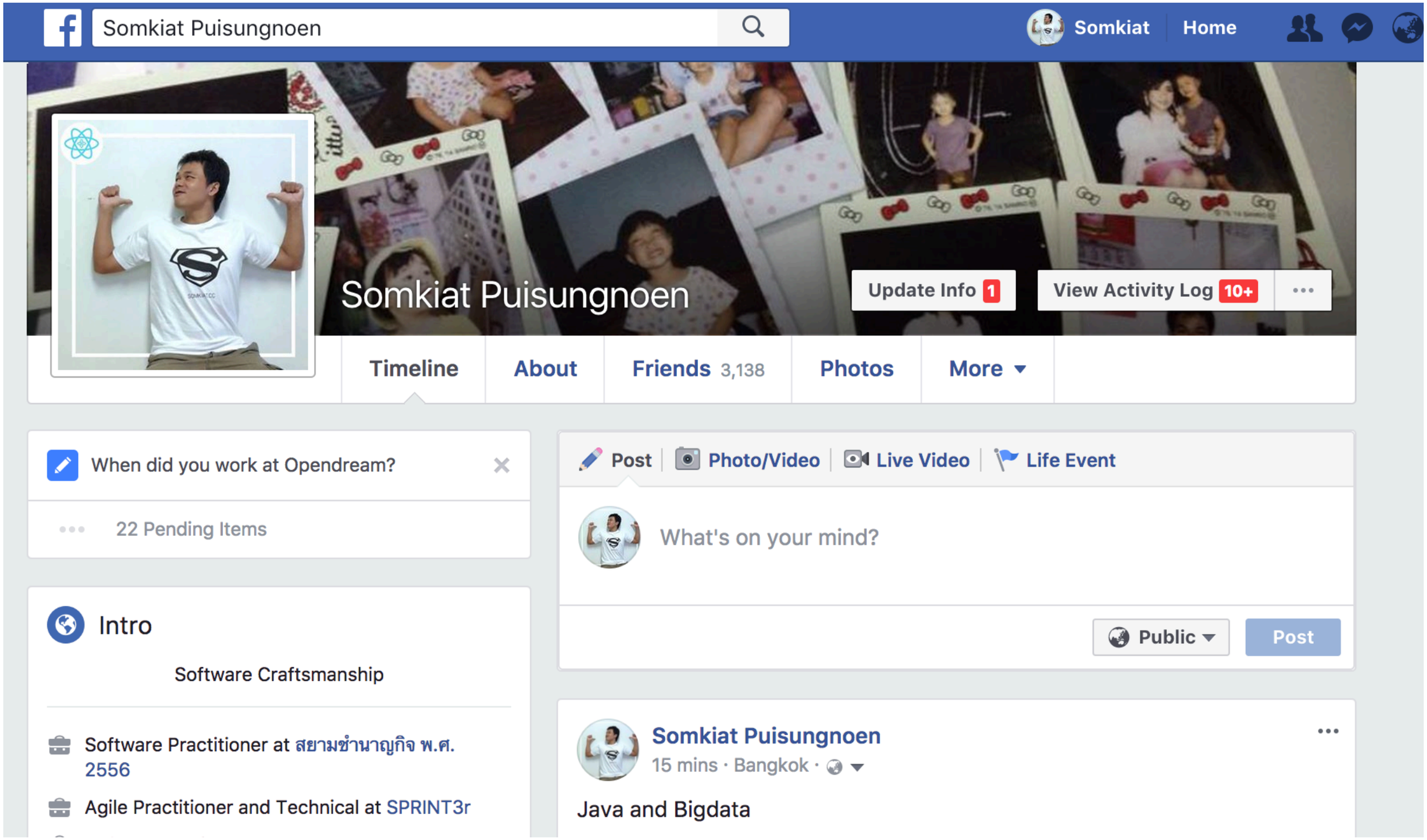
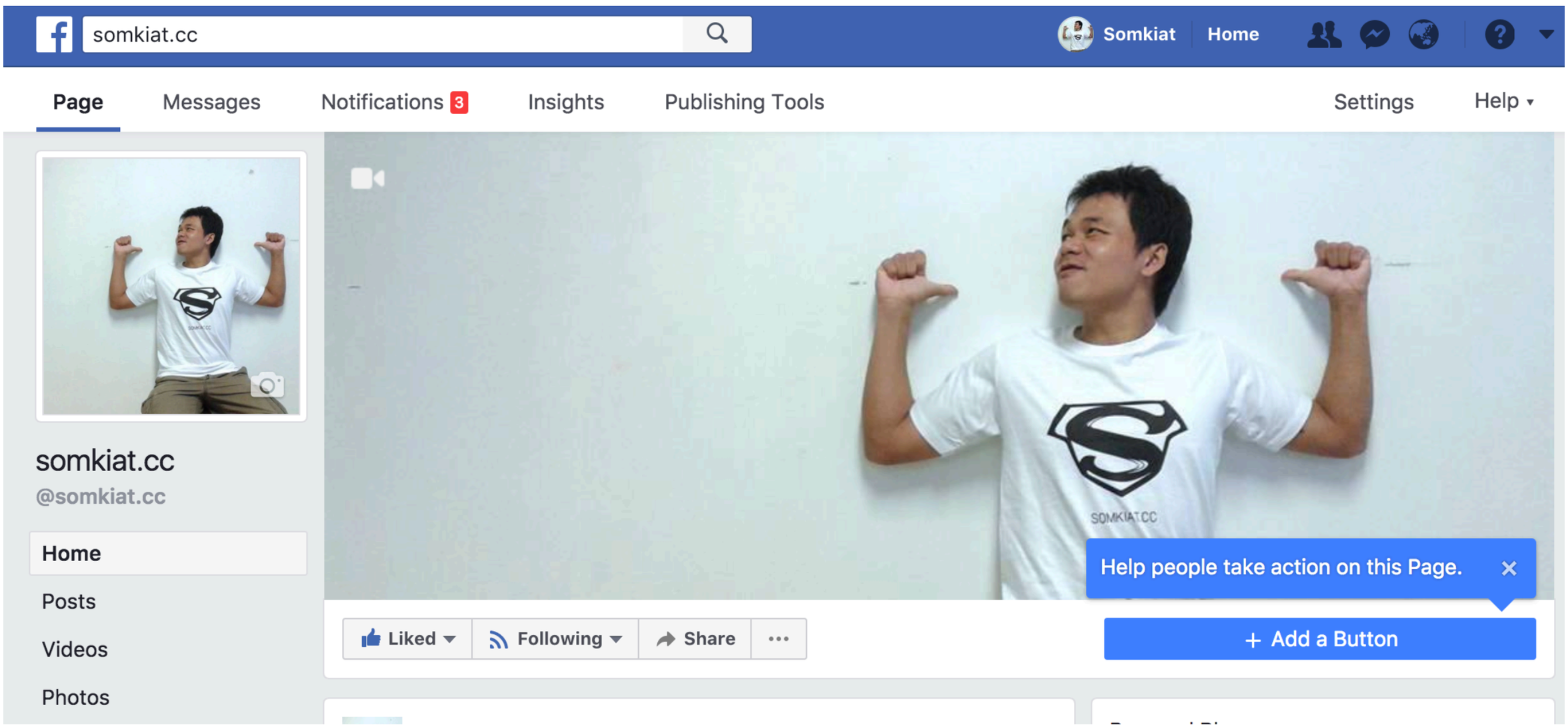
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Automated testing for Android app





Agenda

- Introduction of testing
- Why we need to test ?
- Types of tests
- Testing pyramid concept
- Android testing
- Workshop (step-by-step)

Agenda

- UI testing with Espresso and Kakao
- Expresso workshop
- Testable application
- Code coverage
- Black box testing with Appium

Testing for Android app

"Program testing can be used to show the presence of bugs, but never their absence."

Edsger Dykstra, 1970, Notes on Structured Programming

Why we need to test ?

Help you to **catch bugs**
Develop features **faster**
Enforce **modularity** of your project

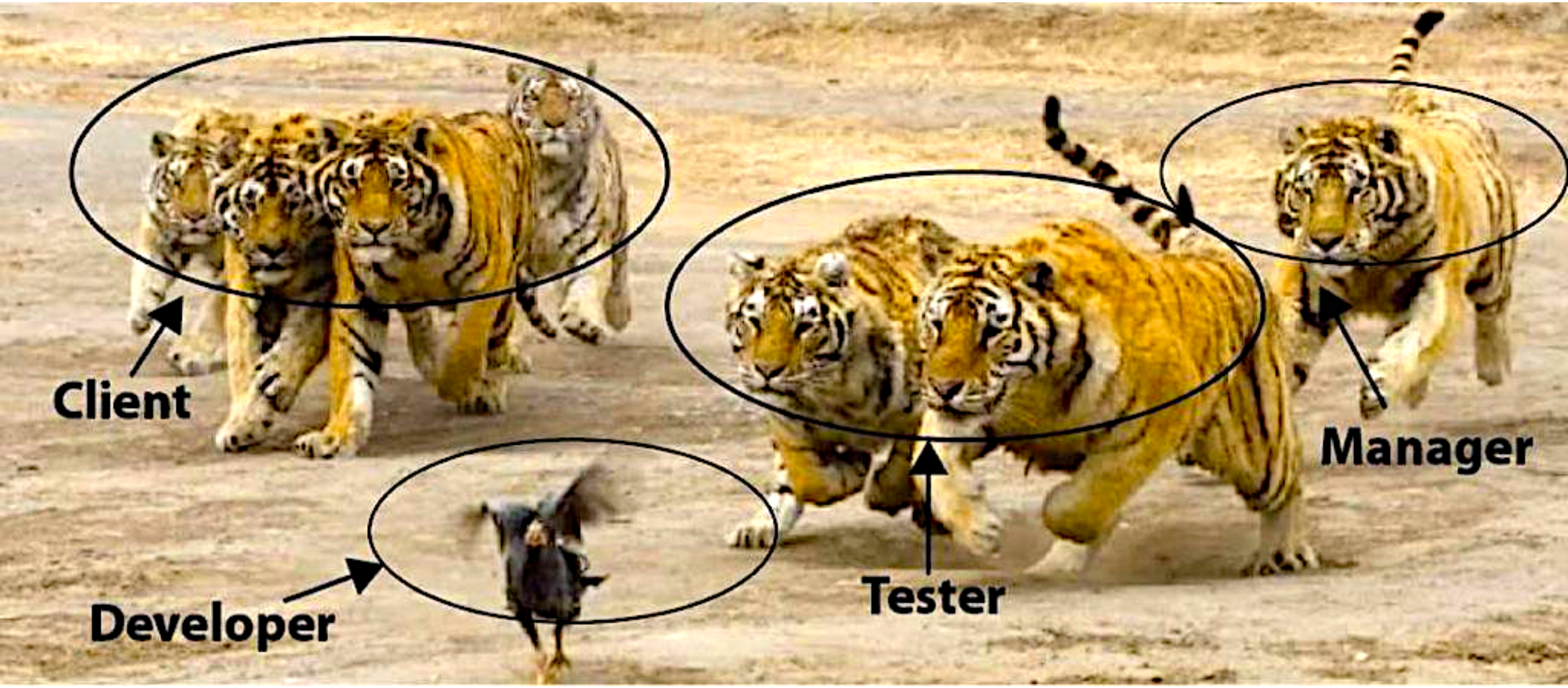
But,

It’s <u>take time</u> to learning and practice !!

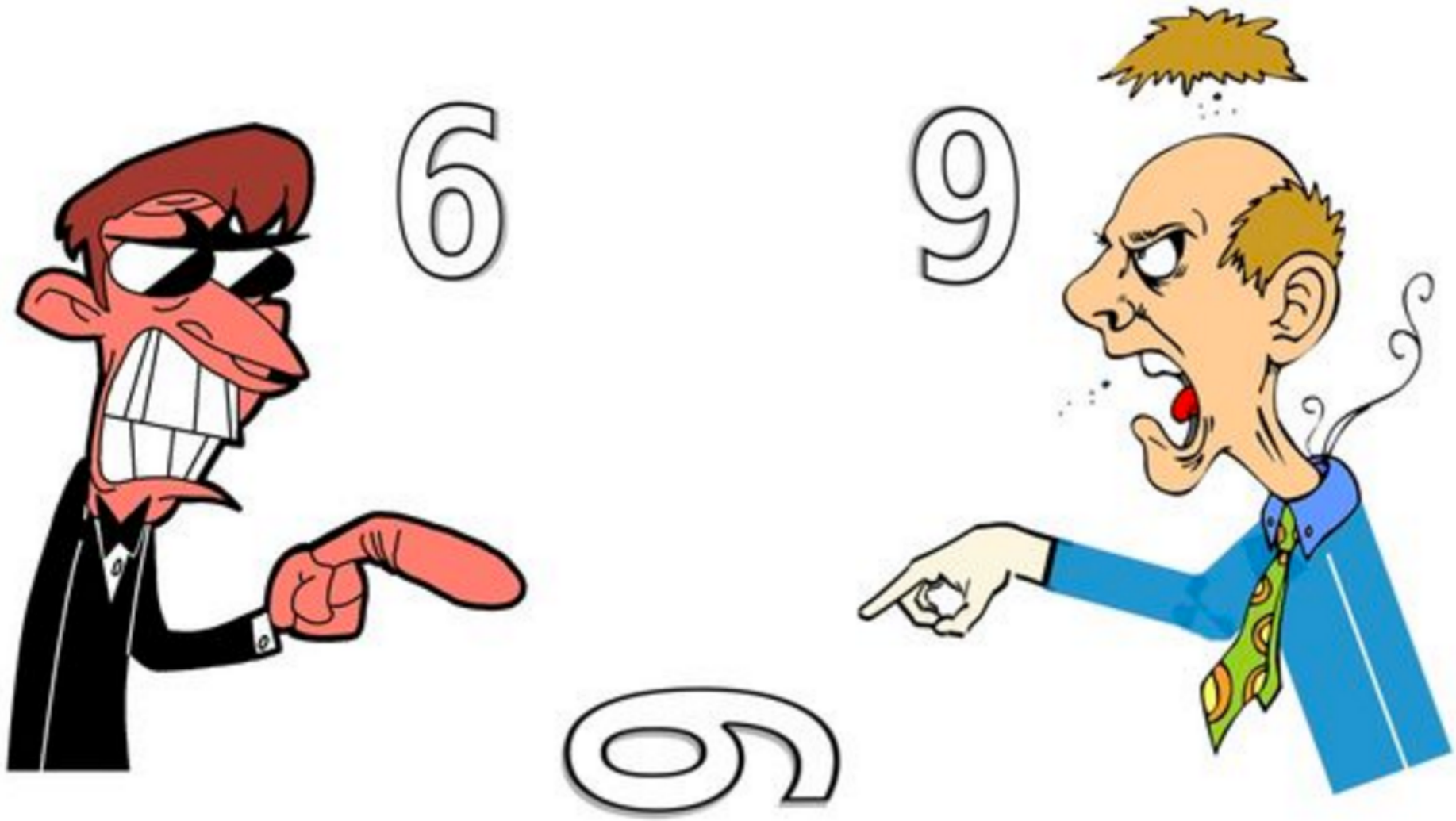
Goals

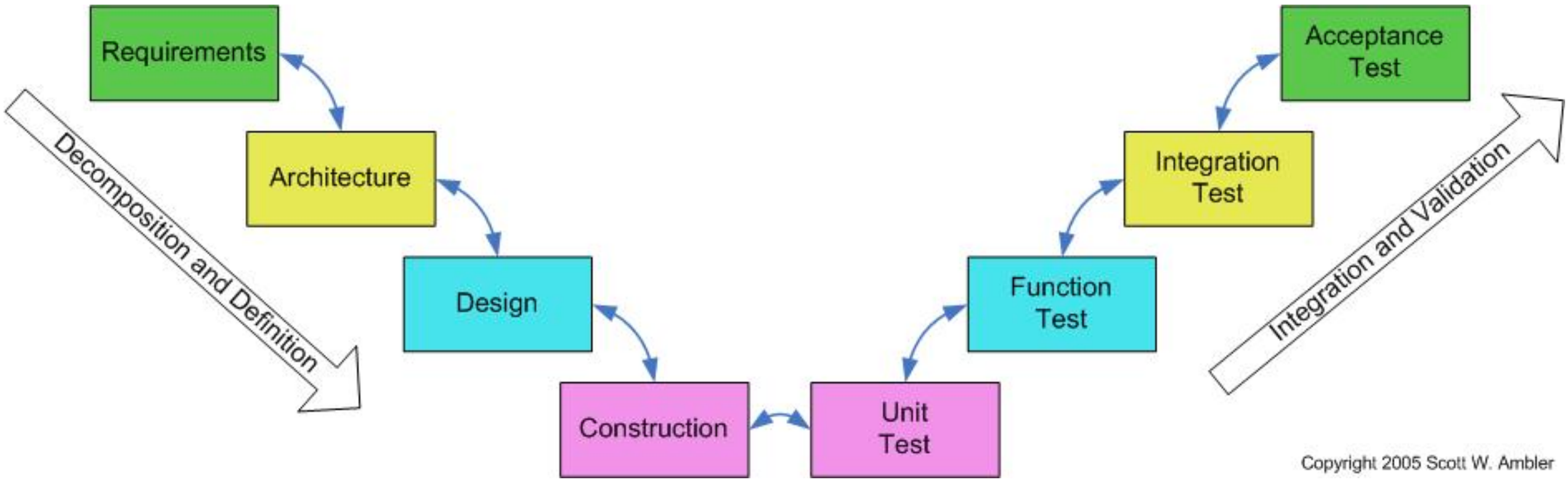
How to **THINK** when and where you should test ?

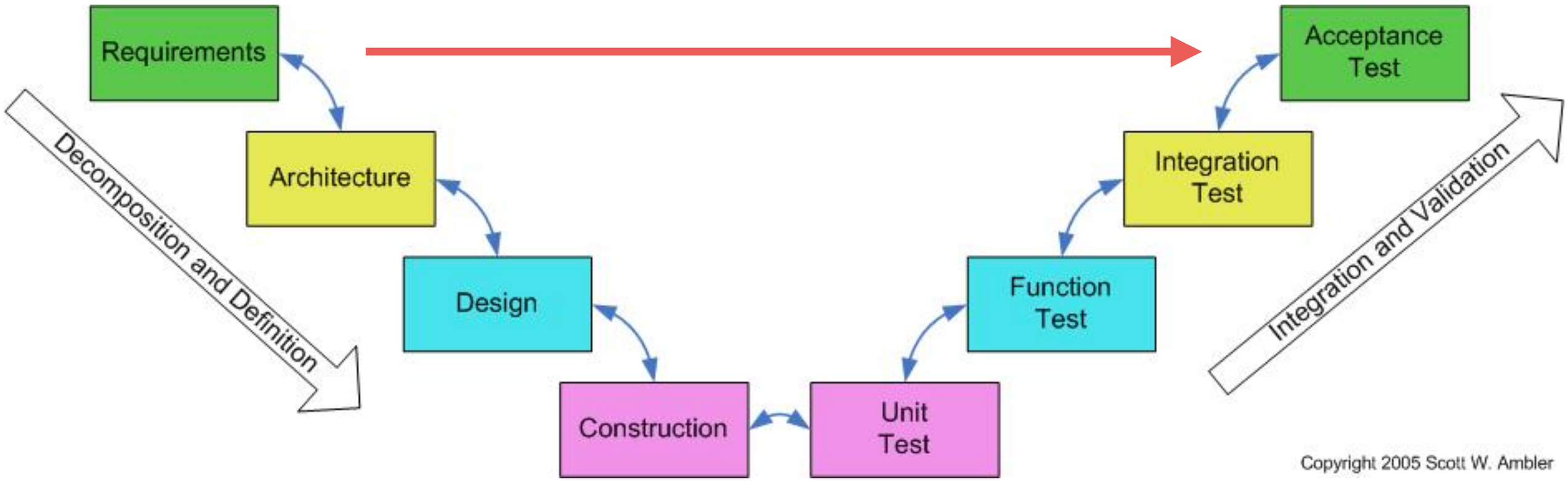
All about Testing

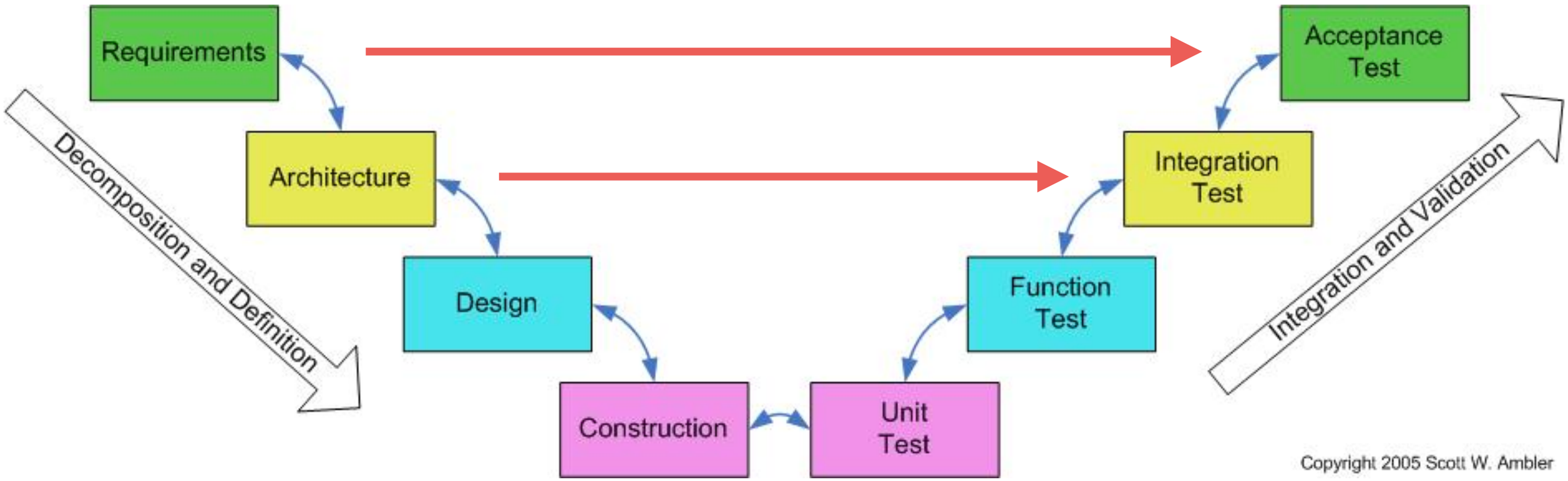


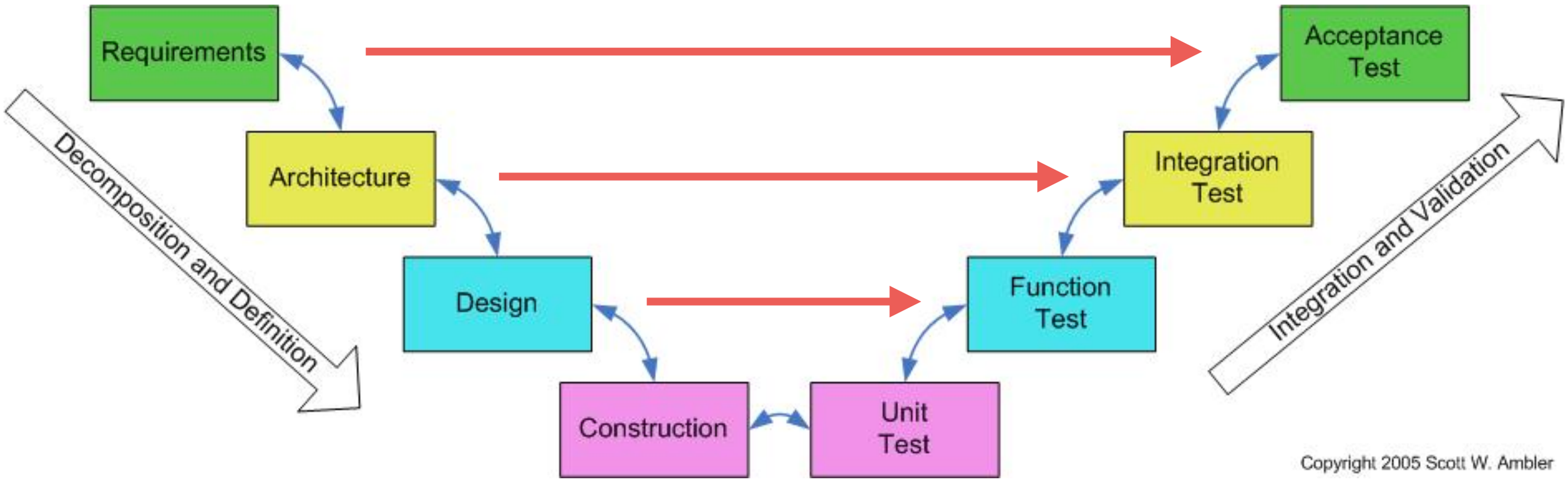
Developer vs Tester

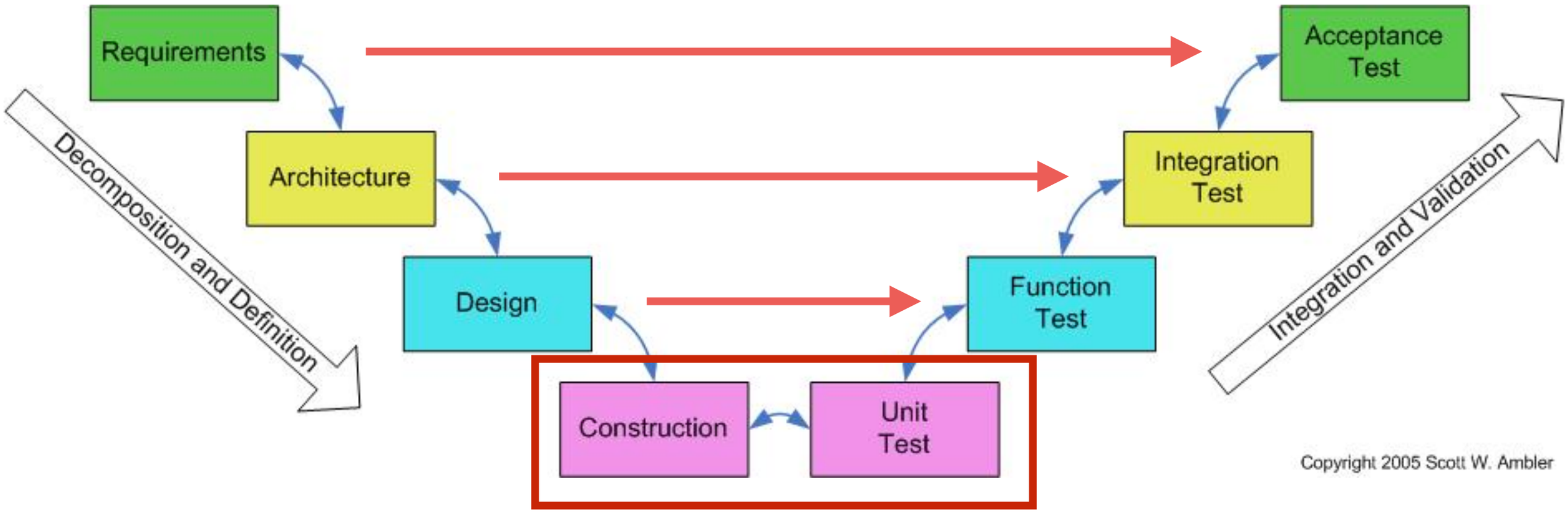




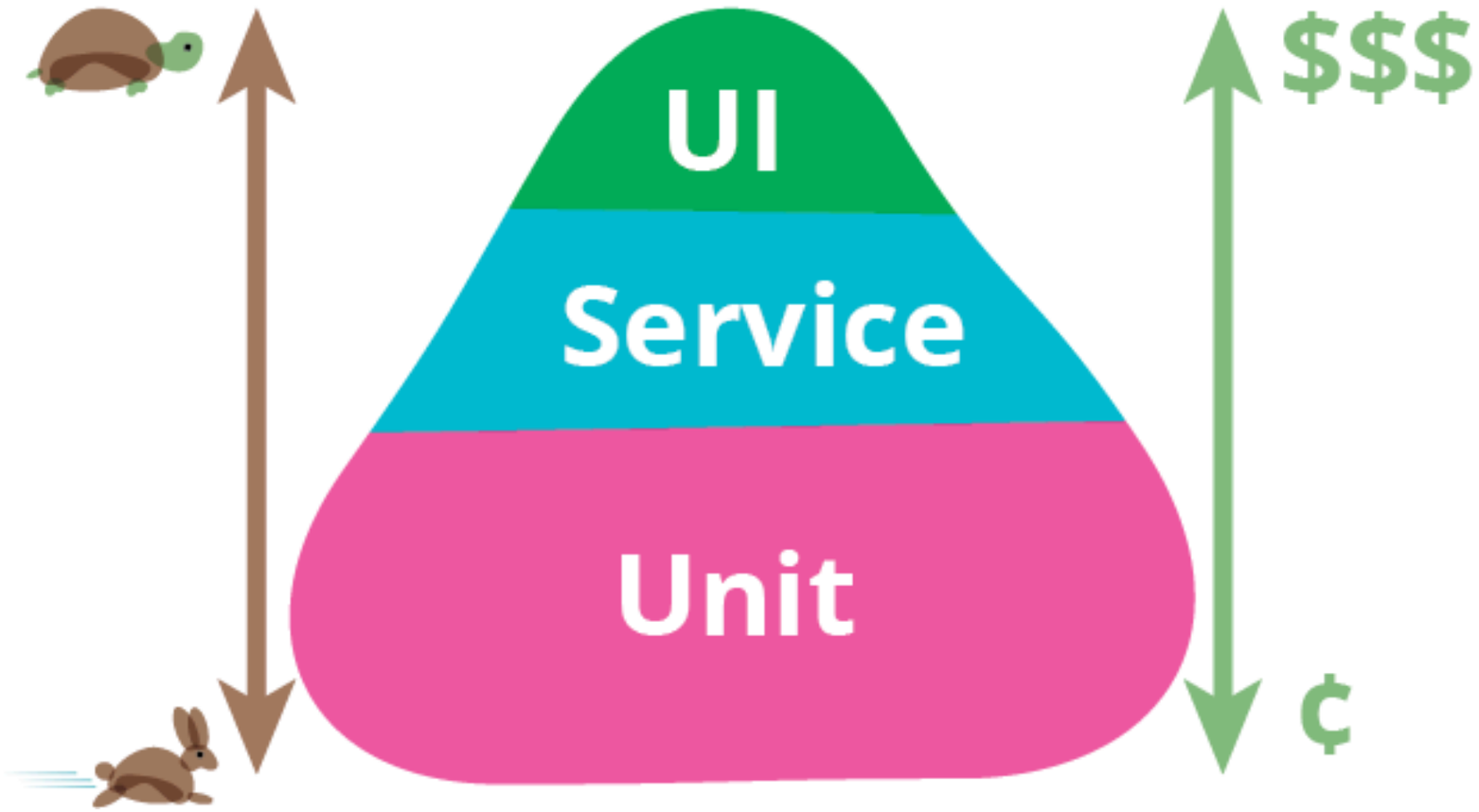






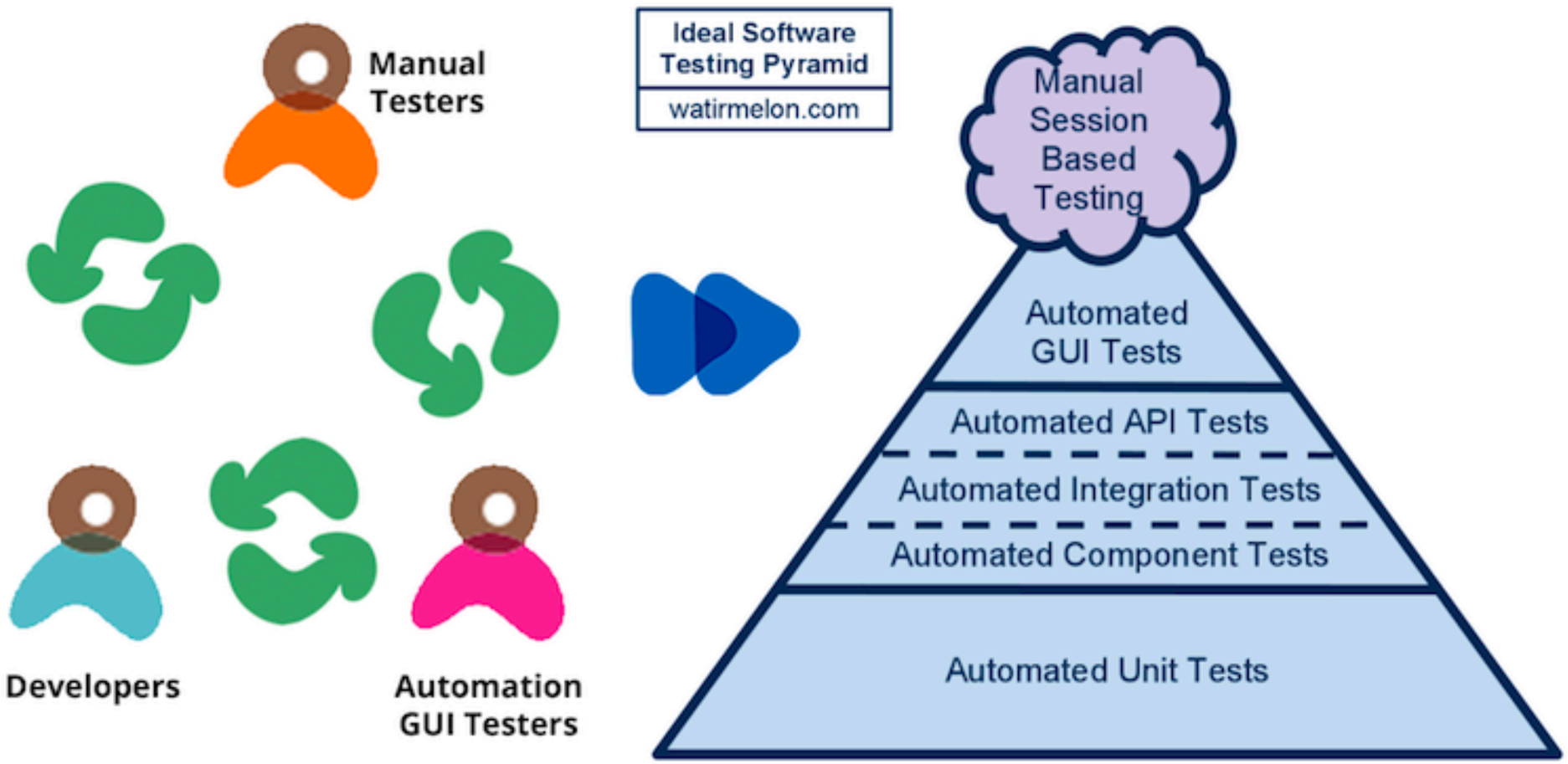


Testing Pyramid

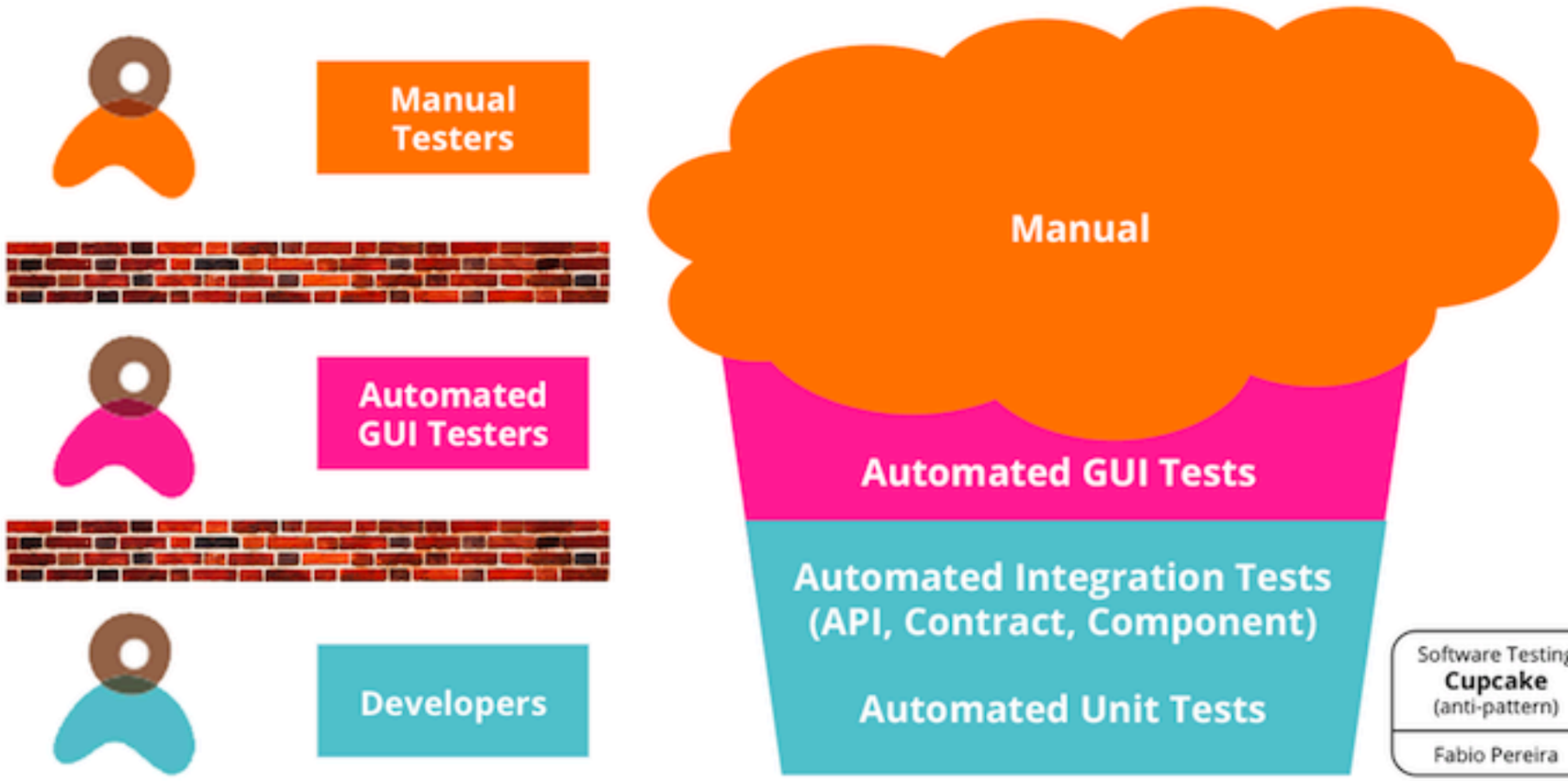


https://martinfowler.com/bliki/TestPyramid.html

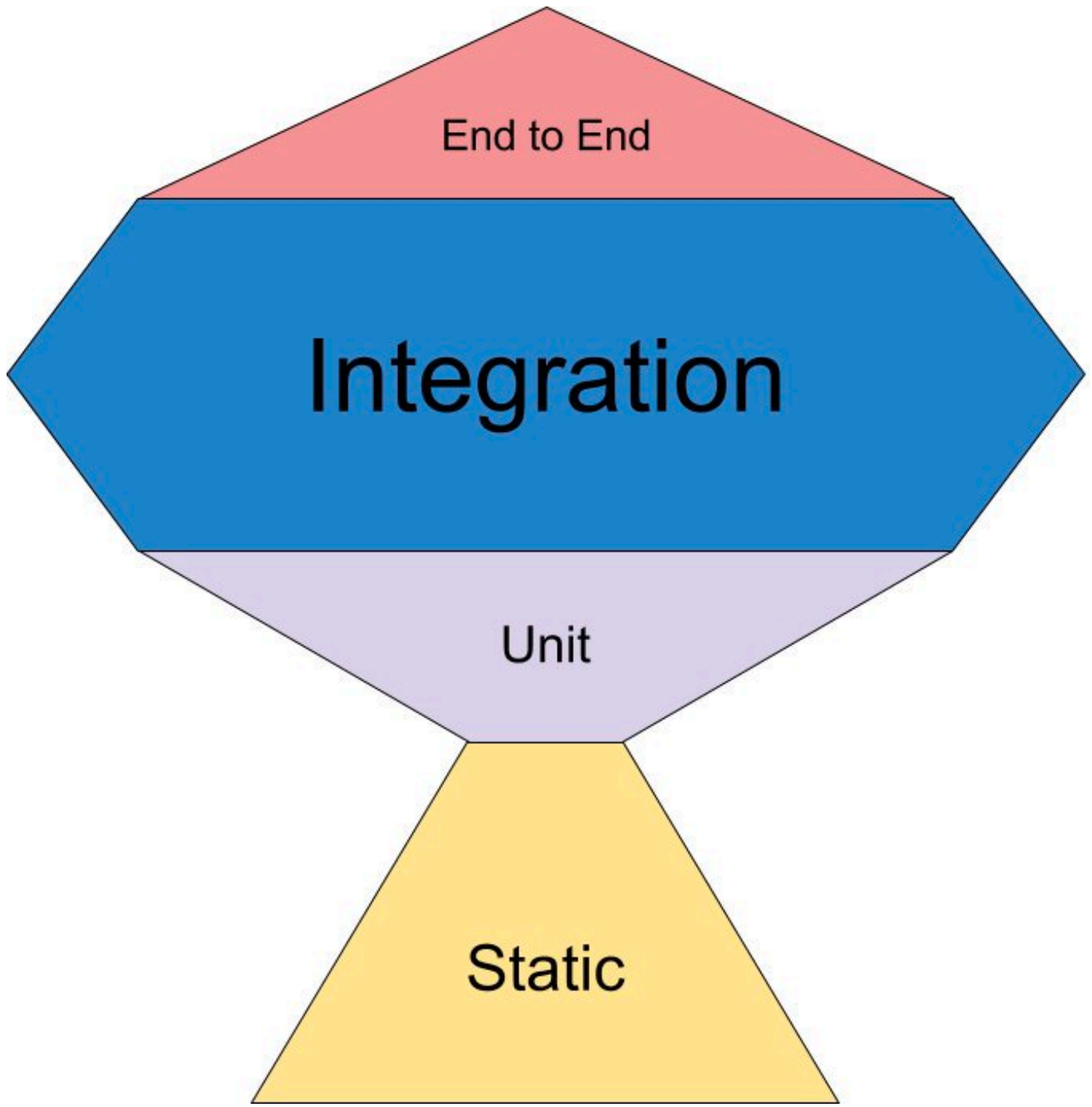
Pyramid testing



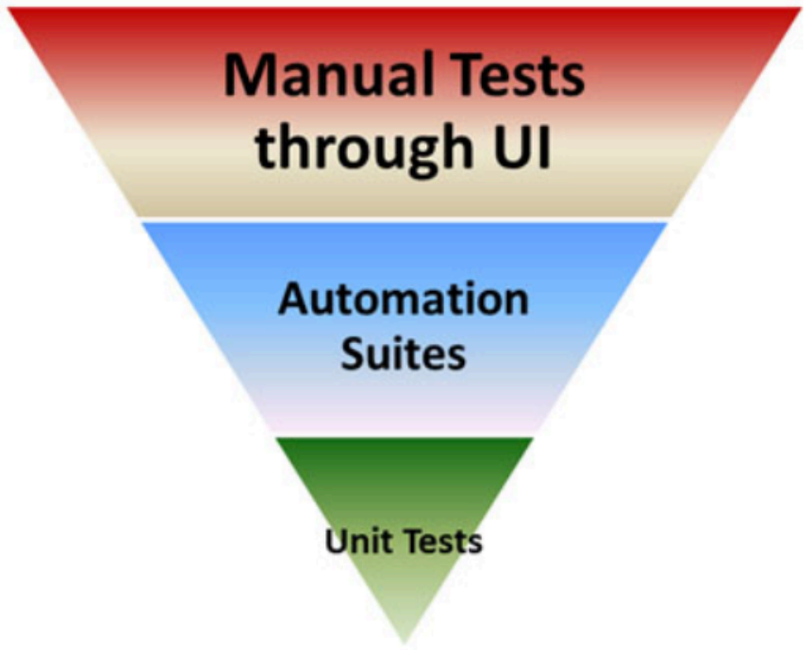
Cupcake testing



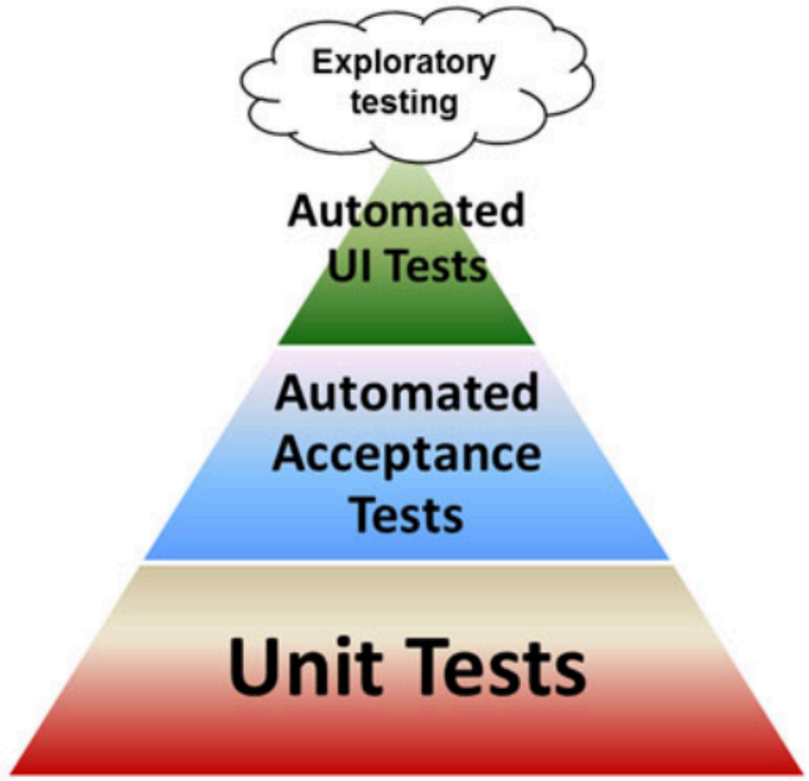
Trophy testing



Testing Pyramid



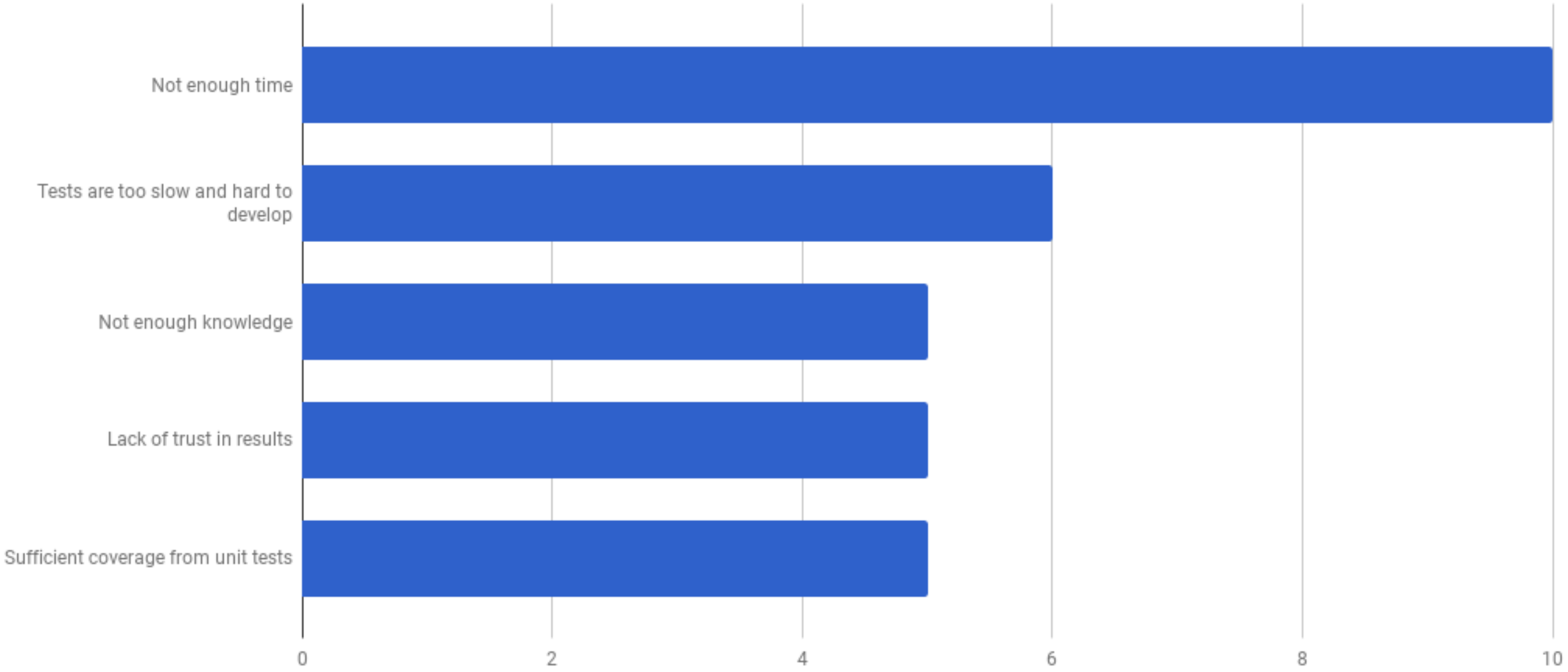
Traditional
(find bugs)



Agile
(prevent bugs)

Testing not a phase
Testing is activity

Why not write automated tests ?



https://slack.engineering/android-ui-automation-part-1-building-trust-de3deb1c5995

Android Testing

Android Testing

Rapid feedback on failures
Early failure detection in development cycle
Safer code refactoring
Stable development velocity

https://developer.android.com/training/testing

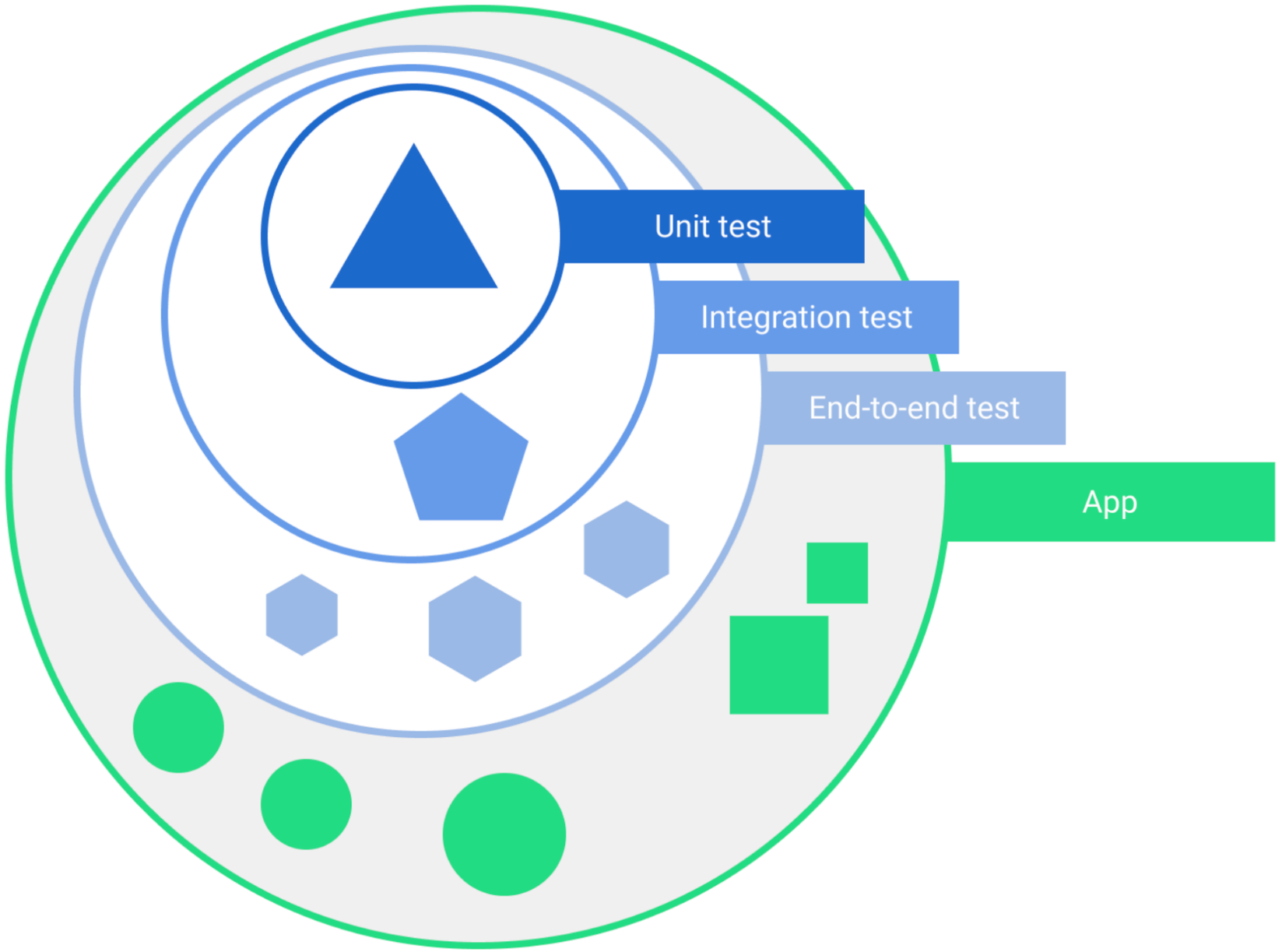
What you need to know ?

Java/Android/**Kotlin**
Android Studio
JUnit 4
Espresso

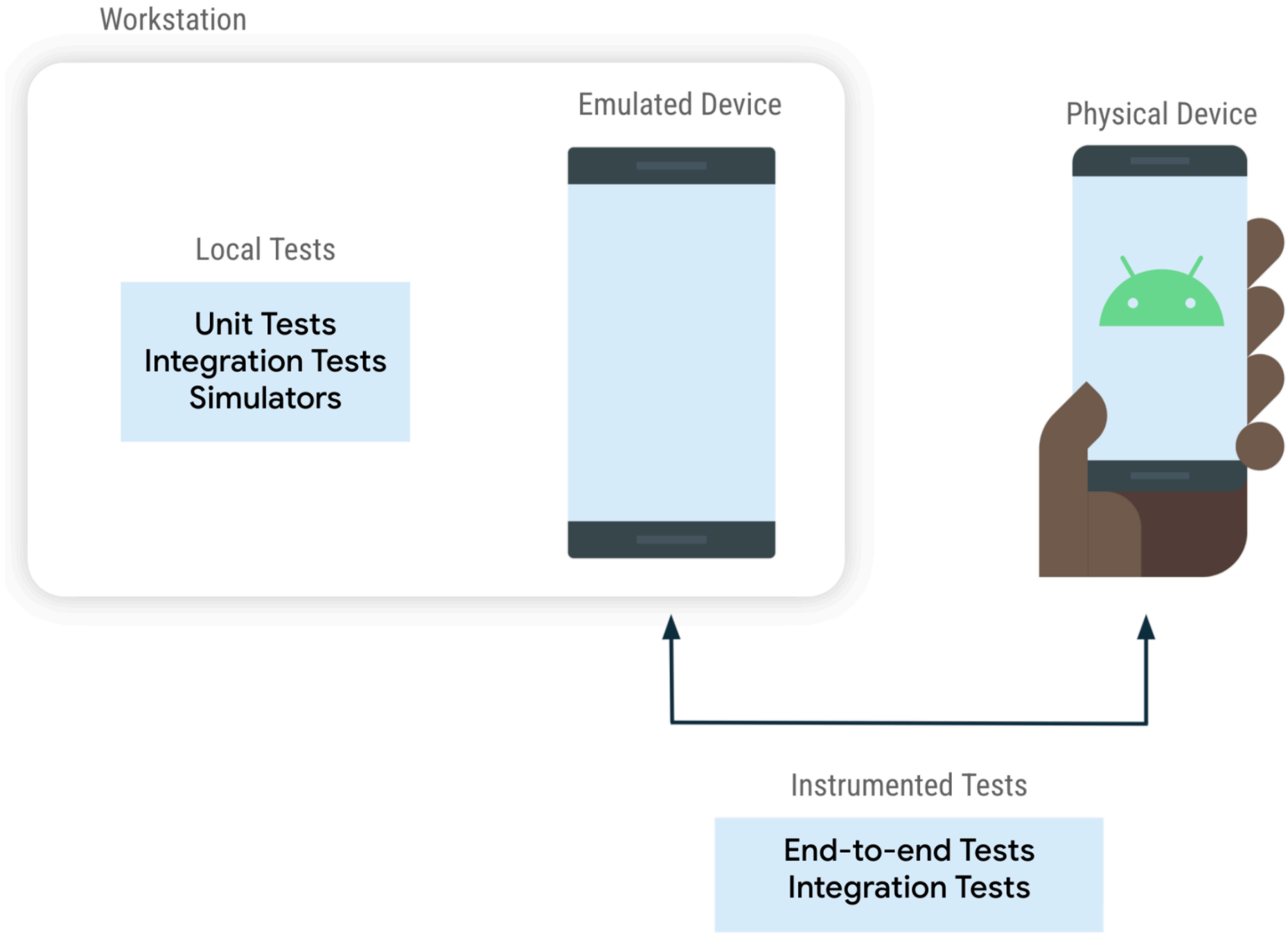
Types of tests in Android

Testing	Goal
Functional	Does my app do what it's supposed to ?
Performance	Does it do it quickly and efficiently ?
Accessibility	Does it work well with accessibility services ?
Compatability	Does it work well on every device and API level?

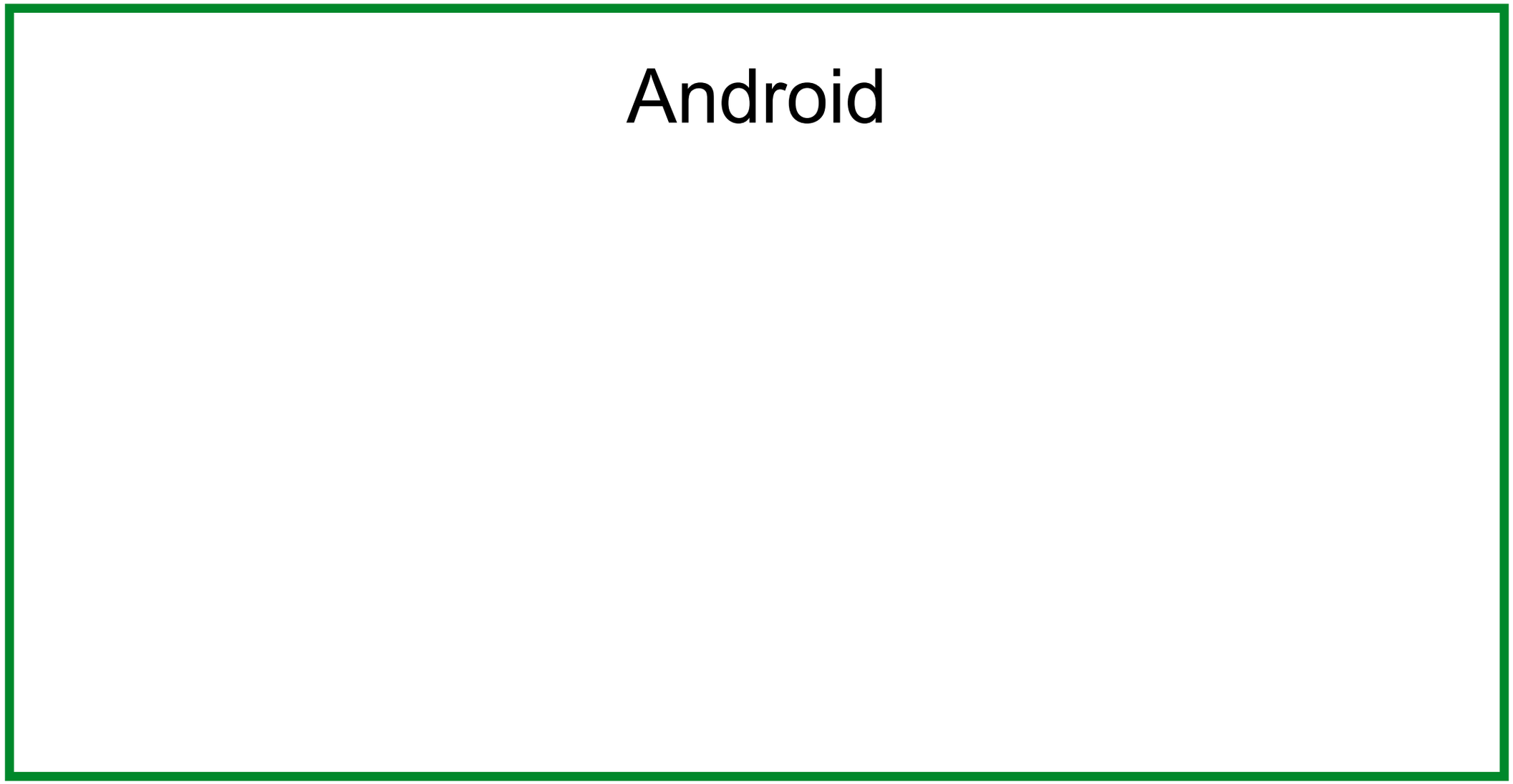
Scope of tests



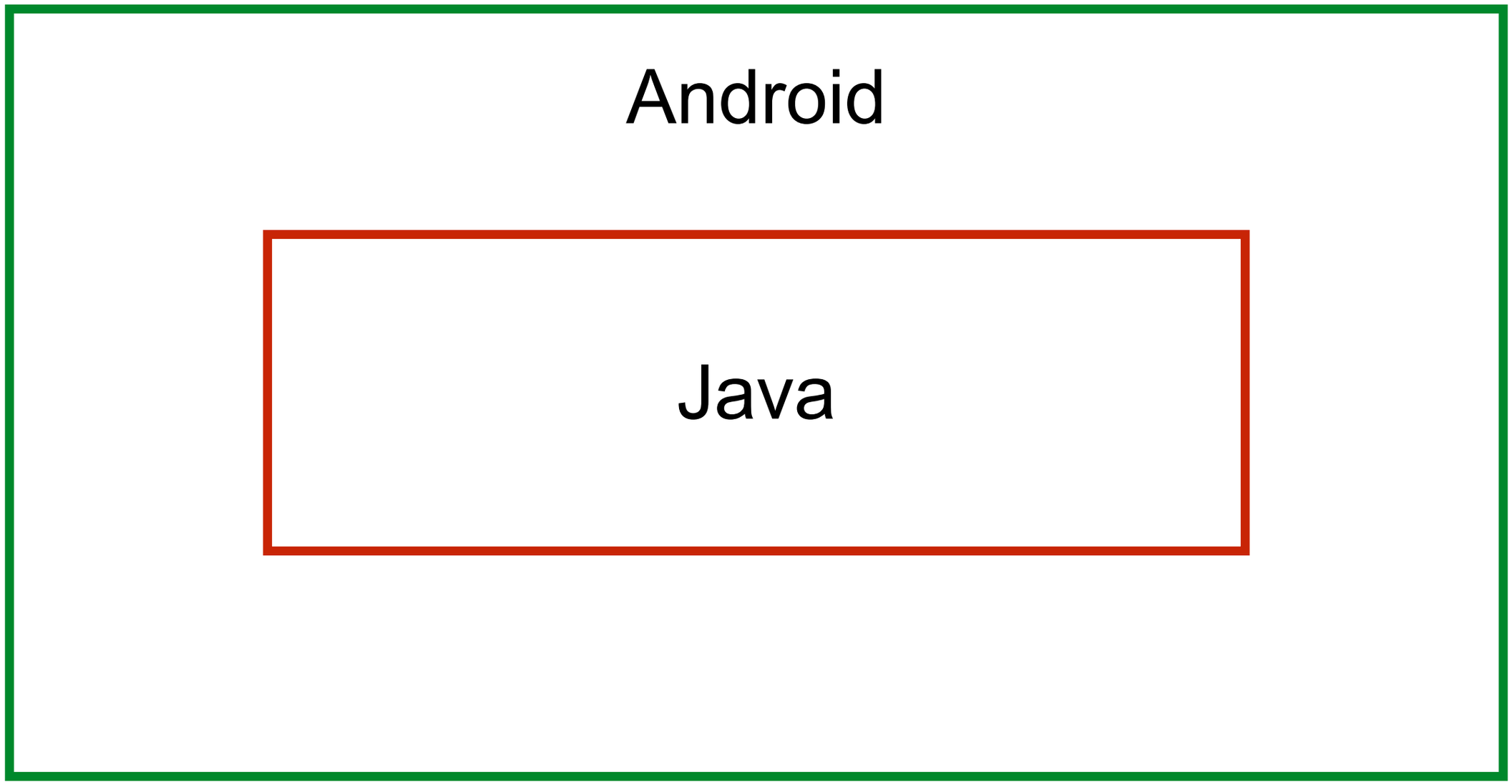
Different types of instrumented tests



Android Testing



Android Testing



Java run on JVM

JVM (Java Virtual Machine)

Run android need device

Build app -> Install to device -> Test

Android Testing

JVM

Device

/src/test

/src/androidTest

JVM unit test

JVM

Device

JVM unit test

/src/test

Business logic with pure java code

Instrumentation unit test

JVM	Device
JVM unit test	Instrumentation unit test
/src/test	/src/androidTest

Working with Android specific code, you need run on device such as AssetManager, SharedPreference

UI vs Non-UI

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI		

/src/test

/src/androidTest

Instrumentation UI test

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI		Instrumentation UI test

/src/test

/src/androidTest

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI		Instrumentation UI test

UI TESTING FOR ANDROID
espresso

/src/test /src/androidTest

UI test on JVM ?

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI	? ? ?	Instrumentation UI test

/src/test

/src/androidTest

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI	Robolectric	Instrumentation UI test

/src/test

/src/androidTest

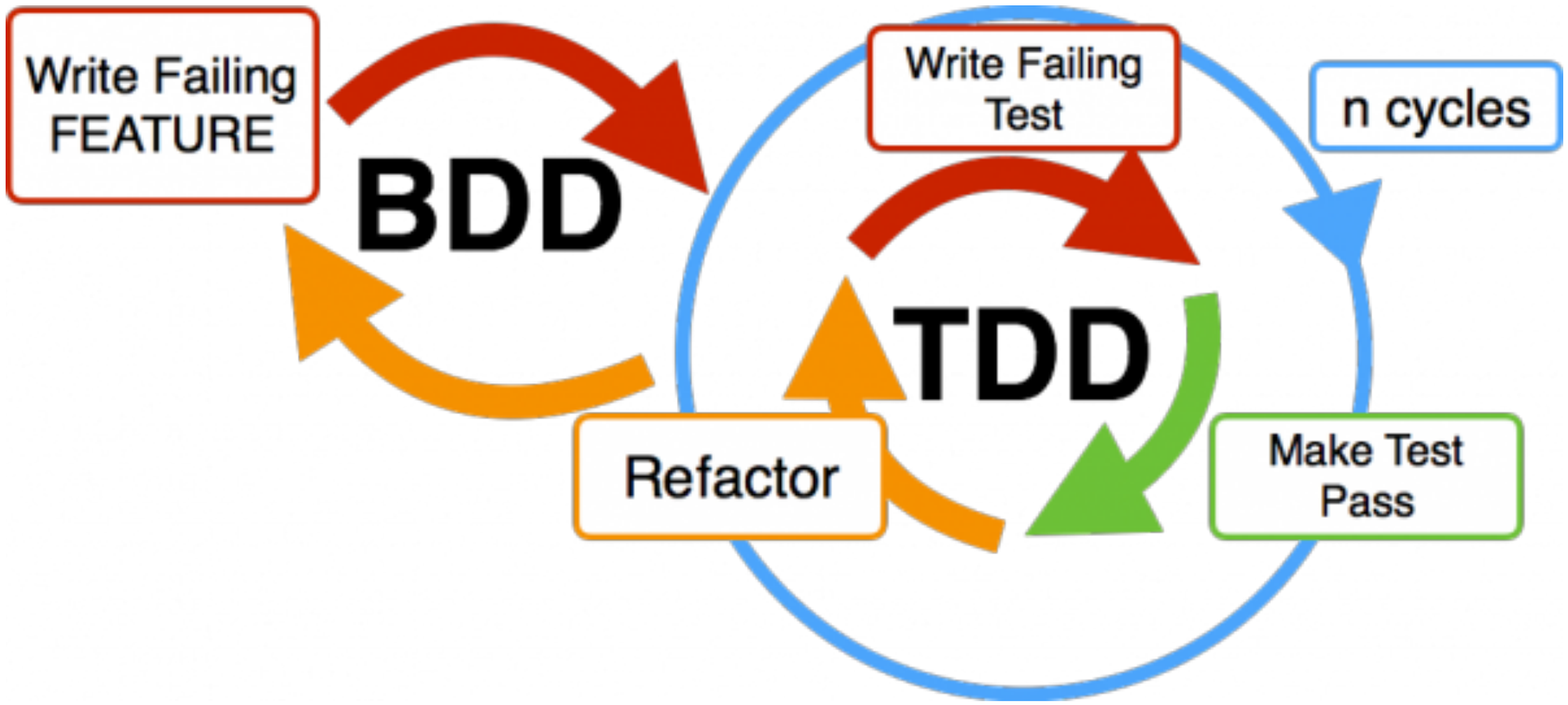
Rule of thumb

Instrumentation tests are slower than JVM tests.

Try to separate the standard Java code from Android-dependent code.

Start to write UI testing

Using espresso



Let’ start to write UI tests

UI tests in Android

Screen UI tests (single screen)
User flow tests or Navigation tests

What is Espresso ?

A simple API for writing reliable UI tests
Develop by Google team
Add to Android Testing Support Library in 2015

https://developer.android.com/training/testing/espresso/

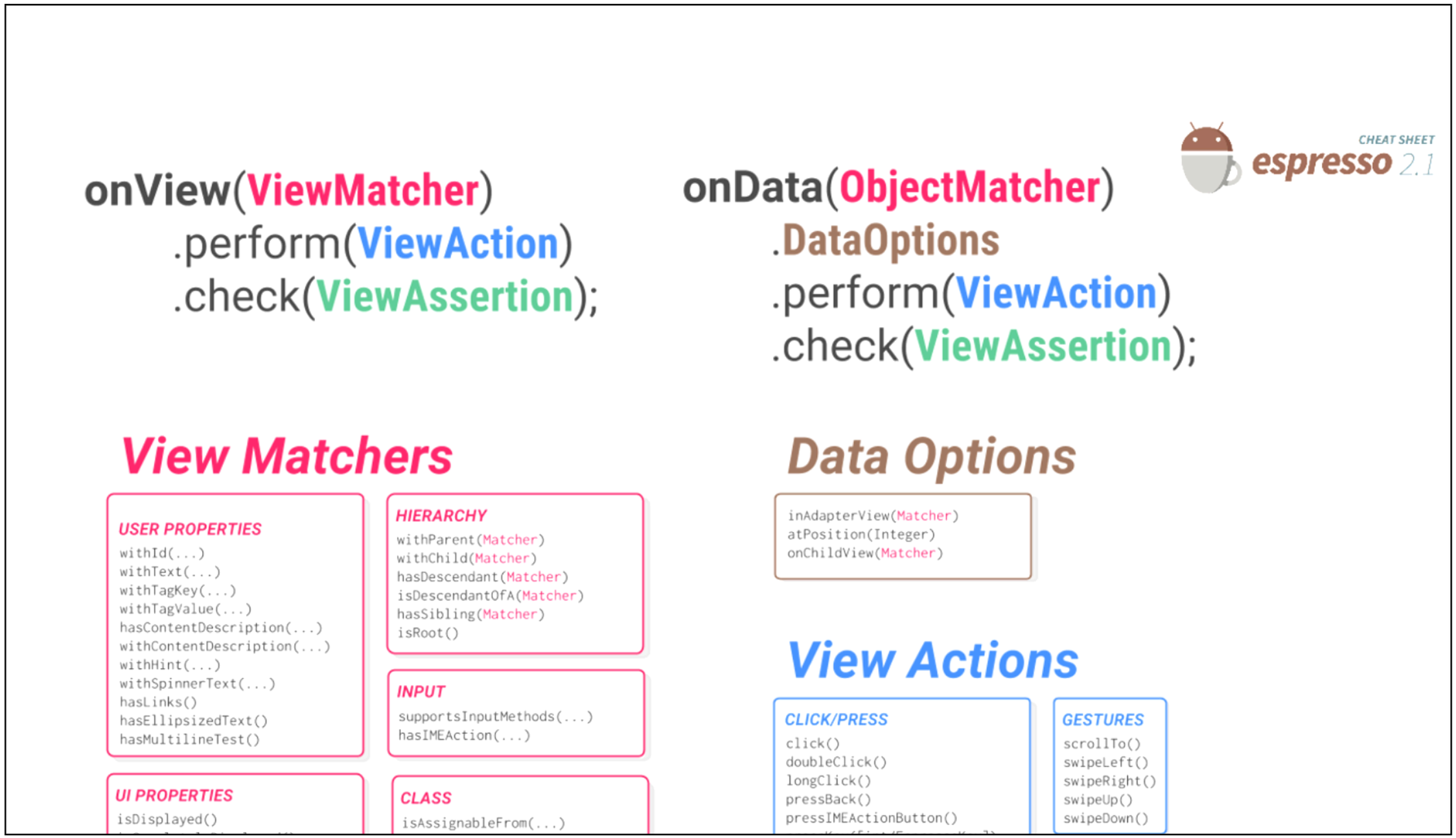
Why use Espresso ?

Small API and predictable
Easy to learn and understand
Extensible
Fast - UI thread synchronization
No wait, No Sleep
Support JUnit4

Android Studio support by default

Try to create a new project
Open file /app/build.gradle

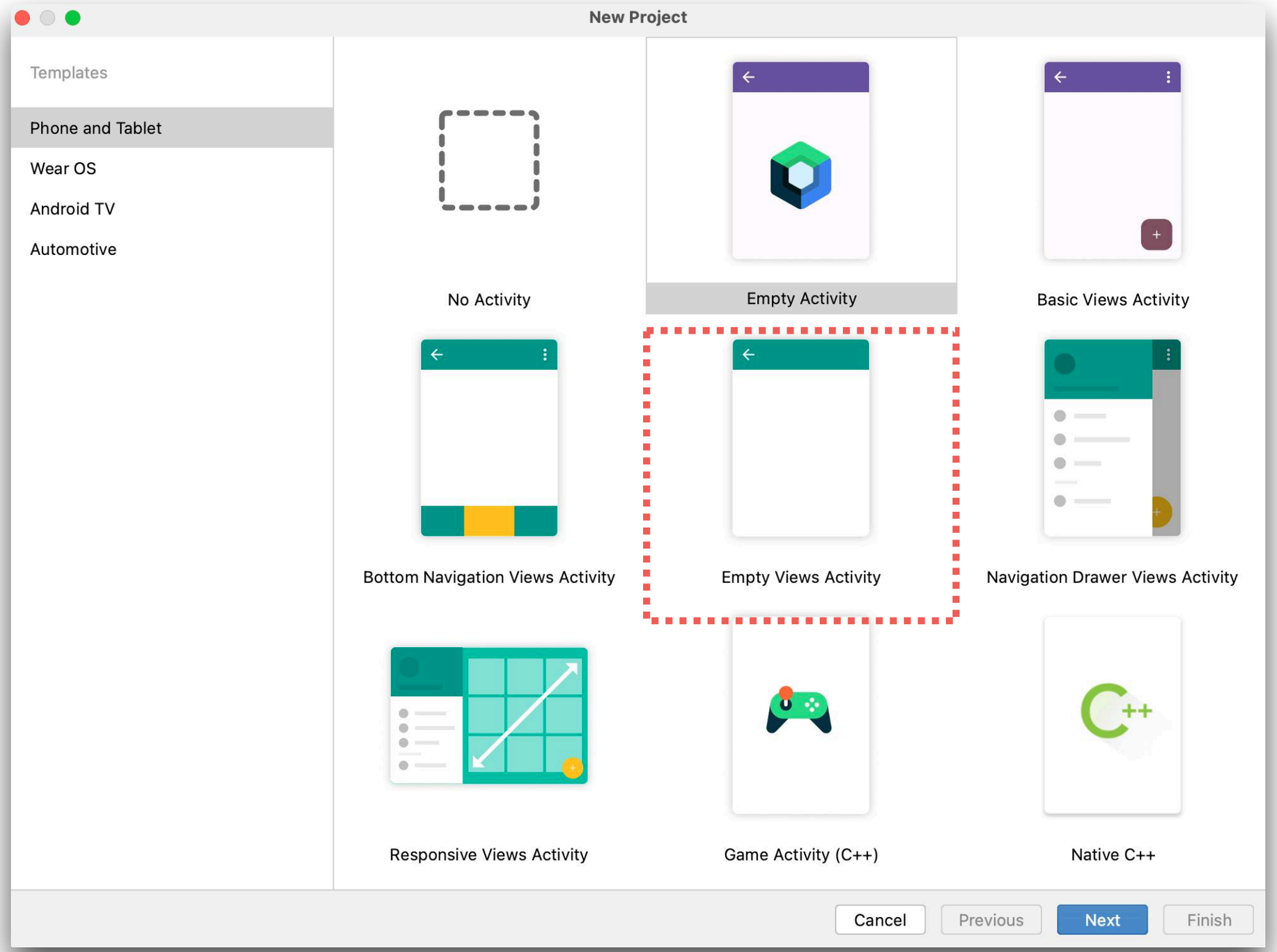
Cheat sheet



https://developer.android.com/training/testing/espresso/cheat-sheet

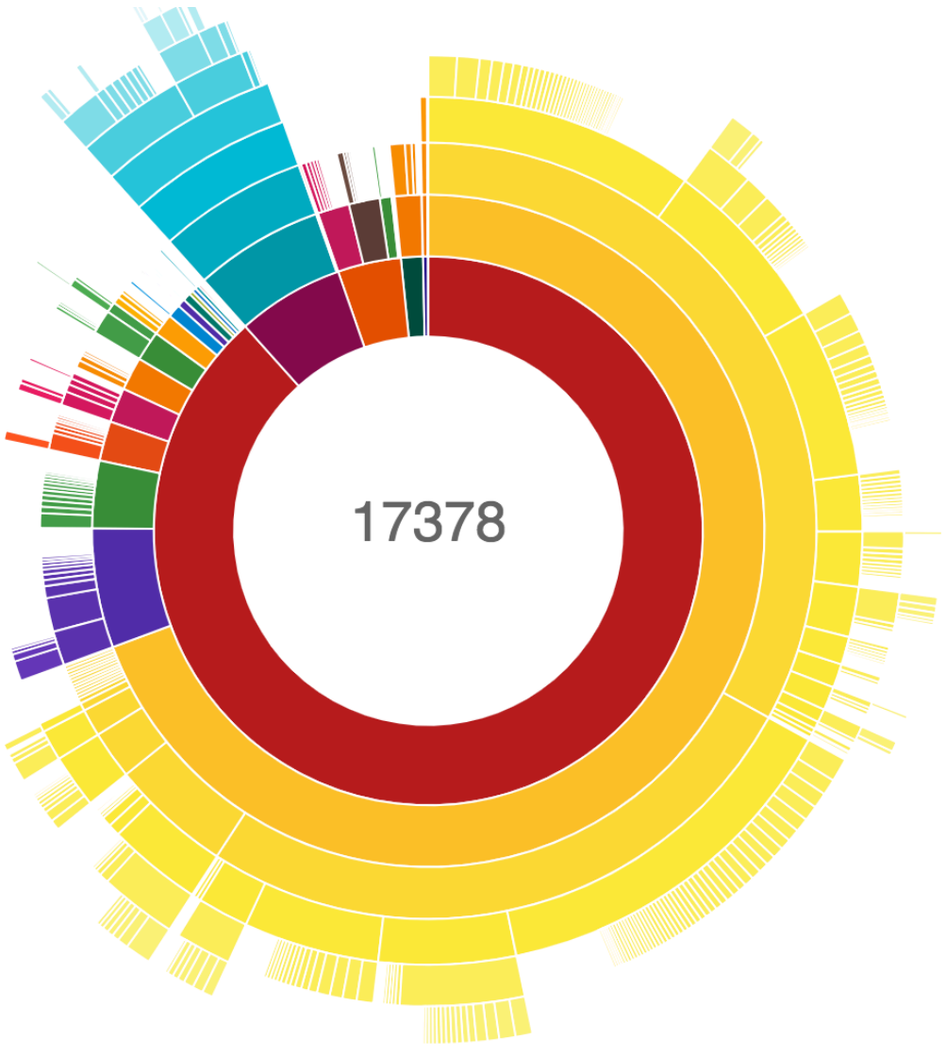
Let’ start to write tests

Create a new project



Count number of methods (65K)

Using dexcount plugin



https://github.com/KeepSafe/dexcount-gradle-plugin

Android Testing Workshop

Good Unit Test

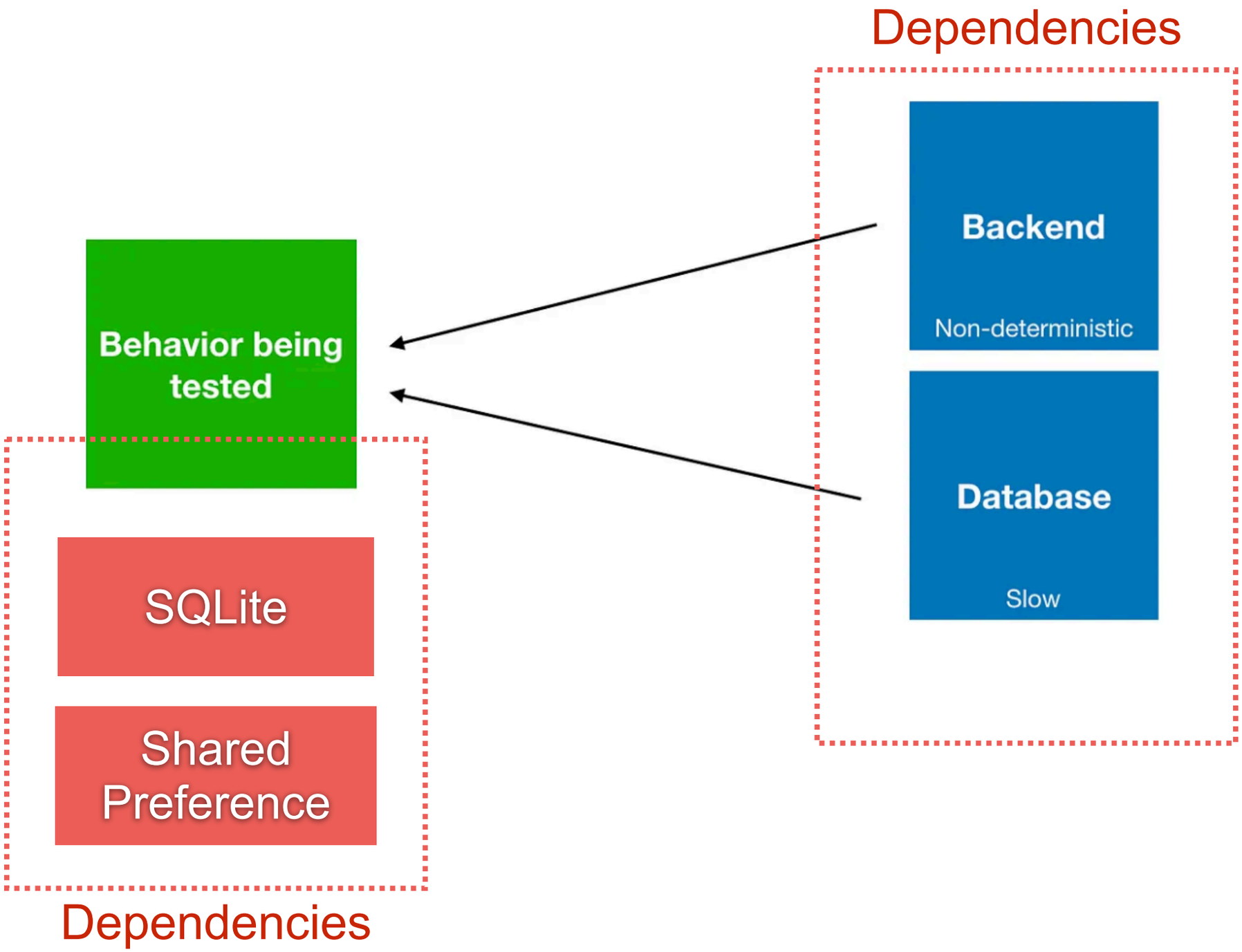
F.I.R.S.T

Fast
Isolate/Indepondece
Repeatable
Self-verify
Timely

Basic of Good tests

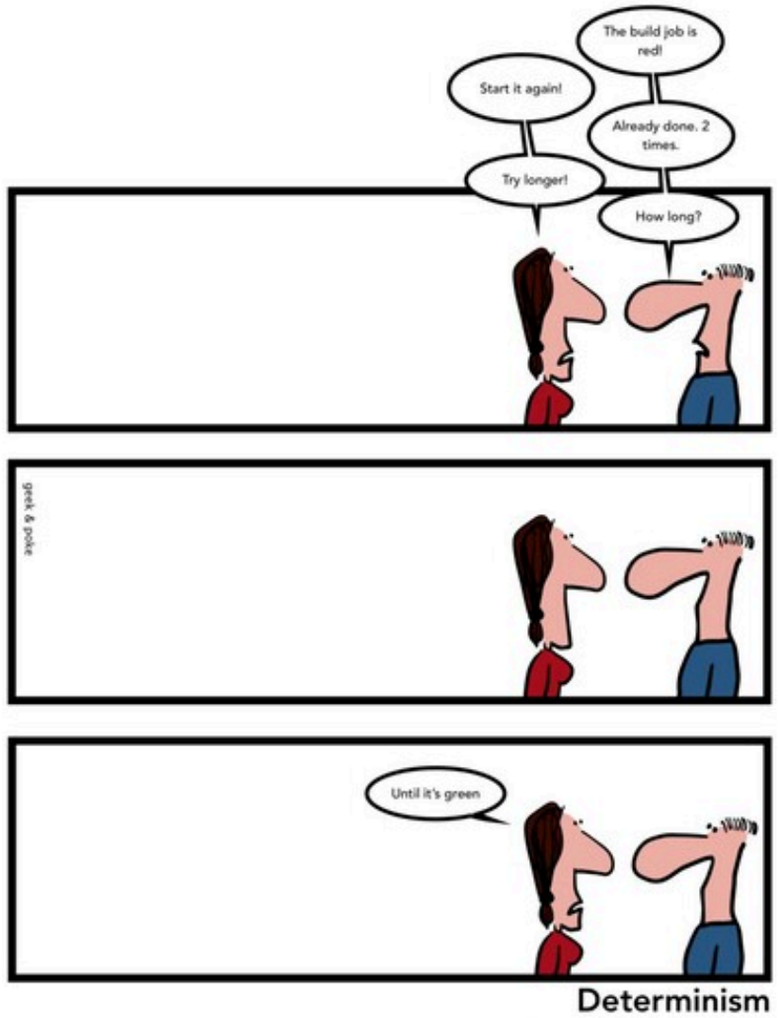
Speed (fast is better)
Determinism
Easy configuration

Back to App !!



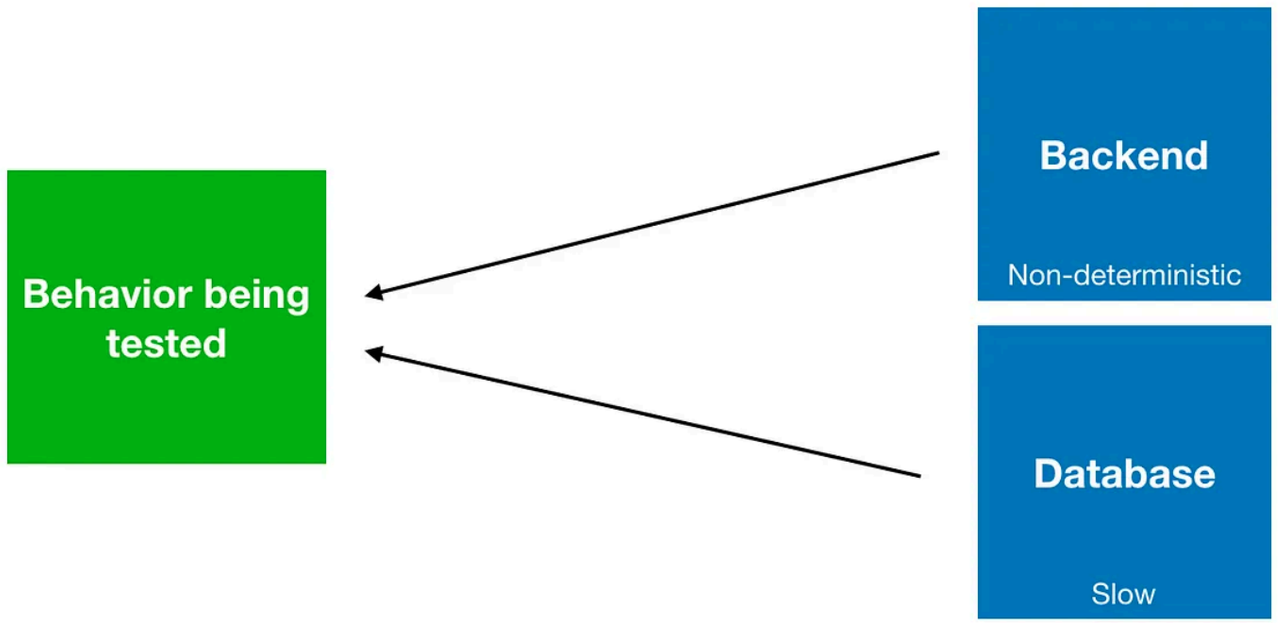
Flaky testing !!

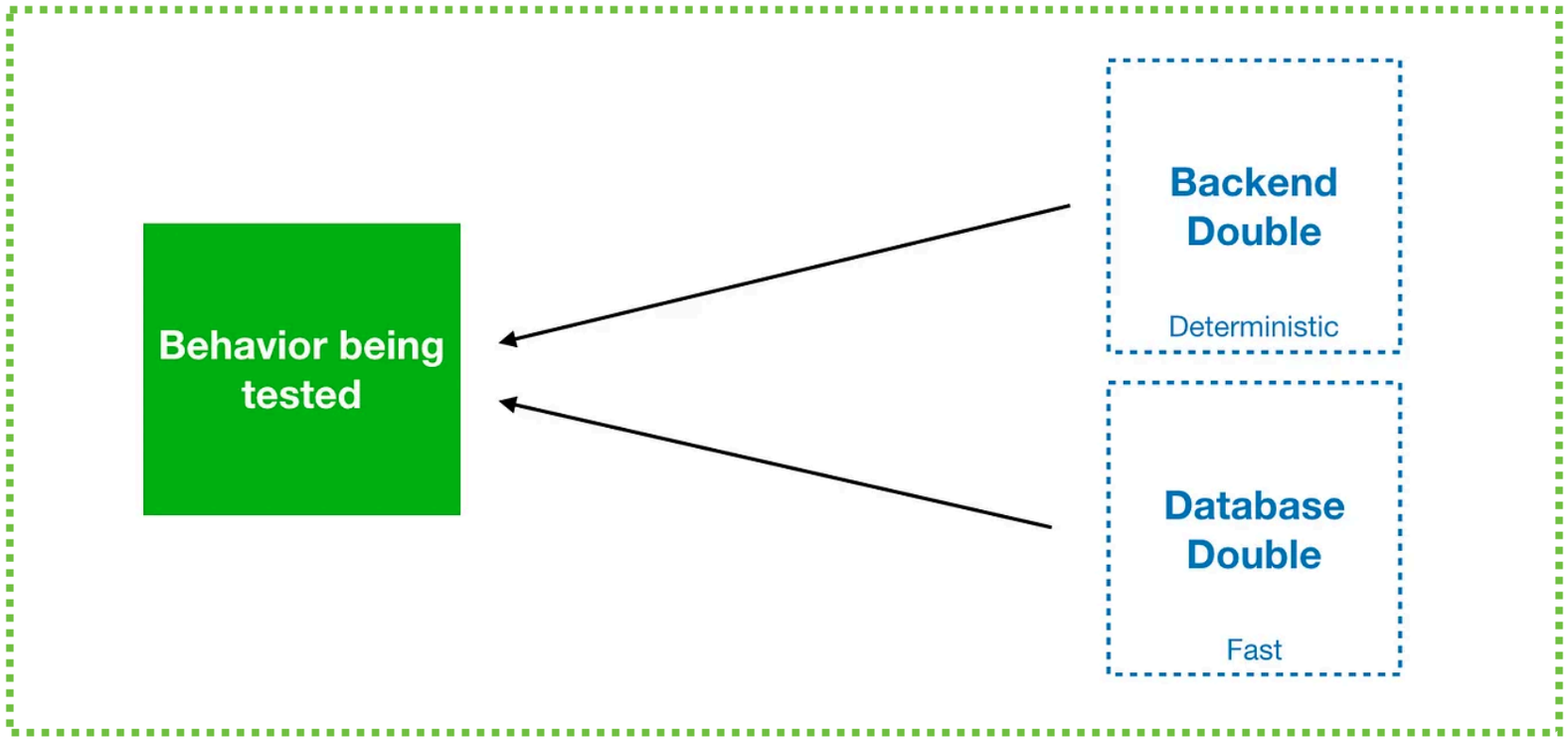
Issue: Flaky test cases makes build job execution unstable



By geek & poke

Need to improve

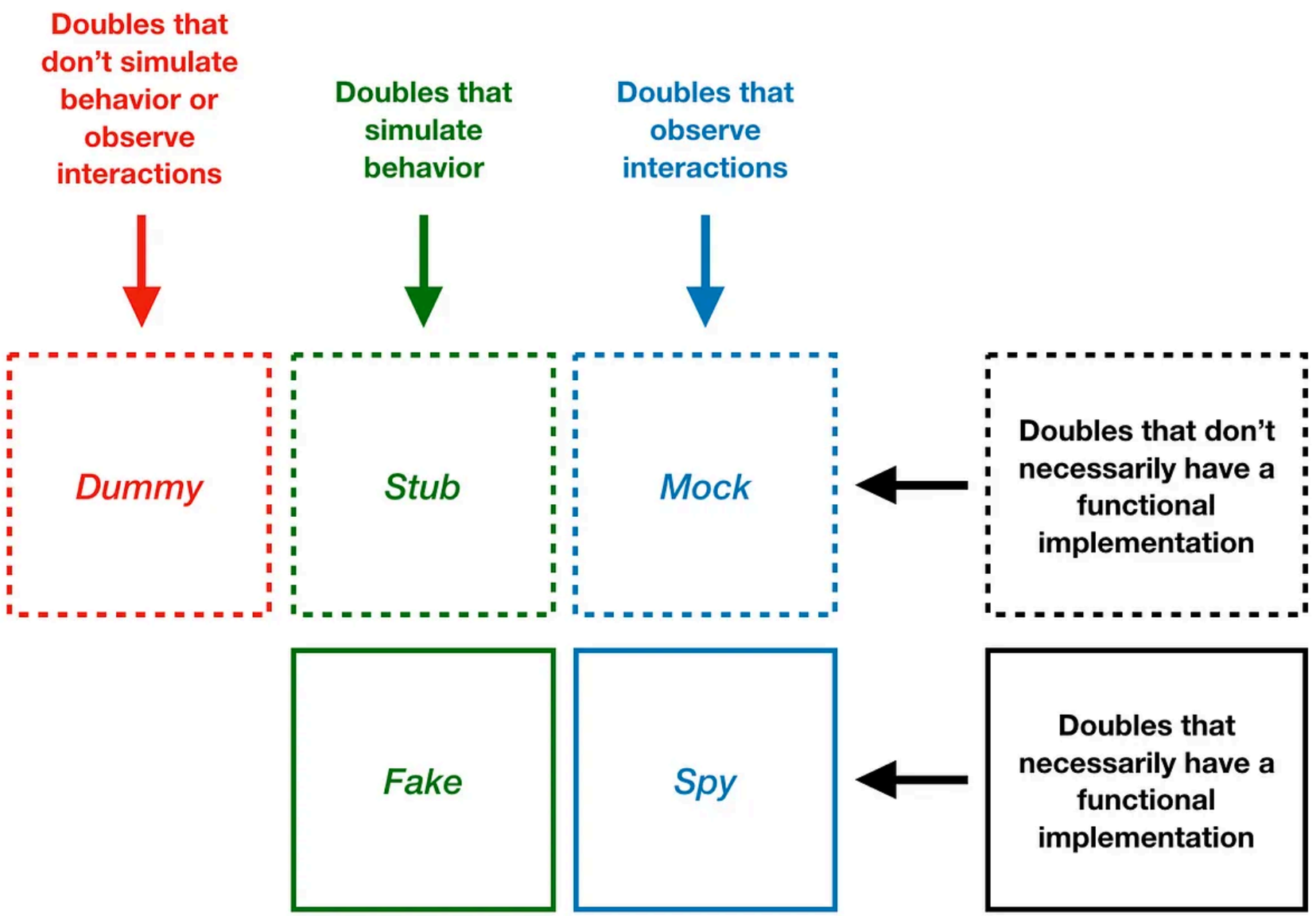




Test Doubles in Android

Dummy
Stub
Fake
Spy
Mock
Shadow

Test Doubles in Android

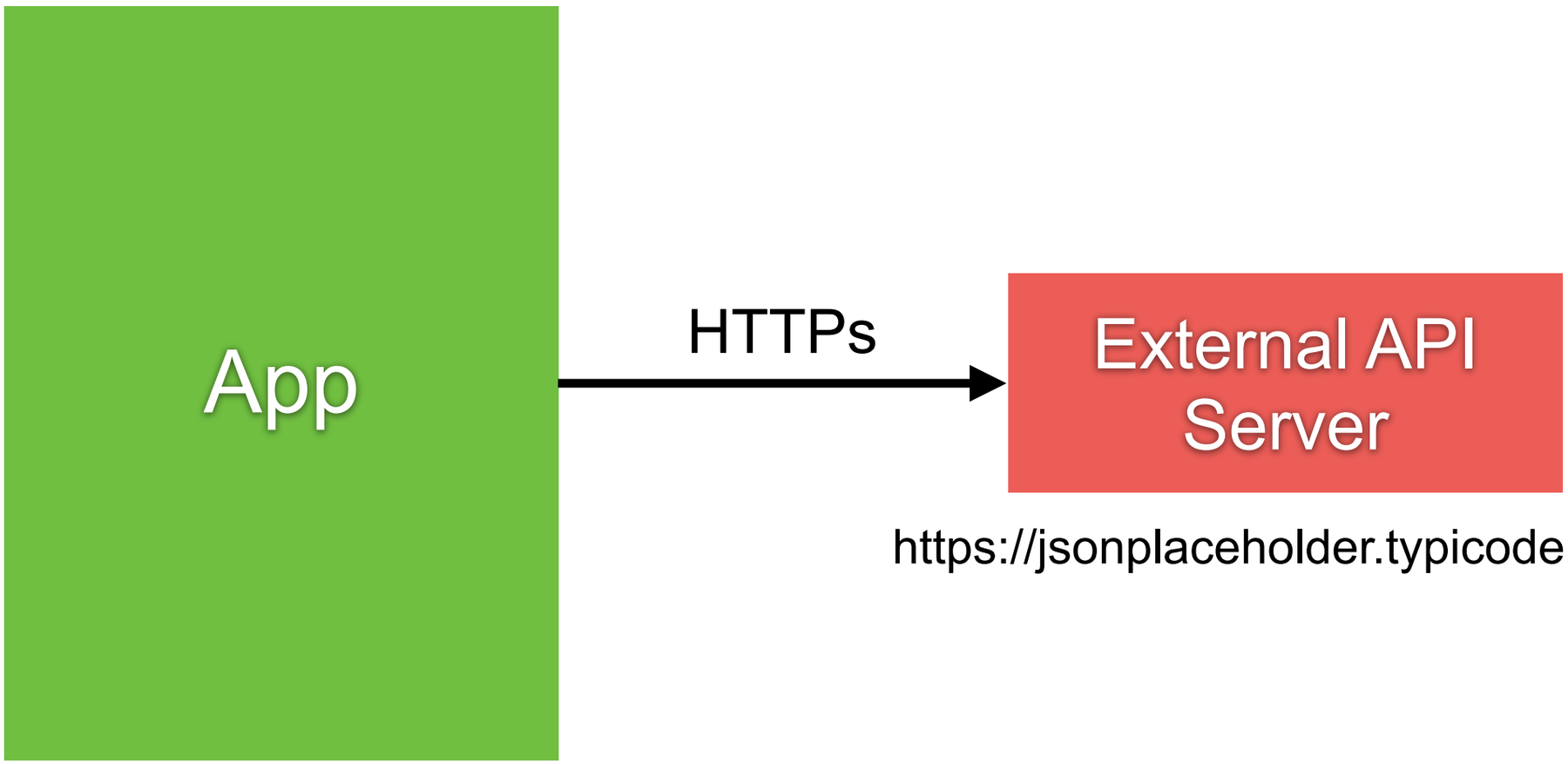


Test Doubles in Android

MockWeb Server

Workshop with Test Double

Demo App for Testing



https://jsonplaceholder.typicode.com/

https://github.com/up1/course-effective-android-testing-2023

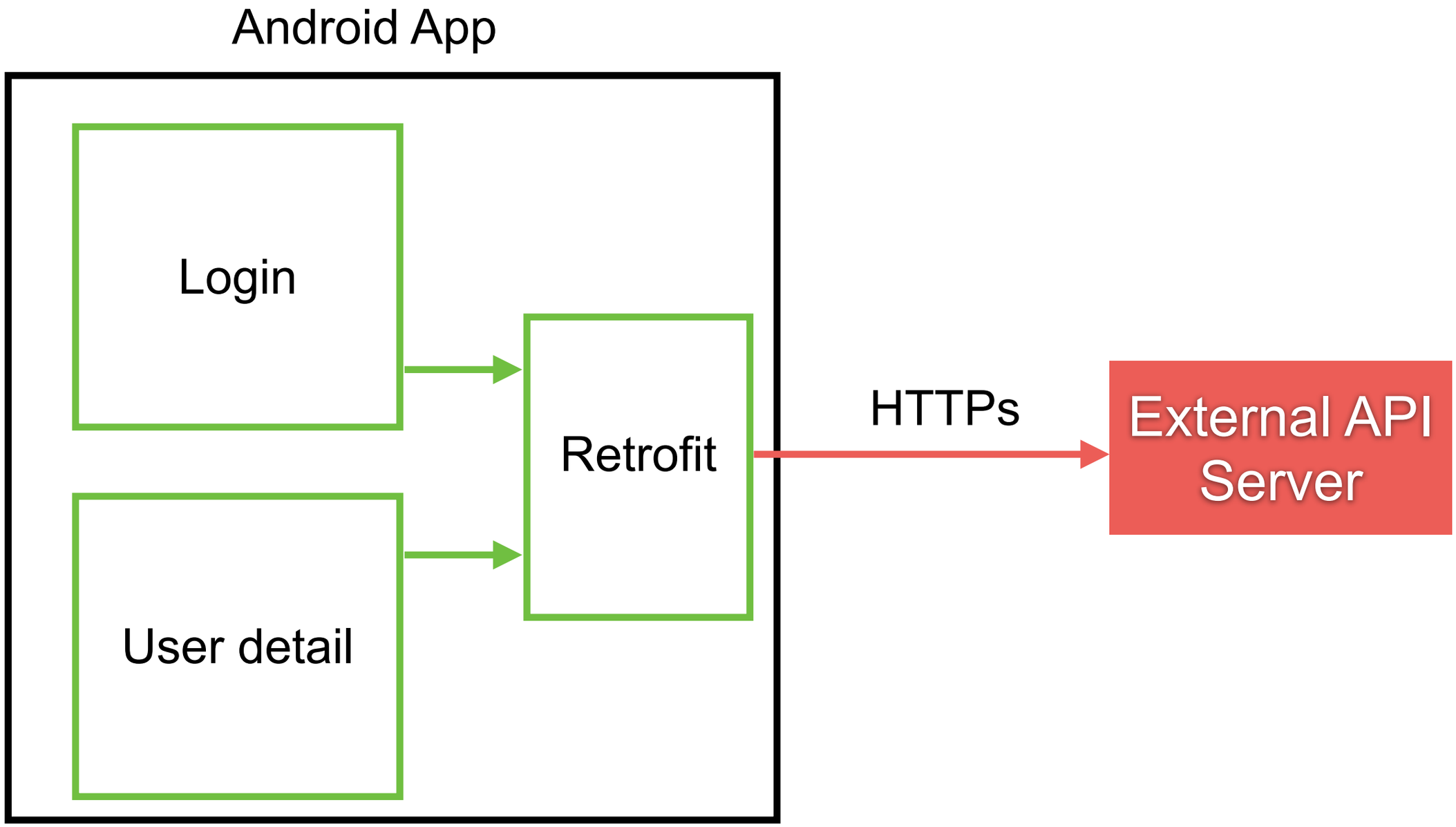
JSON Placeholder

Use to login and get user detail

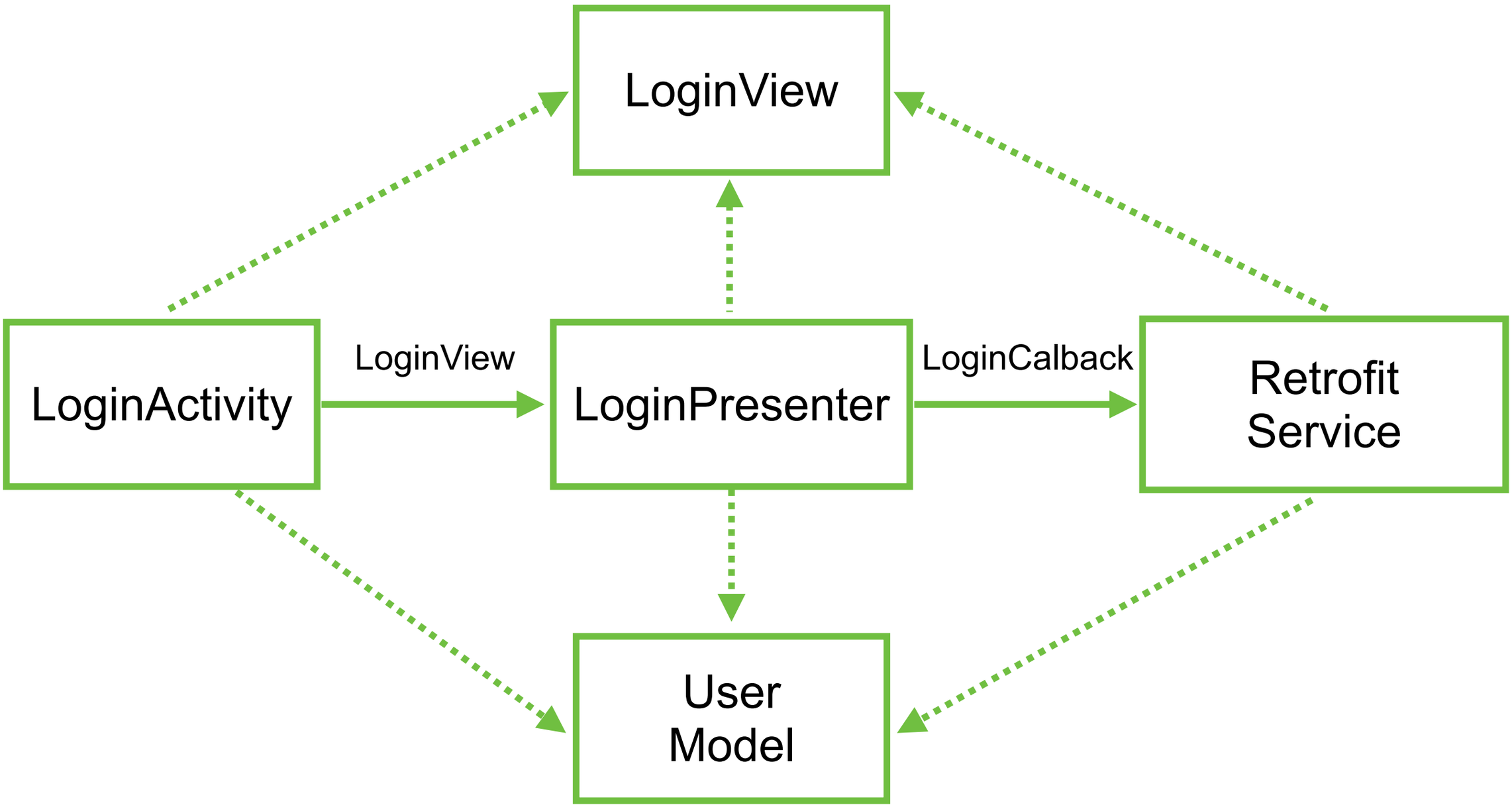
Endpoint	Description
GET /users?email=?&password=	Login with email and password email=Sincere@april.biz password=CorrectPassword123
GET /user/:id	Get user detail by id

https://jsonplaceholder.typicode.com/

Sample App



MVP (Model-View-Presenter)



Goals of Workshop

Unit testing with Unit 4
Working with MVP architecture
Integration test with Robolectric
UI test with Espresso
End-to-End test with UIAutomator

Design your test cases ?

Test Case Name	Email	Password	Expected Result
Login success	Sincere@april.biz	CorrectPassword123	Show hello username
Login fail ?			Show fail message
Login fail ?			Show fail message
Login fail ?			Show fail message
Login fail ?			Show fail message
Login fail ?			Show fail message

UI test with Espresso

https://developer.android.com/training/testing/espresso

What to test ?

https://developer.android.com/training/testing/fundamentals/what-to-test

UI test in Android app

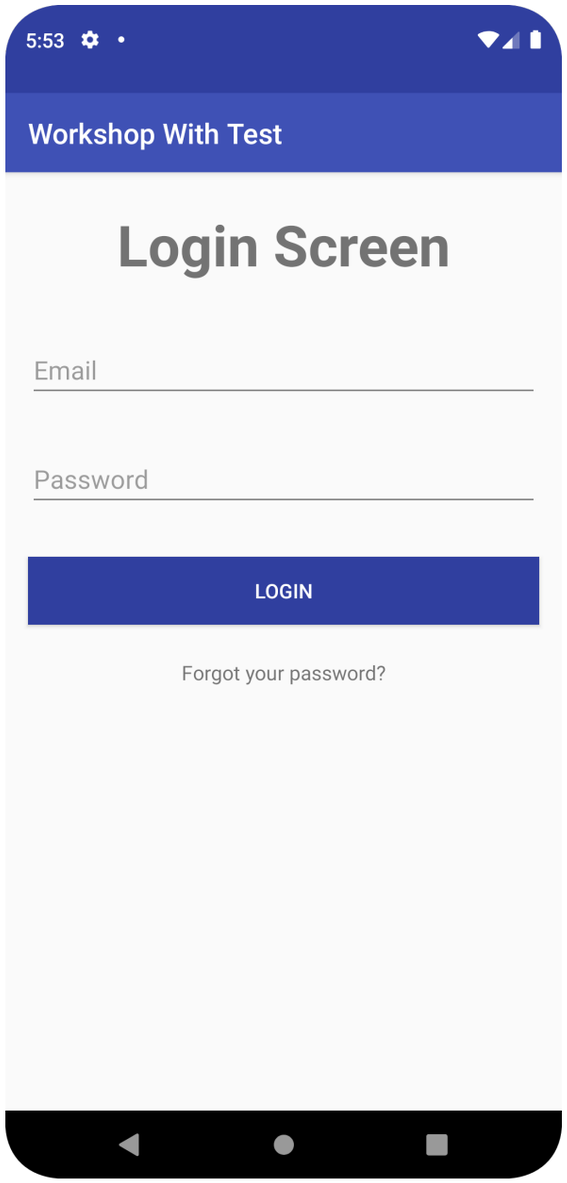
Screen UI tests (single screen)
User flow tests or Navigation tests

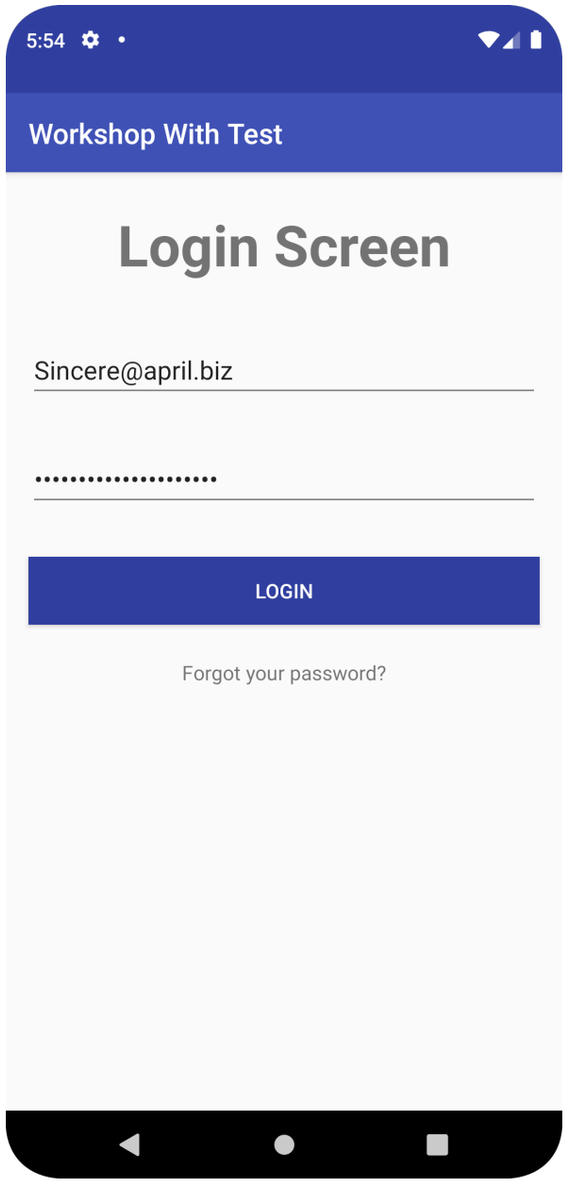
Add dependencies

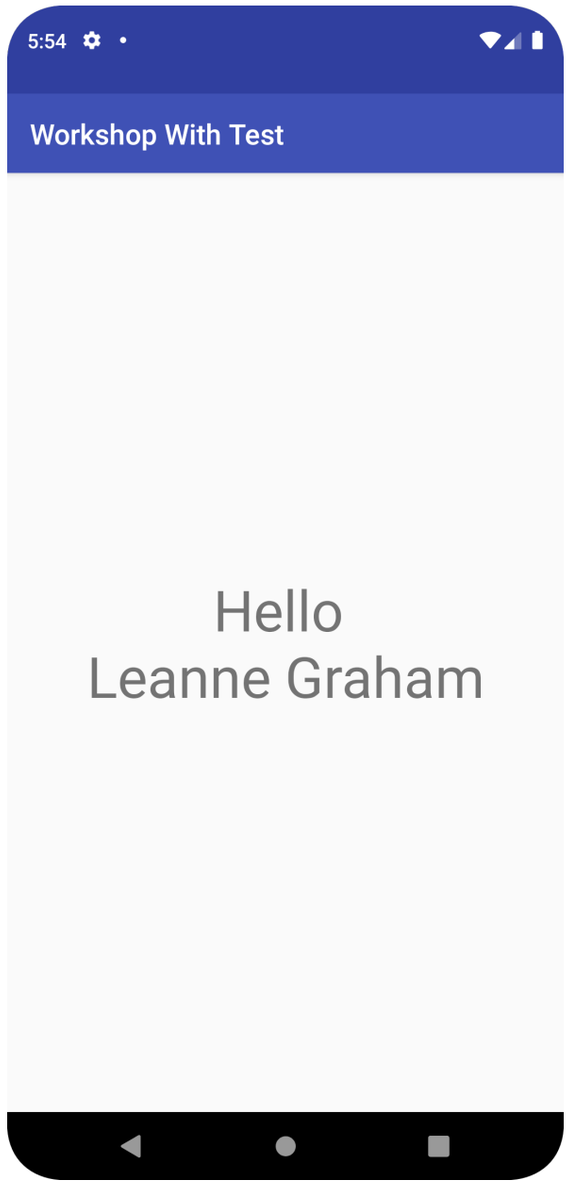
Update file **app/build.gradle**

```
dependencies {
    // Instrumentation test and espresso
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation 'androidx.test:runner:1.5.2'
    androidTestImplementation 'androidx.test:rules:1.5.0'
}
```

Success case

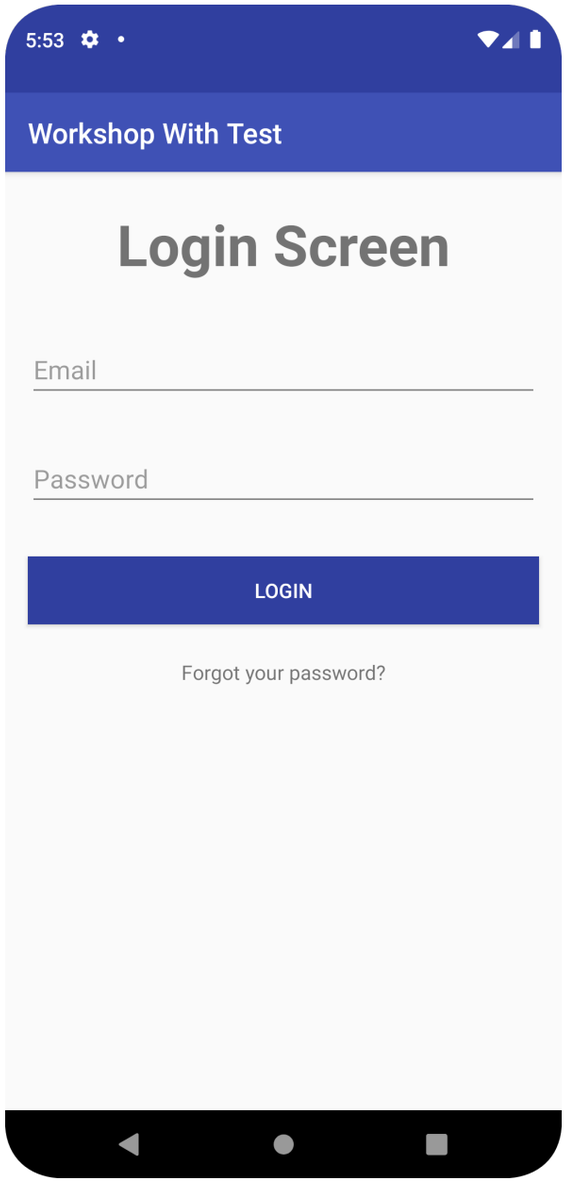


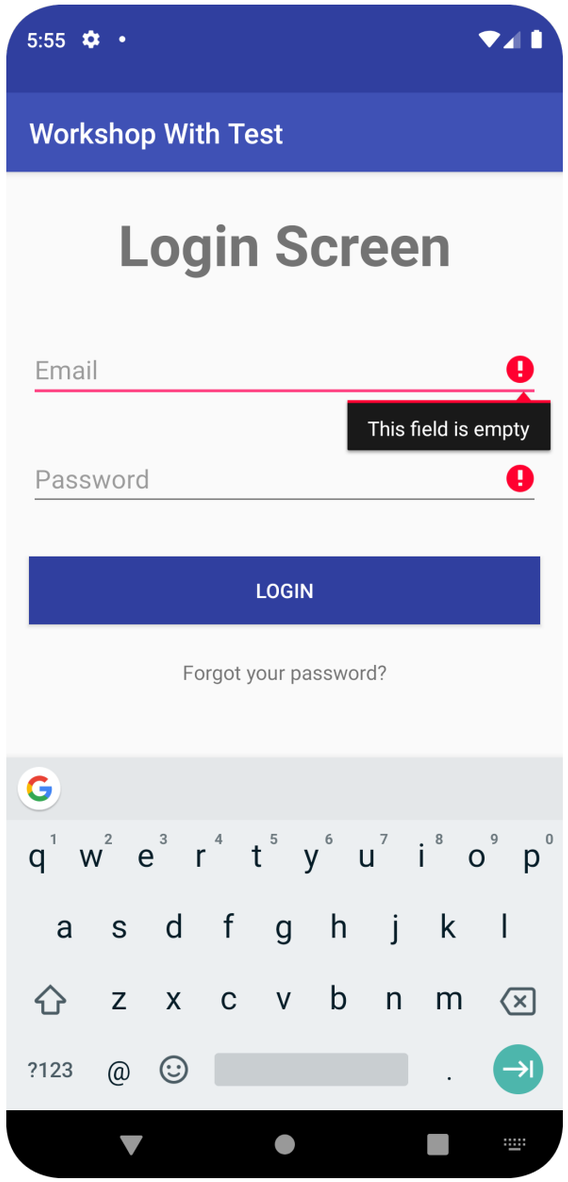




Failure cases

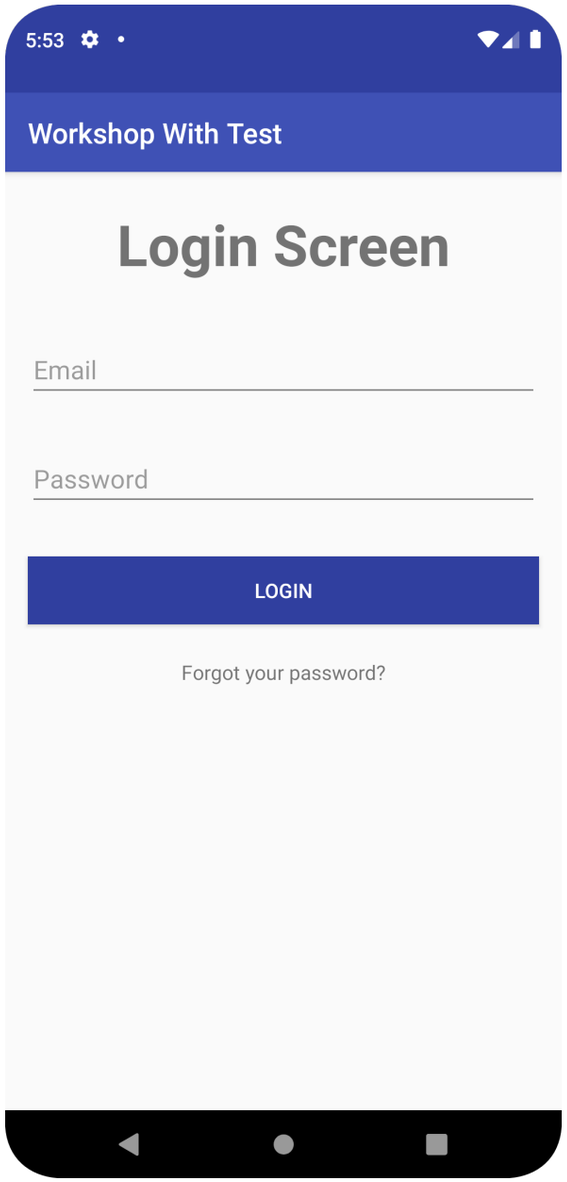
Empty email

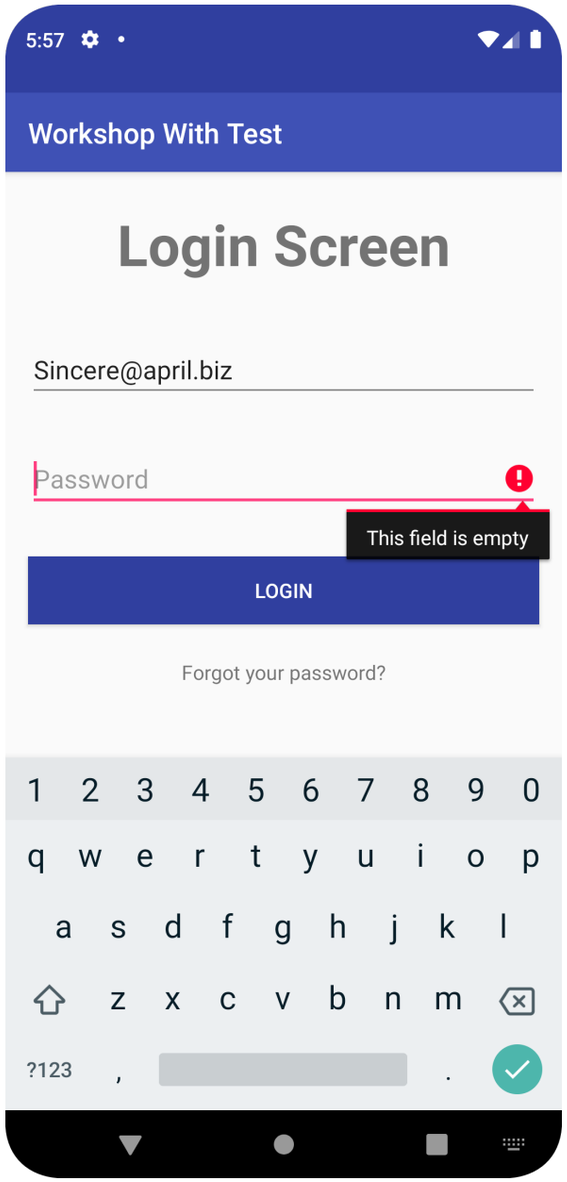




Failure cases

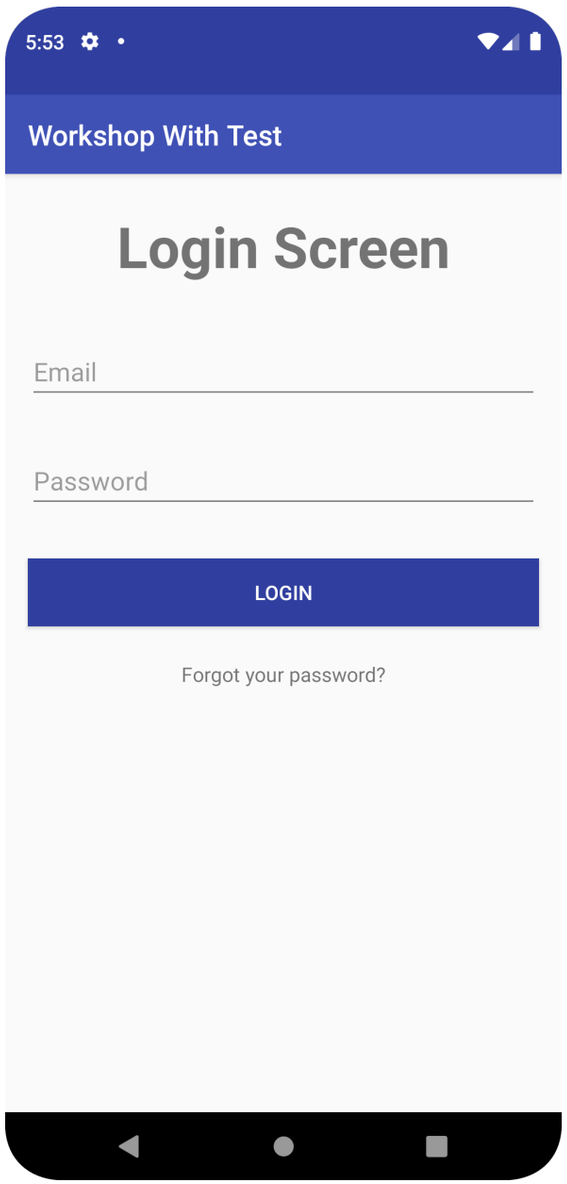
Empty password

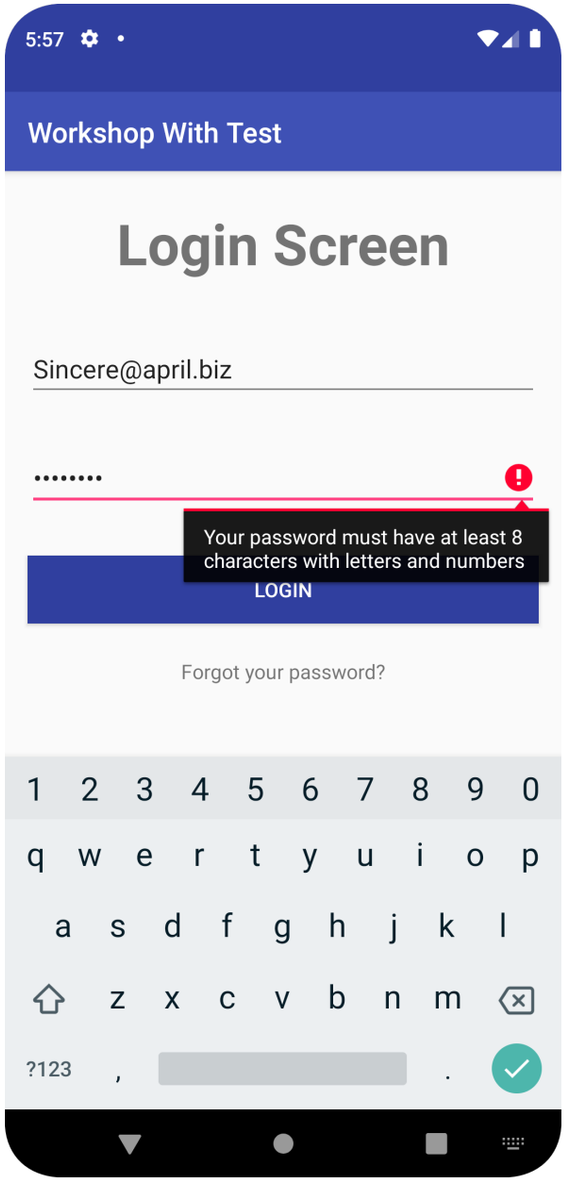




Failure cases

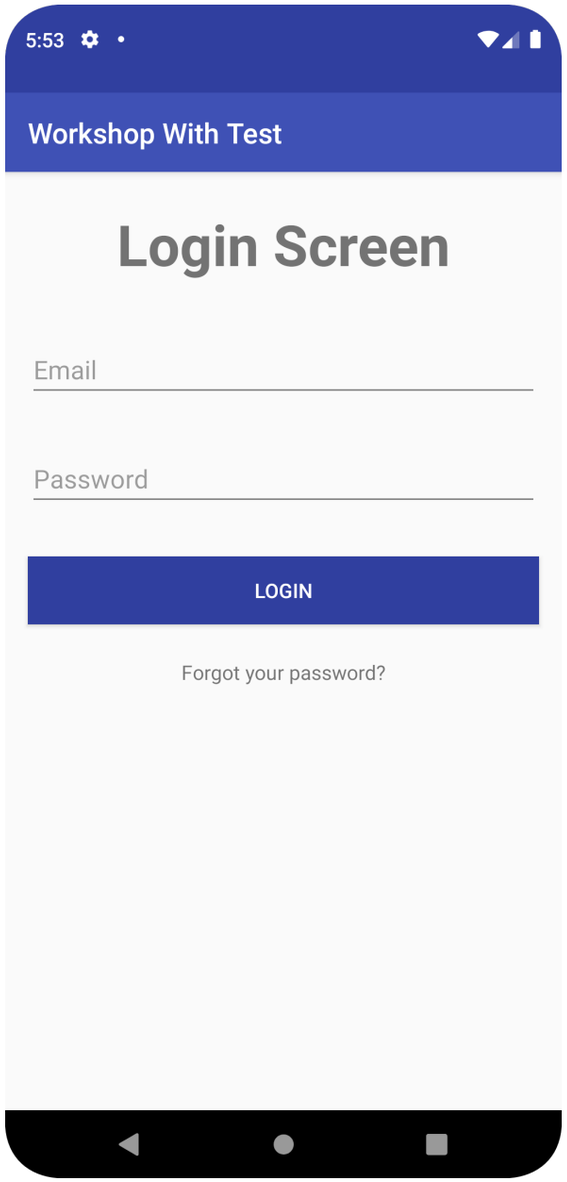
Invalid password

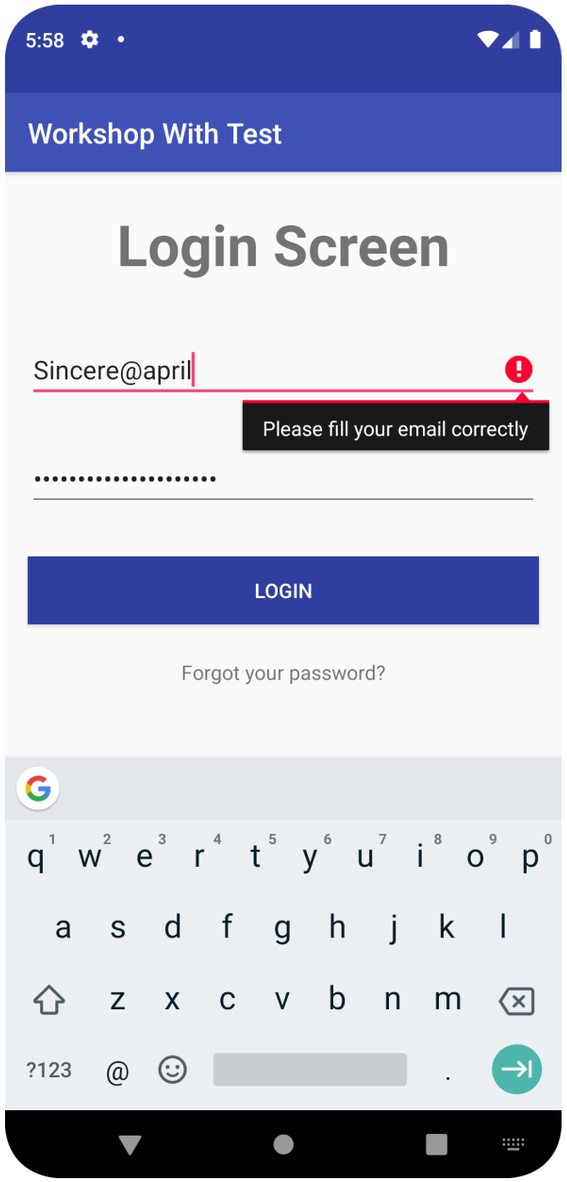




Failure cases

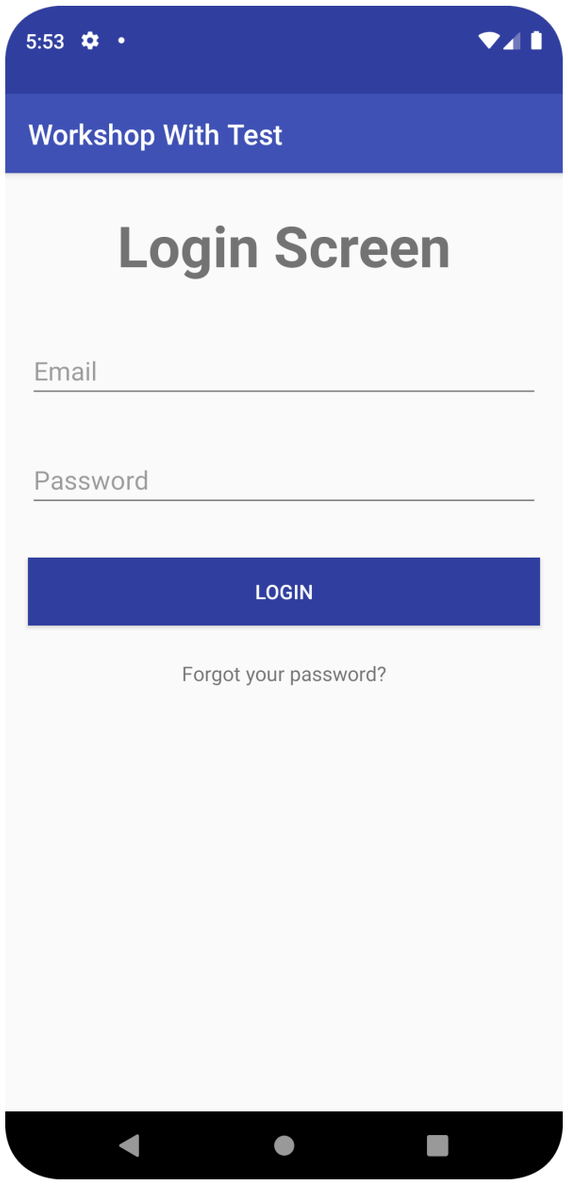
Invalid email format





Write first case

Verify login page when loaded



Write first case

```
@RunWith(AndroidJUnit4::class)
class LoginActivityEspressoTest {

    @get:Rule
    val activityRule = ActivityScenarioRule(LoginActivity::class.java)

    @Test
    fun shouldShowTitleWhenOpenActivity() {
        onView(withText("Login Screen")).check(matches(isDisplayed()))
    }

    @Test
    fun shouldShowLoginButtonWhenOpenActivity() {
        onView(withId(R.id.btnLogin)).check(matches(withText("Login")))
    }

}
```

Write first case

```
@RunWith(AndroidJUnit4::class)
class LoginActivityEspressoTest {
```

Run with Junit 4

```
    @get:Rule
    val activityRule = ActivityScenarioRule(LoginActivity::class.java)

    @Test
    fun shouldShowTitleWhenOpenActivity() {
        onView(withText("Login Screen")).check(matches(isDisplayed()))
    }

    @Test
    fun shouldShowLoginButtonWhenOpenActivity() {
        onView(withId(R.id.btnLogin)).check(matches(withText("Login")))
    }

}
```

Write first case

```
@RunWith(AndroidJUnit4::class)
class LoginActivityEspressoTest {

    @get:Rule
    val activityRule = ActivityScenarioRule(LoginActivity::class.java)

    @Test
    fun shouldShowTitleWhenOpenActivity() {
        onView(withText("Login Screen")).check(matches(isDisplayed()))
    }

    @Test
    fun shouldShowLoginButtonWhenOpenActivity() {
        onView(withId(R.id.btnLogin)).check(matches(withText("Login")))
    }

}
```

Load and launch activity

Write first case

```
@RunWith(AndroidJUnit4::class)
class LoginActivityEspressoTest {

    @get:Rule
    val activityRule = ActivityScenarioRule(LoginActivity::class.java)

    @Test
    fun shouldShowTitleWhenOpenActivity() {
        onView(withText("Login Screen")).check(matches(isDisplayed()))
    }

    @Test
    fun shouldShowLoginButtonWhenOpenActivity() {
        onView(withId(R.id.btnLogin)).check(matches(withText("Login")))
    }

}
```

Verify elements in login activity after loaded

Run test

$gradlew connectedAndroidTest

Try to open test report in

app/build/reports/androidTests/connected/flavors/mocked/index.html

https://developer.android.com/studio/test/command-line

Test report in HTML format

Test Summary

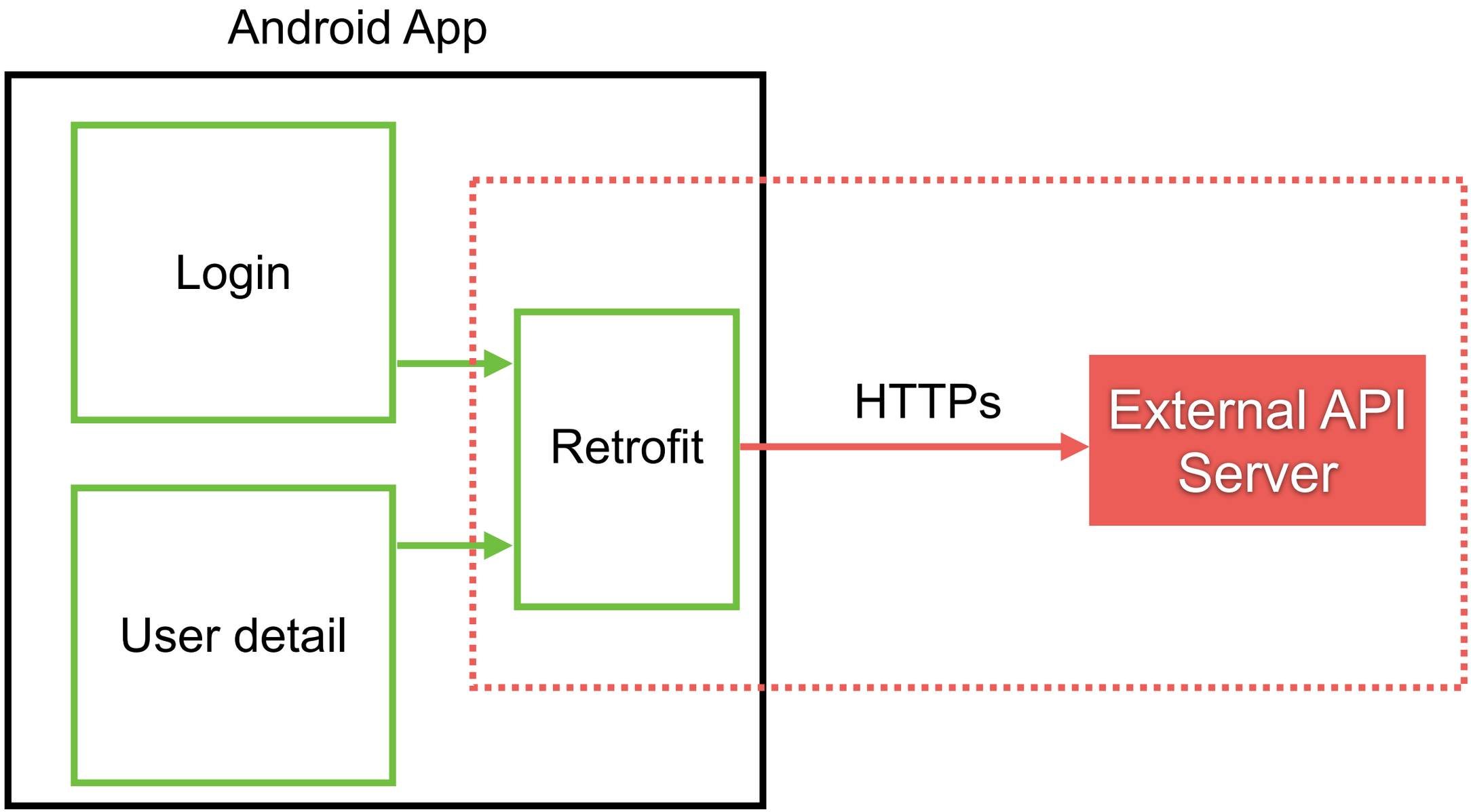
3	0	0	0.408s
tests	failures	skipped	duration

100%
successful

Packages Classes

Package	Tests	Failures	Skipped	Duration	Success rate
com.example.demowithtest	1	0	0	0.008s	100%
com.example.demowithtest.login	2	0	0	0.400s	100%

Improve reliable of testing



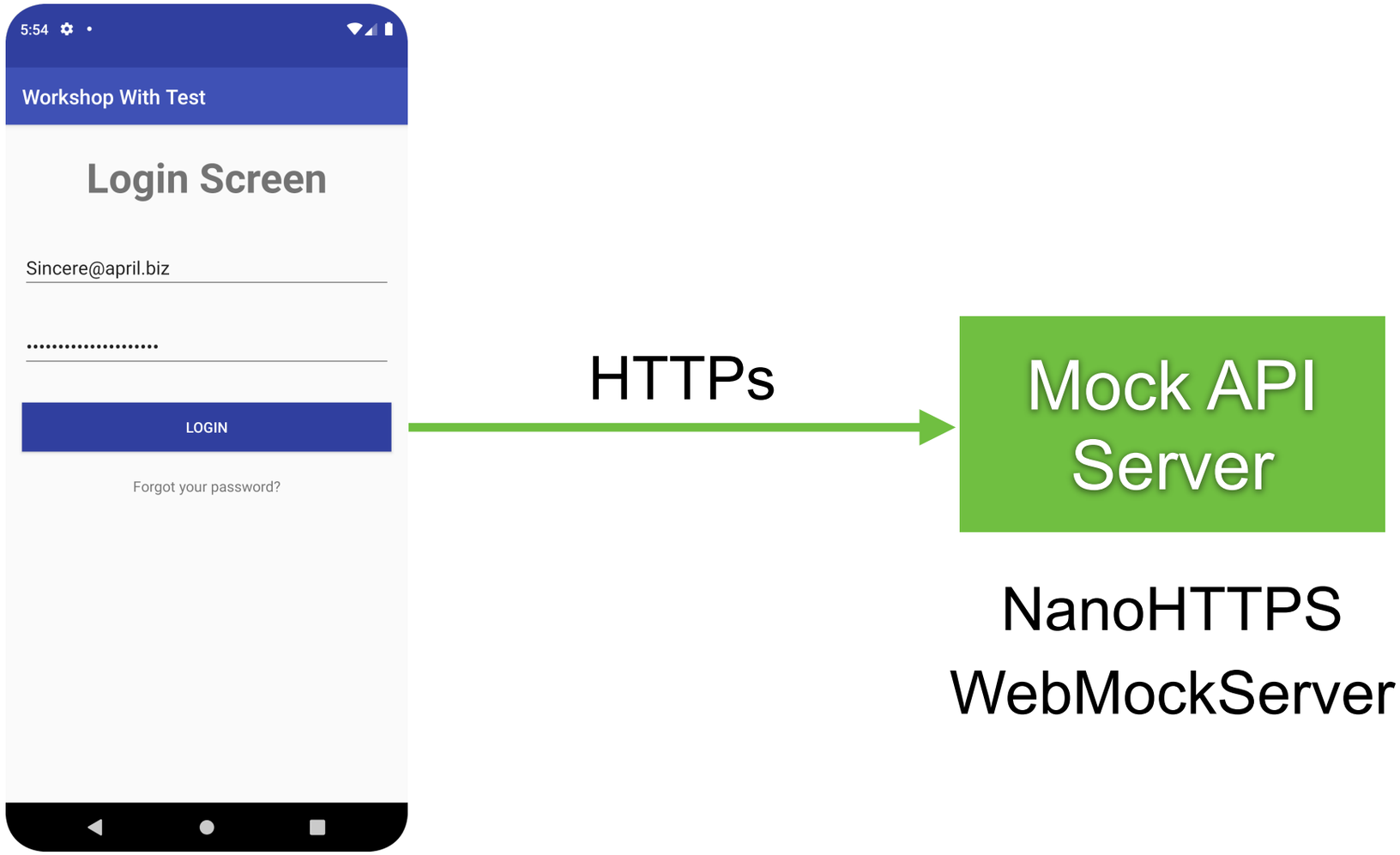
Fake

Like a real dependencies
Not class/interface
Test reliable
Don’t require mocking framework

In-memory database

External API Server

Working with Mock API



https://github.com/NanoHttpd/nanohttpd

https://github.com/square/okhttp/tree/master/mockwebserver

Working with NonoHTTPD

Add dependency in app/build.gradle

```
dependencies {

    implementation 'com.nanohttpd:nanohttpd:2.1.1'

}
```

https://github.com/NanoHttpd/nanohttpd

Create MockedServer

Create new class in /androidTest folder

```
class MockedServer : NanoHTTPD(PORT) {

    companion object {
        private const val PORT = 4444
    }

    private var response: String? = null

    override fun serve(session: IHTTPSession): Response {
        return Response(response)
    }

    fun setResponse(response: String) {
        this.response = response
    }
}
```

https://github.com/NanoHttpd/nanohttpd

Create Dummy Response

Create new object in /androidTest folder

```
object ReadDummyServerResponse {

    fun readJsonFile(fileName: String): String {
        val context: Context =
            InstrumentationRegistry.getInstrumentation().getContext();
        val assetManager: AssetManager = context.getAssets()
        val jsonFixtureFile = assetManager.open("files/$fileName")
        val s = Scanner(jsonFixtureFile).useDelimiter("\\A")
        return if (s.hasNext()) s.next() else ""
    }
}
```

https://github.com/NanoHttpd/nanohttpd

Steps in test case

Create and start a Mock Server

Setup response data in test case

Run test case

Stop a Mock Server

Write test case

Start a mock server with dummy response

```
@Test
fun shouldShowUserDetailsWhenLoginCompletesSuccessfully() {
    val mockedServer = MockedServer()
    mockedServer.start()
    mockedServer.setResponse(
            ReadDummyServerResponse.readJsonFile("loginresponse.json"))

    onView(withId(R.id.edtEmail)).perform(typeText("Sincere@april.biz"))
    onView(withId(R.id.edtPassword)).perform(typeText("CorrectPassword123456"))

    onView(withId(R.id.scrollView)).perform(swipeUp())
    onView(withId(R.id.btnLogin)).perform(click())

    onView(withId(R.id.txtWelcomeMsg))
            .check(matches(withText("Hello \nName from mock server")))

    mockedServer.stop()

}
```

Write test case

Create steps in test case

```
@Test
fun shouldShowUserDetailsWhenLoginCompletesSuccessfully() {

    val mockedServer = MockedServer()
    mockedServer.start()
    mockedServer.setResponse(
            ReadDummyServerResponse.readJsonFile("loginresponse.json"))

    onView(withId(R.id.edtEmail)).perform(typeText("Sincere@april.biz"))
    onView(withId(R.id.edtPassword)).perform(typeText("CorrectPassword123456"))

    onView(withId(R.id.scrollView)).perform(swipeUp())
    onView(withId(R.id.btnLogin)).perform(click())

    onView(withId(R.id.txtWelcomeMsg))
          .check(matches(withText(“Hello \nName from mock server")))

    mockedServer.stop()

}
```

Write test case

Stop a mock server

```
@Test
fun shouldShowUserDetailsWhenLoginCompletesSuccessfully() {

    val mockedServer = MockedServer()
    mockedServer.start()
    mockedServer.setResponse(
            ReadDummyServerResponse.readJsonFile("loginresponse.json"))

    onView(withId(R.id.edtEmail)).perform(typeText("Sincere@april.biz"))
    onView(withId(R.id.edtPassword)).perform(typeText("CorrectPassword123456"))

    onView(withId(R.id.scrollView)).perform(swipeUp())
    onView(withId(R.id.btnLogin)).perform(click())

    onView(withId(R.id.txtWelcomeMsg))
            .check(matches(withText("Hello \nName from mock server")))

    mockedServer.stop()
}
```

Config with BuildConfig

Edit in file app/build.gradle

```
flavorDimensions "dimension"
productFlavors {
    mocked {
        buildConfigField 'String', 'login_url', '"http://localhost:4444"'
    }
    prod {
        buildConfigField 'String', 'login_url', '"https://
jsonplaceholder.typicode.com"'
    }
}
```

Code coverage

"Code coverage can show the high risk areas in a program, but never the risk-free."

Paul Reilly, 2018, Kotlin TDD with Code Coverage

Code coverage

A tool to measure how much of your code is covered by tests that break down into classes, methods and lines.

Code coverage

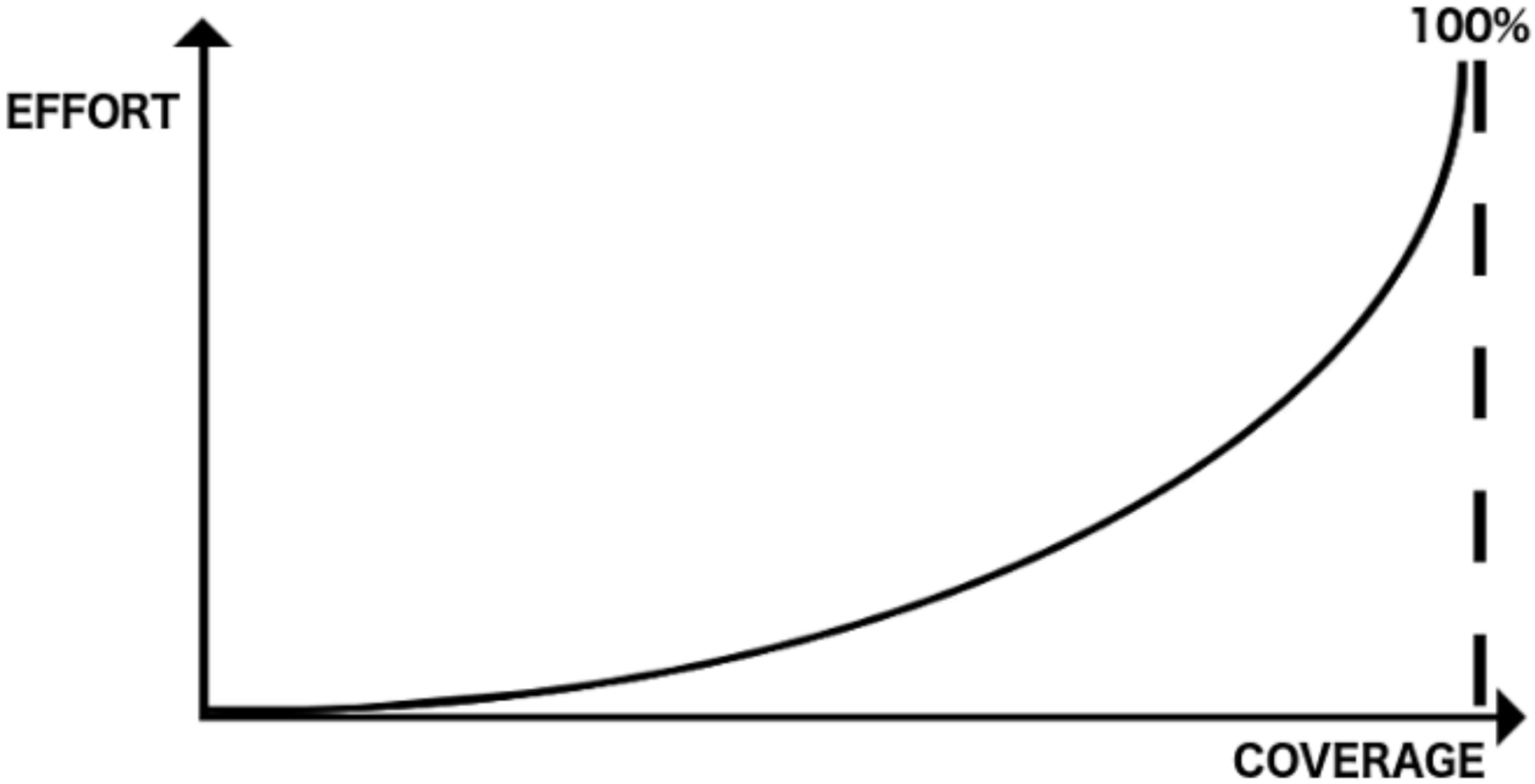
But 100% of code coverage **does not mean** that your code is 100% correct

Code coverage

Powerful tool to improve the quality of your code

Code coverage != quality of tests

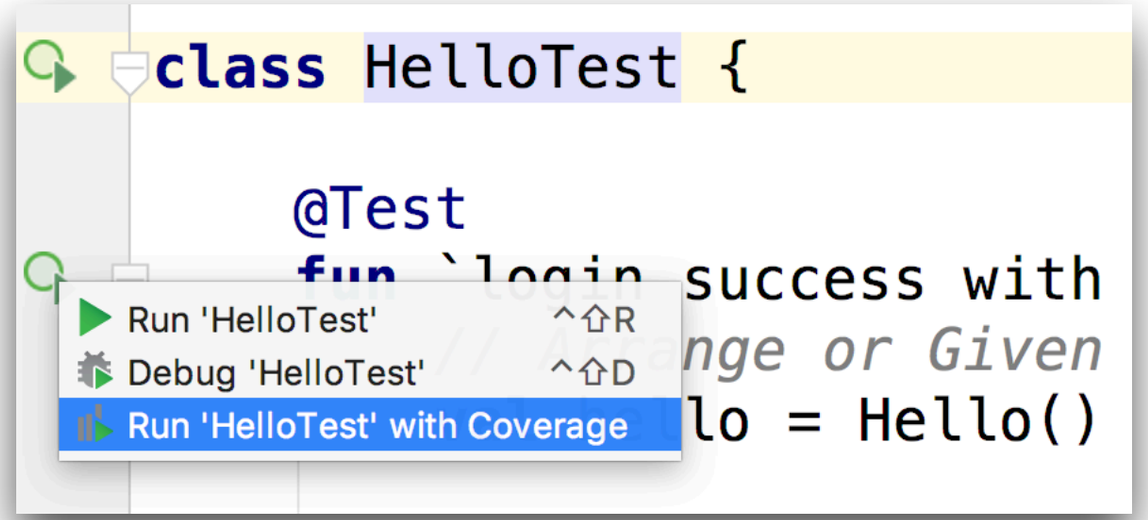
Code coverage 100% ?



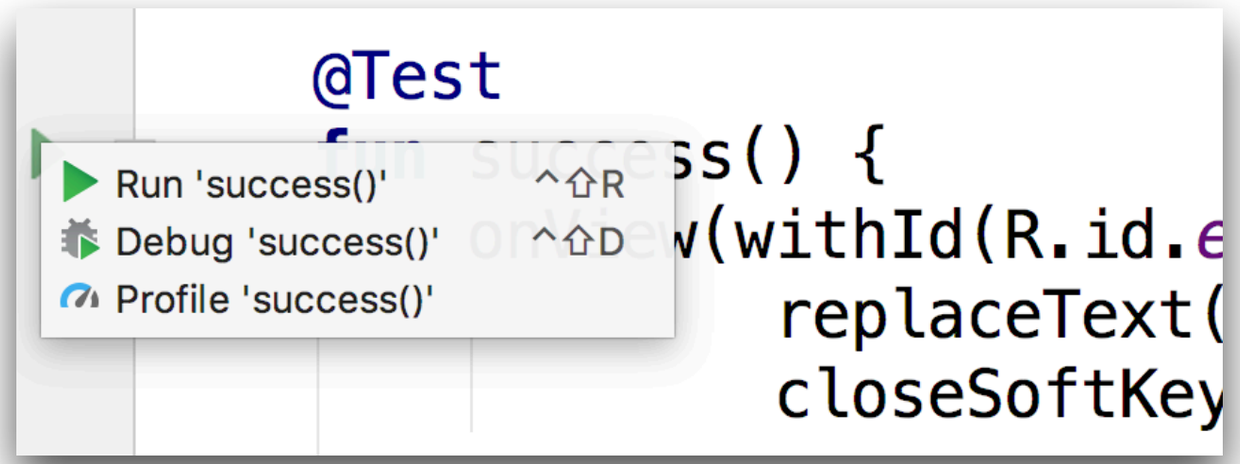
Code coverage for android

Disabled by default !!

Unit test



androidTest



Enable Code Coverage

1. Enable coverage in debug

In file /app/build.gradle

```
buildTypes {
    debug {
        testCoverageEnabled true
    }
    release {
        minifyEnabled false
        proguardFiles getDefaultProg
    }
}
```

2. Use jacoco library

Create a new file **/app/jacoco.gradle**

https://github.com/up1/course-effective-android-testing-2023/blob/testing/demo/demo-with-test/app/jacoco.gradle

3. Use jacoco library

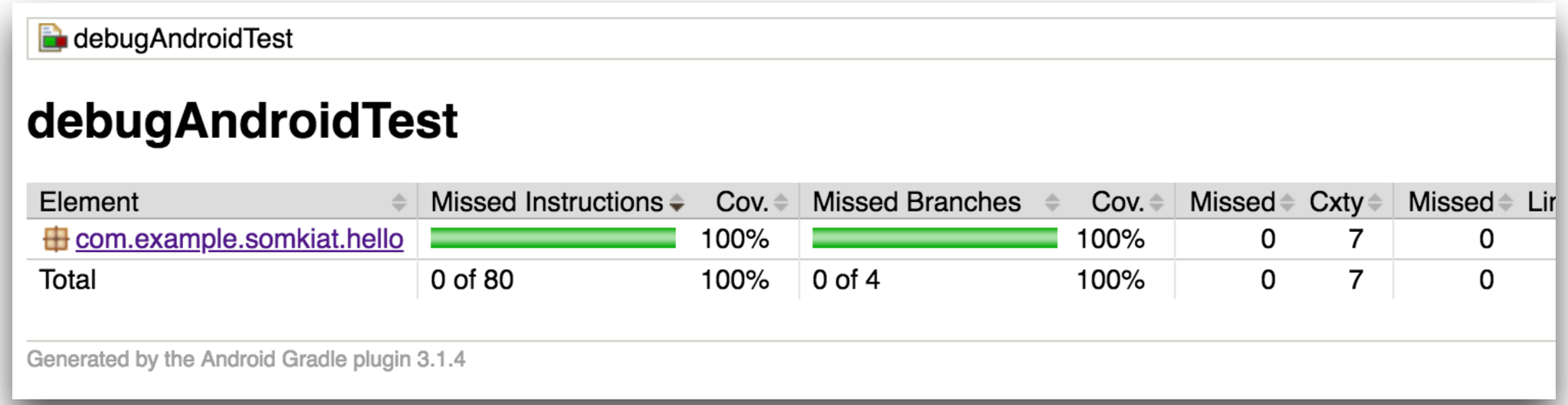
Edit file **/app/build.gradle** to refer to jacobo.gradle

```
1     plugins {
2         id 'com.android.application'
3         id 'org.jetbrains.kotlin.android'
4     }
5
6     apply from: 'jacoco.gradle'
7
```

4. Run code coverage of Android Test

$gradlew clean testCoverageReport

Result in /app/build/reports/jacoco/testCoverageReport/html/index.html


debugAndroidTest

debugAndroidTest

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lir
com.example.somkiat.hello		100%		100%	0	7	0	
Total	0 of 80	100%	0 of 4	100%	0	7	0	

Generated by the Android Gradle plugin 3.1.4

End-to-End test with UIAutomator

https://developer.android.com/training/testing/other-components/ui-automator

Shadow with Robolectric

http://robolectric.org/

Robolectric

Run tests outside of emulator/device
SDK, Resources method simulation
No mock frameworks required

1. Add dependency

File /app/build.gradle

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    testImplementation "org.robolectric:robolectric:4.2.1"
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test:rules:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-c
}
```

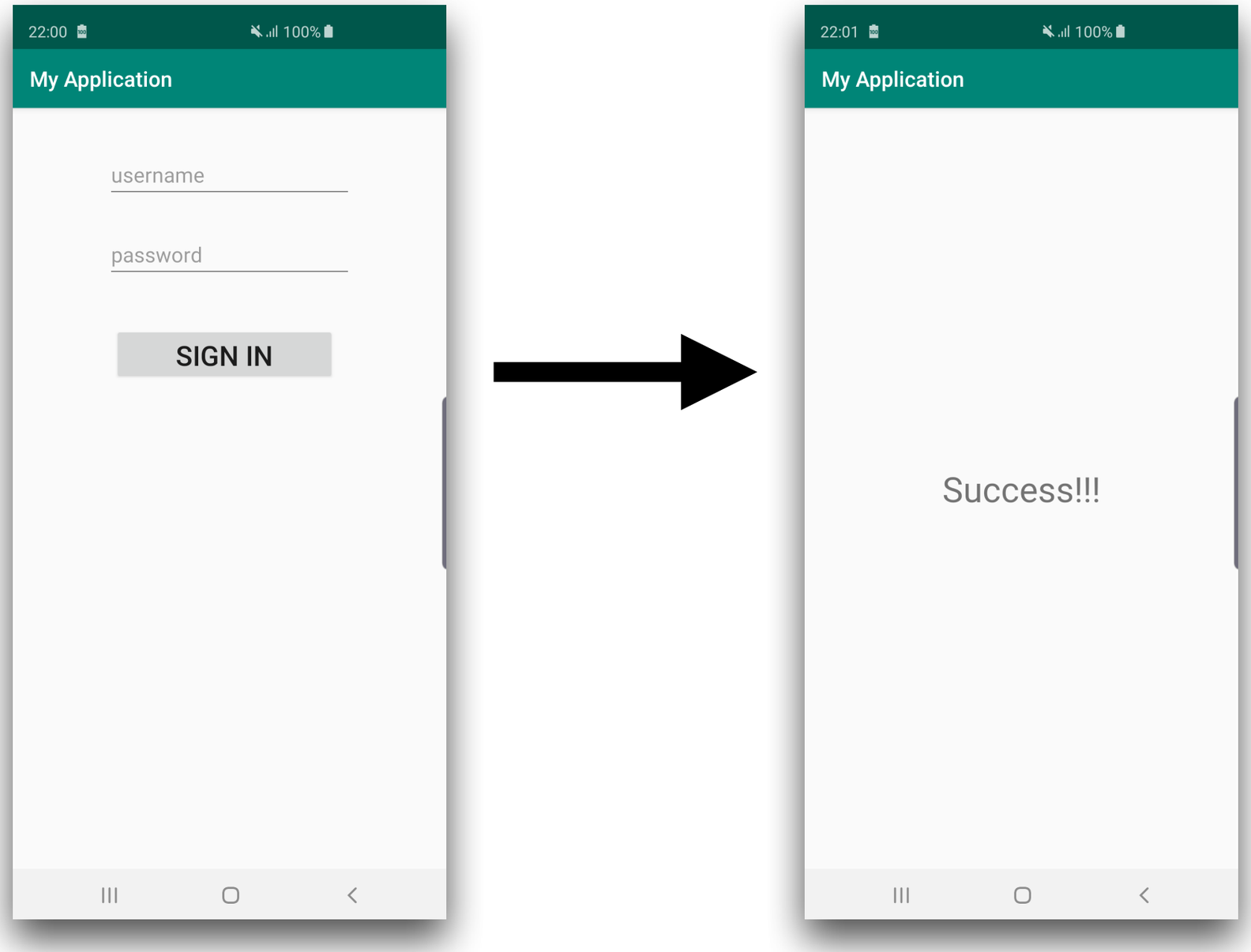
2. Add resources to Unit tests

File /app/build.gradle

```
android {
    testOptions {
        unitTests {
            includeAndroidResources = true
        }
    }
}
```

3. Write tests in /src/test

Working with Robolectric



3.1 Change test runner

File MainActivityRobolectricTest.java

```
@RunWith(RobolectricTestRunner.class)
public class MainActivityRobolectricTest {

    @Test
    public void clickingButton_shouldShowResult() {
        // Arrange
        MainActivity activity =
                Robolectric.buildActivity(MainActivity.class)
                .create()
                .resume()
                .get();
```

3.2 Create Activity for test

File MainActivityRobolectricTest.java

```
@RunWith(RobolectricTestRunner.class)
public class MainActivityRobolectricTest {

    @Test
    public void clickingButton_shouldShowResult() {
        // Arrange
        MainActivity activity =
                Robolectric.buildActivity(MainActivity.class)
                .create()
                .resume()
                .get();
```

3.3 Input data for test

File MainActivityRobolectricTest.java

```
// Act
EditText username = activity.findViewById(R.id.et_username);
username.setText("admin");
EditText password = activity.findViewById(R.id.et_passsword);
password.setText("password");
Button button = activity.findViewById(R.id.btn_signin);
button.performClick();
```

3.4 Check result

Test APIs from Robolectric

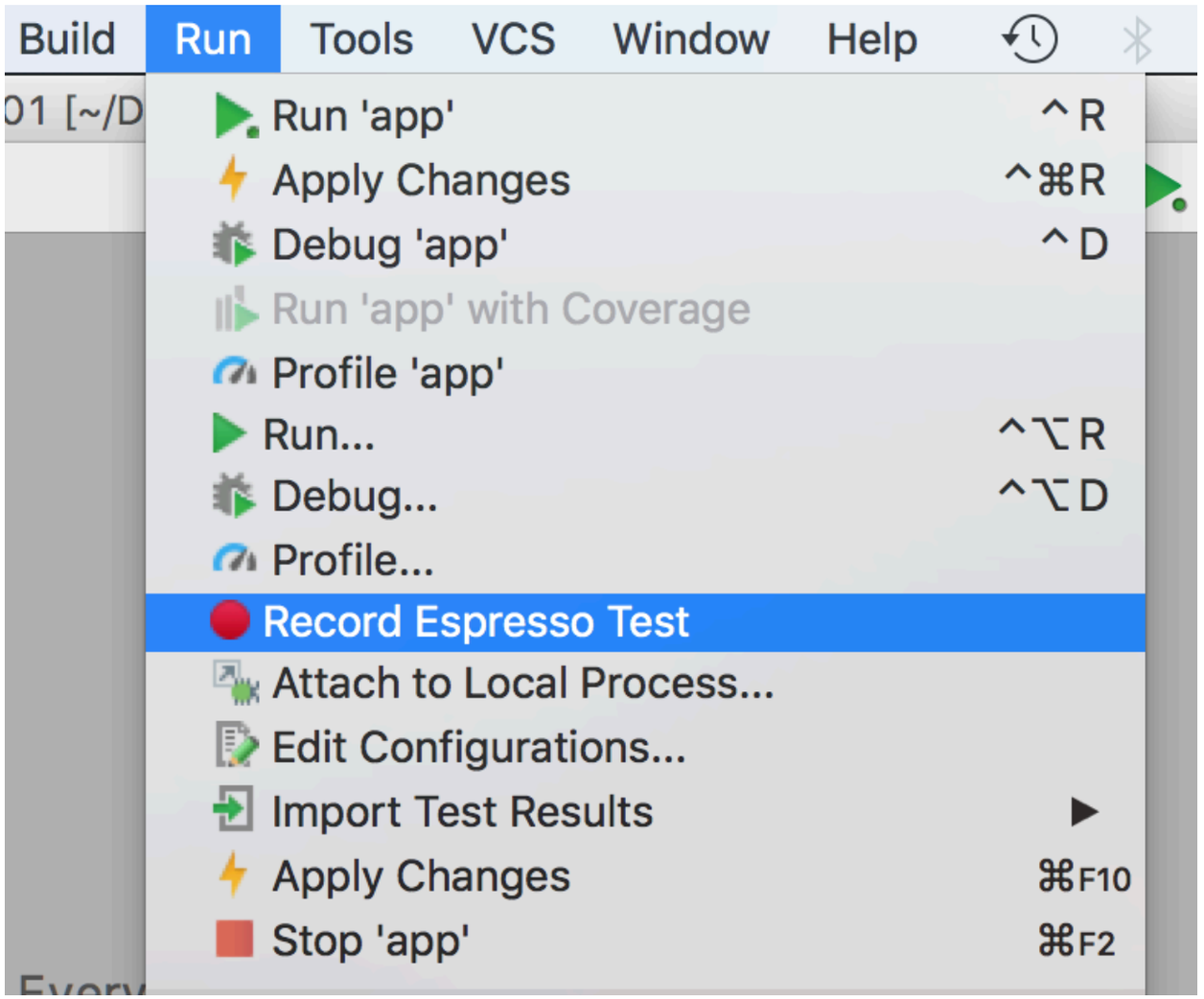
```
// Assert
Intent expectedIntent
        = new Intent(activity, ResultActivity.class);
Intent actual
        = shadowOf(activity).getNextStartedActivity();

assertEquals(expectedIntent.getComponent(),
             actual.getComponent());
assertEquals("Success!!!",
             actual.getStringExtra("RESULT"));
```

Espresso Test Recorder

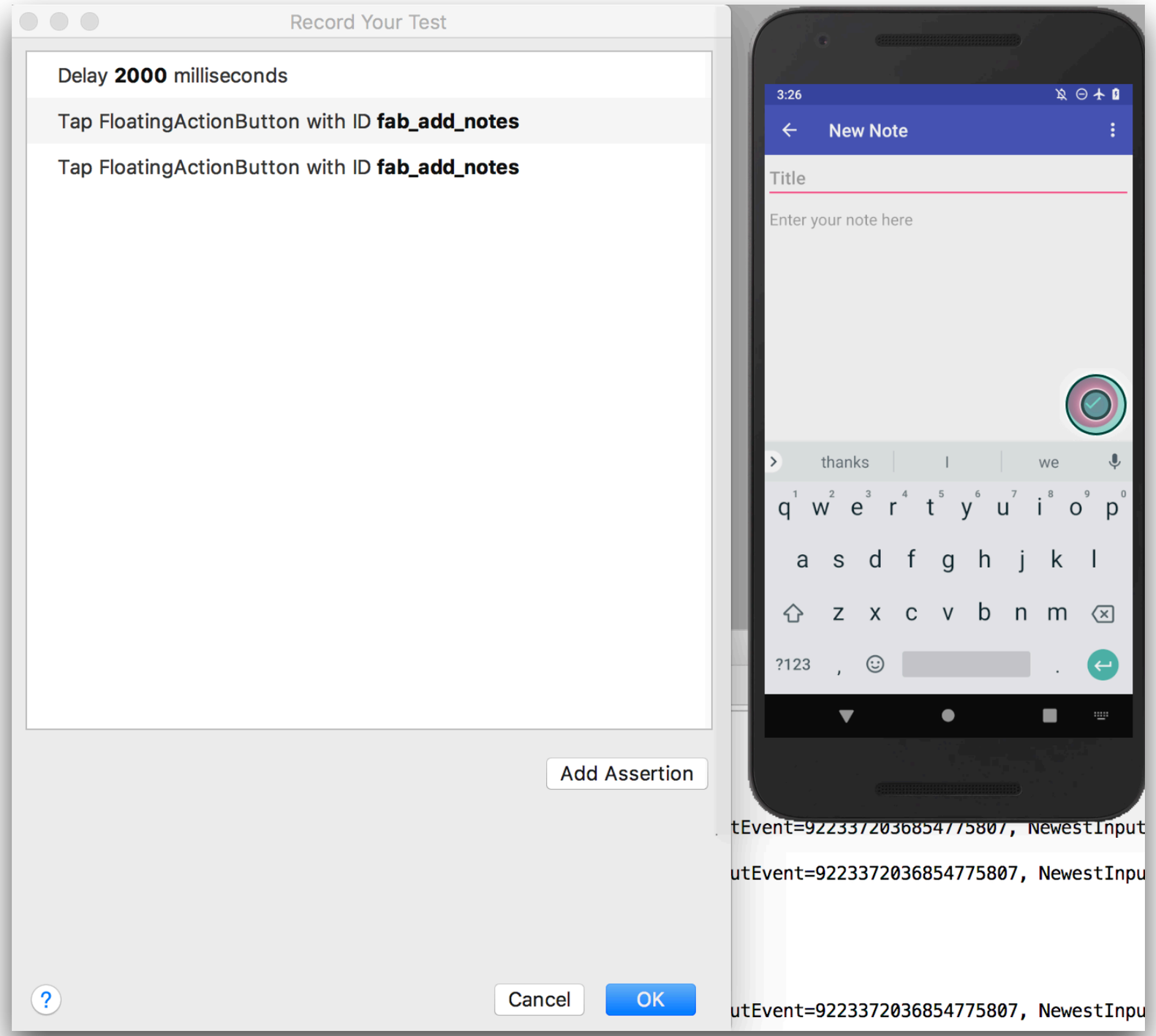
https://developer.android.com/studio/test/other-testing-tools/espresso-test-recorder

Record Espresso Test

Run -> Record Espresso Test



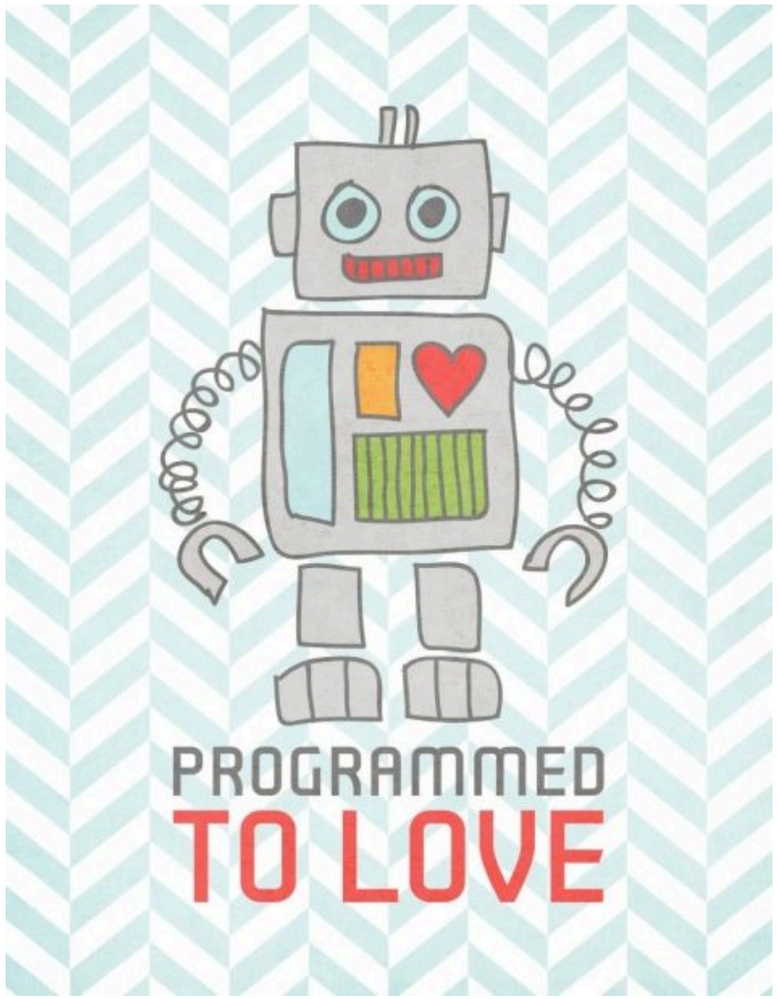
Record Espresso Test



Improve UI Testing with Espresso

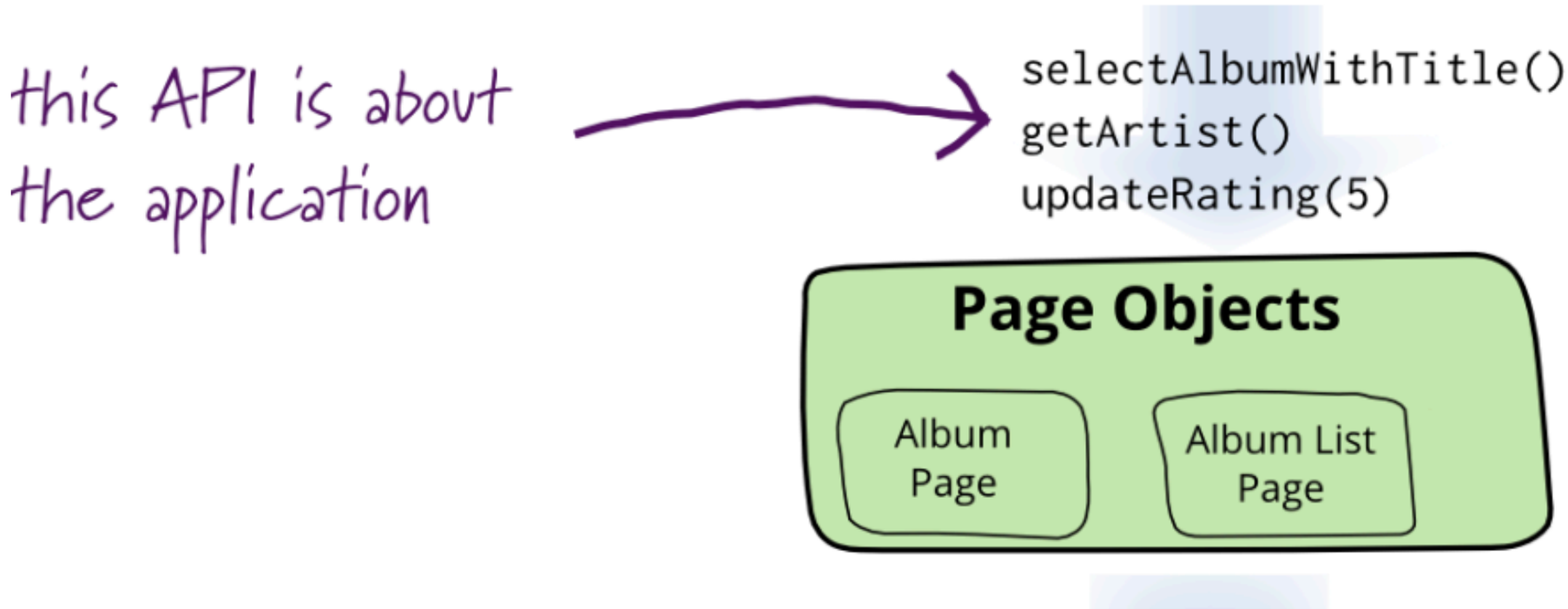
1. Improve test structure

Robot or Page Object pattern



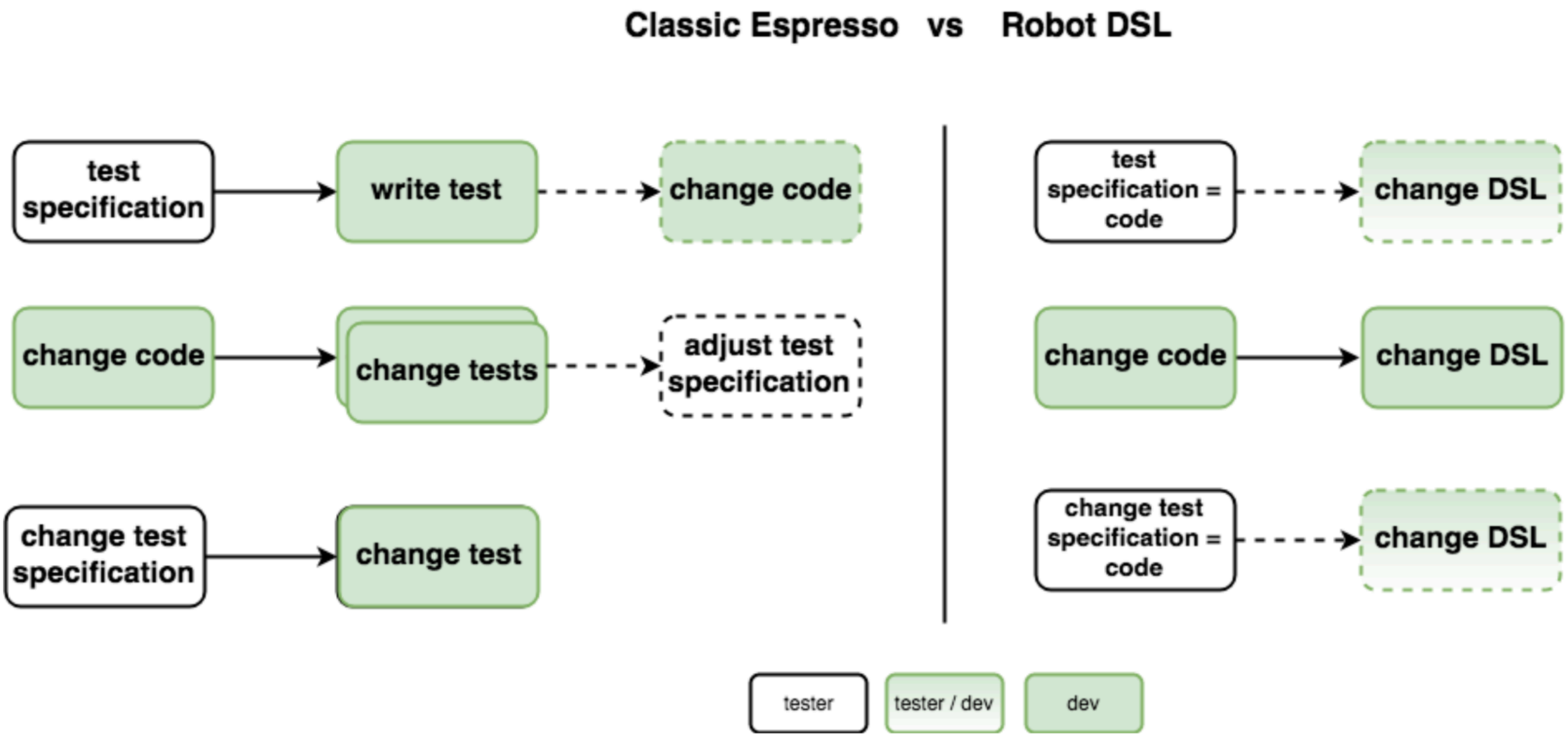
https://martinfowler.com/bliki/PageObject.html

Page Object / Robot pattern



https://proandroiddev.com/kotlin-using-test-robots-to-make-espresso-8cec2d746973

Page Object / Robot pattern



https://proandroiddev.com/kotlin-using-test-robots-to-make-espresso-8cec2d746973

Easy to read and understand

Robot or Page Object pattern

```
@Test
fun success_with_robot() {
    robot
            .setEmail("somkiat@xxx.com")
            .setPassword("")
            .clickLogin()
            .mustShow( text: "Success")
}
```

Create Login page

Create file LoginRobot.kt

```
class LoginRobot(val activity: Activity){

    fun setEmail(email: String)
            = apply { fillEditText(R.id.email, email) }
    fun setPassword(password: String)
            = apply { fillEditText(R.id.password, password) }
    fun clickLogin() = apply { clickButton(R.id.email_sign_in_button) }
    fun mustShow(text: String) = apply { toast(text) }
```

Create Login page

Create file LoginRobot.kt

```
private fun fillEditText(resId: Int, text: String): ViewInteraction =
        onView(withId(resId))
                .perform(replaceText(text),
                        closeSoftKeyboard())

private fun clickButton(resId: Int): ViewInteraction =
        onView((withId(resId))).perform(click())

private fun matchText(viewInteraction: ViewInteraction,
                      text: String): ViewInteraction = viewInteraction
        .check(ViewAssertions.matches(ViewMatchers.withText(text)))
```

2. Use library/DSL :: Kakao

https://github.com/KakaoCup/Kakao

Add dependencies

In file /app/build.gradle

```
dependencies {

    androidTestImplementation 'com.agoda.kakao:kakao:1.3.0'
}
```

Easy to read and understand

```
fun success_with_kakao() {
    screen {
        email {
            replaceText("somkiat@xxx.com")
            closeSoftKeyboard()
        }
        password {
            replaceText("")
        }
        login {
            click()
        }
    }
```

Login Screen

```
open class TestLoginScreen: Screen<TestLoginScreen>() {

    val email: KEditText = KEditText { withId(R.id.email)}
    val password: KEditText = KEditText { withId(R.id.password)}
    val login: KButton = KButton { withId(R.id.email_sign_in_button) }

}
```

Local/Unit testing

https://developer.android.com/training/testing/local-tests

Local/Unit testing

Write in folder **src/test**

Working with Test double

Verify the behavior of small section of code

Run very fast

Good Unit Test

F.I.R.S.T

Fast
Isolate/Indepondece
Repeatable
Self-verify
Timely

Dependencies

JUnit 4

Mockito

Mockk

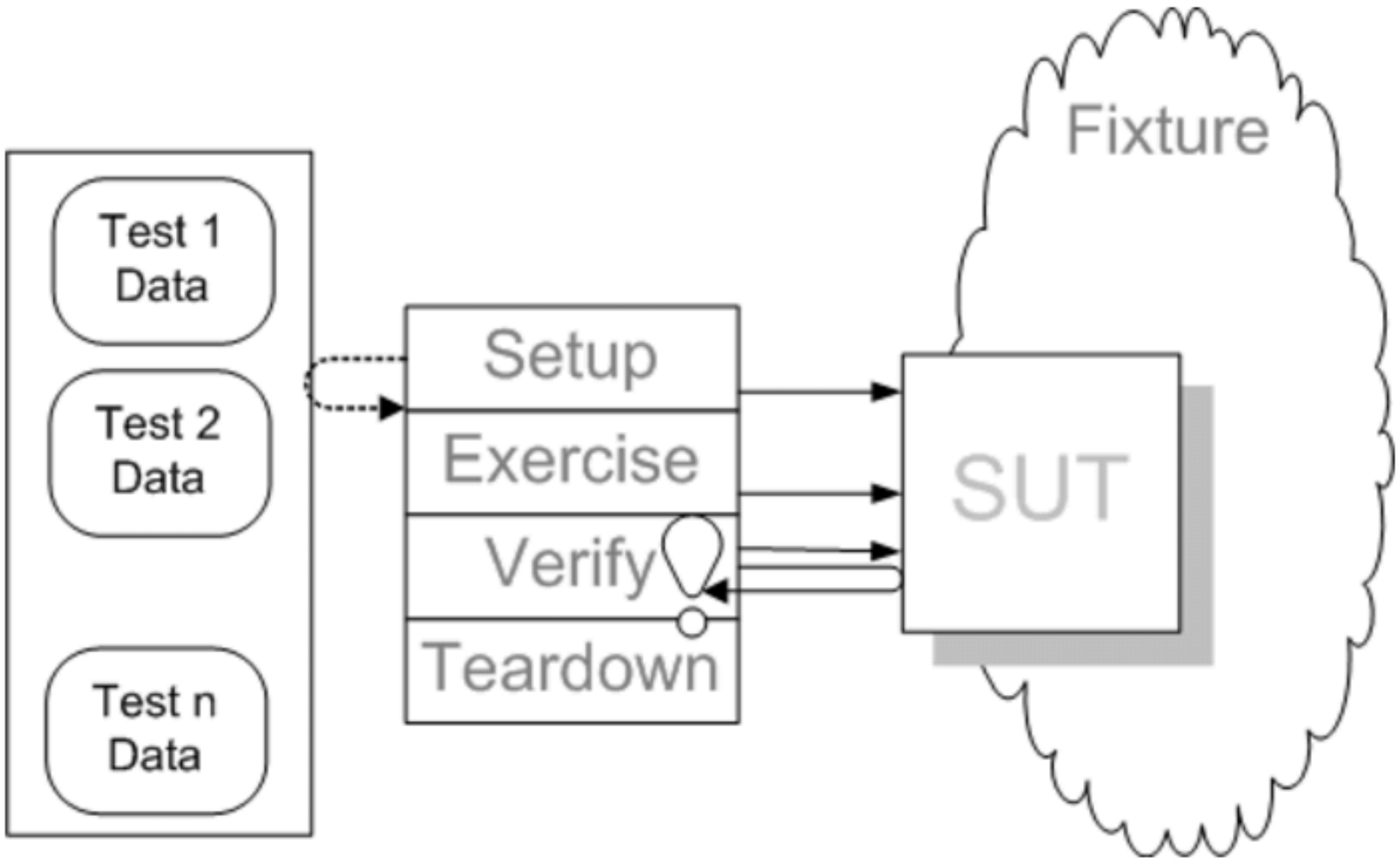
Robolectric

Dependencies

```
dependencies {
  // Required -- JUnit 4 framework
  testImplementation “junit:junit:$jUnitVersion"

  // Optional -- Robolectric environment
  testImplementation “androidx.test:core:$androidXTestVersion"

  // Optional -- Mockito framework
  testImplementation “org.mockito:mockito-core:$mockitoVersion"

  // Optional -- mockito-kotlin
  testImplementation “org.mockito.kotlin:mockito-kotlin:$mockitoKotlinVersion"

  // Optional -- Mockk framework
  testImplementation "io.mockk:mockk:$mockkVersion"
}
```

Working with data-driven testing

Data-Driven Testing



Data-Driven Testing

JUnit Parameterized
JUnitParams

1. JUnit Parameterized

JUnit Parameterized
JUnitParams

https://github.com/junit-team/junit4/wiki/parameterized-tests

Working with Parameterized

RunWith class Parameterized
Add @Parameters in data method
Create constructor of test to receive data
Write test case with data

Working with Parameterized (1)

1

```
@RunWith(Parameterized.class)
public class MainActivityValidateInputParameterizedTest {
```

```
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
                { "", "12345", "Invalid email" }
        });
    }
}
```

2

Working with Parameterized (2)

Create datas and constructor

```
private String email;
private String password;
private String expectedResult;

public MainActivityValidateInputParameterizedTest(
        String email, String password, String expectedResult) {
    this.email = email;
    this.password = password;
    this.expectedResult = expectedResult;
}
```

Working with Parameterized (3)

Create test case and use datas

```
@Test
public void validate() {
    onView(withId(R.id.edtEmail))
            .perform(
                    typeText(this.email),
                    closeSoftKeyboard());
    onView(withId(R.id.edtPassword))
            .perform(
                    typeText(this.password),
                    closeSoftKeyboard()
            );
    onView(withId(R.id.btnLogin))
            .perform(click());

    onView(withId(R.id.edtEmail))
            .check(matches(hasErrorText(this.expectedResult)));
}
```

1

2

3

Run your test

$gradlew cAT

Class com.example.somkiat.hello.MainActivityValidateInputParameterizedTest

all > com.example.somkiat.hello > MainActivityValidateInputParameterizedTest

1	0	2.585s
tests	failures	duration

100%
successful

Tests

Test	Nexus_5X_API_28(AVD) - 9
validate[0]	passed (2.585s)

2. JUnitParams

Add dependency in /app/build.gradle

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0-rc01'
    implementation 'com.android.support.constraint:constraint-layout:
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test:rules:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espre
```

```
androidTestImplementation 'pl.pragmatists:JUnitParams:1.1.1'
```

https://github.com/Pragmatists/JUnitParams

Using JUnitParams

```
@RunWith(JUnitParamsRunner.class)
public class MainActivityValidateInputJUnitParamsTest {
```

1

```
    @Test
    @Parameters({", 12345, Invalid email" })
    public void validate(String email, String password, String expectedResult) {
        onView(withId(R.id.edtEmail))
                .perform(
                        typeText(email),
                        closeSoftKeyboard());
        onView(withId(R.id.edtPassword))
                .perform(
                        typeText(password),
                        closeSoftKeyboard()
                );
```

2

https://github.com/Pragmatists/JUnitParams

Unit testing

Unit testing with validation input

How to testing with unit test ?
How to write test cases ?

Create new class in /app/src/main

Class Validation

```
import android.text.TextUtils;

public class Validation {

    public boolean isEmpty(String input) {
        return TextUtils.isEmpty(input);
    }

}
```

Create unit test of validation

Create file in /app/test

```
public class ValidationTest {

    @Test
    public void isEmpty() {
        Validation validation = new Validation();
        assertTrue(validation.isEmpty(""));
    }
}
```

Run unit test

$gradlew testDebugUnitTest

Found error :: Method isEmpty in android.text.TextUtils not mock

Change code to use pure Java

```
public class Validation {

    public boolean isEmpty(String input) {
        return input == null ||
                input.trim().length() == 0;
    }

}
```

Add more test cases

```
@Test
public void with_empty_string() {
    Validation validation = new Validation();
    assertTrue(validation.isEmpty(""));
}

@Test
public void with_null() {
    Validation validation = new Validation();
    assertTrue(validation.isEmpty(null));
}

@Test
public void with_space() {
    Validation validation = new Validation();
    assertTrue(validation.isEmpty(" "));
}
```

Run unit test again

$gradlew testDebugUnitTest

Class com.example.somkiat.hello.ValidationTest

all > com.example.somkiat.hello > ValidationTest

3	0	0	0.005s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
with_empty_string	0s	passed
with_null	0s	passed
with_space	0.005s	passed

Monkey testing

https://developer.android.com/studio/test/other-testing-tools/monkey

Using monkey test (Stress testing)

$**adb** shell monkey -p your.package.name -v 500

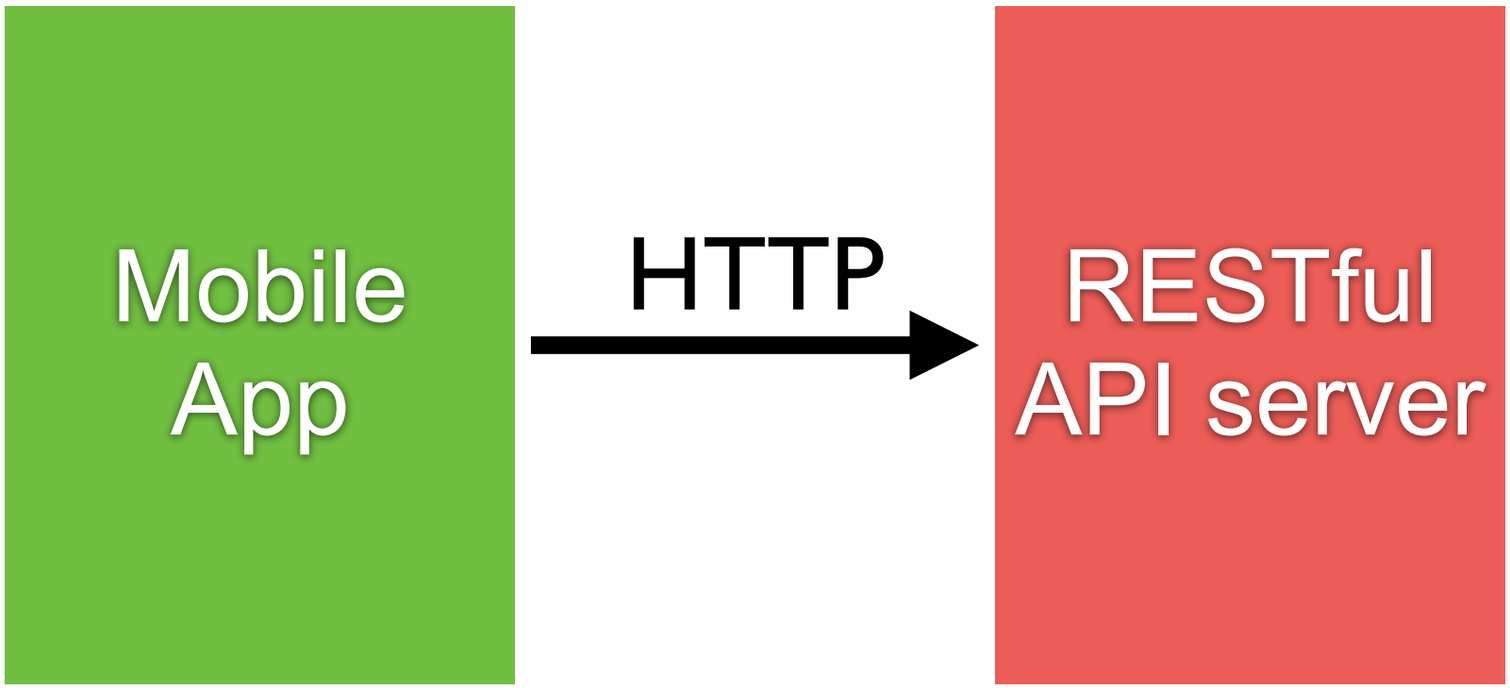
Number of events for testing

https://developer.android.com/studio/test/other-testing-tools/monkey

Networking testing

https://github.com/up1/workshop_android_testing_with_networking

How to testing ?



How to testing with network ?

Mock API Server (Internal or External)
Configurable endpoint of API
No need to change code

First test

Call real API

```
@Test
public void success_with_realAPI() {
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    onView(withId(R.id.followers))
            .check(matches(withText("up1: 468")));
}
```

First test

Improve with IdleResource

IDLING RESOURCES

UI TESTING FOR ANDROID
espresso

Idle Resource ?

An idling resource represents
an **asynchronous operation** whose results affect
subsequent operations in a UI test.

Why we need Idle Resource ?

By **registering idling resources** with Espresso, you can validate these asynchronous operations **more reliably** when testing your app.

First test

Improve with IdleResource

```
@Test
public void success_with_realAPI_and_IdleResource() {
    IdlingResource idlingResource = OkHttp3IdlingResource.create(
            "okhttp", OkHttp.getInstance());
    IdlingRegistry.getInstance().register(idlingResource);

    onView(withId(R.id.followers))
            .check(matches(withText("up1: 468")));

    IdlingRegistry.getInstance().unregister(idlingResource);
}
```

https://github.com/JakeWharton/okhttp-idling-resource

Mock API server

External (Stubby4j, Wiremock)
Internal (WebMockServer)

How to change URL of API ?

Constant variable

Build config

Dependency Injection (DI)

Dependency Injection

Using build variant
Custom Test Runer
Dependency Injection framework
(Dagger, AndroidInjector)

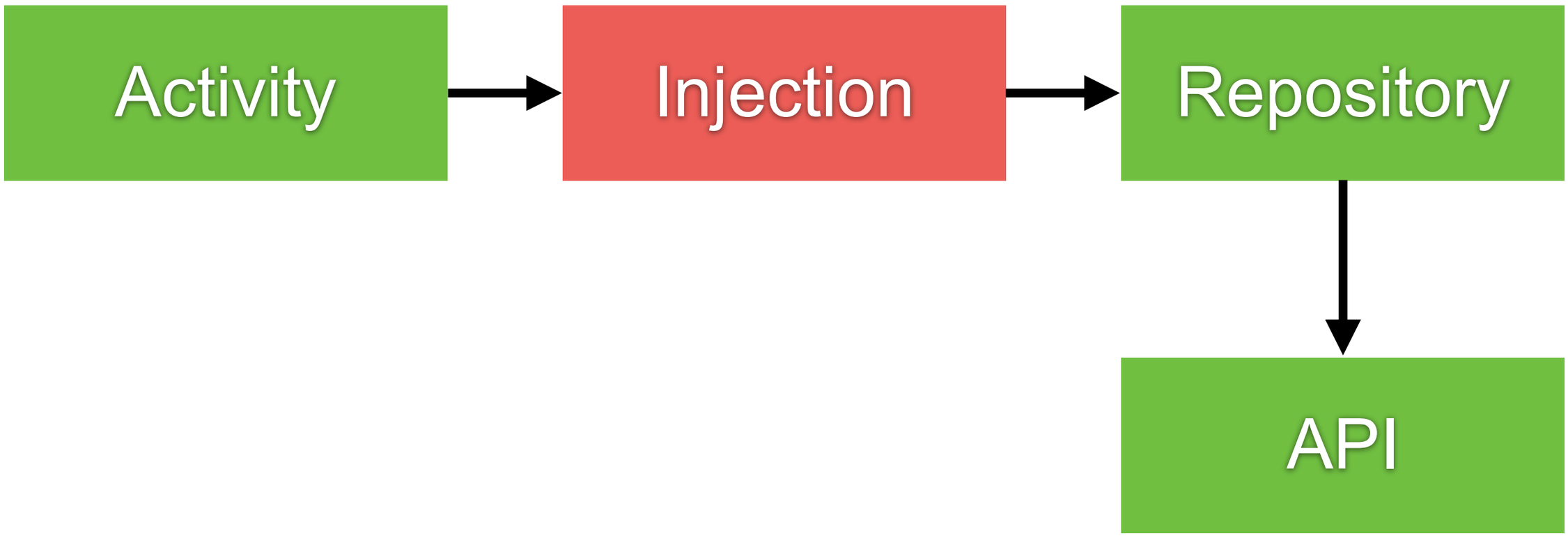
Workshop

Dependency Injection

Your project can test or not ?

Testable application ?

Look inside your project

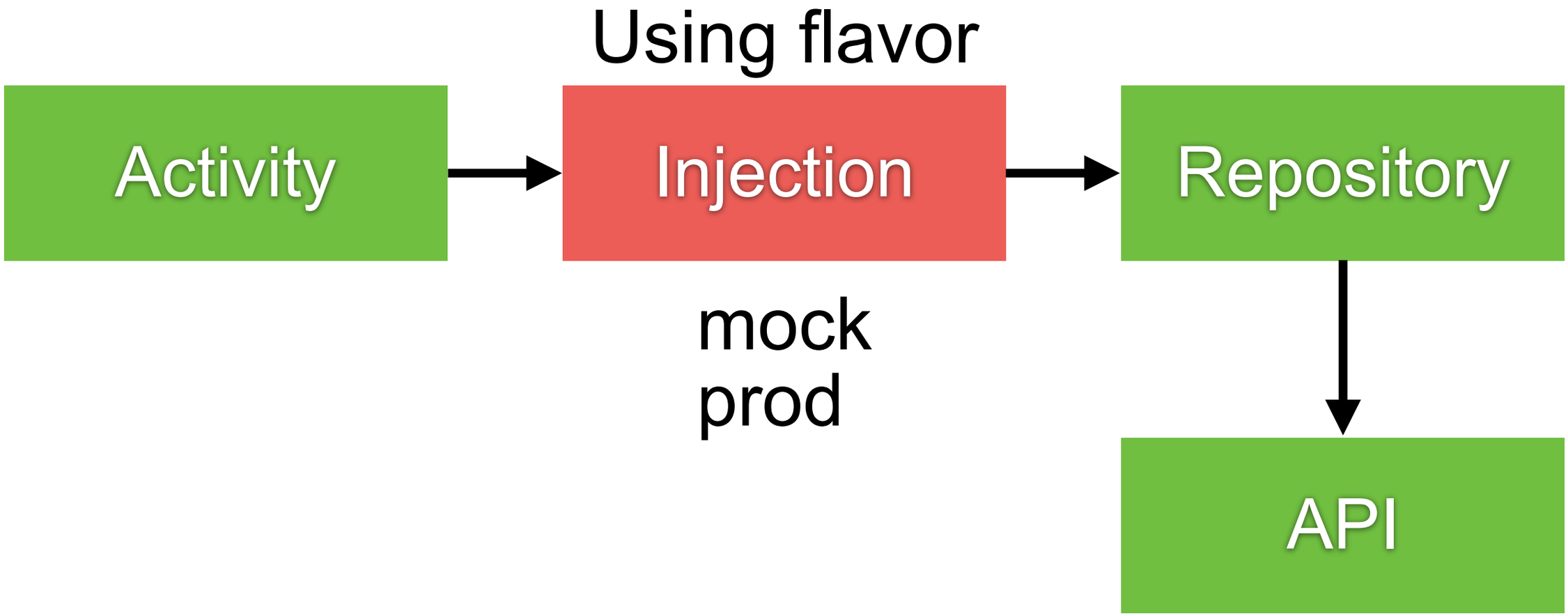


Injection.java

Use to create Repository and API

```
public class Injection {
    public static ImageStorage provideImageFile() {
        return new ImageFile();
    }

    public static NoteRepository provideNotesRepository() {
        return new InMemoryNotesRepository(new MockNoteServiceApi());
    }
}
```

Concept :: More testable



Add product flavour

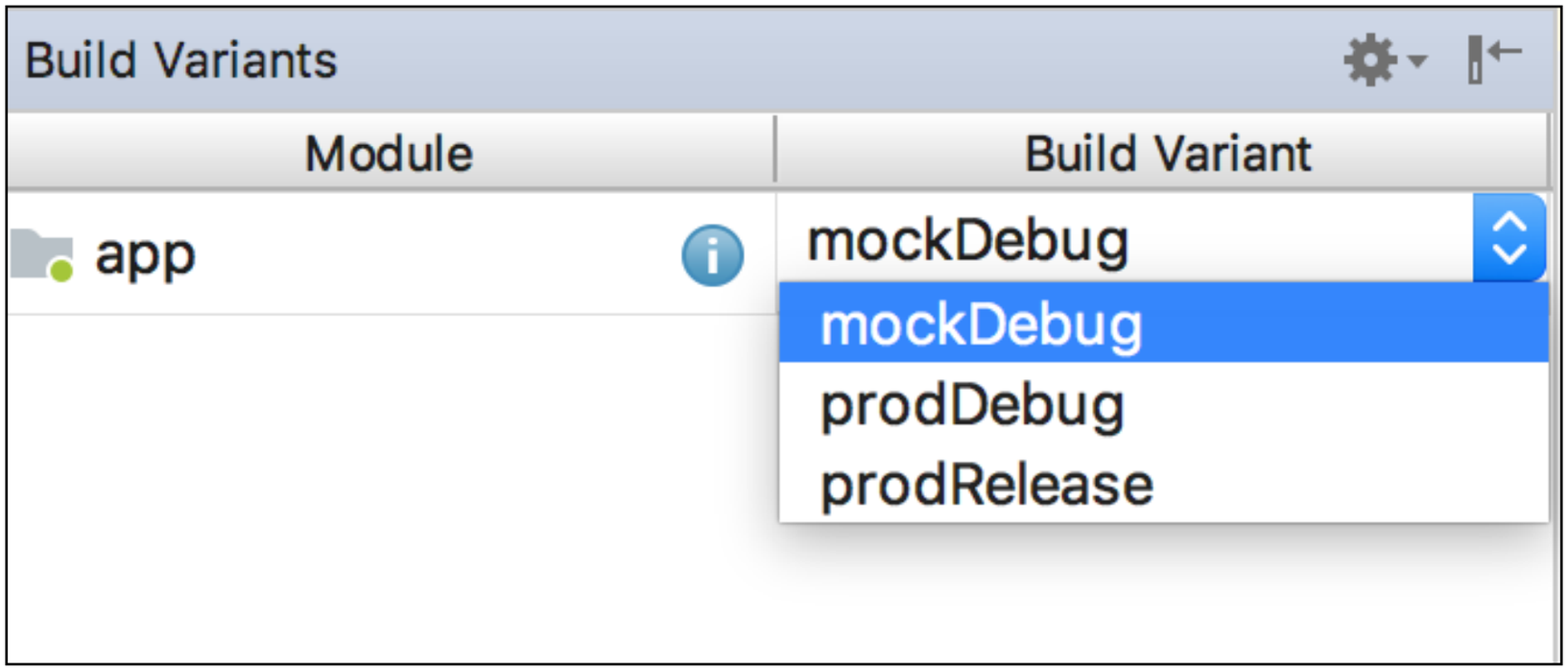
To file /app/build.gradle

```
flavorDimensions "mock"

// If you need to add more flavors, consider using flavor dimensions.
productFlavors {
    mock {
        applicationIdSuffix = ".mock"
        dimension "mock"
    }
    prod {

    }
}

// Remove mockRelease as it's not needed.
android.variantFilter { variant ->
    if(variant.buildType.name.equals('release')
            && variant.getFlavors().get(0).name.equals('mock')) {
        variant.setIgnore(true);
    }
}
```

https://github.com/up1/android-testing-workshop/wiki

Add product flavour

See result in Build Variants



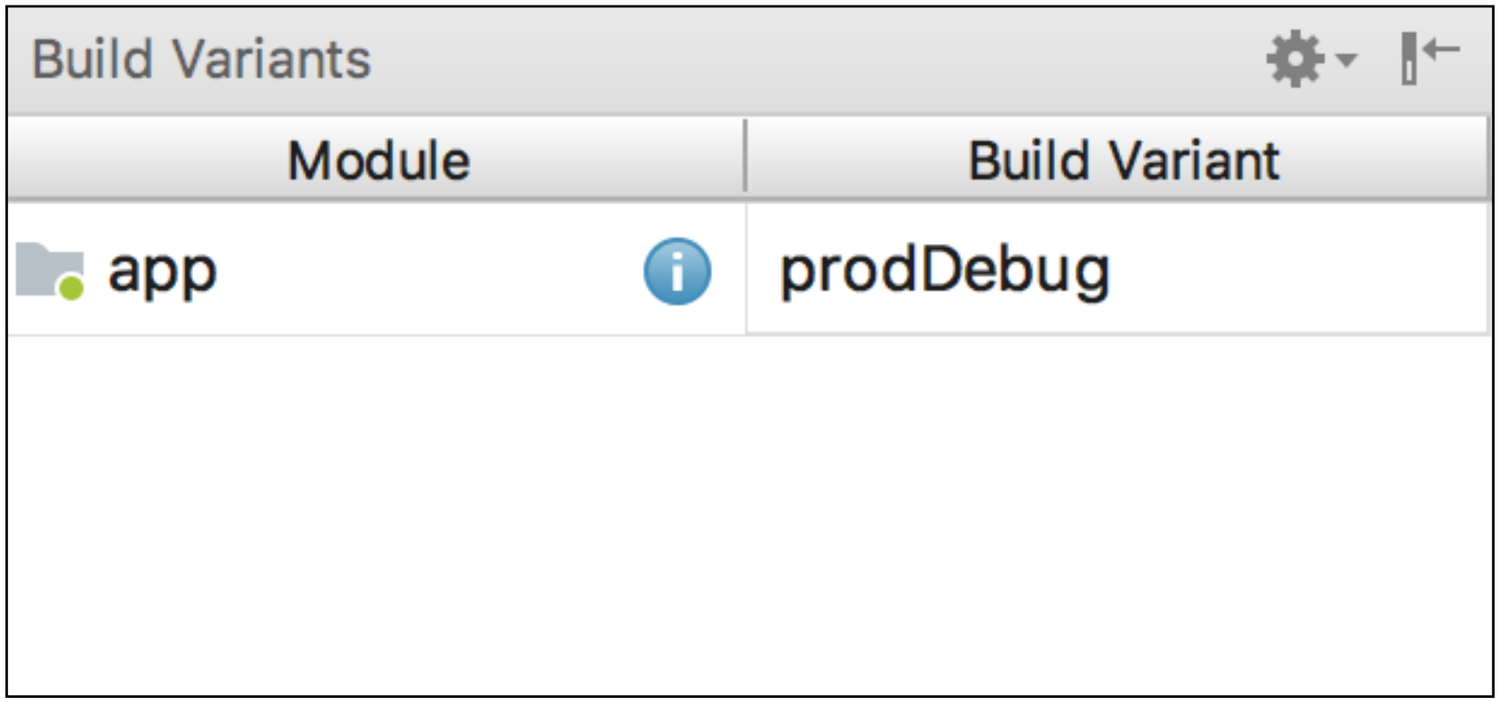
Create new folder in /app/src

mock/java/com/example/somkiat/sample
prod/java/com/example/somkiat/sample

Move class Injection into prod

prod/java/com/example/somkiat/sample

Run with prodDebug



Run with prodDebug

$./gradlew clean connectedProdDebugAndroidTest

Create class Injection in mock

mock/java/com/example/somkiat/sample

```
public class Injection {
    public static ImageStorage provideImageFile() {
        return new FakeImageFile();
    }

    public static NoteRepository provideNotesRepository() {
        return new InMemoryNotesRepository(new FakeNoteServiceApi());
    }
}
```

Create file **FakeServiceApi**

Create class FakeServiceApi

mock/java/com/example/somkiat/sample

```
class FakeNoteServiceApi implements NoteServiceApi {
    private static final Map<String, Note> NOTES_SERVICE_DATA = new HashMap<>();

    @Override
    public void getAllNotes(NotesServiceCallback<List<Note>> callback) {
        callback.onLoaded(Lists.newArrayList(NOTES_SERVICE_DATA.values()));
    }

    @Override
    public void getNote(String noteId, NotesServiceCallback<Note> callback) {
        Note note = NOTES_SERVICE_DATA.get(noteId);
        callback.onLoaded(note);
    }

    @Override
    public void saveNote(Note note) {
        NOTES_SERVICE_DATA.put(note.getId(), note);
    }

    @VisibleForTesting
    public static void addNotes(Note... notes) {
        for (Note note : notes) {
            NOTES_SERVICE_DATA.put(note.getId(), note);
        }
    }
}
```

https://github.com/up1/android-testing-workshop/wiki

Create new folder in /app/src

androidTestMock/java/com/example/somkiat/sample

Move your test class to this folder

Back to show detail test

Show detail of note

```
@Test
fun show_detail_of_note_in_screen() {
    // Arrange and Act
    val newNote = Note("Fake title", "Fake description");
    FakeNoteServiceApi.addNotes(newNote);
    val startIntent = Intent()
    startIntent.putExtra(NoteDetailActivity.EXTRA_NOTE_ID, newNote.id)
    rule.launchActivity(startIntent)

    // Assert
    onView(withId(R.id.note_detail_title))
            .check(matches(withText(newNote.title)))
    onView(withId(R.id.note_detail_description))
            .check(matches(withText(newNote.description)));
}
```

Run with mockDebug

$./gradlew clean connectedMockDebugAndroidTest

Test result

Test Summary

4	0	11.064s
tests	failures	duration

100%
successful

Packages Classes

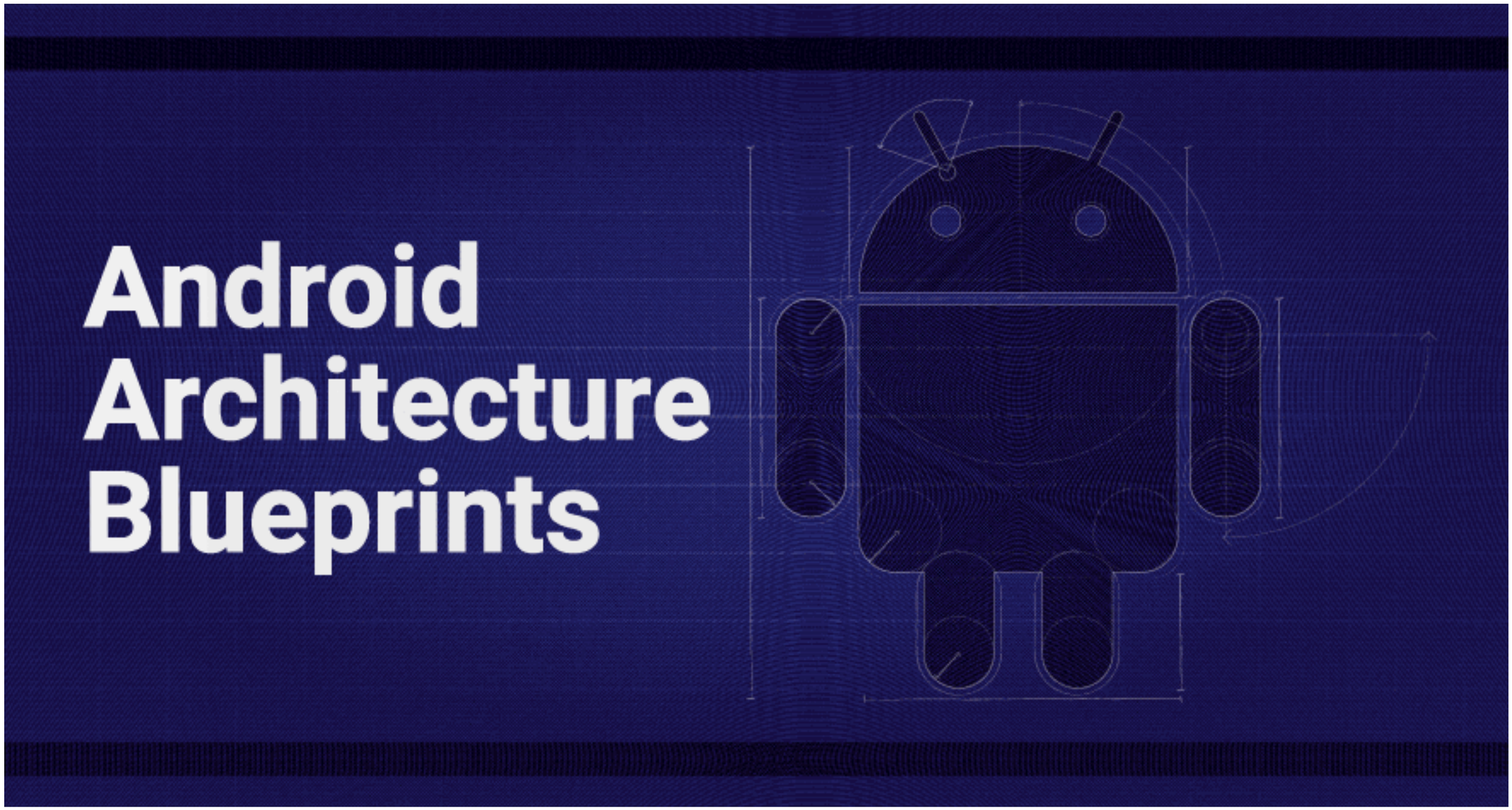
Package	Tests	Failures	Duration	Success rate
com.example.somkiat.sample	2	0	10.707s	100%
com.example.somkiat.sample.addnote	1	0	0.220s	100%
com.example.somkiat.sample.notedetail	1	0	0.137s	100%

Testable architecture

Testable architecture

MVC
MVP
MVVM
VIPER
Android Architecture Component
Clean Architecture
MVPR

Android Architecture Blueprint

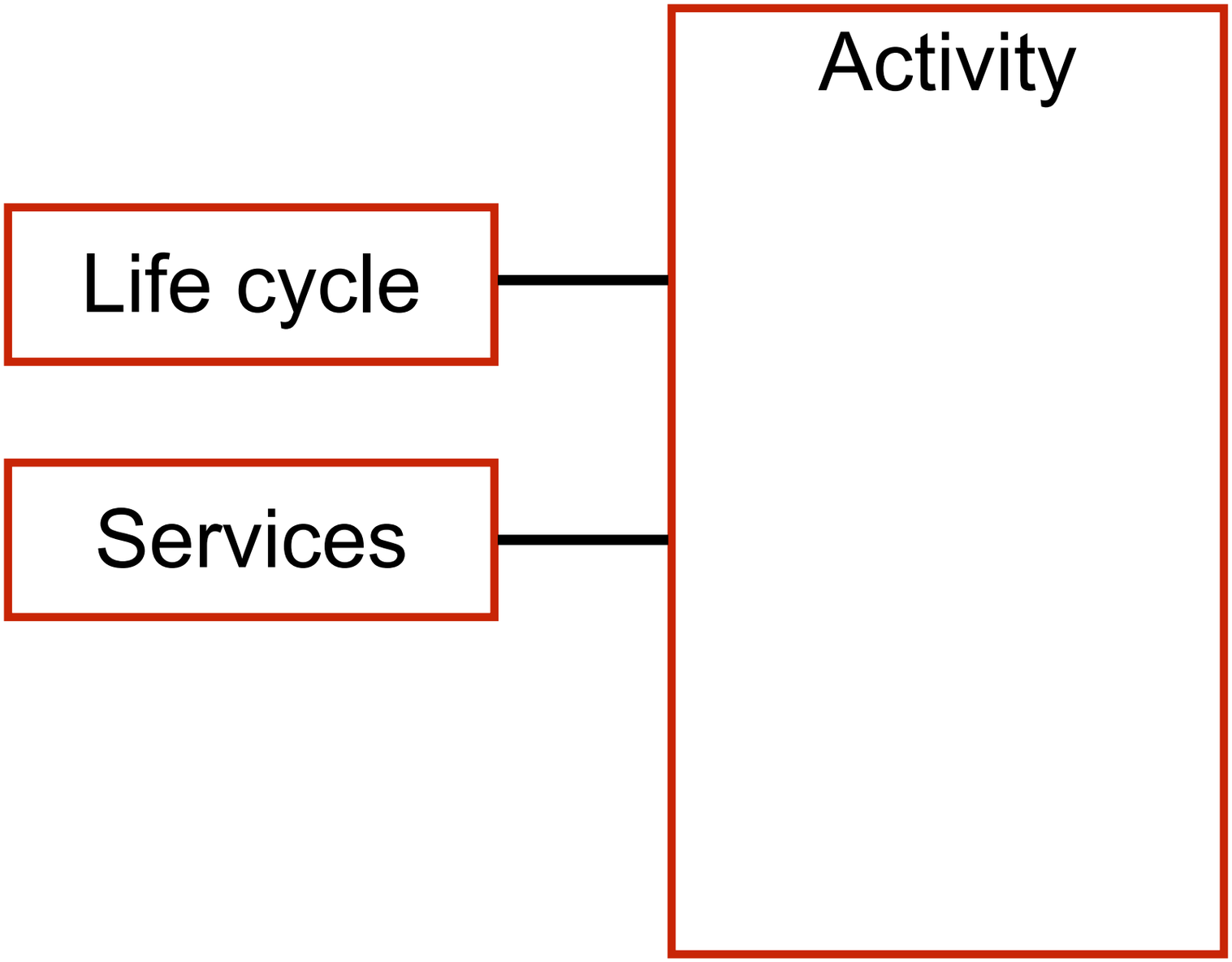


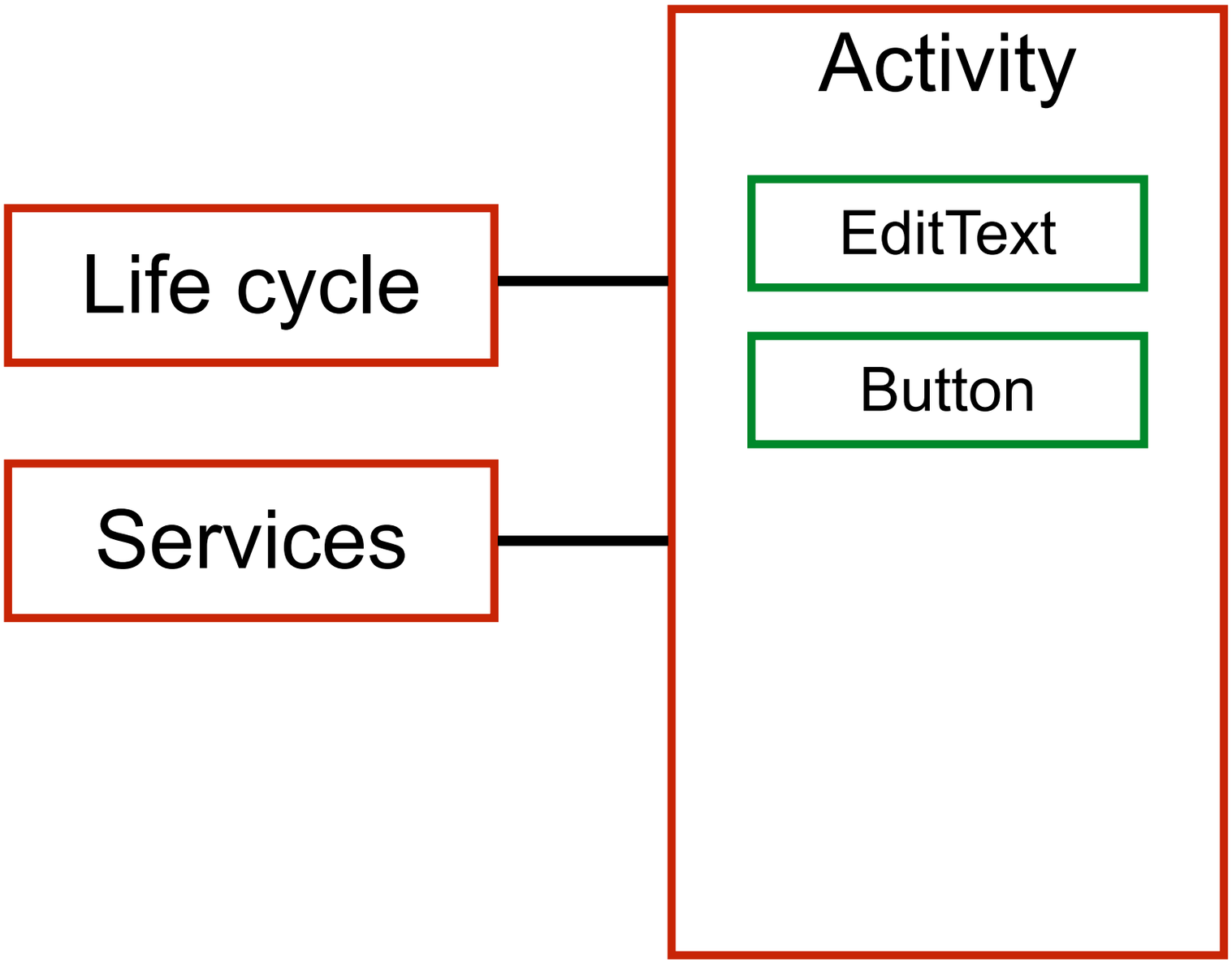
https://github.com/googlesamples/android-architecture

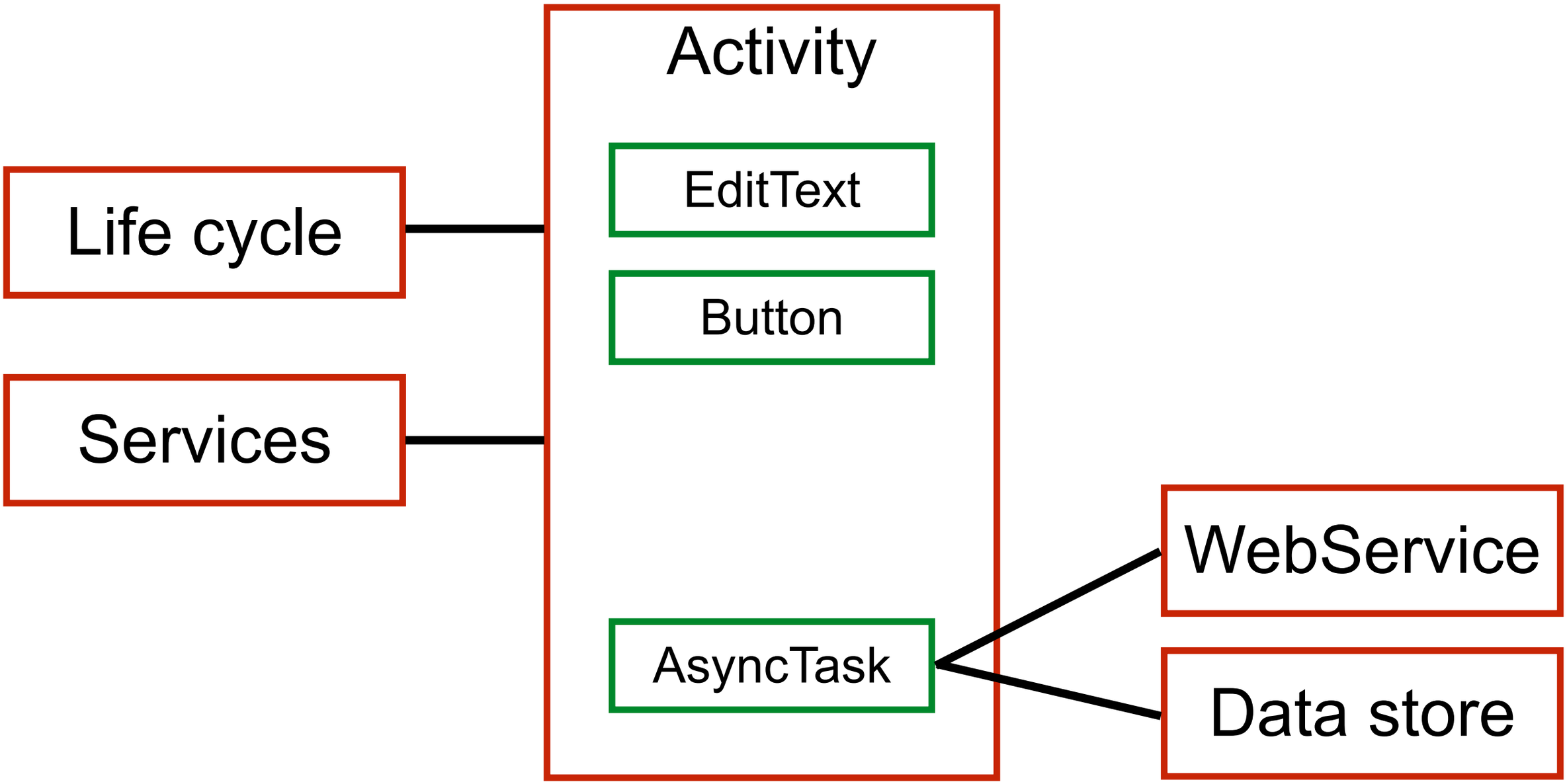
“Art is anything you can do well.
Anything you can do with **Quality**”

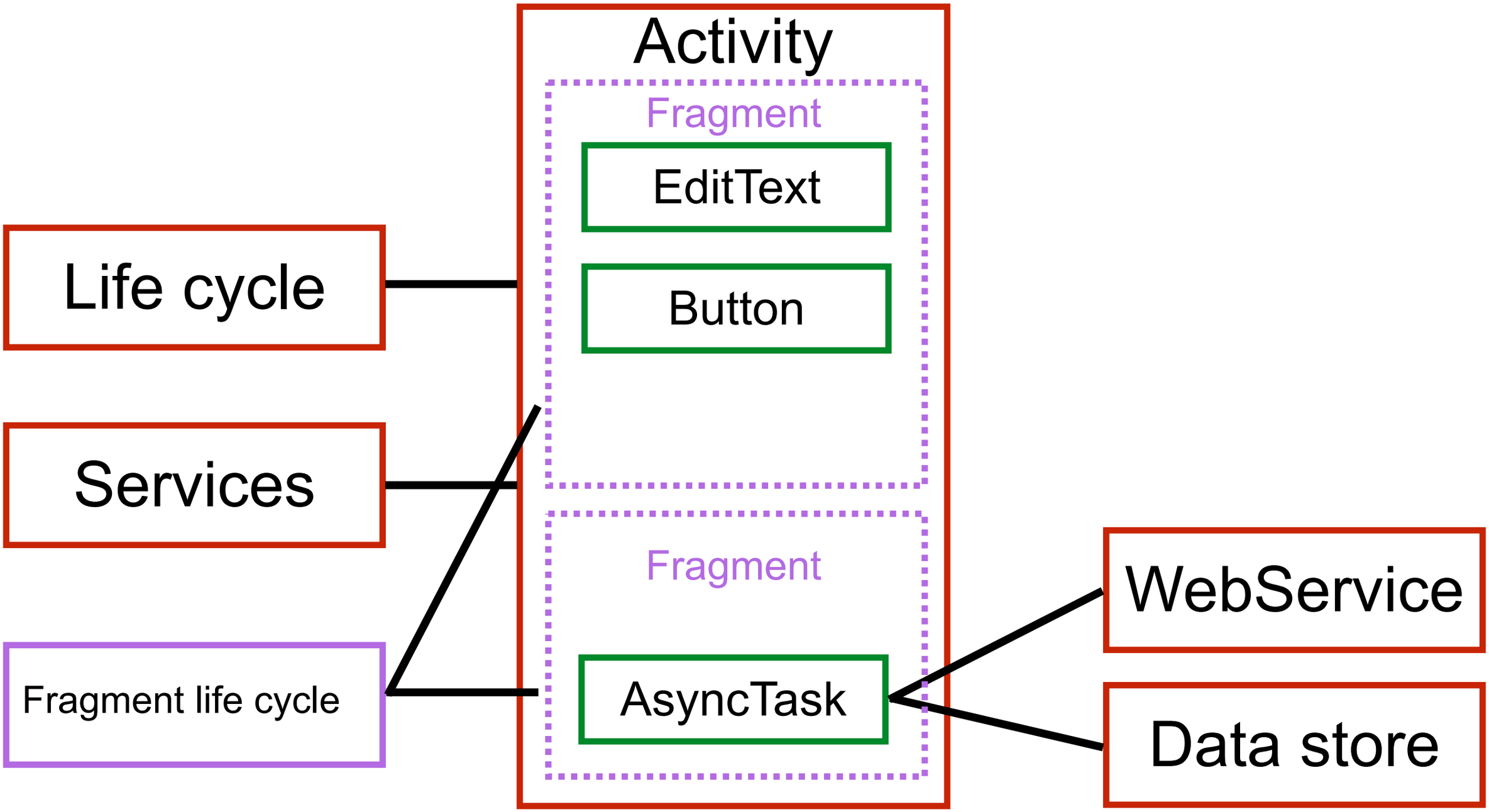
Robert M. Pirsig

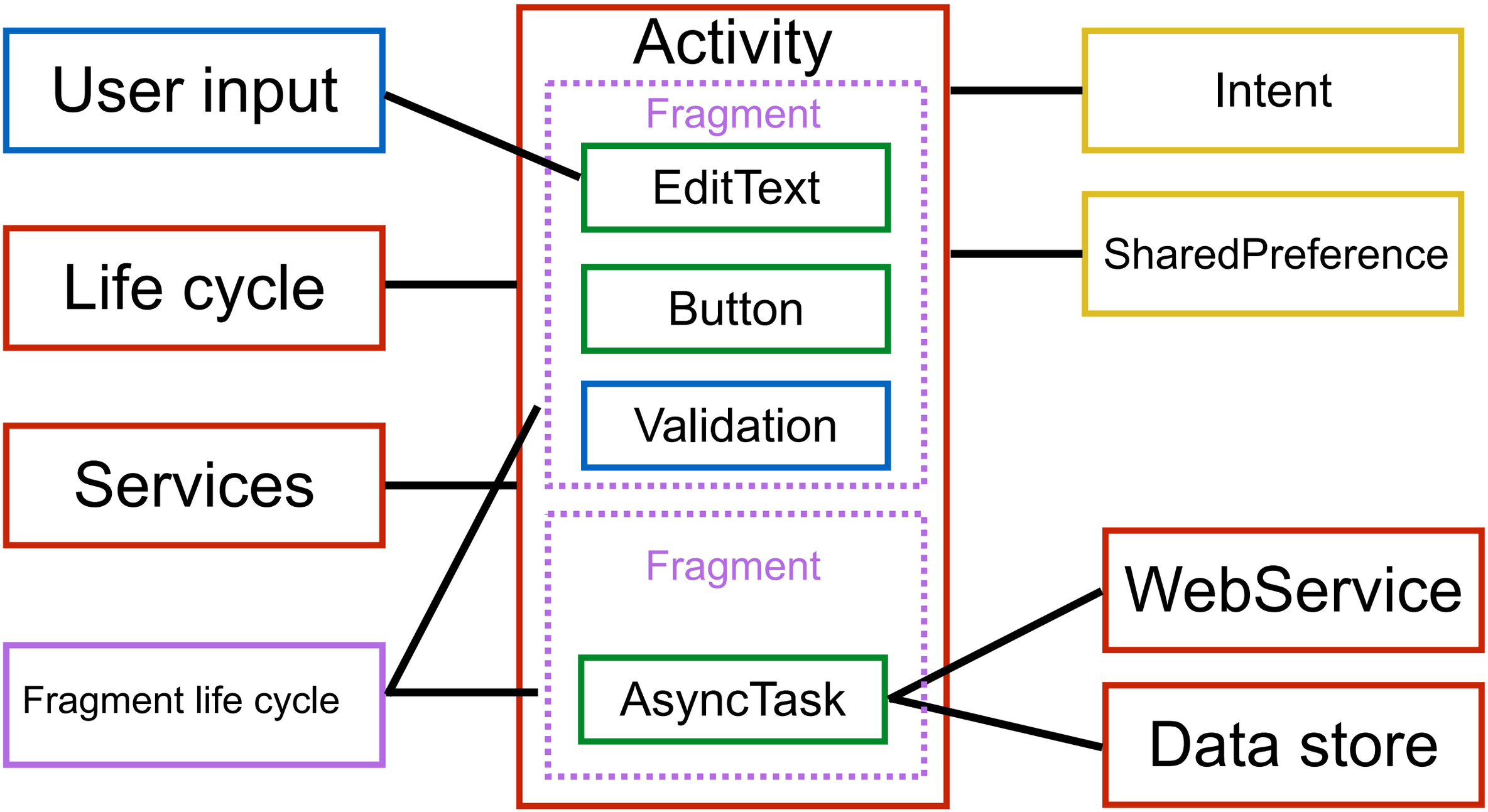
Start …









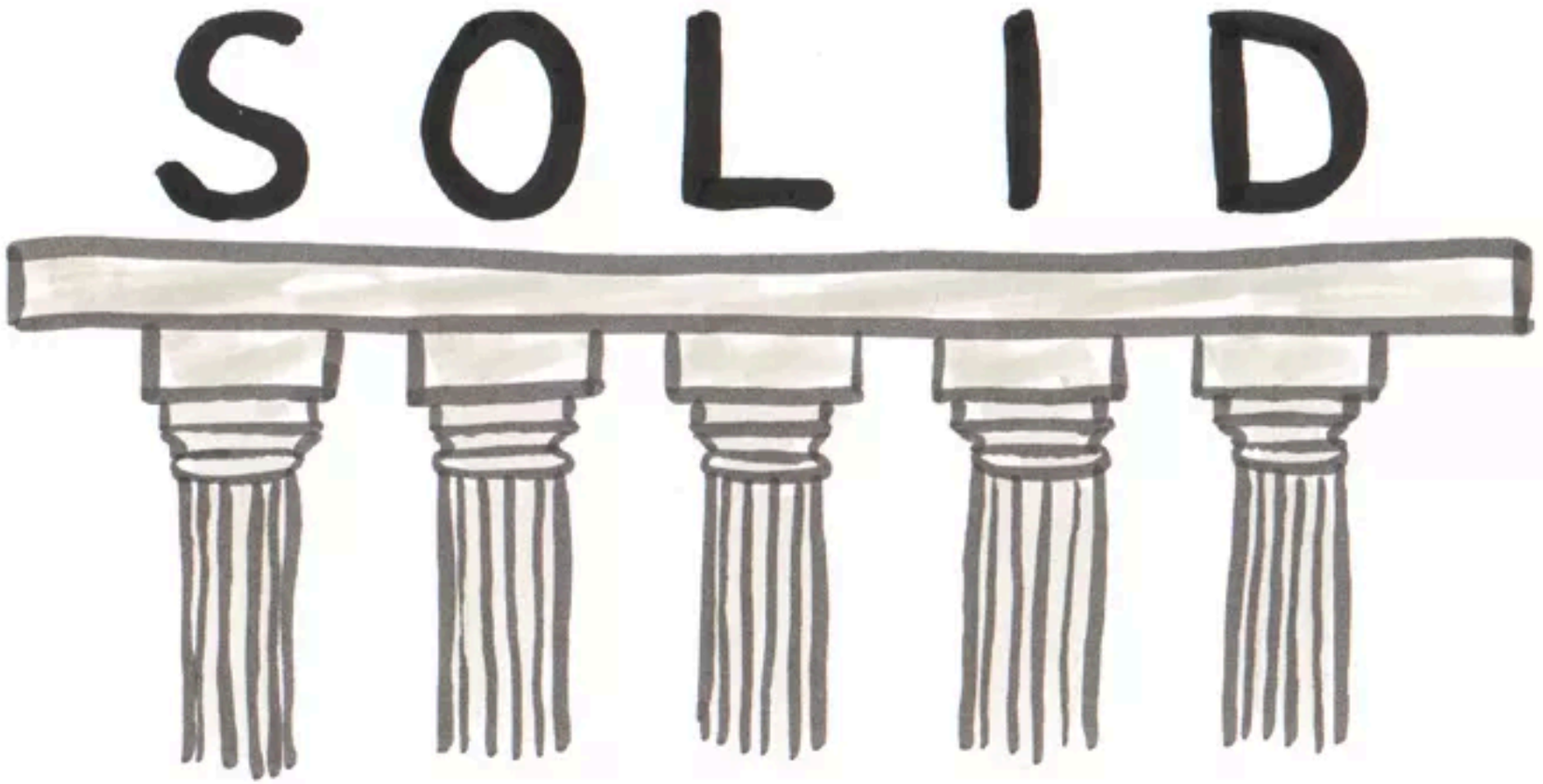


So please don't...

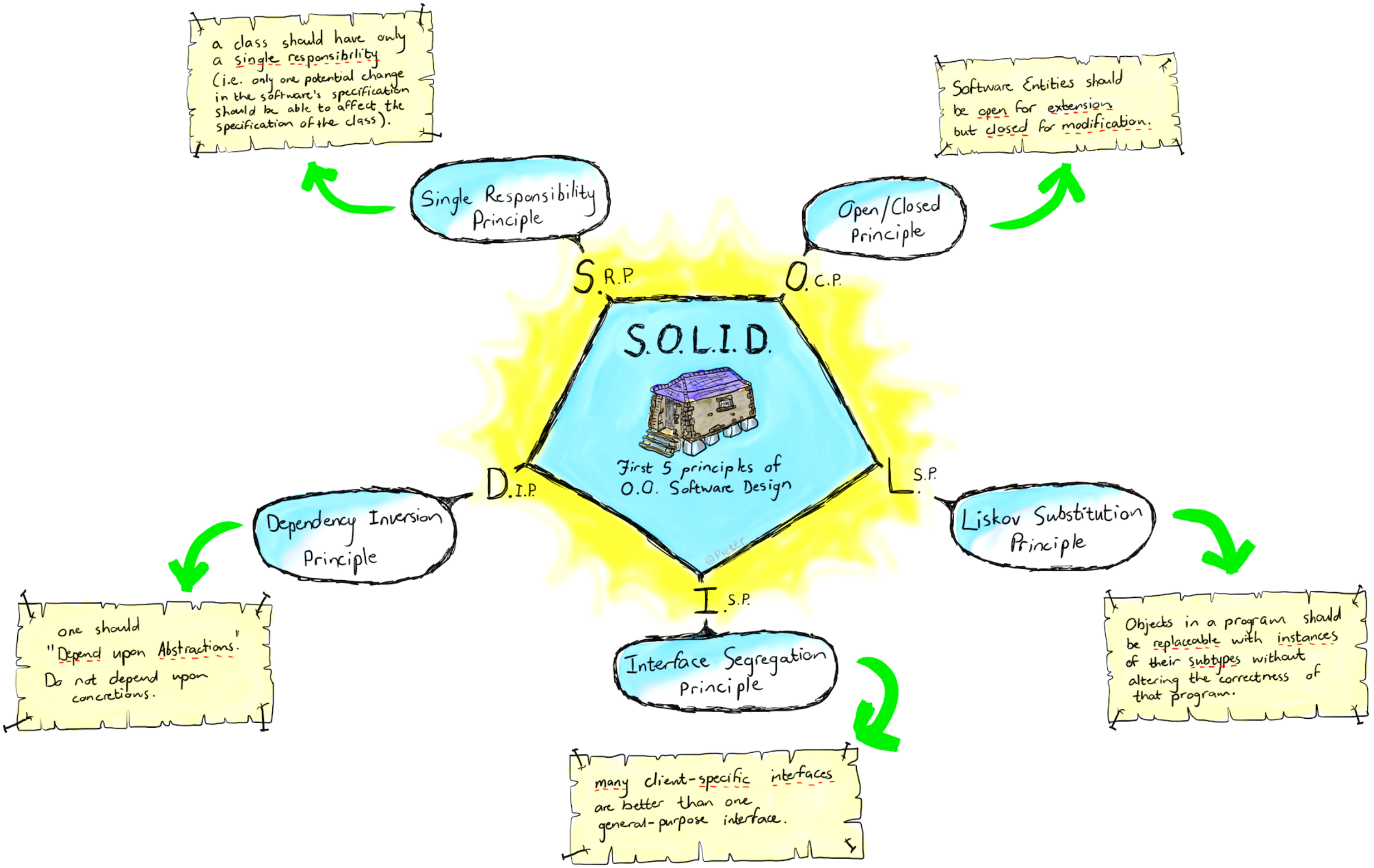
http://3.bp.blogspot.com/-FQZ4VT_gbRY/T8RvXLTPWMI/AAAAAAAABPo/JCckSpENM88/s640/SingleResponsibilityPrinciple.jpg

Separation of Concern

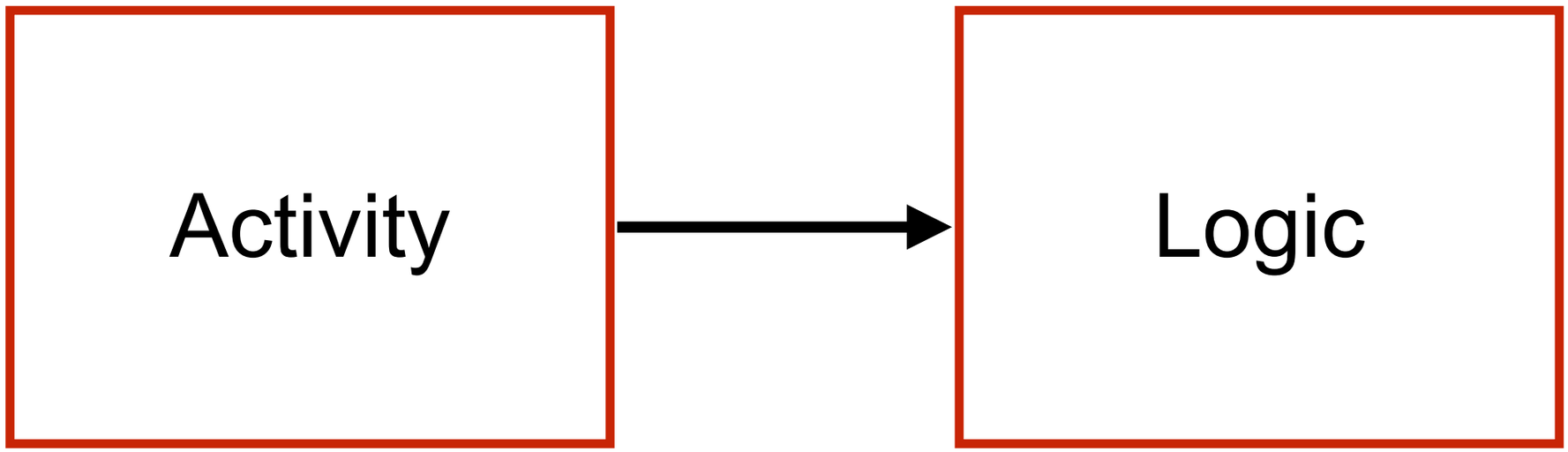
Single Responsibility Principle

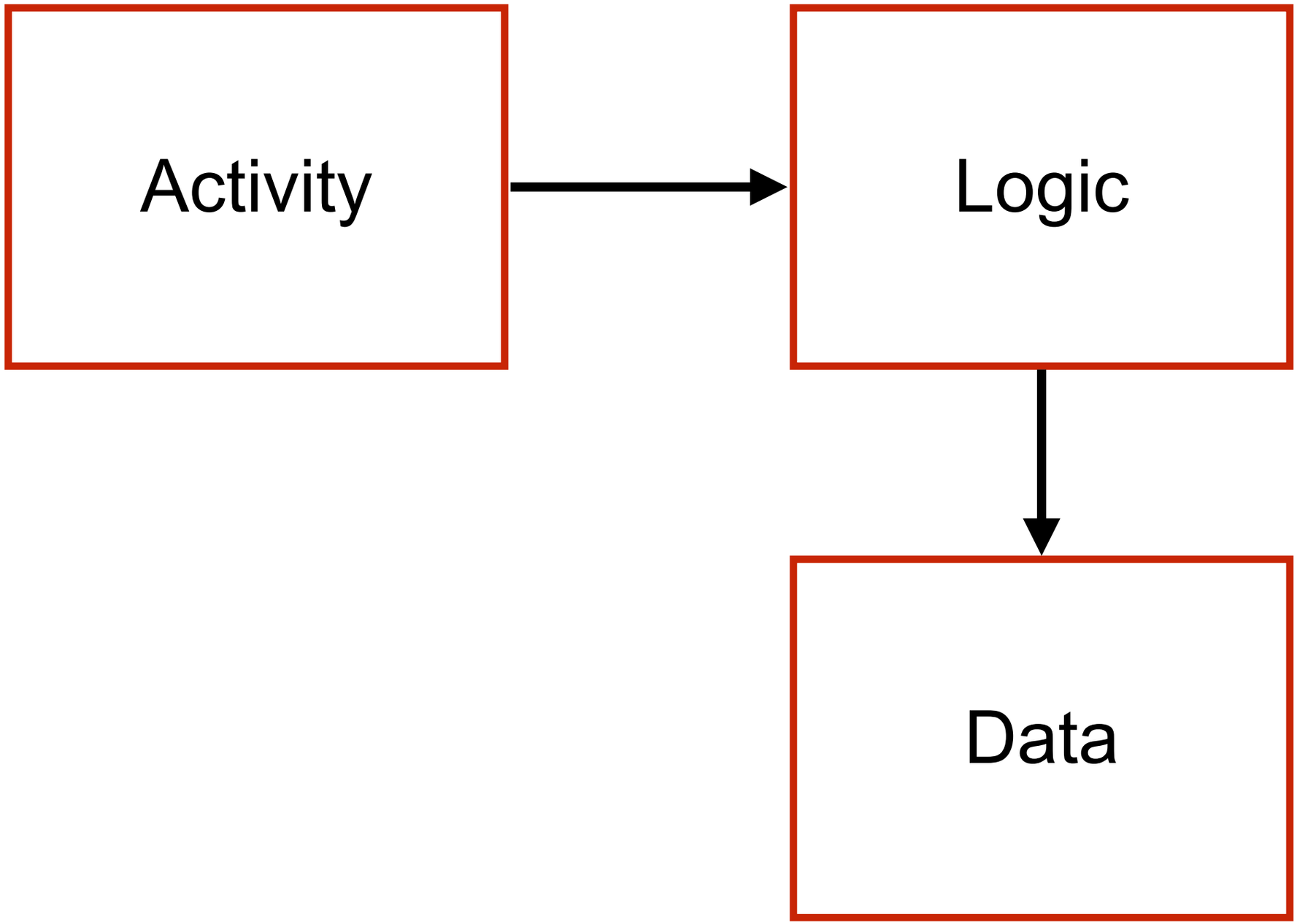


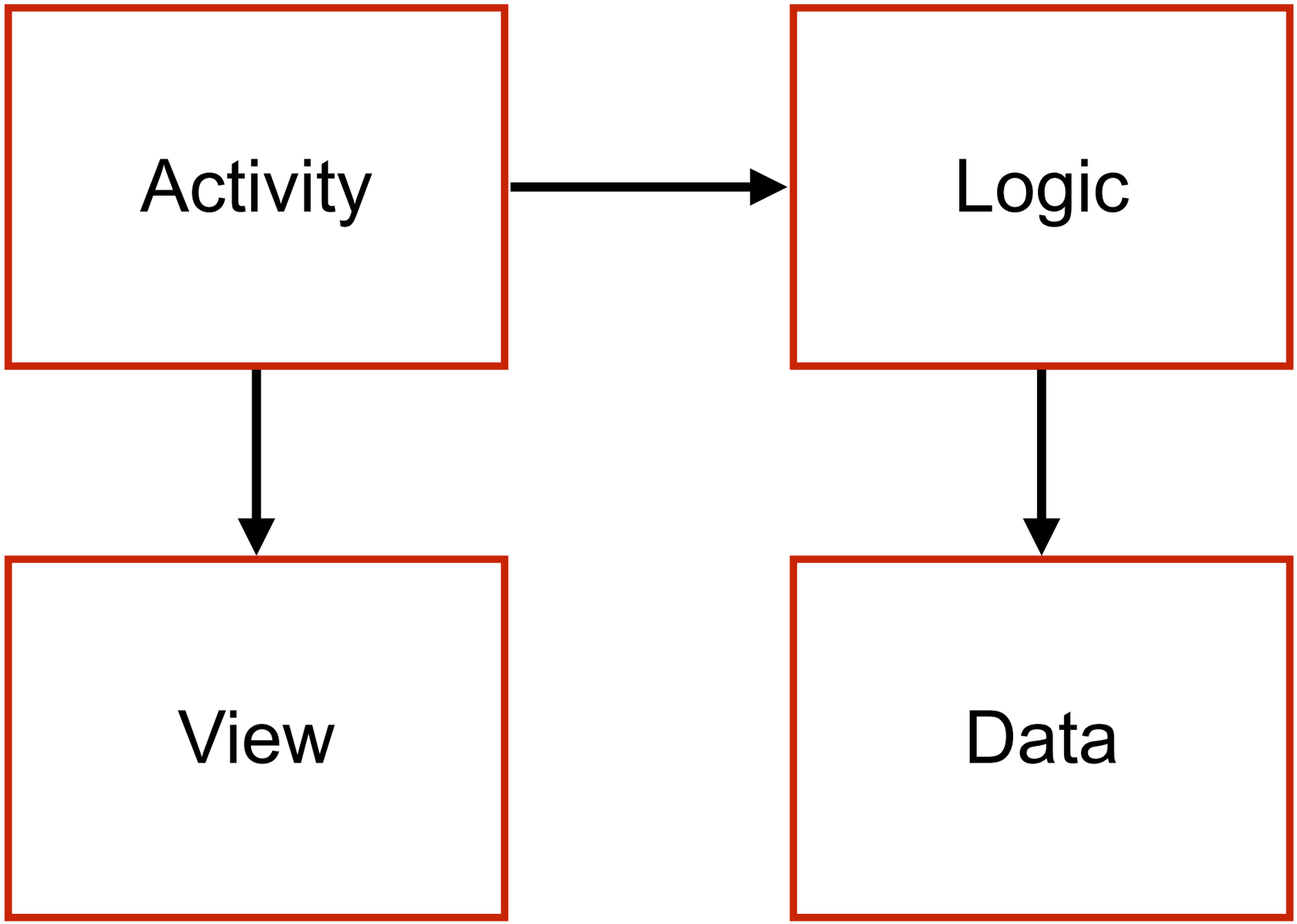
https://en.wikipedia.org/wiki/SOLID



Better way ?

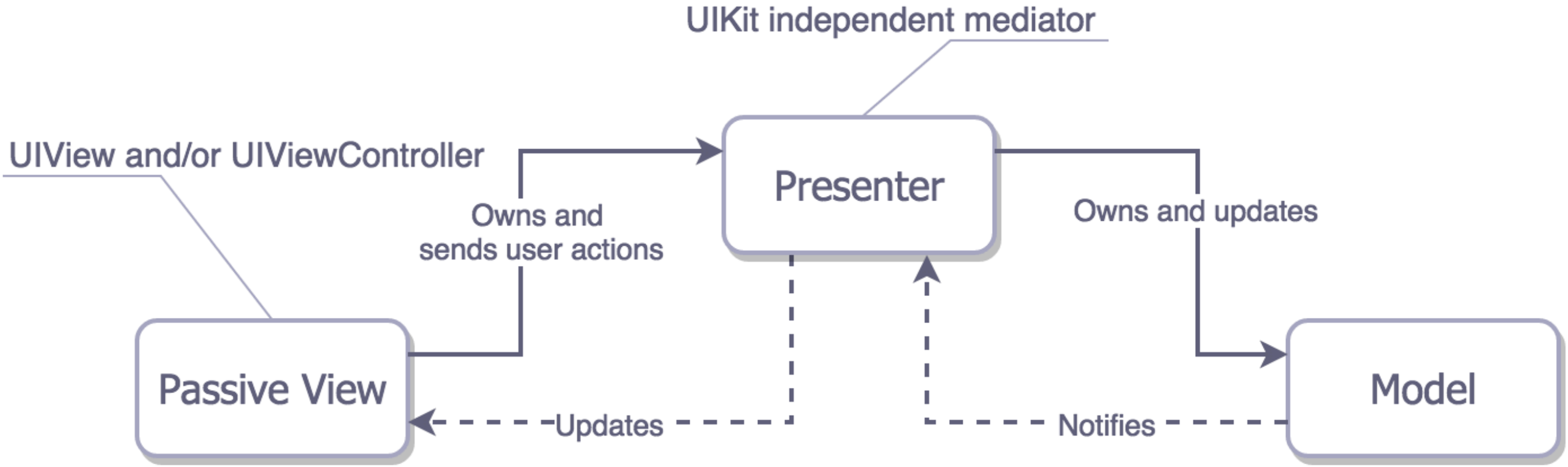




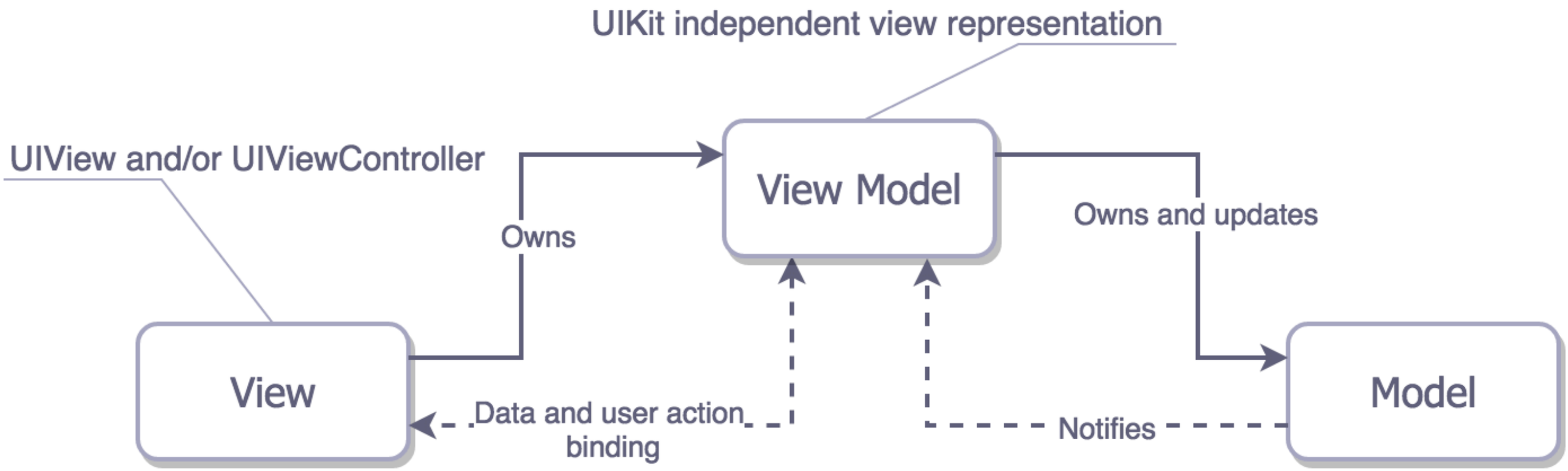


More ...

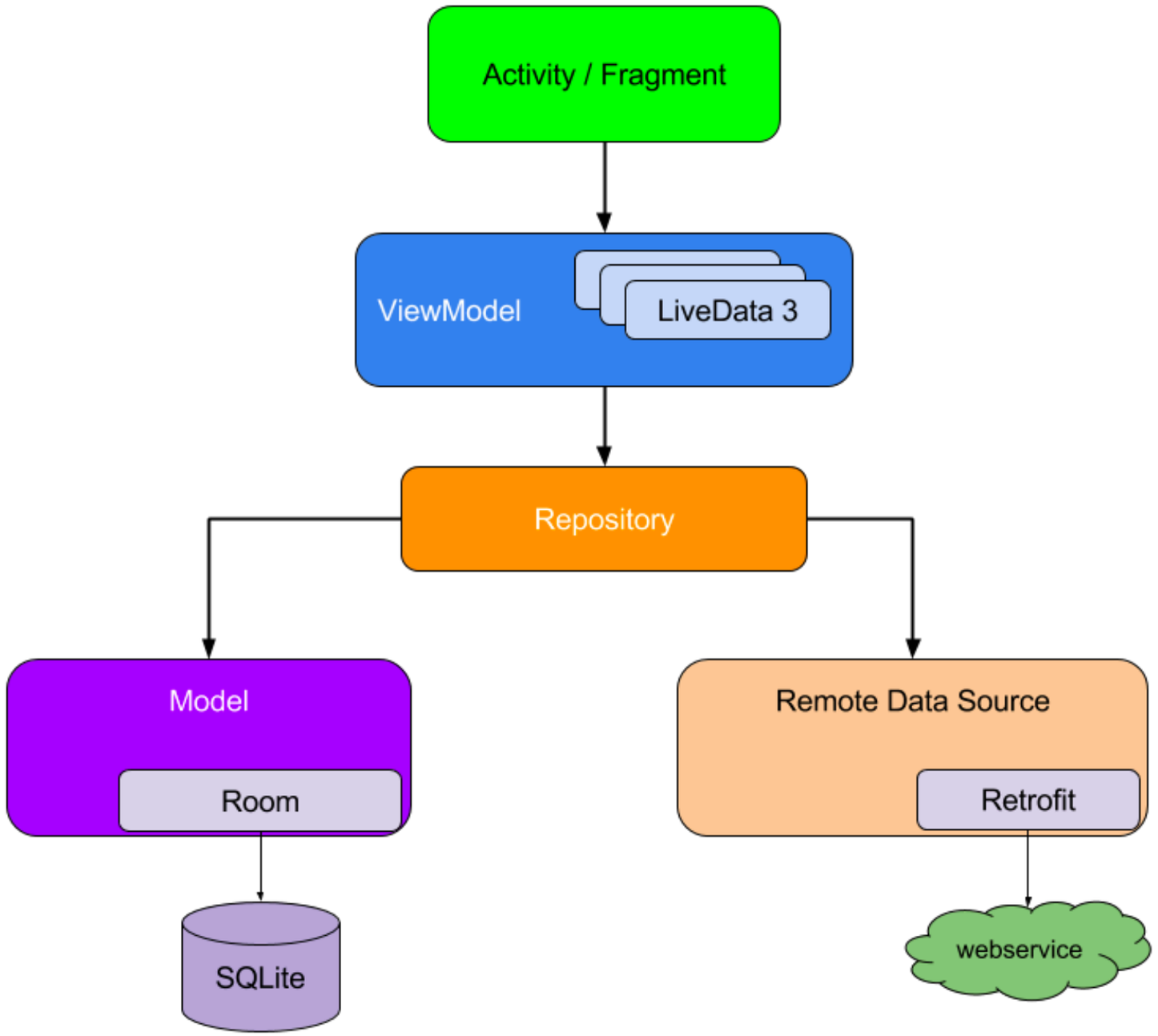
MVP



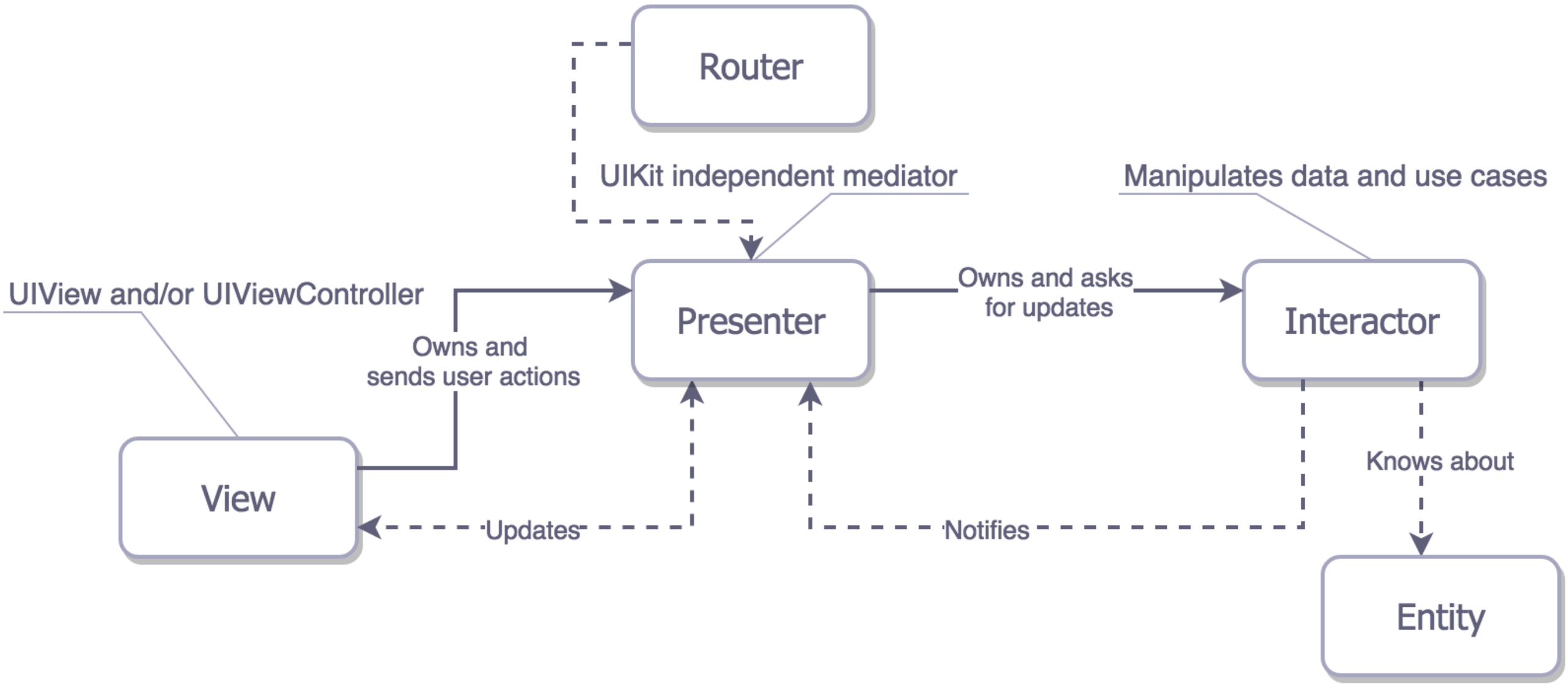
MVVM



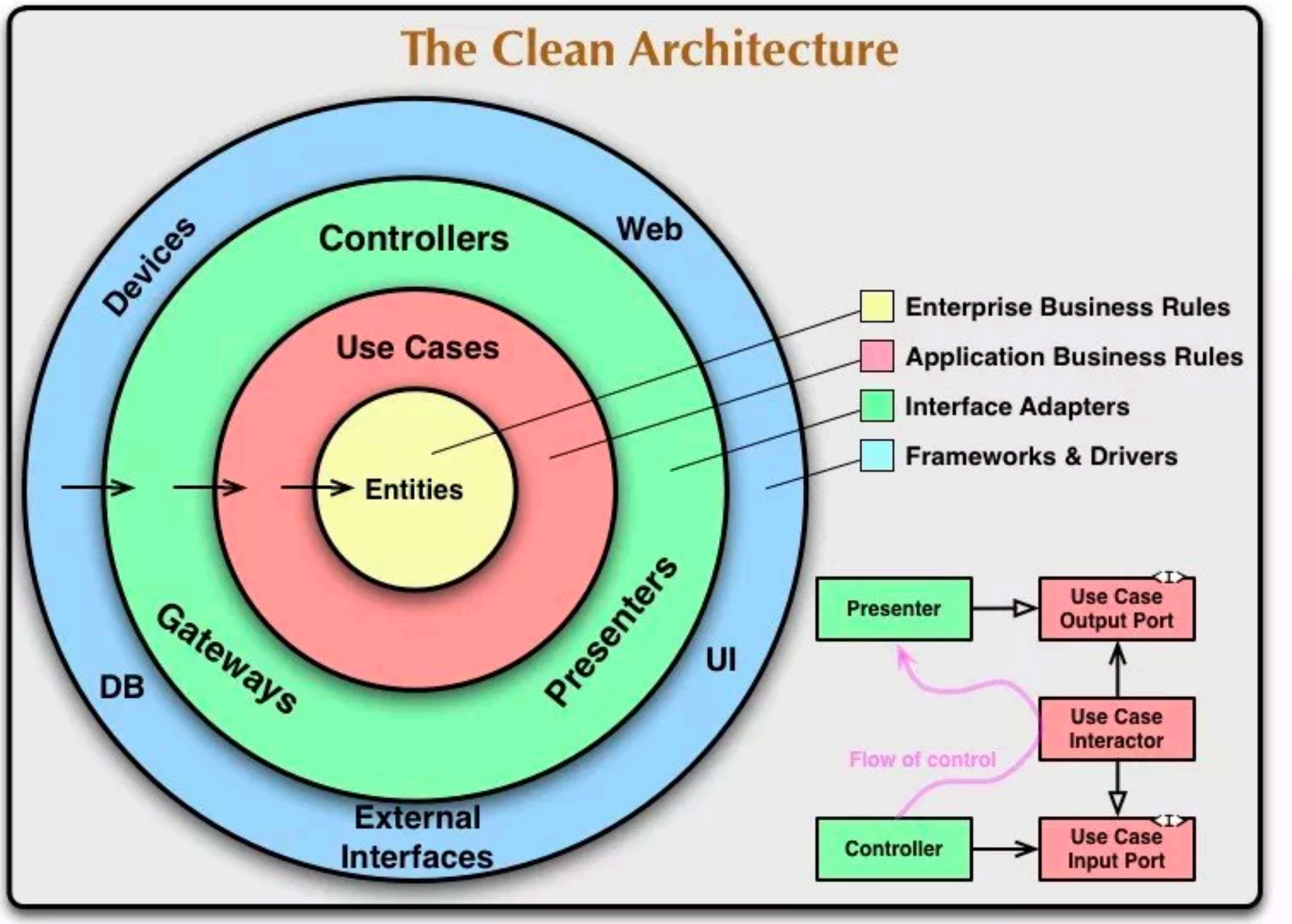
Android Architecture Component



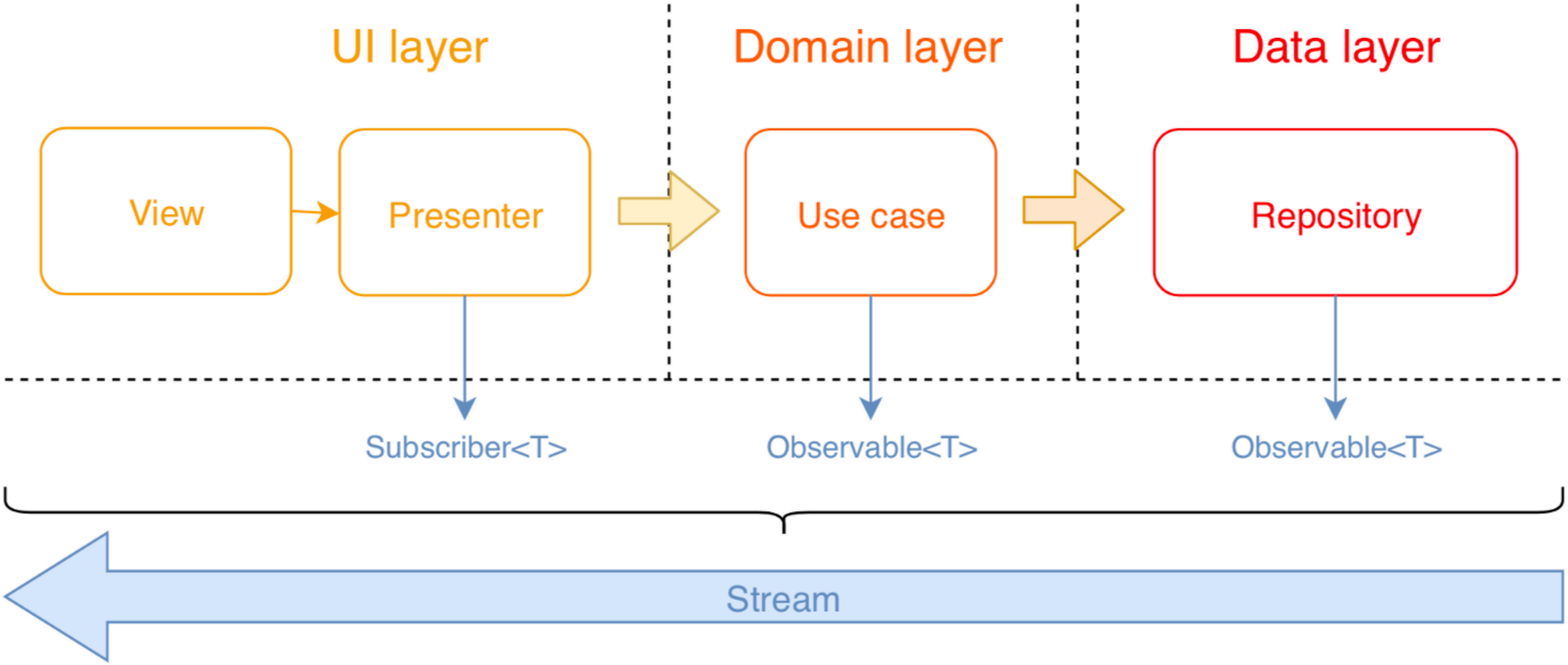
VIPER



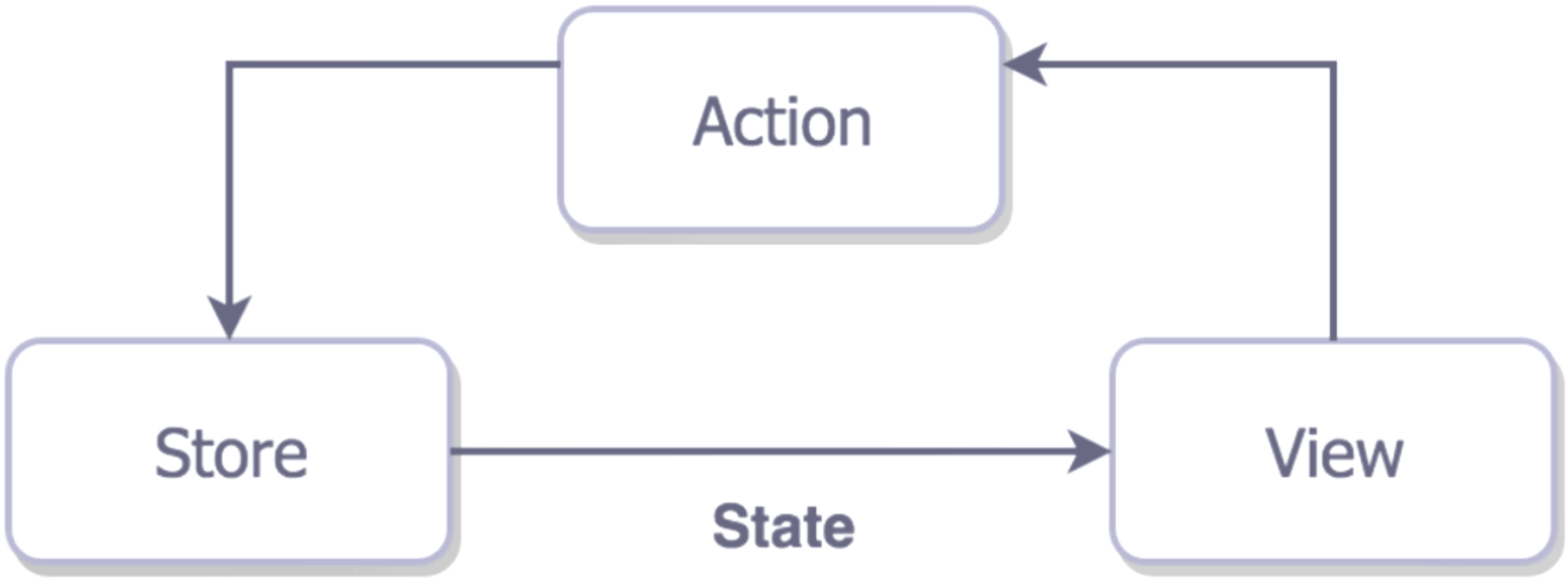
Clean Architecture



Clean Architecture



Redux-Like



No silver bullet

Working with

https://fastlane.tools/

Fastlane for Android

Screengrab
Supply

https://docs.fastlane.tools/getting-started/android/setup/

Capture screenshot with Screengrab

https://docs.fastlane.tools/getting-started/android/screenshots/

1. Install fastlane

$xcode-select --install
$sudo gem install fastlane -NV

https://fastlane.tools/

Screengrab

Automated localized **screenshots** of your Android app on every device

Working with UI testing (Espresso)

https://docs.fastlane.tools/actions/screengrab/

Screengrab

$fastlane screengrab init

```
[✔] 🚀
[22:08:07]: Successfully created new Screengrabfile at './Screengrabfile'
```

2. Add dependency

In file /app/build.gradle

```
dependencies {

    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1
    androidTestImplementation 'com.android.support.test:rules:1.
    androidTestImplementation 'com.android.support.test.espresso

    androidTestImplementation 'com.agoda.kakao:kakao:1.3.0'

    androidTestImplementation 'tools.fastlane:screengrab:1.0.0'
}
```

3. Create file AndroidManifest.xml

In folder **/app/src/debug/**

```
<!-- Allows unlocking your device and activating its screen so UI tests can succeed -->
<uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>

<!-- Allows for storing and retrieving screenshots -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

<!-- Allows changing locales -->
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION"
    tools:ignore="ProtectedPermissions" />
```

4. Add screengrab to UI test

Try to change Locale on device

```
public class LoginSuccessUIWithScreengrabTest {

    @ClassRule
    public static final LocaleTestRule localeTestRule
            = new LocaleTestRule();

    @BeforeClass
    public static void init() {
        Screengrab.setDefaultScreenshotStrategy(
                new UiAutomatorScreenshotStrategy());
    }
```

4. Add screengrab to UI test

Config strategy of UI Automator

```
public class LoginSuccessUIWithScreengrabTest {

    @ClassRule
    public static final LocaleTestRule localeTestRule
            = new LocaleTestRule();

    @BeforeClass
    public static void init() {
        Screengrab.setDefaultScreenshotStrategy(
                new UiAutomatorScreenshotStrategy());
    }
```

4. Add screengrab to UI test

```
@Test
public void login_succeed() {

    activityRule.launchActivity(new Intent());

    Screengrab.screenshot("step_01");

    // 2. Act (When)
    onView(withId(R.id.et_username))
            .perform(replaceText("admin"), closeSoftKeyboard());
    onView(withId(R.id.et_passsword))
            .perform(replaceText("password"), closeSoftKeyboard());

    Screengrab.screenshot("step_02");

    onView(withId(R.id.btn_signin)).perform(click());

    // 3. Assert (Then)
    onView(withId(R.id.tv_result)).check(matches(withText("Success!!!")));
    Screengrab.screenshot("step_03");

}
```

5. Try to config screngrabfile

$fastlane screengrab init

```
app_package_name('com.example.myapplication')

app_apk_path('app/build/outputs/apk/debug/app-debug.apk')
tests_apk_path('app/build/outputs/apk/AndroidTest/debug/ap

locales(['en-US'])

clear_previous_screenshots(true)
```

6. Run

$gradlew assembleDebug assembleAndroidTest

$fastlane screengrab

```
[✔] 🚀
[07:19:15]: Get started using a Gemfile for fastlane https:/
ls/getting-started/ios/setup/#use-a-gemfile
[07:19:15]: To not be asked about this value, you can specif
ckage_name'
[07:19:15]: The package name of the app under test (e.g.
com.yourcompany.yourapp):
```

7. See result

In folder **/fastlane**