

Stephen Walther

ASP.NET MVC Framework

UNLEASHED



SAMS

ASP.NET MVC Framework Unleashed

Copyright © 2010 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-32998-2

ISBN-10: 0-672-32998-0

Library of Congress Cataloging-in-Publication data

Walther, Stephen.

ASP.NET MVP framework unleashed / Stephen Walther.

p. cm.

ISBN 978-0-672-32998-2

1. Active server pages. 2. Microsoft .NET Framework. 3. Web site development. I. Title.

TK5105.8885.A26W3522 2010

006.7'882-dc22

2009021084

Printed in the United States of America

First Printing July 2009

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

international@pearson.com

Editor-in-Chief

Karen Gettman

Executive Editor

Neil Rowe

Development Editor

Mark Renfrow

Managing Editor

Kristy Hart

Project Editor

Betsy Harris

Copy Editor

Apostrophe Editing
Services

Indexer

Erika Millen

Proofreader

Keith Cline

Technical Editor

Rebecca Riordan

Publishing

Coordinator

Cindy Teeters

Book Designer

Gary Adair

Compositor

Jake McFarland

Introduction

ASP.NET MVC is Microsoft's newest technology for building web applications. Although ASP.NET MVC is new, there are already several large and successful websites that are built on the ASP.NET MVC framework including StackOverflow.com and parts of CodePlex.com.

ASP.NET MVC was created to appeal to several different audiences. If you are the type of developer who wants total control over every HTML tag and pixel that appears in a web page, the ASP.NET MVC framework will appeal to you.

ASP.NET MVC also enables you to expose intuitive URLs to the world. Exposing intuitive URLs is important for getting your website indexed by search engines. If you care about Search Engine Optimization, you will be happy with ASP.NET MVC.

The ASP.NET MVC framework enables you to build web applications that are easier to maintain and extend over time. The Model View Controller pattern encourages a clear separation of concerns. The framework encourages good software design patterns.

Finally, the ASP.NET MVC framework was designed from the ground up to support testability. In particular, the ASP.NET MVC framework enables you to practice test-driven development. You are not required to practice test-driven development when building an ASP.NET MVC application, but the ASP.NET MVC framework makes test-driven development possible.

How This Book Is Organized

The book is divided into two parts. The first part of the book describes the ASP.NET MVC framework feature-by-feature. For example, there are chapters devoted to the subject of controllers, caching, and validation.

The second part of this book contains a walkthrough of building a full ASP.NET MVC application: We build a simple blog application. We implement features such as data access and validation.

Because one of the primary benefits of the ASP.NET MVC framework is that it enables test-driven development, we build the blog application by using test-driven development. The blog application illustrates how you can overcome many challenges that you face when writing real-world applications with the ASP.NET MVC framework.

You can approach this book in two ways. Some readers might want to read through the first chapters of this book before reading the chapters on building the blog application. Other readers might want to read the walkthrough of building the blog application before reading anything else.

What You Should Know Before Reading This Book

I make few assumptions about your technical background. I assume that you know either the C# or the Visual Basic .NET programming language—all the code samples are included in both languages in the body of the book. I also assume that you know basic HTML.

ASP.NET MVC uses many advanced features of the C# and Visual Basic .NET language. The first appendix of this book, Appendix A, “C# and VB.NET Language Features,” contains an overview of these new features. For example, if you are not familiar with anonymous types or LINQ to SQL, you should take a look at Appendix A.

The other two appendixes, Appendix B, “Using a Unit Testing Framework,” and Appendix C, “Using a Mock Object Framework,” are devoted to explaining how to use the main tools of test-driven development. In Appendix B, you learn how to use both the Visual Studio Unit Test framework and how to use the NUnit Unit Test framework. Appendix C is devoted to Mock Object Frameworks.

Throughout the book, when a line of code is too long for the printed page, a code-continuation arrow (➞) has been used to mark the continuation. For example:

```
ReallyLongClassName.ReallyLongMethodName("Here is a value",  
➞ "Here is another value")
```

What Software Do You Need?

You can download all the software that you need to build ASP.NET MVC applications by visiting the www.ASP.net/mvc website. You need to install three software components:

1. **Microsoft .NET Framework 3.5 Service Pack 1**—The Microsoft .NET framework includes the Microsoft ASP.NET framework.
2. **Microsoft ASP.NET MVC 1.0**—The actual ASP.NET MVC framework that runs on top of the ASP.NET framework.
3. **Microsoft Visual Web Developer 2008 Service Pack 1 or Microsoft Visual Studio 2008 Service Pack 1**—The development environment for creating ASP.NET applications. Also includes the option of installing Microsoft SQL Server Express.

The Microsoft .NET framework, Microsoft ASP.NET MVC, and Microsoft Visual Web Developer are all free. You can build ASP.NET MVC applications without paying a single cent.

Instead of downloading and installing each of these software components one-by-one, you can take advantage of the Microsoft Web Platform Installer to manage the download and installation of all these components. You can launch the Microsoft Web Platform Installer from the www.ASP.net/mvc site.

Where Do You Download the Code Samples?

The code samples for the book are located on the book's product page, www.informit.com/title/9780672329982.

If You Like This Book

After you read this book, if you discover that this book helped you to understand and build ASP.NET MVC applications, please post a review of this book at the www.Amazon.com website.

To get the latest information on ASP.NET MVC, I encourage you to visit the official Microsoft ASP.NET MVC website at www.ASP.net/mvc. I also encourage you to subscribe to my blog at StephenWalther.com that contains ASP.NET MVC tips and tutorials. I also use my blog to post any errata that is discovered after the book is published.

CHAPTER 1

An Introduction to ASP.NET MVC

"There is nothing permanent except change."

Heraclitus

This chapter provides you with an overview and introduction to the Microsoft ASP.NET MVC framework. The goal of this chapter is to explain why you should build web applications using ASP.NET MVC.

Because the ASP.NET MVC framework was designed to enable you to write good software applications, the first part of this chapter is devoted to a discussion of the nature of good software. You learn about the software design principles and patterns that enable you to build software that is resilient to change.

Finally, we discuss the architecture of an ASP.NET MVC application and how this architecture enables you to write good software applications. We provide you with an overview of the different parts of an MVC application including models, views, and controllers and also introduce you to the sample application that you get when you create a new ASP.NET MVC project.

A Story with a Moral

I still remember the day that my manager came to my office and asked me to build the *Single Button Application*. He explained that he needed a simple call manager application to help interviewers dial phone numbers while conducting a health survey. The call manager application would load a list of phone numbers and dial each number one-by-one when you hit a button. What could be simpler?

IN THIS CHAPTER

- ▶ A Story with a Moral
- ▶ What Is Good Software?
- ▶ What Is ASP.NET MVC?
- ▶ The Architecture of an ASP.NET MVC Application
- ▶ Understanding the Sample ASP.NET MVC Application

I said, with great earnestness and confidence, that I would have the call manager application done that same afternoon. I closed my office door, put on my cowboy hat, turned up the music, and pounded out some code. By the end of the day, I had completed the application. My manager was happy, and I went home that night with the happy thought that I had done a good day of work.

The next morning, my manager appeared again at my office door. Worried, I asked if there was a problem with the call manager application. He reassured me that the application worked fine. In fact, he liked it so much that he wanted me to add another feature. He wanted the call manager application to display a survey form when a number is dialed. That way, survey answers could be stored in the database.

With heroic determination, I once again spent the day knocking out code. By the end of the day, I had finished updating the call manager and I proudly presented the finished application to my manager.

I won't continue this story, because anyone who builds software for a living knows how this story ends. The story never ends. When a software project is brought to life, it is almost impossible to kill it. A software application needs to be continuously fed with new features, bug fixes, and performance enhancements.

Being asked to change software that you have created is a compliment. Only useless software goes stagnant. When people care about software, when software is actively used, it undergoes constant change.

I no longer work at the company where I created the call manager application. (I am currently sitting in an office at Microsoft.) But I still have friends at the company and every once in a while I get a report on how the application has changed. Needless to say, it has turned into a massively complex application that supports different time zones, complicated calling rules, and advanced reporting with charts. It can no longer be described as the Single Button Application.

What Is Good Software?

I dropped out of graduate school at MIT to launch an Internet startup in the earliest days of the Web. At that time, building a website was difficult. This was before technologies such as Active Server Pages or ASP.NET existed. (We had only stone knives.) Saving the contents of an HTML form to a database table was a major accomplishment. Blinking text was the height of cool.

When I first started writing software, simply getting the software to do what I wanted was the goal. Adding as many features to a website in the shortest amount of time was the key to survival in the ferociously competitive startup world of the '90s. I used to sleep in my office under my desk.

During my startup phase, I would define good software like this:

Good software is software that works as you intended.

If I was feeling particularly ambitious, I would worry about performance. And maybe, just maybe, if I had extra time, I would add a comment or two to my code. But really, at the end of the day, my criterion for success was simply that the software worked.

For the past 8 years, I've provided training and consulting to large companies and organizations such as Boeing, NASA, Lockheed Martin, and the National Science Foundation. Large organizations are not startups. In a large organization, the focus is not on building software applications as fast as possible; the focus is on building software applications that can be easily maintained over time.

Over the years, my definition of good software has shifted substantially. As I have been faced with the scary prospect of maintaining my own monsters, I've changed my definition of good software to this:

Good software is software that works as you intended and that is easy to change.

There are many reasons that software changes over time. Michael Feathers, in his excellent book *Working Effectively with Legacy Code*, offers the following reasons:

1. You might need to add a new feature to existing software.
2. You might need to fix a bug in existing software.
3. You might need to optimize existing software.
4. You might need to improve the design of existing software.

For example, you might need to add a new feature to an application. The call manager application started as a Single Button Application. However, each day, more and more features were added to the application.

You also need to change software when you discover a bug in the software. For instance, in the case of the call manager, we discovered that it did not calculate daylight savings time correctly. (It was waking some people up in the morning!) We rushed to change the broken code.

You also might need to modify a software application to make the application run faster. At one point, the call manager application took as long as 12 seconds to dial a new phone number. The business rules were getting complex. We had to rewrite the code to get the phone number retrieval time down to the millisecond range.

Finally, you might need to modify software to improve its design. In other words, you might need to take badly written code and convert it into good code. You might need to make your code more resilient to change.

Avoiding Code Smells

Unless you are careful, a software application quickly becomes difficult to change. We all have had the experience of inheriting an application that someone else has written and being asked to modify it. Think of the fear that strikes your heart just before you make your first change.

In the game of Pick-Up Sticks, you must remove stick after stick from a pile of sticks without disturbing the other sticks. The slightest mistake and the whole pile of sticks might scatter.

Modifying an existing software application is similar to the game of Pick-Up Sticks. You bump the wrong piece of code and you introduce a bug.

Bad software is software that is difficult to change. Robert and Micah Martin describe the markers of bad software as *code smells*. The following code smells indicate that software is badly written:

- ▶ **Rigidity**—Rigid software is software that requires a cascade of changes when you make a change in one place.
- ▶ **Fragility**—Fragile software is software that breaks in multiple places when you make a change.
- ▶ **Needless complexity**—Needlessly complex software is software that is overdesigned to handle any possible change.
- ▶ **Needless repetition**—Needlessly repetitious software contains duplicate code.
- ▶ **Opacity**—Opaque software is difficult to understand.

NOTE

These code smells are described by Micah and Robert Martin in their book *Agile Principles, Patterns, and Practices in C#* on page 104. This book is strongly recommended!

Notice that these code smells are all related to change. Each of these code smells is a barrier to change.

Software Design Principles

Software does not need to be badly written. A software application can be designed from the beginning to survive change.

The best strategy for making software easy to change is to make the components of the application *loosely coupled*. In a loosely coupled application, you can make a change to one component of an application without making changes to other parts.

Over the years, several principles have emerged for writing good software. These principles enable you to reduce the dependencies between different parts of an application. These software principles have been collected together in the work of Robert Martin (AKA Uncle Bob).

Robert Martin did not invent all the principles; however, he was the first one to gather the principles into a single list. Here is his list of software design principles:

- ▶ **SRP**—Single Responsibility Principle
- ▶ **OCP**—Open Closed Principle

- ▶ **LSP**—Liskov Substitution Principle
- ▶ **ISP**—Interface Segregation Principle
- ▶ **DIP**—Dependency Inversion Principle

This collection of principles is collectively known by the acronym SOLID. (Yes, SOLID is an acronym of acronyms.)

For example, according to the Single Responsibility Principle, a class should have one, and only one, reason to change. Here's a concrete example of how this principle is applied: If you know that you might need to modify your application's validation logic separately from its data access logic, then you should not mix validation and data access logic in the same class.

NOTE

There are other lists of software design principles. For example, the *Head First Design Patterns* book has a nice list. You should also visit the C2.com website.

Software Design Patterns

Software design patterns represent strategies for applying software design principles. In other words, a software design principle is a good idea and a software design pattern is the tool that you use to implement the good idea. (It's the hammer.)

The idea behind software design patterns was originally promoted by the book *Design Patterns: Elements of Reusable Object-Oriented Software*. (This book is known as the Gang of Four book.) This book has inspired many other books that describe software design patterns.

The *Head First Design Pattern* book provides a more user-friendly introduction to the design patterns from the Gang of Four book. The *Head First Design* book devotes chapters to 14 patterns with names like Observer, Façade, Singleton, and Adaptor.

Another influential book on software design patterns is Martin Fowler's book *Patterns of Enterprise Application Architecture*. This book has a companion website that lists the patterns from the book: www.martinfowler.com/eaCatalog.

Software design patterns provide you with patterns for making your code more resilient to change. For example, in many places in this book, we take advantage of a software design pattern named the Repository pattern. Eric Evans, in his book *Domain-Driven Design*, describes the Repository pattern like this:

"A REPOSITORY represents all objects of a certain type as a conceptual set (usually emulated). It acts like a collection, except with more elaborate querying capability. Objects of the appropriate type are added and removed, and the machinery behind the REPOSITORY inserts them or deletes them from the database" (see page 151).

According to Evans, one of the major benefits of the Repository pattern is that it enables you to "decouple application and domain design from persistence technology, multiple

database strategies, or even multiple data sources.” In other words, the Repository pattern enables you to shield your application from changes in how you perform database access.

For example, when we write our blog application at the end of this book, we take advantage of the Repository pattern to isolate our blog application from a particular persistence technology. The blog application will be designed in such a way that we could switch between different data access technologies such as LINQ to SQL, the Entity Framework, or even NHibernate.

Writing Unit Tests for Your Code

By taking advantage of software design principles and patterns, you can build software that is more resilient to change. Software design patterns are architectural patterns. They focus on the gross architecture of your application.

If you want to make your applications more change proof on a more granular level, then you can build unit tests for your application. A unit test enables you to verify whether a particular method in your application works as you intend it to work.

There are many benefits that result from writing unit tests for your code:

1. Building tests for your code provides you with a safety net for change.
2. Building tests for your code forces you to write loosely coupled code.
3. Building tests for your code forces you to take a user perspective on the code.

First, unit tests provide you with a safety net for change. This is a point that Michael Feathers emphasizes again and again in his book *Working Effectively with Legacy Code*. In fact, he defines legacy code as “simply code without tests” (see xvi).

When your application code is covered by unit tests, you can modify the code without the fear that the modifications will break the functionality of your code. Unit tests make your code safe to refactor. If you can refactor, then you can modify your code using software design patterns and thus produce better code that is more resilient to change.

NOTE

Refactoring is the process of modifying code without changing the functionality of the code.

Second, writing unit tests for your code forces you to write code in a particular way. Testable code tends to be loosely coupled code. A unit test performs a test on a unit of code in isolation. To build your application so that it is testable, you need to build the application in such a way that it has isolatable components.

One class is loosely coupled to a second class when you can change the first class without changing the second class. Test-driven development often forces you to write loosely coupled code. Loosely coupled code is resistant to change.

Finally, writing unit tests forces you to take a user’s perspective on the code. When writing a unit test, you take on the same perspective as a developer who will use your code in the

future. Because writing tests forces you to think about how a developer (perhaps, your future self) will use your code, the code tends to be better designed.

Test-Driven Development

In the previous section, we discussed the importance of building unit tests for your code. Test-driven development is a software design methodology that makes unit tests central to the process of writing software applications. When you practice test-driven development, you write tests first and then write code against the tests.

More precisely, when practicing test-driven development, you complete three steps when creating code (Red/Green/Refactor):

1. Write a unit test that fails (Red).
2. Write code that passes the unit test (Green).
3. Refactor your code (Refactor).

First, you write the unit test. The unit test should express your intention for how you expect your code to behave. When you first create the unit test, the unit test should fail. The test should fail because you have not yet written any application code that satisfies the test.

Next, you write just enough code for the unit test to pass. The goal is to write the code in the laziest, sloppiest, and fastest possible way. You should not waste time thinking about the architecture of your application. Instead, you should focus on writing the minimal amount of code necessary to satisfy the intention expressed by the unit test.

Finally, after you write enough code, you can step back and consider the overall architecture of your application. In this step, you rewrite (refactor) your code by taking advantage of software design patterns—such as the Repository pattern—so that your code is more maintainable. You can fearlessly rewrite your code in this step because your code is covered by unit tests.

There are many benefits that result from practicing test-driven development. First, test-driven development forces you to focus on code that actually needs to be written. Because you are constantly focused on just writing enough code to pass a particular test, you are prevented from wandering into the weeds and writing massive amounts of code that you will never use.

Second, a “test first” design methodology forces you to write code from the perspective of how your code will be used. In other words, when practicing test-driven development, you constantly write your tests from a user perspective. Therefore, test-driven development can result in cleaner and more understandable APIs.

Finally, test-driven development forces you to write unit tests as part of the normal process of writing an application. As a project deadline approaches, testing is typically the first thing that goes out the window. When practicing test-driven development, on the other hand, you are more likely to be virtuous about writing unit tests because test-driven development makes unit tests central to the process of building an application.

Short-Term Pain, Long-Term Gain

Building software designed for change requires more upfront effort. Implementing software design principles and patterns takes thought and effort. Writing tests takes time. However, the idea is that the initial effort required to build software the right way will pay huge dividends in the future.

There are two ways to be a developer. You can be a cowboy or you can be a craftsman. A cowboy jumps right in and starts coding. A cowboy can build a software application quickly. The problem with being a cowboy is that software must be maintained over time.

A craftsman is patient. A craftsman builds software carefully by hand. A craftsman is careful to build unit tests that cover all the code in an application. It takes longer for a craftsman to create an application. However, after the application is created, it is easier to fix bugs in the application and add new features to the application.

Most software developers start their programming careers as cowboys. At some point, however, you must hang up your saddle and start building software that can stand the test of time.

What Is ASP.NET MVC?

The Microsoft ASP.NET MVC framework is Microsoft's newest framework for building web applications. The ASP.NET MVC framework was designed from the ground up to make it easier to build good software in the sense of good software discussed in this chapter.

The ASP.NET MVC framework was created to support *pattern-based* software development. In other words, the framework was designed to make it easier to implement software design principles and patterns when building web applications.

Furthermore, the ASP.NET MVC framework was designed to its core to support unit tests. Web applications written with the ASP.NET MVC framework are highly testable.

Because ASP.NET MVC applications are highly testable, this makes the ASP.NET MVC framework a great framework to use when practicing test-driven development.

ASP.NET MVC Is Part of the ASP.NET Framework

Microsoft's framework for building software applications—any type of application including desktop, web, and console applications—is called the *.NET framework*. The .NET framework consists of a vast set of classes, tens of thousands of classes, which you can use when building any type of software application. For example, the .NET framework includes classes for working with the file system, accessing a database, using regular expressions, and generating images.

The ASP.NET framework is one part of the .NET framework. The ASP.NET framework is Microsoft's framework for building web applications. It contains a set of classes that were created specifically to support building web applications. For example, the ASP.NET framework includes classes for implementing web page caching, authentication, and authorization.

Microsoft has two frameworks for building web applications built on top of the ASP.NET framework: ASP.NET Web Forms and ASP.NET MVC (see Figure 1.1).

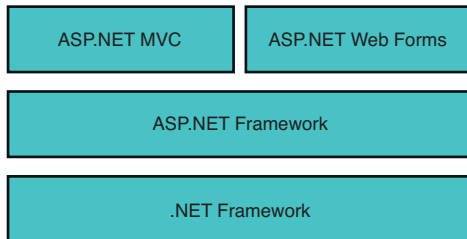


FIGURE 1.1 The ASP.NET frameworks

ASP.NET MVC is an alternative to, but not a replacement for, ASP.NET Web Forms. Some developers find the style of programming represented by ASP.NET Web Forms more compelling, and some developers find ASP.NET MVC more compelling. Microsoft continues to make heavy investments in both technologies.

NOTE

This book is devoted to the topic of ASP.NET MVC. If you want to learn about ASP.NET Web Forms, buy my book *ASP.NET Unleashed*.

The Origins of MVC

The ASP.NET MVC framework is new; however, the MVC software design pattern itself has a long history. The MVC pattern was invented by Trygve Reenskaug while he was a visiting scientist at the Smalltalk group at the famed Xerox Palo Alto Research Center. He wrote his first paper on MVC in 1978. He originally called it the Thing Model View Editor pattern, but he quickly changed the name of the pattern to the Model View Controller pattern.

NOTE

Trygve Reenskaug, the inventor of the MVC pattern, currently works as a professor of informatics at the University of Oslo in Norway.

The MVC pattern was first implemented as part of the Smalltalk-80 class library. It was originally used as an architectural pattern for creating graphical user interfaces (GUIs).

The meaning of MVC shifted radically when the pattern was adapted to work with web applications. In the context of web applications, the MVC pattern is sometimes referred to as the Model 2 MVC pattern.

The MVC pattern has proven to be very successful. Today, the MVC pattern is used by several popular web application frameworks including Ruby on Rails, Merb, and Django. The MVC pattern is also popular in the Java world. In the Java world, MVC is used in the Struts, Spring, and Tapestry frameworks.

The first major MVC framework for ASP.NET was the open source MonoRail project (see CastleProject.org). There continues to be an active developer community around this project.

The Microsoft ASP.NET MVC framework was originally created by Scott Guthrie on an airplane trip to Austin, Texas, to speak at the first Alt.NET conference in October 2007. (Scott Guthrie was one of the creators of ASP.NET.) Scott Guthrie's talk generated so much excitement that the ASP.NET MVC framework became an official Microsoft product. ASP.NET MVC 1.0 was released in the first part of 2009.

The Architecture of an ASP.NET MVC Application

An MVC application, a Model View Controller application, is divided into the following three parts:

- ▶ **Model**—An MVC model contains all of an application's logic that is not contained in a view or controller. The model includes all of an application's validation logic, business logic, and data access logic. The MVC model contains model classes that model objects in the application's domain.
- ▶ **View**—An MVC view contains HTML markup and view logic.
- ▶ **Controller**—An MVC controller contains control-flow logic. An MVC controller interacts with MVC models and views to control the flow of application execution.

Enforcing this separation of concerns among models, views, and controllers has proven to be a useful way of structuring a web application.

First, sharply separating views from the remainder of a web application enables you to redesign the appearance of your application without touching any of the core logic. A web page designer (the person who wears the black beret) can modify the views independently of the software engineers who build the business and data access logic. People with different skills and roles can modify different parts of the application without stepping on each other's toes.

Furthermore, separating the views from the remainder of your application logic enables you to easily change the view technology in the future. One fine day, you might decide to re-implement the views in your application using Silverlight instead of HTML. If you entangle your view logic with the rest of your application logic, migrating to a new view technology will be difficult.

Separating controller logic from the remainder of your application logic has also proven to be a useful pattern for building web applications. You often need to modify the way that a user interacts with your application. You don't want to touch your view logic or model logic when modifying the flow of execution of your application.

Understanding the Sample ASP.NET MVC Application

A good way to get a firmer grasp on the three logical parts of an MVC application is to take a look at the sample application that is created automatically when you create a new ASP.NET MVC project with Visual Studio.

NOTE

We discuss installing ASP.NET MVC in the Introduction.

Follow these steps:

1. Launch Visual Studio.
2. Select the menu option File, New Project.
3. In the New Project dialog, select your favorite programming language (C# or VB.NET) and select the ASP.NET MVC Web Application template. Give your project the name `MyFirstMvcApp` and click the OK button (see Figure 1.2).

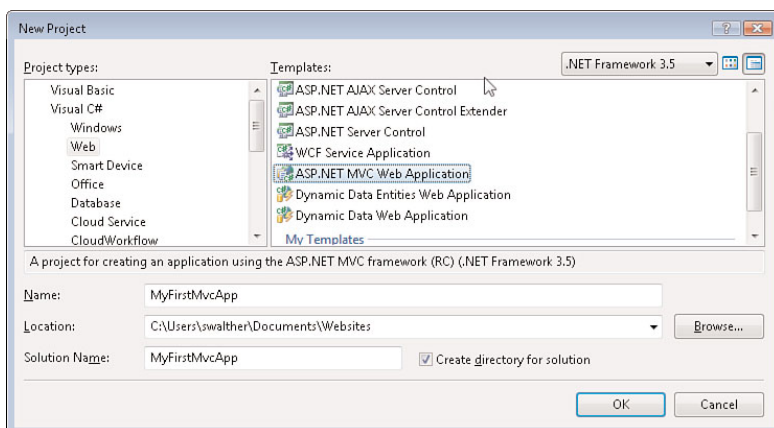


FIGURE 1.2 Creating a new ASP.NET MVC project

Immediately after you click the OK button to create a new ASP.NET MVC project, you see the Create Unit Test Project dialog in Figure 1.3. Leave the default option selected—**Yes, Create a Unit Test Project**—and click the OK button.

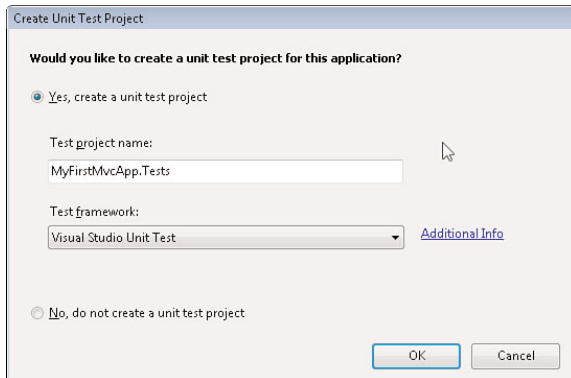


FIGURE 1.3 Creating a unit test project

Your computer hard drive will churn for a few seconds while Visual Studio creates the default files for a new ASP.NET MVC project. After all the files are created, the Solution Explorer window should contain the files in Figure 1.4.

The Solution Explorer window in Figure 1.4 contains two separate projects: the ASP.NET MVC project and the Test project. The Test project contains all the unit tests for your application.

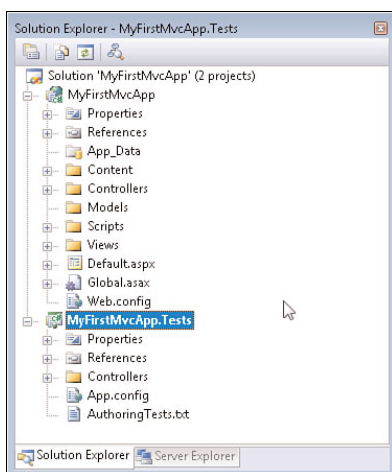


FIGURE 1.4 Files in a new ASP.NET MVC project

ASP.NET MVC Folder Conventions

The ASP.NET MVC framework emphasizes convention over configuration. There are standard locations for each type of file in an ASP.NET MVC project. The ASP.NET MVC application project contains the following folders:

- ▶ **App_Data**—Contains database files. For example, the App_Data folder might contain a local instance of a SQL Server Express database.
- ▶ **Content**—Contains static content such as images and Cascading Style Sheet files.
- ▶ **Controllers**—Contains ASP.NET MVC controller classes.
- ▶ **Models**—Contains ASP.NET MVC model classes.
- ▶ **Scripts**—Contains JavaScript files including the ASP.NET AJAX Library and jQuery.
- ▶ **Views**—Contains ASP.NET MVC views.

When building an ASP.NET MVC application, you should place controllers only in the Controllers folder, JavaScript scripts only in the Scripts folder, ASP.NET MVC views only in the Views folder, and so on. By following these conventions, your application is more easily maintained, and it can be more easily understood by others.

Running the Sample ASP.NET MVC Application

When you create a new ASP.NET MVC application, you get a simple sample application. You can run this sample application by selecting the menu option Debug, Start Debugging (or press the F5 key).

NOTE

When running an ASP.NET MVC application, make sure that the ASP.NET MVC project and not the Test project is selected in the Solution Explorer window.

The first time that you run a new ASP.NET MVC application in Visual Studio, you receive a dialog asking if you want to enable debugging. Simply click the OK button.

When you run the application, your browser opens with the page in Figure 1.5.

You can use the tabs that appear at the top of the page to navigate to either the Home or the About page. You also can click the Login link to register or log in to the application. And, that is all you can do with the application.

This sample application is implemented with one ASP.NET MVC controller and two ASP.NET MVC views. The sample application does not contain any business or data access logic, so it does not contain any ASP.NET MVC model classes.

The controller is located in the Controllers folder:

(C#)

\Controllers\HomeController.cs

(VB)

\Controllers\HomeController.vb

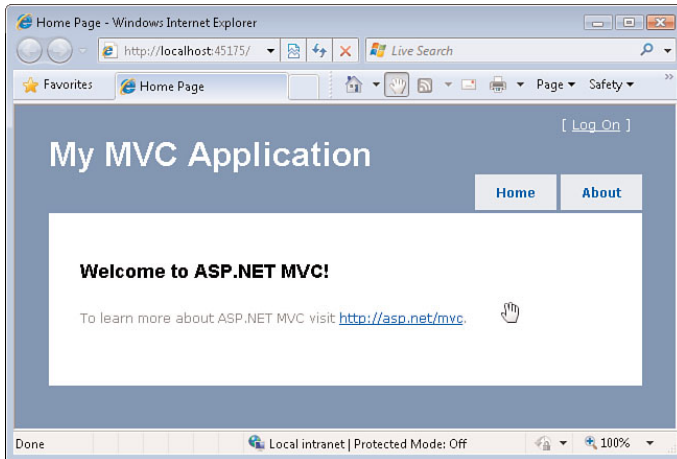


FIGURE 1.5 The sample application

If you open the HomeController in the Code Editor window, you see the file in Listing 1.1.

LISTING 1.1 Controllers\HomeController.cs (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MyFirstMvcApp.Controllers
{
    [HandleError]
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewData["Message"] = "Welcome to ASP.NET MVC!";

            return View();
        }

        public ActionResult About()
        {
            return View();
        }
    }
}
```

```
    }  
}
```

LISTING 1.1 Controllers\HomeController.vb (VB)

```
<HandleError()> _  
Public Class HomeController  
    Inherits System.Web.Mvc.Controller  
  
    Function Index() As ActionResult  
        ViewData("Message") = "Welcome to ASP.NET MVC!"  
  
        Return View()  
    End Function  
  
    Function About() As ActionResult  
        Return View()  
    End Function  
End Class
```

The file in Listing 1.1 contains a class with two methods named `Index()` and `About()`. Methods exposed by a controller are called actions. Both the `Index()` and `About()` actions return a view.

When you first run the sample application, the `Index()` action is invoked and this action returns the Index view. If you click the About tab, the `About()` action is invoked and this action returns the About view.

The two views can be found in the Views folder at the following location:

\Views\Home\About.aspx

\Views\Home\Index.aspx

The content of the Index view is contained in Listing 1.2.

LISTING 1.2 Views\Home\Index.aspx (C#)

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"  
    Inherits="System.Web.Mvc.ViewPage" %>  
  
<asp:Content ID="indexTitle" ContentPlaceHolderID="TitleContent" runat="server">  
    Home Page  
</asp:Content>  
  
<asp:Content ID="indexContent" ContentPlaceHolderID="MainContent" runat="server">
```

```

    <h2><%= Html.Encode(ViewData["Message"]) %></h2>
    <p>
        To learn more about ASP.NET MVC visit <a href="http://asp.net/mvc"
➔title="ASP.NET MVC Website">http://asp.net/mvc</a>.
    </p>
</asp:Content>

```

LISTING 1.2 Views\Home\Index.aspx (VB)

```

<%@ Page Language="VB" MasterPageFile="~/Views/Shared/Site.Master"
➔Inherits="System.Web.Mvc.ViewPage" %>

<asp:Content ID="indexTitle" ContentPlaceHolderID="TitleContent" runat="server">
    Home Page
</asp:Content>

<asp:Content ID="indexContent" ContentPlaceHolderID="MainContent" runat="server">
    <h2><%= Html.Encode(ViewData["Message"]) %></h2>
    <p>
        To learn more about ASP.NET MVC visit <a href="http://asp.net/mvc"
➔title="ASP.NET MVC Website">http://asp.net/mvc</a>.
    </p>
</asp:Content>

```

Notice that a view consists mostly of standard HTML content. For example, the view contains standard `<h2>` and `<p>` tags. A view generates a page that is sent to the browser.

Summary

The goal of this chapter was to provide you with an overview of the ASP.NET MVC framework. The first part of this chapter was devoted to a discussion of a definition of good software. You were provided with a brief introduction to software design principles and patterns and the importance of unit tests. You learned how software design principles and patterns and unit tests enable you to create software that is resilient to change.

Next, you were provided with an introduction to the Model View Controller software design pattern. You learned about the history and benefits of this pattern. You learned how the ASP.NET MVC framework implements the Model View Controller pattern and how ASP.NET MVC enables you to perform pattern-based software development.

Finally, we explored the sample ASP.NET MVC application that is created when you create a new ASP.NET MVC project. We took our first look at an ASP.NET MVC controller and an ASP.NET MVC view.

Index

Symbols

^= operator, 487

=> (goes to) operator, 655

A

About() method, 20-21

AcceptAjax attribute, 473-476

AcceptVerbs attribute (actions), 65-69

access, testing

with fake generic repository, 155-157

with mock repository, 150-155

overview, 149-150

Account controller, users and roles, 367-369

action filters

definition of, 207

FeaturedProductActionFilter, 316-318

log action filter, 237-240

overview, 236

Action() HTML helper, 161-162

ActionLink() HTML helper, 160-161

ActionMethodSelector attribute (actions), 72-75

ActionName attribute (actions), 70-71

ActionResults, returning

ContentResult, 57-59

FileResult, 63-65

JsonResult, 59-62

overview, 51-52

RedirectResult, 55-57

types of ActionResult, 51-52

ViewResult, 52-55

actions

AcceptVerbs attribute, 65-69

action filters

definition of, 207

FeaturedProductActionFilter, 316-318

log action filter, 237-240

overview, 236

ActionMethodSelector attribute, 72-75

ActionName attribute, 70-71

ActionResults, returning

overview, 51-52

types of ActionResults, 51-52

invoking, 51

testing, 78-81

unknown actions, handling, 76-78

Add Controller dialog, 47-48**Add menu commands**

Controller, 47

New Item, 26

New Test, 661

Add New Item dialog, 26**Add New Test dialog, 514, 660-661****Add Reference command (Project menu), 274****Add Reference dialog box, 274****Add View dialog, 37-45, 84, 83**

AddAttribute() method, 180

AddCssClass() method, 176

AddModelError() method, 244

AddStyleAttribute() method, 180

ADO.NET Entity Designer, 123

Agile Principles, Patterns, and Practices in C#
(Martin and Martin), 10

Ajax

AcceptAjax attribute, 473-476

AjaxOptions class

LoadingElementId property, 436

OnBegin property, 439

OnComplete property, 439

debugging routes, 428-429

helpers, 462

Ajax.ActionLink() helper, 454, 462, 468.
See also asynchronous content
retrieval

Ajax.BeginForm() helper, 430. See also
posting forms asynchronously

Ajax.BlogPager() helper, 618-619

required libraries, 427-428

and jQuery, 491-498

MicrosoftAjax.js library, including in pages,
427-428

MicrosoftMvcAjax.js library, including,
427-428

overview, 426-427

posting forms asynchronously

displaying progress, 435-442

downlevel browser support, 452-455

sample application, 430-435

updating content after posting, 443-447

validation, 447-452

retrieving content asynchronously

creating delete links, 462-467

downlevel browser support, 468-473

highlighting selected link, 459-462

sample application, 454-459

supporting in UnleashedBlog application

Ajax.BlogPager() helper, 618-619

BlogEntries partial, 616-617

Index_Ajax() method, 614-615

Index_AjaxReturnsPartialViewResult()
method, 614

modified Index view, 615-616

overview, 612-613

ajax() method, 491

Ajax.ActionLink() helper, 454, 462, 468. See
also asynchronous content retrieval

Ajax.BeginForm() helper, 430. See also posting
forms asynchronously

AjaxMethodAttribute class, 72

AjaxOptions class

LoadingElementId property, 436

OnBegin property, 439

OnComplete property, 439

ajaxSetup() method, 496

alternative view engines

Brail, 98

custom view engine

creating, 99-104

testing, 114-117

NHaml, 98

nVelocity, 98

overview, 97-98

Spark, 98

animations

displaying as progress indicators, 439-442

jQuery animations, 489-491

anonymous types, 649-651

antiforgery tokens, 169-173

AntiForgeryToken() HTML helper, 169-173

App_Data folder, 19

Application_Start() method, 271

ApplicationController class, 308-309

ApplicationName setting, 378

applications

architecture, 16-17

bin deployment, 424-425

blog. See *UnleashedBlog* application

MyFirstMvcApp sample application

code listings, 20-22

creating, 17-18

folder conventions, 19-19

running, 19-20

Toy Store. See *Toy Store* application

architecture of ASP.NET MVC applications, 16-17

ArchiveController class, 549-552

ArchiveControllerTests class, 544-549, 572-573

ArchiveYear test, 559

ArchiveYearMonth test, 559

ArchiveYearMonthDay test, 559

ArchiveYearMonthDayName() method, 559, 641-642

AreEqual() method, 669-670

AreEquivalent() method, 669

ASP.NET MVC 1.0, 1

ASP.NET Unleashed (Walther), 15

ASP.NET Web Forms,

combining with ASP.NET MVC, 424

modifying to support ASP.NET MVC

Global.asax file, 422-424

overview, 414

required assemblies, 415-416

Visual Studio project files, 415

web configuration files, 416-422

assemblies

adding, 415-416

System.Web.Abstractions, 415

System.Web.Mvc, 415

System.Web.Routing assembly, 415

Assert class, 669

assertions, 669-672

asynchronous content retrieval

creating delete links, 462-467

downlevel browser support, 468-473

highlighting selected link, 459-462

sample application, 454-459

Asynchronous JavaScript and XML. See Ajax

asynchronous posting of forms

displaying progress, 435-442

downlevel browser support, 452-455

sample application, 430-435

updating content after posting, 443-447

validation, 447-452

attacks

CSRF (cross-site request forgery) attacks, 169

JavaScript injection attacks, 95-97

Attributes property (TagBuilder class), 176

AuthenticatedConstraint, 280-283

authentication

authorizing users

with Authorize attribute, 368-370

authorizing particular roles, 371-372

authorizing particular users, 370-371

overview, 368

with User property, 372-374

membership, configuring

with Membership and Role Manager API, 381-385

membership database, 375-379

membership settings, 378-380

overview, 363-365

testing

for Authorize attribute, 390-392

with user model binder, 393-400

users and roles, creating

with Account controller, 367-369

with Web Site Administration Tool, 365-366

Windows authentication

authenticating Windows users and groups, 386-390

configuring, 385-387

overview, 385

types of authentication, 386

AuthenticationType property (Identify object), 373

Authorize attribute, 368-370

testing for, 390-392

authorizing users

with Authorize attribute, 368-370

authorizing particular roles, 371-372

authorizing particular users, 370-371

overview, 368

with User property, 372-374

avoiding code smells, 9-10

B

Basic authentication, 386

Beck, Kent, 509

BeginForm() HTML helper, 162, 166-167

bin deployment, 424-425

Bind attribute (model binders)

applying to classes, 221-225

applying to method parameters, 218-221

prefixes, 225-228

binding to complex classes, 212-218

blog application. See UnleashedBlog application

blog entries, creating, 520-523

BlogArchive route, 276

BlogController class

BlogArchive route, 277

_blogEntries field, 522-523

BlogRepositoryBase class. See BlogRepositoryBase class

BlogService. See BlogService class

CreateNameFromTitle() method, 589-590

creating, 517-518

Entity Framework blog repository, 537-539

ignoring Id property, 580-581

Index_Ajax() method, 614-615

paging support, 597-601

validating blog entry title, 570-571

validating length of property, 577-578

BlogControllerTests class

CreateBlogEntry() method, 520-521
 CreateNamelsValid() method, 587
 CreateTitleMaximumLength500() method, 576-577
 CreateTitleRequired() method, 568-569
 FakeBlogRepository(), 528-530
 Index_AjaxReturnsPartialViewResult() method, 614
 IndexReturnsBlogEntriesByYear() test, 575
 paging tests, 596-600
 ShowNewBlogEntries() method, 515-516

_blogEntries field, 588-589

BlogEntries partial, 607-608, 616-617

BlogEntriesIncludeCommentCount() method, 631-632

BlogEntry class, 517

BlogEntryEntity class, 536-537

BlogEntryFactory class, 573-576

BlogLink() HTML helper, 608-609

BlogLinkHelper class, 608-609

BlogPager() Ajax helper, 618-619

BlogPager() HTML helper, 610-612

BlogPagerHelper class, 610-612

BlogRepositoryBase class

first iteration, 524-525
 ListBlogEntries() method, 552-553
 paging support, 602-605

BlogService class

blog entry Name property, 588-589
 initial code listing, 581-583
 ListBlogEntries() method, 601-603

Brail, 98

browsers, downlevel browser support

posting forms asynchronously, 452-455
 retrieving content asynchronously, 468-473

BulletdListHelper class, 180-182

business rules in UnleashedBlog application, 586-591

C**C# language features**

anonymous types, 649-651
 extension methods, 652-654
 generics, 654-655
 lambda expressions, 655
 LINQ (Language Integrated Query), 656-658
 nullable types, 651-652
 object initializers, 648
 type inference, 647-648

Cache class, 347-353

Cache-Control HTTP header, 345

CacheControlController class, 345-346

CacheWrapper class, 360

caching, 333-335

with Cache class, 347-353
 with HttpCachePolicy class, 345-346
 with OutputCache attribute, 330-331
 cache location, setting, 333-335
 cache profiles, 343-344
 Location property, 333-335
 removing items from output cache, 341-343
 sample application, 325-330
 security issues, 330-331
 VaryByContentEncoding property, 337
 VaryByCustom property, 338-341
 VaryByHeader property, 337-338
 VaryByParam property, 335-337
 varying output cache, 335-341
 what gets cached, 331-333
 overview, 323-325
 sliding expiration cache policy, 351-352
 testing cache
 OutputCache attribute, 353-355
 overview, 353
 verifying that database data is cached, 355-362

call manager application case study, 7-8

Cascading Style Sheets (CSS), error message styles, 247-248

CatalogController class, 77-78

catch-all parameter (routes), 285-288

Certification authentication, 386

ChangePassword() method, 367

ChangePasswordSuccess() method, 367

changing passwords, 368-369

CheckBox() HTML helper, 162-165

classes, 325-330

AjaxMethodAttribute, 72

AjaxOptions

LoadingElementId property, 436

OnBegin property, 439

OnComplete property, 439

ApplicationController, 308-309

ArchiveController, 549-552

ArchiveControllerTests, 544-549, 572-573

Assert, 669

binding to complex classes, 212-218

BlogController

BlogArchive route, 277

_blogEntries field, 522-523

BlogRepositoryBase class. See BlogRepositoryBase class

BlogService. See BlogService class

CreateNameFromTitle() method, 589-590

creating, 517-518

Entity Framework blog repository, 537-539

ignoring Id property, 580-581

Index_Ajax() method, 614-615

paging support, 597-601

validating blog entry title, 570-571

validating length of property, 577-578

BlogControllerTests

CreateBlogEntry() method, 520-521

CreateNameIsValid() method, 587

CreateTitleMaximumLength500() method, 576-577

CreateTitleRequired() method, 568-569

FakeBlogRepository(), 528-530

Index_AjaxReturnsPartialViewResult() method, 614

IndexReturnsBlogEntriesByYear() test, 575

paging tests, 596-600

ShowNewBlogEntries() method, 515-516

BlogEntry, 517

BlogEntryEntity, 536-537

BlogEntryFactory, 573-576

BlogLinkHelper, 608-609

BlogPagerHelper, 610-612

BlogRepositoryBase, 602-605

first iteration, 524-525

ListBlogEntries() method, 552-553

BlogService

blog entry Name property, 588-589

initial code listing, 581-583

ListBlogEntries() method, 601-603

BulletedListHelper, 180-182

Cache, 347-353

CacheControlController, 345-346

CacheWrapper, 360

CatalogController, 77-78

CollectionAssert, 669

Comment, 624-625

CommentController, 625-626

CommentControllerTests

BlogEntriesIncludeCommentCount() method, 631-632

CommentsOrderByDatePublished() method, 629-630

CreateAndThenGetComment() method, 627-629

CreateComment() method, 622-623

- CompanyController, 369-372
- ContentManagerController, 63-64
- Controller, 372-374
 - controller classes, creating, 681-682
 - creating from interfaces, 681-690
- Customer, binding to, 213-218
- CustomerController, 53-54
- data model classes. *See* data models
- DataGridHelperBasic, 183-187
- DataGridHelperTests, 201-205
- DeleteController, 463-465
- DownlevelController, 453-454
- DownLinkController, 468-471
- DynamicController, 302-303
- EmployeeController, 65-67
- EntityFrameworkBlogRepository, 534-537, 634-636
- Enumerable, 656
- FakeBlogRepository, 526-528, 632-633
- FakeClass, 360
- FakeIdentity, 398-400
- FakeMovieRepository, 267
- FakePrincipal, 396-398
 - generics, 654-655
- GuestBookController, 446-447
- HelloController, 58
- HomeController, 20-21
 - caching, 325-330
 - creating, 30-37
 - HomeController class listing in C#, 31-32, 34-35
 - HomeController class listing in VB, 32-34, 35-36
 - testing, 106-108
- HomeControllerTest, 106-108
- HomeControllerTestFake, 155-157
- HomeControllerTestMock, 153-155
- HTMLTextWriter, 180-183
- HttpCachePolicy, 345-346
- ImageLinkHelper, 177-180
- JackController, 391
- JackControllerTests, 391-392
- JillController, 393-394
- JillControllerTests, 394-396
- LogActionFilter, 237-239
- LogController, 239-240
- LookupController, 382-383
- MasterDetailController, 457-458
- MathUtility, 666
- MathUtilityTests, 666-668
- Membership, 381-385
- MembershipUser, 381-385
- MerchandiseController, 70-71
- Movie2Controller, 252-253
- Movie2ControllerTests, 264-266
- MovieController, 684-686
 - model state, 242-244
 - posting forms asynchronously, 430-435
 - retrieving movies, 496-498
- MovieRepository, 257-258, 347-349, 686-687
- MovieService, 254-256, 684-686
- MovieServiceTests, 688-689
- NewsController, 73-74, 493-494
- PagedList, 193-195
- PageList, 594-596
- PagingLinqExtensions, 195-197
- PersonController, 78-81
- Product, 261-263
- ProductController, 48-51, 259-261, 318-319
- ProductControllerTests
 - C# code listing, 661-663
 - VB code listing, 663-664
- ProductHelper, 110-112

ProductHelperTest, 112-114
 ProductRepository, 133-136
 ProfileController, 344
 Queryable, 656-657
 QuotationController, 59-60
 RemoveController, 342-343
 Repository, 148-149
 Roles, 382
 RouteTest, 293-294, 289-290
 RouteTests, 554-558, 641-642
 SelectorController, 457-476
 ServerValidateController, 450-451
 SimpleControllerTest, 114-116
 SimpleMovieController, 358-359
 SimpleMovieRepository, 356
 SimpleMovieService, 356-358
 SimpleView, 100-103
 SimpleViewEngine, 99-104
 SlidingController, 351-352
 SortController, 288
 StringAssert, 669
 TagBuilder, 176-180
 TheaterController, 319-320
 UserController, 373-374
 UserModelBinder, 233-236
 VaryCustomController, 340
 ViewDataDictionary, 91
 VirtualPathProviderViewEngine, 99
 WidgetController, 55-56
 WindowsController, 387-390

classic mode (IIS), 402-403

code samples, downloading, 3

code smells, avoiding, 9-10

CollectionAssert class, 669

commands. *See specific commands*

Comment class, 624-625

CommentController class, 625-626

CommentControllerTests class

BlogEntriesIncludeCommentCount() method, 631-632

CommentsOrderByDatePublished() method, 629-630

CreateAndThenGetComment() method, 627-629

CreateComment() method, 622-623

comments, adding to blog application

adding comments to database, 633-637

BlogEntriesIncludeCommentCount() method, 631-632

Comment class, 624-625

CommentsOrderByDatePublished() method, 629-630

CreateAndThenGetComment() method, 627-629

CreateComment() method, 622-623, 627-628

displaying comments and comment counts, 637-643

modified FakeBlogRepository class, 632-633

overview, 619-622

Comments database table, adding comments to, 633-637

CommentsOrderByDatePublished() method, 629-630

CompanyController class, 369-372

complexity in software, 10

configuring

ASP.NET Web Forms to support ASP.NET MVC

Global.asax file, 422-424

overview, 414

required assemblies, 415-416

Visual Studio project files, 415

web configuration files, 416-422

- default routes, 272-273
- IIS (Internet Information Services)
 - hosted server, 408-410
 - integrated versus classic mode, 402-403
 - overview, 401-402
 - route table, adding extensions to, 403-408
 - wildcard script maps, 410-414
- membership
 - with Membership and Role Manager API, 381-385
 - membership database, 375-379
 - membership settings, 378-380
- Windows authentication, 385-387
- CONNECT operation (HTTP), 68, 462**
- constraints**
 - route constraints, creating
 - AuthenticatedConstraint, 280-283
 - HttpMethod constraint, 280-281
 - NotEqual constraint, 283-285
 - regular expression constraints, 278-279
 - testing routes with, 292-294
- Contains() method, 669**
- Content folder, 19**
- Content() method, overloading, 59**
- ContentManagerController class, 63-64**
- ContentResult, 52**
 - returning, 57-59
- Controller class, 372-374**
- Controller command (Add menu), 47**
- controllers, 16**
 - ActionResults, returning
 - ContentResult, 57-59
 - FileResult, 63-65
 - JsonResult, 59-62
 - overview, 51-52
 - RedirectResult, 55-57
 - types of ActionResult, 51-52
 - ViewResult, 52-55
 - actions
 - AcceptVerbs attribute, 65-69
 - ActionMethodSelector attribute, 72-75
 - ActionName attribute, 70-71
 - invoking, 51
 - testing, 78-81
 - unknown actions, handling, 76-78
 - ApplicationController class, 308-309
 - ArchiveController class, 549-552
 - BlogController class
 - BlogArchive route, 277
 - _blogEntries field, 522-523
 - BlogRepositoryBase class. See BlogRepositoryBase class
 - BlogService. See BlogService class
 - CreateNameFromTitle() method, 589-590
 - Entity Framework blog repository, 537-539
 - ignoring Id property, 580-581
 - Index_Ajax() method, 614-615
 - paging support, 597-601
 - validating blog entry title, 570-571
 - validating length of property, 577-578
 - CacheControlController class, 345-346
 - CatalogController class, 77-78
 - CommentController class, 625-626
 - CompanyController class, 369-372
 - ContentManagerController class, 63-64
 - Controller class, 372-374
 - creating, 47-51, 681-684
 - CustomerController class, 53-54
 - DeleteController class, 463-465
 - DownlevelController class, 453-454
 - DownLinkController class, 468-471
 - DynamicController class, 302-303
 - EmployeeController class, 65-67

GuestBookController class, 446-447

HelloController class, 58

HomeController class, 20-21

 caching, 325-330

 creating, 30-37

 HomeController class listing in C#,
 31-32, 34-35

 HomeController class listing in VB,
 32-34, 35-36

 testing, 106-108

JackController class, 391

JillController class, 393-394

LookupController class, 382-383

MasterDetailController class, 457-458

MerchandiseController class, 70-71

Movie2Controller class, 252-253

MovieController class, 684-686

 model state, 242-244

 posting forms asynchronously, 430-435

 retrieving movies, 496-498

NewsController class, 73-74, 493-494

overview, 46-47

PersonController class, 78-81

ProductController class, 259-261, 318-319

ProfileController class, 344

QuotationController class, 59-60

RemoveController class, 342-343

SelectorController class, 457-476

ServerValidateController class, 450-451

setting view master pages from, 302-304

SimpleMovieController class, 358-359

SlidingController class, 351-352

SortController class, 288

TheaterController class, 319-320

UserController class, 373-374

VaryCustomController class, 340

WidgetController class, 55-56

WindowsController class, 387-390

Controllers folder, 19

controls. See user controls

ConvertCommentToCommentEntity() method, 637

Create() method, 65

 creating records, 127-128

 HomeController class, 34-37

 UnleashedBlog application, 521-522

Create Unit Test Project dialog, 24, 513, 511, 660

Create view

 Ajax posts, 432-435

 for Toy Store application, 42-45

CreateAndThenGetComment() method, 627-629

createBeing() method, 442

CreateBlogEntry() method, 520-523, 536, 588-591, 627-628

CreateComment() method, 622-623, 627-628

createComplete() method, 442

CreateItems() method, 205

CreateMovie() method, 349

 fake values, returning, 690-693

CreateNameFromTitle() method, 589-590

CreateNamelsValid() method, 587

createSuccess() method, 435

CreateTitleMaximumLength500() method, 576-577

CreateTitleRequired() method, 568-569

CreateWithBadMovieReturnsView() method, 693

CreateWithGoodMovieReturnsRedirect() method, 693

cross-site scripting (XSS) attacks, 96

CSRF (cross-site request forgery) attacks, 169

CSS (Cascading Style Sheets), error message styles, 247-248

custom HTML helpers, creating

HTML.DataGrid() helper, 183-201

C# code listing, 183-185

calling, 187-188

paging support, 192-201

reflection, 188-189

sorting support, 190-192

testing, 201-205

VB code listing, 186-187

HTML.SubmitButton() example, 173-176

with HTMLTextWriter class, 180-183

with TagBuilder class, 176-180

custom model binders, creating, 233-236**custom routes, creating, 275-277****custom view engine**

creating, 99-104

testing, 114-117

Customer class, binding to, 213-218**CustomerController class, 53-54**

D

data access, testing

with fake generic repository, 155-157

with mock repository, 150-155

overview, 149-150

data models

creating with Microsoft Entity Framework, 120-124

data access, testing

with fake generic repository, 155-157

with mock repository, 150-155

overview, 149-150

for Entity Framework blog repository, 532-533

overview, 117-119

records

creating, 127-128

deleting, 131-132

editing, 128-130

listing, 124-126

retrieving single record, 126

Repository pattern

Dependency Injection pattern, 138-139

generic repositories, 139-149

overview, 132

product repositories, 133-138

for Toy Store application, 27-30

databases

adding comments to, 633-637

database objects for Entity Framework blog repository, 531-532

membership database, configuring, 375-379

Toy Store database, creating, 23-27

DataGrid() HTML helper, 183-201

C# code listing, 183-185

calling, 187-188

paging support, 192-201

PagedList class, 193-195

PagedSortedProducts() action, 200-201

PagingLinqExtensions class, 195-197

reflection, 188-189

sorting support, 190-192

testing, 201-205

VB code listing, 186-187

DataGridHelperBasic class, 183-187**DataGridHelperTests class, 201-205****dateReleased variable, 651****debugging**

Ajax, 428-429

routes, 274-275

default model binder

- binding to complex classes, 212-218
- overview, 210-212

default routes

- configuring, 272-273
- Global.asax file, 269-271

DefaultRoute test, 559**DefaultRouteMatchesHome() method, 289-290****delete links, creating, 462-467****Delete() method, 131-132****DELETE operation (HTTP), 68-69, 462****Delete_GET() action, 465****Delete_POST() action, 465****DeleteController class, 463-465****deleting records, 131-132****Dependency Injection pattern, 138-139****deployment, bin, 424-425****Description setting, 378****design (software)**

- design patterns, 11-12
- design principles, 10-11
- short-term versus long-term planning, 14
- test-driven development, 13
- unit tests, 12-13

Design Patterns: Elements of Reusable Object-Oriented Software, 11**Details() method, 34**

- retrieving single record, 126

Details view for UnleashedBlog application, 637-641**DetailsWithId() method, 80-81****DetailsWithoutId() method, 80-81****development, test-driven. See test-driven development****dialogs. See specific dialogs****Digest authentication, 386****disabling request validation, 97****divLoading element, 436*****Domain-Driven Design (Evans), 11*****doubles, 680****downlevel browser support**

- asynchronous content retrieval, 468-473
- posting forms asynchronously, 452-455

DownlevelController class, 453-454**DownLinkController class, 468-471****downloading**

- code samples, 3
- jQuery plug-ins, 498-499
- NUnit, 672

drop-down lists, rendering, 167-168**DropDownList() HTML helper, 162, 167-168****DynamicController class, 302-303****E****Edit() method, 34****editing records, 128-130****EFGenericRepository project, 140****EFMvcApplication project, 142-144****embedding scripts in views, 86-87****EmployeeController class, 65-67****EmptyResult, 52****EnablePasswordReset setting, 379****EnablePasswordRetrieval setting, 379****Encode() HTML helper, 169****encoding HTML content, 169****EndForm() HTML helper, 162, 166-167****Entity Data Model Wizard, 28-30****Entity Framework. See Microsoft Entity Framework****EntityFrameworkBlogRepository class, 534-537, 634-636****Enumerable class, 656**

error messages

- Sys Is Undefined error, 428
- Type Is Undefined error, 428
- validation error messages
 - prebinding versus postbinding, 248-250
 - styling, 247-248
 - in UnleashedBlog application, 578-581

ETag HTTP header, 345**Evans, Eric, 11-12****event handlers in jQuery, 487-488. See also specific event handlers****evolutionary design, 507-508****Exclude property (Bind attribute), 218****Expires HTTP header, 345****expressions, lambda, 655****extending generic repositories, 147-149****extension methods, 652-654****extensions, adding to route table, 403-408**

F

Factoring: Improving the Design of Existing Code* (Fowler), 506*fake repositories**

- testing data access with, 155-157
- for UnleashedBlog application, 526-530, 632-633

fake values, returning, 690-693**FakeBlogRepository class, 526-528, 632-633****FakeCache class, 360****FakeIdentity class, 398-400****FakeMovieRepository class, 267****FakePrincipal class, 396-398****fakes, definition of, 680****Feathers, Michael, 9, 12,****FeaturedProductActionFilter, 316-318****Fiddler, 428-429****File menu commands, New Project, 23****File() method, overloading, 64****FileResult, 52, 63-65****files. See specific files****filters. See action filters****Firebug, 429****folders, conventions for, 19****form collection model binder, 228-231****form validation. See validation****forms**

- form elements, rendering, 162-165
- posting asynchronously
 - displaying progress, 435-442
 - downlevel browser support, 452-455
 - sample application, 430-435
 - updating content after posting, 443-447
 - validation, 447-452
- rendering, 166-167
- validating. See validation

Fowler, Martin, 11, 506, 680**fragility in software, 10**

G

GenerateId() method, 176**generic repositories**

- creating, 139-141
- extending, 147-149
- with LINQ to SQL, 144-147
- with Microsoft Entity Framework, 141-144
- testing data access with fake generic repositories, 155-157

GenericRepository project, 140

GenericRepository.Tests project, 140**generics, 654-655****get() method, 491****GET operation (HTTP), 67, 462****getJSON() method, 491, 494****GetProductCount() method, 148****GetRandomProducts() method, 318****getScript() method, 491****GetWithDatePublished() method, 574, 573****Global.asax file, 269-271**

adding routes to, 561-563, 642

configuring ASPNET Web Forms files to support ASPNET MVC, 422-424

hosted server configuration, 408-410

registering custom view engines in, 103

route table, adding extensions to, 403-408

wildcard script maps, 412-413

goes to (=>) operator, 655**guestbook application**

updating content after posting, 443-447

validation, 447-452

GuestBookController class, 446-447**Guthrie, Scott, 16**

H

HandleUnknownAction() method, 76-78**HasErrorMessage() method, 569*****Head First Design Patterns, 11*****HEAD operation (HTTP), 67, 462****headers (HTTP), 345****HelloController class, 58****helpers, Ajax**

Ajax.ActionLink() helper, 454-457, 462.

See also asynchronous content retrieval

Ajax.BeginForm() helper, 430. See also posting forms asynchronously

Ajax.BlogPager() helper, 618-619

required libraries, 427-428

helpers, HTML. See HTML helpers**Heraclitus, 7****Hidden() HTML helper, 162-165****highlighting selected links, 459-462****HomeController class, 20-21**

caching, 325-330

creating, 30-37

HomeController class listing in C#, 31-32, 34-35

HomeController class listing in VB, 32-34, 35-36

testing, 106-108

HomeControllerTest class, 106-108**HomeControllerTestFake class, 155-157****HomeControllerTestMock class, 153-155****host servers, 408-410****hover() method, 487-488****HTML content, encoding, 169****HTML helpers**

custom HTML helpers, creating

HTML.ImageLink() example, 177-180

HTML.SubmitButton() example, 173-176

with HTMLTextWriter class, 180-183

with TagBuilder class, 176-180

HTML.ActionLink() helper, 160-161

HTML.AntiForgeryToken() helper, 169-173

HTML.BeginForm() helper, 162, 166-167

Html.BlogLink() helper, 608-609

Html.BlogPager() helper, 610-612

HTML.CheckBox() helper, 162-165

HTML.DataGrid() helper. See HTML.DataGrid() helper

HTML.DropDownList() helper, 162, 167-168

HTML.Encode() helper, 169

HTML.EndForm() helper, 162, 166-167

HTML.Hidden() helper, 162-165

HTML.ListBox() helper, 162-165

HTML.Password() helper, 162-165
HTML.RadioButton() helper, 162-165
HTML.TextArea() helper, 162-165
HTML.TextBox() helper, 162-165
overview, 157-160
testing, 108-114, 201-205
URL.Action() helper, 161-162

HTML links

creating delete links, 462-467
highlighting selected link, 459-462
image links, rendering, 161-162
rendering, 160-161

HTML.ActionLink() helper, 160-161

HTML.AntiForgeryToken() helper, 169-173

HTML.BeginForm() helper, 162, 166-167

Html.BlogLink() helper, 608-609

Html.BlogPager() helper, 610-612

HTML.CheckBox() helper, 162-165

HTML.DataGrid() helper, 183-201

C# code listing, 183-185
calling, 187-188
paging support, 192-201
 PagedList class, 193-195
 PagedSortedProducts() action, 200-201
 PagingLinqExtensions class, 195-197
reflection, 188-189
sorting support, 190-192
testing, 201-205
VB code listing, 186-187

HTML.DropDownList() helper, 162, 167-168

HTML.Encode() helper, 169

HTML.EndForm() helper, 162, 166-167

HTML.Hidden() helper, 162-165

HTML.ImageLink() helper, 177-180

HTML.ListBox() helper, 162-165

HTML.Password() helper, 162-165

HTML.RadioButton() helper, 162-165

HTML.SubmitButton() helper, 173-176

HTML.TextArea() helper, 162-165

HTML.TextBox() helper, 162-165

HTMLTextWriter class, 180-183

Html.ValidationMessage() helper, 245-247

Html.ValidationSummary() helper, 245-247

HTTP

headers, 345

HTTP posted file base model binder,
231-233

operations, 67-68, 462

HttpCachePolicy class, 345-346

HttpMethod constraint, 280-281

HttpUnauthorizedResult, 52

I

ICache interface, 359

**IDataErrorInfo interface, validating form data
with, 258-263**

**IdAttributeDotReplacement property (TagBuilder
class), 176**

Identity object, 373

IEnumerable interface, 91, 656

IGenericRepository interface, 140

IIS (Internet Information Services) configuration

hosted server, 408-410

integrated versus classic mode, 402-403

overview, 401-402

route table, adding extensions to, 403-408

wildcard script maps, 410-414

image links, rendering, 161-162

ImageLink() HTML helper, 177-180

ImageLinkHelper class, 177-180

importing namespaces

overview, 519

UnitTesting namespace, 665

Include property (Bind attribute), 218

including libraries. See libraries

Index() method, 20-21, 34-37, 49

listing records, 124-126

UnleashedBlog application, 519-520

Index view

MyFirstMvcApp sample application, 21-22

for Toy Store application, 37-42, 37-45,
39-42

for UnleashedBlog application, 605-611,
615-616

Index_Ajax() method, 614-615

**Index_AjaxReturnsPartialViewResult() method,
614**

IndexAcceptsPage() method, 597-600

IndexAddsMoviesToCache() method, 360-362

IndexedCached() method, 325

**IndexRetrievesMoviesFromCache() method,
360-362**

IndexReturnsBlogEntriesByYear() test, 571, 575

**IndexReturnsBlogEntriesInOrderOfDatePublished
() method, 597-600**

**IndexReturnsLessThan6BlogEntries() method,
597-600**

**IndexReturnsPagedListForPage() method,
597-600**

initializers (object), 648

injection attacks, preventing, 95-97

InnerHTML property (TagBuilder class), 176

installing

Moq, 680-681

NUnit, 673

integrated mode (IIS), 402-403

Integrated Windows authentication, 386

Intellisense (Visual Studio) and jQuery, 481-482

interfaces

creating classes from, 681-690

generics, 654-655

ICache, 359

IDataErrorInfo, validating form data with,
258-263

IEnumerable, 91, 656

IGenericRepository, 140

IProductRepository, 133

IQueryable, 657

IRepository, 147-148

**Internet Information Services. See IIS (Internet
Information Services) configuration**

invoking

Archive controller, 564-565

controller actions, 51

IProductRepository interface, 133

IQueryable interface, 657

IRepository interface, 147-148

***Is Design Dead?* (Fowler), 509**

IsAjaxRequest() method, 454

IsAuthenticated property (Identity object), 373

IsInRole() method, 373

IsInstanceOfType() method, 669

J-K

JackCanAccessIndex() method, 391-392

JackCannotAccessIndex() method, 394-396

JackController class, 391

JackControllerTests class, 391-392

JavaScript, injection attacks, preventing, 95-97

JavaScriptResult, 52

JillCanAccessIndex() method, 394-396

JillController class, 393-394

JillControllerTests class, 394-396

jQuery

and Ajax, 491-498

animations, 489-491

event handlers, 487-488

- including in views, 480-481
- overview, 480
- plug-ins, 498-501
 - downloading, 498-499
 - tablesorter, 499-501
- selectors, 482-487
- and Visual Studio Intellisense, 481-482

Json() method, overloading, 62

JsonResult, 52, 59-62

keyboard combinations for running unit tests, 665-666

KISS Principle (Keep It Simple Stupid), 507

L

lambda expressions, 655

Language Integrated Query (LINQ), 656-658

- LINQ to SQL, 144-147

Last-Modified HTTP header, 345

launching Microsoft Web Platform Installer, 2

length of property, validating, 576-578

libraries

- jQuery. *See* jQuery

- MicrosoftAjax.js library, including, 427-428

- MicrosoftMvcAjax.js library, including in pages, 427-428

limiting unit test results, 671-672

links (HTML)

- creating delete links, 462-467
- highlighting selected link, 459-462
- image links, rendering, 161-162
- rendering, 160-161

LINQ (Language Integrated Query), 656-658

- LINQ to SQL, 144-147

ListBlogEntries() method, 536, 552, 553, 601

ListBox() HTML helper, 162-165

listing records, 124-126

ListMovies() method, 349

ListMoviesCached() method, 349

lists, drop-down lists, 167-168

load() method, 491

LoadingElementId property (AjaxOptions class), 436

location of cache, setting, 333-335

Location property (OutputCache attribute), 333-335

LogActionFilter class, 237-239

LogController class, 239-240

LogOff() method, 367

LogOn() method, 367

long-term versus short-term planning, 14

LookupController class, 382-383

LSGenericRepository project, 140

LSMvcApplication project, 144-147

M

MapRoute() method, 272

maps, wildcard script maps, 410-414

Martin, Micah, 10

Martin, Robert, 10

master pages. *See* view master pages

MasterDetailController class, 457-458

Matches() method, 669

MathUtility class

- C# code listing, 667

- VB code listing, 668

MathUtilityTests class

- C# code listing, 675

- VB code listing, 676

MaxInvalidPasswordAttempts settings, 379**membership, configuring**

- with Membership API, 381-385
- membership database, 375-379
- membership settings, 378-380

Membership API, 381-385**Membership class, 381-385****MembershipUser class, 381-385****memory caching. See caching****MerchandiseController class, 70-71****MerchandiseRepository() method, 70-71****MergeAttribute() method, 176****messages (error). See error messages****Meszaros, Gerard, 680****methods. See specific methods****Microsoft ASP.NET MVC 1.0, 1****Microsoft Entity Framework**

data models

- creating models, 120-124
- creating records, 127-128
- deleting records, 131-132
- editing records, 128-130
- listing records, 124-126
- retrieving single record, 126

Entity Framework blog repository, creating

- database objects, 531-532

Entity Framework data model, 532-533

EntityFrameworkBlogRepository class, 534-537, 634-636

overview, 530-531

testing, 537-541

generic repositories, 141-144

Microsoft .NET Framework 3.5 Service**Pack 1, 1****Microsoft SQL Server Express, 25****Microsoft Visual Web Developer 2008 Service****Pack 1, 2****Microsoft Web Platform Installer, launching, 2****MicrosoftAjax.js library, including in pages, 427-428****MicrosoftMvcAjax.js library, including in pages, 427-428****MinRequiredNonalphanumericCharacters setting, 379****MinRequiredPasswordLength setting, 379****Mock Object Frameworks**

- doubles, 680
- fake values, returning, 690-693
- fakes, 680
- mocks, 680
- Moq, 679
 - classes, creating from interfaces, 681-690
 - installing, 680-681
 - unblocking, 681
- overview, 679
- Rhino Mocks, 679
- stubs, 680
- Typemock Isolator, 679

mocks

- definition of, 680
- mock repositories, testing data access with, 150-155

Mocks Aren't Stubs (Fowler), 510**model binders**

- Bind attribute
 - applying to classes, 221-225
 - applying to method parameters, 218-221
 - prefixes, 225-228
- custom model binders, creating, 233-236
- default model binder
 - binding to complex classes, 212-218
 - overview, 210-212
- form collection model binder, 228-231

HTTP posted file base model binder,
231-233

overview, 205-210

testing authentication with, 393-400

model state, 241-244

model state dictionary, 241

models. See data models

Models folder, 19

MonoRail, 16

Moq, 679

- classes, creating from interfaces, 681-690
- installing, 680-681
- unblocking, 681

Movie2Controller class, 252-253

Movie2ControllerTests class, 264-266

MovieController class

- C# code listing, 682-683
- model state, 242-244
- posting forms asynchronously, 430-435
- retrieving movies, 496-498
- VB code listing, 683-684

MovieMaster page, 310-311

MovieRepository class, 686-688

- caching, 347-349

MovieRepository class, 257-258

MovieService class, 254-256

- C# code listing, 684-685
- VB code listing, 685-686

MovieServiceTests class, 688-689

MovieTemplate user control, 322-323

MVC pattern, 16

MVCFakes assembly, 289

MyFirstMvcApp sample application

- code listings, 20-22
- creating, 17-18
- folder conventions, 19
- running, 19-20

N

Name property (Identity object), 373

Name setting, 379

namespaces

- importing, 519
- System.Linq namespace, 656
- UnitTesting, importing, 665

naming conventions for views, 39

needless complexity in software, 10

needless repetition in software, 10

nested master pages, 306-307

.NET framework, 14

.NET Framework 3.5 Service Pack 1, 1

New Item command (Add menu), 26

New Project command (File menu), 23

New Test command (Add menu), 660-661

Newkirk, James, 510

NewsController class, 73-74, 493-494

NHaml, 98

NotEqual constraint, 283-285

NTLM authentication, 385-386

nullable types, 651-652

NUnit, 508-509, 672-678

- creating tests, 660-666
- downloading, 672
- installing, 673
- running tests, 669-671

nVelocity, 98

O

object initializers, 648

objects

- database objects for Entity Framework blog repository, 531-532

Mock Object Frameworks. See Mock Object Frameworks

object initializers, 648

OnBegin property (AjaxOptions class), 439

OnComplete property (AjaxOptions class), 439

OnNameChanging() event handler, 263

OnPriceChanging() event handler, 263

opacity in software, 10

OPTIONS operation (HTTP), 67, 462

origins of ASP.NET MVC framework, 14

OutputCache attribute

cache location, setting, 333-335

cache profiles, 343-344

Location property, 333-335

removing items from output cache, 341-343

sample application, 325-330

security issues, 330-331

testing, 353-355

VaryByContentEncoding property, 337

VaryByCustom property, 338-341

VaryByHeader property, 337-338

VaryByParam property, 335-337

varying output cache, 335-341

what gets cached, 331-333

overloading

Content() method, 59

File() method, 64

Json() method, 62

RedirectToAction() method, 57

View() method, 55

P

PagedList class, 193-195

PagedSortedProducts() action, 200-187

PageList class, 594-596

pageReady() method, 462, 485

paging

supporting in HTML.DataGrid() HTML helper, 192-201

PagedList class, 193-195

PagedSortedProducts() action, 200-201

PagingLinqExtensions class, 195-197

supporting in UnleashedBlog application

BlogController Index() method, 600-601

BlogRepositoryBase class, 602-605

controller tests, 596-600

overview, 591

PageList class, 594-596

PagingLinqExtensions class, 195-197

PartialView() method, 52

PartialViewResult, 51

passing

view data to user controls, 314-319

view data to view master pages, 308-311

Password() HTML helper, 162-165

PasswordAttemptWindow setting, 379

PasswordFormat setting, 379

passwords, changing, 368-369

PasswordStrengthRegularExpression setting, 379

patterns

Dependency Injection pattern, 138-139

Repository pattern. See Repository pattern

Patterns of Enterprise Application Architecture (Fowler), 11

PersonController class, 78-81

plug-ins (jQuery), 498-501

downloading, 498-499

tablesorter, 499-501

Poole, Charlie, 509

post() method, 491

POST operation (HTTP), 67, 462

postbinding validation error messages, 248-250

posting forms asynchronously

- displaying progress, 435-442
- downlevel browser support, 452-455
- sample application, 430-435
- updating content after posting, 443-447
- validation, 447-452

prebinding validation error messages, 248-250

Prefix property (Bind attribute), 218, 225

prefixes when binding, 225-228

preventing JavaScript injection attacks, 95-97

private data, caching, 330-331

Product class, 261-263

product repositories, creating, 133-138

ProductController class, 48-51, 259-261, 318-319

ProductControllerTests class, 660-661

ProductHelper class, 110-112

ProductHelperTest class, 112-114

ProductInsertDoesNotMatchGet() method, 293-294

ProductInsertMatchesPost() method, 293-294

ProductRepository class, 133-136

Products table (ToyStoreDB), 27

ProfileController class, 344

profiles (cache), 343-344

progress indicators, displaying, 435-442

Project menu commands, Add Reference, 274

properties, validating length of, 576-578

PUT operation (HTTP), 67, 462

Q-R

Queryable class, 656-657

QuotationController class, 59-60

RadioButton() HTML helper, 162-165

records

- creating, 127-128
- deleting, 131-132
- editing, 128-130
- listing, 124-126
- retrieving single record, 126

Red/Green/Reactor process, 505-506

RedirectResult, 52

returning, 55-57

RedirectToAction() method, 52

RedirectToRouteResult, 52

Reenskaug, Trygve, 15

Reeves, Jack, 509

refactoring

- overview, 12, 506
- UnleashedBlog application to use Repository pattern, 524-526

referencing jQuery, 480-481

reflection in HTML.DataGrid() helper, 188-189

Refresh() method, 494

Register() method, 367

RegisterRoutes() method, 271-272

regular expression constraints, 278-279

RemoveController class, 342-343

removing items from output cache, 341-343

Render() method, 103

RenderBeginTag() method, 180

RenderEndTag() method, 180

RenderHead() method, 190-191

rendering

- drop-down lists, 167-168
- form elements, 162-165
- forms, 166-167
- HTML links, 160-161
- image links, 161-162

RenderPagerRow() method, 199

RenderPartial() method, 313

repetition in software, 10

repositories

data access, testing with mock repository, 150-155

FakeMovieRepository class, 267

generic repositories

creating, 139-141

extending, 147-149

with LINQ to SQL, 144-147

with Microsoft Entity Framework, 141-144

MovieRepository class, 257-258, 347-349

product repositories, creating, 133-138

Repository class, 148-149

repository classes, creating, 686-688

SimpleMovieRepository class, 356

for UnleashedBlog application

BlogRepositoryBase class, 552-553

Entity Framework repository, creating, 530-541

fake blog repository, creating, 526-530, 632-633

Repository class, 148-149

Repository pattern, 11-12

creating product repositories, 133-138

Dependency Injection pattern, 138-139

generic repositories

creating, 139-141

extending, 147-149

with LINQ to SQL, 144-147

with Microsoft Entity Framework, 141-144

overview, 132

refactoring UnleashedBlog application to use, 524-526

request validation, disabling, 97

RequiresQuestionAndAnswer setting, 379

retrieving content asynchronously.

See asynchronous content retrieval

Rhino Mocks, 679

rigidity in software, 10

roles

authorizing, 371-372

creating

with Account controller, 367-369

with Web Site Administration Tool, 365-366

Roles class, 382

Roles class, 382

route constraints, creating

AuthenticatedConstraint, 280-283

HttpMethod constraint, 280-281

NotEqual constraint, 283-285

regular expression constraints, 278-279

Route Debugger, 274-275

route table, adding extensions to, 403-408

RouteDebugger, 289

routes. See routing

RouteTest class, 293-294, 289-290

RouteTests class, 554-558, 641-642

routing

catch-all parameter, 285-288

custom routes, creating, 275-277

debugging routes, 274-275

default routes, 269-273

configuring, 272-273

Global.asax file, 269-271

overview, 268-269

route constraints, creating

AuthenticatedConstraint, 280-283

HttpMethod constraint, 280-281

NotEqual constraint, 283-285

regular expression constraints, 278-279

testing routes

with constraints, 292-294

MvcFakes and RouteDebugger assemblies, 289

- overview, 288
- testing if URL matches route, 289-292
- UnleashedBlog application routes
 - adding to Global.asax file, 642
 - archive routes, 561-563
 - controller tests, 543-553
 - invoking Archive controller, 564-565
 - overview, 541-544
 - route tests, 553-560, 641-642

running

- MyFirstMvcApp sample application, 19-20
- unit tests
 - with NUnit, 669-671
 - with Visual Studio Unit Test, 669-671

S

- SalesFigures()** method, 389
- SaveChanges()** method, 128
- scripts, embedding in views, 86-87
- Scripts folder, 19
- Secrets()** method, 368-370
- SecretStuff()** method, 390
- security issues
 - authentication. See authentication
 - caching private data, 330-331
 - passwords, 368-369
- selectLink()** method, 462
- SelectorController** class, 457-476
- selectors (jQuery), 482-487
- ServerValidateController** class, 450-451
- service layers
 - in UnleashedBlog application, 581-586
 - validation with, 251-258

services

- BlogService class
 - blog entry Name property, 588-589
 - initial code listing, 581-583
 - ListBlogEntries() method, 601
- MovieService class, 254-256, 684-686
- SimpleMovieService class, 356-358

SetCacheability() method, 345-346

SetInnerText() method, 176

SetMaxAge() method, 346

Setup attribute (NUnit tests), 666

short-term versus long-term planning, 14

ShowNewBlogEntries() method, 515-516

SimpleControllerTest class, 114-116

SimpleMovieController class, 358-359

SimpleMovieRepository class, 356

SimpleMovieService class, 356-358

SimpleView class, 100-103

SimpleViewEngine class, 99-104

Single-Responsibility Principle (SRP), 581

slideDown() animation, 489-491

slideUp() animation, 489-491

sliding expiration cache policy, 351-352

SlidingController class, 351-352

software

- characteristics of bad software, 10

- code smells, avoiding, 9-10

- design

- design patterns, 11-12

- design principles, 10-11

- short-term versus long-term planning, 14

- test-driven development, 13

- unit tests, 12-13

- nature of good software

- call manager application case study, 7-8

- definition of, 8-9

- overview, 3-7

- software requirements, 2

SOLID (design principles), 11

SortController class, 288

sorting, supporting in HTML.DataGrid() HTML helper, 190-192

Spark, 98

SQL

LINQ to SQL, 144-147

SQL Server Express, 25

SRP (Single-Responsibility Principle), 581

state, model state, 241-244

strongly typed views, 94-95

stubs

creating, 688-689

definition of, 680

styles for validation error messages, 247-248

SubmitButton() HTML helper, 173-176

SuperSecrets() method, 370-371

SuperSuperSecrets() method, 392-393

Sys Is Undefined error, 428

System.Linq namespace, 656

System.Web.Abstractions assembly, 415

System.Web.Mvc assembly, 415

System.Web.Routing assembly, 415

T

tables

Products (ToyStoreDB), 27

route table, adding extensions to, 403-408

tablesorter plug-in (jQuery), 499-501

TagBuilder class, 176-180

TagName property (TagBuilder class), 176

TDD. See test-driven development

templates, user controls as, 319-323

Test attribute (NUnit tests), 666

test-driven development

benefits of, 506

bibliography and resources, 509

definition of, 505-506

KISS Principle (Keep It Simple Stupid), 507

overview, 13, 502-505

Red/Green/Reactor process, 505-506

TDD tests versus unit tests, 508

test flow from user stories, 508-509

Unit Testing Frameworks, 508-509

in UnleashedBlog application, 514-520

waterfall versus evolutionary design, 507-508

YAGNI Principle (You Ain't Gonna Need It), 507

***Test-Driven Development by Example* (Beck), 509**

***Test-Driven Development in Microsoft in .NET* (Newkirk and Vorontsov),**

TestFixture attribute (NUnit tests), 666

testing

authentication

for Authorize attribute, 390-392

with user model binder, 393-400

cache

OutputCache attribute, 353-355

overview, 353

verifying that database data is cached, 355-362

controller actions, 78-81

data access

with fake generic repository, 155-157

with mock repository, 150-155

overview, 149-150

Entity Framework blog repository, 537-541

HTML helpers, 201-205

routes

- with constraints, 292-294

- MvcFakes and RouteDebugger assemblies, 289

- testing if URL matches route, 289-292

test-driven development, 13, 506

- bibliography and resources, 509

- definition of, 505-506

- KISS Principle (Keep It Simple Stupid), 507

- overview, 502-505

- Red/Green/Reactor process, 505-506

- TDD tests versus unit tests, 508

- test flow from user stories, 508-509

- Unit Testing Frameworks, 508-509

- in UnleashedBlog application, 514-520

- waterfall versus evolutionary design, 507-508

- YAGNI Principle (You Ain't Gonna Need It), 507

- unit tests. *See* unit tests

- UnleashedBlog application.

- BlogControllerTests class

- validation, 264-268

views

- custom view engine, 114-117

- HTML helpers, 108-114

- overview, 105

- view results, 105-108

TextArea() HTML helper, 162-165**TextBox() HTML helper, 162-165****TheaterController class, 319-320****Time() method, 353****TimesCached() method, 354-355****titles**

- master page titles, 303-305

- titles of blog entries, validating, 567-573

- ArchiveControllerTests class, 572-573

- BlogController class, 570-571

- BlogControllerTests class, 568-569

- IndexReturnsBlogEntriesByYear() test, 571

ToString() method, 176**Toy Store application**

- controller

- creating, 30-37

- HomeController class listing in C#, 31-32, 34-35

- HomeController class listing in VB, 32-34, 35-36

- creating, 23-25

- data model classes, 27-30

- database, 23-27

- overview, 22-23

- views

- Create view, 42-45

- creating, 37-45

- Index view, 37-42

- naming conventions, 39

TRACE operation (HTTP), 68, 462**type inference, 647-648****Type Is Undefined error, 428****typed views, 88-95****types**

- anonymous types, 649-651

- nullable types, 651-652

- typed versus untyped views, 88-95

U**unblocking Moq, 681****unit testing frameworks, 508-509**

- NUnit, 672-678

- creating tests, 660-666

- downloading, 672

- installing, 673

- running tests, 669-671

overview, 659-660

Visual Studio Unit Test, 660-672

assertions, 669-672

creating unit tests, 660-664

limiting test results, 671-672

running tests, 669-671

test attributes, 666

Unit Test Wizard, 291

unit tests, 12-13

assertions, 669-672

compared to TDD tests, 508

creating

with NUnit, 660-666

PersonController class example, 78-81

with Visual Studio Unit Test, 60-661

frameworks. See unit testing frameworks

limiting test results, 671-672

RouteTest class, 289-290

running

with NUnit, 669-671

with Visual Studio Unit Test, 669-671

test attributes, 666

Unit Test Wizard, 291

UnitTesting namespace, importing, 665

unknown actions, handling, 76-78

UnleashedBlog application, 205

Ajax support

Ajax.BlogPager() helper, 618-619

BlogEntries partial, 616-617

Index_Ajax() method, 614-615

Index_AjaxReturnsPartialViewResult()
method, 614

modified Index view, 615-616

overview, 612-613

blog entries, creating, 520-523

blog projects, creating, 511-514

BlogArchive route, 276

BlogController class. See BlogController
class

BlogControllerTests class. See
BlogControllerTests class

comments

adding to database, 633-637

BlogEntriesIncludeCommentCount()
method, 631-632

Comment class, 624-625

CommentsOrderByDatePublished()
method, 629-630

CreateAndThenGetComment() method,
627-629

CreateComment() method, 622-623,
627-628

displaying comments and comment
counts, 637-643

modified FakeBlogRepository class,
632-633

overview, 619-622

Details view, 637-641

overview, 510-511

paging support

BlogController Index() method, 597-601

BlogRepositoryBase class, 602-605

controller tests, 596-600

overview, 591

PageList class, 594-596

refactoring to use Repository pattern,
524-526

repositories

BlogRepositoryBase class.
See BlogRepositoryBase class

Entity Framework repository, creating,
530-541

fake blog repository, creating, 526-530,
632-633

routes

adding to Global.asax file, 561-563

archive routes, 561-563

- controller tests, 543-553
- invoking Archive controller, 564-565
- overview, 541-544
- route tests, 553-560
- tests, creating, 514-520
 - Add New Test dialog, 514
 - BlogController class, 517-518
 - BlogControllerTests class, 515-516
 - BlogEntry class, 517
 - Index() method, 519-520
 - namespaces, importing, 519
- validation
 - BlogEntryFactory class, 573-576
 - business rules, 586-591
 - overview, 565-568
 - refactoring to use service layer, 581-586
 - validating blog entry title, 567-573
 - validating length of property, 576-578
 - validation error messages, 578-581
- views, 605-612
 - BlogEntries partial, 607-608
 - Html.BlogLink() helper, 608-609
 - Html.BlogPager() helper, 610-612
 - Index view, 605-611

untyped views, 88-95

UpdateModel() method, 228-230

updating form content, 443-447

URL.Action() helper, 161-162

URLs, testing if URL matches route, 289-292

user controls

- adding to views, 313-314
- creating, 312-313
- MovieTemplate user control, 322-323
- overview, 311-312
- passing view data to, 314-319
- as templates, 319-323

User property (Controller class), 372-374

user stories, test flow from, 508-509

UserController class, 373-374

UserModelBinder class, 233-234

users. See also user controls

- authentication. See authentication
- authorizing
 - with Authorize attribute, 368-370
 - authorizing particular users, 371-372
 - overview, 368
 - with User property, 372-374
- creating
 - with Account controller, 367-369
 - with Web Site Administration Tool, 365-366
- membership, configuring
 - with Membership and Roles Manager API, 381-385
 - membership database, 375-379
 - membership settings, 378-380
- passwords, changing, 368-369
- user stories, test flow from, 508-509

V

validation

- with Ajax posts, 447-452
- error messages
 - prebinding versus postbinding, 248-250
 - styling, 247-248
- with IDataErrorInfo interface, 258-263
- model state, 241-244
- overview, 240-241
- request validation, disabling, 97
- with service layers, 251-258

testing, 264-268

in UnleashedBlog application

BlogEntryFactory class, 573-576

business rules, 586-591

overview, 565-568

refactoring to use service layer, 581-586

validating blog entry title, 567-573

validating length of property, 576-578

validation error messages, 578-581

validation helpers, 245-247

A Value Is Required error message, 579-580

values, returning fake values, 690-693

variables, dateReleased, 651

VaryByContentEncoding property (OutputCache attribute), 337

VaryByCustom property (OutputCache attribute), 338-341

VaryByHeader property (OutputCache attribute), 337-338

VaryByParam property (OutputCache attribute), 335-337

VaryCustomController class, 340

VB language features

anonymous types, 649-651

extension methods, 652-654

generics, 654-655

lambda expressions, 655

LINQ (Language Integrated Query), 656-658

nullable types, 651-652

object initializers, 648

type inference, 647-648

verifying caching of database data, 355-362

view content pages, creating, 300-301

view data, 87-88

passing to user controls, 314-319

passing to view master pages, 308-311

view master pages

creating, 295-299

master page titles, 303-305

nested master pages, 306-307

overview, 294-295

passing view data to, 308-311

setting from controller, 302-304

view content pages, 300-301

View() method, 52, 54-55

view results, testing, 105-108

ViewDataDictionary class, 91

ViewResult, 52-55

views, 16. See also view master pages

alternative view engines

Brail, 98

custom view engines, 99-104, 114-117

NHaml, 98

nVelocity, 98

overview, 97-98

Spark, 98

Create, 432-435

creating, 83-87

embedding scripts in, 86-87

Index. See Index view

JavaScript injection attacks, preventing, 95-97

naming conventions, 39

overview, 82-83

testing

custom view engines, 114-117

HTML helpers, 108-114

overview, 105

view results, 105-108

for Toy Store application

Create view, 42-45

creating, 37-45

Index view, 37-42

- typed versus untyped views, 88-95
- for UnleashedBlog application, 605-612
 - BlogEntries partial, 607-608
 - Html.BlogLink() helper, 608-609
 - Html.BlogPager() helper, 610-612
 - Index view, 605-611
- user controls. See user controls
- view data, 87-88

Views folder, 19

VirtualPathProviderViewEngine class, 99

Visual Studio Intellisense and jQuery, 481-482

Visual Studio project files, modifying to support ASP.NET MVC, 415

Visual Studio Unit Test, 660-672, 508

- assertions, 669-672
- creating unit tests, 60-661
- limiting test results, 671-672
- running tests, 669-671
- test attributes, 666

Visual Web Developer 2008 Service Pack 1, 2

Vorontsov, Alexei, 510

WidgetController class, 55-56

wildcard script maps, 410-414

Windows authentication

- authenticating Windows users and groups, 386-390
- configuring, 385-387
- overview, 385
- types of authentication, 386

WindowsController class, 387-388

wizards, Entity Data Model Wizard, 28-30

***Working Effectively with Legacy Code* (Feathers), 9, 12, 509**

Write() method, 180

WriteLine() method, 180

X-Y-Z

XSS (cross-site scripting) attacks, 96

***xUnit Design Patterns: Refactoring Test Code* (Meszaros), 680**

YAGNI Principle (You Ain't Gonna Need It), 507

W

waterfall design, 507-508

web configuration files, configuring ASP.NET Web Forms files to support ASP.NET MVC, 416-422

Web Forms. See ASP.NET Web Forms

Web Platform Installer, launching, 2

Web Site Administration Tool, 365-366

web.config files, configuring ASP.NET Web Forms files to support ASP.NET MVC, 416-422

***What Is Software Engineering?* (Reeves), 509**

Where() method, 656