CSCI 423&523
Advanced Software Engineering

Test-Driven Development

TDD

- **Definition**:
  - **Test-Driven Development (TDD)** is a **software development process** that relies on **repeating very short development cycles**.
  - In TDD, a developer first writes an **initial failing test**, then **writes minimal code** to pass that test, and finally **refactors** the new code.
- History:
  - Popularized by Kent Beck, one of the pioneers of Extreme Programming (XP).
  - Gained widespread adoption in agile environments, focusing on frequent releases and high-quality code.
- Key Principle:
  - Write test → Run test (fail) → Write code → Run test (pass) → Refactor.

TDD is not primarily about testing; it is a mechanism for evolving software design.

Test-Driven Development (TDD) relies on repeating very short development cycles. While it produces a test suite, its primary focus is on designing software effectively. Design evolves through the refactoring phase, not the initial coding phase.

Key Distinction

- **Traditional View:** Testers write tests to find bugs after coding.
- **TDD View:** Developers write unit tests to understand requirements BEFORE coding.

"It is rather about design – where design is evolved through refactoring."

Adopting TDD creates a safety net that accelerates development and improves maintainability.

The Value Proposition

Higher Code Quality

Forces developers to think through edge cases early, reducing regressions.

Immediate Feedback

Granular pass/fail results reveal issues instantly.

Improved Design

Writing tests first forces code to be modular and loosely coupled.

Reduced Fear

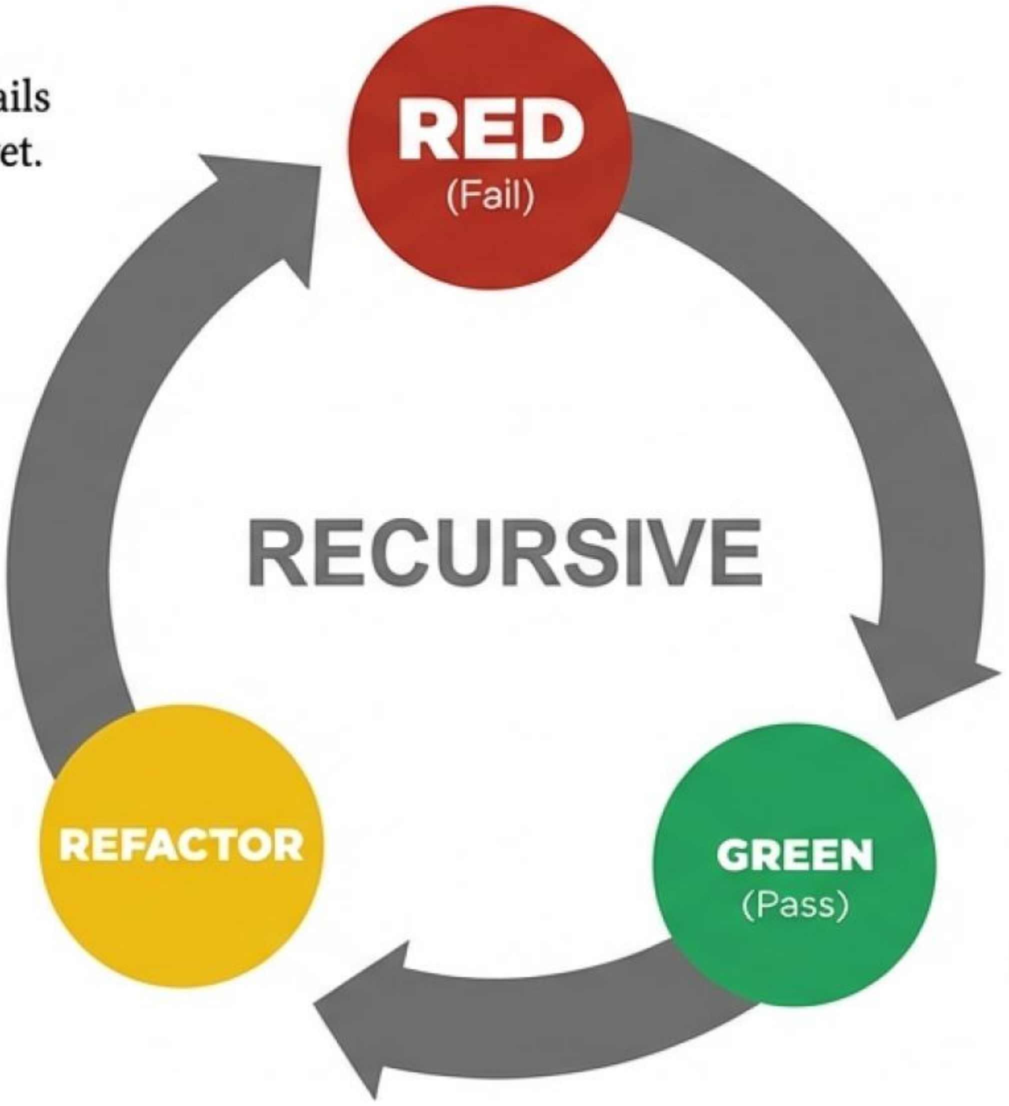
Comprehensive tests allow cleaning code without breaking functionality.

Faster Onboarding

The test suite acts as live documentation for new team members.

RED (Fail): Write a small test that fails because functionality doesn't exist yet. This serves as the specification.

Write a test that fails



REFACTOR: Improve code by eliminating duplication and simplifying logic while keeping tests passing.

Improve code quality

GREEN (Pass): Write minimal code to pass the test. Focus on correctness, not perfection.

Make only enough code for it to pass

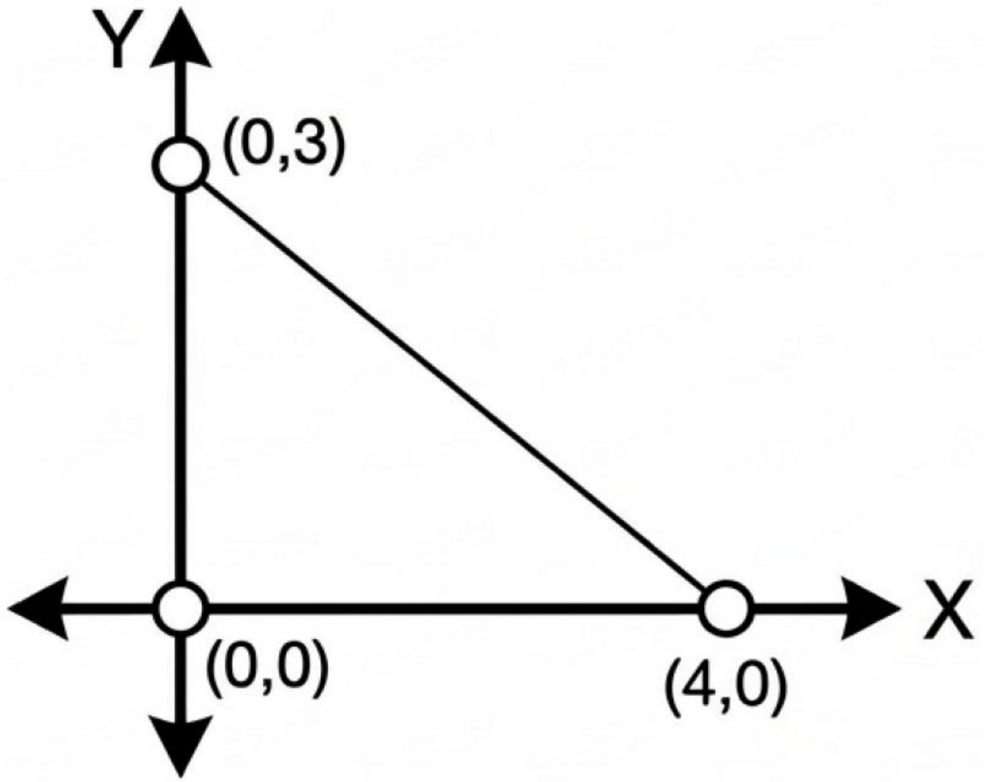
Practical Application: The 2D Triangle Problem.

Problem Definition

Scenario: Write a function that takes three Point objects in 2D space and evaluates if they form a valid triangle.

Input: Three Point objects (x, y).

Output: Boolean (True/False).



DEMO: https://colab.research.google.com/drive/1ii7yKkSJcUQ-ldqUzNrup7jjtZsM9c2V?usp=sharing

If we know that the code can be buggy, which motivates us to write tests to reveal code bugs; then what do we know if the tests are buggy?

We write failing tests first to recursively validate the tests themselves.

The Paradox:
If code can be buggy, tests (which are also code) can be buggy.

$\longrightarrow$

The Solution:
Recursive Validation.

$\longrightarrow$

The Red Phase:
By watching the test fail first, we prove the test is capable of detecting a missing requirement.

Key Insight: Red (let the tests fail) $\Rightarrow$ tests have some qualities in them.

- In TDD, what is the primary purpose of writing a failing test before writing the code?

  a. To ensure the test is properly designed to catch potential bugs

  b. To verify that the test framework is working correctly

  c. To help developers understand the requirements before implementing the code

  d. To motivate developers to write better code

Writing Good Tests

- **Clarity:** A test should be **easy to read** and **self-explanatory** about what it’s testing.
- **Independence:** Tests should **not** depend on each other; they should run in **any order** and still pass.
- **Determinism:** A good test should **always** have the same **same result** given the same code (no flakiness).
- **Small & Focused:** Each test covers **one behavior or scenario**, ensuring quick feedback if something breaks.
- **AAA Pattern:** (Arrange, Act, Assert)
  - **Arrange** the test data/environment
  - **Act** on the code
  - **Assert** the results match expectations

Class activity on Github issue

- Using TDD on your project,
- Describe and list the prominent test cases.