
FlutterRS

PRECISAMOS FALAR SOBRE TESTES

E nossa responsabilidade como programadores

TÓPICOS

- Motivos para não testar
- Introdução ao teste de software
- Por onde começar
- Testes com Flutter e Dart

“

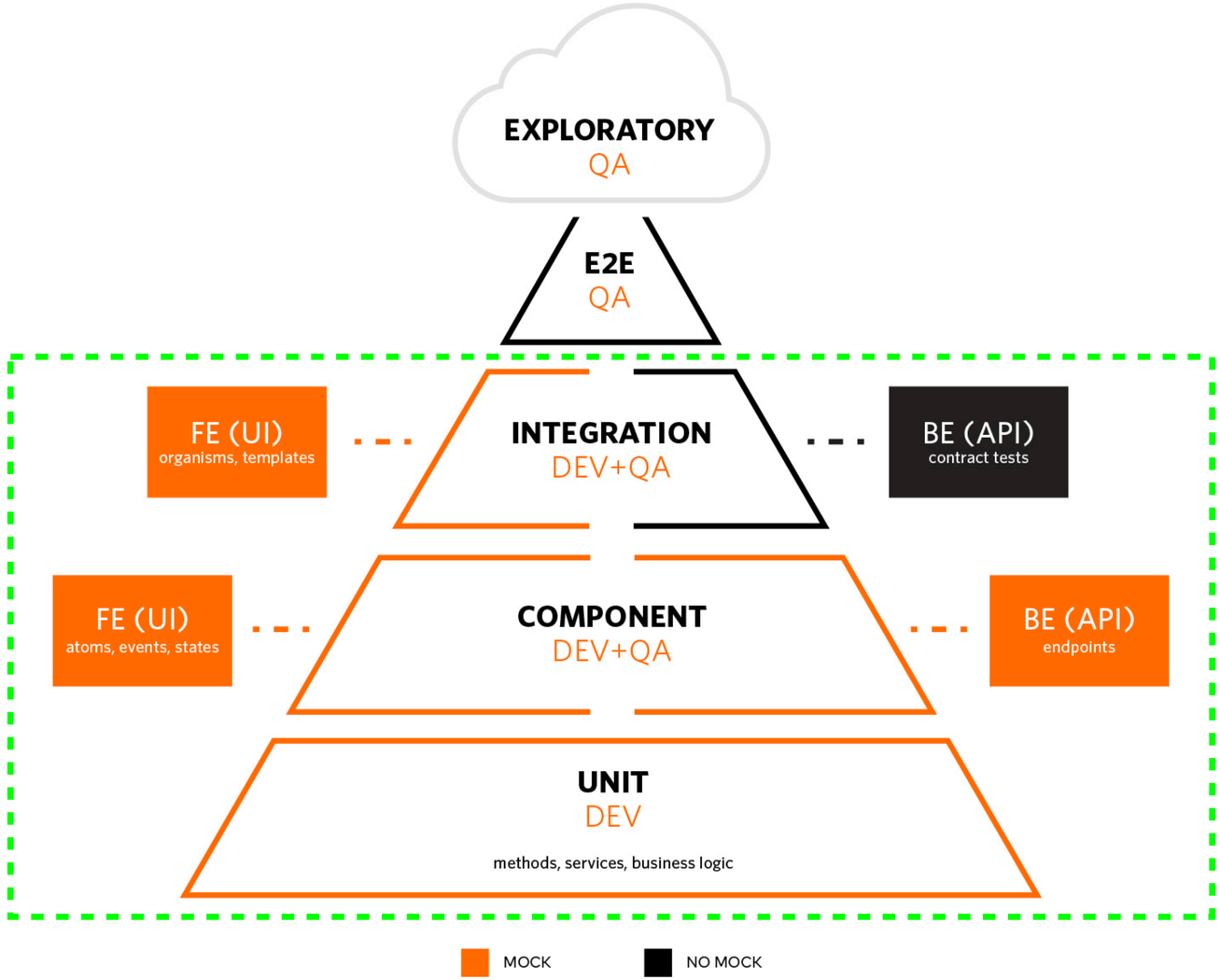
Nos dias de hoje, é irresponsabilidade um programador publicar uma linha de código que não tenha sido coberta por um teste unitário

-Robert C. Martin

MOTIVOS PARA NÃO TESTAR

- É perda de tempo
- É trabalho do testador
- Meu prazo é curto
- É só uma POC (Prova de Conceito)
- O cliente não se importa

INTRODUÇÃO AO TESTE DE SOFTWARE



POR ONDE COMEÇAR?

- Arquitetura deve facilitar os testes
- Siga os princípios do S.O.L.I.D.
- Pratique TDD

TESTES COM FLUTTER E DART

Vamos ao que interessa

TESTES UNITÁRIOS

- https://pub.dev/packages/test
- Testes são declarados em funções `` `test()` ``
- Asserções são feitas com `` `expect()` ``
- Testes podem ser agrupados com `` `group()` ``
- Algumas funções de comparação:
  - `equals()`
  - `contains()`
  - `isTrue / isFalse`

https://pub.dev/documentation/matcher/latest/matcher/matcher-library.html

TESTES UNITÁRIOS

```dart
class Counter {
  var value = 0;

  void increment() => value++;

  void decrement() => value--;
}
```

```dart
import 'package:flutter_test_demo/counter/counter.dart';
import 'package:test/test.dart';

void main() {
  group('Counter Test', () {
    test('Valor deve ser incrementado', () {
      final counter = Counter();

      counter.increment();

      expect(counter.value, 1);
    });

    test('Valor deve ser decrementado', () {
      final counter = Counter();

      counter.decrement();

      expect(counter.value, -1);
    });
  });
}
```

POR QUE USAR MOCKS?

- Dependências podem interferir no resultado dos testes
- Dependências podem deixar os testes lentos
- É difícil simular variações de cenários sem controle sobre as dependências

TESTES UNITÁRIOS

```dart
class CounterExtended implements Counter {
  final IncrementFactor _incrementFactor;
  var value = 0;

  CounterExtended(this._incrementFactor);

  @override
  void increment() =>
      value += _incrementFactor.getFactor();

  @override
  void decrement() =>
      value -= _incrementFactor.getFactor();
}
```

```dart
///Testes Unitários COM dependências

class _MockIncrementFactor extends Mock
                    implements IncrementFactor {}

void main() {
  test('Quando fator for 17 deve adicionar 17 ao valor',(){
    ///Instanciando Mock
    final factor = _MockIncrementFactor();

    ///Configurando o retorno esperado para o mock
    when(factor.getFactor()).thenReturn(17);

    /// Utilizando o mock para suprir a
    /// dependência da nossa classe
    final counter = CounterExtended(factor);

    counter.increment();

    expect(counter.value, 17);
  });
}
```

TESTES DE WIDGETS

- Testa um único Widget
- Verifica se o Widget aparece e se comporta da maneira desejada
- Depende do pacote `flutter_test`, já presente na instalação do Flutter:
  - WidgetTester para construir e interagir com Widgets
  - testWidgets() substitui o `test()` e cria o WidgetTester
  - Finder para encontrar Widgets
  - Matchers específicos de Widgets

TESTES DE WIDGET

```dart
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter_test_demo/counter/counter.dart';
import 'package:flutter_test_demo/main.dart';
import 'package:mockito/mockito.dart';

void main() {
  testWidgets('Tela deve ler valor do contador',
              (WidgetTester tester) async {
    ///Configurando nosso Mock
    final counter = _MockCounter();
    when(counter.value).thenReturn(37);

    // Build our app and trigger a frame.
    await tester.pumpWidget(
      MaterialApp(
        home: MyHomePage(title: 'Title Test',
                         counter: counter),
      ),
    );

    // Verify that our counter starts at 0.
    expect(find.text('37'), findsOneWidget);
  });
}

class _MockCounter extends Mock implements Counter {}
```

TESTES DE INTEGRAÇÃO

- Testa um app completo ou grandes pedaços do app
- O objetivo principal é verificar se os widgets funcionam em conjunto como esperado
- Geralmente roda em um dispositivo físico ou em emuladores

TESTES DE INTEGRAÇÃO

```
void main() {
  group('Counter App', () {
    final counterTextFinder = find.byValueKey('counter');
    final buttonFinder = find.byValueKey('increment');

    FlutterDriver driver;

    setUpAll(() async {
      driver = await FlutterDriver.connect();
    });

    tearDownAll(() async {
      if (driver != null) {
        driver.close();
      }
    });

    test('starts at 0', () async {
      // Use the `driver.getText` method to verify the counter
starts at 0.
      expect(await driver.getText(counterTextFinder), "0");
    });

    test('increments the counter', () async {
      // First, tap the button.
      await driver.tap(buttonFinder);

      // Then, verify the counter text is incremented by 1.
      expect(await driver.getText(counterTextFinder), "1");
    });
  });
}
```

INTEGRAÇÃO CONTÍNUA

- Execução de testes em ambiente neutro
- Pare de quebrar a master
- Reduz o tempo de Code Review
- Algumas ferramentas
  - Gitlab CI
  - Bitrise
  - Codemagic
  - Circle CI
  - Bitbucket Pipelines
  - Github Actions

DÚVIDAS?

João Rutkoski

Email: *joaortk@gmail.com*

Slack Flutter: *http://flutterbr.herokuapp.com/*

Slack Android: *http://slack.androiddevbr.org/*

LinkedIn: *https://www.linkedin.com/in/joaortk/*

Obrigado!