Microsoft® Official Course

Module06

Testing and Debugging ASP.NET MVC 4 Web Applications

Microsoft®

Module Overview

- Unit Testing MVC Components
  Implementing an Exception Handling Strategy

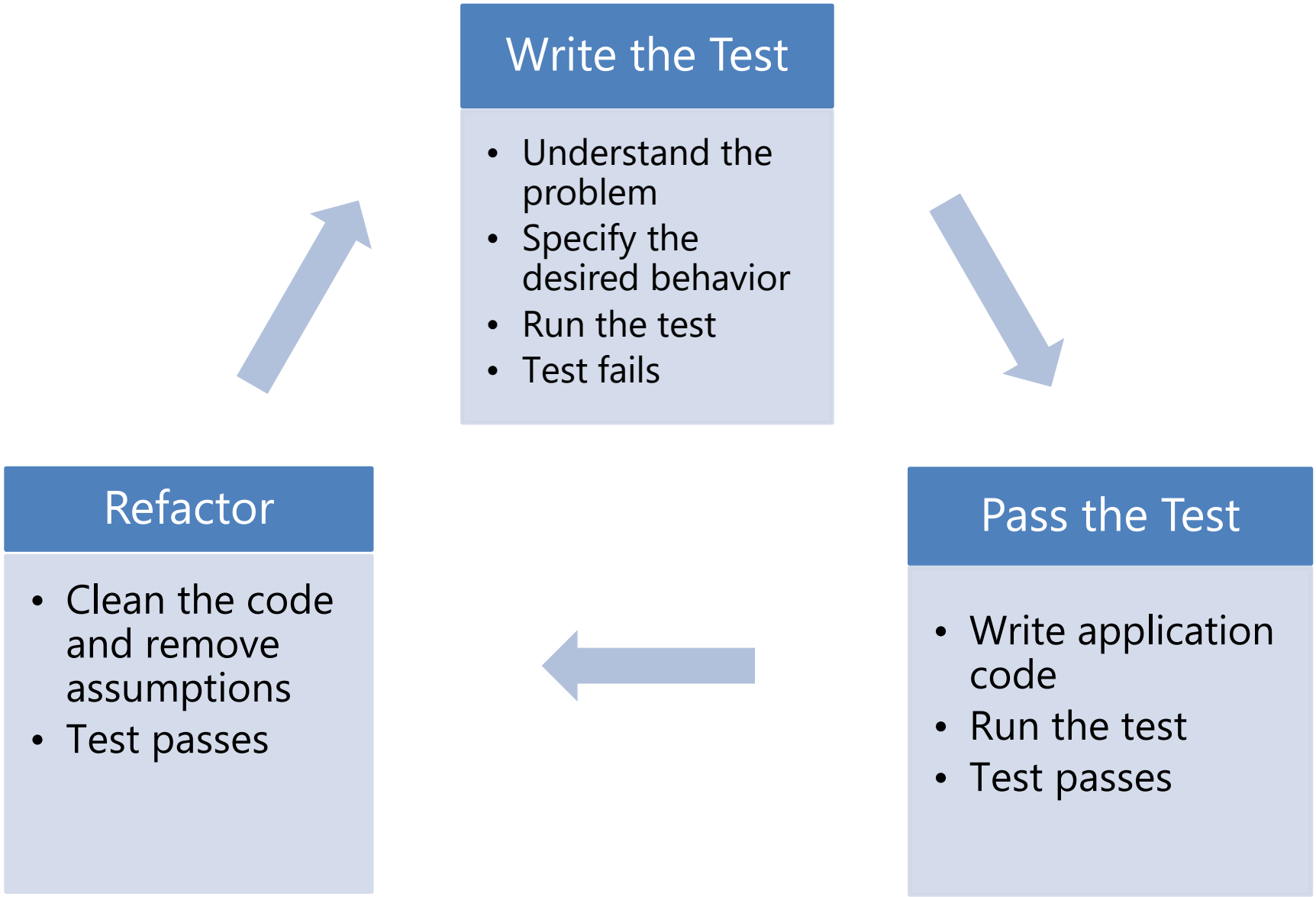
Lesson 1: Unit Testing MVC Components

- Why Perform Unit Tests?
  Principles of Test Driven Development
  Writing Loosely Coupled MVC Components
  Writing Unit Tests for MVC Components
  Specifying the Correct Context
  Demonstration: How to Run Unit Tests
  Using Mocking Frameworks

Why Perform Unit Tests?

- Types of Tests:
  - Unit Tests
  - Integration Tests
  - Acceptance Tests
- Unit tests verify that small units of functionality work as designed
  - Arrange: This phase of a unit test arranges data to run the test on
  - Act: This phase of the unit test calls the methods you want to test
  - Assert: This phase of the unit test checks that the results are as expected
- Any unit test that fails is highlighted in Visual Studio whenever you run the test or debug the application
- Once defined, unit tests run throughout development and highlight any changes that cause them to fail

Principles of Test Driven Development



Writing Loosely Coupled MVC Components

- Loose coupling means that each component in a system requires few or no internal details of the other components in the system
- A loosely-coupled application is easy to test because it is easier to replace a fully functional instance of a class with a simplified instance that is specifically designed for the test
- Loose coupling makes it easier to replace simple components with more sophisticated components

Writing Unit Tests for MVC Components

- You can test an MVC web application project by adding a new project to the solution
- Model classes can be tested by instantiating them in-memory, arranging their property values, acting on them by calling a method, and asserting that the result was as expected
- You can test a controller by:
  - Creating a repository interface
  - Implementing and using a repository in the application
  - Implementing a test double repository
  - Using a test double to test a controller

Using a Test Double in a Unit Test

```
[TestMethod]
public void Test_Index_Model_Type()
{
   var context = new FakeWebStoreContext();
   context.Products = new[] {
      new Product(),
      new Product (),
   }.AsQueryable();
   var controller = new ProductController(context);
   var result = controller.Index() as ViewResult;
   Assert.AreEqual(typeof(List<Product>),
      result.Model.GetType());
}
```

Specifying the Correct Context

To set the correct context while testing:

- Use a test double context in unit tests
- Use an Entity Framework context at other times
- Use constructors to specify the context
- Use IoC containers to specify the context

Using Constructors to Specify Repositories

```
public class ProductController : Controller
{
   private IWebStoreContext context;
   public ProductController()
   {
      context = new WebStoreContext();
   }
   public ProductController(IWebStoreContext Context)
   {
      context = Context;
   }
   //Add action methods here
}
```

Demonstration: How to Run Unit Tests

In this demonstration, you will see how to:

1. Add a new test project, OperasWebSiteTests, to an existing MVC web application solution
2. Create code for a simple unit test
3. Observe the results of a failed test
4. Observe the results of a passed test

Using Mocking Frameworks

- A mocking framework automates the creation of mock objects during tests
  - You can automate the creation of a single object
  - You can automate the creation of multiple objects of the same type
  - You can automate the creation of multiple objects that implement different interfaces
- The mocking framework saves time when writing unit tests

Lesson 2: Implementing an Exception Handling Strategy

- Raising and Catching Exceptions

  Configuring Exception Handling

  Using Visual Studio IntelliTrace in MVC

  Logging Exceptions

  Health Monitoring

Raising and Catching Exceptions

- The most common method to catch an exception is to use the **try/catch** block
- You can also override the OnException method
- You can also catch exceptions by using the **[HandleError]** annotation

```
[HandleError(ExceptionType=
   typeof(NotImplementedException),
   View="NotImplemented")]

[HandleError]

public ActionResult Index()

{

   //Place action code here

}
```

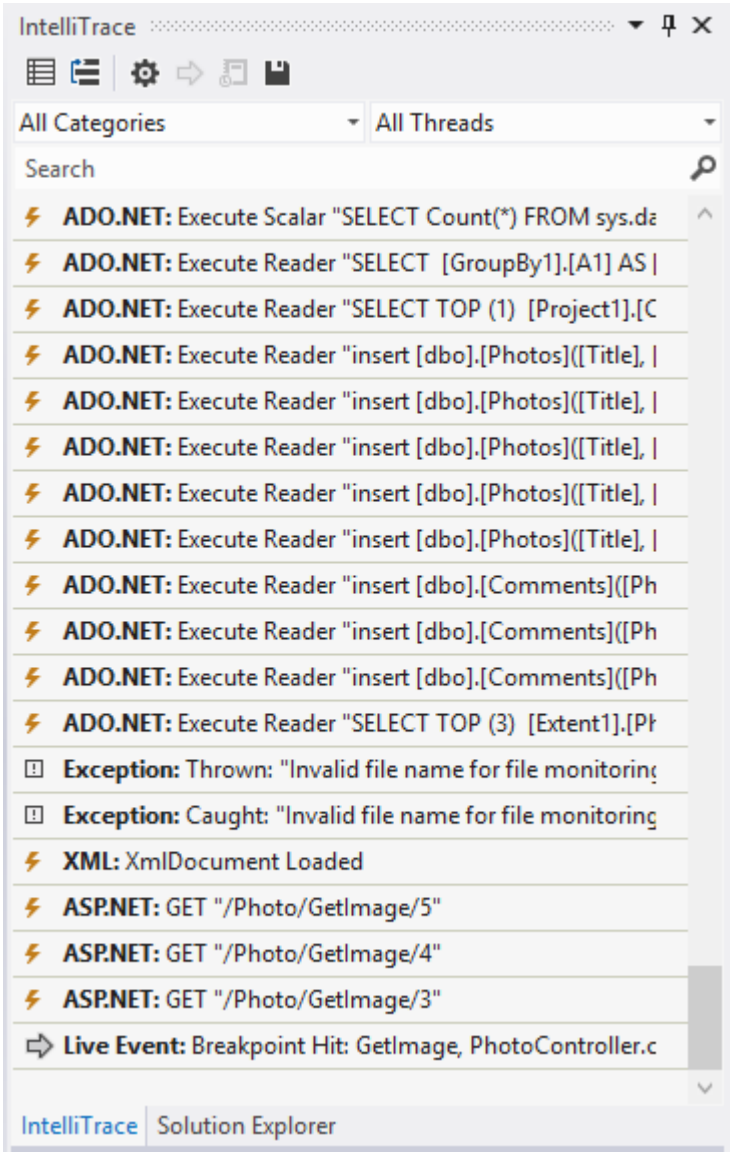
Configuring Exception Handling

- You can configure custom error messages by:
  - Configuring custom errors in Web.config
  - Using the <customError> element to specify a custom view for unhandled errors
  - Using the <error> element to handle HTTP error codes

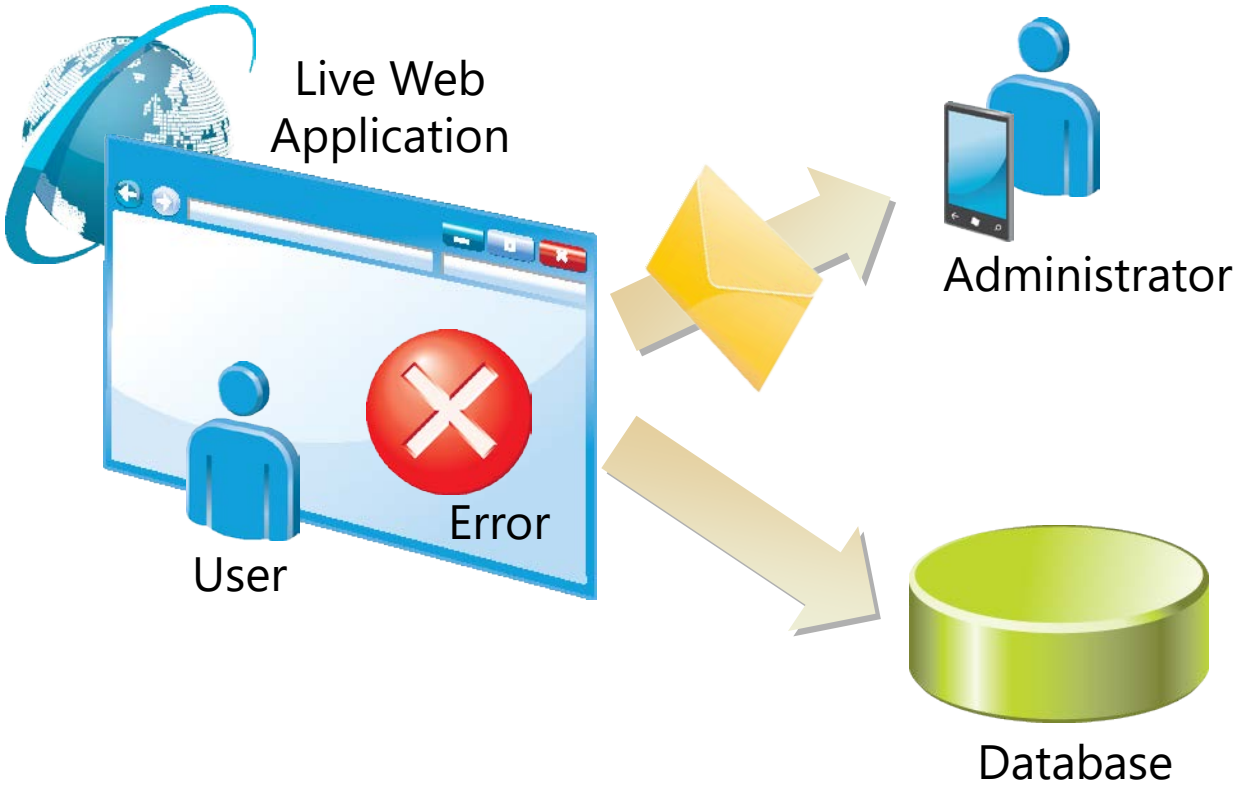
```
<customErrors mode="On"
defaultRedirect="CustomError" />
```

Using Visual Studio IntelliTrace in MVC

- You can use IntelliTrace to:
  - Improve the debugging workflow by recording a timeline of code execution
  - Review events that happened *before* the current execution point
  - Save events in an IntelliTrace file when you enter debug mode



Logging Exceptions



When an exception occurs, the application sends an email message to the administrators, and logs full details of the exception to a database.

Health Monitoring

- You can use the following health monitoring features to monitor your web application:
  - Event Categories help you customize the category of events that health monitoring records
  - Health Providers customize the location where health monitoring stores event details
- To configure health monitoring, you can use the **<healthMonitoring>** element in Web.config

Lab: Testing and Debugging ASP.NET MVC 4 Web Applications

- Exercise 1: Performing Unit Tests

  Exercise 2: Optional—Configuring Exception Handling

Logon Information

Virtual Machine: **20486B-SEA-DEV11**
User name: **Admin**
Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the **MSL-TMG1** virtual machine if it is not already running.

Estimated Time: 90 minutes

Lab Scenario

The Photo Sharing application is in the early stages of development. However, frequent errors are hindering the productivity of the development team. The senior developer advises that you intercept exceptions and other flaws as early as possible. You have been asked to perform unit tests of the PhotoController to ensure that all scenarios work as expected and to avoid problems later in the web application development life cycle. You have also been asked to ensure that when critical errors occur, developers can obtain helpful technical information.

Lab Review

- When you ran the tests for the first time in Exercise 1, why did Test_Index_Return_View pass, while Test_GetImage_Return_Type and Test_PhotoGallery_Model_Type failed?

  In Exercise 1, why did all the tests pass during the second run?

Module Review and Takeaways

- Review Question(s)

  Real-world Issues and Scenarios

  Tools

  Best Practice

  Common Issues and Troubleshooting Tips