TESTING FRAMEWORKS: UNIT TESTS, FUNCTIONAL TESTS, TDD & BDD EXPLAINED

Programmers can write unit and functional tests using frameworks. **Unit tests** test individual lines of code. **Functional tests** test something larger, such as whether a transaction can still be executed. Other frameworks test that the application works on multiple versions of the targeted operating systems, different screen orientations on mobile devices, different browsers, and with different screen sizes. And there are volume testing tools as well. Here we look at one unit test, **Mocha**, and one functional test, **Cucumber**, framework. And we describe the logic behind using these and where they fit into project management.

Test Driven Development and Behavior Driven Development

Test Driven Development (TDD) is a clever idea to get programmers to focus on just what is important and not get stuck in the time-consuming task of solving esoteric problems or those that are not germane to the main task. TDD is an extension of the Agile Framework, whose goal is speed through simplicity and simplicity by delivering small discrete tasks and tracking those instead of trying to write an entire application per some giant GANTT chart, a process that is usually doomed to failure, say the Agile advocates.

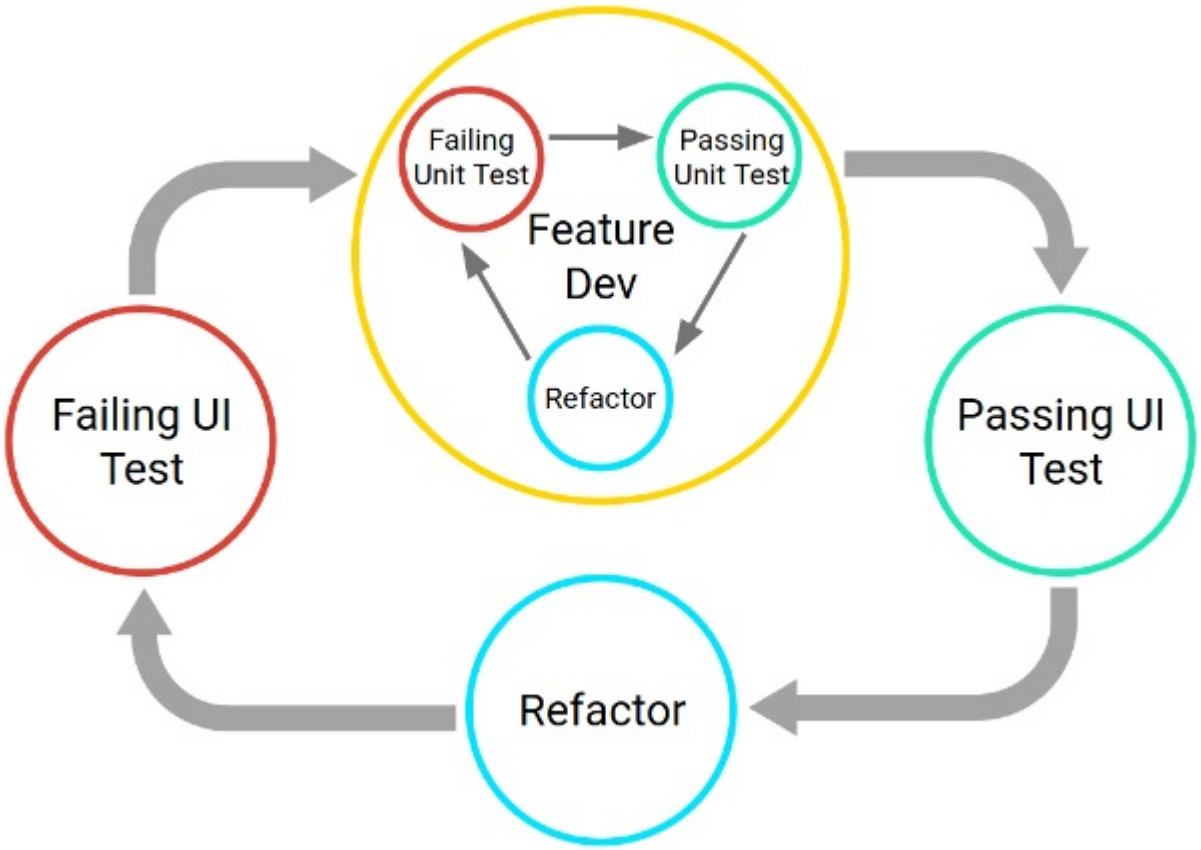
The basic idea with TDD and BDD is to write the test code first then write just enough of the application code to pass the test. For people in a hurry, such as trying to meet the deadline of an Agile story iteration, programmers can mock certain items, like writing fake database calls. Then in

the next Agile iteration they can write code to make those actually work. The goal is to keep making forward progress.

Behavior Drive Development is based on the same idea, but its focus is on the application and not testing individual paragraphs of code. So it is automated functional testing.

Android explain this process it in the graphic shown below. They mention the UI since they are focused on testing the Android app interface. But it is applicable to all types of programming.

The programmers runs the test and then when it fails they do **refactoring**, meaning fix the code. Big teams using Jenkins or other apps to coordinate the larger project can even put up a display on the wall to show which code sections are red (broken) or green (working) to let the whole project see at a glance what is the status of the **build**.



Mocha

Mocha is a unit test framework for Node.js JavaScript.

You can put it into a web page like this:

```
<script>
    mocha.checkLeaks();
    mocha.globals();
    mocha.run();
  </script>
```

You include the framework like this:

```
<script src="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.js"></script>
```

While in node.js (which is JavaScript for middleware) you install it like this:

```
npm install mocha -g
```

Then you can run the test script using **mocha** and not **node**:

```
mocha test.js
```

Or you can put tests into package.json. Then run **npm test** to run the tests. The code below tells npm to run the command **mocha test.js** to have mocha run the **test.js** script.

```
"scripts": {
    "test": "mocha test.js"
  },
```

Then write test.js:

```
var assert = require('assert');

function isEven(i) {

     return i%2
}

describe('iseven', function() {
    it('check to see it number divisible by 2', function() {
      assert.equal(0, isEven (2));
    });
});
```

And then run:

```
mocha test
```

This code below shows which tests to run using the keyword **describe**. You add additional **describe** commands to add additional tests and to build dependencies between them. In the example below, it uses the JavaScript command **assert** to test that 2 is an even number. (The function **isEven** returns the remainder of division by 2. If that is 0 then the number is even. Of course, for the finished product, we should also write code to test that 2 is an integer.)

The results will be something like:

```
iseven
    ✓ check to see it number divisible by 2


  1 passing (11ms)
```

Cucumber

Cucumber is a natural language testing framework suitable to functional testing. It is called **natural language** since it seeks to replace some pure code with something that is easier to understand.

For example, we can test that when the order entry system sells three items then the inventory system reports three less items in inventory.

Put the English (natural language) part in the file shown below. It includes the keywords **Given**, **When**, and **Then**. There must be code found for each of those as explained below. The section at the top is basically documentation.

```
# Comment
@tag
Feature: Sale Should Result in Decrease in Inventory

  When we make a sale inventory should go down
  Scenario: Make a Sale Check Inventory
    Given sell 3 items of ABC
    When inventory on hand is 9
    Then remaining inventory is 6
```

Here we see how we associate the natural language part with actual code. Each function below would execute some larger function in the larger application. For example, in this case, we would instantiate and run objects and methods to make a sale and then check inventory.

Notice that the text lines up with each **Given**, **When**, and **Then**. It uses regular expressions **"^inventory on hand is 9$"** to match the text. The caret (^) means the beginning of the line and the dollar sign ($) means the end.

If the function does not work as expected then **throw** a **PendingException()** error so that Cucumber will report that.

```
@Given("^sell 3 items of ABC$")
public void makeSale() {
    // Write code here that instantiates sale function in larger order entry
system
    throw new PendingException();
}

@When(("^inventory on hand is 9$")
public void checkInventoryNow() {
// put some code here
      throw new PendingException();
}

@Then("^remaining inventory is 6$")
public void checkInventoryAgain() {
    // put some code here
    throw new PendingException();
}
```

You can also write all of this code in Ruby, which is simpler and much shorter than Java. Ruby can instantiate Java objects.

These are only two testing products. There are an enormous number of those.