# Лабораторная работа №9 “ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЙ ДЛЯ ANDROID”

**Цель работы:** научиться проводить тестирование мобильных приложений на Android методом белого ящика в среде разработки Android Studio и методом черного ящика с использованием симуляторов или реальных устройств. Научиться снимать логи с Android устройств.

**Задание:**

1. Провести тестирование приложений в Android Studio методом белого ящика как показано в первом разделе.

2. Провести тестирование приложений методом черного ящика, руководствуясь стратегией тестирования, представленной выше.

3. Провести автоматизацию тестирования с использованием calabash.

4. Снять логи приложений на Android.

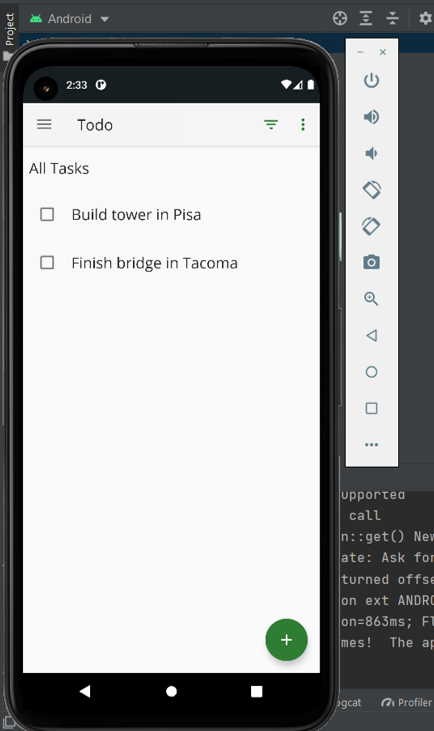


Рисунок 1 – Эмулятор устройства с запущенным приложением

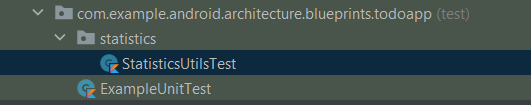


Рисунок 2 – Структура приложения

Функция для проверки:

internal fun getActiveAndCompletedStats(tasks: List<Task>?): StatsResult {

val totalTasks = tasks!!.size

val numberOfActiveTasks = tasks.count { it.isActive }

return StatsResult(

activeTasksPercent = 100f \* numberOfActiveTasks / tasks.size,

completedTasksPercent = 100f \* (totalTasks - numberOfActiveTasks) / tasks.size

)}

Тест:

package com.example.android.architecture.blueprints.todoapp.statistics

import com.example.android.architecture.blueprints.todoapp.data.Task

import org.junit.Assert.\*

import org.junit.Test

class StatisticsUtilsTest{

@Test

fun getActiveAndCompletedStats\_noCompleted\_returnsHundredZero() {

// Create an active task (the false makes this active)

val tasks = listOf<Task>(

Task("title", "desc", isCompleted = false)

)

// Call your function

val result = getActiveAndCompletedStats(tasks)

// Check the result

assertEquals(result.completedTasksPercent, 0f)

assertEquals(result.activeTasksPercent, 100f)

}}

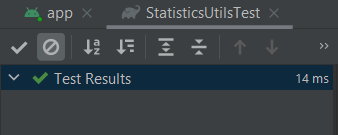


Рисунок 3 – Тест StatisticsUtilsTest пройден

Улучшим код класса функции:

internal fun getActiveAndCompletedStats(tasks: List<Task>?): StatsResult {

return if (tasks == null || tasks.isEmpty()) {

StatsResult(0f, 0f)

} else {

val totalTasks = tasks.size

val numberOfActiveTasks = tasks.count { it.isActive }

StatsResult(

activeTasksPercent = 100f \* numberOfActiveTasks / tasks.size,

completedTasksPercent = 100f \* (totalTasks - numberOfActiveTasks) / tasks.size

)}}

Напишем больше тестов:

package com.example.android.architecture.blueprints.todoapp.statistics

import com.example.android.architecture.blueprints.todoapp.data.Task

import org.hamcrest.CoreMatchers.`is`

import org.junit.Assert.\*

import org.junit.Test

class StatisticsUtilsTest {

@Test

fun getActiveAndCompletedStats\_noCompleted\_returnsHundredZero() {

val tasks = listOf(

Task("title", "desc", isCompleted = false)

)

// When the list of tasks is computed with an active task

val result = getActiveAndCompletedStats(tasks)

// Then the percentages are 100 and 0

org.hamcrest.MatcherAssert.assertThat(result.activeTasksPercent, `is`(100f))

org.hamcrest.MatcherAssert.assertThat(result.completedTasksPercent, `is`(0f))

}

@Test

fun getActiveAndCompletedStats\_noActive\_returnsZeroHundred() {

val tasks = listOf(

Task("title", "desc", isCompleted = true)

)

// When the list of tasks is computed with a completed task

val result = getActiveAndCompletedStats(tasks)

// Then the percentages are 0 and 100

assertThat(result.activeTasksPercent, `is`(0f))

assertThat(result.completedTasksPercent, `is`(100f))

}

@Test

fun getActiveAndCompletedStats\_both\_returnsFortySixty() {

// Given 3 completed tasks and 2 active tasks

val tasks = listOf(

Task("title", "desc", isCompleted = true),

Task("title", "desc", isCompleted = true),

Task("title", "desc", isCompleted = true),

Task("title", "desc", isCompleted = false),

Task("title", "desc", isCompleted = false)

)

// When the list of tasks is computed

val result = getActiveAndCompletedStats(tasks)

// Then the result is 40-60

assertThat(result.activeTasksPercent, `is`(40f))

assertThat(result.completedTasksPercent, `is`(60f))

}

@Test

fun getActiveAndCompletedStats\_error\_returnsZeros() {

// When there's an error loading stats

val result = getActiveAndCompletedStats(null)

// Both active and completed tasks are 0

org.hamcrest.MatcherAssert.assertThat(result.activeTasksPercent, `is`(0f))

org.hamcrest.MatcherAssert.assertThat(result.completedTasksPercent, `is`(0f))

}

@Test

fun getActiveAndCompletedStats\_empty\_returnsZeros() {

// When there are no tasks

val result = getActiveAndCompletedStats(emptyList())

// Both active and completed tasks are 0

org.hamcrest.MatcherAssert.assertThat(result.activeTasksPercent, `is`(0f))

org.hamcrest.MatcherAssert.assertThat(result.completedTasksPercent, `is`(0f))

}}

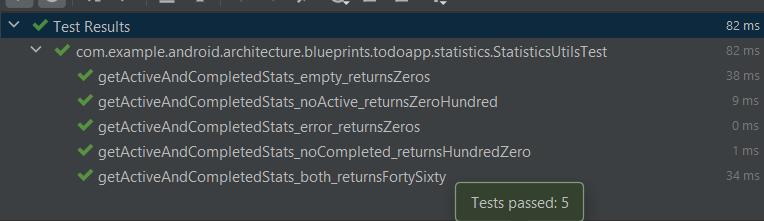


Рисунок 4 – Тест StatisticsUtilsTest пройден

Тест TasksViewModelTest:

package com.example.android.architecture.blueprints.todoapp.tasks

import androidx.test.core.app.ApplicationProvider

import androidx.test.ext.junit.runners.AndroidJUnit4

import org.junit.Test

import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)

class TasksViewModelTest {

@Test

fun addNewTask\_setsNewTaskEvent() {

// Given a fresh ViewModel

val tasksViewModel = TasksViewModel(ApplicationProvider.getApplicationContext())

// When adding a new task

tasksViewModel.addNewTask()

// Then the new task event is triggered

// TODO test LiveData}}

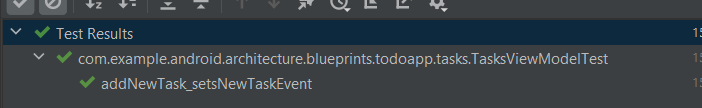


Рисунок 5 – Тест TasksViewModelTest пройден

Модифицированный тест TasksViewModelTest:

package com.example.android.architecture.blueprints.todoapp.tasks

import androidx.arch.core.executor.testing.InstantTaskExecutorRule

import androidx.test.core.app.ApplicationProvider

import androidx.test.ext.junit.runners.AndroidJUnit4

import org.hamcrest.MatcherAssert.assertThat

import org.hamcrest.Matchers.not

import org.hamcrest.Matchers.nullValue

import org.junit.Rule

import org.junit.Test

import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)

class TasksViewModelTest {

@get:Rule

var instantExecutorRule = InstantTaskExecutorRule()

@Test

fun addNewTask\_setsNewTaskEvent() {

// Given a fresh ViewModel

val tasksViewModel = TasksViewModel(ApplicationProvider.getApplicationContext())

// When adding a new task

tasksViewModel.addNewTask()

// Then the new task event is triggered

val value = tasksViewModel.newTaskEvent.getOrAwaitValue()

assertThat(value.getContentIfNotHandled(), not(nullValue()))}}

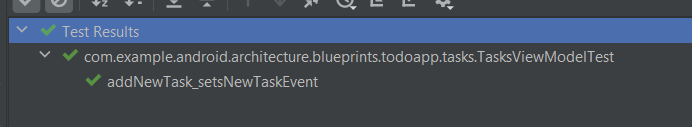


Рисунок 6 – Тест TasksViewModelTest пройден

Написание нескольких тестов ViewModel:

package com.example.android.architecture.blueprints.todoapp.tasks

import androidx.arch.core.executor.testing.InstantTaskExecutorRule

import androidx.test.core.app.ApplicationProvider

import androidx.test.ext.junit.runners.AndroidJUnit4

import org.hamcrest.MatcherAssert.assertThat

import org.hamcrest.Matchers.\*

import org.junit.Before

import org.junit.Rule

import org.junit.Test

import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)

class TasksViewModelTest {

// Subject under test

private lateinit var tasksViewModel: TasksViewModel

// Executes each task synchronously using Architecture Components.

@get:Rule

var instantExecutorRule = InstantTaskExecutorRule()

@Before

fun setupViewModel() {

tasksViewModel = TasksViewModel(ApplicationProvider.getApplicationContext())

}

@Test

fun addNewTask\_setsNewTaskEvent() {

// When adding a new task

tasksViewModel.addNewTask()

// Then the new task event is triggered

val value = tasksViewModel.newTaskEvent.getOrAwaitValue()

assertThat(

value?.getContentIfNotHandled(), (not(nullValue()))

)

@Test

fun getTasksAddViewVisible() {

// When the filter type is ALL\_TASKS

tasksViewModel.setFiltering(TasksFilterType.ALL\_TASKS)

// Then the "Add task" action is visible

assertThat(tasksViewModel.tasksAddViewVisible.getOrAwaitValue(), `is`(true))}}

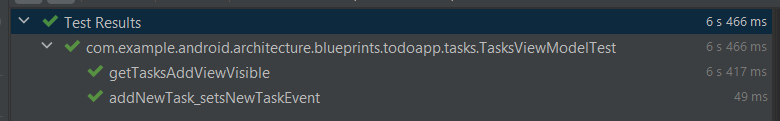


Рисунок 7 – Тесты пройдены

**Снятие логов на Android**

**Вывод:** проведено тестирование приложений в Android Studio методом белого ящика и методом черного, проведена автоматизация тестирования с использованием Calabash, сняты логи приложений на Android

**Контрольные вопросы:**

1. **Перечислите и кратко опишите основные инструменты Android Studio для тестирования.**

Некоторые полезные инструменты для тестирования, связанные с изменением вашего приложения: Appium, Robotium, Ranorex.

Полезные инструменты для тестирования локализации: Ubertesters, eggplant.

Полезные инструменты для проверки безопасности мобильных приложений: сообщества CS, OWASP Zed Attack Proxy, Veracode, Google Nogotofail и SQL Map.

Полезные инструменты для тестирования производительности мобильных приложений: neoload by Neotys, Apteligent (ранее Crittercism), New Relic.

Полезные инструменты для проверки совместимости мобильных приложений: browserstack, crossbrowsertesting от Smart Bear, Litmus, Browsera, Rational Clearcase от IBM, Ghostlab.

Полезные инструменты для тестирования интерфейса мобильных приложений: fitnesse, imacros, Jubula, loadui.

Некоторые полезные инструменты для тестирования удобства использования мобильных приложений: Zoom, Loop11. Тестирование пользовательского интерфейса.

Некоторые полезные инструменты для тестирования функциональности мобильных приложений: Appium, Selendroid, Robotium, Ranorex.

1. **В чем основное отличие симуляторов от эмуляторов?**

**Эмулятор** – это оригинальная замена устройства, а симулятор не копирует аппаратное обеспечение устройства, но у вас есть возможность настроить аналогичную среду, такую как в ОС оригинального устройства

1. **Как создать Android Virtual Device в Android Studio?**

Запустим **Android** **Studio**, в верхнем меню нажмем **Tools->Android->AVD Manager**, в нем нажмем кнопку внизу с зеленым плюсом Create **Virtual** **Device**...

1. **Перечислите основные особенности тестирования мобильных приложений.**

**Тестирование** **мобильных** **приложений** в целом соответствует общим принципам **тестирования**, изложенным выше, но и в силу некоторых обстоятельств имеет ряд **особенностей**. Этими обстоятельствами являются: специфичность операционных систем для **мобильных** платформ, различные компании-изготовители устройств и конфигурации комплектующих, функциональность устройств как коммуникаторов и т. д.

1. **В чем основное отличие calabash от остальных инструментов тестирования?**

Calabash состоит из библиотек, которые дают возможность тесту программно взаимодействовать с нативными и гибридными приложениями. Взаимодействие заключается в виде реализованных действиях пользователя. Такими действиями могут быть:

* **Жесты**  
  Касания и жесты (tap, swipe, rotate и т.п.).
* **Проверки**  
  Например: Должна присутствовать кпопка «Login» на экране.
* **Получения снимка с экрана мобильного устройства**

Calabash можно сравнить с Selenium WebDriver. Однако, следует понимать, что взаимодействие с вэб-приложением с компьютера существенно отличается от взаимодействия нативных приложений использующих сенсорный экран.  
  
Calabash поддерживает Сucumber. Cucumber позволяет нам описать поведение программы используя простой «человеческий» язык, который понятен бизнес-аналитику, менеджеру и тестировщику без знаний программирования.

1. **Что такое Behavior Driven Development?**

**BDD** (сокр. от англ. **Behavior**-**driven** **development**, дословно «разработка через поведение») — это методология разработки программного обеспечения, являющаяся ответвлением от методологии разработки через тестирование (TDD). Основной идеей данной методологии является совмещение в процессе разработки чисто технических интересов и интересов бизнеса, позволяя тем самым управляющему персоналу и программистам говорить на одном языке.

1. **Опишите основные способы снятия логов на Android?**

Также существуют другие способы снятия логов с Android девайсов. Есть несколько способов с помощью IDE для разработки мобильных приложений «Android Studio».

**Для первого нам понадобится:**

1. Скачать «Android Studio» из архива официального сайта (так как в последних версиях студии нужный нам инструмент был удален, нам необходимо скачать более старую версию «Android Studio», например 2.3.3. Найти и скачать предыдущие версии можно в архиве (<https://developer.android.com/studio/archive)>).

2. Установить «Android Studio» на компьютер (в дальнейшем первые 2 шага повторять не придется).

3. Создать новый проект в «Android Studio» (во время создания нового проекта, нужно верно указать версию Android девайса, с которого будем снимать логи).

4. Нажать View → Tools Windows → Android Device Monitor (после этого откроется окно «Android Monitor»).

5. Выбрать опцию «Debug» в выпадающем списке (также желательно активировать режим «Scroll to down»).

6. Подключить устройство через USB (прежде необходимо включить отладку через USB на мобильном девайсе так, как описано ранее).

7. Воспроизвести ошибку на девайсе.

8. Изъять USB-кабель.

9. Выделить логи и копировать их (правой клавишей мыши → Copy или Ctrl + C).

10. Создать файл .txt и вставить в него наши логи.

11. Сохранить

Теперь у нас есть текстовый файл с логами, который можно прикреплять к баг-репорту.

**Для второго способа нужно:**

1. Скачать и установить последнюю версию «Android Studio».

2. Запустить «Android Studio».

3. Подключить устройство к ПК, предварительно включив отладку через USB.

4. Нажать вкладку LogCat в Android Studio.

5. Дождаться загрузки устройства.

6. Выбрать приложение из списка запущенных.

7. Скопировать логи (нажав Ctrl+A, а затем Ctrl+C).

8. Вставить логи в предварительно созданный текстовый документ.

Документ с логами для прикрепления в баг-репорт готов.

**Для третьего способа нужно:**

1. Скачать и установить последнюю версию «Android Studio».

2. Запустить один раз «Android Studio» для подкачивания нужных файлов.

3. Подключить устройство к ПК, предварительно включив отладку через USB.

4. Зайти в папку Android → sdk → tools → lib → monitorx86\_64.   
5. Запустить файл monitor.exe (и сделать ярлык на рабочий стол для быстрого доступа).

6. Выбрать устройство.

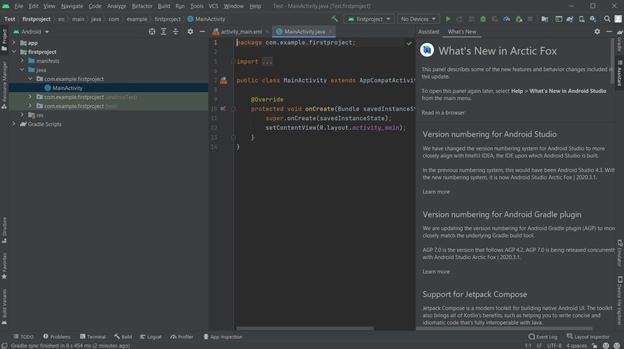
7. Выбрать вкладку «LogCat»

8. Дождаться полной загрузки логов.

9. Скопировать логи (нажав Ctrl+A).

10. Нажать кнопку «Save»

11. Выбрать нужный путь и сохранить.



**Вывод:** благодаря этой лабораторной работе я научиться проводить тестирование мобильных приложений на Android методом белого ящика в среде разработки Android Studio и методом черного ящика с использованием симуляторов или реальных устройств. Научилась снимать логи с Android устройств.