UNIT TESTING STRATEGIES & PATTERNS IN C#

BILL DINGER
Solutions Architect | @adazlian
bill.dinger@vml.com

TITANIUM SPONSORS

epiq

PAIGE TECHNOLOGIES
INTELLIGENT PAIRING. PERPETUAL SUCCESS.

ADAPTIVE™ SOLUTIONS GROUP

okta

VinSolutions
Make every connection count.

on

jack henry & ASSOCIATES INC.®

VALOREM

NAIC
National Association of Insurance Commissioners

NIPR
NATIONAL INSURANCE PRODUCER REGISTRY

TEKsystems
Our people make IT possible.

KEYHOLE SOFTWARE

smartsheet
Work Better™

Couchbase

EquipmentShare

KOCH™

PLURALSIGHT

Google Cloud

VML

mylo™
POWERED BY LOCKTON

Gold Sponsors

PeopleAdmin

Advantage Tech
IT Staffing & Recruiting Services

CENTRIQ TRAINING

Cerner

KU EDWARDS CAMPUS
The University of Kansas

twilio

{Track:js}
JAVASCRIPT ERROR MONITORING

OBJECT PARTNERS

Balance Innovations

smg
service management group®

stackify

GARMIN.

dsi

Allied Global SERVICES

TouchNet®
+ Heartland

LRS™
CONSULTING SERVICES

Aviture™
ingenuity. engineered.

Mediware®

An affiliate of C&A Industries, Inc.
AUREUS® GROUP
Finance | Systems | Executive

Children's Mercy
KANSAS CITY

Commerce Bank

SECURITY PS
STRATEGIC INFORMATION SECURITY

ProKarma

DEMO CODE

https://github.com/BillDinger/BikeShop

WHAT IS A UNIT TEST?

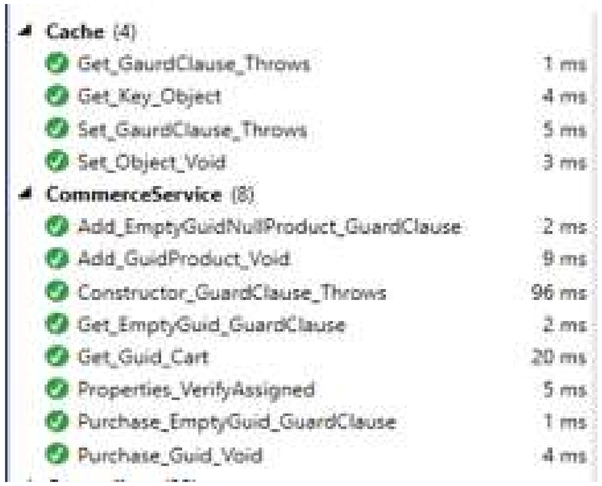
“A test written by a programmer for the purpose of ensuring that the production code does what the programmer expects it to do.” – Robert Martin

OTHER TYPES OF TESTING

- **Acceptance testing** – written by business and typically designed around acceptance criteria – business analysts and QA author these tests.
- **Integration Test** – written by architect/lead to ensure all the system components are written correctly.
- **System Test** - written by architect/lead to ensure subsystem components are written correctly.

GOALS OF UNIT TESTING

- Ensure your code does what it's supposed to
- Reduce brittleness of code
- Produce well-written, productive code – reduce cost of failure
- Enable test-driven development
- Enable rapid developer feedback loops – don’t have to wait for entire application to start



```
◢ Cache (4)
    ✓ Get_GaurdClause_Throws                      1 ms
    ✓ Get_Key_Object                              4 ms
    ✓ Set_GaurdClause_Throws                      5 ms
    ✓ Set_Object_Void                             3 ms
◢ CommerceService (8)
    ✓ Add_EmptyGuidNullProduct_GuardClause        2 ms
    ✓ Add_GuidProduct_Void                        9 ms
    ✓ Constructor_GuardClause_Throws             96 ms
    ✓ Get_EmptyGuid_GuardClause                   2 ms
    ✓ Get_Guid_Cart                              20 ms
    ✓ Properties_VerifyAssigned                   5 ms
    ✓ Purchase_EmptyGuid_GuardClause              1 ms
    ✓ Purchase_Guid_Void                          4 ms
```

OUR UNIT TESTS SHOULD BE

- Fast
- Pass/fail ⍰ never inconclusive
- Repeatable
- Order Independent
- Easy to set up
- Test one small piece of functionality
- Test *public* interfaces only

BUILDING FOR UNIT TESTING

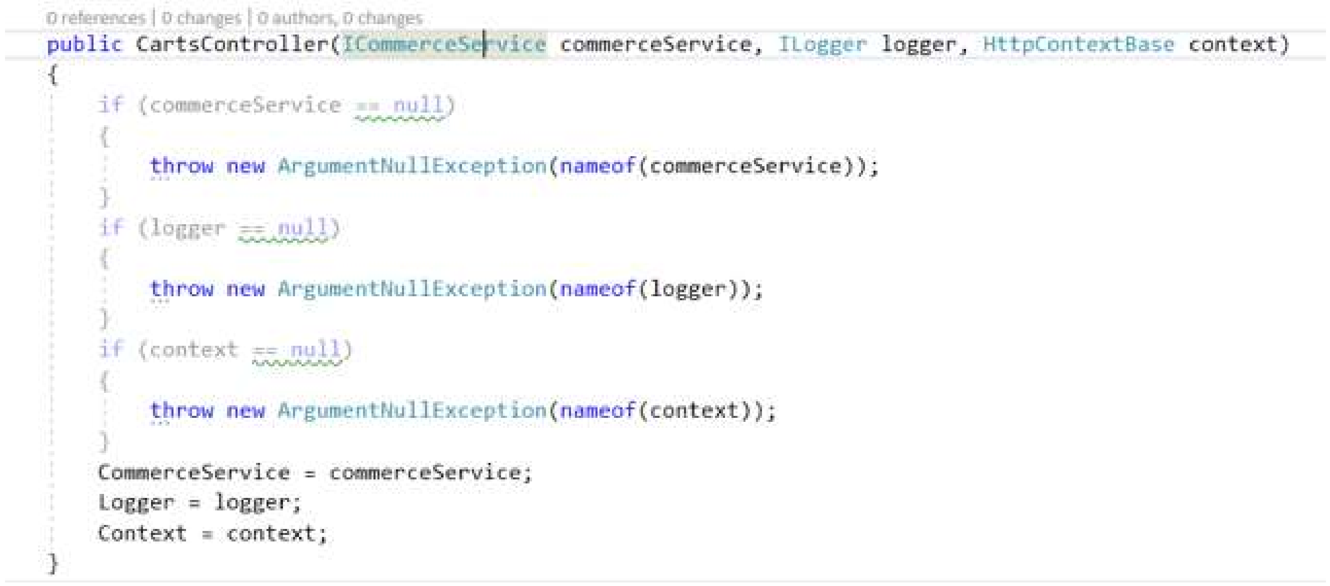
Building software as small, testable units is much easier if you adhere to SOLID design principles.

S	Single Responsibility Principle. A class should have one reason to change or a class should do one thing and do it well.
O	Open/Closed Principle. Open for extension, closed for modification.
L	Liskov Substitution Principle. Any derived class can be used in place of its base class. In .NET this is usually found when a class throws a NotSupportedException.
I	Interface Segregation Principle. Clients should not be forced to depend on methods they do not use.
D	Dependency Inversion Principle. • High-level modules should not depend on low-level modules. Both should depend on abstractions. • Abstractions should not depend upon details. Details should depend upon abstractions.

BUILDING TIP #1

USE DEPENDENCY INJECTION

- New is your enemy. Specifically, new forces your class to take on a dependency on another class. Use dependency injection to give your classes all their dependencies up-front in the constructor.

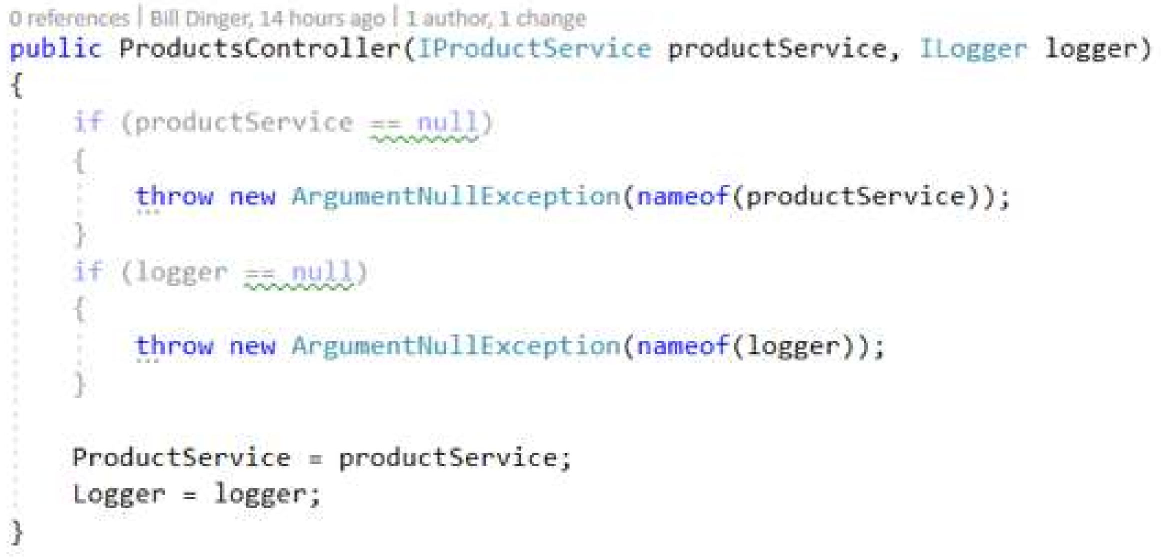


```
0 references | 0 changes | 0 authors, 0 changes
public CartsController(ICommerceService commerceService, ILogger logger, HttpContextBase context)
{
    if (commerceService == null)
    {
        throw new ArgumentNullException(nameof(commerceService));
    }
    if (logger == null)
    {
        throw new ArgumentNullException(nameof(logger));
    }
    if (context == null)
    {
        throw new ArgumentNullException(nameof(context));
    }
    CommerceService = commerceService;
    Logger = logger;
    Context = context;
}
```

BUILDING TIP #2

RELY ON ABSTRACTIONS

- Relying on interfaces and abstractions in your code allows your Moq frameworks to easily replace behavior.

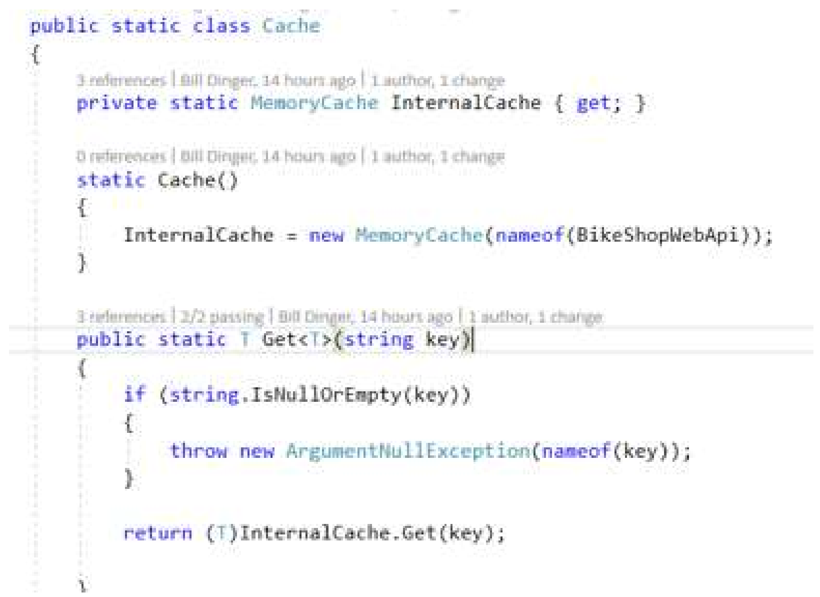


```
0 references | Bill Dinger, 14 hours ago | 1 author, 1 change
public ProductsController(IProductService productService, ILogger logger)
{
    if (productService == null)
    {
        throw new ArgumentNullException(nameof(productService));
    }
    if (logger == null)
    {
        throw new ArgumentNullException(nameof(logger));
    }

    ProductService = productService;
    Logger = logger;
}
```

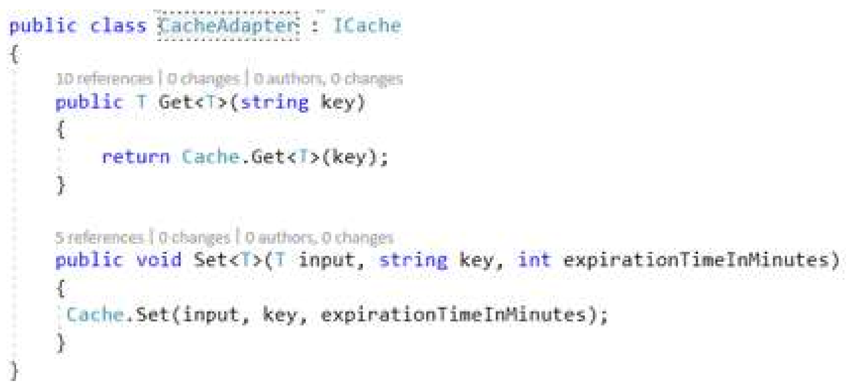
BUILDING TIP #3

AVOID STATIC CLASSES

- Static classes cannot be mocked. If they must be used, hide them behind a shim or adapter class that can be injected.



```
public static class Cache
{
    3 references | Bill Dinger, 14 hours ago | 1 author, 1 change
    private static MemoryCache InternalCache { get; }

    0 references | Bill Dinger, 14 hours ago | 1 author, 1 change
    static Cache()
    {
        InternalCache = new MemoryCache(nameof(BikeShopWebApi));
    }

    3 references | 2/2 passing | Bill Dinger, 14 hours ago | 1 author, 1 change
    public static T Get<T>(string key)
    {
        if (string.IsNullOrEmpty(key))
        {
            throw new ArgumentNullException(nameof(key));
        }

        return (T)InternalCache.Get(key);

    }
```



```
public class CacheAdapter : ICache
{
    10 references | 0 changes | 0 authors, 0 changes
    public T Get<T>(string key)
    {
        return Cache.Get<T>(key);
    }

    5 references | 0 changes | 0 authors, 0 changes
    public void Set<T>(T input, string key, int expirationTimeInMinutes)
    {
        Cache.Set(input, key, expirationTimeInMinutes);
    }
}
```

BUILDING TIP #4

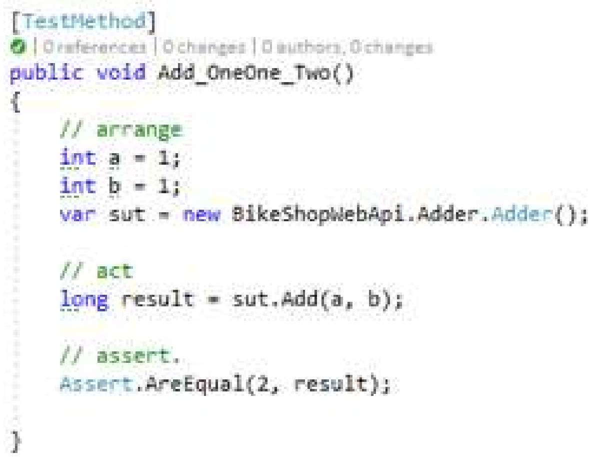
THE SMALLER YOUR CLASSES AND METHODS, THE EASIER YOUR LIFE IS

“The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that. Functions should not be 100 lines long. Functions should hardly ever be 20 lines long.” – Robert Martin, Clean Code

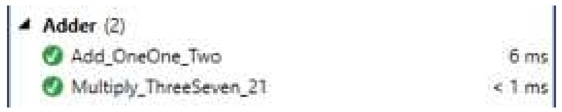
Single Responsibility Principle: Your class or method should only do one thing.

WHAT IS A UNIT?

- Whatever the collective you think it is.
- Can be method, class or group of classes.
- Done by convention. In our code, the examples are tests of a single method grouped by class.



```
[TestMethod]
| 0 references | 0 changes | 0 authors, 0 changes
public void Add_OneOne_Two()
{
    // arrange
    int a = 1;
    int b = 1;
    var sut = new BikeShopWebApi.Adder.Adder();

    // act
    long result = sut.Add(a, b);

    // assert.
    Assert.AreEqual(2, result);

}
```



```
◢ Adder (2)
   Add_OneOne_Two              6 ms
   Multiply_ThreeSeven_21    < 1 ms
```

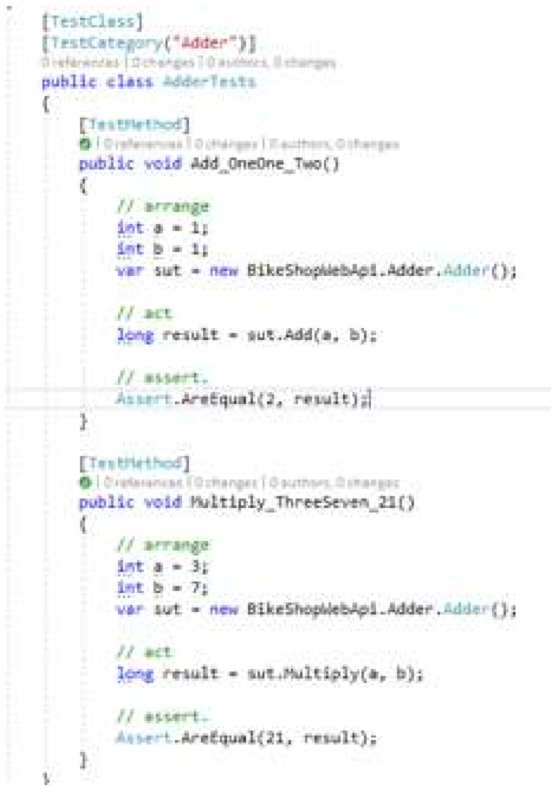
OUR DEMO PROJECT

BIKE SHOP API

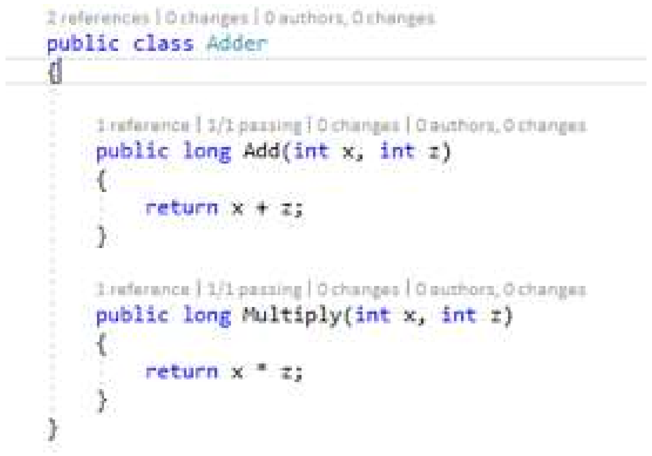
- Our Bike Shop Web API is built to handle REST calls from a separate front end service.
- It exposes a product service, which calls an old SOAP service and turns the results into JSON. This product service – due to how slow our SOAP service is – also needs a cache to store results.
- A cart service is also exposed, which calls a back-end database using Entity Framework.

THE BASIC TESTING PATTERN

- **Arrange** – create all the preconditions of your test.
- **Act** – run your test.
- **Assert** – verify your test succeeded.



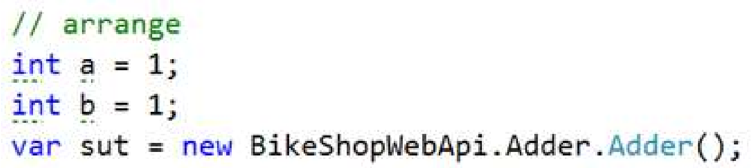
```
[TestClass]
[TestCategory("Adder")]
0 references | 0 changes | 0 authors, 0 changes
public class AdderTests
{
    [TestMethod]
    ● | 0 references | 0 changes | 0 authors, 0 changes
    public void Add_OneOne_Two()
    {
        // arrange
        int a = 1;
        int b = 1;
        var sut = new BikeShopWebApi.Adder.Adder();

        // act
        long result = sut.Add(a, b);

        // assert.
        Assert.AreEqual(2, result);
    }

    [TestMethod]
    ● | 0 references | 0 changes | 0 authors, 0 changes
    public void Multiply_ThreeSeven_21()
    {
        // arrange
        int a = 3;
        int b = 7;
        var sut = new BikeShopWebApi.Adder.Adder();

        // act
        long result = sut.Multiply(a, b);

        // assert.
        Assert.AreEqual(21, result);
    }
}
```



```
2 references | 0 changes | 0 authors, 0 changes
public class Adder
{
    1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes
    public long Add(int x, int z)
    {
        return x + z;
    }

    1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes
    public long Multiply(int x, int z)
    {
        return x * z;
    }
}
```

ARRANGE PHASE

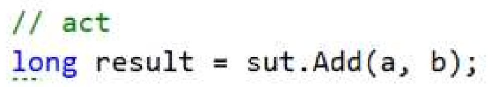
Set up all the preconditions of your test. This can include any data and mocks you need.



```
// arrange
int a = 1;
int b = 1;
var sut = new BikeShopWebApi.Adder.Adder();
```

ACT PHASE

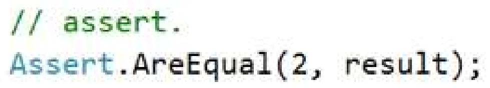
Executing your test



```
// act
long result = sut.Add(a, b);
```

ASSERT PHASE

Validate your behavior



```
// assert.
Assert.AreEqual(2, result);
```

NAMING CONVENTIONS

- “Only two hard problems in computer science: naming conventions, off by one errors, and cache consistency.”
- Pick something and go with it. I use Method_Parameters_Result, but anything works.

TOOLS OF THE TRADE

Using MSTest as our testing framework.
https://docs.microsoft.com/en-us/visualstudio/test/unit-test-your-code

Using AutoFixture as a mocking container.
https://github.com/AutoFixture/AutoFixture

Using Moq as our mocking library.
https://github.com/moq/moq4

THE HOWS

MSTEST

Microsoft official testing framework, enabled through attributes placed on classes.

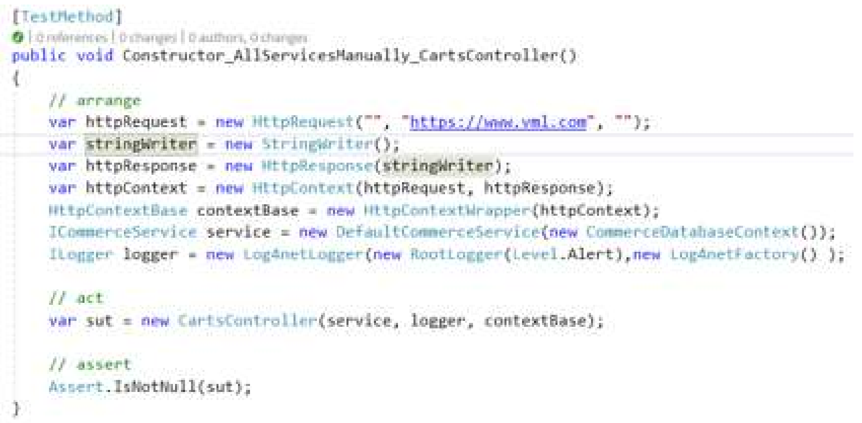
ATTRIBUTE	USE
TESTCLASS	PLACED ON THE CLASS LEVEL; MARKS IT AS CONTAINING TESTS.
TESTMETHOD	MARKS AN ATTRIBUTE AS TESTING.
REFLECTED TESTCATEGORY	CATEGORIES A GROUP AS A TEST IN THE TEST EXPLORER.
TESTINITIALIZE	MARKS A METHOD THAT IS RUN BEFORE EVERY TEST.
TESTCLEANUP	MARKS A METHOD THAT IS RUN AFTER EVERY TEST.
EXPECTEDEXCEPTION	MARKS THE METHOD AS EXPECTING A CERTAIN TYPE OF EXCEPTION

THE HOWS

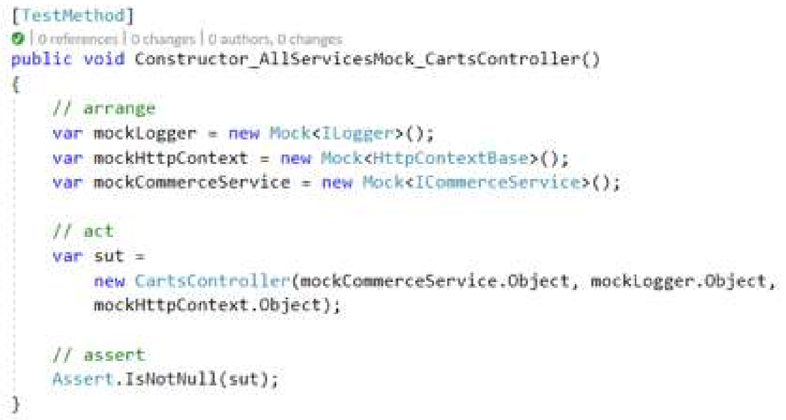
MOCKING

“A test Mock is a object setup in a specific way for the calls the are about to receive." – Martin Fowler

We use [Moq](https://github.com/moq/moq4) (https://github.com/moq/moq4).



```
[TestMethod]
| 0 references | 0 changes | 0 authors, 0 changes
public void Constructor_AllServicesManually_CartsController()
{
    // arrange
    var httpRequest = new HttpRequest("", "https://www.vml.com", "");
    var stringWriter = new StringWriter();
    var httpResponse = new HttpResponse(stringWriter);
    var httpContext = new HttpContext(httpRequest, httpResponse);
    HttpContextBase contextBase = new HttpContextWrapper(httpContext);
    ICommerceService service = new DefaultCommerceService(new CommerceDatabaseContext());
    ILogger logger = new Log4netLogger(new RootLogger(Level.Alert),new Log4netFactory() );

    // act
    var sut = new CartsController(service, logger, contextBase);

    // assert
    Assert.IsNotNull(sut);
}
```

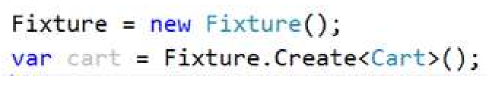


```
[TestMethod]
| 0 references | 0 changes | 0 authors, 0 changes
public void Constructor_AllServicesMock_CartsController()
{
    // arrange
    var mockLogger = new Mock<ILogger>();
    var mockHttpContext = new Mock<HttpContextBase>();
    var mockCommerceService = new Mock<ICommerceService>();

    // act
    var sut =
        new CartsController(mockCommerceService.Object, mockLogger.Object,
        mockHttpContext.Object);

    // assert
    Assert.IsNotNull(sut);
}
```

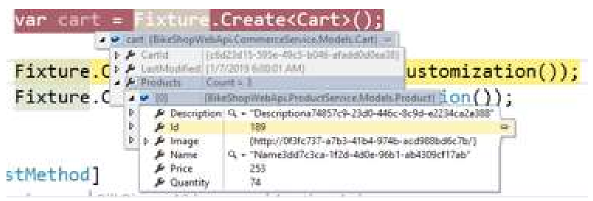
THE HOWS

AUTOFIXTURE

A test data container, designed to minimize the amount of “arrange” code you need

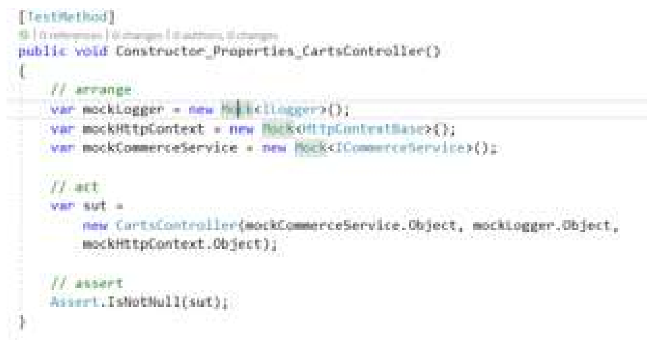


```
Fixture = new Fixture();
var cart = Fixture.Create<Cart>();
```

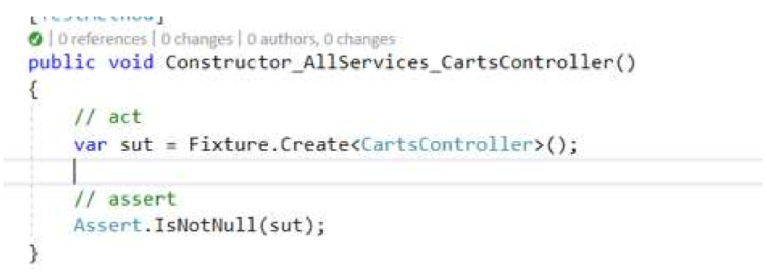


```
var cart = Fixture.Create<Cart>();
  cart {BikeShopWebApi.CommerceService.Models.Cart}
    CartId        {c6d23d15-595e-40c5-b046-afadd0d0ea28}
    LastModified  {1/7/2019 6:00:01 AM}
    Products      Count = 3
      [0]         {BikeShopWebApi.ProductService.Models.Product}
        Description  "Descriptiona74857c9-23d0-446c-8c9d-e2234ca2a388"
        Id           189
        Image        {http://0f3fc737-a7b3-41b4-974b-acd988bd6c7b/}
        Name         "Name3dd7c3ca-1f2d-4d0e-96b1-ab4309cf17ab"
        Price        253
        Quantity     74
Fixture.C                                  ustomization());
Fixture.C                                  ion());
stMethod]
```

Reduces test maintenance



```
[TestMethod]
0 references | 0 changes | 0 authors, 0 changes
public void Constructor_Properties_CartsController()
{
    // arrange
    var mockLogger = new Mock<ILogger>();
    var mockHttpContext = new Mock<HttpContextBase>();
    var mockCommerceService = new Mock<ICommerceService>();

    // act
    var sut =
        new CartsController(mockCommerceService.Object, mockLogger.Object,
        mockHttpContext.Object);

    // assert
    Assert.IsNotNull(sut);
}
```

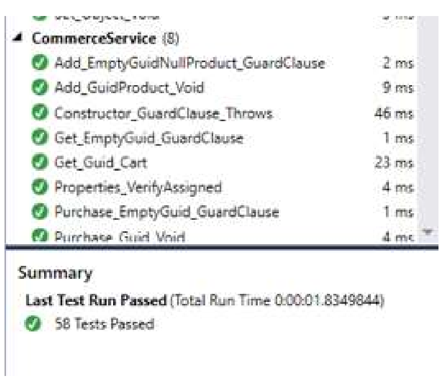


```
[TestMethod]
0 references | 0 changes | 0 authors, 0 changes
public void Constructor_AllServices_CartsController()
{
    // act
    var sut = Fixture.Create<CartsController>();

    // assert
    Assert.IsNotNull(sut);
}
```

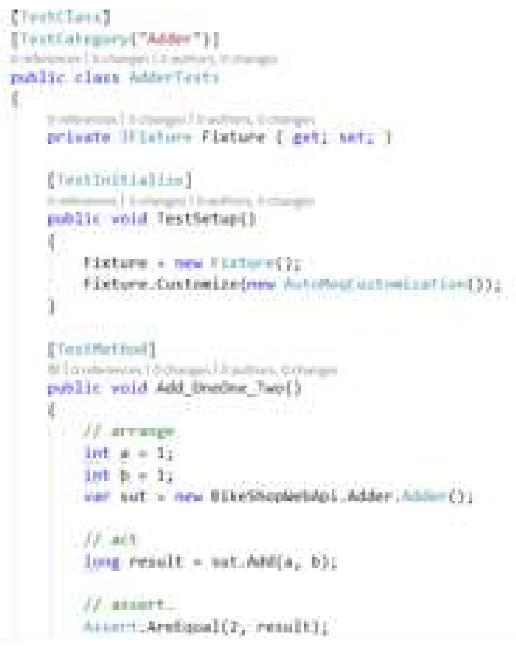
MSTEST TIPS

BASIC STRUCTURE

- Project is created as a tests project and references NuGet packages.
- Use the referenced attributes to make your tests discoverable by the test engine.
- Use the Test Explorer to run your tests.



```
◢ CommerceService (8)
  ✓ Add_EmptyGuidNullProduct_GuardClause     2 ms
  ✓ Add_GuidProduct_Void                     9 ms
  ✓ Constructor_GuardClause_Throws          46 ms
  ✓ Get_EmptyGuid_GuardClause                1 ms
  ✓ Get_Guid_Cart                           23 ms
  ✓ Properties_VerifyAssigned                4 ms
  ✓ Purchase_EmptyGuid_GuardClause           1 ms
  ✓ Purchase_Guid_Void                       4 ms

Summary
 Last Test Run Passed (Total Run Time 0:00:01.8349844)
 ✓ 58 Tests Passed
```

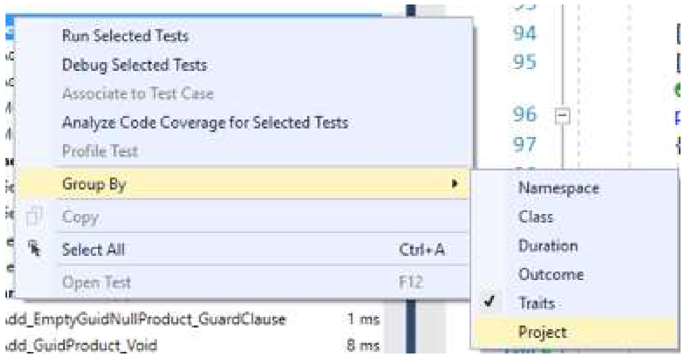


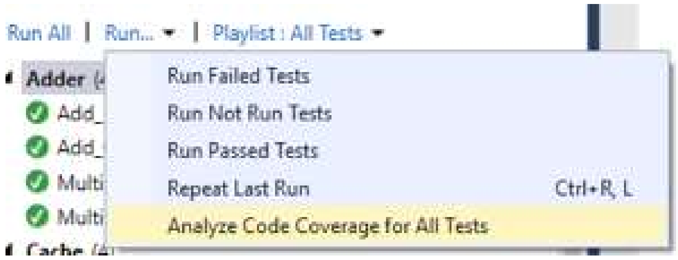
```
[TestClass]
[TestCategory("Adder")]
0 references | 0 changes | 0 authors, 0 changes
public class AdderTests
{
    0 references | 0 changes | 0 authors, 0 changes
    private IFixture Fixture { get; set; }

    [TestInitialize]
    0 references | 0 changes | 0 authors, 0 changes
    public void TestSetup()
    {
        Fixture = new Fixture();
        Fixture.Customize(new AutoMoqCustomization());
    }

    [TestMethod]
    0 | 0 references | 0 changes | 0 authors, 0 changes
    public void Add_OneOne_Two()
    {
        // arrange
        int a = 1;
        int b = 1;
        var sut = new BikeShopWebApi.Adder.Adder();

        // act
        long result = sut.Add(a, b);

        // assert
        Assert.AreEqual(2, result);
```

MSTEST TIPS

TEST EXPLORER

- Allows grouping by specific project, categories and classes.
- Lets you run various combinations of tests.

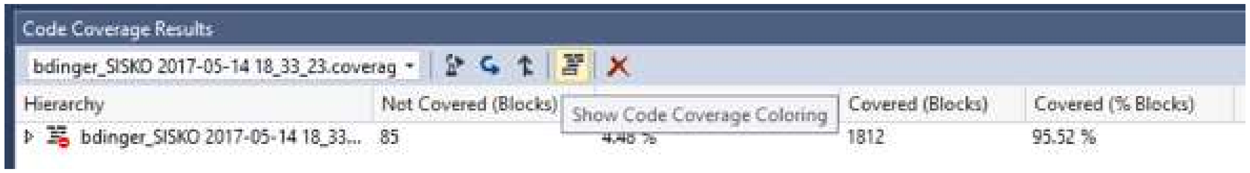



Run All | Run... | Playlist : All Tests
Adder
Add_
Add_
Multi
Multi
Cache
Run Failed Tests
Run Not Run Tests
Run Passed Tests
Repeat Last Run Ctrl+R, L
Analyze Code Coverage for All Tests

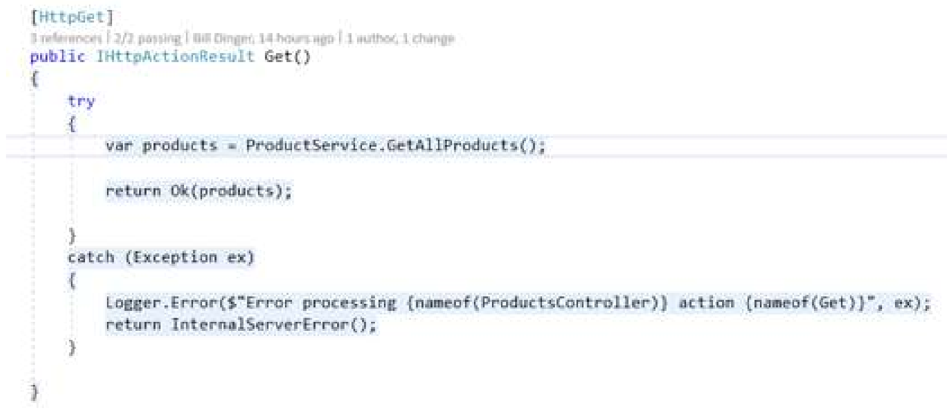
MSTEST TIPS

CODE COVERAGE HIGHLIGHTING

TEST – Analyze Code Coverage – All Tests


Code Coverage Results
bdinger_SISKO 2017-05-14 18_33_23.coverag
Show Code Coverage Coloring

Hierarchy	Not Covered (Blocks)		Covered (Blocks)	Covered (% Blocks)
bdinger_SISKO 2017-05-14 18_33...	85	4.48 %	1812	95.52 %



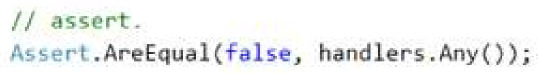
```
[HttpGet]
3 references | 2/2 passing | Bill Dinger, 14 hours ago | 1 author, 1 change
public IHttpActionResult Get()
{
    try
    {
        var products = ProductService.GetAllProducts();

        return Ok(products);

    }
    catch (Exception ex)
    {
        Logger.Error($"Error processing {nameof(ProductsController)} action {nameof(Get)}", ex);
        return InternalServerError();
    }

}
```

MSTEST TIPS

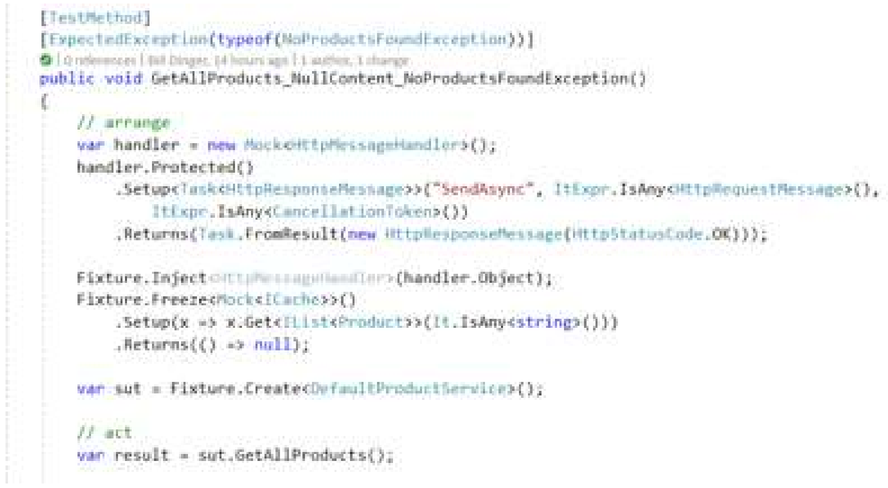
THE ASSERT CLAUSES

- All contained under the Microsoft.VisualStudio.TestTools.UnitTesting namespace.

- CollectionAssert, StringAssert and ExpectedException give you more flexibility and options for assertions.



```
// assert.
Assert.AreEqual(false, handlers.Any());
```

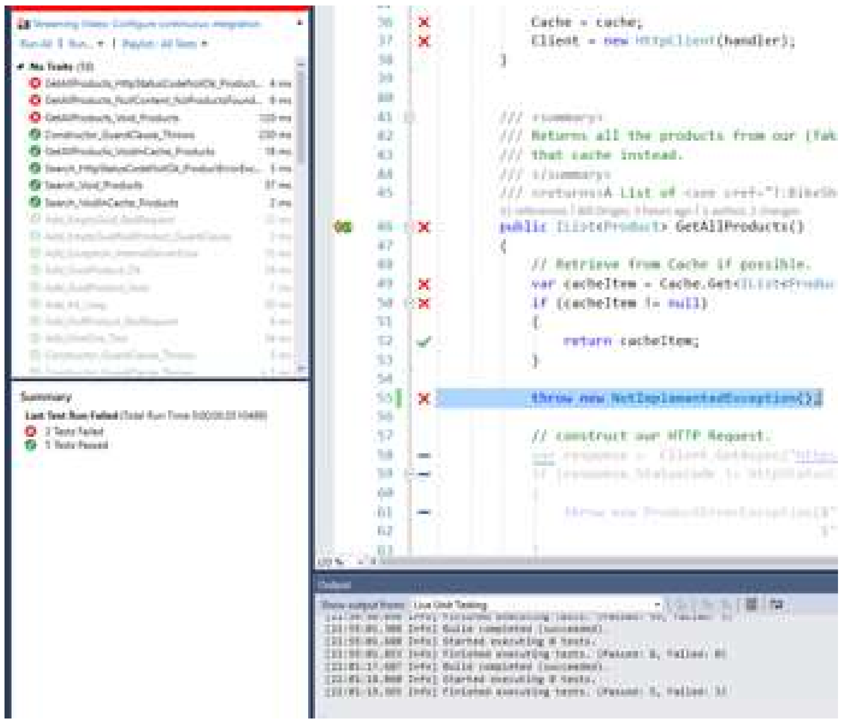


```
[TestMethod]
[ExpectedException(typeof(NoProductsFoundException))]
0 references | Bill Dinger, 14 hours ago | 1 author, 1 change
public void GetAllProducts_NullContent_NoProductsFoundException()
{
    // arrange
    var handler = new Mock<HttpMessageHandler>();
    handler.Protected()
        .Setup<Task<HttpResponseMessage>>("SendAsync", ItExpr.IsAny<HttpRequestMessage>(),
            ItExpr.IsAny<CancellationToken>())
        .Returns(Task.FromResult(new HttpResponseMessage(HttpStatusCode.OK)));

    Fixture.Inject<HttpMessageHandler>(handler.Object);
    Fixture.Freeze<Mock<ICache>>()
        .Setup(x => x.Get<IList<Product>>(It.IsAny<string>()))
        .Returns(() => null);

    var sut = Fixture.Create<DefaultProductService>();

    // act
    var result = sut.GetAllProducts();
```

MSTEST TIPS

LIVE UNIT TESTING

- Available in 2017 Enterprise edition.
- As you change classes, it continuously runs unit tests.



```
        Cache = cache;
        Client = new HttpClient(handler);
    }


    /// <summary>
    /// Returns all the products from our (fak
    /// that cache instead.
    /// </summary>
    /// <returns>A List of <see cref="T:BikeSh
    public IList<Product> GetAllProducts()
    {
        // Retrieve from Cache if possible.
        var cacheItem = Cache.Get<IList<Produ
        if (cacheItem != null)
        {
            return cacheItem;
        }

        throw new NotImplementedException();

        // construct our HTTP Request.
        var response = Client.GetAsync("http
        if (response.StatusCode != HttpStatusC
        {
            throw new ProductsErrorException($"
                $"
```

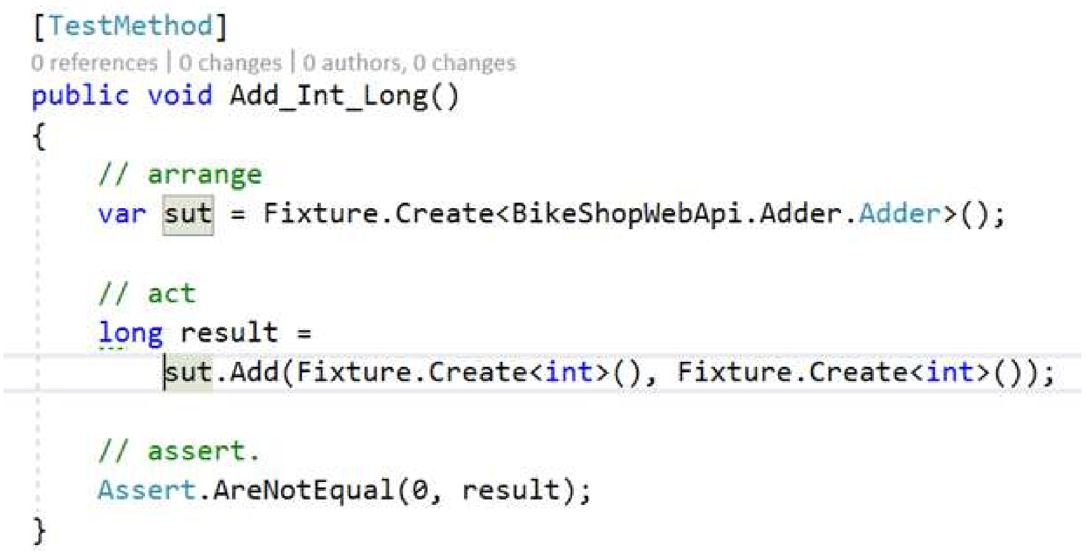
Summary
Last Test Run Failed (Total Run Time [illegible])
2 Tests Failed
1 Tests Passed

```
[22:59:05.300 Info] Build completed (Succeeded).
[22:59:05.400 Info] Started executing 8 tests.
[22:59:05.833 Info] Finished executing tests. (Passed: 8, Failed: 0)
[22:01:17.607 Info] Build completed (Succeeded).
[22:01:18.000 Info] Started executing 8 tests.
[22:01:19.305 Info] Finished executing tests. (Passed: 5, Failed: 3)
```

AUTOFIXTURE TIPS

OBJECT CREATION

AutoFixture simplifies arrange method by creating types using built-in creation algorithms.

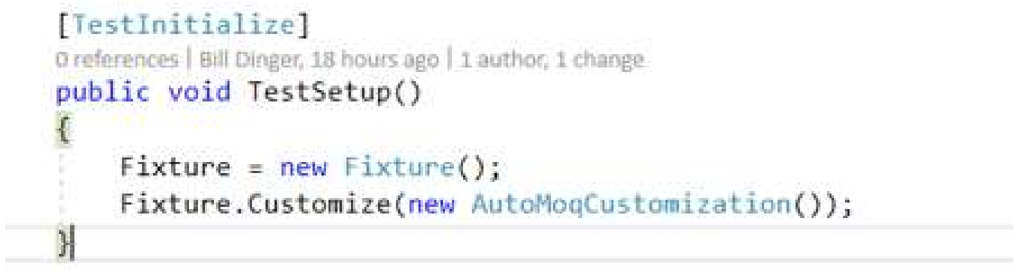


```
[TestMethod]
0 references | 0 changes | 0 authors, 0 changes
public void Add_Int_Long()
{
    // arrange
    var sut = Fixture.Create<BikeShopWebApi.Adder.Adder>();

    // act
    long result =
        sut.Add(Fixture.Create<int>(), Fixture.Create<int>());

    // assert.
    Assert.AreNotEqual(0, result);
}
```

AUTOFIXTURE TIPS

AUTOMOCK

Uses the AutoMoqCustomization customization and NuGet package to let AutoFixture Moq interfaces, abstract classes and classes without public constructors automatically.



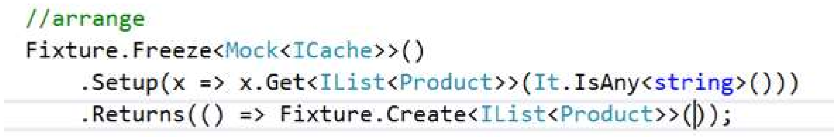
```
[TestInitialize]
0 references | Bill Dinger, 18 hours ago | 1 author, 1 change
public void TestSetup()
{
    Fixture = new Fixture();
    Fixture.Customize(new AutoMoqCustomization());
}
```

AUTOFIXTURE TIPS

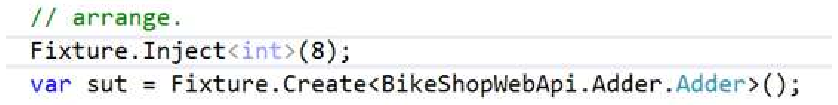
FREEZE AND INJECT

- Use Fixture.Freeze<T> to tell AutoFixture’s creation algorithm to use that specific behavior every time an object is created.

- Use Fixture.Inject<T> to tell AutoFixture to use that instance every time an object is created.



```
//arrange
Fixture.Freeze<Mock<ICache>>()
    .Setup(x => x.Get<IList<Product>>(It.IsAny<string>()))
    .Returns(() => Fixture.Create<IList<Product>>());
```

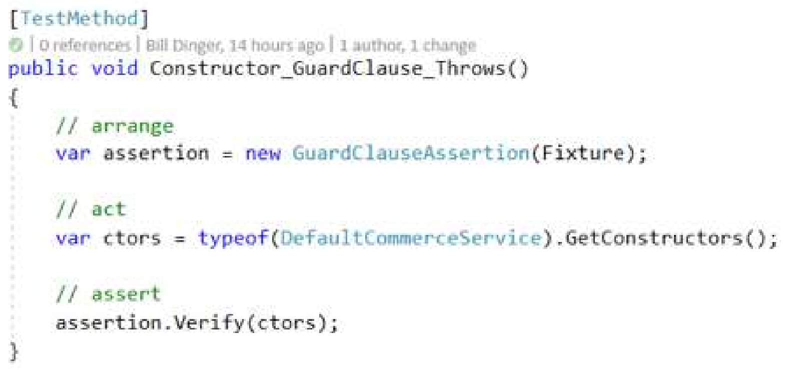


```
// arrange.
Fixture.Inject<int>(8);
var sut = Fixture.Create<BikeShopWebApi.Adder.Adder>();
```

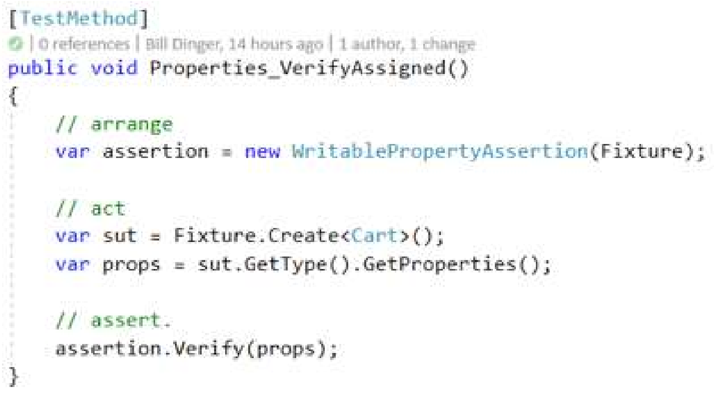
AUTOFIXTURE TIPS

IDIOMS

AutoFixture Idioms allow you to test constructor and method guard clauses easier with automated assertions. Also things such as property assignments.



```
[TestMethod]
| 0 references | Bill Dinger, 14 hours ago | 1 author, 1 change
public void Constructor_GuardClause_Throws()
{
    // arrange
    var assertion = new GuardClauseAssertion(Fixture);

    // act
    var ctors = typeof(DefaultCommerceService).GetConstructors();

    // assert
    assertion.Verify(ctors);
}
```

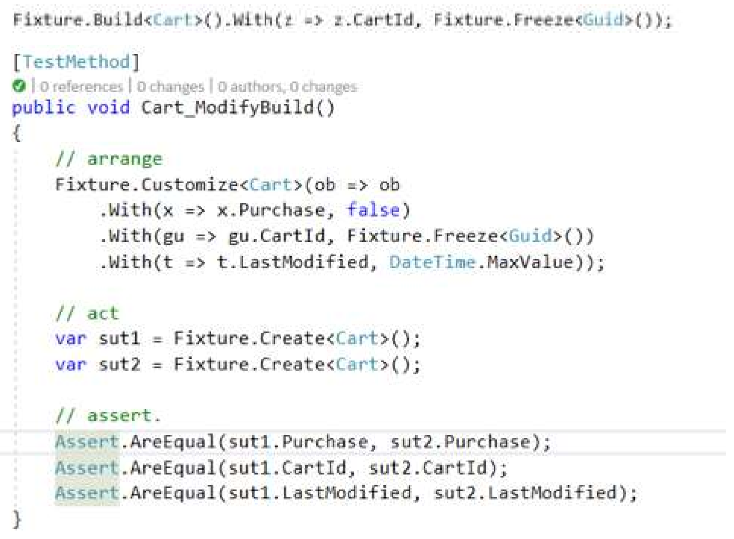


```
[TestMethod]
| 0 references | Bill Dinger, 14 hours ago | 1 author, 1 change
public void Properties_VerifyAssigned()
{
    // arrange
    var assertion = new WritablePropertyAssertion(Fixture);

    // act
    var sut = Fixture.Create<Cart>();
    var props = sut.GetType().GetProperties();

    // assert.
    assertion.Verify(props);
}
```

AUTOFIXTURE TIPS

CUSTOMIZE

Lets you customize object creation, overriding the build script that AutoFixture normally calls.

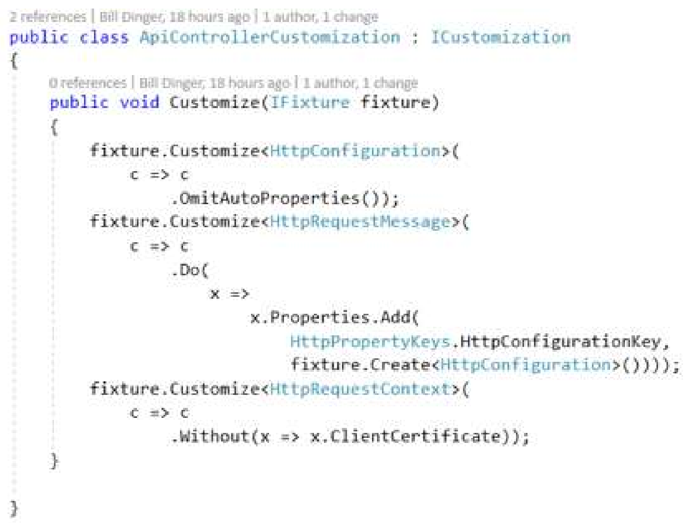


```
Fixture.Build<Cart>().With(z => z.CartId, Fixture.Freeze<Guid>());

[TestMethod]
| 0 references | 0 changes | 0 authors, 0 changes
public void Cart_ModifyBuild()
{
    // arrange
    Fixture.Customize<Cart>(ob => ob
        .With(x => x.Purchase, false)
        .With(gu => gu.CartId, Fixture.Freeze<Guid>())
        .With(t => t.LastModified, DateTime.MaxValue));

    // act
    var sut1 = Fixture.Create<Cart>();
    var sut2 = Fixture.Create<Cart>();

    // assert.
    Assert.AreEqual(sut1.Purchase, sut2.Purchase);
    Assert.AreEqual(sut1.CartId, sut2.CartId);
    Assert.AreEqual(sut1.LastModified, sut2.LastModified);
}
```

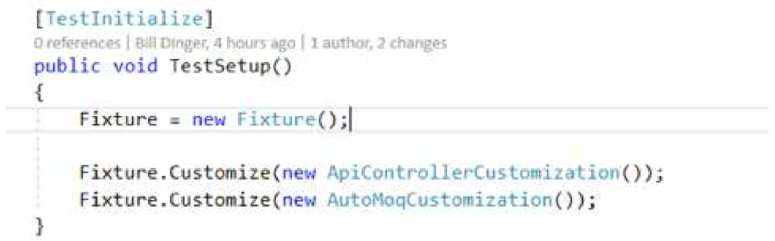
AUTOFIXTURE TIPS

CUSTOMIZATIONS

Supplied to the AutoFixture container to help control item creation. For example, our ApiControllerCustomization.



```
2 references | Bill Dinger, 18 hours ago | 1 author, 1 change
public class ApiControllerCustomization : ICustomization
{
    0 references | Bill Dinger, 18 hours ago | 1 author, 1 change
    public void Customize(IFixture fixture)
    {
        fixture.Customize<HttpConfiguration>(
            c => c
                .OmitAutoProperties());
        fixture.Customize<HttpRequestMessage>(
            c => c
                .Do(
                    x =>
                        x.Properties.Add(
                            HttpPropertyKeys.HttpConfigurationKey,
                            fixture.Create<HttpConfiguration>())));
        fixture.Customize<HttpRequestContext>(
            c => c
                .Without(x => x.ClientCertificate));
    }

}
```

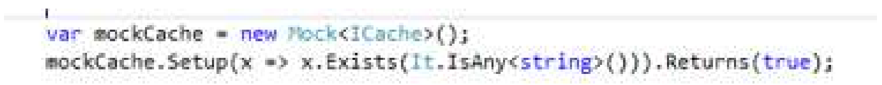


```
[TestInitialize]
0 references | Bill Dinger, 4 hours ago | 1 author, 2 changes
public void TestSetup()
{
    Fixture = new Fixture();

    Fixture.Customize(new ApiControllerCustomization());
    Fixture.Customize(new AutoMoqCustomization());
}
```

MOQ TIPS

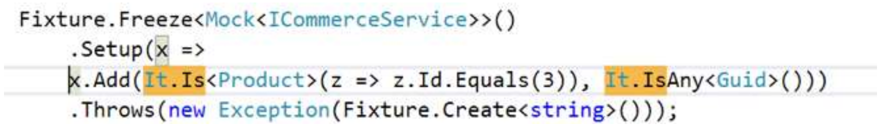
MATCHING VALUES WITH IT.IS

- Match any value and type with It.IsAny<T>



```
var mockCache = new Mock<ICache>();
mockCache.Setup(x => x.Exists(It.IsAny<string>())).Returns(true);
```

- Match a specific value with It.Is

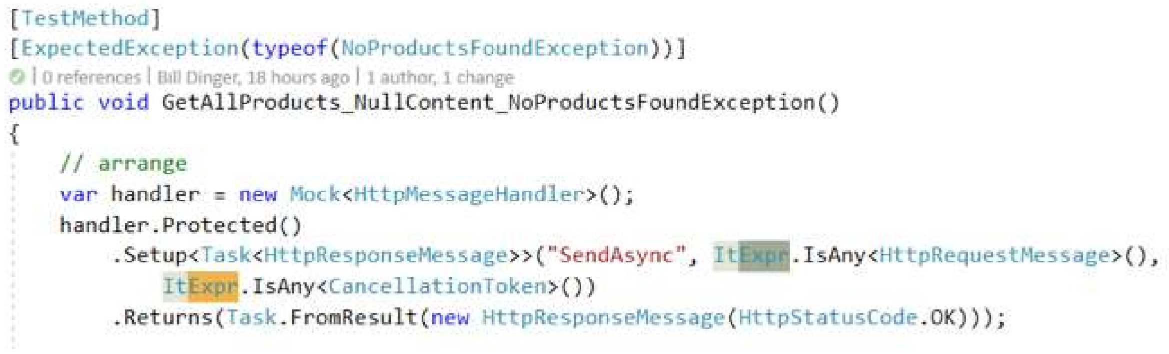


```
Fixture.Freeze<Mock<ICommerceService>>()
    .Setup(x =>
    x.Add(It.Is<Product>(z => z.Id.Equals(3)), It.IsAny<Guid>()))
    .Throws(new Exception(Fixture.Create<string>()));
```

MOQ TIPS

MOCKING PROTECTED MEMBERS

Use the .Protected keyword. No intellisense; must rely on string matching on the method name.

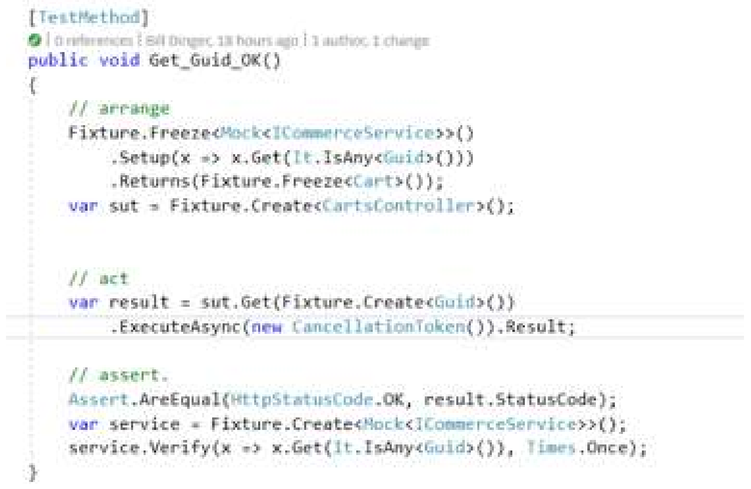


```
[TestMethod]
[ExpectedException(typeof(NoProductsFoundException))]
0 references | Bill Dinger, 18 hours ago | 1 author, 1 change
public void GetAllProducts_NullContent_NoProductsFoundException()
{
    // arrange
    var handler = new Mock<HttpMessageHandler>();
    handler.Protected()
        .Setup<Task<HttpResponseMessage>>("SendAsync", ItExpr.IsAny<HttpRequestMessage>(),
            ItExpr.IsAny<CancellationToken>())
        .Returns(Task.FromResult(new HttpResponseMessage(HttpStatusCode.OK)));
```

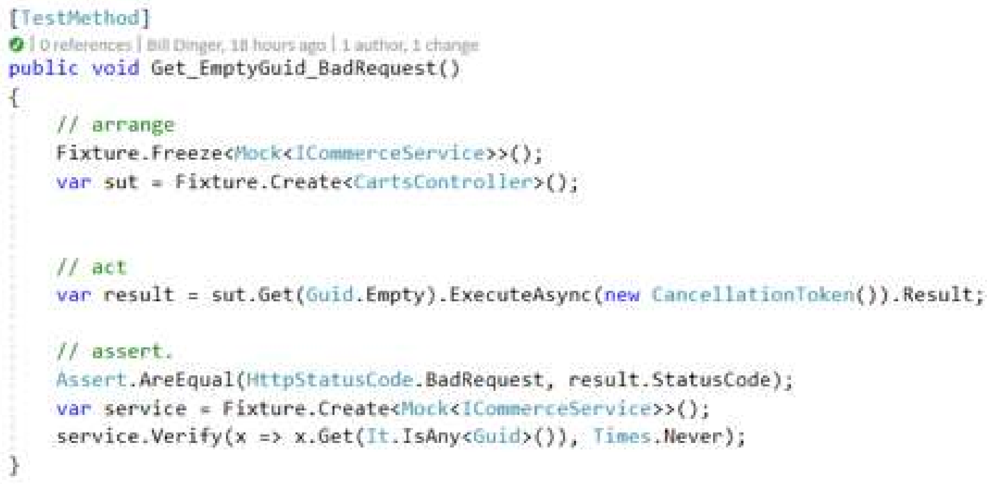
MOQ TIPS

VERIFY SERVICES ARE CALLED

Use *verify* to determine if a service has been called *n* number of times.



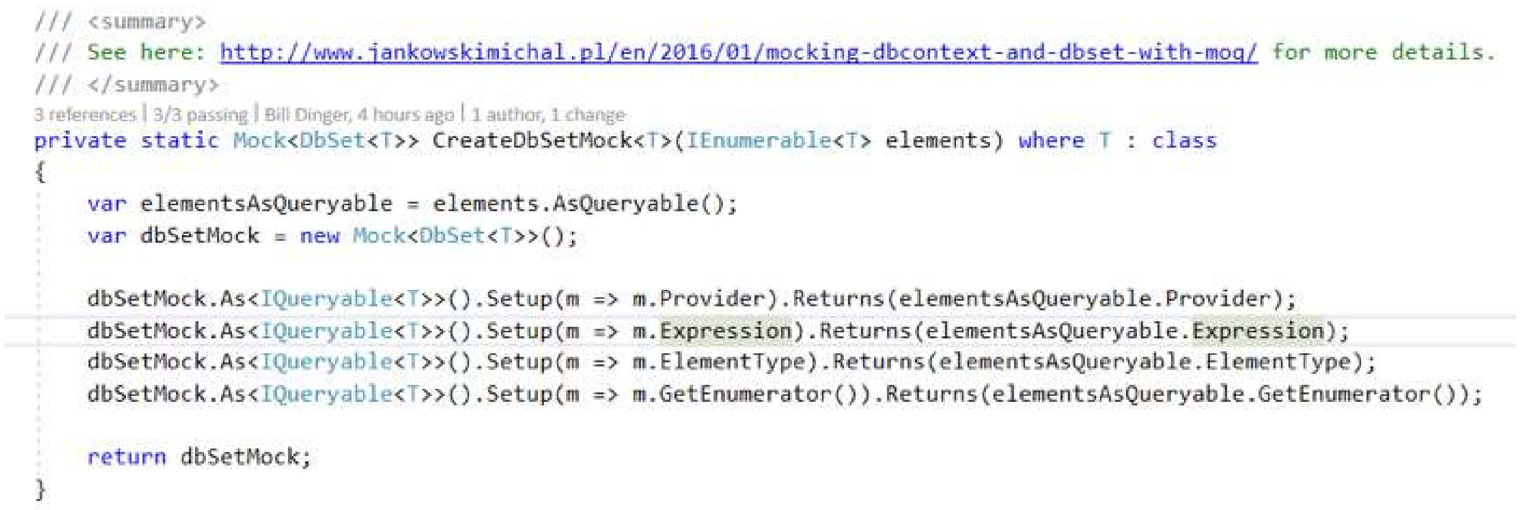
```
[TestMethod]
| 0 references | Bill Dinger, 18 hours ago | 1 author, 1 change
public void Get_Guid_OK()
{
    // arrange
    Fixture.Freeze<Mock<ICommerceService>>()
        .Setup(x => x.Get(It.IsAny<Guid>()))
        .Returns(Fixture.Freeze<Cart>());
    var sut = Fixture.Create<CartsController>();


    // act
    var result = sut.Get(Fixture.Create<Guid>())
        .ExecuteAsync(new CancellationToken()).Result;

    // assert.
    Assert.AreEqual(HttpStatusCode.OK, result.StatusCode);
    var service = Fixture.Create<Mock<ICommerceService>>();
    service.Verify(x => x.Get(It.IsAny<Guid>()), Times.Once);
}
```



```
[TestMethod]
| 0 references | Bill Dinger, 18 hours ago | 1 author, 1 change
public void Get_EmptyGuid_BadRequest()
{
    // arrange
    Fixture.Freeze<Mock<ICommerceService>>();
    var sut = Fixture.Create<CartsController>();


    // act
    var result = sut.Get(Guid.Empty).ExecuteAsync(new CancellationToken()).Result;

    // assert.
    Assert.AreEqual(HttpStatusCode.BadRequest, result.StatusCode);
    var service = Fixture.Create<Mock<ICommerceService>>();
    service.Verify(x => x.Get(It.IsAny<Guid>()), Times.Never);
}
```

MOQ TIPS

MOCKING A DbSET

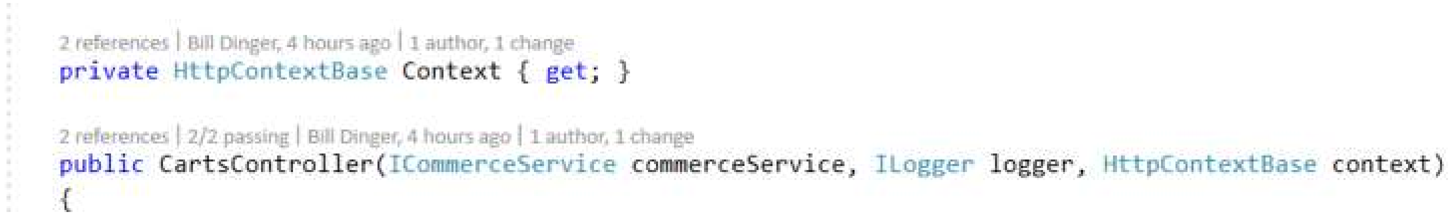


```
/// <summary>
/// See here: http://www.jankowskimichal.pl/en/2016/01/mocking-dbcontext-and-dbset-with-moq/ for more details.
/// </summary>
3 references | 3/3 passing | Bill Dinger, 4 hours ago | 1 author, 1 change
private static Mock<DbSet<T>> CreateDbSetMock<T>(IEnumerable<T> elements) where T : class
{
    var elementsAsQueryable = elements.AsQueryable();
    var dbSetMock = new Mock<DbSet<T>>();

    dbSetMock.As<IQueryable<T>>().Setup(m => m.Provider).Returns(elementsAsQueryable.Provider);
    dbSetMock.As<IQueryable<T>>().Setup(m => m.Expression).Returns(elementsAsQueryable.Expression);
    dbSetMock.As<IQueryable<T>>().Setup(m => m.ElementType).Returns(elementsAsQueryable.ElementType);
    dbSetMock.As<IQueryable<T>>().Setup(m => m.GetEnumerator()).Returns(elementsAsQueryable.GetEnumerator());

    return dbSetMock;
}
```

GENERAL TIPS

HANDLING HTTPCONTEXT

Rely on *abstraction,* not concrete

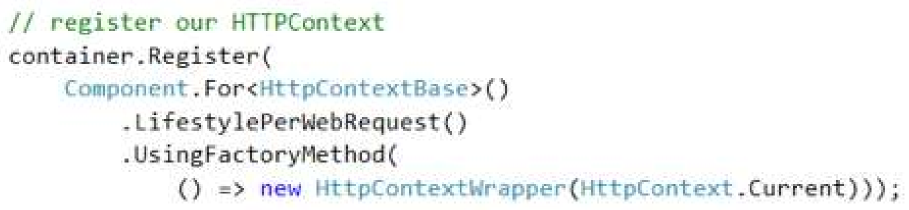


2 references | Bill Dinger, 4 hours ago | 1 author, 1 change

```
private HttpContextBase Context { get; }
```

2 references | 2/2 passing | Bill Dinger, 4 hours ago | 1 author, 1 change

```
public CartsController(ICommerceService commerceService, ILogger logger, HttpContextBase context)
{
```

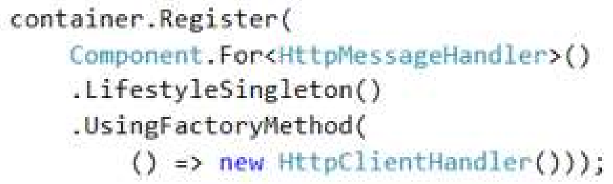


```
// register our HTTPContext
container.Register(
    Component.For<HttpContextBase>()
        .LifestylePerWebRequest()
        .UsingFactoryMethod(
            () => new HttpContextWrapper(HttpContext.Current)));
```

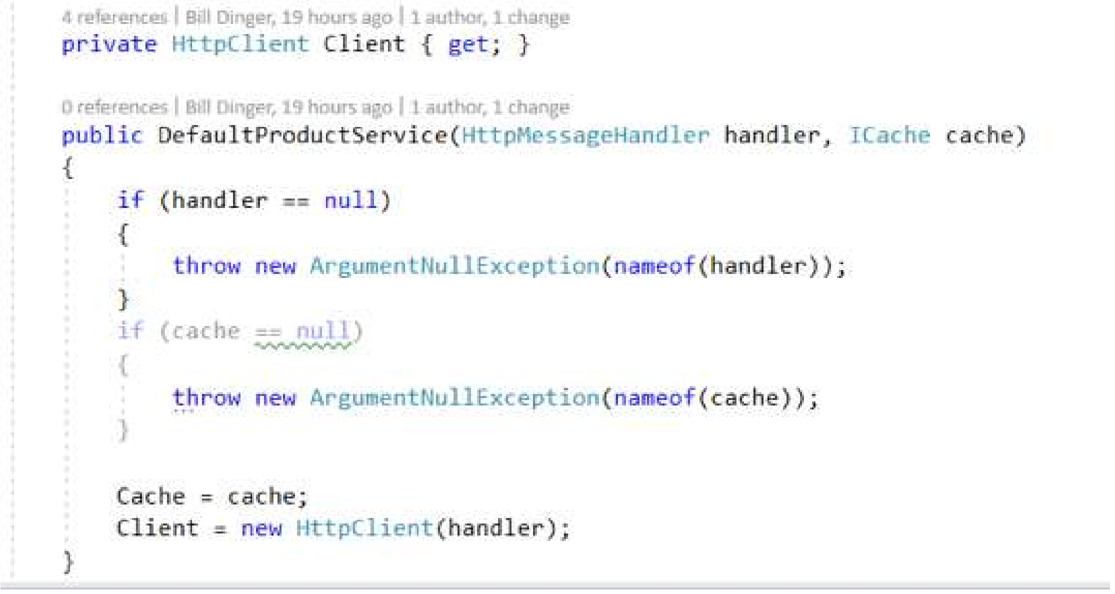
GENERAL TIPS

MOCKING HTTPCLIENT CALLS

Rely on HttpMessageHandler to inject into our Httpclient



```
container.Register(
    Component.For<HttpMessageHandler>()
    .LifestyleSingleton()
    .UsingFactoryMethod(
        () => new HttpClientHandler()));
```



```
4 references | Bill Dinger, 19 hours ago | 1 author, 1 change
private HttpClient Client { get; }

0 references | Bill Dinger, 19 hours ago | 1 author, 1 change
public DefaultProductService(HttpMessageHandler handler, ICache cache)
{
    if (handler == null)
    {
        throw new ArgumentNullException(nameof(handler));
    }
    if (cache == null)
    {
        throw new ArgumentNullException(nameof(cache));
    }

    Cache = cache;
    Client = new HttpClient(handler);
}
```

RESOURCES

MOQ — https://github.com/moq/moq4

AutoFixture — https://github.com/AutoFixture/AutoFixture

AutoFixture Cheat Sheet — https://github.com/AutoFixture/AutoFixture/wiki/Cheat-Sheet

Pluralsight Course on AutoFixture — https://app.pluralsight.com/courses/autofixture-dotnet-unit-test-get-started

MSTest Basics — https://docs.microsoft.com/en-us/visualstudio/test/unit-test-your-code

Testing in Visual Studio 2015 (Microsoft Virtual Academy) — https://mva.microsoft.com/en-US/training-courses/16459

Mark Seeman (maker of Autofixture) Posts on AutoFixture — http://blog.ploeh.dk/tags/#AutoFixture-ref

Mark Seeman on Encapsulation & Solid — https://app.pluralsight.com/library/courses/encapsulation-solid/table-of-contents

Applying SOLID principles in .NET/C# (Tech Ed 2014) — https://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DEV-B315

Introduction to Unit Testing (Tech Ed 2014) — https://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DEV-H213

THANK YOU.

VML

BILL DINGER
Solutions Architect | @adazlian
bill.dinger@vml.com