Testing Code

Write code which runs other code making sure everything works!

Contents

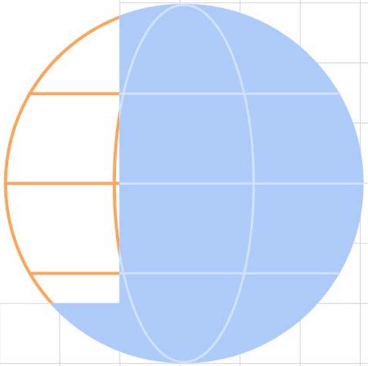
GDG

What is testing?

Basically, software testing is like carefully checking if an app or program **works right**. We want to see if it does what it's **supposed** to do, and if it's **good quality.**

GDG

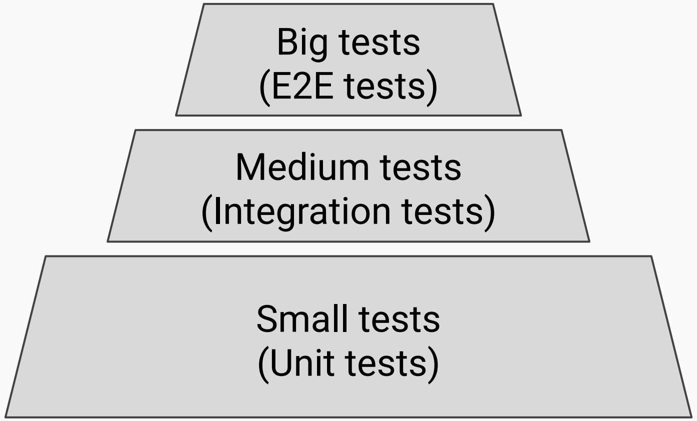
Types of testing



Types of tests

- Black box testing
- White box testing
- Performance testing
- Load testing
- Unit testing
- Integration testing
- UI Testing
- …

The Testing Pyramid



GDG

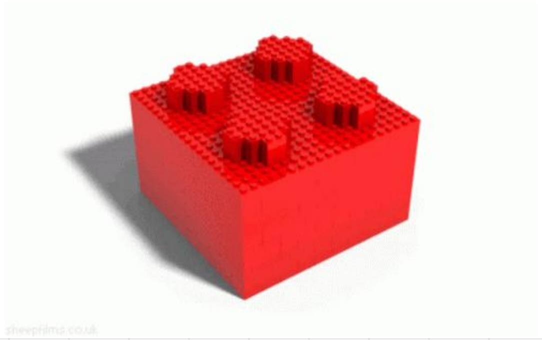
Black Box testing

Blackbox testing verifies application functionality from a **user's perspective**, without code knowledge, often using user stories and frameworks like Appium.

GDG

Unit testing

Unit testing is like double-checking every **small** part of your app's code to make sure it works perfectly. It helps you find and fix problems **early**, so your app runs smoothly.



Why Should I Write Unit Tests?

- **Catch mistakes early** - saves time and money
- **Easier maintenance** - allows you to make changes without breaking things
- **Quick and easy to run** - big number of tests in small amounts of time, often
- **Build developer confidence**
- **Documentation** - always up-to-date, shows edge-cases and concrete examples
- **Clean code** - you have to write good code to be able to test it

GDG

Integration testing

Essentially, integration testing verifies that **different parts** of your Android application function correctly when **combined**.

Why Should I Write Integration Tests?

- **Detecting Interface Issues** - how different components cooperate
- **Verifying Data Flow** - data passed correctly between components
- **Testing external interactions** - validate app communication with services (api, db...)
- **Real-World Scenarios**

End to End/UI testing

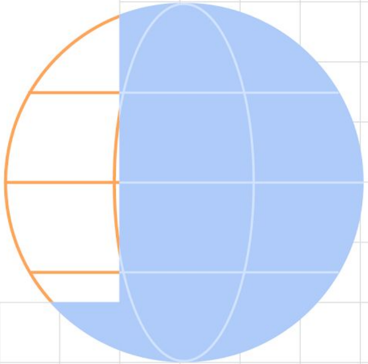
Benefits of E2E Testing

- **Comprehensive Validation** - multiple screens, interactions, and data flows
- **Real-World Simulation** - mimics the actual usage of the app
- **Detecting regression bugs** - adding new code may break old code
- **Increased Confidence** - build confidence in the app stability

Tools for E2E Testing

- **Espresso** - official UI testing framework
- **UI Automator** - A framework for cross-app functional testing
- **Emulator -** allows you to test your app on different screen sizes and device configurations
- **Real devices** - provide the most accurate representation

JUnit



JUnit is the fundamental framework for unit testing in the Java Virtual Machine (JVM) ecosystem. Since Kotlin runs on the JVM, JUnit is directly compatible.

JUnit benefits

- Android Studio integration
- Test organization - @Test, @Before, @After
- Assertions
- Running tests locally (without Android device)

Mockito/Mockk

Popular mocking frameworks for Java & Kotlin.
Most likely you will use on of the two.

What They Do?

- **Mocking dependencies** - unreal objects used for testing
- **Stubbing** - providing “fake” but consistent responses from invocations
- **Verification** - verify that something happened
- **Isolation** - allow you to isolate units of code
- **Improve testability** - make your testing code easier

GDG

Test Doubles

Fakes

- **Natural** behavior, but fake implementation
- Example: In-memory database
- **Doesn’t change** the behavior of the system under test, but **simplifies** the implementation

Stubs

- **Behaves unnaturally** - preconfigured with specific inputs & outputs
- Used to get the system under test into a **specific state**
- You decide if something is a stub based on its **purpose**.

Mocks

- Similar to a stub, adds **verification**
- The purpose of a mock is to make assertions about how your system under test interacted with the dependency.
- Used when doing **interaction** testing

GDG

Arrange, Act & Assert (AAA)

Arrange, Act & Assert

- Each piece of code has three sections
- **Arrange:** Setup (parameters, data transformations before API calls…)
- **Act:** The “core” functionality (calling the API, fetching more data…)
- **Assert:** Final steps (returning values, expecting a result…)

Arrange, Act & Assert

- This generally applies to testing code, but you can observe it in regular code as well

GDG

Verifications

Verifications

- There are two main types of verifications:
  - **Assertions**: Expect a value to be present or not, otherwise the test crashes (fails)
  - **Mock Verifications**: Expects an interaction (or lack of) with a certain test double.

GDG

Explicit Verifications

Explicit Verifications

- When writing verifications and/or assertions, always make sure to cover all possible cases to avoid unwanted side effects.
- You might expect that ***UserRepositor.getUser()*** is called, but you don't want calls to ***AccountRepository.getUserAccount()***.

GDG

Why Testing Matters?

Why testing matters?

- Risk mitigation
- Cost-effectiveness
- Enhancing security
- Brand reputation
- Improving User experience
- Historical documentation
- Compliance with regulations