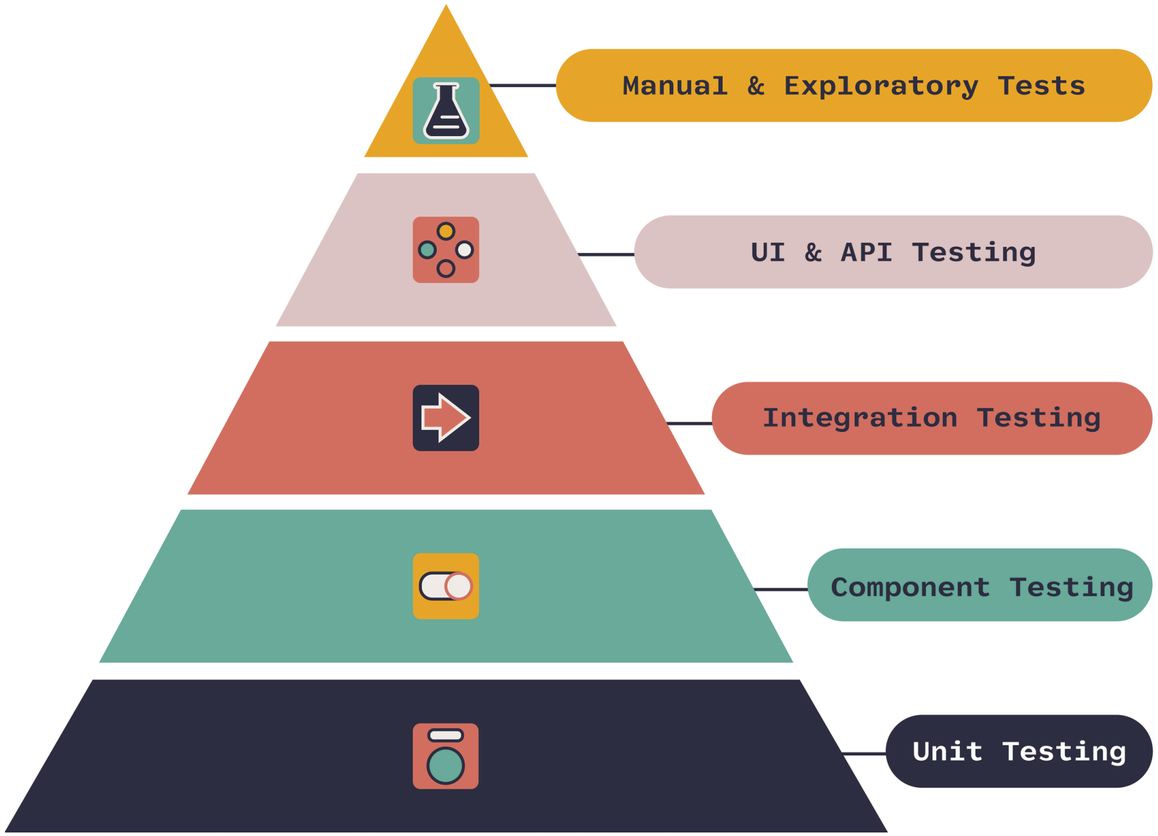
COMP 3550

8.1 — UNIT VS. INTEGRATION VS. END-TO-END TESTS

Week 8: Advanced Testing

THE TESTING PYRAMID (REVISITED)

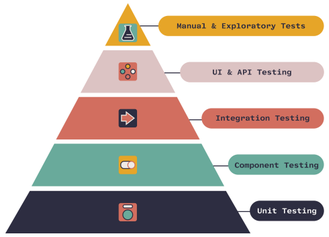


https://www.onpathtesting.com/blog/qa-testers-what-is-the-agile-testing-pyramid

The higher you go, the fewer tests you should write.

The lower you go, the faster, cheaper, and easier to maintain.

THE TESTING PYRAMID (REVISITED)



Layer	Purpose	Traits
Manual /UI Tests	Validate full workflows via the interface	Slowest, fragile, least automated
End-to-End (System)	Tests all layers working together	Slower, brittle, fewer in number
Integration Tests	Ensure components work together correctly	Medium speed, medium coverage
Unit Tests	Test single class/function in isolation	Fastest, most reliable, most of them

UNIT TESTS RECAP

Small, fast, and focused

- A test that verifies the behavior of a single class or function in isolation
- does not rely on external systems like databases, APIs, or UI

Typical Setup:

- **Test target**: One method or class
- **Dependencies**: Replaced with mocks, stubs, or fakes
- **Assertions**: Focus on return values or state changes

INTEGRATION TESTS

Do these parts play nicely together?

- A test that checks if multiple components or modules work together correctly
- Often includes real systems like a database, file system, or API call

Common Examples:

- UserService calls UserRepository which talks to a real test database
- Controller + Service + DAO layers wired together
- File processing logic writing to disk

INTEGRATION TESTS

Do these parts play nicely together?

Purpose	Example
Verify real-world wiring/config	Spring bean configs, DI setup
Test against real dependencies	H2 database, file system, etc.
Catch edge cases missed in unit tests	Transaction failure, null configs

They’re Not:

- full end-to-end
- single isolated class tests
- UI tests

END-TO-END / SYSTEM TESTS

Test the whole app, just like a real user would.

What Are E2E/System Tests?

- Tests that exercise the entire system — from UI or API entry point down through all layers to the real database and back.
- No mocks. No shortcuts. Just real-world behavior.

Examples:

- Simulate a user logging in, placing an order, and receiving a confirmation
- Submit a form and verify that the confirmation screen + database + email all updated correctly
- Run test scripts through the full UI or API

END-TO-END / SYSTEM TESTS

Tradeoffs

Pros	Cons
Highest confidence	Slow (seconds, minutes)
Covers real-world behavior	Brittle (small UI change = fail)
Good for CI "smoke tests"	Harder to isolate bugs

END-TO-END / SYSTEM TESTS

Tradeoffs

Pros	Cons
Highest confidence	Slow (seconds, minutes)
Covers real-world behavior	Brittle (small UI change = fail)
Good for CI "smoke tests"	Harder to isolate bugs

E2E Tests are like dress rehearsals,they won’t catch every typo but they are great to do before opening night

TEST COVERAGE LAYERS

Test Type	What it Covers	What it Ignores
Unit Test	One class/function in isolation	All external systems (mocked)
Integration Test	Multiple modules (e.g., service + DB)	UI, full user flow
E2E Test	Full app, UI/API to DB and back	Nothing, full stack is tested

PROJECT PAUSE & REFLECT

Draw a pyramid of your own project.

Do you have tests at each level?