ТИНЬКОФФ

5 уроков, вынесенных из опыта реализации pixel-perfect тестирования дизайн-системы в Android

Рустам Гумеров

01.06.2024

Рустам Гумеров

Android Lead
команды разработки
дизайн-системы
Tinkoff

psrustik@yandex.ru

@psrustik

План доклада

1. Основной проект команды
2. Проблемы обеспечения качества
3. Пути и решения
4. Android UI изнутри
5. Что получилось в итоге и какие уроки извлекли

Основной проект

Дизайн-система

Атомы

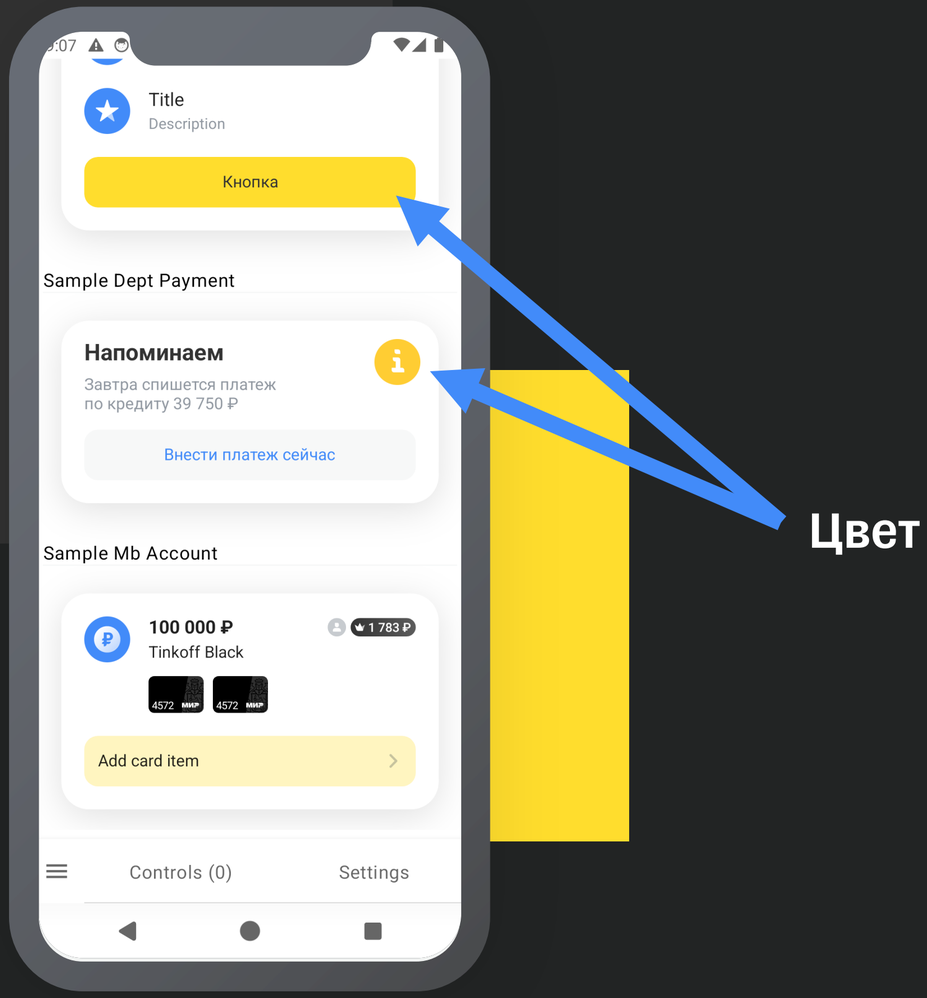
- Базовые кирпичики
- Палитра
- Текстовые стили

Молекулы

- Инпуты
- Формы
- Ячейки

Организмы

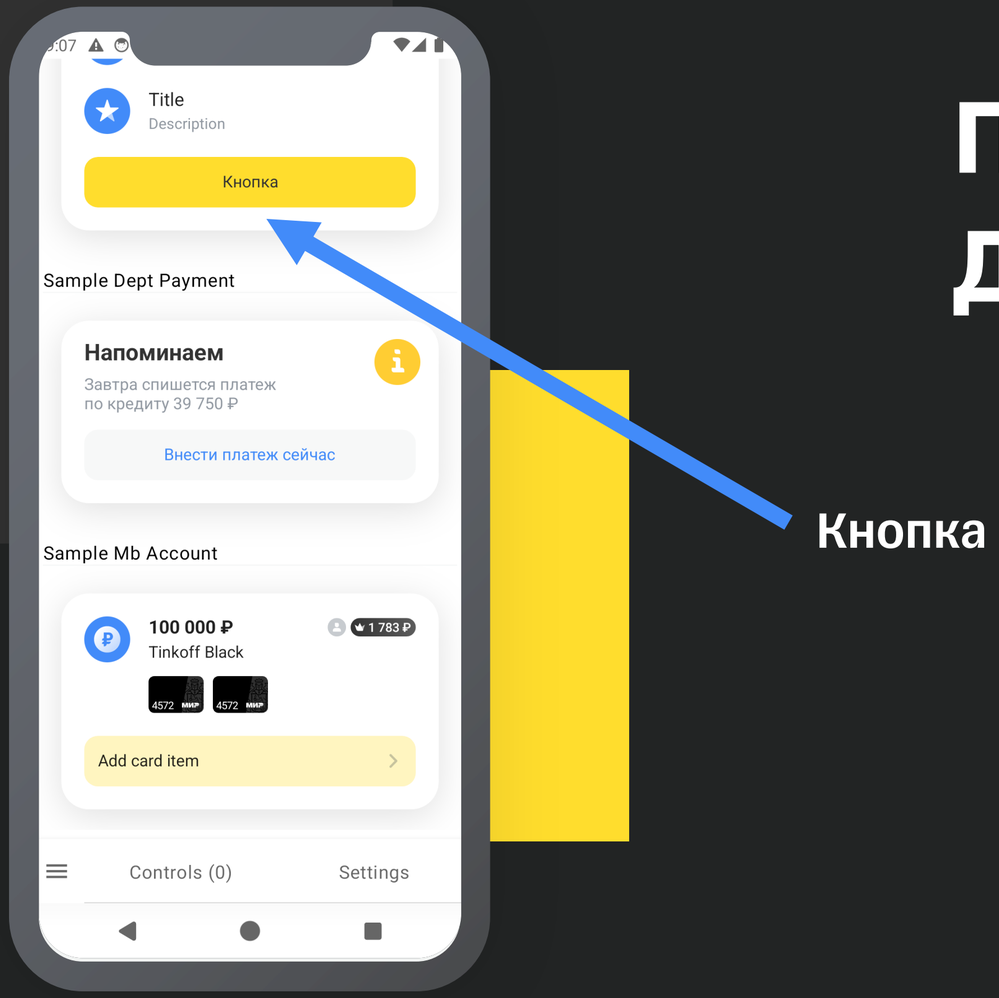
- Онбординги
- Карточки
- etc.



Пример компонента Дизайн-системы

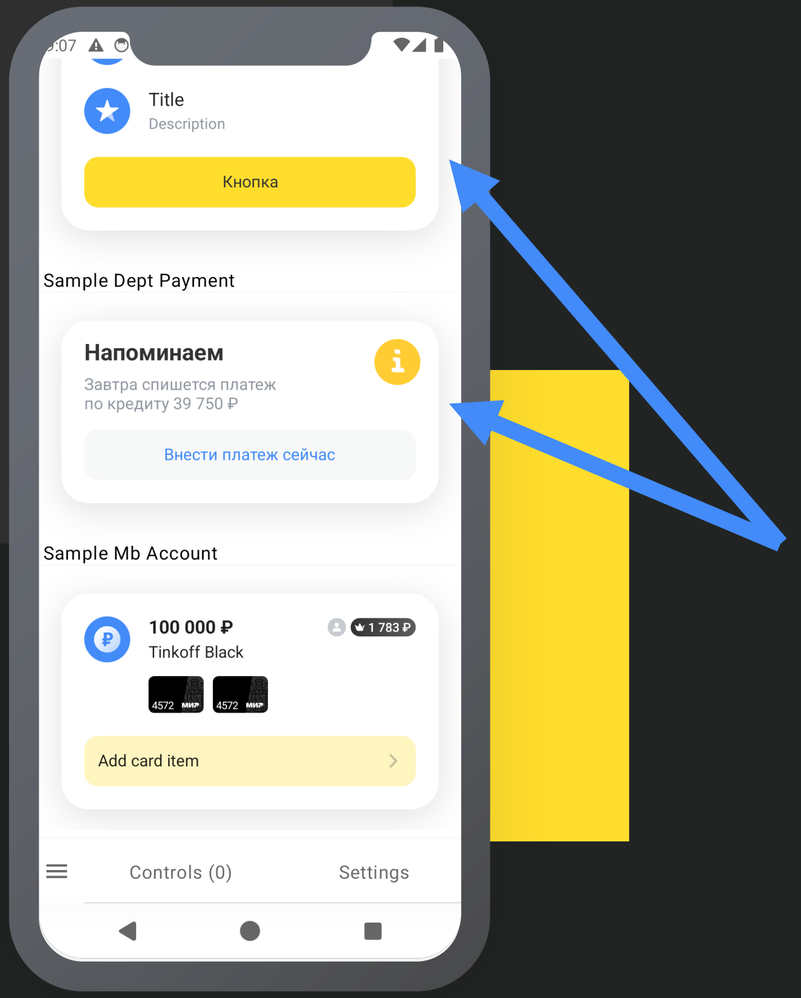
Цвет

Пример компонента Дизайн-системы



Кнопка

Пример компонента Дизайн-системы



Карточка

Дизайн-система как Ui Kit

Состав

Набор общих решений, View, кода для построения UI

Android Library

Несколько библиотек и модулей, публикуемых как Maven artefact

Kotlin, Java

Актуальный и Legacy код

Demo App,

Документация и примеры

Основной пользователь - разработчик

В чем особенности ?

Ui Kit на 2022

Плюсы

- Множество компонентов
- Актуальный и устаревший код

Минусы

- Затяжные ручные регресс-тестирования
- Мало автотестов
- Неготовность к масштабированию

Планируемое развитие

01

Кратное увеличение компонентов

02

Новые функции в существующих

Accessibility

03

Увеличение штата разработки

04

Compose

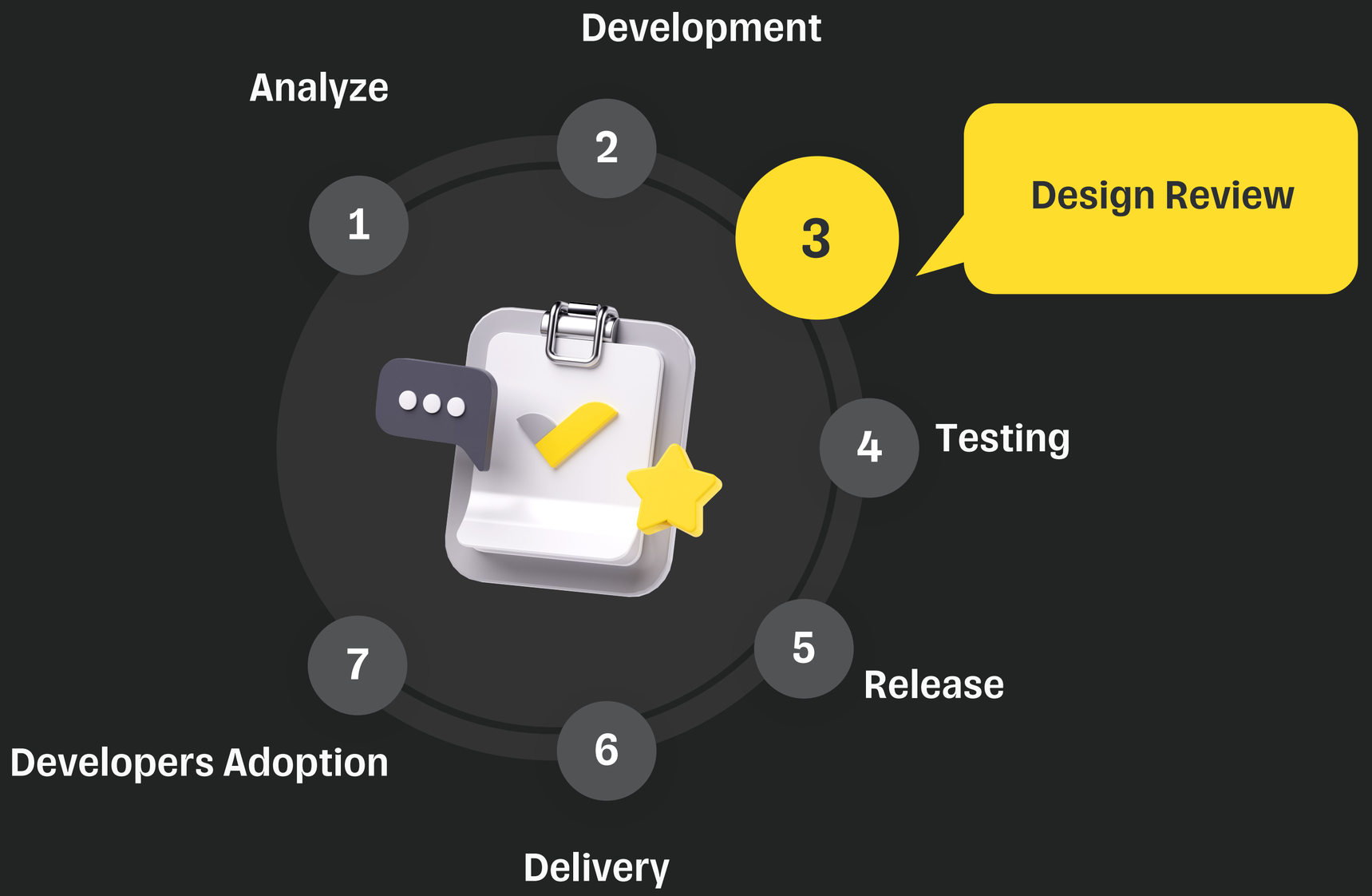
05

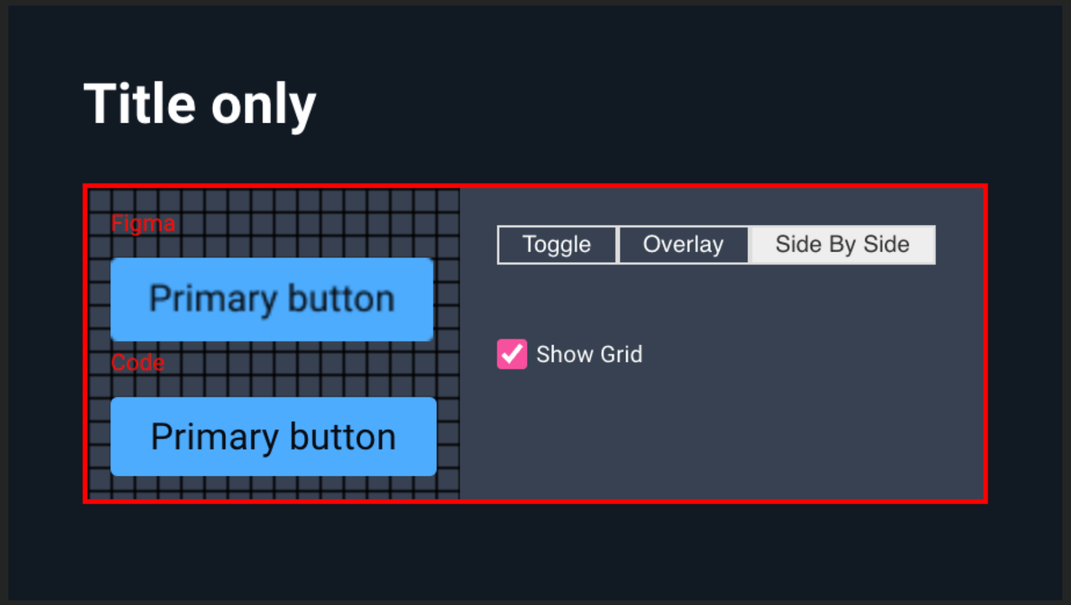
Innersource Contributing

06

Подход Quality assistance

Value Delivery





Дизайн-ревью компонента

1. Установить демо приложение
2. Проверить визуал в соответствии с ТЗ
3. Снять скриншоты
4. Сравнить с оригиналом
   Figma - Pixel Perfect

Организация процесса скриншот-тестирования

Автоматизация скриншот-тестирования

Эталонный снимок

Снимается после дизайн-ревью и релиза

Актуальный снимок

Снимается на изменениях (Merge Request)

Есть разница ?

Выясняем это:

Bug

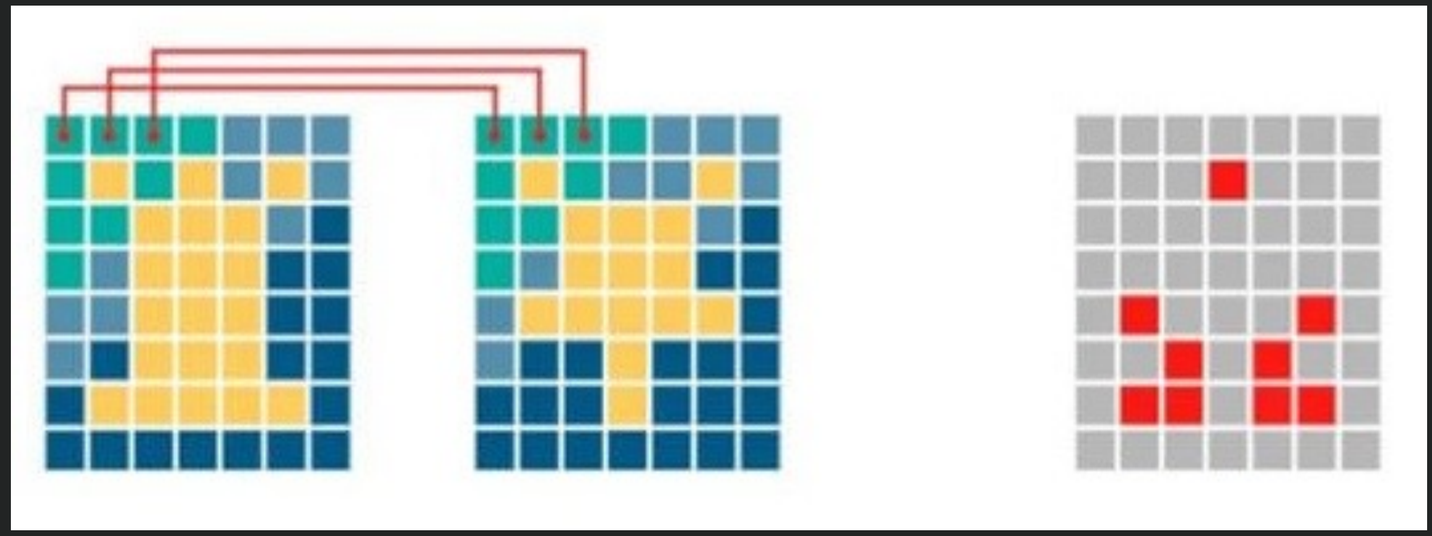
Feature

Попарное сопоставление пикселей

Bitmap 1
Reference

Bitmap 2
Actual

Comparison Result:
разница отмечена красным



Инструментальный тест

```
/**
 * Instrumented test, which will execute on an Android device.
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun testScreenshot() {
        launch(MainActivity::class.java).use {
            // 1. Считываем эталон
            val referenceBitmap = BitmapFactory.decodeFile("assets/1.jpg")
        }
    }
}
```

Инструментальный тест

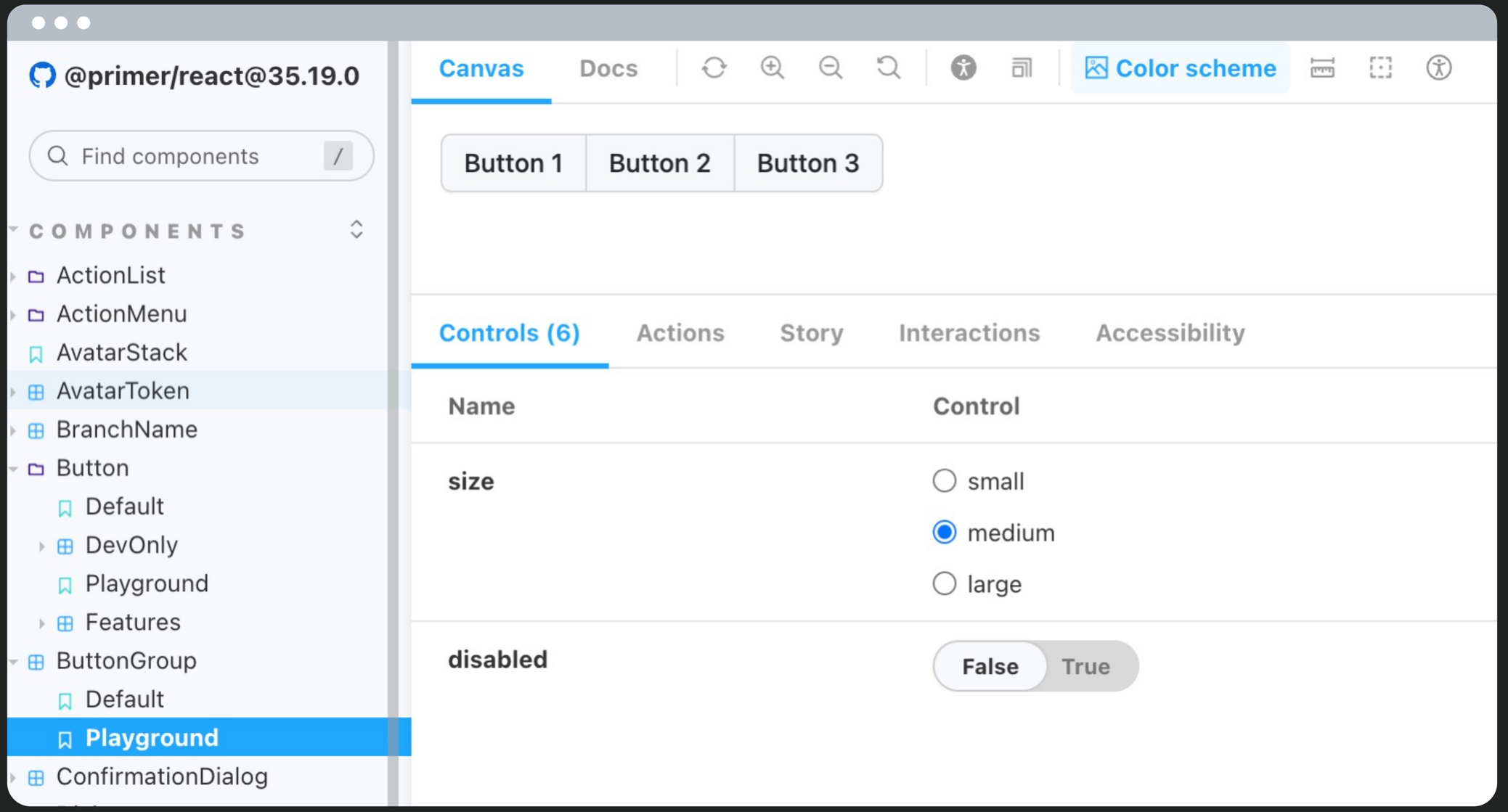
```
/**
 * Instrumented test, which will execute on an Android device.
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun testScreenshot() {
        launch(MainActivity::class.java).use {
            // 1. Считываем эталон
            val referenceBitmap = BitmapFactory.decodeFile("assets/1.jpg")
            // 2. Снимем текущий снимок
            val actualBitmap =
onView(withId(R.id.mainContent)).captureToBitmap()
        }
    }
}
```

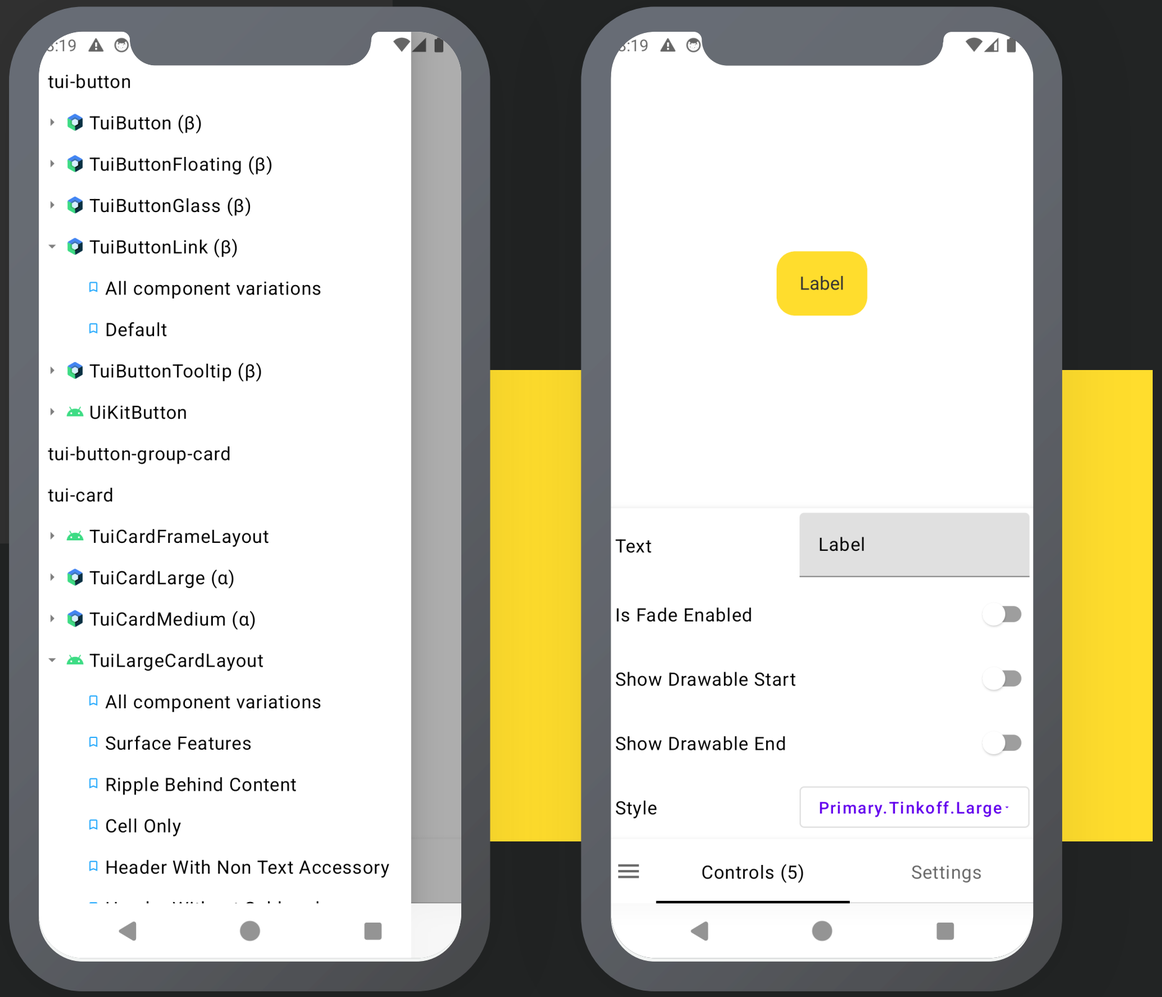
Инструментальный тест

```
/**
 * Instrumented test, which will execute on an Android device.
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun testScreenshot() {
        launch(MainActivity::class.java).use {
            // 1. Считываем эталон
            val referenceBitmap = BitmapFactory.decodeFile("assets/1.jpg")
            // 2. Снимем текущий снимок
            val actualBitmap =
onView(withId(R.id.mainContent)).captureToBitmap()
            // 3. Сравним и проверим результат
            val isSameImage = compareBitmaps(referenceBitmap, actualBitmap)

            assertTrue(isSameImage)
        }
```

Demo приложение

Подход StoryBook.js





Storybook в demo приложении

Описываем StoryBook

Внедряем сущности Story и StoryBook container

```
object ButtonStories {
    val primary: View by Story {

    }
    val secondary: View by Story {
    }
}

interface StoryContext {
    val context: Context
}
```

Реализация Story

Наполняем контентом через XML layout или программным способом

```
object ButtonStories {
    val primary: View by Story {
        LayoutInflater.from(it.context).inflate(R.layout.activity_main)
    }
    val secondary: View by Story {
        Button(it.context).apply {
            text = "SampleButton"
        }
    }
}

interface StoryContext {
    val context: Context
}
```

Demo Storybook

Плюсы

Component Driven Tools

Demo для пользователя, дизайнера, тестировщика

Demo Storybook

Плюсы

Component Driven Tools

Demo для пользователя, дизайнера, тестировщика

А может быть это и есть Test Case?

Скриншот-тестирование V1

- Junit4 Parameterized tests
- Запускает Activity на эмуляторе
- Вставляет контент story View
- Сравнивает скриншоты

Прототип тестирования V1

```
@RunWith(Parameterized::class)
class ButtonStoriesTest(private val storyId: Array<Int>) {
    companion object {
        @JvmStatic
        @Parameterized.Parameters
        fun dataProvider(): List<Array<String>> {
            return listOf(/*ButtonStories ids*/)
        }
    }

    @Test
    fun shouldCorrectSnapshot() {
        val story: View = ButtonStories.getStory(storyId[0])
        launchActivity<MainActivity>().use { scenario ->
            scenario.onActivity { activity ->
                checkStory(story, activity)
            }
        }
    }
```

Итоги скриншот-тестирования V1

- Stories as Test
- Pixel Perfect
- Интеграция с платформой

- Начали переписывать демо под формат StoryBook = 40 стори-кейсов
- Появились первые скриншот-автотесты
- Эталоны снимали вручную с эмулятора

Первый урок – готовь структуру приложения к модульному скриншот-тестированию

Недостатки V1

Минусы

Скорость прогона автотестов

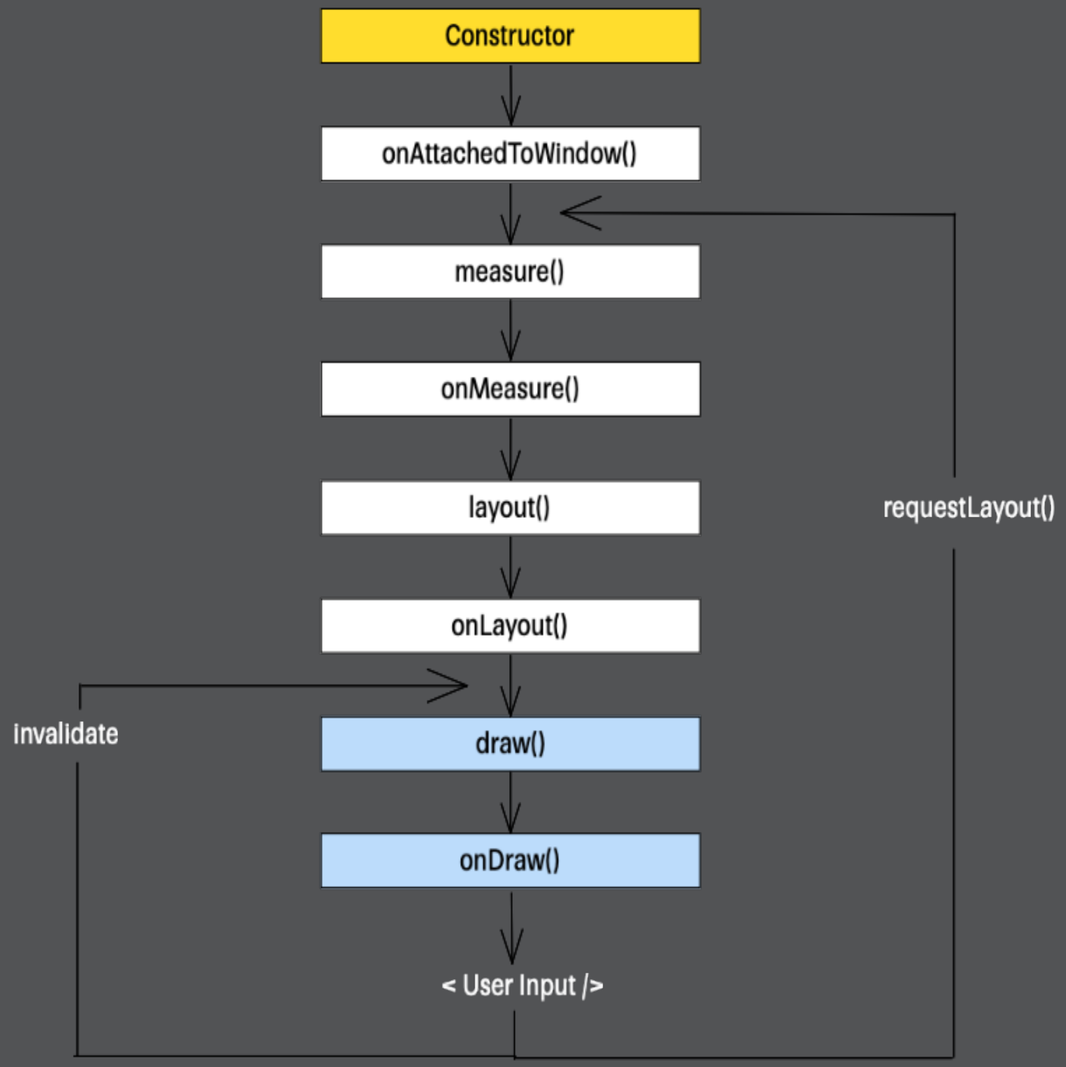
Повышение производительности

Повышение производительности

Можно ли прогнать UI-тесты без запуска Activity ?

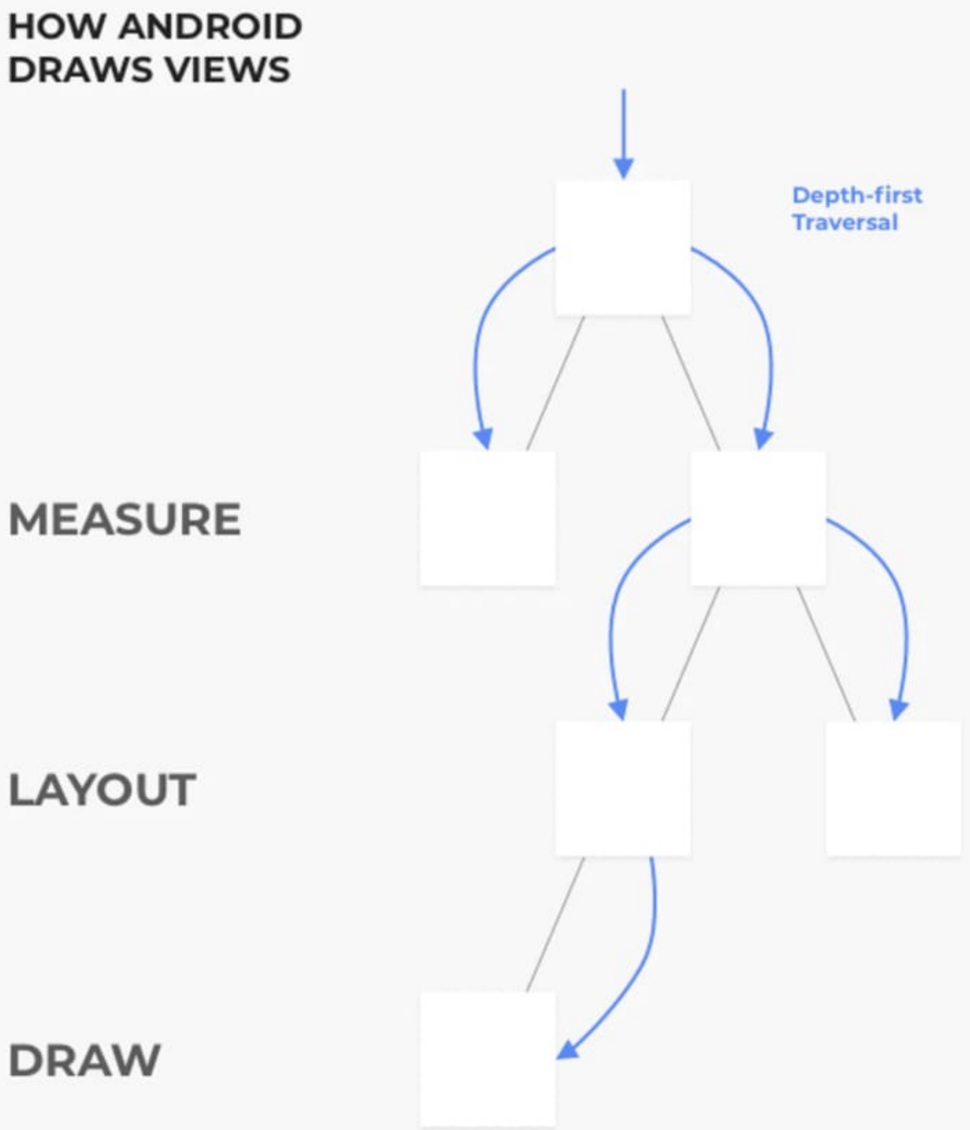
Отрисовка Android Views

Классическое описание работы Views



Обход View Tree

Depth-first Traversal

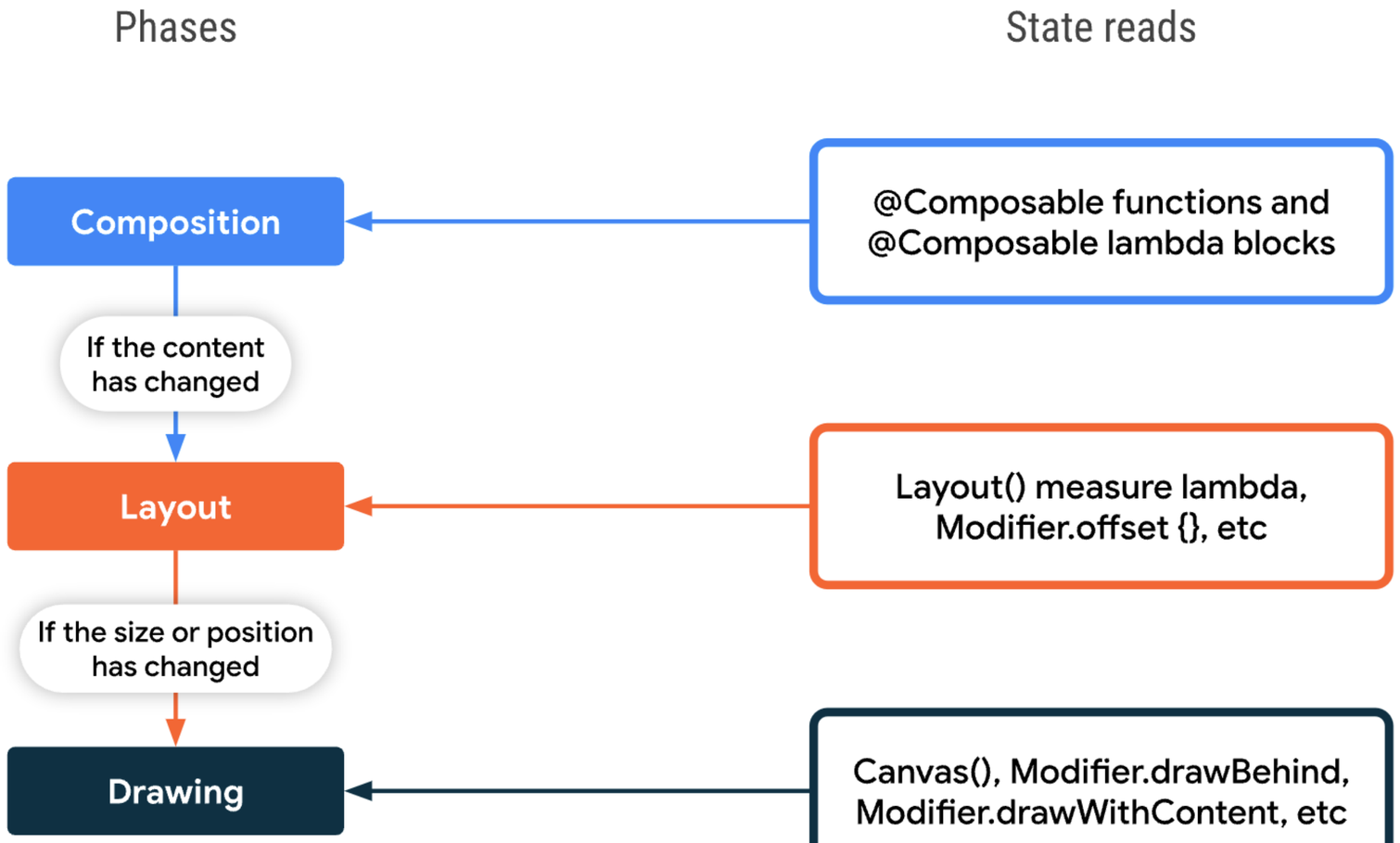


Draw() – что там внутри?

Примеры Canvas операций в TextView – как родителя всех кнопок

```
public class TextView {
    …
  public void drawText(BaseCanvas c, int start, int end,float x, float y, Paint p) {
      c.drawText(mChars, start + mStart, end - start, x, y, p);
  }

  public void drawTextRun(BaseCanvas c, int start, int end, int contextStart, int contextEnd, float
x, float y, boolean isRtl, Paint p) {
      int count = end - start;
      int contextCount = contextEnd - contextStart;
     c.drawTextRun(mChars, start + mStart, count, contextStart + mStart,contextCount, x, y, isRtl, p)
  }
  …
}
```

Compose drawing



https://developer.android.com/jetpack/compose/phases

Compose LayoutNode → OwnerLayer → RenderNodeLayer

```
override fun drawLayer(canvas: Canvas) {
    val androidCanvas = canvas.nativeCanvas
    if (androidCanvas.isHardwareAccelerated) {
        updateDisplayList()
        drawnWithZ = renderNode.elevation > 0f
        if (drawnWithZ) {
            canvas.enableZ()
        }
        renderNode.drawInto(androidCanvas)
        if (drawnWithZ) {
            canvas.disableZ()
        }
    }
}
```

Все сходится к отрисовке на Canvas

The Canvas class holds the "draw" calls.

To draw something, you need 4 basic components: A Bitmap to hold the pixels, a Canvas to host the draw calls (writing into
the bitmap), a drawing primitive (e.g. Rect, Path, text, Bitmap), and a paint (to describe the colors and styles for the drawing).

Способ отрисовать View на Bitmap

View (или ViewGroup) – заранее подготовлен, измерен и присоединен к окну

```
val view = activity.findViewById<View>(R.id.mainContent)
val bitmap = Bitmap.createBitmap(view.measuredWidth,
view.measuredHeight, Bitmap.Config.ARGB_8888)
```

Способ отрисовать View на Bitmap

```
val view = activity.findViewById<View>(R.id.mainContent)
val bitmap = Bitmap.createBitmap(view.measuredWidth,
view.measuredHeight, Bitmap.Config.ARGB_8888)

val canvas = Canvas(bitmap)
view.draw(canvas)
```

Таким образом выведем View на своем, новом Canvas

Но ведь не хотим запускать Activity

Activity DecorView

DecorView содержит все другие вью и связан с lifecycle

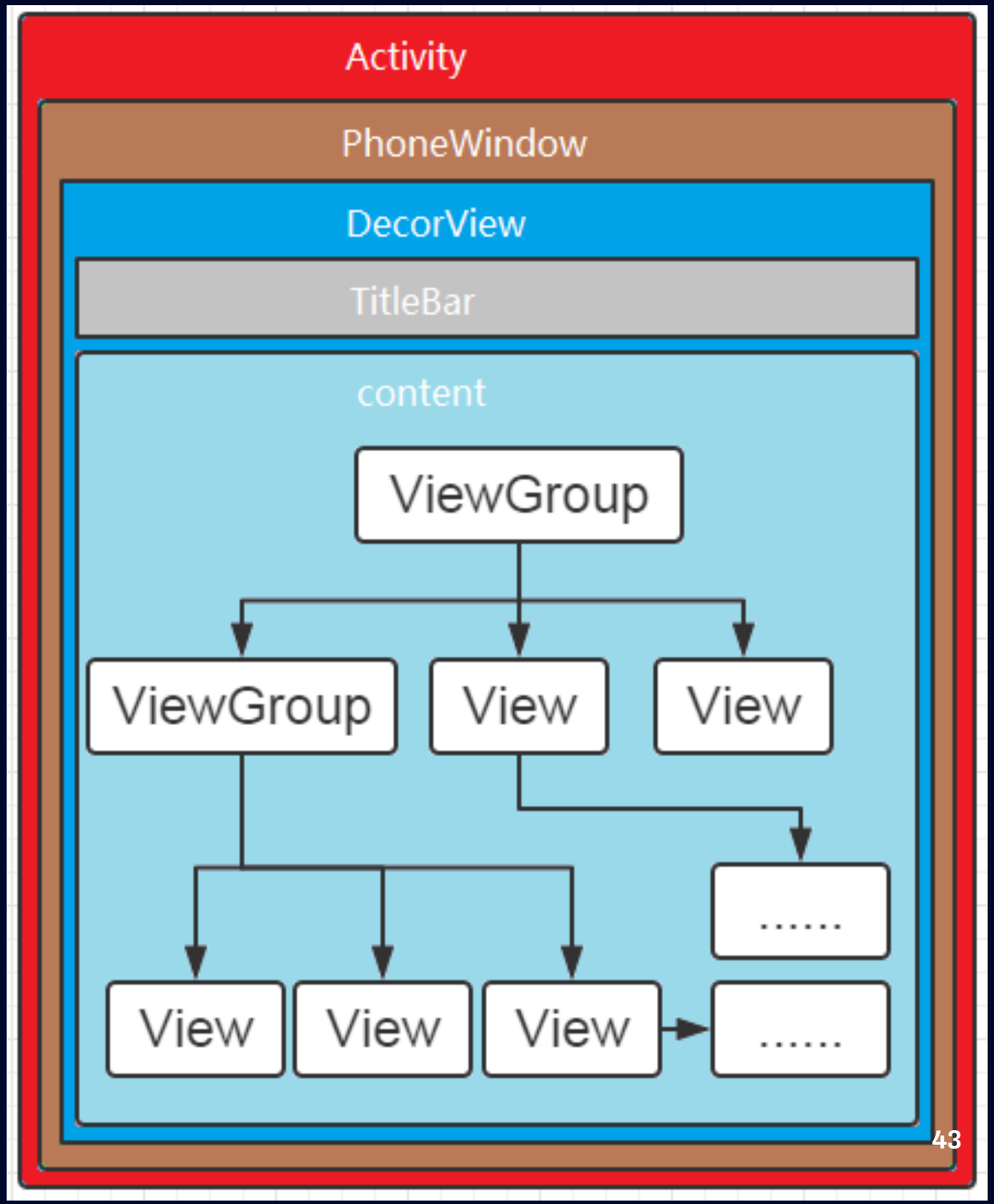
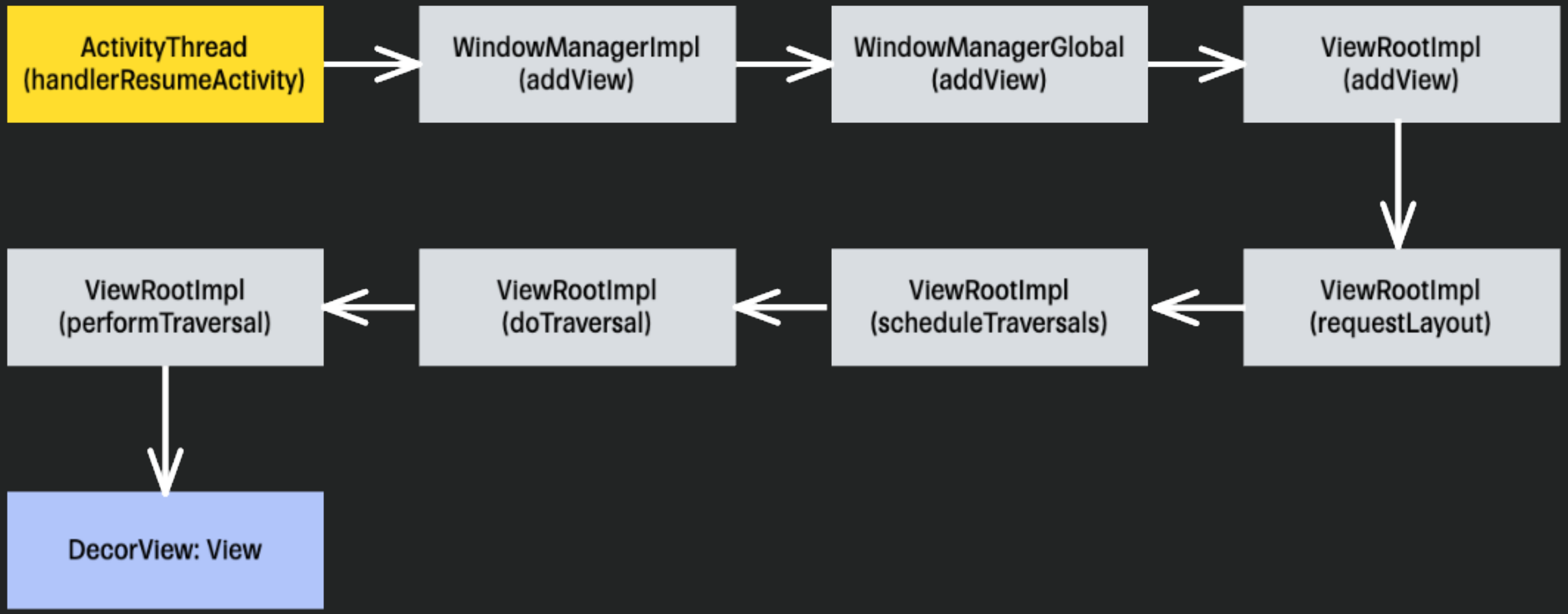


Схема отрисовки DecorView и его потомков



Facebook Screenshot Testing

https://github.com/facebook/screenshot-tests-for-android

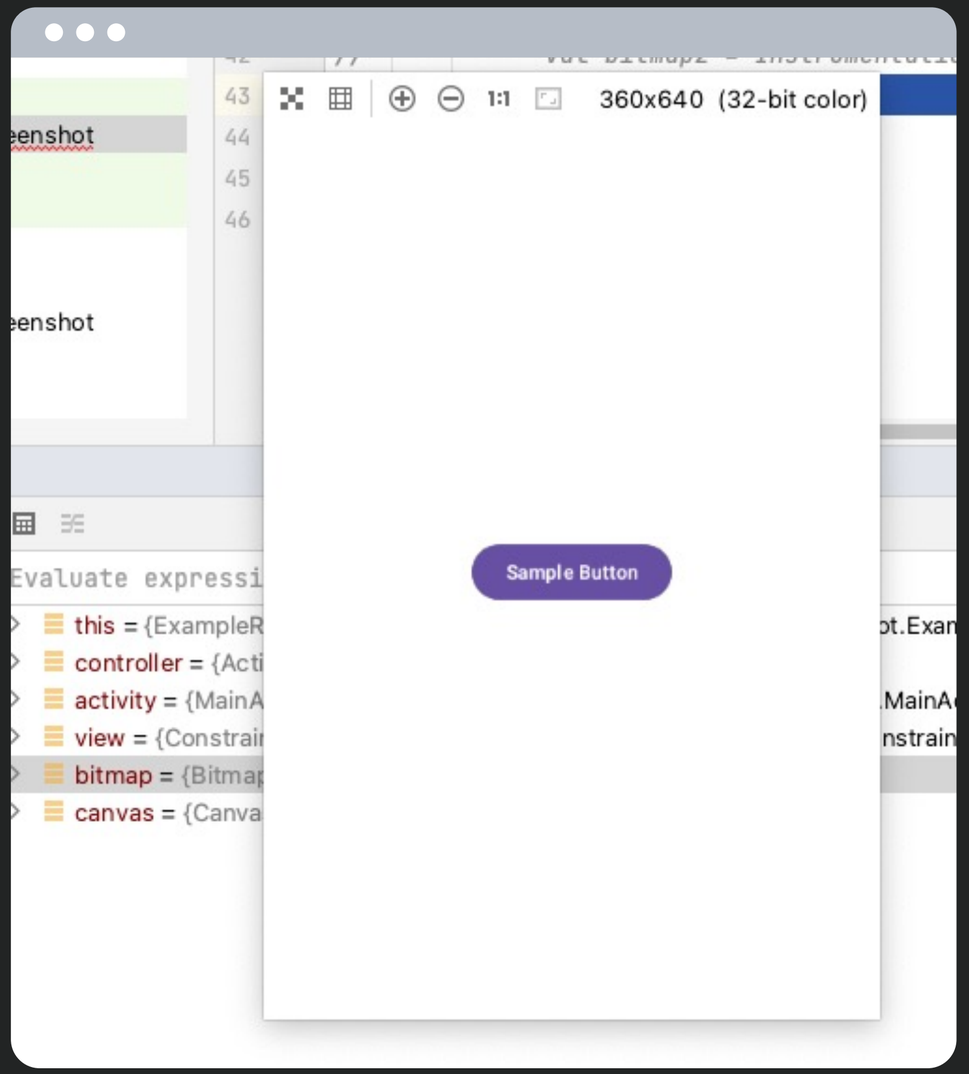
```
/**
 * A collection of static utilities for measuring and pre-drawing a view, usually a pre-requirement
 * for taking a Screenshot.
 *
 * <p>This will mostly be used something like this: <code>
 *   ViewHelpers.setupView(view)
 *     .setExactHeightPx(1000)
 *     .setExactWidthPx(100)
 *     .layout();
 * </code>
 */
```

Отрисовать View на canvas (без Activity)

С помощью ViewHelpers.**setupView инициалируем Story**

```
@RunWith(AndroidJUnit4::class)
class MyScreenshotInstrumentedTest {
    @Test
    fun testScreenshot() {
        ActivityScenario.launch(MainActivity::class.java).use {
            // 1. Считываем эталон
            val referenceBitmap = BitmapFactory.decodeFile("assets/01.jpg")
            // 2. Снимем текущий снимок
            val view: android.view.View = ButtonStories.primary
            ViewHelpers.setupView(view)
            // 3. Сравним и проверим результат
        }
    }
}
```

Полученный Bitmap



Автоматизация идеи

Custom AndroidJUnitRunner

01

Собрать @Test кейсы
из сторибука

02

Подготовить Contex
MDPI

w360dp-h640dp

03

- Выбрать Story
- Инициализировать predraw/dispatch
- Отрисовать на Canvas

04

Сравнить с эталоном

Собственный Runner

```
// When creating a custom runner, in addition to implementing the abstract methods here you must also
// provide a constructor that takes as an argument the Class containing the tests.
public abstract class Runner implements Describable {

    public abstract Description getDescription();

    public abstract void run(RunNotifier notifier);

    public int testCount() {
        return getDescription().testCount();
    }
}
```

Как выглядит тест со своим раннером

```
@RunWith(StorybookScreenshotJUnit4::class)
@StorybookScreenshotJUnit4.Stories(ButtonStories::class)
class TuiButtonScreenshotTest
```

Итоги скриншот-тестирования V2

- Производительность
- Меньше кода
- Не запускаем Activity

- 200 тест-кейсов за несколько минут
- Можно прогонять тесты из Android Studio*
- Есть немного ручной работы (снять эталон например)

Урок 2 – сокращай время прогона автотестов

Проблема «M1»

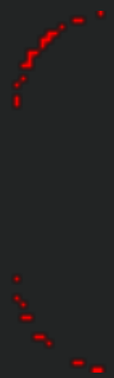
CI Emulator

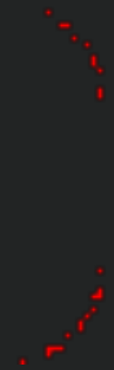
Emulator Intel

Emulator M1

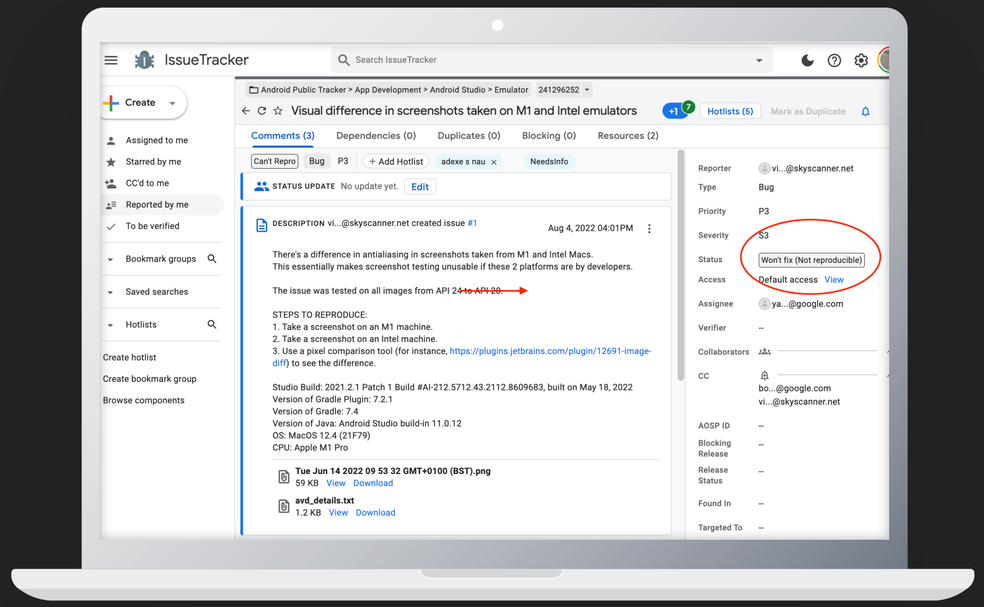
Одна и та же кнопка

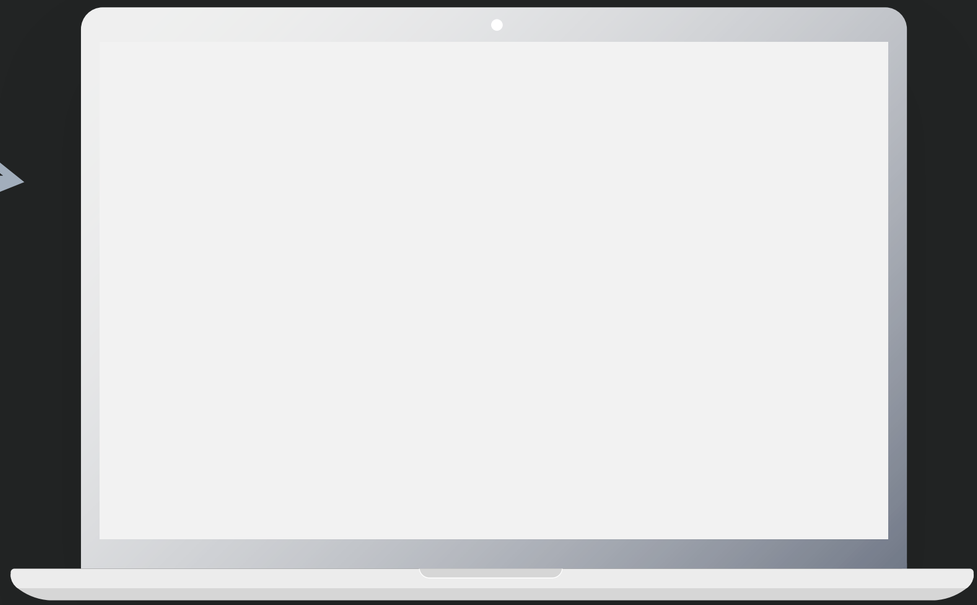
Снятые скрины отличаются





Баги про разницу M1, Intel emulators





Wont'Fix

https://issuetracker.google.com/issues/241296252

RenderNode внутри View

```
class View {
    fun measure(widthMeasureSpec: Int, heightMeasureSpec: Int): Unit {}

    fun layout(left: Int, top: Int, right: Int, bottom: Int): Unit {}

    fun draw(canvas: Canvas, parent: ViewGroup, drawingTime: Long): Boolean {
        val renderNode = RenderNode("view")
        renderNode.setPosition(Rect(0, 0, this.measuredWidth, this.measuredHeight))
        val recordingCanvas: RecordingCanvas = renderNode.beginRecording()
        draw(recordingCanvas)
        renderNode.endRecording()
        return true
    }

    open fun draw(canvas: Canvas) {

    }

}
```

Типичный наследник View

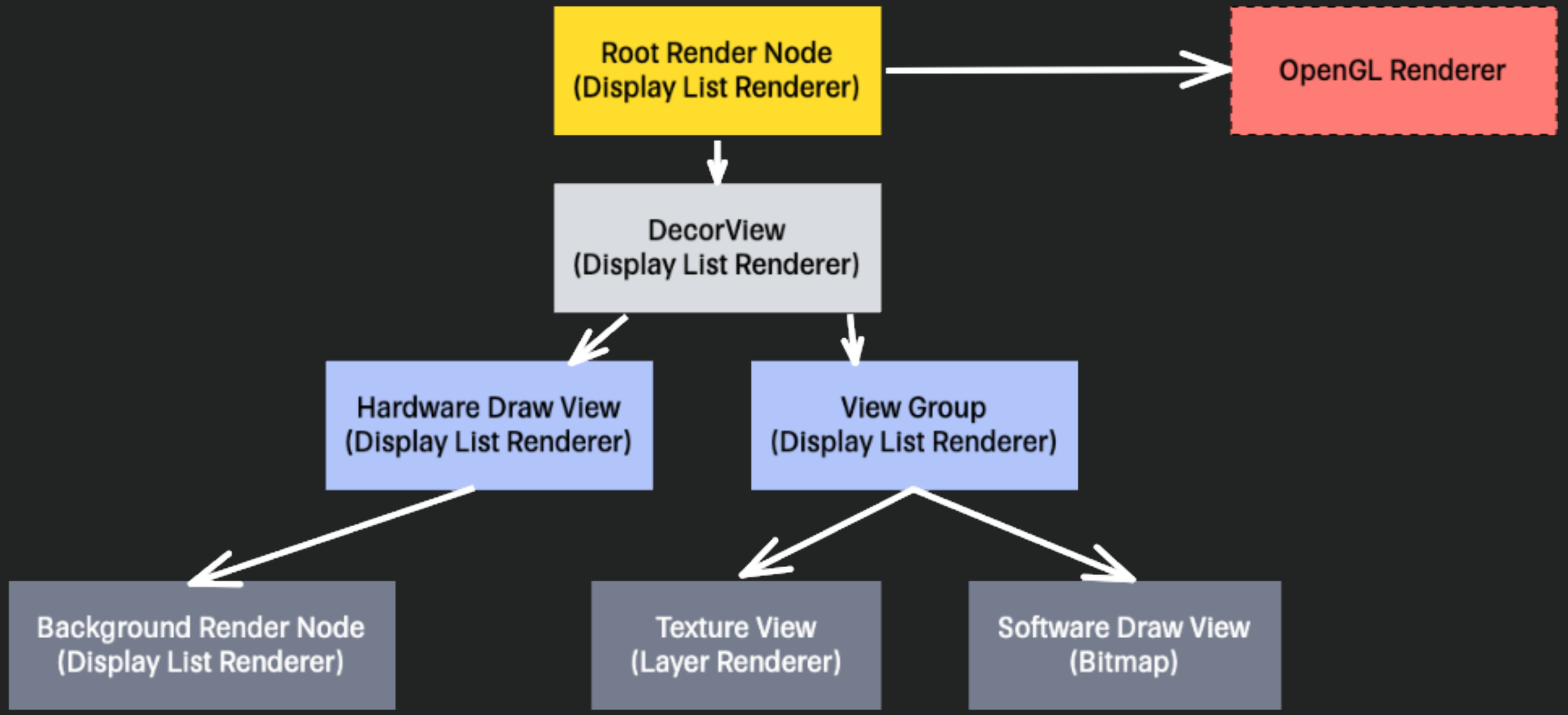
```
class Button: View() {

    override fun draw(canvas: Canvas) {
        super.draw(canvas)
        // View draw operations: drawLine, drawPath, drawText()
    }
}
```

- RecordingCanvas передается наследникам
- Наследники «рисуют» на Canvas

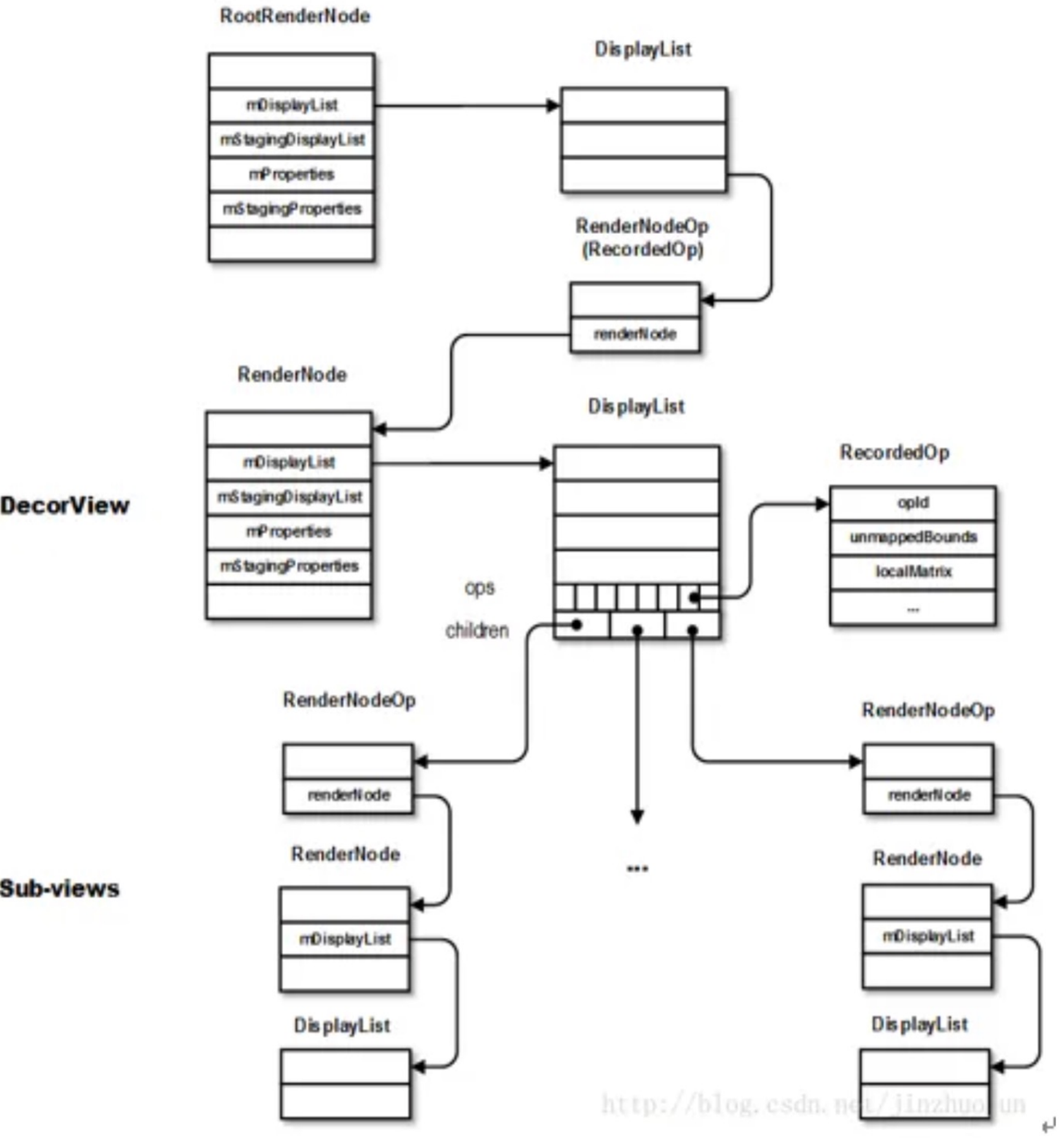
RenderNode & RecordingCanvas

- По умолчанию применяется внутри любых вьюх (начиная с Android 5)
- Работает только hardwareAccelerated режиме
- Накапливает все операции в DisplayList
- Не нужна полная перерисовка экрана, когда меняется конкретная View

RenderNode с каждой View



最终生成的DisplayList如下图14。



Display List

Подробнее

Накапливает операции

В массивы

Агрегируется

В рутовой RenderNode

DisplayList пример для операции ListView.setAlpha(0.5)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Text" />
</LinearLayout>
```

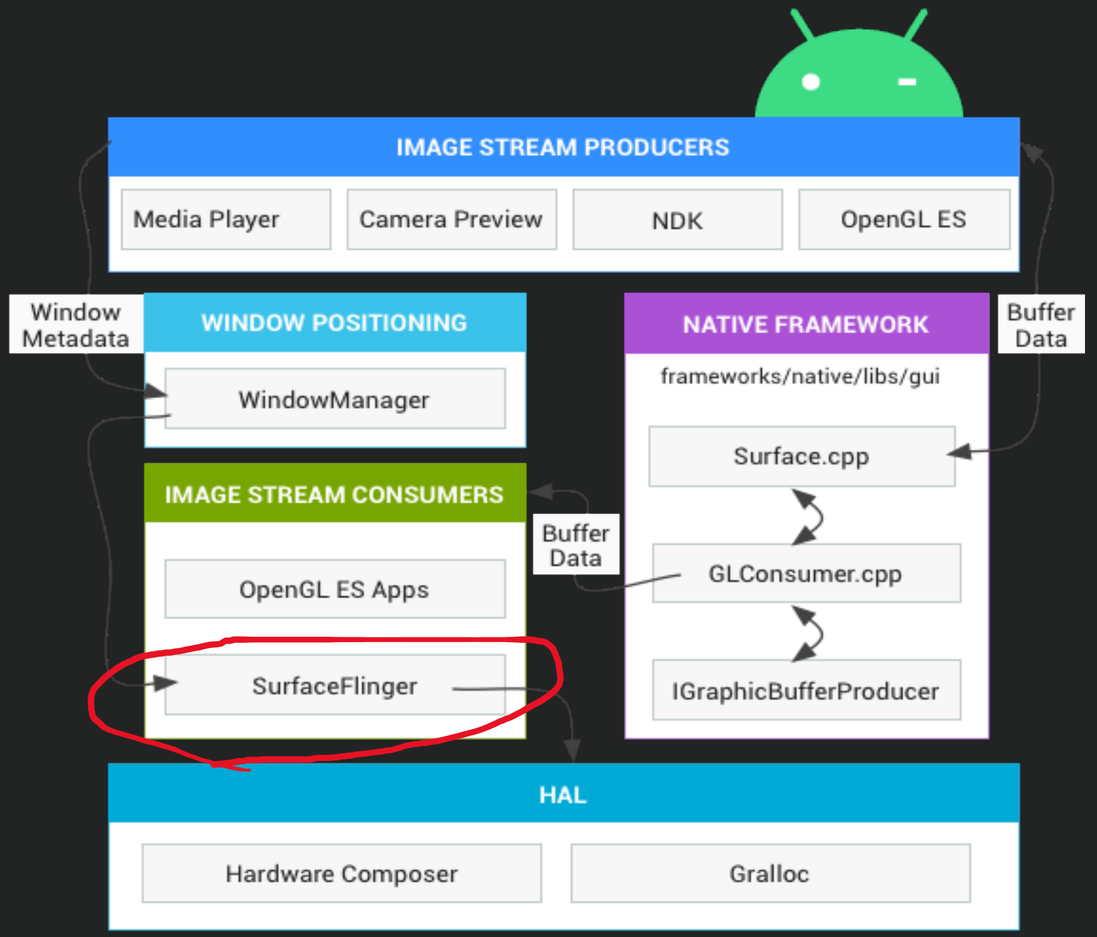
DisplayList пример для операции ListView.setAlpha(0.5)

SaveLayerAlpha(0.5)
DrawDisplayList(ListView)
Restore
DrawDisplayList(Button)

От View до OpenGL

- View
- RecordingCanvas
- RenderNode
- DisplayList
- DisplayList Arrays
- Root Renderer (GL)

Графические компоненты Android



SurfaceFlinger dump

Получим характеристики системного сервиса

```
adb shell dumpsys SurfaceFlinger
```

SurfaceFlinger dump (M1 Emulator)

~ adb shell dumpsys SurfaceFlinger

GLES: Google (Apple), Android Emulator OpenGL ES Translator (Apple M2 Max), OpenGL ES 3.0 (4.1 Metal - 83.1)

GL_EXT_debug_marker
GL_EXT_robustness
GL_OES_EGL_sync
// остальной вывод модулей OpenGL...

SurfaceFlinger dump (Intel Emulator)

```
adb shell dumpsys SurfaceFlinger | grep GL

GLES: Google (VMware, Inc.), Android Emulator OpenGL ES Translator (llvmpipe (LLVM 10.0.0, 256 bits))
```

SurfaceFlinger dump (Intel Emulator)

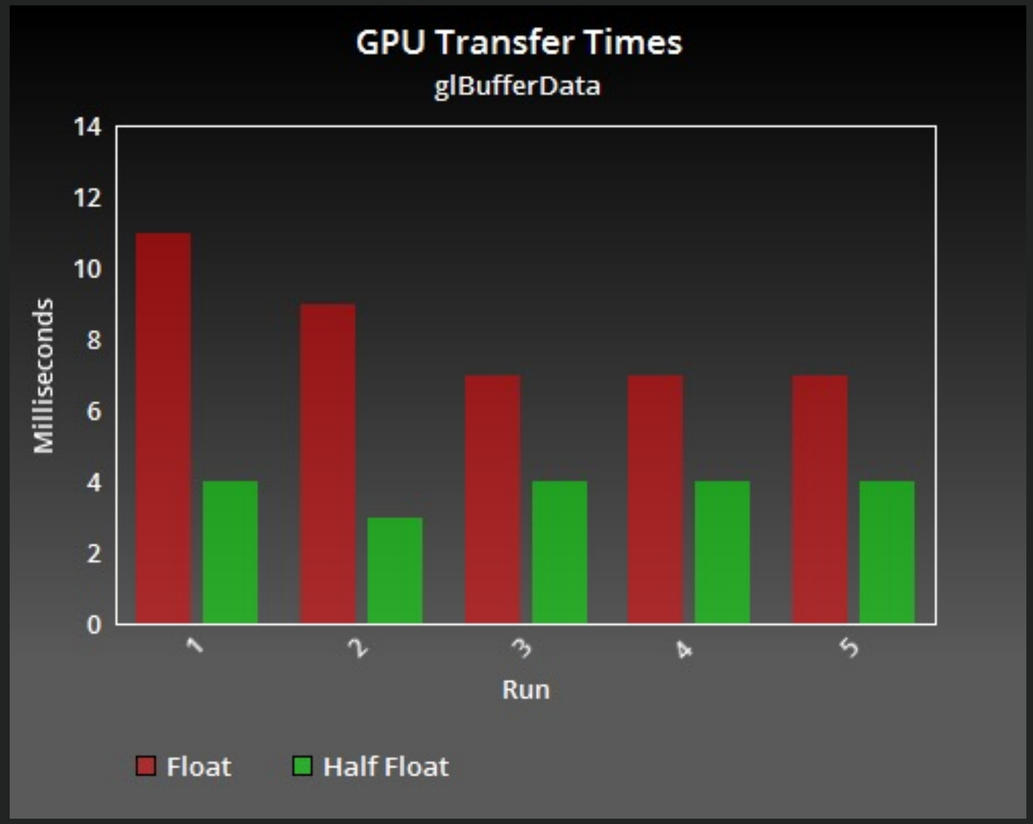
```
adb shell dumpsys SurfaceFlinger | grep GL

GLES: Google (VMware, Inc.), Android Emulator OpenGL ES Translator (llvmpipe (LLVM 10.0.0, 256 bits))
```

Нет модулей
OES_vertex_half_float, ...

OES_vertex_half_float

Overview

This extension adds a 16-bit floating pt data type (aka half float) to vertex data specified using vertex arrays. The 16-bit floating-point components have 1 sign bit, 5 exponent bits, and 10 mantissa bits. The half float data type can be very useful in specifying vertex attribute data such as color, normals, texture coordinates etc



На результат рендера Canvas влияют

Версии софта

Android SDK

OpenGL

Архитектуры

CPU

GPU драйвер

OpenGL extensions

layerType

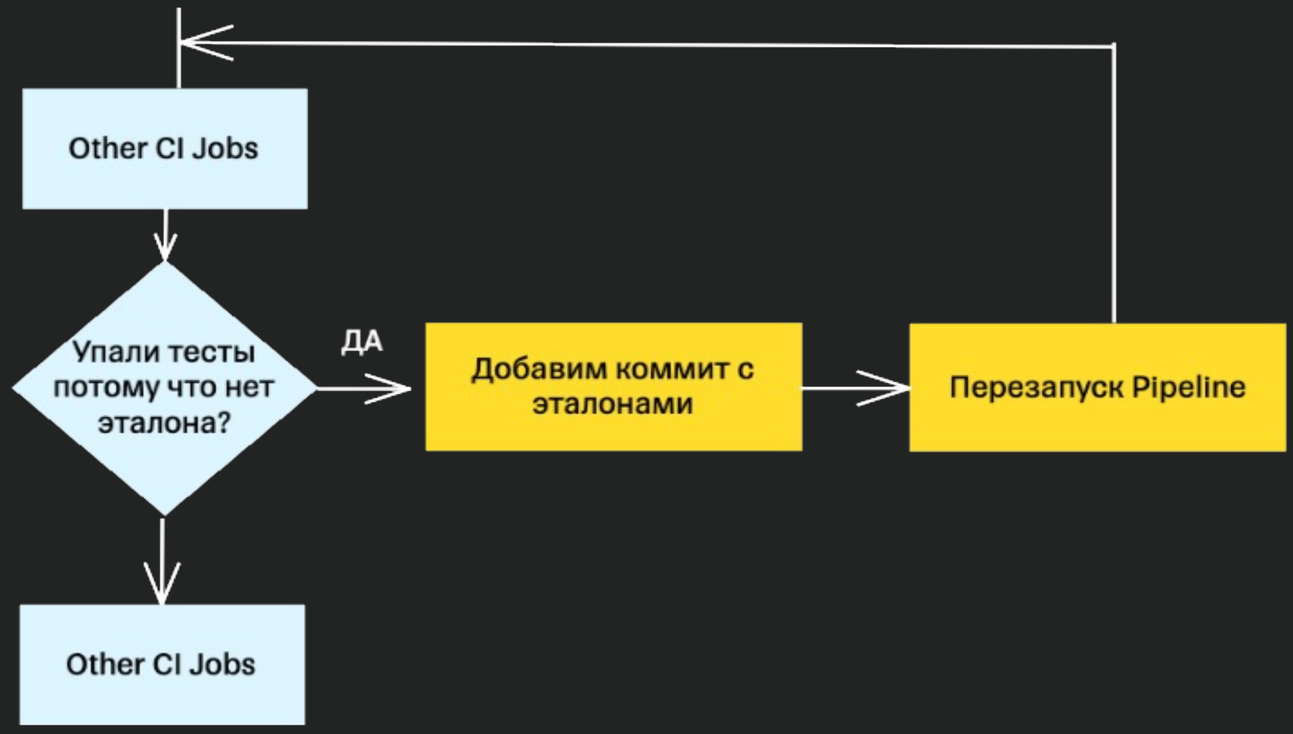
Урок 3 – выбери один источник правды для эталонов

Недостатки V2

Минусы

Эталоны снимать неудобно

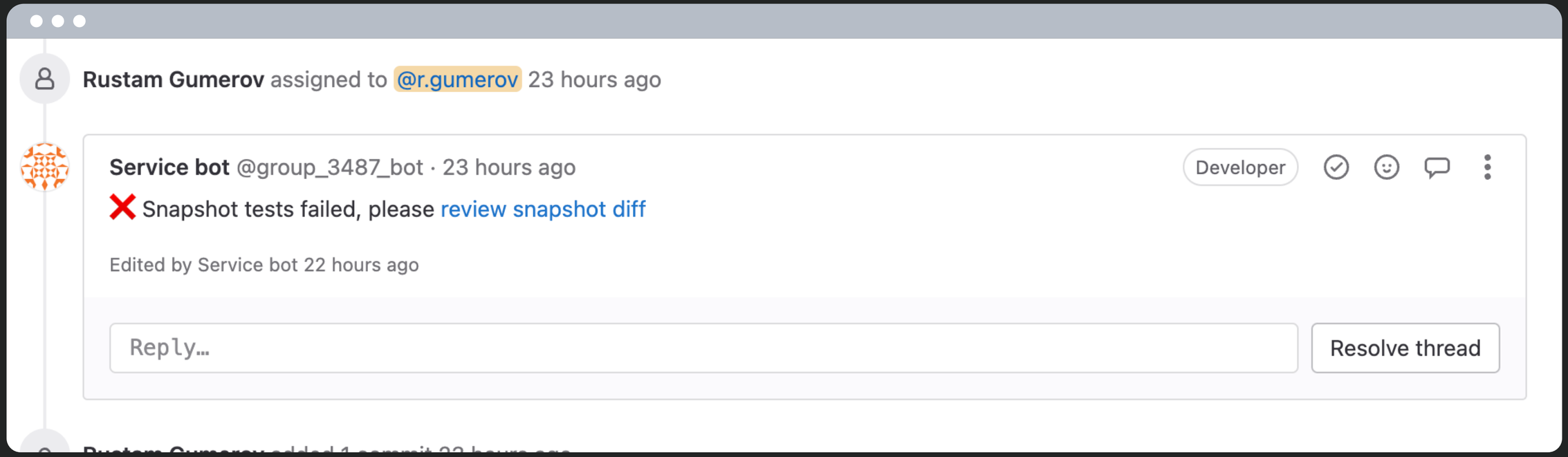
Автозапись скриншотов на CI



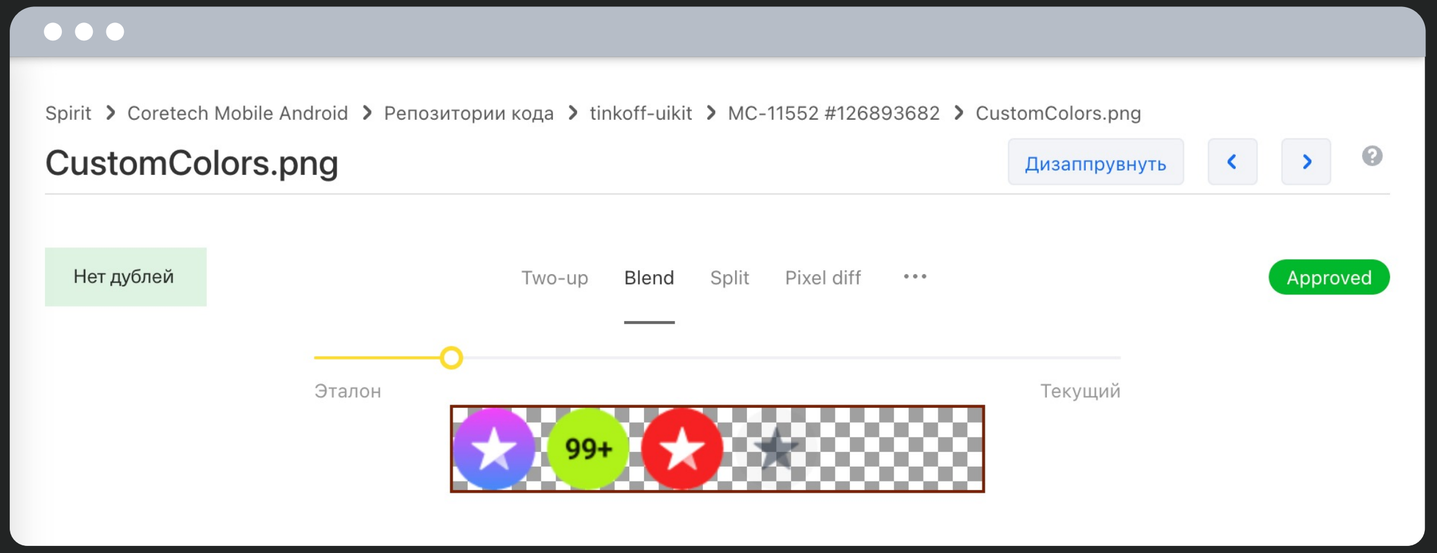
Fast Fail подход в автозаписи

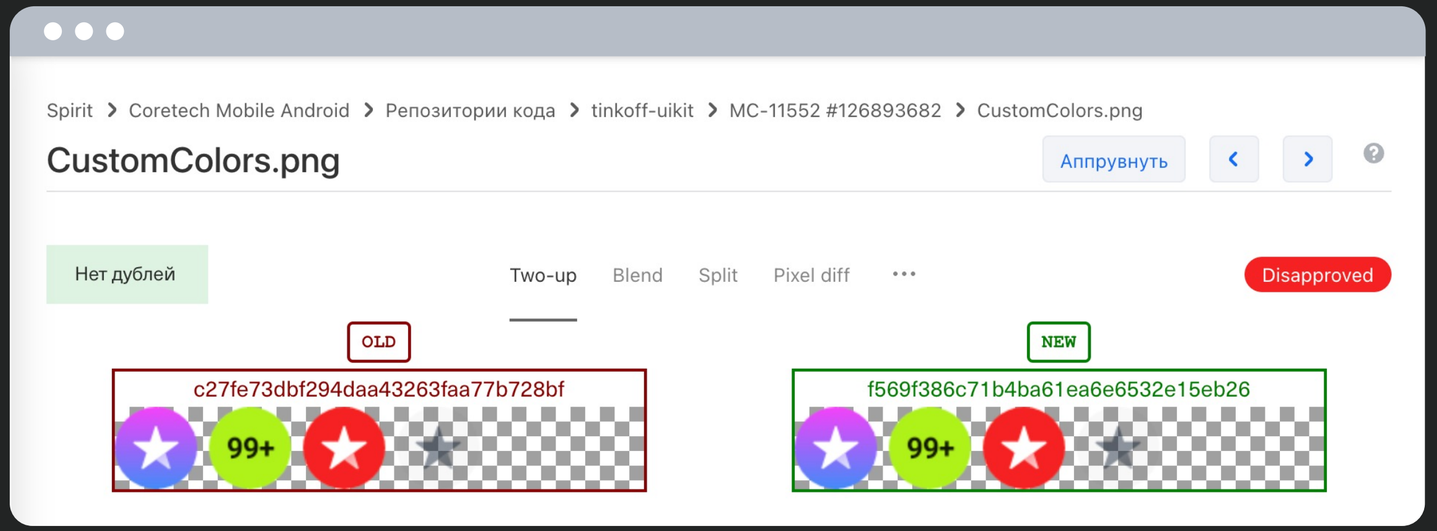
1. Разработчик изначально добавляет «сломанный» автотест – без эталона
2. Тест падает
3. Во время падения сохраняется актуальный снимок (failed_***.png)
4. Failed_снимок считается эталоном
5. Commit

Автоматизация CI: откроем дискуссию, когда тесты упали



Tooling CI: в браузере видно разницу





V3 тестирование

Плюсы

Интеграция с тулингом сравнения

Автозапись эталонов

Урок 4 – повышай удобство разработки автотестов

Недостатки V3

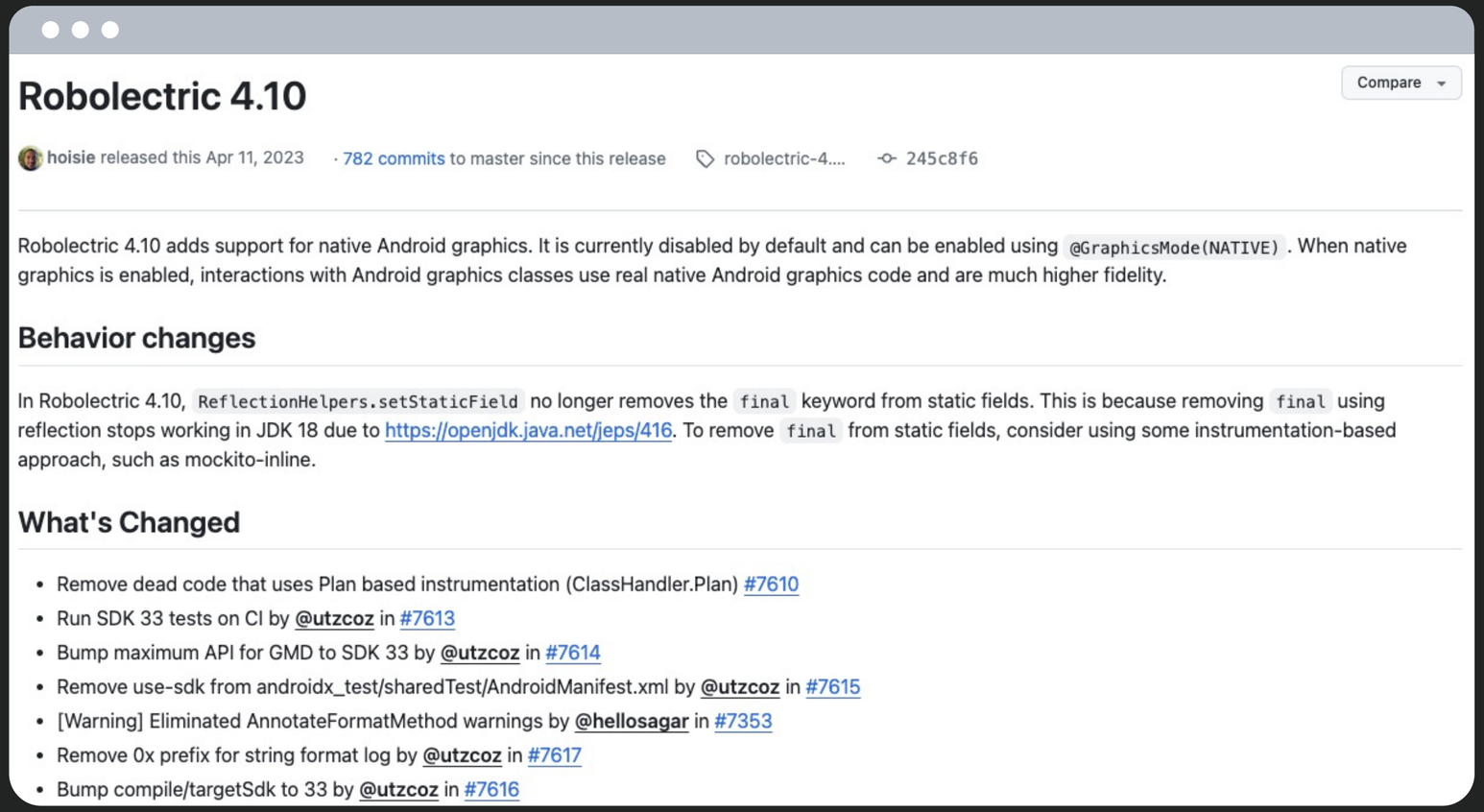
Минусы

Производительность на большом объеме тестов

Проблемы с эмуляцией

UI тесты без эмулятора?

Next Level Android Visual Testing



Robolectric 4.10

Compare

hoisie released this Apr 11, 2023 · 782 commits to master since this release · robolectric-4.... · 245c8f6

Robolectric 4.10 adds support for native Android graphics. It is currently disabled by default and can be enabled using `@GraphicsMode(NATIVE)`. When native graphics is enabled, interactions with Android graphics classes use real native Android graphics code and are much higher fidelity.

Behavior changes

In Robolectric 4.10, `ReflectionHelpers.setStaticField` no longer removes the `final` keyword from static fields. This is because removing `final` using reflection stops working in JDK 18 due to https://openjdk.java.net/jeps/416. To remove `final` from static fields, consider using some instrumentation-based approach, such as mockito-inline.

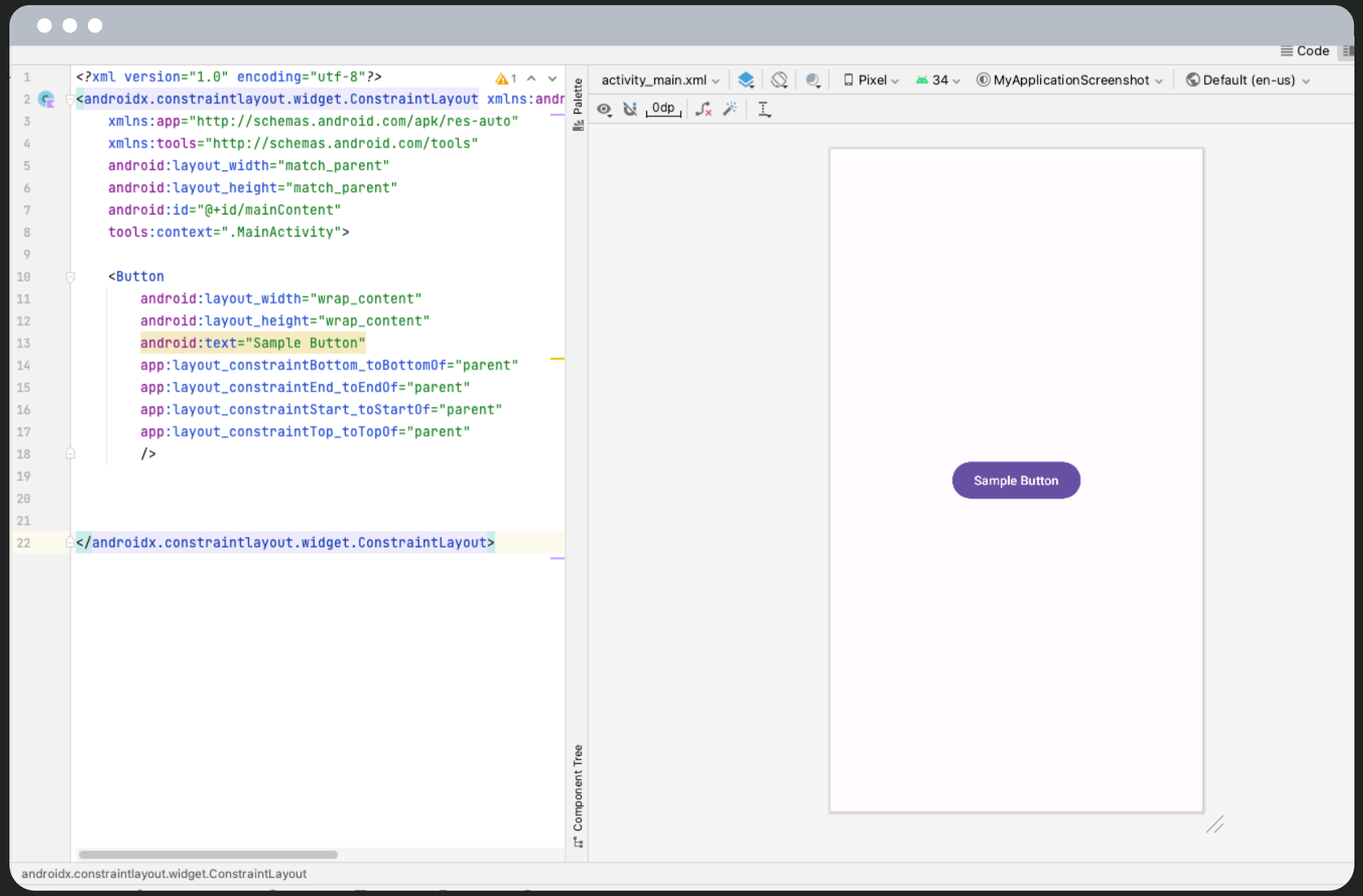
What's Changed

- Remove dead code that uses Plan based instrumentation (ClassHandler.Plan) #7610
- Run SDK 33 tests on CI by **@utzcoz** in #7613
- Bump maximum API for GMD to SDK 33 by **@utzcoz** in #7614
- Remove use-sdk from androidx_test/sharedTest/AndroidManifest.xml by **@utzcoz** in #7615
- [Warning] Eliminated AnnotateFormatMethod warnings by **@hellosagar** in #7353
- Remove 0x prefix for string format log by **@utzcoz** in #7617
- Bump compile/targetSdk to 33 by **@utzcoz** in #7616

Robolectric 4.10 adds support for native Android graphics. It is currently disabled by default and can be enabled using @GraphicsMode(NATIVE).

When native graphics is enabled, interactions with Android graphics classes use real native Android graphics
code and are much higher fidelity.

Android Studio Design Preview + layoutlib




```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:andr
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/mainContent"
    tools:context=".MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sample Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Пример теста на Robolectric (JVM)

- **Включаем GraphicsMode.Native**

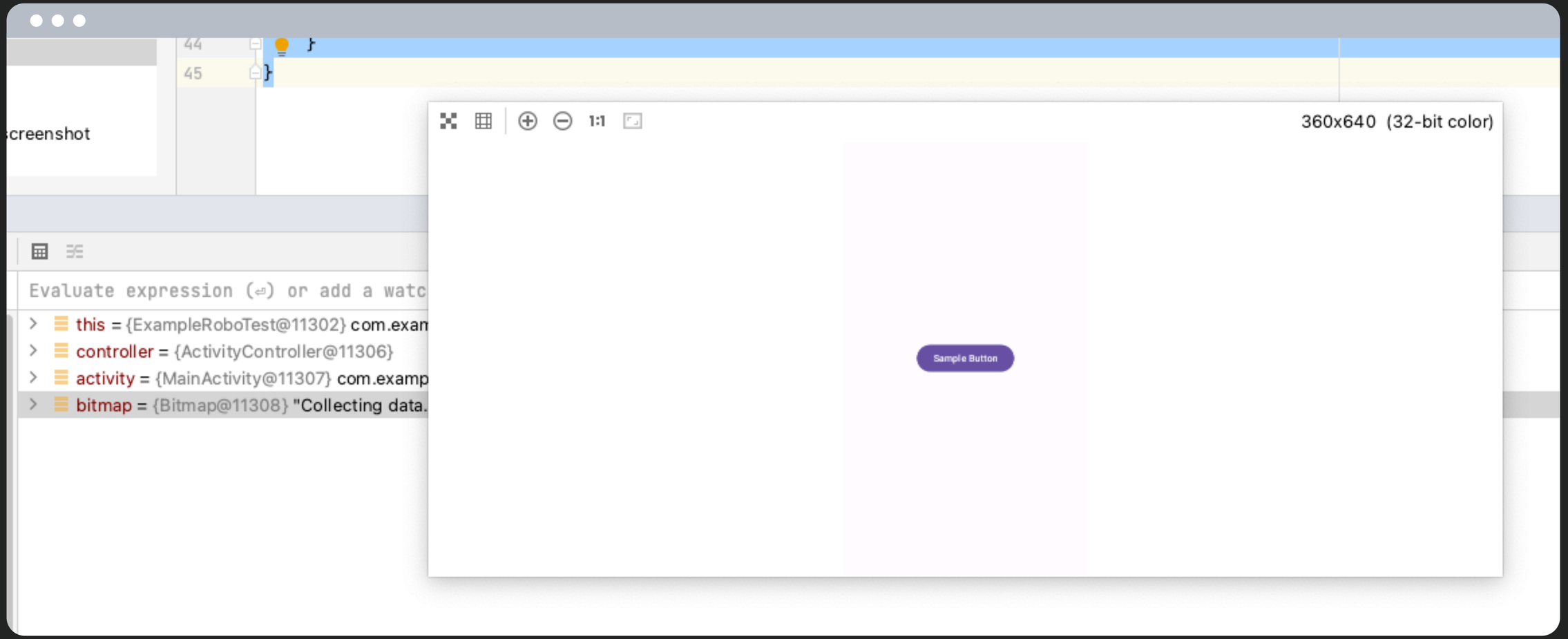
```
@RunWith(RobolectricTestRunner::class)
@GraphicsMode(GraphicsMode.Mode.NATIVE)
@Config(qualifiers = "w360dp-h640dp-mdpi")
class ExampleRobolectricScreenshotTest {
    @Test
    fun test() {
        Robolectric.buildActivity(MainActivity::class.java).use { controller ->
            controller.setup()
            val activity = controller.get()
            val bitmap = InstrumentationRegistry.getInstrumentation().uiAutomation.takeScreenshot()
            // compare bitmaps
        }
    }
}
```

Пример теста на Robolectric (JVM)

- **Включаем GraphicsMode.Native**
- **Снимаем скрин с Activity**

```
@RunWith(RobolectricTestRunner::class)
@GraphicsMode(GraphicsMode.Mode.NATIVE)
@Config(qualifiers = "w360dp-h640dp-mdpi")
class ExampleRobolectricScreenshotTest {
    @Test
    fun test() {
        Robolectric.buildActivity(MainActivity::class.java).use { controller ->
            controller.setup()
            val activity = controller.get()
            val bitmap = InstrumentationRegistry.getInstrumentation().uiAutomation.takeScreenshot()
            // compare bitmaps
        }
    }
}
```

Bitmap из Unit теста без эмулятора



V4 тестирование

Плюсы

Native Graphics

98% тестов без эмулятора

Ускорение прогона 1000 тестов x3

Меньше ресурсов CI

Урок 5 – предпочитай unit-тесты вместо инструментальных тестов на эмуляторе

Развиваем дальше?

V1
- Storybook
- Test provider

V2
- Custom Runner
- Отказ от Activity

V3
- Custom Runner Functions
- Autorecord

V4
- Unit Tests
- Robolectric Native Graphics

V5

Kotlin Symbol Processing

KSP генерирует автотесты

Добавляет сгенерированное в unitTest sourceSet

Мы расставляем аннотации

Как сейчас разрабатываем скриншот-тесты ?

Никак

Код тестов не пишем –
KSP их генерирует

Разметка историй для тестирования

Добавляем аннотацию @ScreenshotTesting

```
object ButtonStories {
    @ScreenshotTesting
    val primary: View by Story {
        //compose content
    }
}
```

Сгенерированные тесты

После сборки проекта в папке module/build/debugUnitTests

```
// Code generated unit test
@RunWith(RobolectricTestRunner::class)
@GraphicsMode(GraphicsMode.Mode.NATIVE)
@Config(qualifiers = "w360dp-h640dp-mdpi")
class ButtonGeneratedRobolectricScreenshotTest {
    @Test
    fun storyPrimaryTest() {
        val story = ButtonStories.primary
        assertCorrectScreenshot(story)
        // other logic..
    }
}
```

И снова повышаем удобство разработки автотестов (урок 4)

Все уроки вместе

01

Готовь структуру демо

02

Минимизируй
время прогона

03

Выбери CI
как источник правды

04

Повышай удобство
разработки

05

Предпочитай Unit < UI

Итоги

Robolectric Native Graphics закрывает большую часть кейсов скриншот-тестирования

Итоги

- Robolectric Native Graphics закрывает большую часть кейсов скриншот-тестирования
- Если его недостаточно, то придется глубоко изучить Android UI

Итоги

- Robolectric Native Graphics закрывает большую часть кейсов скриншот-тестирования
- Если его недостаточно, то придется глубоко изучить Android UI
- Разработка фреймворка тестирования обеспечила масштабирование ДС

Итоги

- Robolectric Native Graphics закрывает большую часть кейсов скриншот-тестирования
- Если его недостаточно, то придется глубоко изучить Android UI
- Разработка фреймворка тестирования обеспечила масштабирование ДС
- Нет проблем поддержки скриншот-тестов, когда тулинг удобный

ТИНЬКОФФ