: Prevents code duplication for common actions.

Example Structure

```
tests/├── unit/│   ├── test_auth.py│   ├── test_payments.py
test_products.py├── fixtures/│   └── user_data.json├── mock
mock_api.py└── utils/    └── helpers.py
```

n improves scalability, readability, and maintainability, making it easier to navigate and automate testing pr

ctices & Essential Tools

actices and leveraging robust frameworks are crucial for effective unit testing and test suite development.

s

ion : Group tests by functionality.

ng : Use clear names like test_user_login_success().

n : Automate test runs on commits and PRs.

ach test should run autonomously.

Isolate external dependencies.

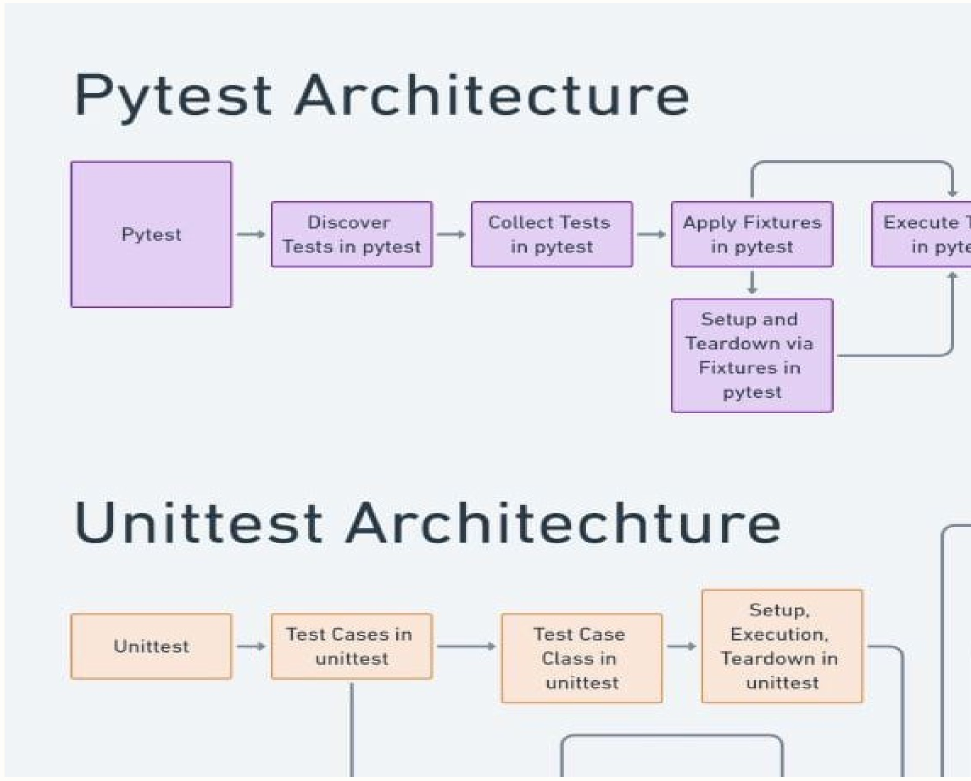
should be easier to understand than the code itself.

: Keep tests updated with code changes.

Popular Frameworks

- Java: JUnit, TestNG
- Python: pytest, unittest
- JavaScript: Jest, Mocha, Chai
- C/C++: GoogleTest (gtest)

These frameworks provide essential tools like test ru
mocking capabilities, and reporting features.



uite development are indispensable for building high-quality, reliable, and maintainable software. They catch errors early, e
and provide confidence during refactoring, protecting the system against regressions. An evolving test suite is a vital asset