# Testing Angular

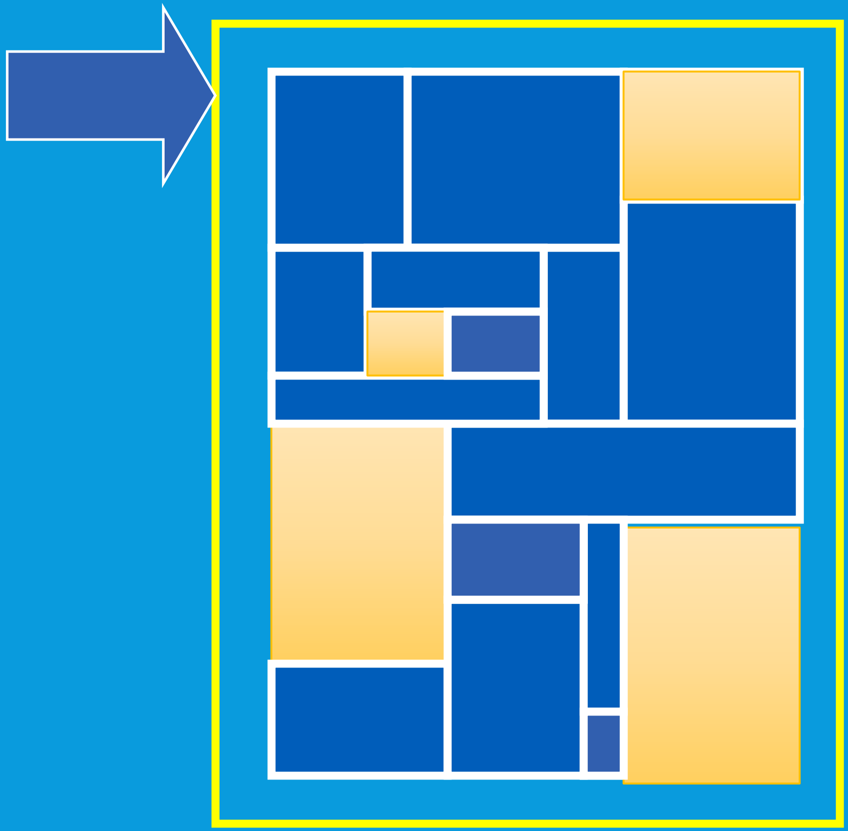
## Agenda

- Testing Basics & Setup
- Karma & Jasime Basics
- Testing Angular Components & Services
- Server Side Testing

## Testing Basics & Setup

## Traditional Testing

- Test the system as a whole
- Individual components rarely tested
- Errors go undetected
- Isolation of errors difficult to track down



## Manual vs Automatic Testing

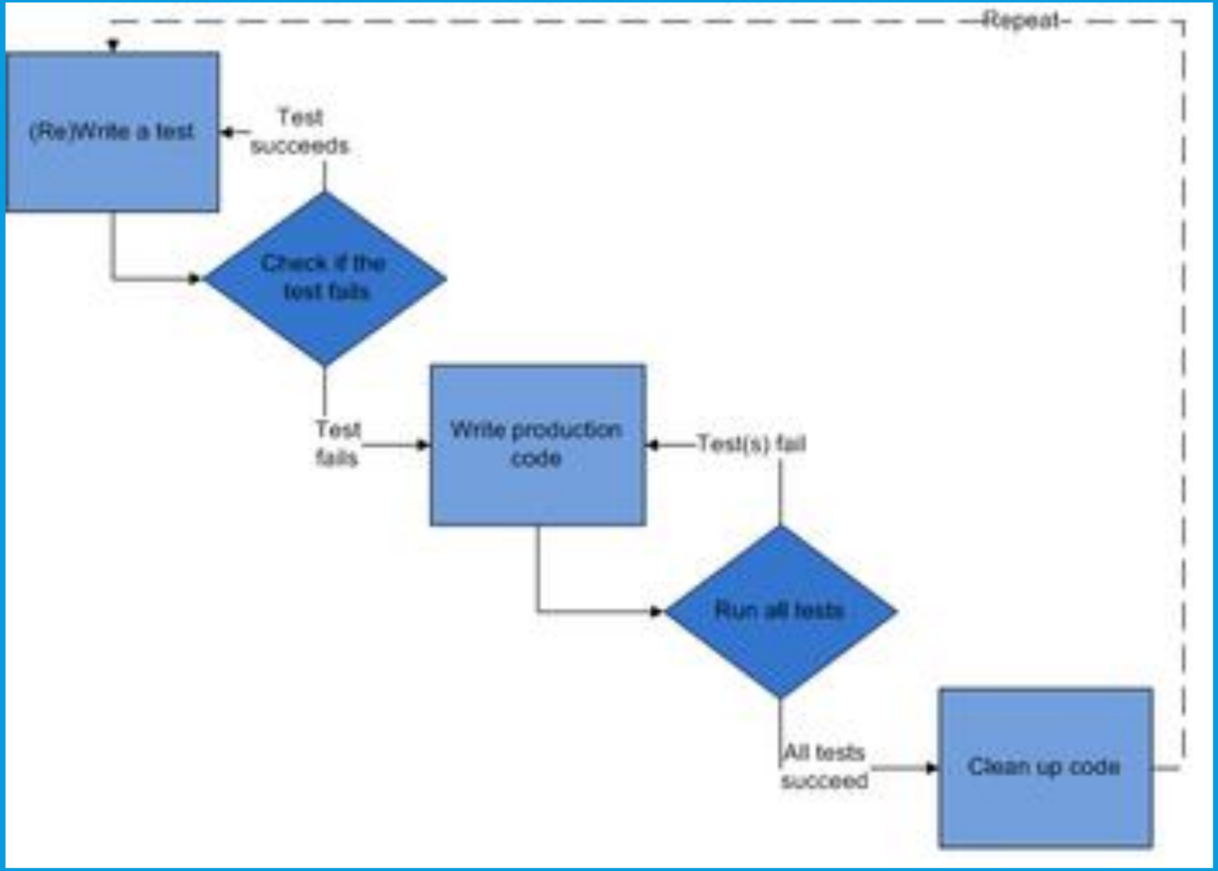
- Manual Tests
  - Print Statements
  - Use of Debugger
  - Debugger Expressions
  - Test Scripts

- Automated Testing
  - Less Human Errors
  - Faster
  - Descreasing Cost
- Types of Tests
  - Unit Tests
  - Integration Tests (End to End, E2E)

# Test-driven development (TDD)

For each new feature in the program:

1. Write (failing) test case
2. Run the test, to see that it fails
3. Write code until the test pass
4. Refactor the code to acceptable standards



## Karma

- Angular CLI installs and konfigures Karam by default
- Karma is a testrunner that executes test
- Supports several Testing Frameworks
  - Jasmine
  - Mocha
  - Qunit
- Supports Continous Integration
- Documentation @ https://karma-runner.github.io

## Karma Components

- Karma Configuration in karma.conf.js
- karma-cli:
  - Command Line Interface for Karma
- karma-chrome-launcher
  - Launches Karma in Chrome @https://github.com/karma-runner/karma-chrome-launcher
- karma-jasmine
  - Jasmine plugin for Karma

## Jasmine

- Popular Testing Framework used by Angular CLI by default
- Jasmine has the following features:
  - Easy-to-read (expressional) syntax
  - Testing async code
  - Spies (mocking objects and methods)
  - DOM testing

## Jasmine Overview

Suite – a suite of related tests, created using "describe"

Specs – expectations to test for, created using "it"

Setup and teardown - beforeEach() and afterEach() methods.

Spec Runner – a simple HTML page or an automated process

## Matchers

- toBe / toEqual
- toMatch:  RegExp match()
- toBeDefined / toBeUndefined
- toBeNull
- toContain
- toBeLessThan/toBeGreaterThan
- toThrow: for catching expected exceptions

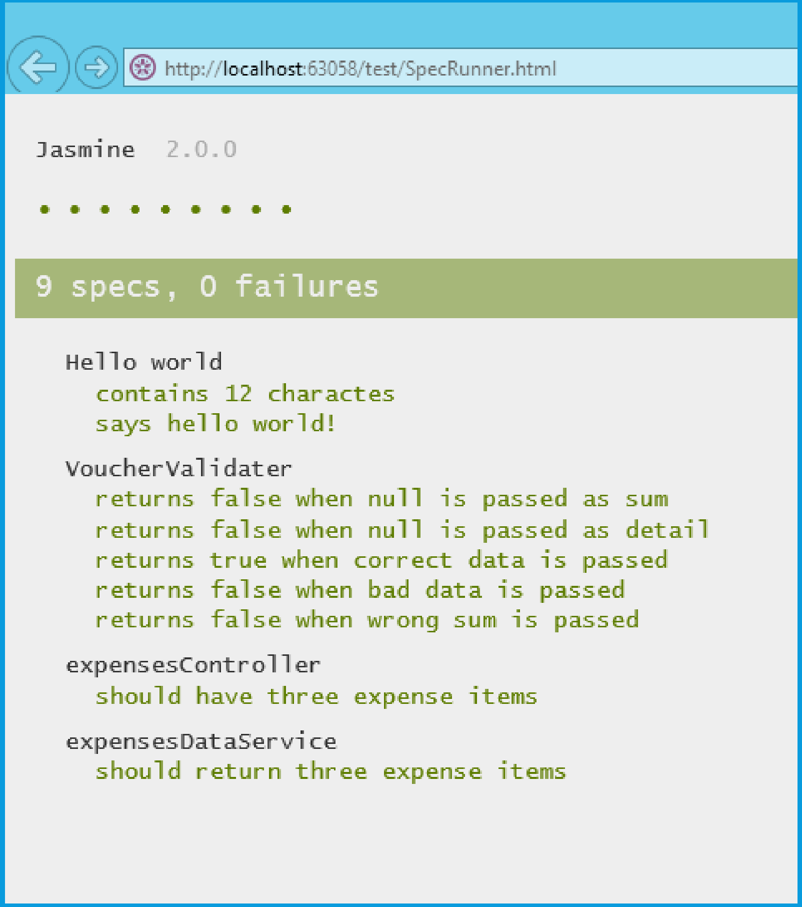
# Hello World Test

- Function to be tested

```
function helloWorld() {
    return "Hello world!";
}
```

- Suite and Specs

```
describe("Hello world", function () {

    it("contains 12 charactes", function () {
        expect(helloWorld().length).toEqual(12);
    });
    it("says hello", function () {
        expect(helloWorld()).toEqual("Hello world!");
    });

});
```



```
http://localhost:63058/test/SpecRunner.html

Jasmine  2.0.0

• • • • • • • • •

9 specs, 0 failures

Hello world
  contains 12 charactes
  says hello world!

VoucherValidater
  returns false when null is passed as sum
  returns false when null is passed as detail
  returns true when correct data is passed
  returns false when bad data is passed
  returns false when wrong sum is passed

expensesController
  should have three expense items

expensesDataService
  should return three expense items
```

## Isolated Tests

## Unit Tesing Basics

- Breaks down the functionality of a program into discrete testable called units
- Use a unit testing framework to create unit tests, run them, and report the results of these test
- With test driven development, you create the unit tests before you write the code, so you use the unit tests as
  - design documentation and
  - functional specifications.

## Elements of Unit Testing

- Functional correctness and completeness
- Error handling
- Checking input values (parameter)
- Correctness of output data (return values)
- Optimizing algorithm and performance

## Benefits of Unit Testing

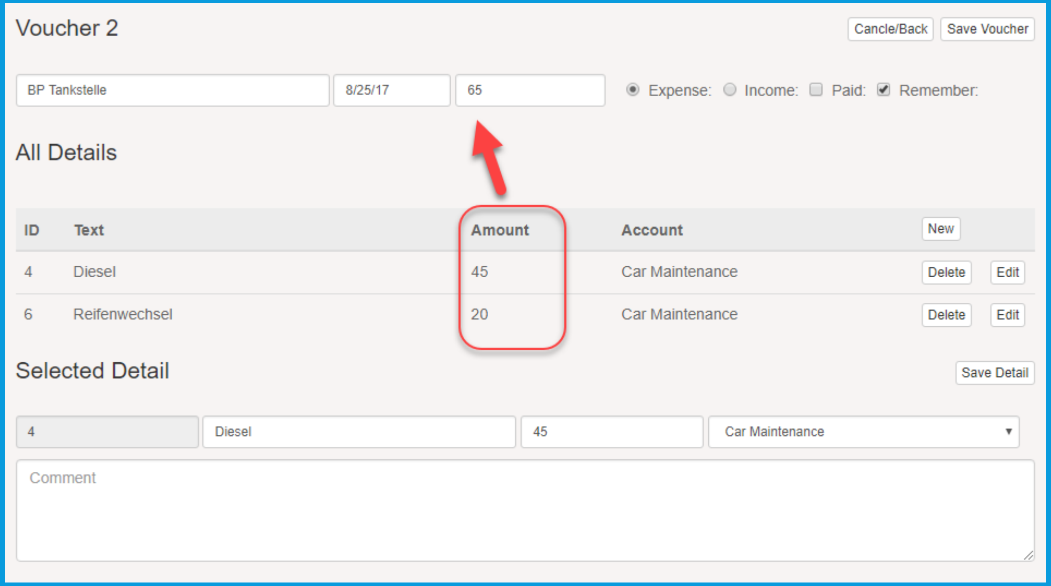
- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly.
- By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.
- Unit testing provides a sort of living documentation of the system.

# GUIDELINES

- Make unit tests simple to run
- Test code must be separated from the code to be tested
- Fix failing tests immediately
- Keep testing at unit level
- Name tests properly
- Test public API
- Provide negative tests

# Jasmine – Testing a simple Class

- VoucherValidator should test:
  - Sum Details == VoucherSum
  - Null handling



Voucher 2

|  |  |  |  |
|---|---|---|---|
| BP Tankstelle | 8/25/17 | 65 | [● Expense:] ○ Income: ☐ Paid: [✓ Remember:] |

All Details

| ID | Text | Amount | Account | New |
|---|---|---|---|---|
| 4 | Diesel | 45 | Car Maintenance | Delete Edit |
| 6 | Reifenwechsel | 20 | Car Maintenance | Delete Edit |

Selected Detail

|  |  |  |  |
|---|---|---|---|
| 4 | Diesel | 45 | Car Maintenance |

Comment

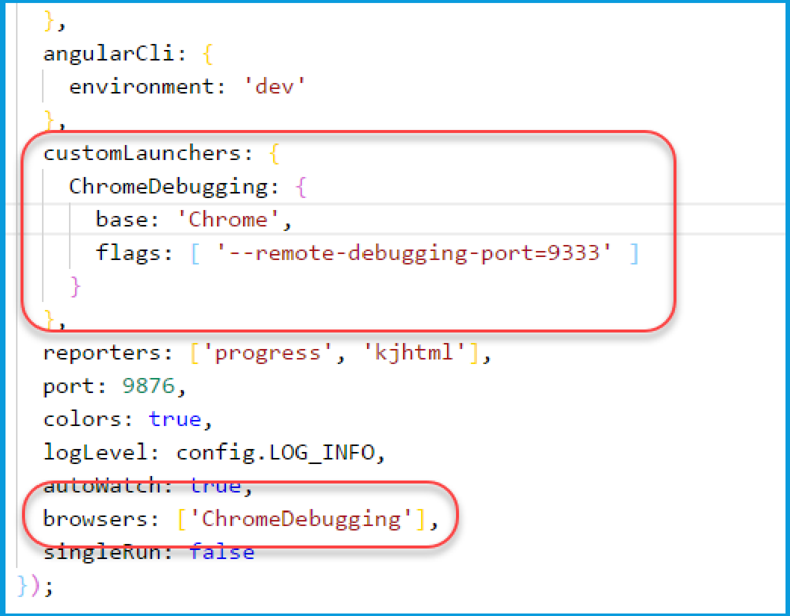
```
export class VoucherValidator{

static validate(voucher : Voucher)
{
 var detailSumOk: boolean;
 if (voucher.Details!=null) {
   var sumD = 0;
   for(let vd of voucher.Details){
     sumD += vd.Amount;
   }
   detailSumOk = sumD == voucher.Amount;
 }
 return detailSumOk;
 }
}
```

# Testing VoucherValidator using Jasmine

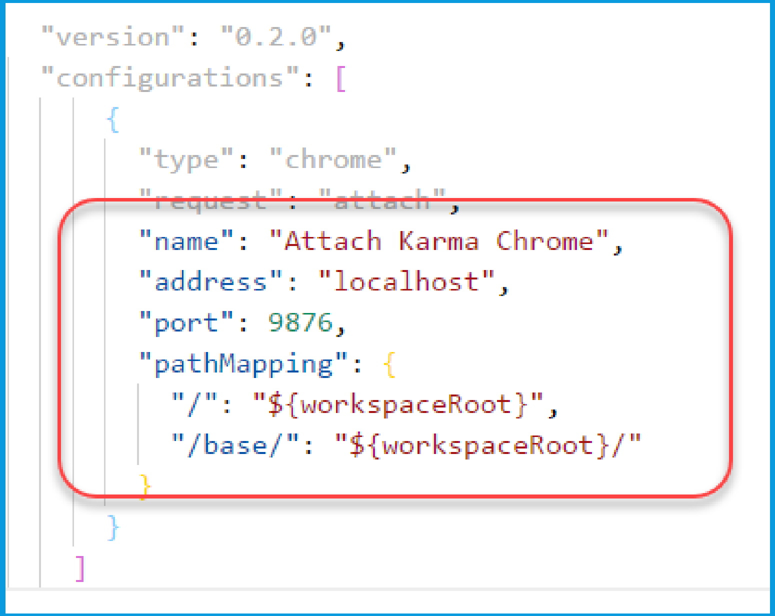
```
var nullVoucher = {
 "ID": 2,
 "Text": "BP Tankstelle",
 "Date": "2016-11-15T00:00:00",
 "Amount": 650,
 "Paid": false,
 "Expense": false,
 "Remark": true,
 "Details": null
};

it("returns true when correct data is passed", function () {
 expect(VoucherValidator.validate(goodvoucher)).toEqual(true);
});

it("returns false when bad data is passed", function () {
 expect(VoucherValidator.validate(goodvoucher)).toEqual(false);
});

it("returns false when null is passed as Details", function () {
 expect(VoucherValidator.validate(nullVoucher)).toEqual(false);
});
```

# Debugging Karma Tests

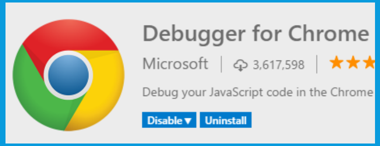
- Requires Debugger for Chrome Extension
- Configuration done in karma.conf.js and launch.json



```
  },
  angularCli: {
    environment: 'dev'
  },
  customLaunchers: {
    ChromeDebugging: {
      base: 'Chrome',
      flags: [ '--remote-debugging-port=9333' ]
    }
  },
  reporters: ['progress', 'kjhtml'],
  port: 9876,
  colors: true,
  logLevel: config.LOG_INFO,
  autoWatch: true,
  browsers: ['ChromeDebugging'],
  singleRun: false
});
```



```
"version": "0.2.0",
"configurations": [
    {
        "type": "chrome",
        "request": "attach",
        "name": "Attach Karma Chrome",
        "address": "localhost",
        "port": 9876,
        "pathMapping": {
          "/": "${workspaceRoot}",
          "/base/": "${workspaceRoot}/"
        }
    }
]
```



Debugger for Chrome
Microsoft | 3,617,598 | ★★★
Debug your JavaScript code in the Chrome

## Mocking

- Mocking is primarily used in unit testing
- Object tested may have dependencies on others
- Replace the other objects by mocks that simulate the behavior of the real objects.
- Canbe done with
  - Fake objects & overriding functions
  - Using a real instance with Spy

## Testing Angular Components using Mocks

```
let spy: any;

beforeEach(() => {
service = new AuthService();
component = new LoginComponent(service);
});

describe('Component: Login', () => {
 let component: LoginComponent;
 let service: MockAuthService;

 beforeEach(() => {
  service = new MockAuthService();
  component = new LoginComponent(service);
 });

 afterEach(() => {
  service = null;
  component = null;
 });

 it('canLogin returns false when the user is not authenticated', () => {
  service.authenticated = false;
  expect(component.needsLogin()).toBeTruthy();
});
```

## Testing Angular Components using Spy

- A Spy is a feature of Jasmine which lets you take an existing class, function, object and mock it & control what gets returned

```
it('canLogin returns false when the user is not authenticated', () => {
 spy = spyOn(service, 'isAuthenticated').and.returnValue(false);
 expect(component.needsLogin()).toBeTruthy();
 expect(service.isAuthenticated).toHaveBeenCalled();
});
```

## Integration Tests

- An integration test is done to demonstrate that different pieces of the system work together
- Integration tests cover whole applications
- Require resources like database instances and hardware to be allocated for them
- Can expose problems with the interfaces among program components

## Integration Test Setup

- Test the Integration of Components and their Templates
- Uses { TestBed, async, ComponentFixture} from '@angular/core/testing'
- Uses { DebugElement, Component, NO_ERRORS_SCHEMA } from '@angular/core'
- Uses { By } from '@angular/platform-browser'

# Fixture & Testbed

- Fixture
  - A Fixture is a wrapper AROUND the Component to be tested
  - Usage: ComponentFixture<TYPE>
- Testbed
  - Configures and initializes environment for testing and provides methods for creating components and services in tests.

## Server Side Testing

## Unit testing frameworks

- MSTest
  - Command-line driven testing utility
- NUnit
  - NUnit is an open source unit testing framework for Microsoft .NET
- xUnit
  - A free, open source, community-focused unit testing tool for the .NET Framework
  - Supports Resharper

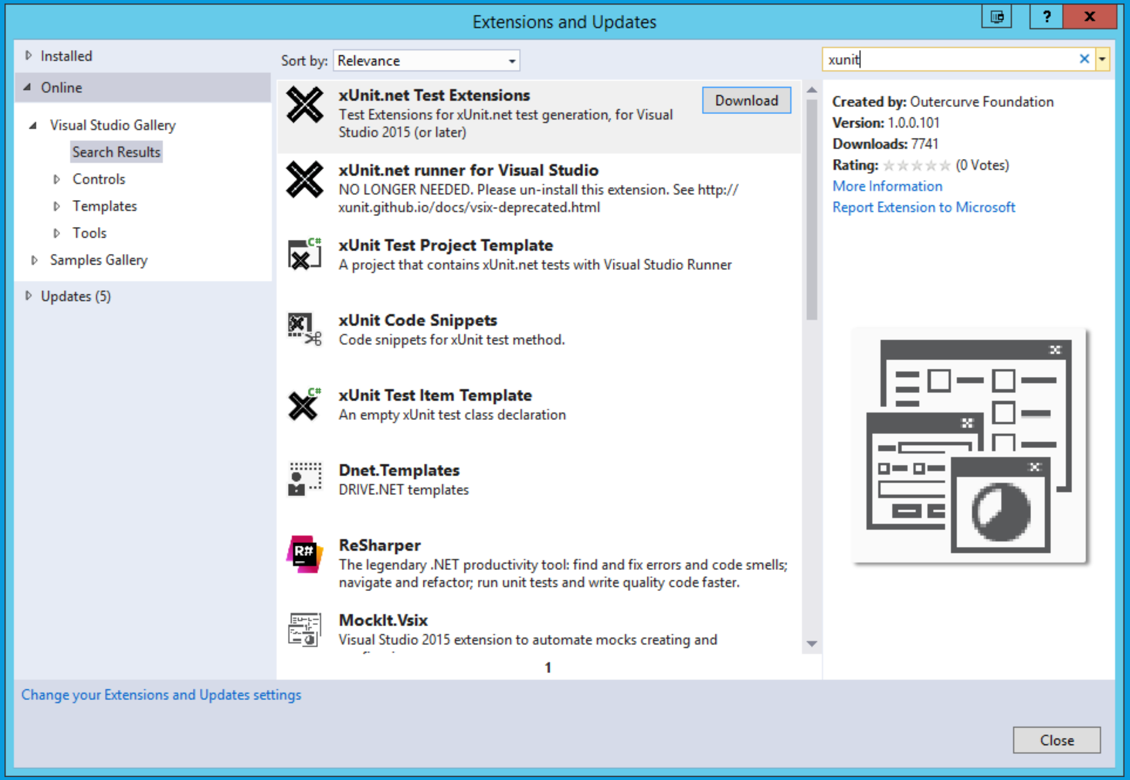
## xUnit

- xUnit.net is a free, open source, community-focused unit testing tool  for the .NET Framework.

```
{
  "version": "1.0.0-*",
  "testRunner": "xunit",
  "dependencies": {
    "VoucherEditor": "1.0.0-*",
    "Microsoft.EntityFrameworkCore": "1.0.0-rc2-final",
    "xunit": "2.1.0",
    "dotnet-test-xunit": "1.0.0-rc2-build10025"
  },
```

# xUnit VS Extensions

- xUnit has supports test extensions for Visual Studio



**xUnit.net Test Extensions**
Test Extensions for xUnit.net test generation, for Visual Studio 2015 (or later)

**xUnit.net runner for Visual Studio**
NO LONGER NEEDED. Please un-install this extension. See http://xunit.github.io/docs/vsix-deprecated.html

**xUnit Test Project Template**
A project that contains xUnit.net tests with Visual Studio Runner

**xUnit Code Snippets**
Code snippets for xUnit test method.

**xUnit Test Item Template**
An empty xUnit test class declaration

**Dnet.Templates**
DRIVE.NET templates

**ReSharper**
The legendary .NET productivity tool: find and fix errors and code smells; navigate and refactor; run unit tests and write quality code faster.

**MockIt.Vsix**
Visual Studio 2015 extension to automate mocks creating and

1

**Created by:** Outercurve Foundation
**Version:** 1.0.0.101
**Downloads:** 7741
**Rating:** (0 Votes)
More Information
Report Extension to Microsoft

Change your Extensions and Updates settings

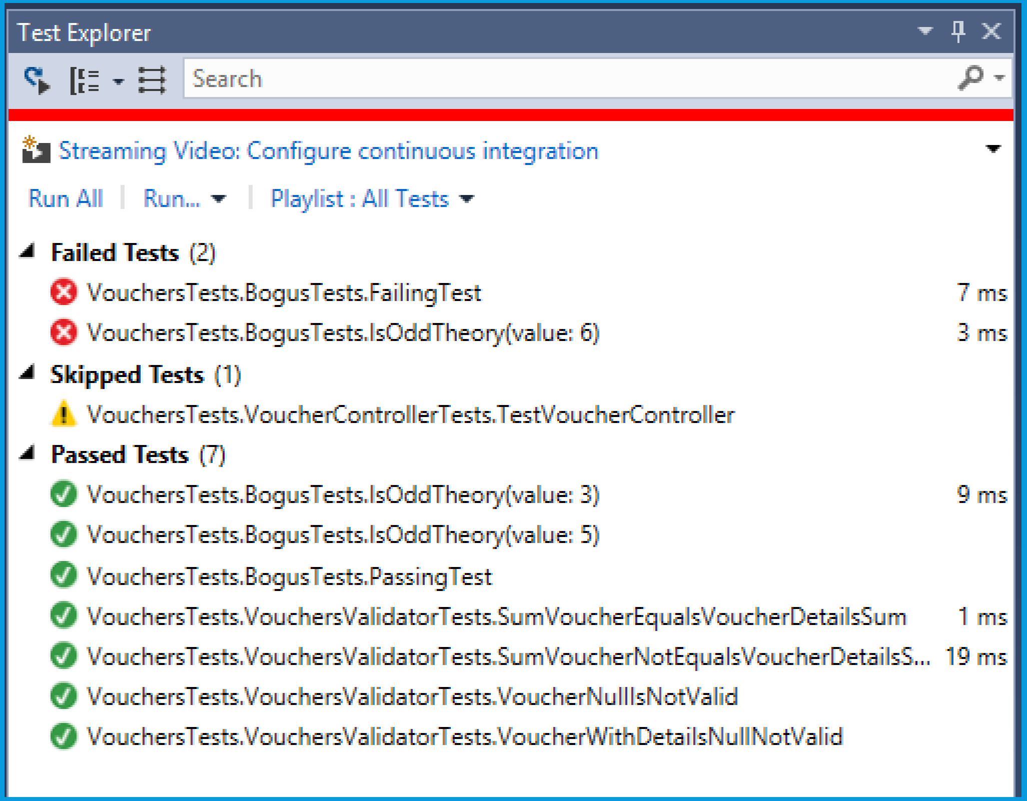
## Writing Tests

- xUnit supports
  - Facts
  - Theories

```
[Fact]
public void SumVoucherEqualsVoucherDetailsSum()
{
  Assert.Equal(VoucherValidator.Validate(data.CurrentVoucher,
  data.CurrentVoucher.Details.ToArray(), data.Accounts.ToArray()),true);
}
```

# Steps to use xUnit

- Create a class library project
- Add a reference to xUnit.net
- Write your tests
- Run your tests
  - From Console
  - Using Test Explorer



```
Failed Tests (2)
  VouchersTests.BogusTests.FailingTest                                            7 ms
  VouchersTests.BogusTests.IsOddTheory(value: 6)                                  3 ms
Skipped Tests (1)
  VouchersTests.VoucherControllerTests.TestVoucherController
Passed Tests (7)
  VouchersTests.BogusTests.IsOddTheory(value: 3)                                  9 ms
  VouchersTests.BogusTests.IsOddTheory(value: 5)
  VouchersTests.BogusTests.PassingTest
  VouchersTests.VouchersValidatorTests.SumVoucherEqualsVoucherDetailsSum          1 ms
  VouchersTests.VouchersValidatorTests.SumVoucherNotEqualsVoucherDetailsS...     19 ms
  VouchersTests.VouchersValidatorTests.VoucherNullIsNotValid
  VouchersTests.VouchersValidatorTests.VoucherWithDetailsNullNotValid
```