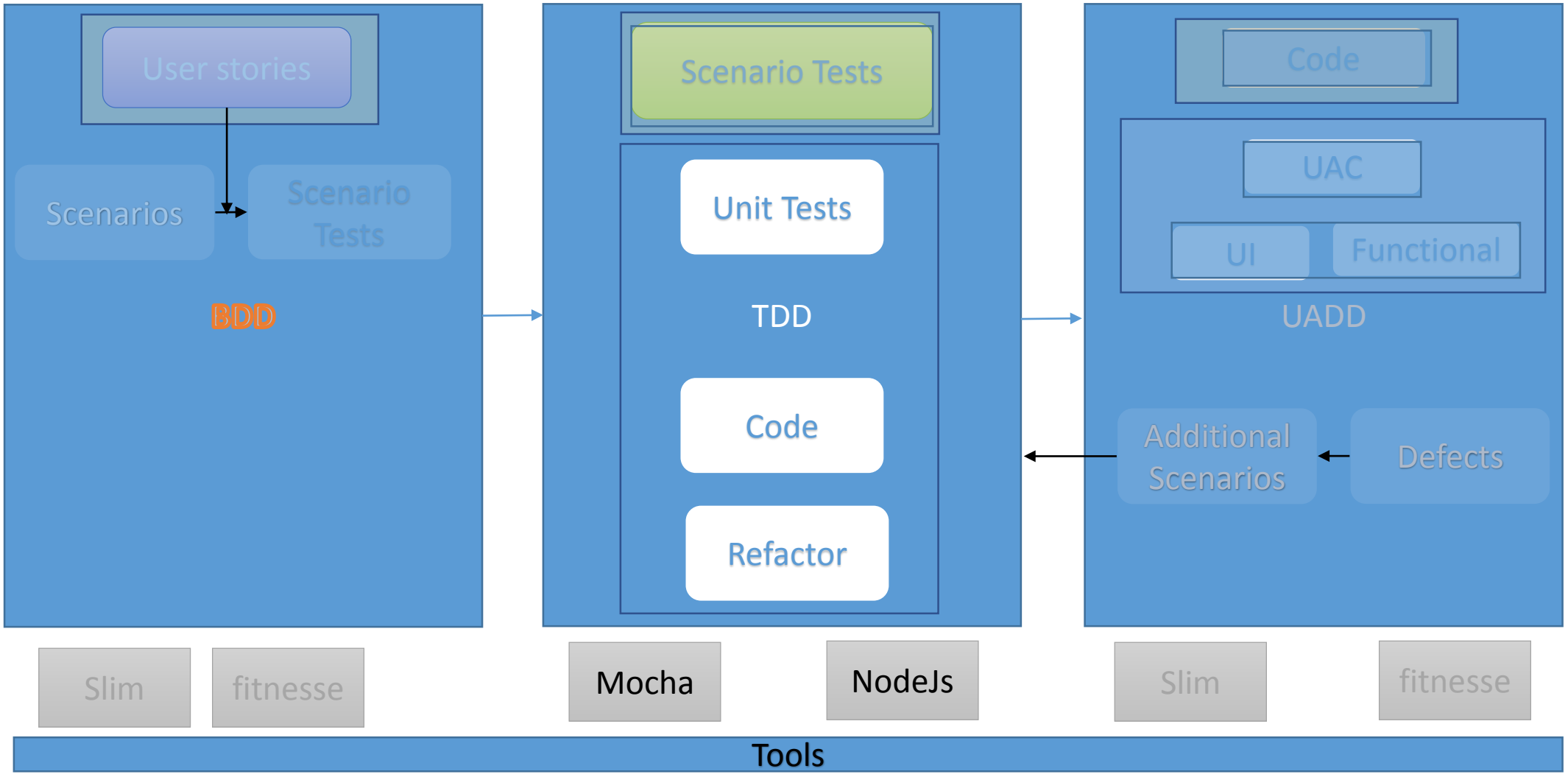
TDD with Mocha

Presented

By

Ravichandran J V

BDD + TDD + UADD


User stories
Scenarios
Scenario Tests
BDD
Scenario Tests
Unit Tests
TDD
Code
Refactor
Code
UAC
UI
Functional
UADD
Additional Scenarios
Defects
Slim
fitnesse
Mocha
NodeJs
Slim
fitnesse
Tools

Given, when, then … in action!

Requirement

- User types some English sentence that must comply to a supplied drill line of text.

- If user misses or adds any additional words from the drill text, then the words must be highlighted as errors for the user to correct.

Given, when, then … in action!

Requirement

Given

- User types some English sentence that must comply to a supplied drill line of text.

when

then

- If user misses or adds any additional words from the drill text, then the words must be highlighted as errors for the user to correct.

Identify Scenarios (Behavior)

Scenario 1 – if user misses words

- If user misses or adds any additional words from the drill text,

Scenario 2 – if user adds words

Scenario testing

// Code – twoscenarios.js

```
function myApi(){};
myApi.prototype = {
twoscenarios:function(drill

this.missingwordscenariote
ction(){
        return "missing";
   };



this.addedwordscenariotes
tion(){
     return "added";
   };
}
}
module.exports=new myApi();
```

The test script

```
!define TEST_SYSTEM {slim}

!define COMMAND_PATTERN {node E:\fitnesse\node_modules\slimjs\src\SlimJS %p}

!path E:\fitnesse\slimtests\src

|import|
|twoscenarios|

!|scenario|Missed word|

!|scenario|Added word|

|script|twoscenarios            |missing word|word|
|Missed word                                    |
|check |missingwordscenariotest|missing |

|script|twoscenarios          |added word|added word added|
|Added word                                               |
|check |addedwordscenariotest|added                       |
```

The test result

FirstBDDTest

classpath: E:\fitnesse\slimtests\src

import
twoscenarios

scenario	Missed word

scenario	Added word

script	twoscenarios	missing word	word
✚ Missed word			
check	missingwordscenariotest	[missing]	expected [missing]

script	twoscenarios	added word	added word added
✚ Added word			
check	addedwordscenariotest	[added]	expected [added]

Test Driven Development

- Test Driven Development is done by testing existing code before adding more to it.
- Test Driven Development (TDD) is done using unit tests
- A Unit test isolates and verifies individual units of source code
- A Unit Test is written first and if no code exists to satisfy the test, the unit test fails, always, in TDD
- A Unit Test drives development thus enabling design to evolve through satisfying the unit tests

TDD or TFD?

- **Test First Development** (TFD) is done by designing tests before satisfying them.

- Because the Scenario tests help in identifying unit tests, the framework that evolves out of this style of development, in which BDD precedes unit testing, can be called as TFD

- But beyond the first stage of code added to satisfy the unit tests, it becomes TDD ! ☺

Mocha – Unit test

Scenario – get missing word

```
describe('#get missing word', function() {
        it('should return missing word',function(){
        assert.equal(missingwordscenario(),"missing");
        });
});
```

As the code exists to make the tests (Scenario tests) pass.

```
// Code – twoscenarios.js

function myApi(){};
myApi.prototype = {
twoscenarios:function(drill,user){
   this.missingwordscenariotest=function(){
            return "missing";
   };

   this.addedwordscenariotest=function(){
     return "added";
   };
}
}
module.exports=new myApi();
```

Mocha – Unit test

Add the implementation as the test demands ie. Given a drill text “Missing word” and the user types “word” only, the code should return the “missing “ part

// test/Missingaddedwordtests.js

```
var app=require(‘.src/twoscenarios.js');
var assert=require('assert');

describe('#get missing word', function() {
     it('should return missing word',function(){
        var ob=new app.twoScenarios("missing word","word");
        assert.equal(ob.missingwordscenario(),"missing ");
    });
});
```

// twoscenarios.js

```
var jsDiff = require("diff");
function myApi(){};
myApi.prototype = {
twoScenarios : function(drill,user){
      var str="";
      var diff = jsDiff.diffWords(drill, user);
      this.missingwordscenario=function(){
          diff.forEach(function(part){
          if(part.removed){
             str+=part.value;
          }
          });
      return str;
   };
```

Mocha – Unit test

```
// test/Missingaddedwordtests.js

describe('#get added word', function() {
  it('should return added word',function(){
      var ob=new app. twoScenarios("word","word added");
            assert.equal("added",ob.addedwordscenariotest());
});
});
```

```
 var diff1 = jsDiff.diffWords(drill, user);
   this.addedwordscenario=function(){
       str="";
          diff1.forEach(function(part){
          if(part.added){
             str+=part.value;
          }
          });
      return str;
   };


}
}
module.exports=new myApi();
```

Run the Tests

- Run the tests

```
tests>mocha missingwordtest.js


  #get missing word
    √ should return missing word

  #get added word
    √ should return added word


  2 passing (33ms)
```

Test results

- Check this

```
1) #get missing word should return missing word:

    AssertionError: 'missing ' == 'missing'
    + expected - actual

    -missing
    +missing
```

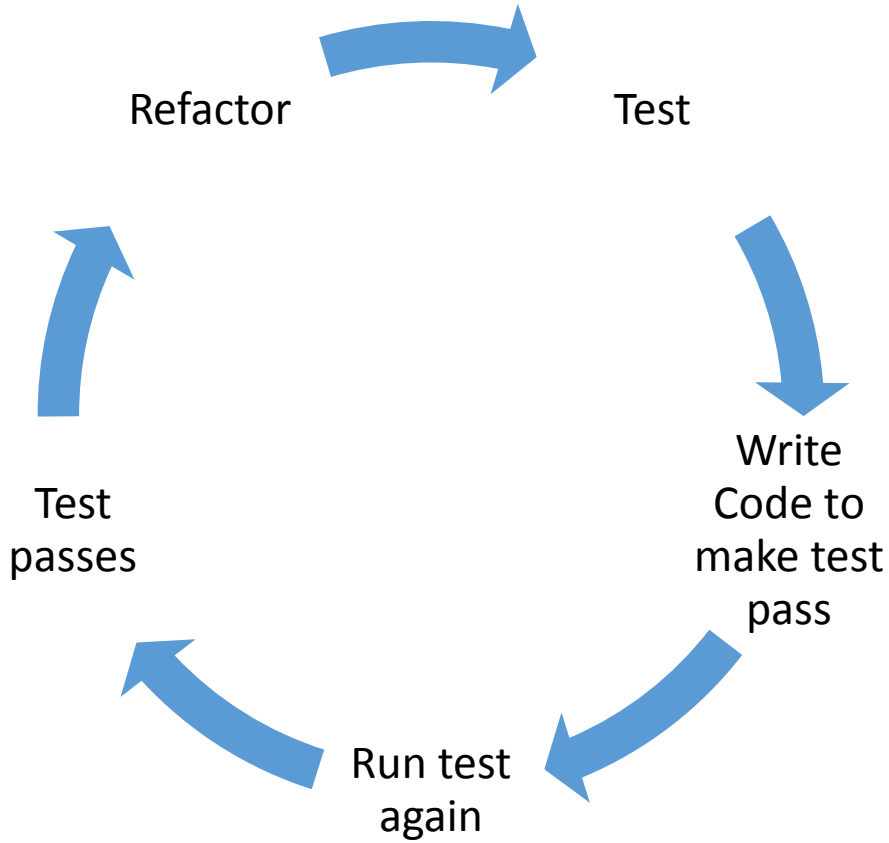
There is no space after “missing” and before “added” so the two tests fail:

```
2) #get added word should return added word:

    AssertionError: 'added' == ' added'
    + expected - actual

    -added
    + added
```

Best practices

- One assertion per test
- Refactor after test passes and run test again
- Add code only if the test result indicates the need
- Don’t add domain code in the test file. To ensure this practice, keep test files in ‘test’ folder and .js files in ‘src’ folder.
- Local variables like “str”, “diff”, “diff1” in this code need not be refactored!

TDD – Test, Code, Test Passes, Refactor Cycle



Scenarios to Unit Tests

- Scenario tests enable an additional level of granularity to the unit tests apart from increasing traceability of business requirements/rules to implementation and back.

- In the scenarios analyzed in the previous step, BDD, the space before or after the missing word and the added word would not have been apparent if we had simply done a unit test!

FirstBDDTest

classpath: E:\fitnesse\slimtests\src

import
twoscenarios

scenario	Missed word

scenario	Added word

script	twoscenarios	missing word	word
✚ Missed word			
check	missingwordscenariotest	[missing]	expected [missing]

script	twoscenarios	added word	added word added
✚ Added word			
check	addedwordscenariotest	[added]	expected [added]

Thank you!