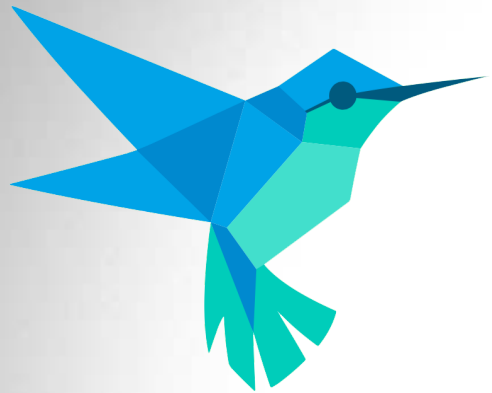


Tests

CIKLUM
EMPOWERING COLLABORATION


Shows presence of errors
Early testing
Exhaustive testing is impossible
Defect clustering
Pesticide paradox
Testing is context dependent
Absence of errors fallacy
SOFTWARE TESTING

QA Engineer

History of Software Testing

1960s - 1980s

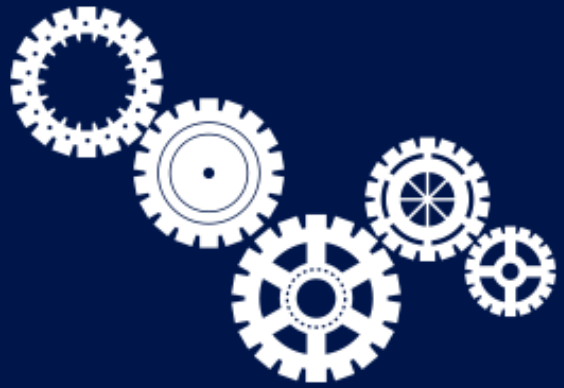
Constraint

1990s

Need

2000+

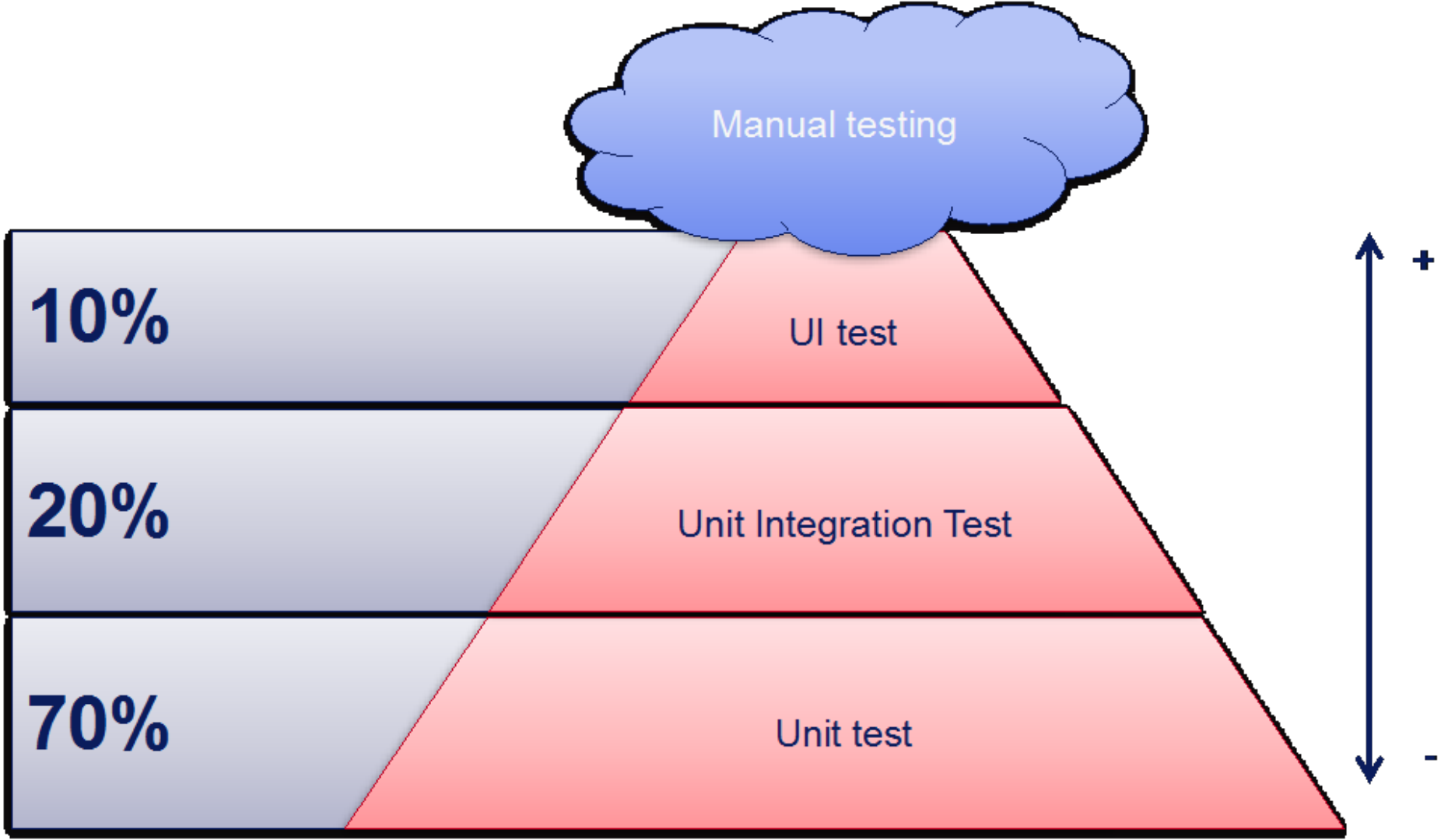
Asset



Automation

Vs

Manual


Manual testing
10%
UI test
20%
Unit Integration Test
70%
Unit test
+
-

UNIT TESTING is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

eat

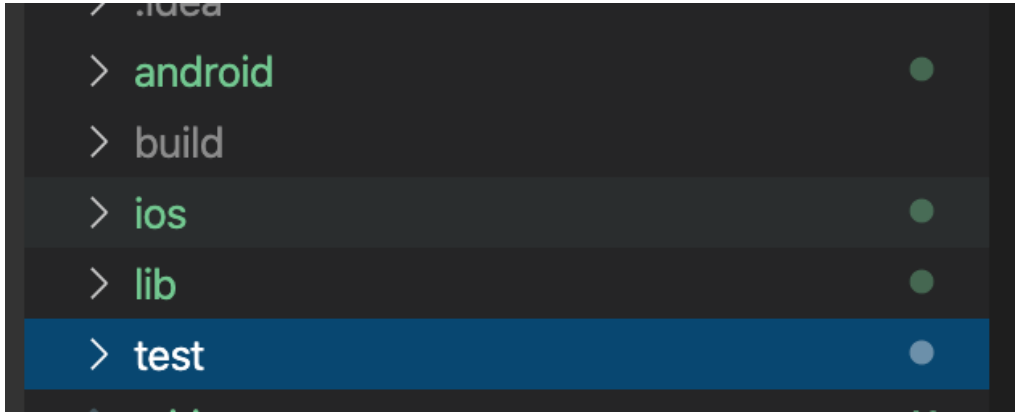
sleep

code

unit test

repeat

Flutter and Unit tests




```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

```dart
class GreatMathLibrary {

  static int sum(int a, int b) {
    return a + b;
  }

}
```

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:tests_example/math/great_math_library.dart';

void main() {
  Run | Debug
  test('Sum of numbers should be correct', () {
    final result = GreatMathLibrary.sum(2, 2);
    expect(result, 4);
  });
}
```

```
✓ Sum of numbers should be correct
Exited
```

```
3
4   void main() {
      Run | Debug
5     test('Sum of numbers should be correct', () {
6       final result = GreatMathLibrary.sum(2, 2);
7       expect(result, 5);
8     });
9   }
```

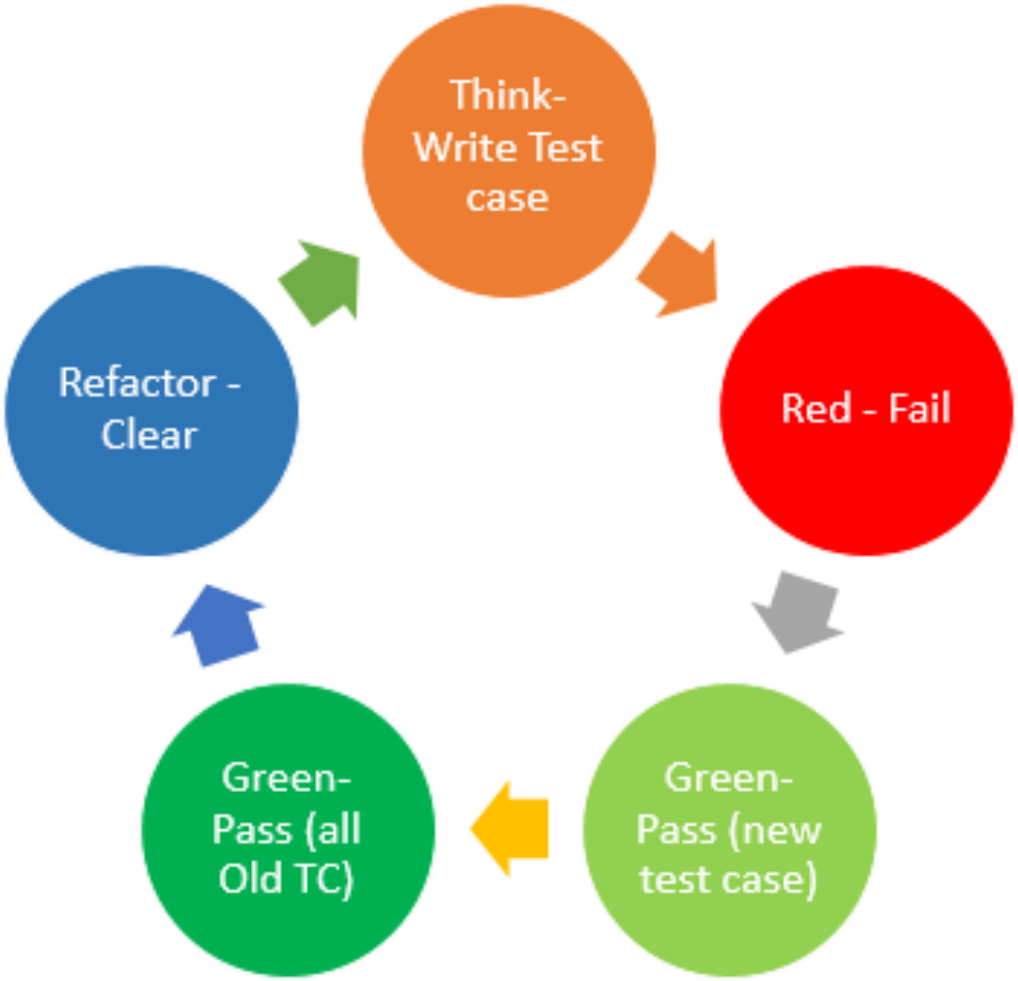
```
228   TestAsyncUtils.guardSync();
229   test_package.expect(actual, matcher, reason: reason, skip: skip);

Exception has occurred.
TestFailure (Expected: <5>
  Actual: <4>
)

230 }
231
```

Run tests using IntelliJ or VSCode

The Flutter plugins for IntelliJ and VSCode support running tests. This is often the best option while writing tests because it provides the fastest feedback loop as well as the ability to set breakpoints.

- **IntelliJ**
  1. Open the `counter_test.dart` file
  2. Select the `Run` menu
  3. Click the `Run 'tests in counter_test.dart'` option
  4. *Alternatively, use the appropriate keyboard shortcut for your platform.*
- **VSCode**
  1. Open the `counter_test.dart` file
  2. Select the `Debug` menu
  3. Click the `Start Debugging` option
  4. *Alternatively, use the appropriate keyboard shortcut for your platform.*

TDD


Think- Write Test case
Red - Fail
Green- Pass (new test case)
Green- Pass (all Old TC)
Refactor - Clear

Test driven development (TDD) is an software development approach in which a test is written before writing the code. Once the new code passes the test, it is refactored to an acceptable standard.

TDD ensures that the source code is thoroughly unit tested and leads to modularized, flexible and extensible code. It focuses on writing only the code necessary to pass tests, making the design simple and clear.

Mock


```yaml
dev_dependencies:
  flutter_test:
    sdk: flutter

  mockito: ^4.1.1
```

```dart
class WeatherProvider {
  Future<Weather> getCurrentWeather(http.Client client) async {
    final response = await client.get(
        'https://api.openweathermap.org/data/2.5/weather?q=Kharkiv&units=metric&APPID=1ea55013049215603ece3fee22806975');
    if (response.statusCode == 200) {
      return Weather.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to load weather data');
    }
  }
}
```

```
Run | Debug | You, a few seconds ago | 1 author (You)
group('fetch weather', () {

  Run | Debug
  test('returns a Weather model if http call completes successfully', () async {
    final client = MockClient();

    when(client.get(url))
    .thenAnswer((_) async => http.Response(successJSON, 200));

    expect((await WeatherProvider.getCurrentWeather(client)).runtimeType, Weather);
  });

  Run | Debug
  test('throws an exception if the http call completes with an error', () async {
    final client = MockClient();

    when(client.get(url))
    .thenAnswer((_) async => http.Response('Not Found', 404));

    expect(WeatherProvider.getCurrentWeather(client), throwsException);
  });
});
```


It's coffee time

15 minutes

```
enum NumberDescriptor {
  small,
  middle,
  big
}

class NumberHundler {

  NumberDescriptor descriptor(int number) {
    if (number <= 0) {
      throw Exception('Number should be bigger than 0');
    } else if (number < 10) {
      return NumberDescriptor.small;
    } else if (number < 100) {
      return NumberDescriptor.middle;
    } else {
      return NumberDescriptor.big;
    }
  }

}
```

BDD

Behavior Driven Development (BDD) is a branch of Test Driven Development (TDD). BDD uses human-readable descriptions of software user requirements as the basis for software tests. Like Domain Driven Design (DDD), an early step in BDD is the definition of a shared vocabulary between stakeholders, domain experts, and engineers. This process involves the definition of entities, events, and outputs that the users care about, and giving them names that everybody can agree on.

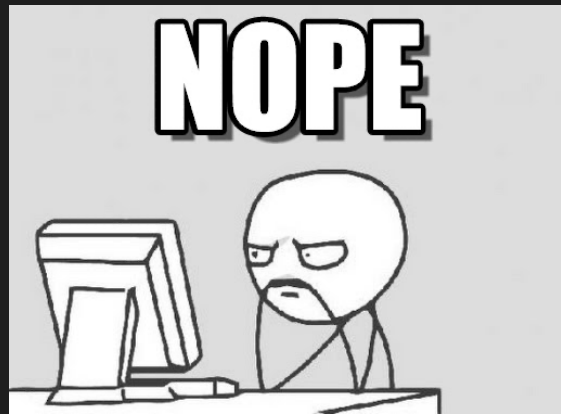
BDD vs TDD

Behavior Driven Development	Test Driven Development
• Start with business value, then drill down to feature sets	• Lots of tests that may or may not meet the business value
• Team gets feedback from the Product Owner	• Coder gets feedback from the code

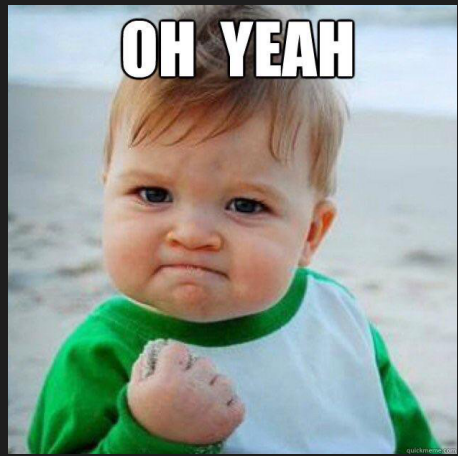
Widget testing

```dart
void main() {

  Run | Debug
  testWidgets('Find text', (WidgetTester tester) async {

    await tester.pumpWidget(MaterialApp(
      home: Scaffold(
        body: Text('H'),
      ), // Scaffold
    )); // MaterialApp

    expect(find.text('H'), findsOneWidget);

  });

}
```

```dart
void main() {

  Run | Debug
  testWidgets('Find text', (WidgetTester tester) async {

    await tester.pumpWidget(TodoList());

    expect(find.byType(TodoList), findsOneWidget);

  });

}
```



```dart
void main() {
  Run | Debug
  testWidgets('Find todo list', (WidgetTester tester) async {

    await tester.pumpWidget(MaterialApp(
      home: Scaffold(
        body: TodoList(),
      ), // Scaffold
    )); // MaterialApp

    expect(find.byType(TodoList), findsOneWidget);
    expect(find.byType(ListView), findsOneWidget);
  });
}
```



```
List<String> items = ['Test object'];
```

```
Run | Debug
testWidgets('Find item in the list', (WidgetTester tester) async {

  await tester.pumpWidget(MaterialApp(
    home: TodoList(),
  )); // MaterialApp

  expect(find.text('Test object'), findsOneWidget);
});
```

```
Run | Debug
testWidgets('Add item to the list', (WidgetTester tester) async {

  await tester.pumpWidget(MaterialApp(
    home: TodoList(),
  )); // MaterialApp

  expect(find.byType(ListTile), findsOneWidget);

  await tester.tap(find.byType(FloatingActionButton));

  await tester.pump();

  await tester.enterText(find.byType(TextField), 'new item');

  await tester.pump();

  await tester.tap(find.text('Add'));

  await tester.pump();

  expect(find.text('new item'), findsOneWidget);
  expect(find.byType(ListTile), findsNWidgets(2));
});
```

Flutter driver

https://flutter.dev/docs/cookbook/testing/integration/introduction

https://api.flutter.dev/flutter/flutter_driver/flutter_driver-library.html

Homework

1. Write tests for result app of module 1.
2. ********** Flutter driver