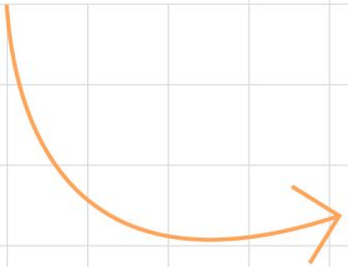
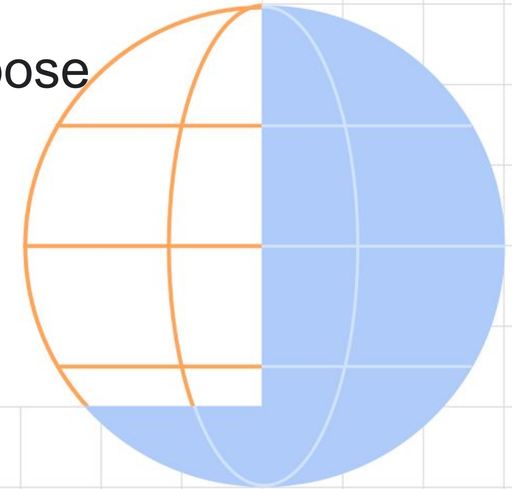
Google Developers

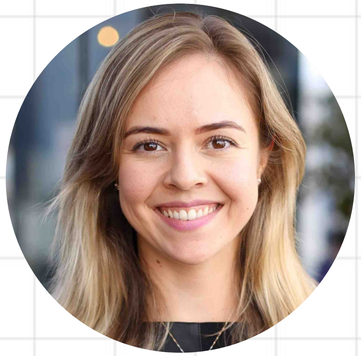


A importância dos testes de UI para o desenvolvimento Android

Desde Espresso até testes para Jetpack Compose



Apresentações



Angélica Oliveira
Developer, ThoughtWorks
@AngOliveiraa

Aline Ayres
Developer, ThoughtWorks
@missayres

Qualidade Mobile

Quem nunca participou do desenvolvimento de uma app onde...

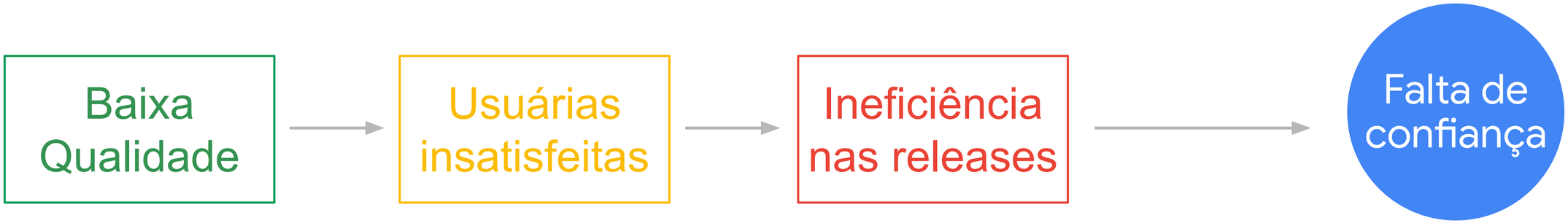
Alguns passos relacionados a qualidade foram "despriorizados" para ser possível "entregar" determinadas funcionalidades?

Experts

"imagem: Freepik.com". Esta imagem foi obtida usando recursos do Freepik.com

Consequências dessa abordagem

Pequeno resumo



Além disso...

Acabamos vendo que essa despriorização também acaba por afetar a comunidade de pessoas desenvolvedoras Mobile

- Muitas pessoas não analisam o impacto de sua arquitetura e a testabilidade de suas aplicações
- Raramente vemos a prática TDD sendo aplicada no contexto Mobile
- Testes escritos pela cobertura, e a qualidade?
- Conhecimento sobre testes de UI pouco disseminados

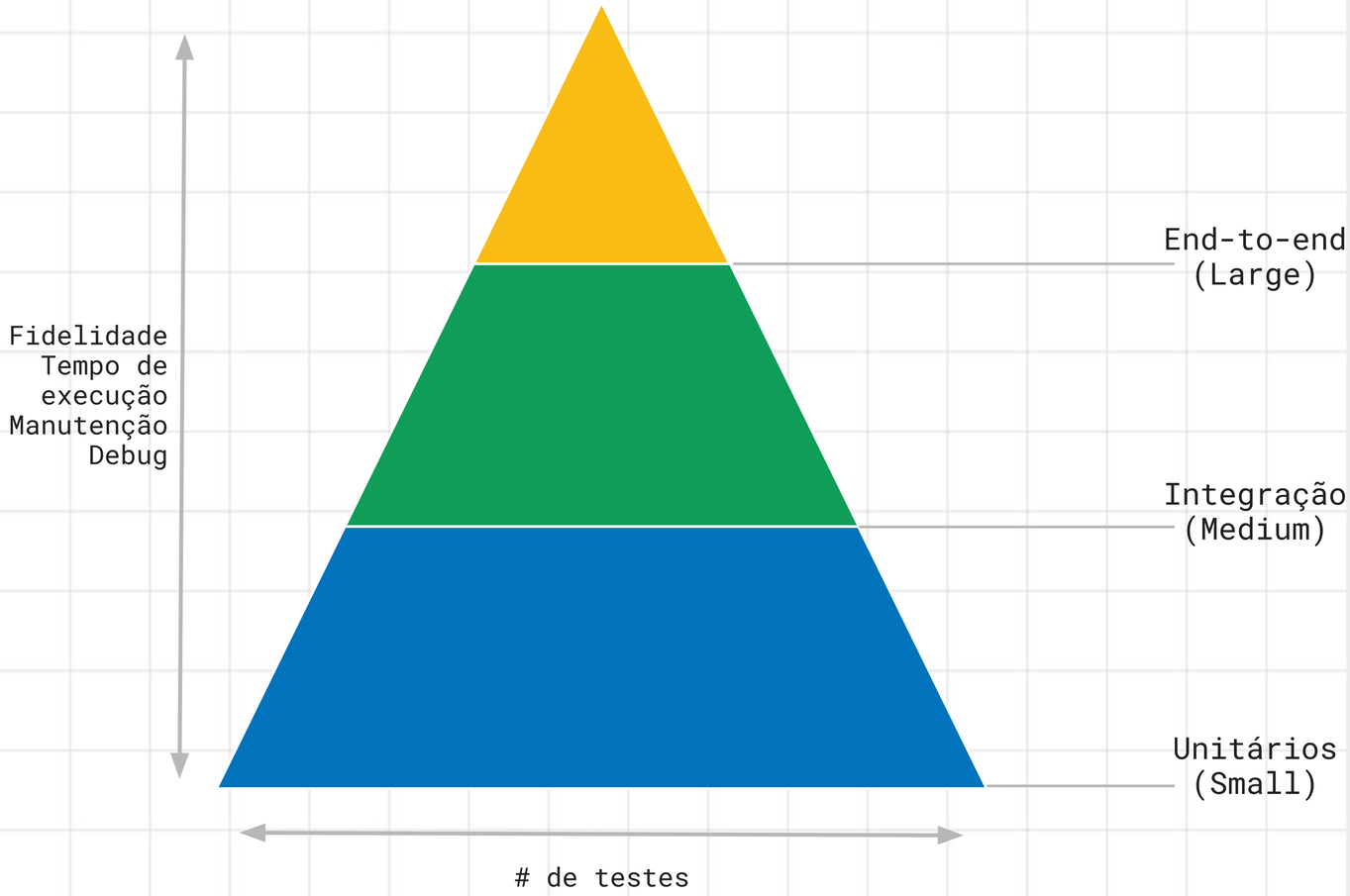

Interesting
GIFSec.com

Pirâmide e estratégia de testes

Existem vários tipos de teste

O que varia entre eles é a fidelidade e o custo de implementação / execução

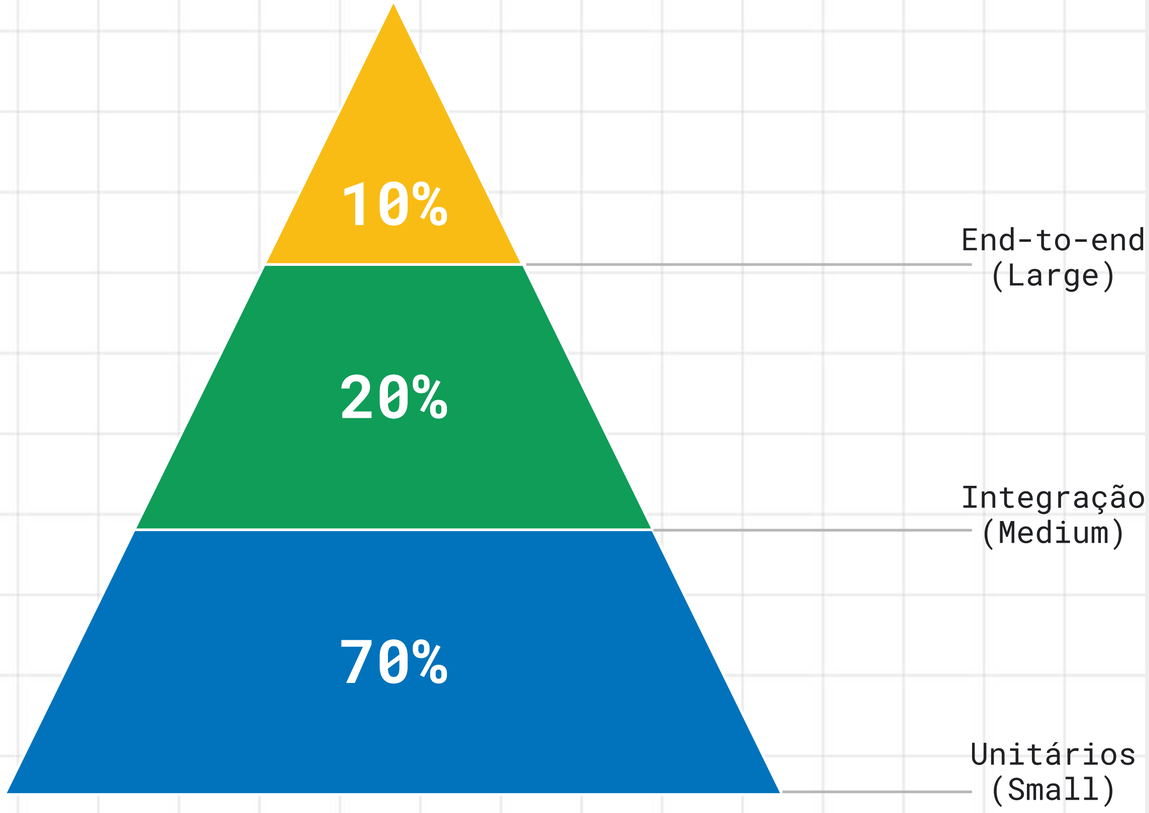
Idealmente devemos ter mais testes de baixo custo



Experts

O número de testes em cada camada pode variar

A recomendação dada pela documentação oficial do Android é:



A importância dos testes de UI

Qual a importância do testes de UI automatizados?

- Testes automatizados de UI podem garantir que o app testado atenda aos requisitos funcionais desejados
- Manter um alto padrão de qualidade da app
- Podem ser executados de maneira rápida, confiável e repetitiva
- Identificar possíveis erros mais rapidamente e corrigi-los

Onde executamos os testes?

CI/CD

Code Checkout	Bundle install	Install dependencies	Check PNG Files	Build	Lint	Unit Test	Instrumented UI Test
42s	8s	52s	298ms	16min 10s	3min 17s	3min 44s	22min 55s
42s	8s	52s	298ms	16min 10s	3min 17s	3min 44s	22min 55s

Experts


[adult swim]
That was all a test, Morty.

Principais Frameworks de teste no Android

Espresso

Permite criar testes concisos, bonitos e confiáveis para a Ui do Android

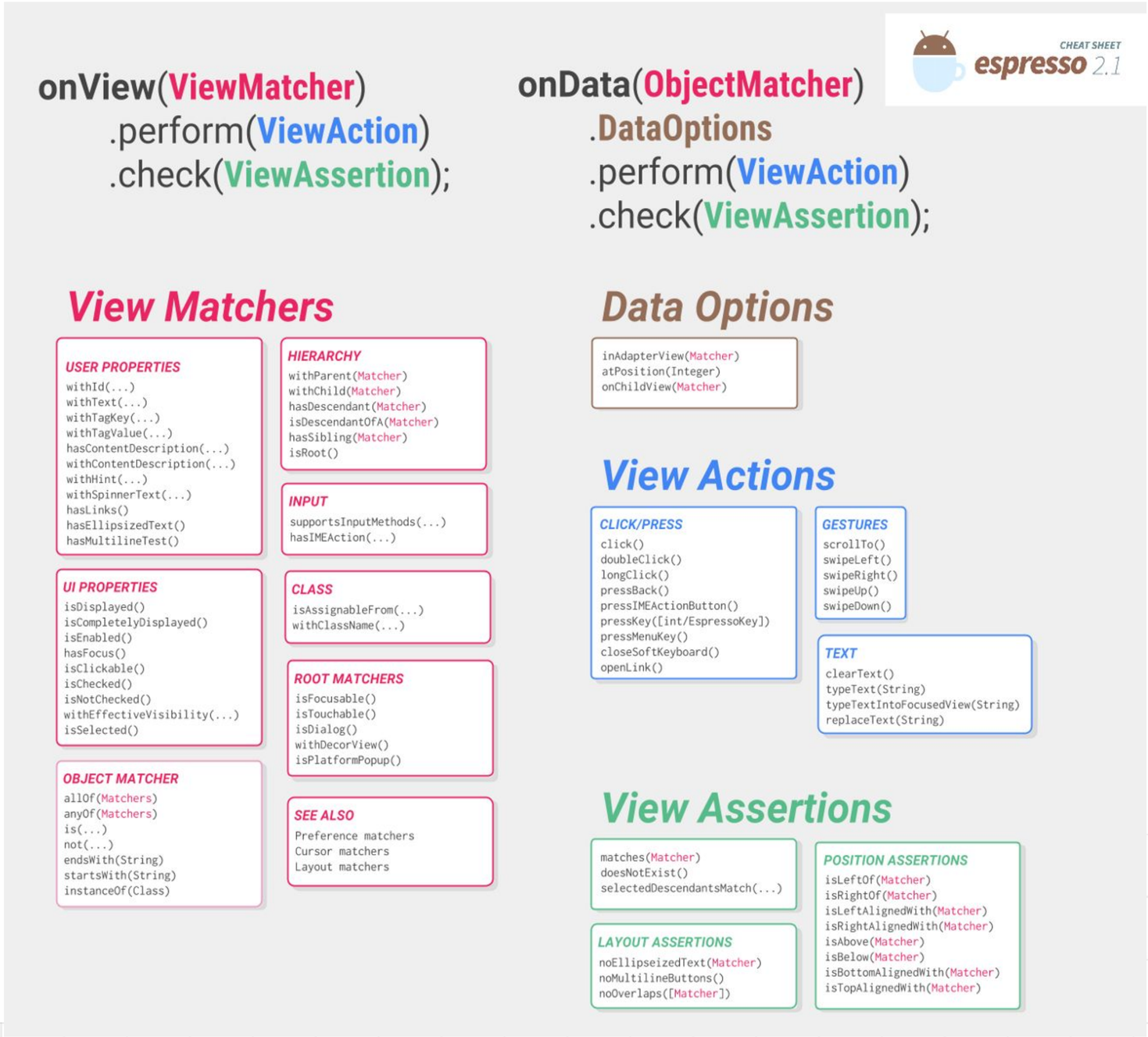
Espresso

Testes de UI que precisam rodar em um device real ou um emulador com um escopo isolado.

- Sua API principal é pequena e previsível, também pode ser personalizada
- É de fácil aprendizado
- São executados de maneira idealmente rápida
- Testa com clareza expectativas, interações e declarações de estado
- Pode ser usado para testes de caixa preta, mas tem maior potencial por aqueles que conhecem o codebase

Espresso

Folha de referência



CHEAT SHEET
espresso 2.1

```
onView(ViewMatcher)
    .perform(ViewAction)
    .check(ViewAssertion);
```

```
onData(ObjectMatcher)
    .DataOptions
    .perform(ViewAction)
    .check(ViewAssertion);
```

View Matchers

USER PROPERTIES
withId(...)
withText(...)
withTagKey(...)
withTagValue(...)
hasContentDescription(...)
withContentDescription(...)
withHint(...)
withSpinnerText(...)
hasLinks()
hasEllipsizedText()
hasMultilineTest()

HIERARCHY
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()

INPUT
supportsInputMethods(...)
hasIMEAction(...)

UI PROPERTIES
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility(...)
isSelected()

CLASS
isAssignableFrom(...)
withClassName(...)

ROOT MATCHERS
isFocusable()
isTouchable()
isDialog()
withDecorView()
isPlatformPopup()

OBJECT MATCHER
allOf(Matchers)
anyOf(Matchers)
is(...)
not(...)
endsWith(String)
startsWith(String)
instanceOf(Class)

SEE ALSO
Preference matchers
Cursor matchers
Layout matchers

Data Options

inAdapterView(Matcher)
atPosition(Integer)
onChildView(Matcher)

View Actions

CLICK/PRESS
click()
doubleClick()
longClick()
pressBack()
pressIMEActionButton()
pressKey([int/EspressoKey])
pressMenuKey()
closeSoftKeyboard()
openLink()

GESTURES
scrollTo()
swipeLeft()
swipeRight()
swipeUp()
swipeDown()

TEXT
clearText()
typeText(String)
typeTextIntoFocusedView(String)
replaceText(String)

View Assertions

matches(Matcher)
doesNotExist()
selectedDescendantsMatch(...)

LAYOUT ASSERTIONS
noEllipseizedText(Matcher)
noMultilineButtons()
noOverlaps([Matcher])

POSITION ASSERTIONS
isLeftOf(Matcher)
isRightOf(Matcher)
isLeftAlignedWith(Matcher)
isRightAlignedWith(Matcher)
isAbove(Matcher)
isBelow(Matcher)
isBottomAlignedWith(Matcher)
isTopAlignedWith(Matcher)

```
@Test
fun greeterSaysHello() {
        onView(withId(R.id.name_field)).perform(typeText("TDC"))
        onView(withId(R.id.greet_button)).perform(click())
        onView(withText("Hello TDC!")).check(matches(isDisplayed()))
}
```

UI Automator

É um framework de testes de IU recomendado para testes funcionais de IU entre apps instalados e do sistema

UI Automator

Framework de testes de UI que nos permite criar testes que vai além da aplicação

- Ferramenta para criação de testes end-to-end
- Permite criar interações com escopo mais abrangente, como configuração do device e outras aplicações
- São testes mais lentos
- É adequado para testes caixa preta, em que o código do teste não se baseia em detalhes internos de implementação do app

UI Automator

Principais recursos do framework UI Automator

- `uiautomatorviewer` - ferramenta para inspecionar a hierarquia de layouts e ver as propriedades de componentes de UI
- `UiDevice` - uma API para recuperar informações de estado e realizar operações no dispositivo
- APIs compatíveis com testes de UI entre apps (`UiObject`, `UiSelector`, `UiScrollable`, `UiCollection`, `Configurator`)

```
 device = UiDevice.getInstance(getInstrumentation())
 device.pressHome()

// Bring up the default launcher by searching for a UI component
// that matches the content description for the launcher button.
val allAppsButton: UIObject = device.findObject(
       UiSelector().description("Apps"))

// Perform a click on the button to load the launcher.
allAppsButton.clickAndWaitForNewWindow()
```

Robolectric

Testes de UI que podem rodar sem termos que executá-los em um emulador.

Robolectric

Testes de UI que podem rodar sem termos que executá-los em um emulador.

- O framework possui classes Shadow, que replicam o framework Android
- Execução de emulador e instalação desnecessárias
- Podemos executar os testes em infra que não possuem emulador
- Maior rapidez na execução

```kotlin
@RunWith(RobolectricTestRunner::class)
class MobileTrackActivityTest {
    @Test
    fun mobileTrackActivity_shouldShowMobileTrackData() {
        val activity = buildActivity(MobileTrackActivity::class.java).setup()

        val textMobile = activity.get().findViewById<TextView>(R.id.text_mobile)
        val imageBannerMobile = activity.get().findViewById<ImageView>(R.id.image_banner_mobile)

        assertEquals("Vejam os speakers para esse evento incrível!", textMobile.text)
        assertEquals(R.drawable.tdc_innovation_mobile,
            shadowOf(imageBannerMobile.drawable).createdFromResId)
        ...
    }
}
```

Jetpack Compose Tests

Jetpack Compose é a nova forma de se escrever UI no Android, seus testes utilizam semântica para serem executados

Experts

Jetpack Compose Tests

Utilização de semântica para a execução dos testes de UI.

- Com Jetpack Compose, não temos elementos de `View`
- Funções compostas que emitem UI
- As funções não possuem ID
- A semântica dá um significado para uma parte da UI

```
@get:Rule
val composeTestRule = createAndroidComposeRule<MainActivity>()
```

```
fun checkTDCHeaderElements(): Unit = with(composeTestRule) {
    setContent {
        TestsPlaygroundTheme {
            TDCHeader(Modifier)
        }
    }

    onNodeWithText("Trilha Mobile").assertIsDisplayed()
    onNodeWithText("Veja aqui as talks dessa trilha!").assertIsDisplayed()
    onNodeWithContentDescription("Imagem com ícone TDC").assertIsDisplayed()
}
```

Links e referências

Algumas referências

Links úteis:

- Conceitos básicos de testes
  - https://developer.android.com/training/testing/fundamentals
- Espresso
  - https://developer.android.com/training/testing/espresso/
- Ui Automator
  - https://developer.android.com/training/testing/ui-automator
- Robolectric
  - http://robolectric.org/
- Como testar o layout do Compose
  - https://developer.android.com/jetpack/compose/testing
- Códigos de exemplo de vários frameworks de teste
  - https://github.com/android/testing-samples

Google Developers

Obrigada!

Angélica Oliveira
@AngOliveiraa

Aline Ayres
@missayres