C# Training Workshop

Contents

[Introduction 2](#_Toc483069255)

[Acknowledgements 2](#_Toc483069256)

[Disclaimer 2](#_Toc483069257)

[Requirements 2](#_Toc483069258)

[Instructions 2](#_Toc483069259)

[Reminders 2](#_Toc483069260)

[Setup 4](#_Toc483069261)

[Install Visual Studio Express Community 2017 (free) 4](#_Toc483069262)

[Visual Studio Settings 4](#_Toc483069263)

[Create a Console Application 4](#_Toc483069264)

[Debugging 6](#_Toc483069265)

[Visual Studio Overview 7](#_Toc483069266)

[Comments 8](#_Toc483069267)

[Regions 8](#_Toc483069268)

[Variable Declarations 8](#_Toc483069269)

[Methods 9](#_Toc483069270)

[Unit Tests 11](#_Toc483069271)

[Create Sandbox class 13](#_Toc483069272)

[Operators 14](#_Toc483069273)

[Conditions 18](#_Toc483069274)

[Update Console Program with a Condition 21](#_Toc483069275)

[Exceptions 22](#_Toc483069276)

[While Loops 25](#_Toc483069277)

[For Loops 27](#_Toc483069278)

[Objects 31](#_Toc483069279)

[Generics and LINQ 36](#_Toc483069280)

[Enum 40](#_Toc483069281)

[Polymorphism 43](#_Toc483069282)

[Interfaces 49](#_Toc483069283)

[Update Console Program with the Repository call 52](#_Toc483069284)

[Resharper 53](#_Toc483069285)

[References 54](#_Toc483069286)

# Introduction

This C# workshop is intended for Beginners who have never programmed before. It introduces you to Visual Studio an integrated development environment (IDE) which is where you type your code. You could use notepad or other tools, but I prefer Visual Studio. The workshop will introduce you to GitHub which is a repository for your source code. I also use Team Foundation Services (TFS) and Team Services (TS). However, GitHub is more common for open source projects like this workshop. This workshop heavily promotes unit test adhering to Test Driven Design (TDD). Introductory discussions on design patterns including TDD and repositories are brought up, though nothing in-depth.

Once you reach the object-oriented programming (OOP) descriptions, don't worry if you don't get it right away. Continue with the workshop and slowly you'll come to put the pieces together. This section is by far the hardest concept for new developers to grasp, so don't despair. There are plenty of WIKI articles on the web that can further assist you. The key point for this workshop is to keep doing the workshop as doing it helps you get to the AHA moment where everything comes together.

If for any reason the installation instruction or extensions don't work as expected, please realize that these tools change very frequently. A quick search on the web will usually help you find what you need. Always try the StackOverflow articles first as they are usually more relevant and on point.

## Last Updated: 05/20/2017

## Acknowledgements

* This workshop was developed by Andrew Hinkle and published by <http://www.danylkoweb.com>
* Social Media
  + Website: <http://www.penblade.com>
  + LinkedIn: <https://www.linkedin.com/in/penblade/>
  + Twitter @AndrewHinkle5

## Disclaimer

* The author and publisher of this workshop hold no liability to any issues that may occur during your use of this material or recommended resources. As an example if you happen to install Visual Studio and your PC catches fire and burns your house down. Not liable. You decide to hack the CIA with the great font of information learned within this resource. Impressive. This is a beginner's class, so I didn't teach you that, but that's pretty cool. Also, you better be a white hat hacker or you're going to jail. Not liable for that either in case you were wondering.

## Requirements

* Estimated length of workshop: 20 hours
* Visual Studio 2017 Community
  + Given this workshop is targeted to Beginners the course is based on the free version accessible to everyone
  + Basic installation instructions are included in the workshop

## Instructions

* Most of this workshop will use the first level of bullets as a title or description of what we are doing followed by the steps to perform the action with notes in italics
* Actual code is provided to check your work against
* This document is intended to be used as a tool by an instructor working with those interested in learning to program in C#. As such there are some gaps where there is assumed knowledge or a mentor ready to assist. For those going solo, Google is your friend.

## Reminders

* Remember to KISS (Keep It Simple Stupid) your apps and don't over architect your solutions
* Run your tests after every change and verify that the unit tests succeed or fail as expected
* C# is Case Sensitive
* Code should be typed as is to avoid errors
  + Example: If the namespaces do not match, then additional using statements to the other namespaces need to be added, or better type as is

## GitHub

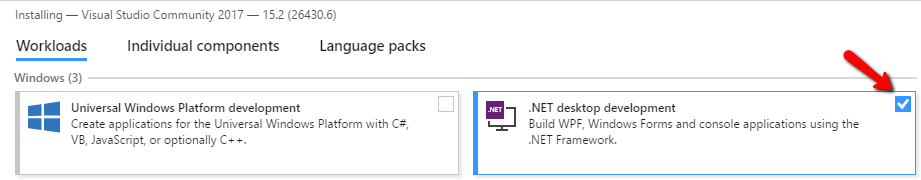
* This workshop is available on GitHub for your review. If your code fails to compile or you don't get your expected results, then compare your code to the repository.
* <https://github.com/penblade/Training.CSharpWorkshop>

# Setup

## Install Visual Studio Express Community 2017 (free)

*Troubleshooting installation issues is outside the scope of this workshop. The following are basic instructions subject to change.*

1. <https://www.visualstudio.com/vs/visual-studio-express/>
2. Click the Download Community 2017 link
3. Sign up for notifications or just exit
4. Run the downloaded executable
5. Follow the on-screen instructions until the Workloads tab appears
6. On the Workloads tab check ".NET desktop development"



1. Click Install
2. If prompted, click the Launch button

## Visual Studio Settings

*If one of these options is already checked, then you are all good and should move on to the next step.*

1. Enable line numbers
   1. Click Tools > Options > Text Editor > All Languages > Check Line numbers > Click OK
2. Track Active Item in Solution Explorer
   1. Click Tools > Options > Projects and Solutions > Check Track Active Item in Solution Explorer > Click OK

## Visual Studio Extensions

*Common extensions include Unit Test Generator, Web Essentials, GhostDoc, and Power Tools.*

1. Install Unit Test Generator
2. Click Tools > Extensions and Updates > Online (left sidebar) > Search (top right) Unit Test Generator > Click Download > Install > Click restart now (bottom)

# Create a Console Application

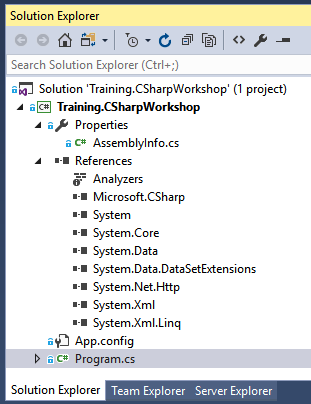
*Goal: Create a simple console app and add some simple logic to the app.*

1. Create a new C# Console application
   1. Click File > New Project
   2. Select Templates (left sidebar) > Visual C# > Windows Classic Desktop > Console App (.NET Framework)
      1. Name: Training.CSharpWorkshop
      2. Location: C:\Projects\Training
      3. Solution name: Training.CSharpWorkshop
      4. Check Create directory for solution
   3. Click OK
2. Read input and write Hello {name} to the console
   1. Open Program.cs (right sidebar)
   2. Update the Main method with the following code (leave the usings alone):



# Solution Explorer Overview

1. A Solution is a container that references projects and files associated with an application
2. A Project references all of the files required to run your application
   1. Applications may consist of multiple projects
3. A Unit Test Project (added later) tests a Project
4. The Properties > AssemblyInfo file describes your Project such as title, name, and version
5. References are a list of Projects that are part of the .NET Framework, NuGet packages, and projects in your solution that your Project requires in order to perform its function
6. App.config stores your configuration settings
   1. The Project that serves as the running application, in this case the Console Project "Training.CSharpWorkshop" must have all configuration settings required by any projects it references in order to run
   2. In the same regard the Unit Test project must also have any configuration settings required to properly test a project
7. Program.cs and any other files you add contain the logic to run your application



# Debugging

*Learn to add a breakpoint and debug. Add a breakpoint to the console application and practice the basic debug steps.*

1. Adding Breakpoints (shortcut: F9)
   1. Click the gray bar to the left of the command you would like to stop at while debugging so you may investigate the variable values and how commands are processed
      1. The breakpoint is noted by the red circle and the line highlighted red
2. Debug
   1. Add breakpoints
   2. Run the application in Debug mode by clicking Debug > Start Debugging (shortcut: F5)
   3. Process the current line and move to the next
      1. Click Debug > Step Over (shortcut: F10)
   4. Go to the method called on the current line and if you can't Step Over
      1. Click Debug > Step Into (shortcut: F11)
      2. If you call a method you didn't write you can't step into the method without proper debug files
   5. Continue processing until the next breakpoint is reached or to continue until the application is finished
      1. Click Debug > Continue (shortcut: F5)
3. Practice
   1. Add a breakpoint on the first Console.WriteLine and debug to that step.
      1. Step Over and Step Into will perform the same action for now, but later you'll see Step Into in action
   2. While debugging you can hover over parts of your code to get information such as the variables stored in variables
4. There are plenty of other debugging features available to use such as Add Watch that you can further research on your own

# Comments

1. Single-line



1. Multi-line



# Regions

1. Regions
   1. Sections of code that can be expanded and collapsed as a group



# Variable Declarations

1. Primitive types



1. Implicit types
   1. var (strongly typed, but the compiler determines the type)



# Methods

*Methods perform a specific task. Given certain parameters your code processes the information and returns the results. Methods don't have to use parameters and don't have to return results, but they should do something useful. There are additional features available for methods such as parameters can use "out" and "ref" and other interesting and useful features that are not covered by this workshop.*

*Methods can call other methods. Keep methods simple and only perform a specific task. Some methods may simply be a checklist that calls other methods to separate concerns. If a task has multiple steps, then break up the steps into additional methods and have your task method call the other step methods.*

1. Method signatures
   1. Parameters
      1. The values passed into a method
   2. Return Types
      1. The values returned from a method
      2. Use void when there is no value to return
   3. Scope
      1. Pass by reference
         1. If a value is passed in by reference the value is also changed outside the method
         2. Objects are passed by reference
      2. Pass by value
         1. If a value is passed in by value the value is not changed outside the method
         2. Primitive types such as string, int, float, etc are passed by value
2. Refactor the code in Main to a new public method named Process  
   *Main should be simple and clean*
   1. Create a new empty public method call Process
   2. Cut the contents of the Main method
   3. Paste into the Process method
   4. Call the Process method in the Main method
   5. Run Debug to verify the console app runs the same



1. Refactor the Role line to a method because we will be adding logic later
   1. Create a new public static method called GetRoleMessage that returns the string "Role: Admin"
   2. Replace the string in the Process method with a call to the GetRoleMessage method



# Unit Tests

*Unit tests verify your Project performs as expected. If you run your suite of tests after any change, you should find that your tests all pass. If they don't, then you know you introduced a bug and should resolve it. Better to find out sooner than later.*

1. Create a test project with all required references to the project we are testing
   1. Right-click the solution > Add > New Project > Expand Visual C# and select Test > Click Unit Test Project
   2. Name: Training.CSharpWorkshop.Test
   3. Location: C:\Projects\Training.CSharpWorkshop\Training.CSharpWorkshop\Training.CSharpWorkshop.Test
   4. Click OK
   5. Rename UnitTest1.cs to ProgramTests.cs
   6. Click Yes to rename the class
   7. Right-Click the test project References folder > Add Reference > Expand Solution > Select Projects > Check the project > Click OK
2. You should only test public methods
3. Private and protected methods can only be tested with a few unit test frameworks
4. Test classes will have the [TestClass] attribute
5. Test methods will have the [TestMethod] attribute
6. Test methods with the [Ignore] attribute will not be run
   1. While convenient at times until a reason for failure is determined, unit tests should always pass before checking in with few exceptions and even then should be resolved ASAP
7. Understanding the triple A's of unit tests
8. Arrange
   * 1. Create variables required to test a method
9. Act
10. Call the method we are testing saving the results to a variable named actual
11. Assert
12. Create a variable named "expected" representing the expected results
13. Call an Assert method for validating the expected result against the actual
14. Rename TestMethod1 to ProgramTest
15. Add the [Ignore] attribute to ProgramTest to show how tests can be ignored
16. Add our first unit test method GetRoleMessageTest



1. Running Tests
2. Run all tests
3. Click Test > Run > All Tests
4. Debug all tests
5. Add breakpoints as necessary
6. Click Test > Debug > All Tests
7. Debug all test methods in a test class
8. Right-click a test class > Run Tests (shortcut: Ctrl-R, Ctrl-T)
9. Debug a test method
10. Right-click a test class > Run Tests (shortcut: Ctrl-R, Ctrl-T)

# Create Sandbox class

*While running this workshop you may find that you are inspired to try something out. Let's create a Sandbox class so you can do just that without worrying about messing up the workshop.*

1. Create a test file for learning techniques on your own
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Sandbox.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



# Operators

1. Create Operators test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Operators.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Comparers



1. And



1. Or



1. Math



1. Math with Assignments



1. Incrementers/Decrementers



1. Conditional



1. Null Coalescing



# Conditions

1. Create Conditions test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Conditions.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. If



1. If/else



1. If/else if/else



1. Switch/case, break



1. Switch - Combined Cases



# Update Console Program with a Condition

1. Open Training.CSharpWorkshop.Program.cs
2. Update GetRoleMessage



1. String.Format is a common .NET method for concatenating strings
   1. {#} represents the elements in the parameters listed to the right of the string
   2. {#:\*} the star represents the format of the data such as currency and zero-padding



1. Open Training.CSharpWorkshop.Tests.ProgramTests
2. Rename GetRoleMessageTest to GetRoleMessageForAdminTest so it is more specific
3. Add unit test for Guest role



# Exceptions

1. Create Exceptions test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Exceptions.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Throw Exception



1. Try/catch



1. Try/catch/finally



# While Loops

1. Create WhileLoops test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to WhileLoops.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. While



1. While, continue/break



# For Loops

1. Create ForLoops test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to ForLoops.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. For, continue/break



1. Foreach, continue/break



# Objects

*Objects are the biggest hurdle for most people to overcome. While this workshop tries to explain the how objects work, don't worry if you don't get it right away. Continue with the workshop and if you still don't quite understand what they are, there are plenty of articles on the web that may assist you getting over that hurdle.*

*Goals: Create a RepositoryTest test class for unit tests. Create a User class to store an Id and Name as properties. Create a Repository to return an instance of a User object given a user name.*

1. Create RepositoryTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to RepositoryTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Create a new class named User
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to User.cs
   2. Open User.cs and add public before "class User"



1. An object is a container of properties and/or methods
2. Example: Every product in a grocery store has properties such as price and weight
3. Example: Every cash register has properties to track the list of products scanned and methods for totaling the price of each product in that list
4. Properties
   1. Properties act as if they are just public variables



* 1. With the benefit of maintaining a private variable that can be changed as needed without changing actual usage of the property



1. Create a new class named Repository
2. Right-click Training.CSharpWorkshop > Add > Class > Change name to Repository.cs
3. Open Repository.cs and add public before "class Repository"



1. Create a new instance of a list of users in the constructor



1. Repository is a design pattern used to manage data retrieval whether it comes from a database, files, or other means
2. Constructors
3. Constructors are used to setup an object and can take in parameters like a method
4. Class Security
5. Public (Common)
   * 1. Can be accessed by all
6. Private (Common)
7. Can only be accessed within this class
8. Protected
9. Can only be accessed within this class and any class that inherits it
10. Internal
11. Can be accessed by all as long as the class is in the same project (assembly)
12. New is used to create a separate instance of an object that follows a specific template
    1. In the case of User, I can create a new instance of a User object for Andrew and another separate instance for Dave; They are of the same type of object, but do not use the same instance
13. Static is used when you want to always use the same version every time without having to new up an instance at all
    1. String.Format is an example where you don't want to create a new instance of a String object just to format a string
14. Constants are set at declaration and never changed
15. Readonly are set by the constructor only
16. Add a method to GetUserByName



1. Create unit tests for the method GetUserByName in the RepositoryTest test class covering a user that should and shouldn't be in the list



# Generics and LINQ

*Developers inevitably want to find quicker and easier ways to accomplish a goal. Generics and LINQ are an example of some features that help us perform tasks in a universal way that can be reused. The following a short overview of some of the capabilities these two features offer.*

1. Create GenericsTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to GenericsTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Object Initializers
   1. Object Initializers allow you to create a new instance of an object or collection as if you defined the object and set the variables immediately after creating the new instance.



1. You can take this example one step further by doing a collection initializer



1. Generic Lists
   1. List is a generic class in the .NET framework that lets you define the type of objects the list will contain with standard CRUD (Create, Read, Update, Delete)
      1. Common methods of List:
         1. Add
         2. Insert
         3. Remove
         4. RemoveAt
         5. Count
         6. Clear
         7. Index
         8. Sort
2. LINQ (Language-Integrated Query)
   1. Lists can be filtered with LINQ and can follow syntax that use object chaining or a more SQL like format that starts with the From and ends with the Select
      1. Common methods of LINQ on lists
         1. Any
         2. Contains
         3. Exists
         4. Where
         5. Single
         6. SingleOrDefault
         7. First
         8. FirstOrDefault
         9. Find
3. In Repository, create a Find method to illustrate LINQ



1. Add Find Unit Tests which are the same as the GetUserByName Unit Tests except the different method call



1. Create an Insert method in the repository



1. Create an Insert Unit Test



1. Create a Delete method in the repository



1. Create a Delete Unit Test



# Enum

1. Create EnumTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to EnumTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Enum
   1. Enums are classes that represent related static content
      1. Very useful to avoid hard coded strings all over your code
2. Create a new class named RoleEnum
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to RoleEnum.cs



1. Add RoleEnum as Role to the User class



1. Add methods to verify against the user's role if the user can add, delete, or find users



1. Set the role in GetUsers



1. Add Unit Tests for Admin and Guest user permissions



# Polymorphism

*Object-Oriented Programming (OOP) at its best is the capability to reuse code and keep it related. In the following steps we have an AdminUser and a GuestUser. Both are a type of User and share those properties and methods. Both also have their own unique functionality, but they are still a User giving you more flexibility.*

1. Create PolymorphismTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to PolymorphismTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Change User class to be an abstract class



1. Abstract Classes
   1. Cannot be instantiated
   2. Child classes can use methods and properties of parent (grand-parent, etc) classes as long as they are public or protected
      1. Example: Can be used to define the roles of the User so there is never an opportunity where the role is never defined
2. Create a new class named AdminUser that inherits User
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to AdminUser.cs



1. Create a new class named GuestUser that inherits User
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to GuestUser.cs



1. Update EnumTest test class method GuestUserPermissions CanInsert assertion to true



1. Inheritance
   1. A class that inherits another class is considered a child class of that parent class
   2. Two classes that share the same parent class are considered siblings
   3. Defining a method as virtual will allow a child class to override the method to perform different logic
   4. A sealed class cannot be extended (polymorphed)
2. Update Repository to use the new polymorphed objects AdminUser and GuestUser appropriately
   1. Refactor the declaration of user away from the first user and define it as User user;
   2. Change the first user to an AdminUser
   3. Change the second user to a GuestUser
   4. Remove the role definitions since they are now handled in the abstract User class



1. Update GenericsTest to create the user as a GuestUser since User is now an abstract and can no longer be instantiated



1. In PolymorphismTest, add unit tests for new User methods



1. Type casting
   1. Typeof
      1. Method will return the class type of the class specificed
   2. GetType
      1. Method will return the type of the object
   3. ({class name}) actual
      1. Attempts to cast the variable to that type and will throw an exception if it fails
   4. actual as {class name}
      1. Attempts to cast the variable to that type and will return null if it fails
   5. Example: Gain access to methods only available in a child object
   6. Example: Perform logic if the object is of a specific type



# Interfaces

*Interfaces are contracts that specify what an object can do.*

1. Create a new class named IUser and change it to an interface
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to IUser.cs



1. Change User class to implement the IUser interface



1. Change Repository to use IUser instead of User in all cases.



1. Interfaces
2. Cannot be instantiated
3. Interfaces are contracts that are implemented by a class so that every property and method (not variables) defined in the interface are also defined in the class
   1. Every property and method defined in the class DO NOT have to be defined in the interface
4. Only need applied to the lowest common class
   1. Typically that is the root parent object, but not always
5. Interfaces are crucial to properly unit test objects in isolation to avoid "Integration Tests" that are reliant upon databases, connections, and files
   1. Example: Dave is a specific user in the database that may be deleted when he leaves the company or his role changes from a Guest to an Admin

# Update Console Program with the Repository call

1. Open Training.CSharpWorkshop.Program.cs
2. Update GetRoleMessage



1. Open Training.CSharpWorkshop.ProgramTest.cs
2. Add the GetRoleMessageForNoneTest test method to test the None case
   1. The existing tests are fine, however, we now have a None case we need to test against



# ReSharper

ReSharper is an add-on that simplifies some complex tasks. The following is the short list of ReSharper features I use frequently.  I’ll demonstrate a few in the next lesson.

* Enhanced Intellisense
  + Better description of what is wrong with the syntax
  + Offers to refactor code to use other techniques such as for loops to LINQ
    - This feature is hit or miss as to whether you like, can read/maintain, however, it is a great learning tool to learn new techniques
      * I learned much of what I know about LINQ with this feature
* If an object is not referenced in a the project, but it is included in another project in the same solution, ReSharper will ask if you want to import it
  + When most students had the issue with the Program class not being referenced, ReSharper would have fixed that with a quick shortcut Alt-Enter
* Extract Method
  + Move highlighted code to a new method and add a call to that method
* Extract Interface
  + Interfaces are contracts that are very useful for unit tests
* Move files
  + Moves the files to the new location
  + Updates all namespaces
  + Updates all references to the file and objects therein throughout the entire solution
* Safe Delete
  + When you deprecate an object and you are ready to delete, this feature will review your solution to ensure there are no other references
    - Still be careful as it won’t detect references from other solutions that may use the DLL such as from a common library file used by multiple solutions
* Adjust Namespaces
  + Fixes namespaces to match your folder structure, much easier than doing it manually
* Menu option: Refactor
  + Gives you access to most of the features above
* Menu option: Run Unit Tests
  + Run Unit Tests option is available at the solution level, test project, test class, test method
* Much, much more!
  + These are just the most commonly used features though it supports most Refactoring techniques.

# References

Here’s a short list of recommended books that transformed me as a developer that I highly recommend any developer to read.

* [Top 10 books every .NET developer should own](http://www.danylkoweb.com/Blog/top-10-books-every-net-developer-should-own-QG) - This is a great list of books. I've personally read and recommend reading the following three books on that list in the following order.
  + [Refactoring: Improving the Design of Existing Code](http://www.amazon.com/gp/product/0201485672/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=0201485672&linkCode=as2&tag=danyl-20&linkId=JWATUXHVFPXJM5LU) By Martin Fowler, Kent Beck, John Brant, Williams Opdyke, and Don Roberts
  + [Design Patterns in C#](http://www.amazon.com/gp/product/0321126971/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=0321126971&linkCode=as2&tag=danyl-20&linkId=GLRAEO3PLQISPDUA) By Steven John Metsker
  + [Patterns of Enterprise Application Architecture](http://www.amazon.com/gp/product/0321127420/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=0321127420&linkCode=as2&tag=danyl-20&linkId=5K6QOZY7V4BE5UMK) By Martin Fowler