Sheridan | Get Creative

Advanced .NET Server Development: Testing

Instructor: Amandeep S. Patti

Objectives

- Intro to Testing
- Intro to Visual Studio Testing
- VS Test Explorer
- Testing Frameworks
- Microsoft Test Manager
- Unit Tests
- Coded UI Tests
- Performance and Load Testing
- Code Analysis
- Microsoft Fakes framework

Introduction to Testing

- Testing is a key part of developing any professional application
- Testing has traditionally been a manual, tedious and expensive process
- Agile development methodologies place even greater emphasis on testing than traditional approaches
- Agile development requires lots of repeat (regression) testing
  - so the need for test automation is very high
- Visual Studio has extensive support
  - for building and executing automated tests
- ASP.NET MVC and Web API were designed to be very testable
  - hence follow Test Driven Development (TDD)
- Testing is very important for your Capstone Projects.

Test Driven Development (TDD)

- An approach where tests are written before or at least in parallel with the actual code
- A major part of Agile Development
- Automated builds and automated tests are key parts of TDD and Continuous Integration strategies
- Visual Studio and ASP.NET MVC have excellent support for TDD.
- Typical sequence in TDD is:
  - RED – design a test that initially fails
  - GREEN – update the code under test just enough so that it passes
  - REFACTOR – refactor the code under test so that it is complete and rerun the test

Testing Frameworks

- Visual Studio provides an open testing architecture that supports several different test frameworks including
  - nUnit – original JUnit port
  - xUnit – updated version of NUnit
  - MSTest – Microsoft's unit testing framework which is included with Visual Studio
- The principles for these unit testing frameworks are very similar so transitioning from one to the other is easy
- We will be using MSTest to minimize installation hassles
- Other test adapters typically require an adapter
  - which is available from Extension Manager and/or project inclusion using NuGet

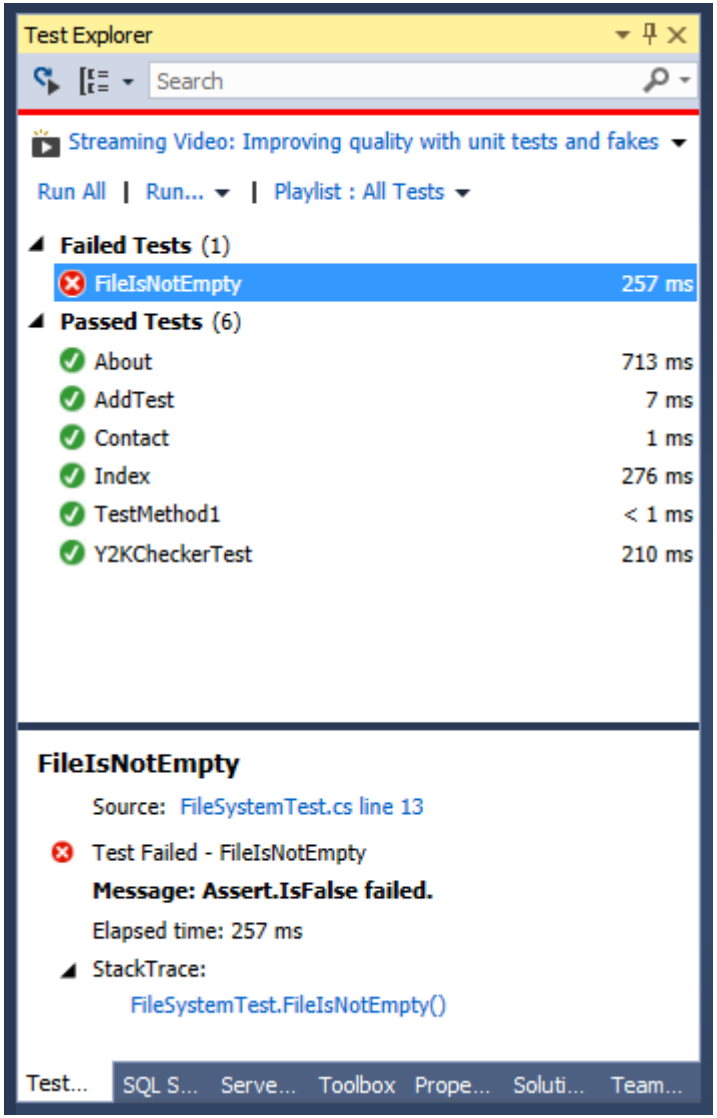
MSTest

- Included in Visual Studio
- Test Classes are annotated with [TestClass]
- Test actions (methods) are annotated with [TestMethod]
- Initialization code that runs before each test is annotated with [TestInitialize]
- Cleanup code that runs after each test is annotated with [TestCleanup]

```
namespace VS2013_Web1.Tests.Controllers
{
    [TestClass]
    -references
    public class HomeControllerTest
    {
        [TestMethod]
        -references
        public void Index()
        {
            // Arrange
            HomeController controller = new HomeController();

            // Act
            ViewResult result = controller.Index() as ViewResult;

            // Assert
            Assert.IsNotNull(result);
        }
```

Test Explorer

- Test Explorer is built into Visual Studio and allows us to:
  - View the existing unit tests
  - Search for tests
  - Execute tests individually or in batches
  - View the results of unit tests



Test Manager

- Test Manager is Microsoft's dedicated Test Tool
- Test Manager provides additional support for:
  - Managing lab environments (virtual machines)
  - Managing test plans
  - Manual testing
  - Detailed bug reporting
- It comes with Visual Studio 2013 Pro and up.
- You should consider using Test Manager for your Capstone projects.

- https://msdn.microsoft.com/en-us/library/jj635157.aspx

Video

- Let's watch a great video on MSTest (1.7) from the "Building Applications with ASP.NET MVC 4" series on Pluralsight

Video

- Let's watch a great video on Visual Studio testing from Channel 9

Unit Tests

- Unit Tests normally consists of three major steps:
  - Arrange – create an object to test
  - Act – do something with the object
  - Assert – check that the object or result changes to the expected state
- The Assert Class in the Microsoft.VisualStudio.TestTools.UnitTesting namespace is commonly used for performing the assertions mentioned above
- The Assert Class has many useful methods for assertions including:
  - IsTrue, IsFalse
  - AreEqual, AreNotEqual
  - AreSame, AreNotSame (to compare references)
  - IsNull, IsNotNull
  - Fail (to fail upon execution of this method)

Further Reading

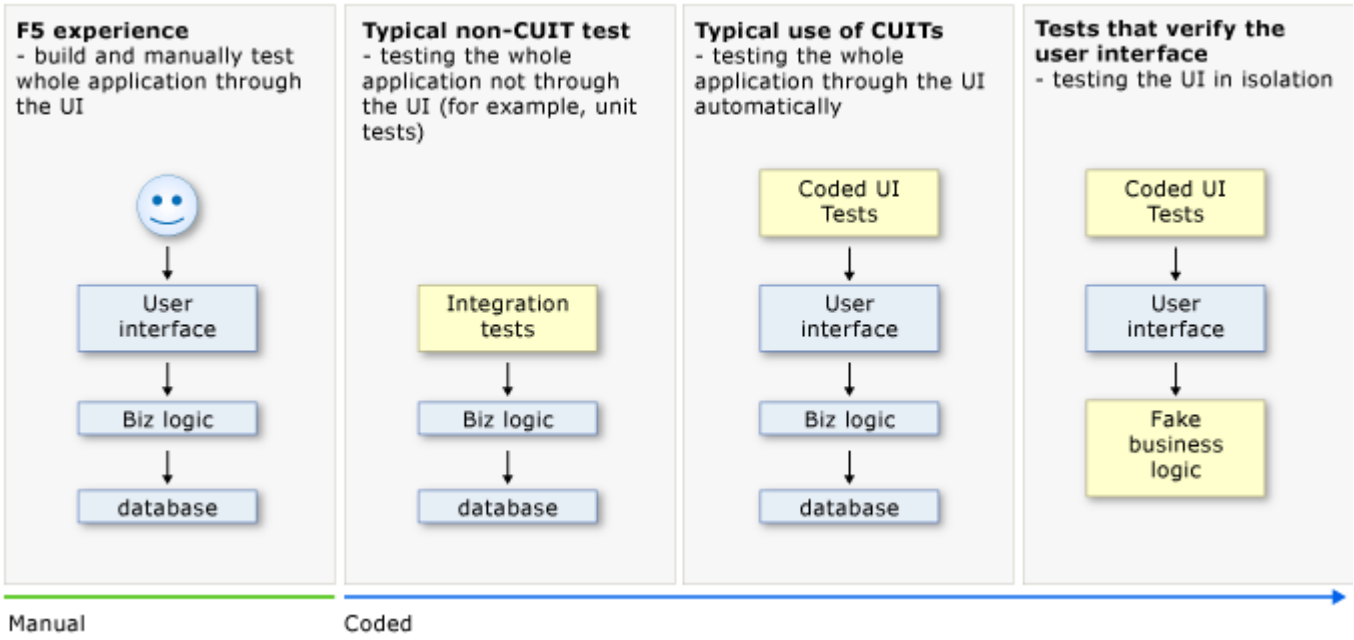
- Microsoft Virtual Academy
  - Software Testing with Visual Studio 2012 Jump Start
- Channel 9
  - Getting Started with Unit Testing Part 1
  - Getting Started with Unit Testing Part 2
  - Getting Started with Unit Testing Part 3

Testing ASP.NET MVC

- ASP.NET MVC and Web API were designed with Testability in mind
- Testing tends to focus on the Controller classes
- Controllers are just POCO classes
  - so they are easily tested
- Test support can be easily included in a separate project
  - by default when a new ASP.NET project is created

UI Automation Tests

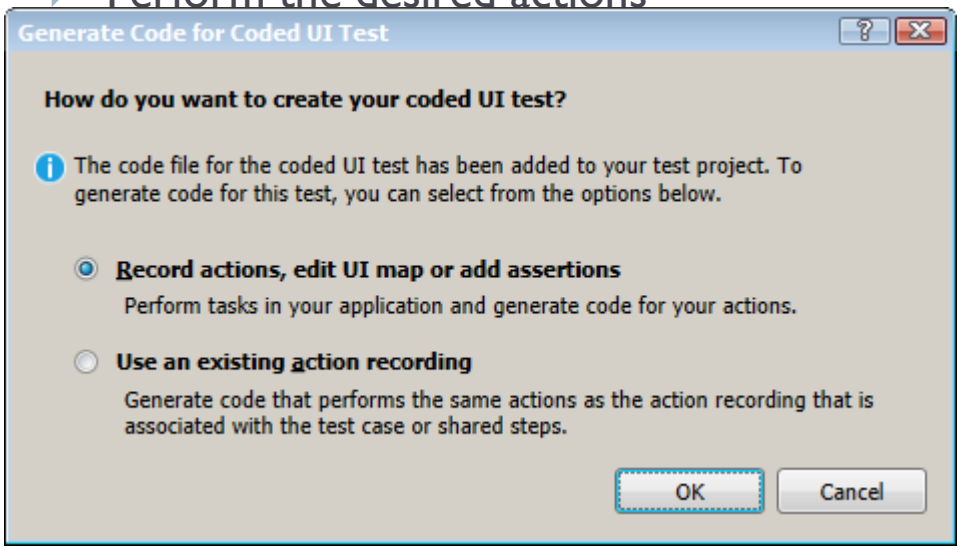
- Visual Studio also supports automated User Interface testing through Coded UI Tests (CUIT)
- These tests allow you to test the whole application from the UI to the backend database



"Verifying Code by Using UI Automation." Microsoft Developer Network. Microsoft, n.d. Web. 02 Mar. 2014. <http://msdn.microsoft.com/en-us/library/dd286726.aspx>.

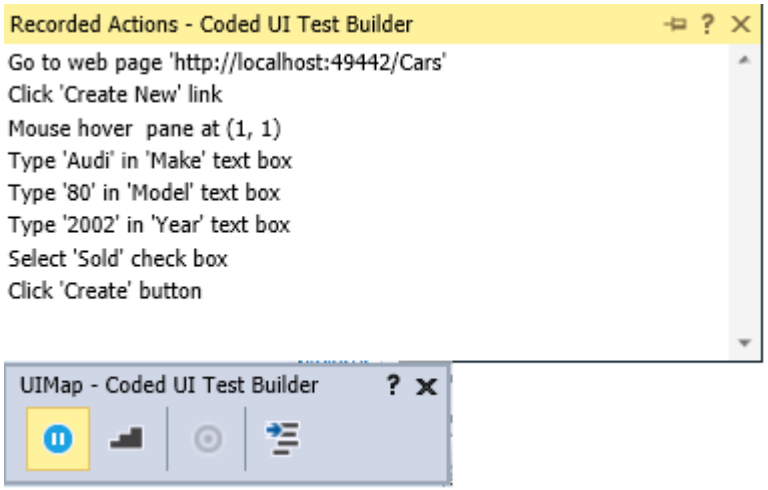
Coded UI Tests

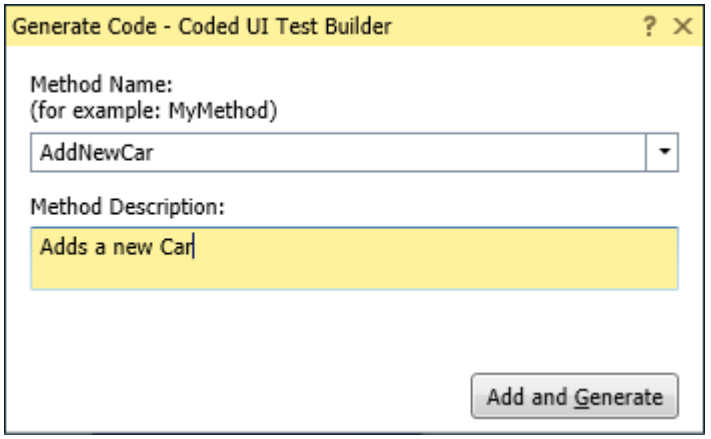
- The Coded UI Test Builder allows you to generate UI tests by recording the series of actions that you perform from the UI
- To Create a Coded UI Test (https://msdn.microsoft.com/en-us/library/dd286726.aspx):
  - Add a Coded UI Test to the project
  - From the dialog select "Record actions, …"
  - Run the application under test without debugging
  - Click the "Record" button from the Coded UI Test Builder Window
  - Perform the desired actions



Coded UI Tests (2)

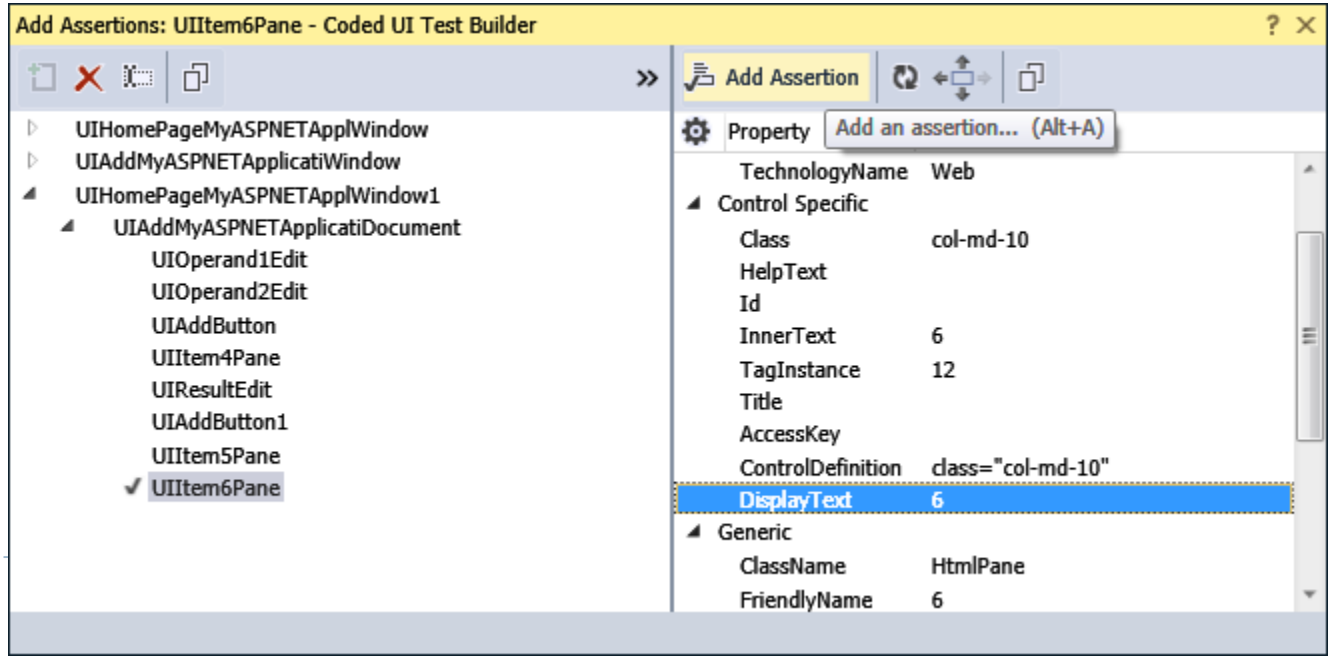
- Click the "Show Recorded Steps" button in the Test Buider Window and verify/adjust the recorded actions listed
- Click the "Generate Code" button and enter the method name and description and click the "Add and Generate" button.





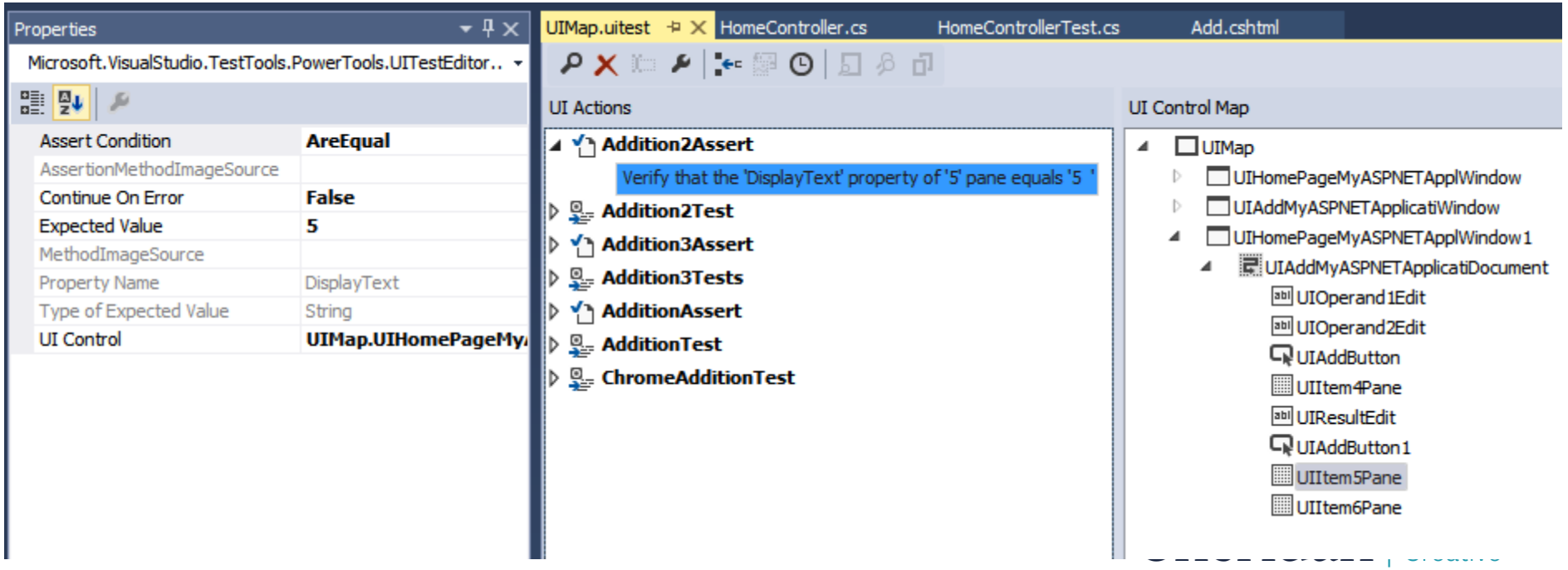
Coded UI Tests (3)

- Click the "Add Assertions" button and select the UI element to be validated (Ctrl + Shift + I).
- Click the "Generate Code" button and enter the method name and description and click the "Add and Generate" button.
- Select the Property of Interest from the Coded UI Test Builder and select the "Add Assertion" Button
- Enter the Assertion condition, Failure message and select OK
- Select the Generate Code Button, enter the method name and select OK
- For more details refer to MSDN



Coded UI Test Editor

- The Code UI Test Editor allows us to quickly change the actions and assertions associated with Code UI Tests
- The UIMap.uitest file stores the details of Coded UI Tests and is opened by the Code UI Test Editor by default
- Right Click and select Properties for any of the actions/assertions to change the details



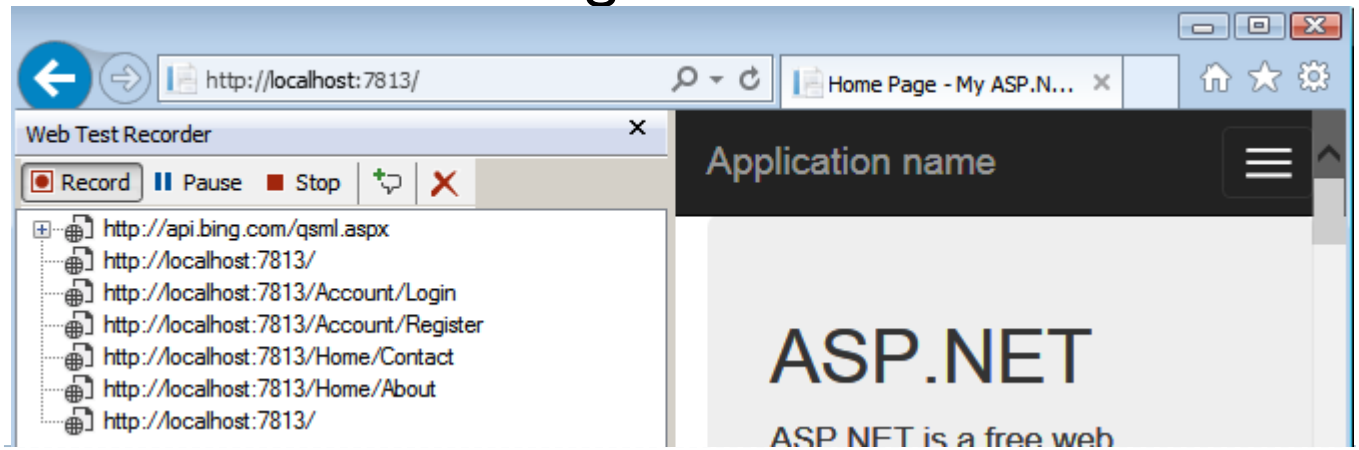
Data-driven Tests

- One Unit or Coded UI Test can be run repeated using different values by setting the DataSource attribute
- To access the "Input1" column for the current row of data from your data source:
  - string param1 = TestContext.DataRow["Input1"]
- Different data source types can be used including:
  - Excel
  - CSV
  - XML
  - SQL Server

- For more details, See MSDN (https://msdn.microsoft.com/library/ee624082(VS.110).aspx)

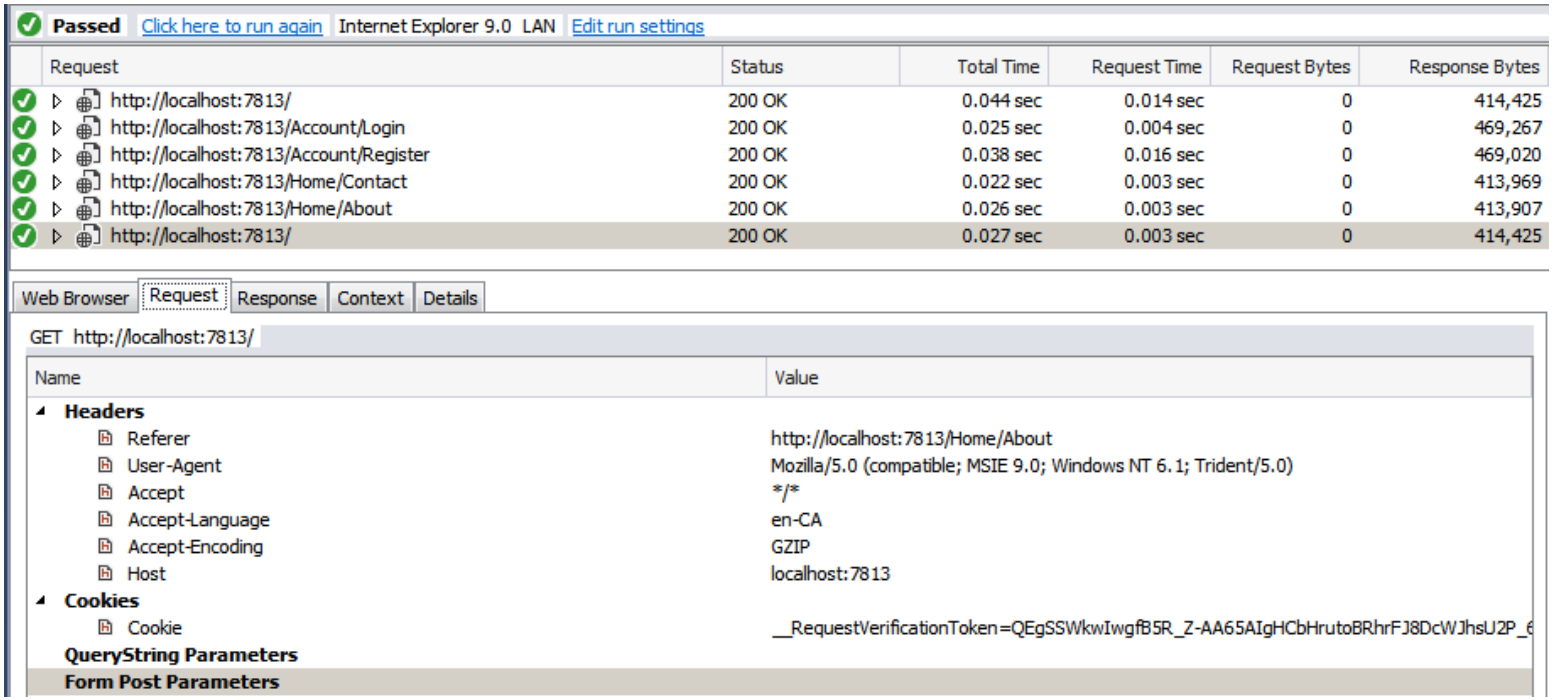
Web Performance Tests

- Can record a series of HTTP messages to/from the browser to define the test using the Web Test Recorder plugin in the browser
- Can run the test in a loop to get better statistics
- Used as building blocks for load testing an ASP.NET project
- Use the Release Configuration and disable Browser Link



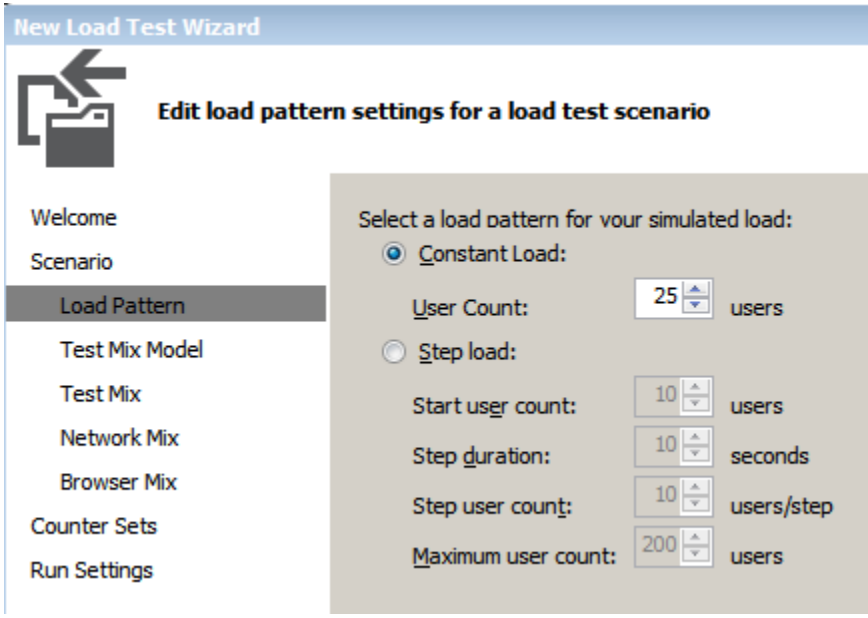
Web Performance Tests II

- When running these tests performance data is collected such as:
  - Request and Total Time required
  - Request and Response Bytes



Web Load Testing

- Can run many concurrent instances of existing Unit and Web Performance tests to see how the system performs under load. Coded UI Tests can also be used but are generally less suitable.
- Can select different combinations of users, browser types, network connection types and tests
- Checks the scalability of the application



Other Test Types

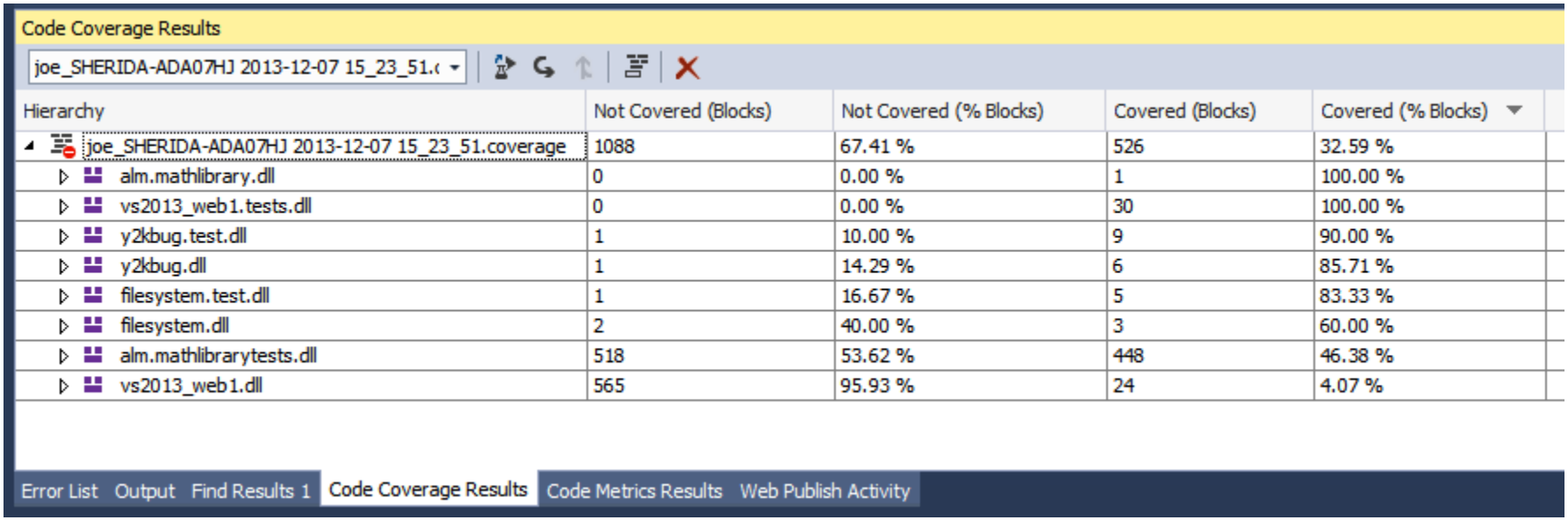
- Ordered Tests
  - Can select a subset of existing tests to run in a particular order
  - Very useful when one test depends on another test
- Generic Tests
  - Allow external programs to be invoked by Visual Studio Tests
  - Good way to test additional programs that are not part of the existing VS solution
  - Can supply command line arguments to the program under test
  - Programs under test are considered to pass if they return 0 and fail on any other return value
- Exploratory Tests
  - Allow the tester to perform "unscripted" testing of an application not covered by the standard test cases using Test Manager.
  - Can be recorded for subsequent playback

Managing and Configuring Tests

- To always run tests after build (good idea for Release builds): Test -> Test Settings -> Run Tests After Build
- Tests can be grouped into Playlists using Test Explorer or Test Manager

Code Coverage

- A key metric in testing is to determine the Code Coverage for the existing tests
- Aim for 80% or greater code coverage. If the code coverage is below this you probably need to add more tests.



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
joe_SHERIDA-ADA07HJ 2013-12-07 15_23_51.coverage	1088	67.41 %	526	32.59 %
alm.mathlibrary.dll	0	0.00 %	1	100.00 %
vs2013_web1.tests.dll	0	0.00 %	30	100.00 %
y2kbug.test.dll	1	10.00 %	9	90.00 %
y2kbug.dll	1	14.29 %	6	85.71 %
filesystem.test.dll	1	16.67 %	5	83.33 %
filesystem.dll	2	40.00 %	3	60.00 %
alm.mathlibrarytests.dll	518	53.62 %	448	46.38 %
vs2013_web1.dll	565	95.93 %	24	4.07 %

Video

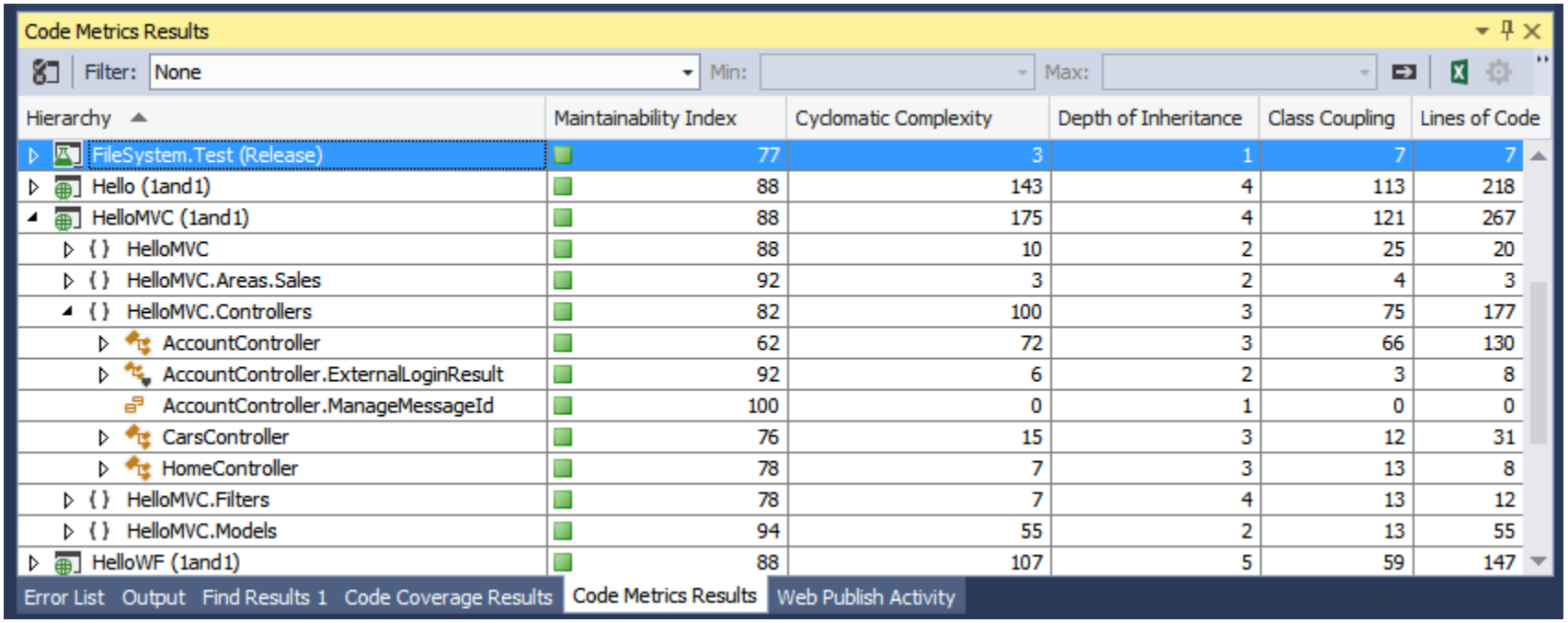
- Let's watch a great video (9.3) from the "Visual Studio 2010 Essential Training" series on Lynda

Code Analysis and Metrics

- Visual Studio provides many tools for analyzing your code in the Analyze menu including:
  - Code Metrics (for performing various measurements on your code including: Lines of Code, Cyclomatic Complexity, Inheritance Depth, Class Coupling, Maintainability Index, etc.)
  - Code Analysis (for Lint type checking of compliance with various coding standards)
  - Searching for Code Clones
  - Performance and Diagnostics / Profiler (performance data, memory allocations, timing, etc.)

Code Metrics

- Visual Studio provides several metrics to better understand the complexity of your code (Lines of Code, Cyclomatic Complexity, Inheritance Depth, Class Coupling, Maintainability Index, etc.)
- Select Analyze-> Calculate Code Metrics…



More Static Code Analysis

- Code Metrics is one of a set of Static Code Analysis Tools
- To check your code for compliance with best practices: Analyze-> Run Code Analysis…
- Code Analysis can be configured for a project in the "Code Analysis" tab of the Project's Properties
- To check for duplicate code: Analyze-> Analyze Solution for Code Clones
- Code Map provides a visual tool for depicting relationships between different entities (methods, objects, etc.) in your code. Right click an identifier in your code and select "Show on Code Map".

Dynamic Code Analysis

- The one Test Tool that all developers should get familiar with is their Debugger!
- Visual Studio has a very powerful Debugger!
- A few handy debugger tips:
  - Use the Watch Window to add new variables to watch
  - Use the Immediate Window to quickly evaluate variables, statements or methods
  - Use Conditional Breakpoints to only break when required (ex. last iteration of loop)
  - Run to Cursor to stop the code at a specific place
  - Set Next Statement to move back or skip some code
  - Show Next Statement if you have lost your place
  - Use Tracepoints to log when an instruction is executed
  - Use Debug.Assert to display failed assumptions in the Output Window
  - Use Debug.WriteLine to insert extra debugging information to the Output Window
  - Debug ASP.NET Projects using the Page Inspector as it will provide additional information over standard browsers

More Dynamic Code Analysis

- Visual Studio also provides more Dynamic Code Analysis tools to better understand how your application behaves at runtime
- To invoke the Performance Wizard to examine performance and memory usage: Analyze-> Performance and Diagnostics
- Using the Release Configuration of your application is typically more appropriate for Profiling!
- For more information on Performance Profiling see MSDN https://msdn.microsoft.com/en-us/library/ms182372.aspx

More Dynamic Code Analysis

- IntelliTrace
  - is a "time machine" that provides a history of events (exceptions, breakpoints, .NET events, system events, etc.) and can be very useful for debugging and stepping back in time.
  - is enabled by default during debugging and can also be collected from Microsoft Test Manager!
  - IntelliTrace events and features are configurable. Adding Call information can be helpful for short durations (uses lots of space)
- For more information
  - MSDN https://msdn.microsoft.com/en-us/library/dd264915.aspx
- Dump Files provide a snapshot of an application at a particular point in time
  - often useful for diagnosing crashes, exceptions, etc.
  - MSDN https://msdn.microsoft.com/en-us/library/dd264915.aspx

More Testing Tools & Techniques

- Use an error logging tool such as elmah (Error Logging Modules and Handlers for ASP.NET)
  - Consider maintaining a custom Audit Trail to track sensitive or complex requests
  - Examine the site logs which are provided by most hosts
  - Examine the Windows Event Log if you have access
  - https://code.google.com/p/elmah/

Repository Pattern

- The Repository Pattern is a very common Design Pattern used in ASP.NET MVC.
- It hides the details of the data interactions (Entity Framework, LINQ, etc.) from the MVC Controllers behind a simple repository class or interface
- It provides the following benefits:
  - Simplifies the Controller logic
  - Allows the data storage to be easily changed with no impact on the Controllers
  - Allows you to easily fake working with an underlying database
  - Makes MVC more testable
- The Unit of Work pattern is often used with the Repository pattern as it helps to group database updates.

Repository Pattern

- It can simplify a standard MVC Controller to look like this:

```
public class CountryController : Controller
{
    //private OlympicsContext db = new OlympicsContext();
    private CountryRepository repository = new CountryRepository();

    // GET: /Country/
    0 references
    public ActionResult Index()
    {
        return View(repository.Get());
    }

    // GET: /Country/Details/5
    0 references
    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Country country = repository.Details(id.Value);
```

Dependency Injection

- Dependency Injection is a common Design Pattern used to:
  - Make code less tightly coupled
  - More testable
- Normally used with the Repository Pattern to make the Repository Pattern more flexible.
- Dependency Injection is so common that many DI frameworks exist including NInject

Dependency Injection

- Normally DI works by replacing hard coded dependencies on classes by interfaces which are "injected" into the constructor at runtime.

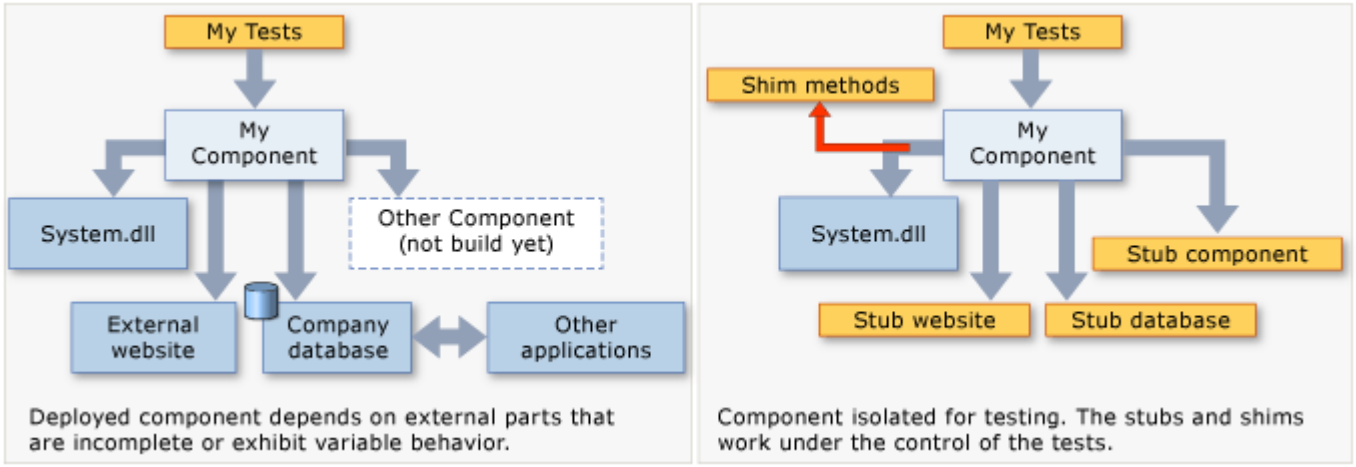
```
public class CountryController : Controller
{
    //private OlympicsContext db = new OlympicsContext();
    //private CountryRepository repository = new CountryRepository();
    private ICountryRepository repository;

    0 references
    public CountryController() : this (new CountryRepository()) {

    }

    1 reference
    public CountryController(ICountryRepository repo)
    {
        repository = repo;
    }
```

Testing Dependencies

- Testing can be complicated by external dependencies that are:
  - NOT implemented (code incomplete, etc.)
  - NOT available (server inaccessible, etc.)
  - Slow things down
  - Distract attention away from the Code Under Test
- Developers often use Dependency Injection AND write lots of extra test code to "mock" or "fake" these dependencies in order to easily complete their tests and focus on the Code Under Test (CUT)
- Consequently, many "mocking" frameworks have been developed including: Moq, Nmock and Microsoft Fakes
- Remember that when testing against a database you may need to:
  - Add Entity Framework support to the Test Project using NuGet
  - Add the Connection String for the database to the app.config file of the Test Project

Microsoft Fakes Framework

- Microsoft Fakes is built into Visual Studio and provides a mocking framework that helps us to deal with dependencies.
- Implemented using
  - Stubs – replace dependent class with test double class that implements the same interface as the "real" dependency. Great for your own code that can be modified.
  - Shims – run time interceptors which replace existing code. Great for code that can't be modified (ex. from libraries). Slower to execute.



Isolating Code Under Test with Microsoft Fakes. (n.d.). Isolating Code Under Test with Microsoft Fakes. Retrieved November 23, 2013, from http://msdn.microsoft.com/en-us/library/hh549175.aspx

Fakes - Stubs

- Rely on the "Dependency Injection" pattern
- Must design the class under test to depend on an interface that is initialized at runtime rather than a concrete class.
- Microsoft Fakes can automatically generate the stub class for us that implements the interface instead of us writing our own.
- Stub class names use a prefix of "Fakes.Stub"
- Code generated relies on Lambda expression and is a bit complex but does save time

```
stubRepository.Get = () =>
    {
        isIndexCalled = true;
        return new List<Country> {new Country(), new Country()};
    };
```

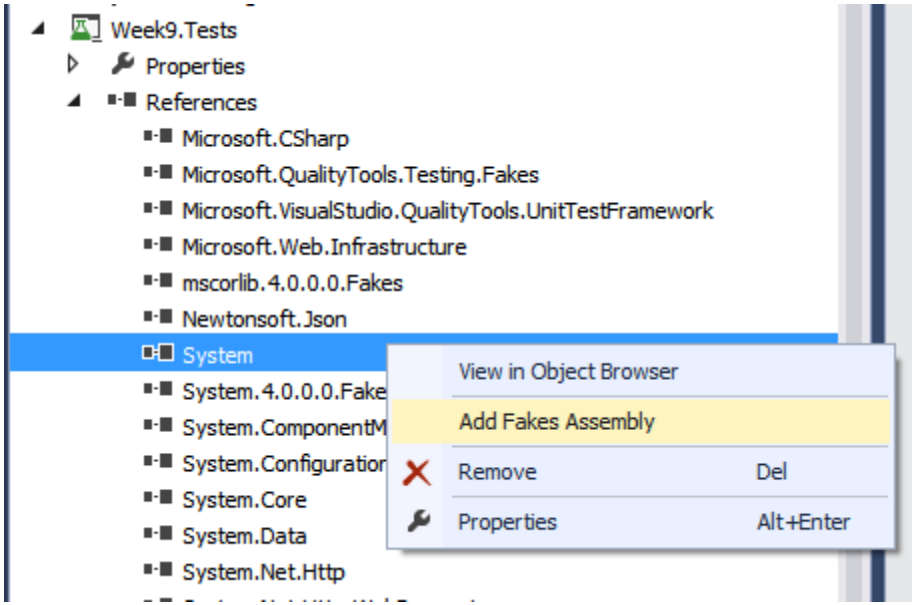
Fakes - Shims

- Run time interceptors which replace existing code
- Great for code that can't be modified (ex. from libraries)
- Slower to execute
- Shim class names use a prefix of "Fakes.Shim"
- Must create a "ShimsContext" to use the shim
- Example:
  - http://blog.nwcadence.com/shims-and-stubs-and-the-microsoft-fakes-framework/

```
ViewResult result;
using (ShimsContext.Create())
{
    System.Fakes.ShimDateTime.NowGet = () => new DateTime(2000, 1, 1);
    CountryController cc = new CountryController(stubRepository);
    result = cc.Index() as ViewResult;
}
```

Adding Support for Microsoft Fakes

- Open the Test Project
- Expand the References Node
- Right click on the assembly you would like to Fake
- Select "Add Fakes Assembly" from the popup menu



Video

- Let's watch a great video on TDD (9.2) from the "Building Applications with ASP.NET MVC 4" series on Pluralsight

Web API Testing

- Web API Testing is similar to ASP.NET MVC Testing
- A few things to keep in mind for these unit tests:
  - Most Web API methods return a type that implements IHttpActionResult
  - Methods that return the result of a call to the ApiController's OK method can have their return values easily converted back to a OkNegotiatedContentResult<T> for the content type using the "as" keyword
  - Methods that return the result of a call to the ApiController's CreatedAtRoute method can have their return values easily converted back to a CreatedAtRouteNegotiatedContentResult<T> for the content type using the "as" keyword
  - In either case, you can access the embedded type using the Content property of the result.

```
var result = sac.PostStudent(student) as CreatedAtRouteNegotiatedContentResult<Student>;
var result = sac.GetStudent(StudentId) as OkNegotiatedContentResult<Student>;
Assert.AreNotEqual(result.Content.Student_Number, 0);
```

Web API Testing

- Remember that when testing against Web API you may need to:
  - Add a Reference to the Web API Project under test
  - Add Web API Core Library support to the Test Project using NuGet