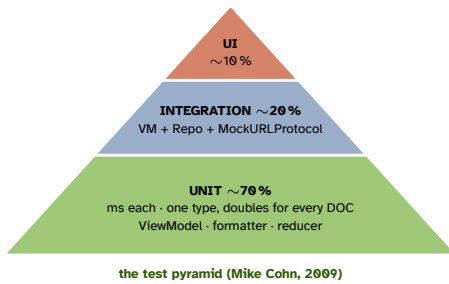


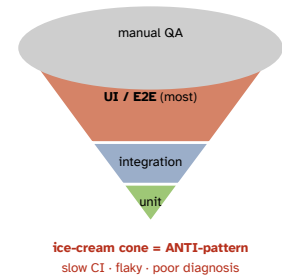
In one line: Many fast **unit** tests (~70%), fewer **integration** tests (~20%), very few **UI/E2E** tests (~10%). Going *up* buys realism and costs **speed, money, stability and failure localisation** — so the shape slopes. Tests are an *executable spec* whose real payoff is the **confidence to change code**.



up: slower (minutes) · costlier to write + maintain · flakier · a red test could be any of 10 layers but closest to what users see

tools: unit = XCTest / Swift Testing
 integ. = MockURLProtocol, in-memory
 Core Data · UI = XCUITest

down: ms · cheap · deterministic
 failure points at one type
 runs on every save (fast feedback)



How it works

- Unit** — one type/function in isolation, *no I/O*, every collaborator replaced by a double. **Integration** — several *real* components together, only the outer boundary stubbed. **UI/E2E** — the built app driven through its screens (XCUITest runs in a *separate process*).
- Why it slopes (4 words): speed · cost · flakiness · feedback** (how fast, how precisely a failure points at the bug). Fix an ice-cream cone by pushing logic *down* into unit-testable ViewModels; keep UI tests for a few critical flows (login, checkout, onboarding).
- SUT** = System Under Test. **DOC** (Depended-On Component) = **collaborator** — anything the SUT talks to (API client, store, clock). A unit test isolates the SUT by replacing DOCs with *test doubles*.
- Behaviour, not implementation:** assert observable outputs and effects, never private fields or “which helper was called” — those break on every refactor.
- Don't test:** Apple's code (URLSession, Codable itself, SwiftUI layout), trivial getters, third-party internals. **Do test:** your logic, branches, boundaries, error paths, one regression test per fixed bug.
- Coverage** = lines *executed*, not behaviour *verified* — a gap finder, not a quality score (100% is possible with zero asserts).

Example — AAA = Given / When / Then

```
@testable import Shop
final class CartTests: XCTestCase {
    func test_givenTenPercentDiscount_whenTotal_thenReduced() {
        // Arrange / Given - SUT + inputs + doubles
        let sut = Cart(items: [Item(price: 100)],
                        prices: StubPriceService()) // DOC -> double
        // Act / When - ONE action
        sut.apply(discount: 0.10)
        // Assert / Then - observable behaviour
        XCTAssertEqual(sut.total(), 90)
        // NOT: XCTAssertEqual(sut.lineCache.count, 1) <- internals
    }
}
```

Swift Testing equivalent: `@Test("10% discount") func discount() + #expect(sut.total() == 90); #require stops the test on failure.`

FIRST — five properties of a good unit test

Fast	milliseconds; you run thousands on every save
Independent	no shared state; passes in <i>any order</i> / in parallel (a.k.a. <i>Isolated</i>)
Repeatable	same result on any machine, offline, any day (no real Date(), network, randomness)
Self-validating	pass/fail by itself — no human reads a log
Timely	written with/just before the code (TDD), not months later

Interview traps

- ! Asked “describe the pyramid”: give **three layers + numbers + the why**. “No clue” was the recorded answer — the minimum is 70/20/10, *cost and speed grow upward*.
- ! “90% of our tests are UI tests” = **ice-cream cone** (inverted pyramid). Not “great coverage”.
- ! A test reading a file an earlier test wrote violates **Independent** (and **Repeatable**) — Xcode can run tests in parallel and in *random order*.
- ! “Hit the real API in unit tests for realism” breaks F, I and R; realism belongs in a few contract/integration tests.
- ! **One logical assertion** per test (several XCTAsserts on one outcome are fine); unrelated asserts = ambiguous failure.
- ! A bug-fix test written *after* the fix that you never saw red proves nothing — reproduce red first.
- ! Flaky test = passes/fails with no code change; causes: sleep, shared state, order, real network, Date()/UUID(). It erodes trust in red.

Remember

“70 · 20 · 10 — up is Slow, Spendy, Shaky, Shallow-feedback.”
 Good test = **FIRST** + **AAA** + asserts *what*, not *how*. Name = the spec line: `test_<given>_<when>_<then>`.

Likely questions

- Pyramid layers + ratio? — unit 70, integration 20, UI 10.
- Why that shape? — higher = slower, costlier, flakier, worse localisation.
- What is the ice-cream cone? — mostly UI/manual tests; the anti-pattern.
- What does FIRST stand for? — Fast, Independent, Repeatable, Self-validating, Timely.
- SUT vs DOC? — thing tested vs the collaborators it depends on.
- Regression vs smoke? — pins a fixed bug vs shallow “app launches” check.