LeanCode

Programming Mobile Applications in Flutter

Testing

What is a Hook?

Testing

A method to check whether the actual software product matches expected requirements and to ensure that software product is defect free

Types

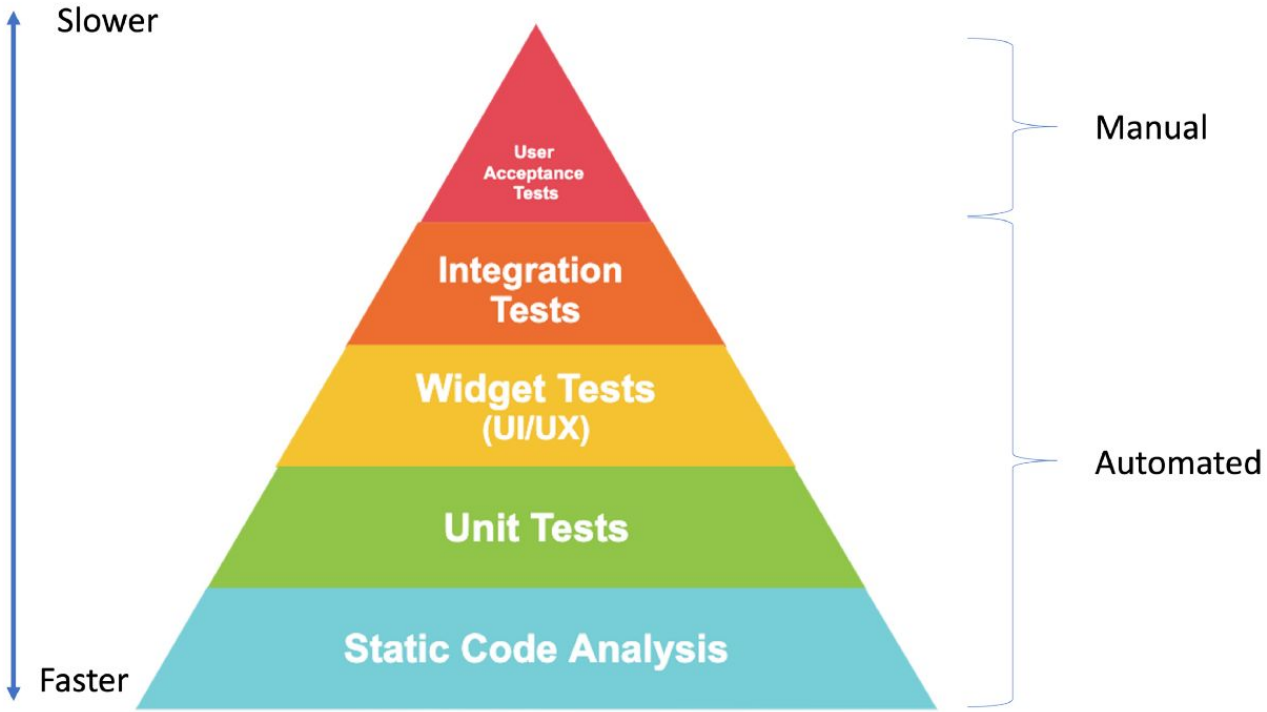
- Functional
  - Unit
  - Integration
  - User Acceptance
  - Localization
  - …
- Non-Functional
  - Performance
  - Scalability
  - …
- Maintenance
  - Regression

White Box Testing

A software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers. One of the goals of **White Box Testing** in software engineering is to verify all the decision branches, loops, statements in the code.

Black Box Testing

A software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. **Black Box Testing** mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as **Behavioral Testing**.



Source: https://njkhanh.com/understand-the-core-concept-of-test-in-flutter-to-write-a-fewer-errors-of-flutter-app-p5f34353335

Static Code Analysis

```
void sampleFunc1(dynamic data) {

}


void sampleFunc2(String data) {

}
```

Static Code Analysis

- Use language features to prevent bugs
- Implement static analysis - Flutter lints
- Annotations for Static Analysis - Meta

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),  // ThemeData
    home: Sample1(),
  );  // MaterialApp
}
```

Unit Tests

Unit tests are handy for verifying the behavior of a single function, method, or class.

Unit Tests

1. Add the test or flutter_test dependency
2. Create a test file
3. Create a class to test
4. Write a test for our class
5. Combine multiple tests in a group
6. Run the tests

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

  void increment() => value++;

  void decrement() => value--;
}

void main() {
  group('Counter', () {
    test('value should start at 0', () {
      expect(Counter().value, 0);
    });
  });
}
```

Unit Tests Matchers

- *equals* - Returns a matcher that matches if the value is structurally equal
- *completion* - Matches a **Future** that completes successfully with any value
- *throwsA* - Checks if an exception is thrown
- *emitsInOrder* - Returns a **StreamMatcher** that matches the stream if each matcher in **matchers** matches, one after another
- *emitsLater* - Just like **expect**, but returns a **Future** that completes when the matcher has finished matching
- *…*

Demo

Unit Tests Rules

- Short, quick, and automated
- They test specific functionality of a method or class
- Have a clear **pass/fail** condition

```dart
class UserRepository {

  final _emailValidator = EmailValidator();

  bool hasValidData(User user) {
    return _emailValidator.isValid(user.email);
  }

}

class EmailValidator {

  final _regex = RegExp(r'\w+@\w+\.\w+');

  bool isValid(String email) {
    return _regex.hasMatch(email);
  }
}

class User {
  String email;

  User(this.email);
}
```

```dart
void main() {
  group('User Repository', () {
    test('validation is correct for corrupted data', () {
      final UserRepository repository = UserRepository();

      expect(repository.hasValidData(User("a")), false);
    });

    test('validation is correct for data', () {
      final UserRepository repository = UserRepository();

      expect(repository.hasValidData(User("a@a.com")), true);
    });

  });
}
```

Don’t test email validator inside user repository!!!

How to fix that?

Mocks

Mocks

- Mocks - simulated objects that mimic the behavior of real objects in controlled ways
- Stub - a piece of code used to stand in for some other programming functionality. A stub may simulate the behavior of existing code
- MockTail
- Mockito

```dart
class EmailValidatorMock extends Mock implements EmailValidator {}
```

How should we modify UserRepository to pass mocked validator?

```dart
class UserRepository2 {

  late EmailValidator emailValidator;

  bool hasValidData(User user) {
    return emailValidator.isValid(user.email);
  }
}
```

```dart
class UserRepository2 {

  @visibleForTesting
  late EmailValidator emailValidator;

  bool hasValidData(User user) {
    return emailValidator.isValid(user.email);
  }
}
```

```dart
class UserRepository2 {

  UserRepository2(this._emailValidator);

  final EmailValidator _emailValidator;

  bool hasValidData(User user) {
    return _emailValidator.isValid(user.email);
  }

}
```

Dependency Injection

```dart
void main() {
  group('User Repository', () {
    late EmailValidatorMock emailValidatorMock;
    late UserRepository2 repository;

    setUp(() {
      emailValidatorMock = EmailValidatorMock();
      repository = UserRepository2(emailValidatorMock);
    });

    tearDown(() {});

    test('validation is correct for corrupted data', () {
      when(() => emailValidatorMock.isValid(any())).thenReturn(false);

      repository.hasValidData(User("a"));

      verify(() => emailValidatorMock.isValid(any())).called(1);
    });
  });
}
```

```dart
void main() {
  group('User Repository', () {
    late EmailValidatorMock emailValidatorMock;
    late UserRepository2 repository;

    setUp(() {
      emailValidatorMock = EmailValidatorMock();
      repository = UserRepository2(emailValidatorMock);
    });

    tearDown(() {});

    test('validation is correct for corrupted data', () {
      when(() => emailValidatorMock.isValid("a")).thenReturn(false);

      repository.hasValidData(User("a"));

      verify(() => emailValidatorMock.isValid("a")).called(1);
    });
  });
}
```

Demo

Bloc testing

```dart
abstract class CounterEvent {}

class CounterIncrementPressed extends CounterEvent {}

class CounterDecrementPressed extends CounterEvent {}

class CounterBloc extends Bloc<CounterEvent, int> {
  CounterBloc() : super(0) {
    on<CounterIncrementPressed>((event, emit) => emit(state + 1));
    on<CounterDecrementPressed>((event, emit) => emit(state - 1));
  }
}
```

Source: https://bloclibrary.dev/#/testing

```dart
void main() {
  group('CounterBloc', () {
    late CounterBloc counterBloc;

    setUp(() {
      counterBloc = CounterBloc();
    });

    test('initial state is 0', () {
      expect(counterBloc.state, 0);
    });

    blocTest(
      'emits [1] when CounterIncrementPressed is added',
      build: () => counterBloc,
      act: (CounterBloc bloc) => bloc.add(CounterIncrementPressed()),
      expect: () => [1],
    );

    blocTest(
      'emits [-1] when CounterDecrementPressed is added',
      build: () => counterBloc,
      act: (CounterBloc bloc) => bloc.add(CounterDecrementPressed()),
      expect: () => [-1],
    );
  });
}
```

Source: https://bloclibrary.dev/#/testing

```dart
abstract class CounterEvent {}
class CounterIncrementPressed extends CounterEvent {}
class CounterDecrementPressed extends CounterEvent {}

class CounterBloc extends Bloc<CounterEvent, int> {
  final SampleBloc sampleBloc;
  CounterBloc(this.sampleBloc) : super(0) {
    sampleBloc.stream.listen((event) {
      add(CounterIncrementPressed());
    });
    on<CounterIncrementPressed>((event, emit) => emit(state + 1));
    on<CounterDecrementPressed>((event, emit) => emit(state - 1));
  }
}


abstract class SampleEvent {}
class TestSampleEvent extends SampleEvent {}

class SampleBloc extends Bloc<SampleEvent, int> {
  SampleBloc() : super(0) {
    on<TestSampleEvent>((event, emit) => emit(state + 5));
  }
}
```

```dart
class SampleBlocMock extends MockBloc<SampleEvent, int> implements SampleBloc {}

void main() {
  group('CounterBloc', () {
    late SampleBlocMock sampleBlocMock;

    setUp(() {
      sampleBlocMock = SampleBlocMock();
    });

    blocTest(
      'emits new state when SampleBloc emits state',
      seed: () => 2,
      setUp: () {
        whenListen(sampleBlocMock, Stream.fromIterable([5, 6]));
      },
      build: () => CounterBloc(sampleBlocMock),
      expect: () => [3, 4],
    );
  });
}
```

TDD

A software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases

BDD

BDD

Behavior-driven development is a testing practice that follows the idea of specification by example. The idea is to describe how the application should behave in a very simple user/business-focused language. **BDD's** business-focused perspective on application behavior allows teams to create living documentation that is easy to maintain and can be consumed by all team members, including testers, developers, and product owners.

BDD

With BDD, tests are created using the gherkin **Given-When-Then** language:

- Given (some context)
- When (something happens)
- Then (outcome)

For example:

- **Given** I am signing up for a free trial
- **When** I submit the required details
- **Then** I receive a link to the download

```
test('Given valid email When validating user data Should call email validator', () {
  //given
  when(() => emailValidatorMock.isValid("a@a.com")).thenReturn(false);

  //when
  repository.hasValidData(User("a@a.com"));

  //then
  verify(() => emailValidatorMock.isValid("a@a.com")).called(1);
});
```

Questions?