android Testing

Krzysztof Marczewski

Dlaczego nie piszemy testów,

i co z tym możemy zrobić?

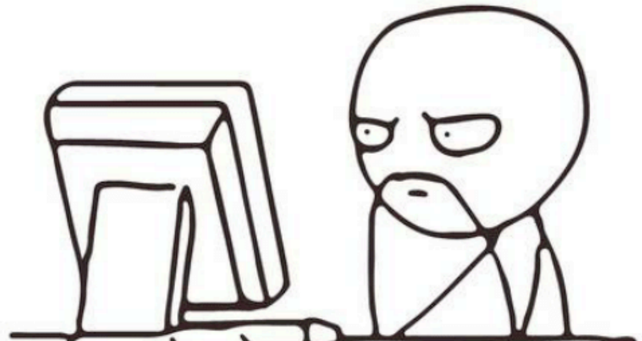
PISANIE TESTÓW

vs

klient, manager i czas

brak testów
w napotkanym projekcie

1 nie wiem jakie testy pisać



2

nie wiem co testować

ani od czego zacząć?

3

nie wiem jak to wszystko testować

klasy robią wszystko i nic

5

architektura nie wspiera testowania

6

nie znam narzędzi i bibliotek

7

nie wiem czego nie wiem

8

od czego zacząć naukę?

9

po co to wszystko?

?

co zrobić z projektem, który nie posiada unit testów?

CZAS vs MANAGER vs KLIENT


PISANIE
TESTÓW
DEV
KLIENT
CZAS
MANAGER
imgflip.com


MANAGER
CZAS
KLIENT
imgflip.com

```kotlin
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
    }
}


@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.selfformat.app", appContext.packageName)
    }
}
```


ALWAYS HOPE, THERE IS
imgflip.com

rozmawiać i tłumaczyć

co zrobić z projektem, który nie posiada unit testów?

pisać testy na bieżąco

Zacznij od dodania jednego unit testu
do następnej dodanej klasy

testy są częścią developmentu

co zrobić z projektem, który nie posiada unit testów?

pull request template

po prostu przypominacz dręczący sumienie

co zrobić z projektem, który nie posiada unit testów?

CI/Project Rules

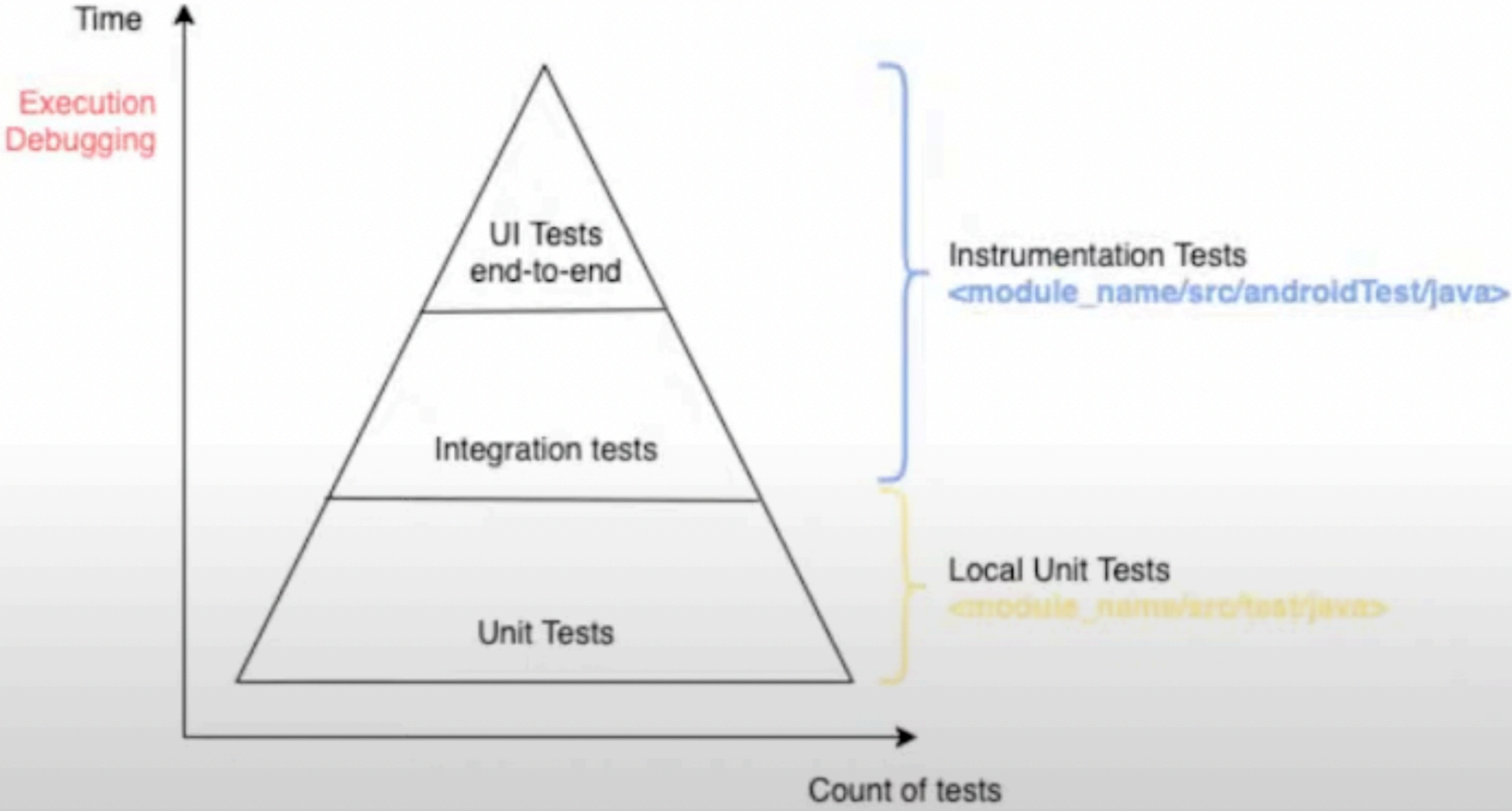
Nie rób wszystkiego samodzielnie

UCZ INNYCH

lub wyślij im te slajdy

podstawy, czyli

… jakie te testy?


Time
Execution
Debugging
UI Tests
end-to-end
Integration tests
Unit Tests
Instrumentation Tests
<module_name/src/androidTest/java>
Local Unit Tests
<module_name/src/test/java>
Count of tests

local tests

instrumented test

testImplementation

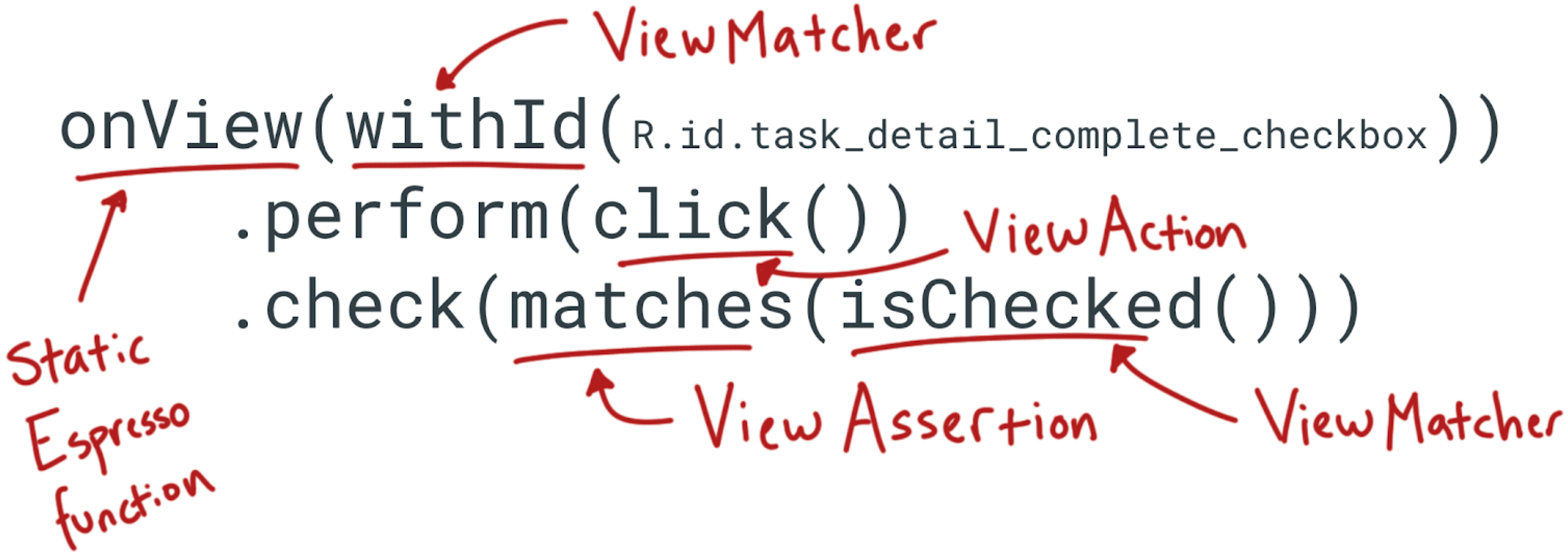
adds dependencies for local test

androidTestImplementation

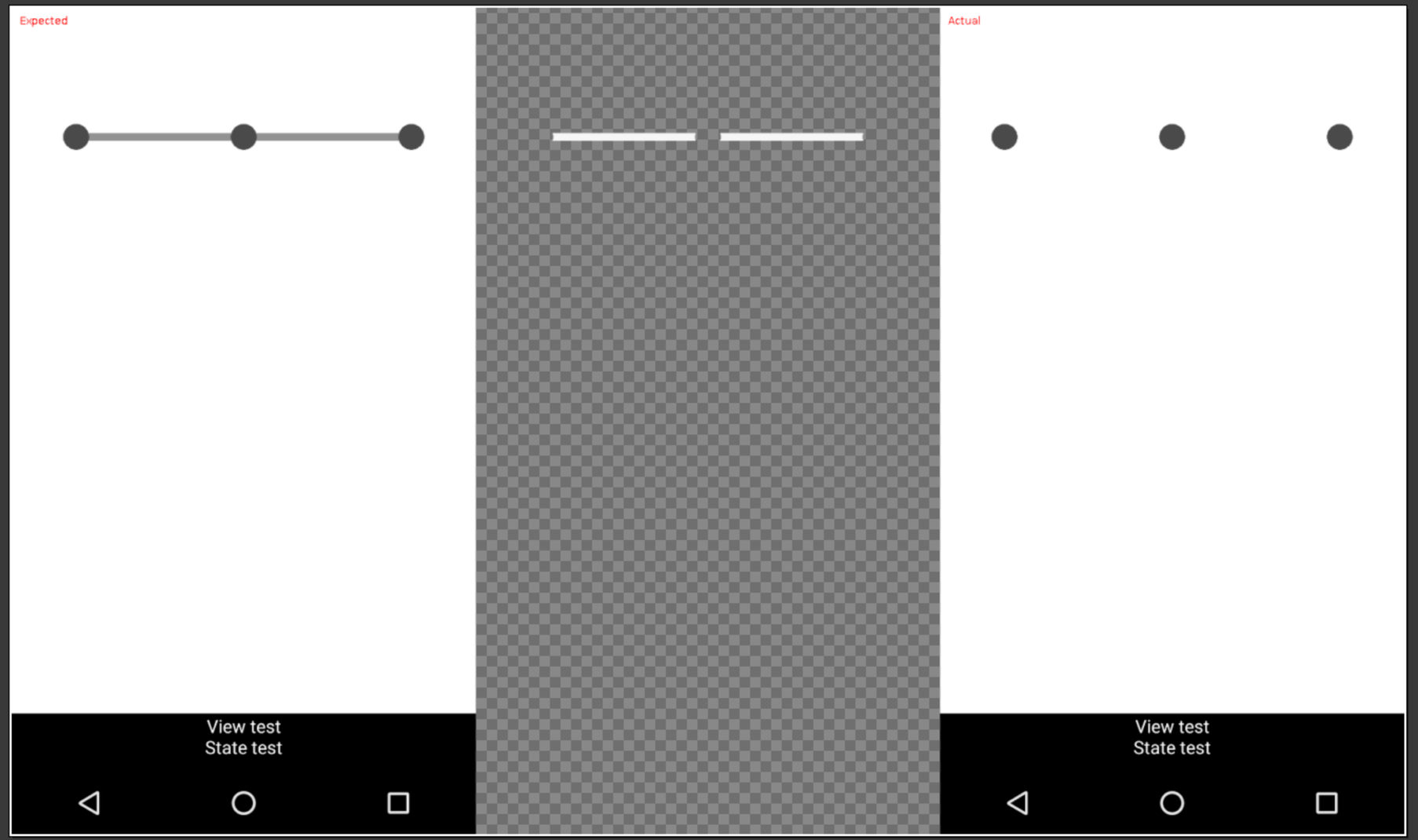
adds dependencies for Instrumented tests

UI (matchery)

... jakie te testy?



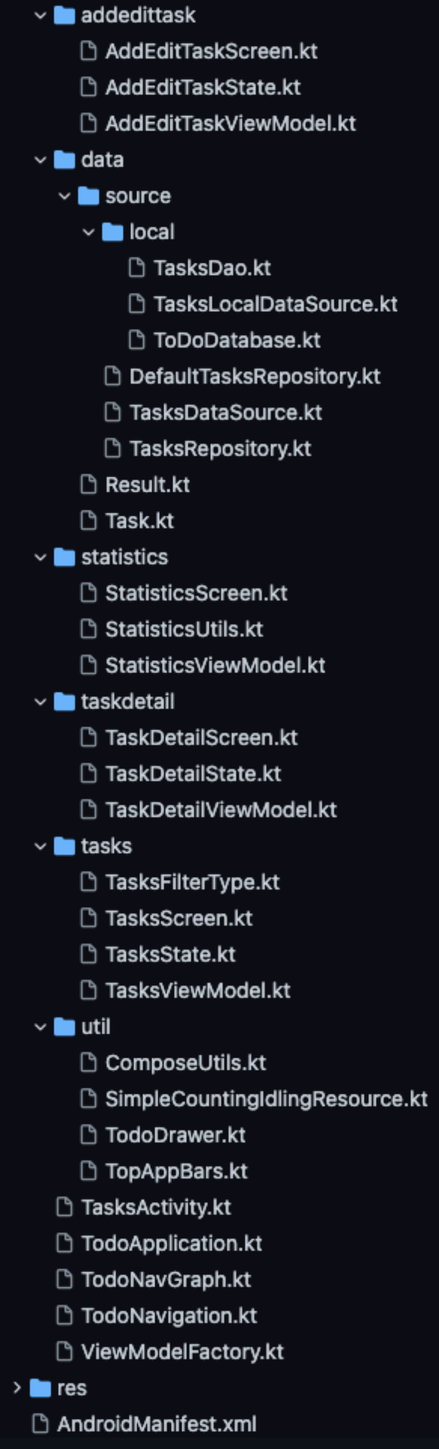
screenshot test (UI)



source

2

co testować?


addedittask
AddEditTaskScreen.kt
AddEditTaskState.kt
AddEditTaskViewModel.kt
data
source
local
TasksDao.kt
TasksLocalDataSource.kt
ToDoDatabase.kt
DefaultTasksRepository.kt
TasksDataSource.kt
TasksRepository.kt
Result.kt
Task.kt
statistics
StatisticsScreen.kt
StatisticsUtils.kt
StatisticsViewModel.kt
taskdetail
TaskDetailScreen.kt
TaskDetailState.kt
TaskDetailViewModel.kt
tasks
TasksFilterType.kt
TasksScreen.kt
TasksState.kt
TasksViewModel.kt
util
ComposeUtils.kt
SimpleCountingIdlingResource.kt
TodoDrawer.kt
TopAppBars.kt
TasksActivity.kt
TodoApplication.kt
TodoNavGraph.kt
TodoNavigation.kt
ViewModelFactory.kt
res
AndroidManifest.xml

najwyższa wartość

logika biznesowa

co testować?

view models • usecases
mappers • repositories • services
calculations • api calls

czytelne

co testować?

czytelne → nieczytelne

miejsca podatne na błędy

miejsca zwracające błędy

happy path

happy path → edge case'y

nie testuj wewnętrznego działania bibliotek

3

w jaki sposób to wszystko testować?

BEFORE - AFTER

test fixtures

w jaki sposób to wszystko testować? - test fixtures

```
@Before
fun setUp() {
    Dispatchers.setMain(testCoroutineDispatcher)
}

@After
fun cleanup() {
    Dispatchers.resetMain()
    testCoroutineDispatcher.cleanupTestCoroutines()
}
```

source

w jaki sposób to wszystko testować?

BEFORE - AFTER

rules

w jaki sposób to wszystko testować? - rules

```
/**
 * A JUnit [TestRule] that sets the Main dispatcher to [testDispatcher]
 * for the duration of the test.
 */
class MainDispatcherRule(
    val testDispatcher: TestDispatcher = UnconfinedTestDispatcher(),
) : TestWatcher() {
    override fun starting(description: Description) {
        Dispatchers.setMain(testDispatcher)
    }

    override fun finished(description: Description) {
        Dispatchers.resetMain()
    }
}
```

Before

After

source

w jaki sposób to wszystko testować? - rules

```
class ForYouViewModelTest {
    @get:Rule
    val mainDispatcherRule = MainDispatcherRule()

    // code
}
```

source

nazewnictwo i struktura

w jaki sposób to wszystko testować?

nazewnictwo i struktura

given - when - then

w jaki sposób to wszystko testować?

nazewnictwo i struktura

given - when - then

lub

arrange - act - assert

arrange act assert

```
@Test
fun `when movie was deleted then usecase was triggered`() = runTest {
    // Arrange
    val movie = Movie("Titanic")
    val viewModel = FavoritesViewModel(fakeDeleteUsecase)

    // Act
    viewModel.delete(movie)

    // Assert
    verify(fakeDeleteUsecase, times(1)).invoke(movie)
}
```

given, when, then

```
@Test
fun `should return empty state`() {

        // given
        val selectedContacts = listOf(contactsMapper.toDomain(fakeContact1))

        // when
        viewModel.deleteSelectedContacts(contacts)

        // then
        assertEquals(viewModel.contactsUiState.state, ContacstState.EMPTY)
}
```

given, when, then

```
@Test
fun `should return empty state`() {

        // given
        val selectedContacts = listOf(contactsMapper.toDomain(fakeContact1))

        // when
        viewModel.deleteSelectedContacts(contacts)

        // then
        assertEquals(viewModel.contactsUiState.state, ContacstState.EMPTY)
}
```

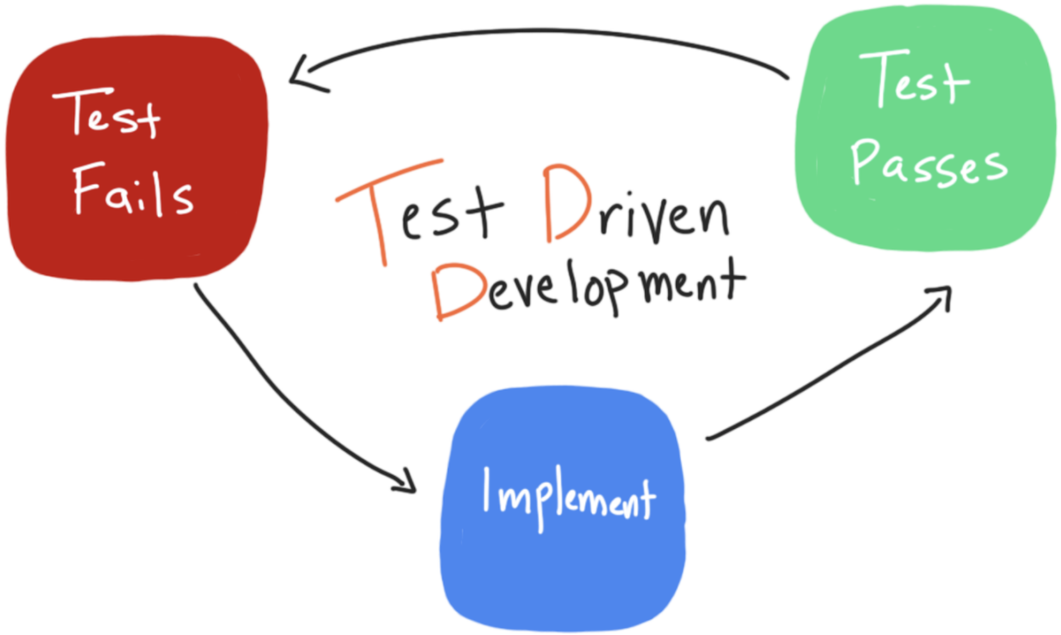
given, when, then

```
@Test
fun `given contact list, when all contacts were deleted, then show empty list
state`() {

        // given
        val selectedContacts = listOf(contactsMapper.toDomain(fakeContact1))

        // when
        viewModel.deleteSelectedContacts(contacts)

        // then
        assertEquals(viewModel.contactsState, ContacstState.EMPTY)
}
```

w jaki sposób to wszystko testować?

TDD

w jaki sposób to wszystko testować?



Black Box Testing

White Box Testing

testy parametryzowane

w jaki sposób to wszystko testować? - testy parametryzowane

```
internal class PasswordValidatorTest {

    private val validator = PasswordValidator()

    @ParameterizedTest(name = "given \"{0}\", when validating the password, then it should return {1}")
    @MethodSource("passwordArguments")
    fun `given input password, when validating it, then is should return if it is valid`(
        password: String,
        expected: Boolean
    ) {
        val actual = validator.isValid(password)
       assertThat(actual).isEqualTo(expected)
    }

    private companion object {
        @JvmStatic
        fun passwordArguments() = Stream.of(
            Arguments.of("Test123!", true),
            Arguments.of("#tesT12!", true),
            Arguments.of("12Es@t123", true),
            Arguments.of("test123!", false),
            Arguments.of("t ", false),
            Arguments.of("    ", false)
        )
    }
}
```

source

w jaki sposób to wszystko testować? - testy parametryzowane

```
internal class PasswordValidatorTest {

    private val validator = PasswordValidator()

    @ParameterizedTest(name = "given \"{0}\", when validating the password, then it should return {1}")
    @MethodSource("passwordArguments")
    fun `given input password, when validating it, then is should return if it is valid`(
        password: String,
        expected: Boolean
    ) {
        val actual = validator.isValid(password)
       assertThat(actual).isEqualTo(expected)
    }

    private companion object {
        @JvmStatic
        fun passwordArguments() = Stream.of(
            Arguments.of("Test123!", true),
            Arguments.of("#tesT12!", true),
            Arguments.of("12Es@t123", true),
            Arguments.of("test123!", false),
            Arguments.of("t ", false),
            Arguments.of("   ", false)
        )
    }
}
```

source

w jaki sposób to wszystko testować? - testy parametryzowane

```
internal class PasswordValidatorTest {

    private val validator = PasswordValidator()

    @TestFactory
    fun `given input password, when validating it, then is should return if it is valid`() =
        listOf(
            "Test123!" to true,
            "#tesT12!" to true,
            "12Es@t123" to true,
            "test123!" to false,
            "t " to false,
            "    " to false
        ).map { (password, expected) ->
            dynamicTest(
                "given \"$password\", " +
                    "when validating the password, " +
                    "then it should be reported as ${if (expected) "valid" else "invalid"}"
            ) {
                val actual = validator.isValid(password)
                assertThat(actual).isEqualTo(expected)
            }
        }
}
```

source

uwaga na FLAKY TEST

Test Doubles

w jaki sposób to wszystko testować?

Test Doubles

STUB - MOCK - SPY - FAKE - DUMMY

w jaki sposób to wszystko testować?

STUB

w jaki sposób to wszystko testować?

Mock

Mock example with MockK

```
val car = mockkClass(Car::class)

every { car.drive(Direction.NORTH) } returns Outcome.OK

car.drive(Direction.NORTH) // returns OK

verify { car.drive(Direction.NORTH) }
```

source

w jaki sposób to wszystko testować?

Spy

Spy example with MockK

```
val car = spyk(Car()) // or spyk<Car>() to call the default constructor

car.drive(Direction.NORTH) // returns whatever the real function of Car returns

verify { car.drive(Direction.NORTH) }

confirmVerified(car)
```

source

w jaki sposób to wszystko testować?

Fake

Fake

```
private val userDataRepository = TestUserDataRepository()
private val authorsRepository = TestAuthorsRepository()
private val topicsRepository = TestTopicsRepository()
private val newsRepository = TestNewsRepository()
private lateinit var viewModel: ForYouViewModel

@Before
fun setup() {
    viewModel = ForYouViewModel(
        userDataRepository = userDataRepository,
        authorsRepository = authorsRepository,
        topicsRepository = topicsRepository,
        newsRepository = newsRepository,
    )
}
```

source

Fake

```
private val userDataRepository = TestUserDataRepository()
private val authorsRepository = TestAuthorsRepository()
private val topicsRepository = TestTopicsRepository()
private val newsRepository = TestNewsRepository()
private lateinit var viewModel: ForYouViewModel

@Before
fun setup() {
    viewModel = ForYouViewModel(
        userDataRepository = userDataRepository,
        authorsRepository = authorsRepository,
        topicsRepository = topicsRepository,
        newsRepository = newsRepository,
    )
}
```

source

Fake

```
class TestAuthorsRepository : AuthorsRepository {

    private val authorsFlow: MutableSharedFlow<List<Author>> =
        MutableSharedFlow(replay = 1, onBufferOverflow =
BufferOverflow.DROP_OLDEST)

    override fun getAuthorsStream(): Flow<List<Author>> = authorsFlow

    override fun getAuthorStream(id: String): Flow<Author> {
        return authorsFlow.map { authors -> authors.find { it.id == id }!! }
    }

    fun sendAuthors(authors: List<Author>) {
        authorsFlow.tryEmit(authors)
    }
}
```

source

Fake

```
class OfflineFirstAuthorsRepository @Inject constructor(
    private val authorDao: AuthorDao,
    private val network: NiaNetworkDataSource,
) : AuthorsRepository {

    override fun getAuthorStream(id: String): Flow<Author> =
        authorDao.getAuthorEntityStream(id).map {
            it.asExternalModel()
        }

    override fun getAuthorsStream(): Flow<List<Author>> =
        authorDao.getAuthorEntitiesStream()
            .map { it.map(AuthorEntity::asExternalModel) }
}
```

source

Dummy

Dummy

```
private val userDataRepository = TestUserDataRepository()
private val authorsRepository = TestAuthorsRepository()
private val topicsRepository = TestTopicsRepository()
private val newsRepository = TestNewsRepository()
private lateinit var viewModel: ForYouViewModel

@Before
fun setup() {
    viewModel = ForYouViewModel(
        userDataRepository = userDataRepository,
        authorsRepository = authorsRepository,
        topicsRepository = topicsRepository,
        newsRepository = newsRepository,
        onClickListener = { },
    )
}
```

source

Dummy

```
private val userDataRepository = TestUserDataRepository()
private val authorsRepository = TestAuthorsRepository()
private val topicsRepository = TestTopicsRepository()
private val newsRepository = TestNewsRepository()
private lateinit var viewModel: ForYouViewModel

@Before
fun setup() {
    viewModel = ForYouViewModel(
        userDataRepository = userDataRepository,
        authorsRepository = authorsRepository,
        topicsRepository = topicsRepository,
        newsRepository = newsRepository,
        onClickListener = { },
    )
}
```

source

4

co z tymi monolitami?

```
import android.view.ViewAnimationUtils;
import android.view.ViewGroup;
import android.view.ViewTreeObserver;
import android.view.Window;
import android.view.WindowManager;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.arch.core.util.Function;
import androidx.core.content.pm.ShortcutInfoCompat;
import androidx.core.content.pm.ShortcutManagerCompat;
import androidx.core.graphics.ColorUtils;
import androidx.recyclerview.widget.ItemTouchHelper;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.google.android.gms.common.api.Status;
import com.google.firebase.appindexing.Action;
import com.google.firebase.appindexing.FirebaseUserActions;
import com.google.firebase.appindexing.builders.AssistActionBuilder;

import org.telegram.PhoneFormat.PhoneFormat;
import org.telegram.messenger.AccountInstance;
import org.telegram.messenger.AndroidUtilities;
import org.telegram.messenger.ApplicationLoader;
import org.telegram.messenger.BuildVars;
import org.telegram.messenger.ChatObject;
import org.telegram.messenger.ContactsController;
import org.telegram.messenger.ContactsLoadingObserver;
import org.telegram.messenger.DialogObject;
import org.telegram.messenger.FileLoader;
import org.telegram.messenger.FileLog;
```

refaktoryzacja

refaktoryzacja

refactoring.guru

modularyzacja

modularyzacja

developer.android.com/topic/modularization

architektura

architektura

developer.android.com/topic/architecture

5

jaka ta architektura?

architektura
która WSPIERA
podmiany
zależności

jaka ta architektura?

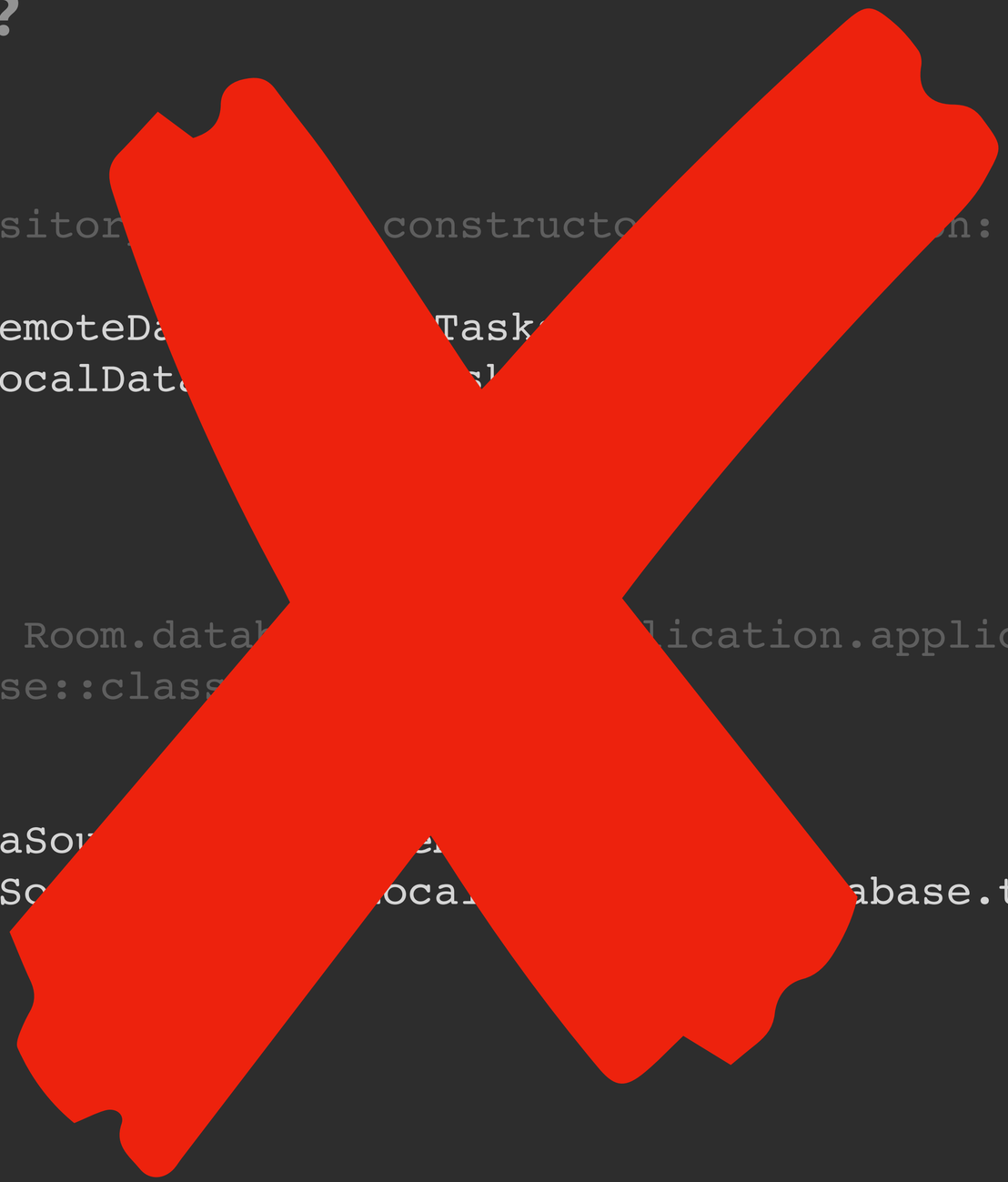
```
class DefaultTasksRepository private constructor(application: Application) {

    private val tasksRemoteDataSource: TasksDataSource
    private val tasksLocalDataSource: TasksDataSource

   // Some other code

    init {
        val database = Room.databaseBuilder(application.applicationContext,
            ToDoDatabase::class.java, "Tasks.db")
            .build()

        tasksRemoteDataSource = TasksRemoteDataSource
        tasksLocalDataSource = TasksLocalDataSource(database.taskDao())
    }
    // Rest of class
}
```

jaka ta architektura?

```
class DefaultTasksRepository private constructor(application: Application) {

    private val tasksRemoteDataSource: TasksDataSource
    private val tasksLocalDataSource: TasksDataSource

   // Some other code

    init {
        val database = Room.databaseBuilder(application.applicationContext,
            ToDoDatabase::class.java, "Tasks.db")
            .build()

        tasksRemoteDataSource = TasksRemoteDataSource
        tasksLocalDataSource = TasksLocalDataSource(database.taskDao())
    }
    // Rest of class
}
```

jaka ta architektura?

```
class DefaultTasksRepositor        constructo        n: Application) {

    private val tasksRemoteDa              Task
    private val tasksLocalData             l

   // Some other code

    init {
        val database = Room.datab                lication.applicationContext,
            ToDoDatabase::class
            .build()

        tasksRemoteDataSou                 e
        tasksLocalDataSo                   ocal                abase.taskDao())
    }
    // Rest of class
}
```



6 narzędzia

testing frameworks

jUnit / spek / kotest

jUnit

```
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
    }
}


@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.selfformat.app", appContext.packageName)
    }
}
```

Spek

```
class CalculatorTest : Spek({
    given("A calculator") {
        val calculator = Calculator()
        on("Adding 3 and 5") {
            val result = calculator.add(3, 5)
            it("Produces 8") {
                assertEquals(8, result)
            }
        }
    }
})
```

source

Kotest

```
class MyTests : StringSpec({
    "length should return size of string" {
        "hello".length shouldBe 5
    }
    "startsWith should test for a prefix" {
        "world" should startWith("wor")
    }
})
```

source

mocking

MockK / Mockito

mocking

Mockito

```
val mockedFile = mock(File::class.java)
`when`(mockedFile.read()).thenReturn("hello world")
```

MockK

```
val mockedFile = mockk<File>()
every { mockedFile.read() } returns "hello world"
```

source

mock server

MockWebServer / WireMock

assertion

Truth / Hamcrest / AssertJ / Kluent

Assertions examples

jUnit

```
assertTrue(notificationText.contains("self format"));
```

Truth

```
assertThat(notificationText).contains("self format")
```

Hamcrest

```
assertThat(notificationText, containsString("self format"))
```

Kluent

```
notificationText shouldContain "self format"
```

source

jUnit error example

```
java.lang.AssertionError: expected:<[guava, dagger, truth, auto,
caliper]> but was:<[dagger, auto, caliper, guava]>
  at org.junit.Assert.failNotEquals(Assert.java:835) <2 internal calls>
  at
com.google.common.truth.example.DemoTest.testBuiltin(DemoTest.java:64)
<19 internal calls>
```

Truth error example

```
value of    : projectsByTeam().valuesForKey(corelibs)
missing (1) : truth
---
expected    : [guava, dagger, truth, auto, caliper]
but was     : [guava, auto, dagger, caliper]
multimap was: {corelibs=[guava, auto, dagger, caliper]}
  at com.google.common.truth.example.DemoTest.testTruth(DemoTest.java:71)
```

source source

code coverage

Jacoco

Test Summary

1	0	0	0.002s
tests	failures	ignored	duration

100% successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
com.example.jacocoexample	1	0	0	0.002s	100%

debugAndroidTest > com.example.jacocoexample.ui.main — Source Files Sessions

com.example.jacocoexample.ui.main

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MainFragment		0%		n/a	4	4	5	5	4	4	1	1
MainViewModel		0%		n/a	2	2	3	3	2	2	1	1
MainFragment.Companion		0%		n/a	2	2	2	2	2	2	1	1
Total	60 of 60	0%	0 of 0	n/a	8	8	10	10	8	8	3	3

Generated by the Android Gradle plugin 4.2.1 — Created with JaCoCo 0.7.9.201702052155

source

fake dependencies

Hilt

Hilt

```
@Module
@InstallIn(SingletonComponent::class)
abstract class AnalyticsModule {

    @Singleton
    @Binds
    abstract fun bindAnalyticsService(
        analyticsServiceImpl: AnalyticsServiceImpl
    ): AnalyticsService
}
```

source

Hilt

```
@Module
@TestInstallIn(
    components = [SingletonComponent::class],
    replaces = [AnalyticsModule::class]
)
abstract class FakeAnalyticsModule {

    @Singleton
    @Binds
    abstract fun bindAnalyticsService(
        fakeAnalyticsService: FakeAnalyticsService
    ): AnalyticsService
}
```

source

Hilt

```
@Module
@TestInstallIn(
    components = [SingletonComponent::class],
    replaces = [AnalyticsModule::class]
)
abstract class FakeAnalyticsModule {

    @Singleton
    @Binds
    abstract fun bindAnalyticsService(
        fakeAnalyticsService: FakeAnalyticsService
    ): AnalyticsService
}
```

source

flows

Turbine

testing flows with Turbine

```
@Test
fun usingTurbine() = runTest {
    val dataSource = FakeDataSource()
    val repository = Repository(dataSource)

    repository.scores().test {
        dataSource.emit(1)
        assertEquals(10, awaitItem())

        dataSource.emit(2)
        awaitItem() // Ignore items if needed, can also use skip(n)

        dataSource.emit(3)
        assertEquals(30, awaitItem())
    }
}
```

source

compose

testing compose

```
class MyComposeTest {

    @get:Rule
    val composeTestRule = createComposeRule()

    @Test
    fun myTest() {
        composeTestRule.setContent { // Start the app
            MyAppTheme { MainScreen(uiState = fakeUiState, /*...*/) }
        }

        composeTestRule.onNodeWithText("Continue").performClick()

        composeTestRule.onNodeWithText("Welcome").assertIsDisplayed()
    }
}
```

source

List of Android Testing libraries

7

tips and tricks

i co dalej?

Współdzielenie klas między test i androidTest

tips and tricks

```
android {
    sourceSets {
        String sharedTestDir = 'src/sharedTest/java'
        test {
            java.srcDirs += sharedTestDir
            resources.srcDirs += 'src/sharedTest/resources'
        }
        androidTest {
            java.srcDirs += sharedTestDir
        }
    }
}
```

build.gradle modułu, source

@VisbleForTesting

tips and tricks

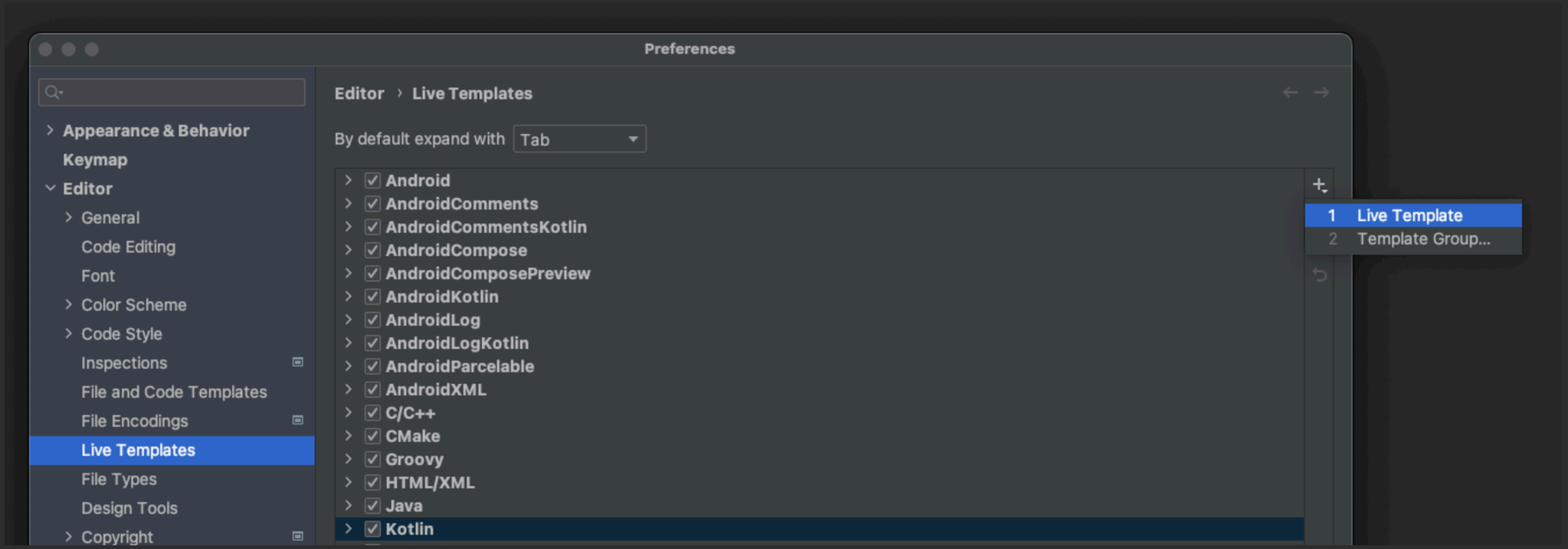
```
@VisibleForTesting(otherwise = VisibleForTesting.PROTECTED)
fun addTasks(vararg tasks: Task) { }

// If not specified, the intended visibility is assumed to be private.
```

read more

live template do pustych testów

tips and tricks

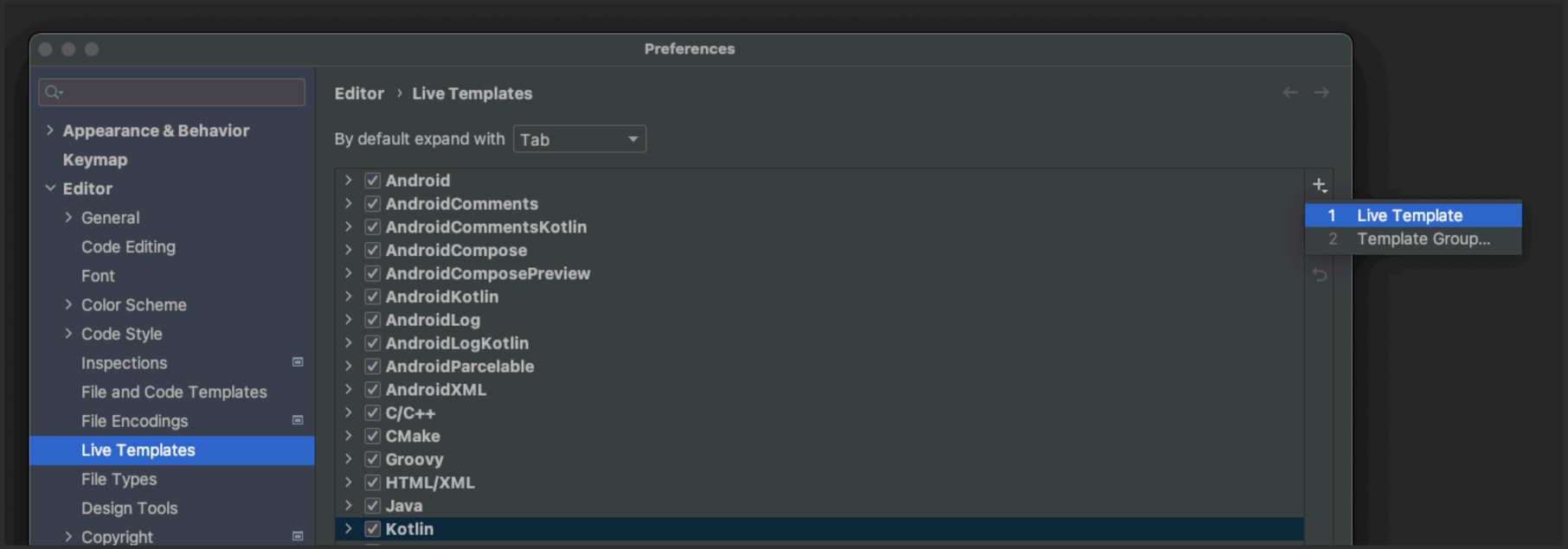
step 1



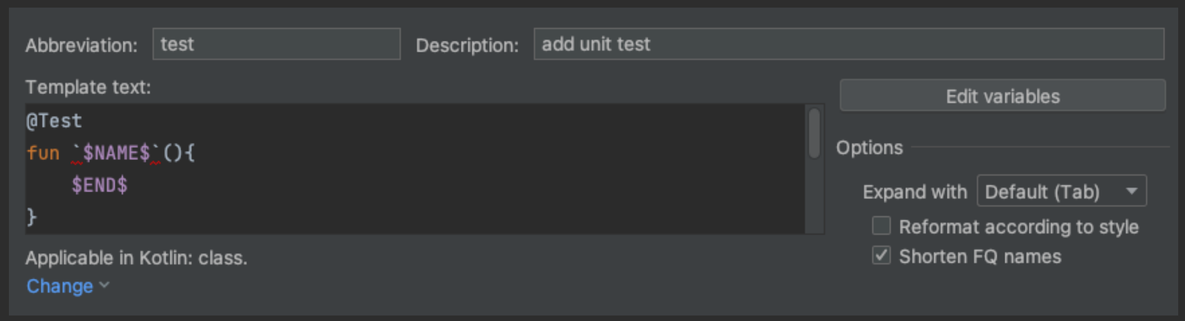
full instruction

tips and tricks

step 1



step 2



full instruction

tips and tricks

Voila

skonfiguruj CI

can't merge if tests failed

8

od czego zacząć naukę?

twórz własne projekty do testowania nowych rozwiązań

od czego zacząć naukę?

Sources & Resources

- test doubles explained
- basic unit testing codelab
- how to add fakes and refactor code without dependencies and interfaces - codelab
- coroutines testing codelab
- Android Development Blog
- Live template for unit tests
- 7 grzechów unit testowania na Androida (i nie tylko) - YouTube
- Unit testing on Android. Testing your code is crucial because of… | by Deniz Demirci | ProAndroidDev
- Understanding Unit Tests for Android in 2021 | by Christopher Elias | ProAndroidDev
- Testing in Jetpack Compose | Android Developers
- How to test Jetpack Compose 2021 | ProAndroidDev
- MockK | mocking library for Kotlin
- Full Guide to Testing Android Applications in 2022

ZACZNIJ TUTAJ

sprawdź jak robią to inni

od czego zacząć naukę?

github.com/android/architecture-samples

od czego zacząć naukę?

github.com/android/nowinandroid

```
// Local unit tests
testImplementation "androidx.test:core:1.4.0"
testImplementation "junit:junit:4.13.2"
testImplementation "androidx.arch.core:core-testing:2.1.0"
testImplementation "org.jetbrains.kotlinx:kotlinx-coroutines-test:1.6.1"
testImplementation "com.google.truth:truth:1.1.3"
testImplementation "com.squareup.okhttp3: mockwebserver:4.9.1"
testImplementation "io.mockk:mockk:1.12.0"
debugImplementation "androidx.compose.uj:ui-test-manifest:1.1.0-alpha04"


// Instrumentation tests
kaptAndroidTest "com.google. dagger:hilt-android-compiler:2.371"
androidTestImplementation 'com.google.dagger:hilt-android-testing:2.42'
androidTestImplementation "junit:junit:4.13.2"
androidTestImplementation "org.jetbrains.kotlinx:kotlinx-coroutines-test:1.6.1"
androidTestImplementation "androidx.arch.core:core-testing:2.1.0"
androidTestImplementation "com.google.truth:truth:1.1.3"
androidTestImplementation "androidx.test.ext:junit:1.1.3"
androidTestImplementation "androidx.test:core-ktx:1.4.0"
androidTestImplementation "com.squareup.okhttp3:mockwebserver:4.9.1"
androidTestImplementation "io.mockk:mockk-android:1.10.5"
androidTestImplementation "androidx.test:runner:1.4.0"
```

9

korzyści

po co mi to wszystko?

mniej błędów od grupy testującej i użytkowników

usprawnienia w architekturze

testy mogą być jak dokumentacja

testy mogą być jak dokumentacja

```
@Test
fun `given contacts list, when one user left, then show summary of todays interactions`() {}


@Test
fun `given empty contacts, when user adds more then 3 contacts, then increment productivity stat`() {}
```

source

"Best thing about writing unit tests is
it forces you to write testable code.
That way your code is cleaner,
more understandable,
more maintainable."

self. format

CONTINUOUS DEVELOPMENT

bio.link/selfformat