Test Driven Development in the .Net Framework

Sam Lee
sl20@txstate.edu

Mayur R. Mehta
mm07@txstate.edu

Jaymeen R. Shah
js62@txstate.edu
Department of CIS & QMST
Texas State University–San Marcos
San Marcos, Texas 78666, USA

Abstract

Test Driven Development (TDD) is powerful to create and maintain complex packaged software for enterprises. The .Net framework and the test-first approach provide synergy for developing high-quality database applications in the fast-changing business world. This paper presents a TDD approach that uses unit test cases in the Visual Studio .Net 2008. The data access classes are quickly built and modified using models in the ADO.Net Entity Framework. NUnit, free and open software, is employed for unit testing to manage the change of entity models.

Keywords: Test Driven Development, ADO.Net, Packaged Software, Database, Unit Test

1. INTRODUCTION

Packaged software is a powerful global industry: the largest firms in this industry – Microsoft, Oracle, IBM and others – are household names (Carmel & Sawyer, 1998). This industry is very competitive, which leads to intense time-to-market pressures and the need to always adapt software in progress to the new functionality of the latest releases of the competitors (Dube, 1998). To deal with the pressure, one of the most powerful tools is the method of Test Driven Development (TDD). With TDD, unit test cases are written for the implementation of any new functionality (Maximillien & Williams, 2003). The new functionality is not ready unless these new unit test cases and every other unit test case written for the existing code base are tested automatically and successfully.

Adding new features one at a time, programmers who write more tests tend to be more productive (Erdogmus et al., 2005). The TDD's advantages, which are especially significant to continuously grow the complexity of software systems, include that it (a) makes the changes virtually risk free such that the development team can flexibly change the behavior of one part of the system without risking side effects in other parts (Martin, 2007); (b) delivers software in smaller units that are less complex (Janzen & Saiedian, 2008); and (c) effectively captures requirements such that development tools can be integrated to continue to improve system quality (Crispin, 2006).

The Microsoft Visual Studio .Net is one of most popular tools to develop enterprise applications. Visual Basic facilitates easy crea-

tion of software components in the .Net environment. This paper discusses the techniques related to component (unit) tests using the Visual Studio .Net 2008 and the Visual Basic language.

2. UNIT TESTING IN .NET

Developing unit test cases is a key to TDD. NUnit, free and open software, is a unit-testing framework for all .Net languages. To start using NUnit, you must have Microsoft Visual Studio .Net installed in your computer. Then you download NUnit from its Web site (www.nunit.org) and follow the instructions to install it.

To start developing a demo application, we first create the project "UnitTestSimple" in Visual Studio .Net 2008 (see Figure 1). Figure 2 shows a simple class to be tested by NUnit. The class, created in the "UnitTestSimple" project, is defined by a method that calculates the distance to the origin for a point (X, Y). Test data should be gathered before writing the NUnit program to test the method. Table 1 shows the test data.

The unit test case (Figure 3), which is also created in the "UnitTestSimple" project, is written in a separate program to test the "Distance to Origin" method. A reference to the nunit.framework DLL file must be added to the project (see Figure 4).

Table 1. Test data for writing a unit test case.

X-axis	Y-axis	Distance to Origin (**Expected Value)
3.5	4.5	5.701
6.2	10.4	12.108

**Expected values are coded in the unit test programs

The basic syntax for developing a unit test class is described as follows.

a) You must import the Nunit.Framework name space in the unit test program.
b) A "Test Fixture" attribute must be applied to the unit test program.
c) A "Test" attribute must be applied to every method that is written to perform tests.
d) The Assert class is used to test whether the expected and actual values are the same.

If there is a code mistake in the "Distance To Origin" method, the actual value returned by the method to be tested is most likely wrong and not the same as the expected value. Thus, the mistake is detected by a method of the Assert class.

To run the unit test class in Figure 3, you must build the Visual Studio project and start the NUnit GUI application. Then, in the application, you open the "UnitTestSimple" project by navigating to its "bin\Release" subdirectory and select the UnitTestSimple.dll file (see Figure 5). After the project is opened, you click the Run button to test it. The "Distance To Origin" method passes the test if a green bar is displayed (see Figure 6); otherwise, a red bar is shown and there is a message to indicate that expected and actual values did not match.

3. UNIT TESTING FOR DATABASE APPLICATIONS

The Visual Studio .Net 2008 introduces the ADO.NET Entity Framework, which allows developers to write less data-access code, reduces maintenance, abstracts the structure of the data into a more business-friendly manner, and facilitates the persistence of data (Papa, 2007). A TDD approach to integrating NUnit and Entity Frameworks provide synergy for developing high-quality and easy-to-maintain database applications.

To demonstrate the TDD approach, we create a database connection in a database application project and follow the steps that are described in this section:

a) Object and Relational Mapping (ORM)
   i. Creating database tables.
   ii. Generating ADO.NET entity classes that are mapped to the tables.
b) Code Development
   i. Creating service classes that perform all of the data operations: create, read, update and delete.
   ii. Testing the methods of the service classes.

Object Relational Mapping

ORM is a technique for converting data models between relational databases and object-oriented programs. Programmers apply the

technique to easily make objects persistent to the database systems.

Tables 2, 3 and 4 show the tables of a simple shopping cart application: Cart, Cart Item and Product. The Cart ID and Product ID fields of the Cart Item table are foreign keys linking to the Cart and Product tables, respectively.

After creating the tables in a Microsoft SQL server, the entity model in Figure 7 is easily created using a template in Visual Studio .Net 2008. The template is shown in Figure 8. It is noteworthy that a library of Visual Basic classes is automatically generated for writing data-access code. The library includes the Cart, Cart Item and Product classes. The ADO.Net Entity Framework provides a container for all the class in the entity model.

Table 2. The Cart table

Cart ID (Primary Key)	Date Of Creation	Tax
1	8/11/2008 10:05:00 AM	3.00
2	8/11/2008 11:05:00 AM	1.50

Table 3. The Product table.

Product ID (Primary Key)	Product Name	List Price
1	Keyboard	20.50
2	Mouse	10.20
3	Monitor	120.80

Table 4. The Cart Item table.

Cart Item ID (Primary Key)	Sale Price	Qty	Cart ID	Product ID
1	20.5	2	1	1
2	120.5	1	2	3
3	10.2	1	1	2

Code Development

Figure 9 shows a service class that includes two methods: (a) to find a cart object by cart ID; and (b) to create a new cart by the ID and quantity of the first product in the cart. In the service class, CartDB is the container for all the entities in the model of Figure 7. Using the container, we write a From statement to find Cart objects from the entity model, a AddToCart statement to add a new object to the Cart entity, and a SaveChanges statement to make the changes of the entity model permanent in the database tables.

The unit test program in Figure 10 is written to test the service class. The FindCart method is tested to see whether a cart is returned with the expected tax stored in the database. The test program also checks whether the CreateCart method creates a cart and its cart item in database. When successful, the method returns a valid cart with a Cart ID greater than 1. Figure 11 shows the test results.

After finishing this project, we can see that the Entity Framework also facilitates the construction of a model driven architecture (MDA) for business application development. MDA is an initiative by the Object Management Group. It deals with the system complexity based on the principles of abstraction, reuse and patterns; and typically supports code generation to justify the time and resources invested in modeling activities. According to Conn & Forrester (2006), MDA uses three sets of models: platform independent models (PIMs), platform specific models (PSMs), and transformation models (TMs). PIMs capture domain-specific knowledge from both organizational environments as well as technical environments. These models are independent of the actual technologies needed to implement their functionality.

The diagram in Figure 7, which is very similar to a class model in Unified Modeling Language (UML), is a PIM. The .Net framework is a PSM that provides implementation structure and functionality. The ORM is a TM that is used to guide the process of transforming the PIM to create a PSM-based code library that is precise, formal and detailed for development.

4. CONCLUSION

This paper presents a TDD approach to develop database applications using the Micro-

soft Visual Studio .Net 2008. In this approach, the classes that access databases can be easily built and updated using the utilities of ADO.Net Entity Framework. NUnit provides tools to develop unit test cases to manage the change of entity models, as systems need consistent improvement.
It is clear that unit tests are still limited to be used in developing user interfaces. The future study will include the application of tiered architecture to improve the proposed TDD approach. It will reduce the amount of code written in user interfaces; thus, majority of the code in a business application can be unit-tested.

After all, it is intriguing to discover convergence of two development methodologies: TDD and MDA. The ideas of visual programming become more model-based and less intuitive than those driven by user interface design. The new development environment will lead to the creation of scalable software modules that can be fully and repeatedly tested.

5. REFERENCES

Carmel, E. & Sawyer, S. (1998). Packaged software development teams: what makes them different? *Information Technology & People*, 11 (1), 7-19.

Conn & Forrester (2006). Model Driven Architecture: A Research Review for Information Systems Educators Teaching Software Development. *Information Systems Education Journal*, 4 (43).

Crispin, L. (2006). Driving Software Quality: How Test-Driven Development Impacts Software Quality, *IEEE Software*, 23 (6), 70-71.

Dube, L. (1998). Teams in packaged software development: The Software Corp. experience, *Information Technology & People*, 11 (1), 36-61.

Erdogmus, H., Morisio, M. & Torchiano, M. (2005). On the Effectiveness of Test-first Approach to Programming, *IEEE Transactions on Software Engineering*, 31(3), 226-237.

Janzen, D.S. & Saiedian, H. (2008). Does Test-Driven Development Really Improve Software Design Quality? IEEE Software, 25 (2), 77-84.

Martin, R. C. (2007) Professionalism and Test-Driven Development, *IEEE Software*, 24 (3), 32-36.

Maximillien, M. & Williams, L. (2003). Assessing Test-Driven Development at IBM, *International Conference on Software Engineering*, Portland, OR, 564-569.

Papa, J. (July 2007). ADO.NET Data Points: ADO.NET Entity Framework Overview, *MSDN Magazine*, Retrieved August 5, 2008, from http://msdn.microsoft.com/en-us/magazine/cc163399.aspx

Appendix

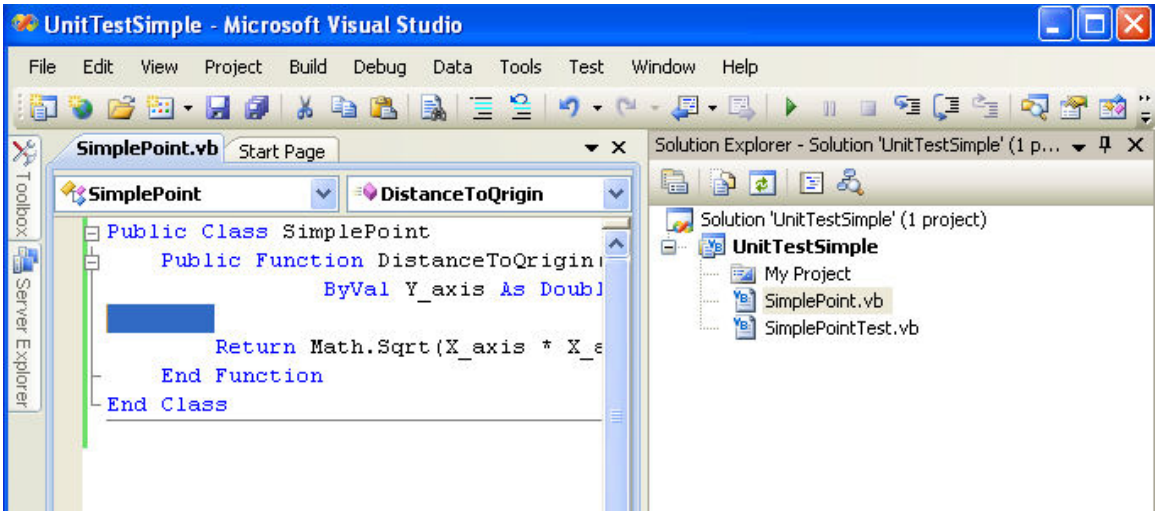

Figure 1. A demo project in Visual Studio .Net 2008

```
Public Class SimplePoint
    Public Function DistanceToOrigin(ByVal X_axis As Double, _
                ByVal Y_axis As Double) As Double

        Return Math.Sqrt(X_axis * X_axis + Y_axis * Y_axis)
    End Function
End Class
```

Figure 2. A simple class to be tested by NUnit.

```
Imports Nunit.FrameWork
<TestFixture()> _
Public Class SimplePointTest
    Dim point As New SimplePoint
    Dim actual As Double
    Dim expected As Double
    <Test()> _
    Public Sub DistanceToQrigin()
        'Test the first point (X, Y) = (3.5, 4.5)
        actual = Point.DistanceToQrigin(3.5, 4.5)
        expected = 5.701
        Assert.AreEqual(expected, actual, 0.001)
        'Test the second point (X, Y) = (6.2, 10.4)
        actual = Point.DistanceToQrigin(6.2, 10.4)
        expected = 12.108
        Assert.AreEqual(expected, actual, 0.001)
    End Sub
End Class
```

Figure 3. A unit test class to test the class method in Figure 2.

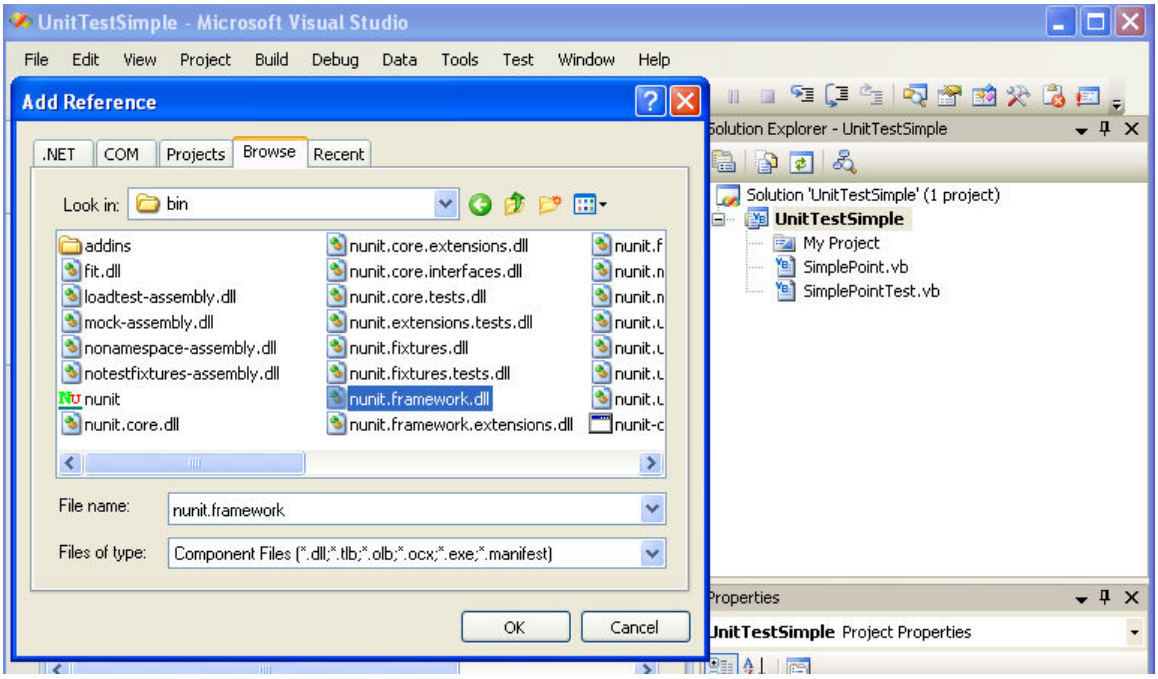

Figure 4. Adding the reference of the nunit.framework DLL file to the demo project

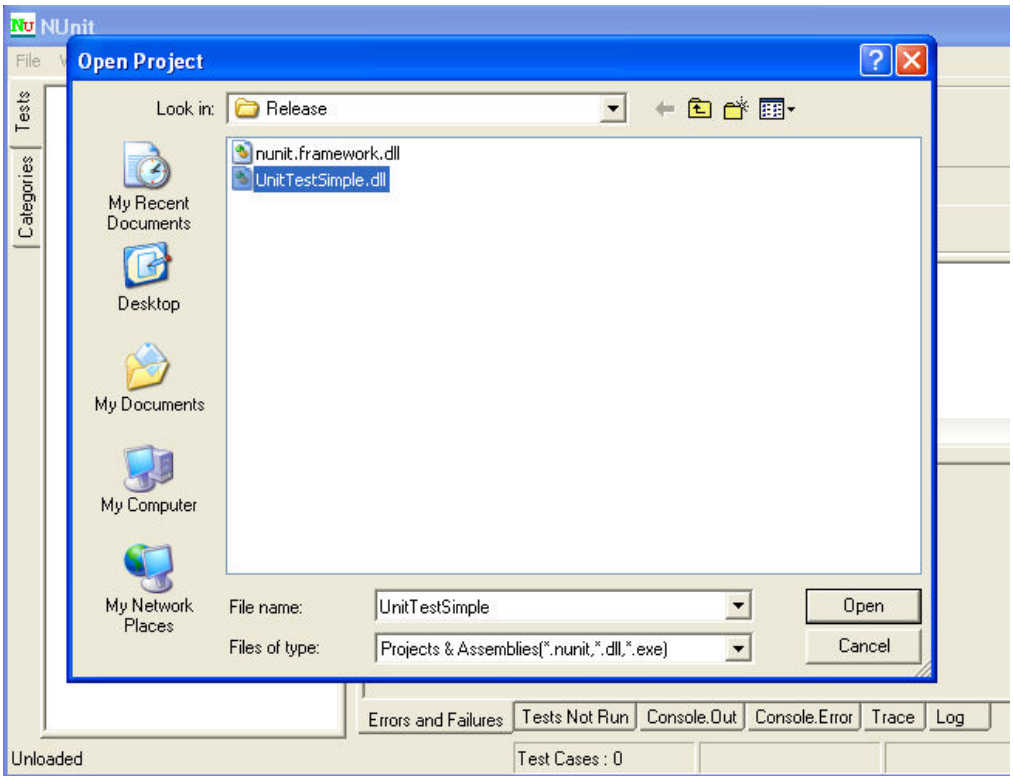

Figure 5. Opening the demo project in the NUnit GUI application.

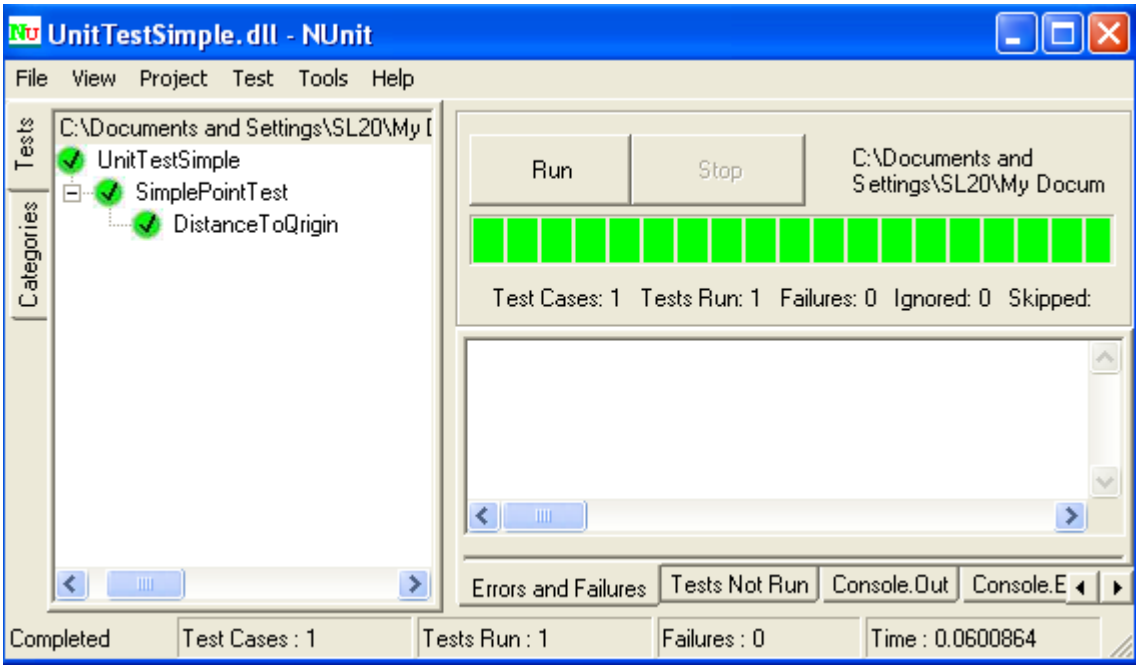

Figure 6. Running the unit test class in Figure 3 using the NUnit GUI application.

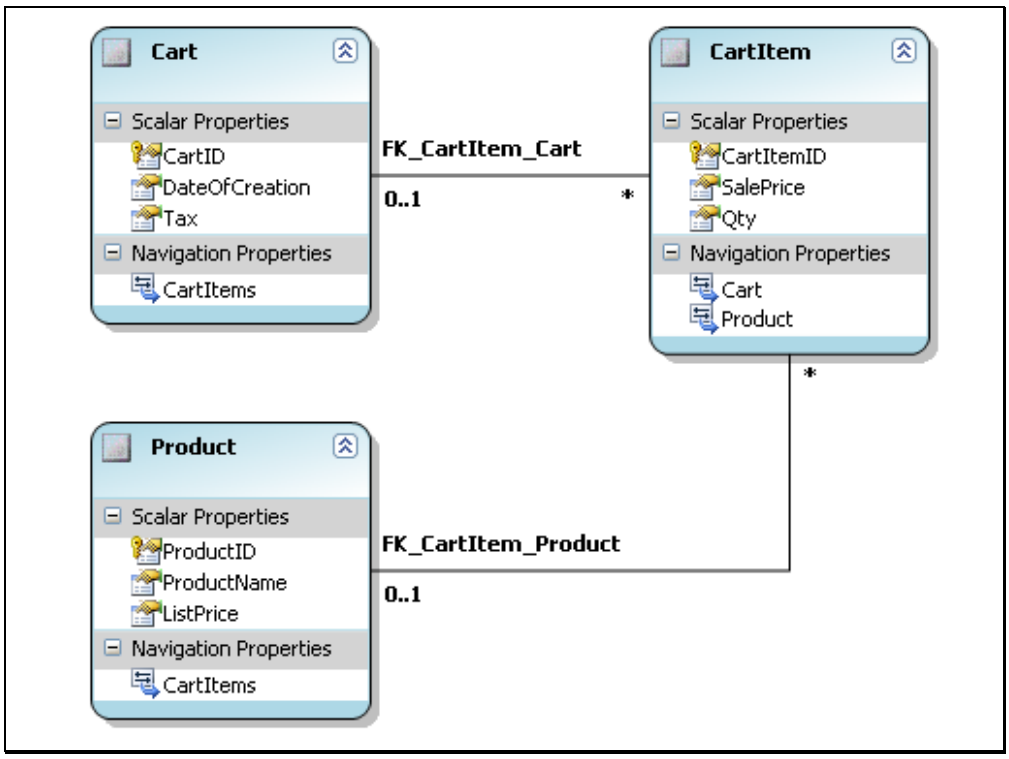

Figure 7. An entity model generated in the Visual Studio .Net 2008.

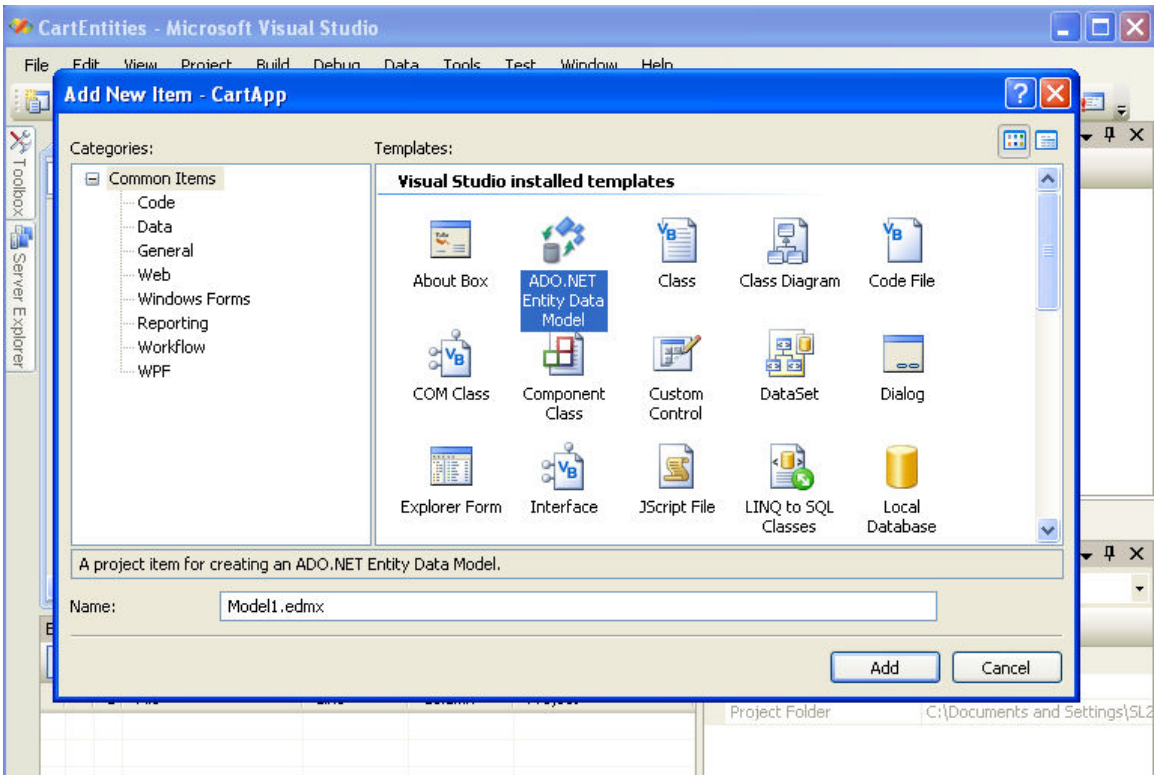

Figure 8. The template to add an entity model into a project of Visual Studio .Net 2008.

```
Imports CartApp.CartModel
Public Class CartService

    'Find a cart by a cart ID
    Public Function FindCart(ByVal cartID As Integer) As Cart
        Dim cart As Cart = Nothing
        'Using the object service of Entity Framework
        Using db As New CartDB
            cart = (From c In db.Cart _
                    Where c.CartID = cartID).First
        End Using
        Return cart
    End Function

    'Create a new cart to contain a product
    Public Function CreateCart(ByVal argProductID As Integer, _
                               ByVal argQty As Integer) As Cart
        'Decalre objects
        Dim cart As Cart = Nothing
        Dim cartItem As CartItem = Nothing
        Dim product As Product = Nothing
        'Using the object service of Entity Framework
        Using db As New CartDB

            'Find the product by ID
            product = (From p In db.Product _
                    Where p.ProductID = argProductID).First
            'Exit if the product ID is not valid
            If product Is Nothing Then Return Nothing

            'Make a new cart item
            cartItem = New CartItem
            cartItem.Qty = argQty
            cartItem.SalePrice = product.ListPrice
            cartItem.Product = product
            'Make a new cart
            cart = New Cart
            cart.CartItems.Add(cartItem)
            cart.DateOfCreation = Now
            cart.Tax = 0.0
            'Persist the cart to a database
            db.AddToCart(cart)
            db.SaveChanges()

        End Using

        'return a new cart successfully created in database
        Return Cart
    End Function
End Class
```

Figure 9. A service class for data operations.

```
Imports NUnit.Framework
Imports CartApp.CartModel
<TestFixture()> _
Public Class CartUnitTest
    Dim service As New CartService

    'This test is passed if a cart
    ' is returned with the expected
    ' tax
    <Test()> _
    Public Sub testFindCart()
        Dim c As Cart
        'Test first cart (ID = 1)
        c = service.FindCart(1)
        Assert.AreEqual(3.0, c.Tax)
        'test second cart (ID = 2)
        c = service.FindCart(2)
        Assert.AreEqual(1.5, c.Tax)
    End Sub

    'This test is passed if there is no
    ' exception and a positive cart ID
    ' is generated after this method is executed.
    <Test()> _
    Public Sub testCreateCart()
        'product ID = 2 & qty =3
        Dim c As Cart = service.CreateCart(2, 3)
        Assert.Less(1, c.CartID)
    End Sub

End Class
```

Figure 10. A unit test class to test the class methods in Figure 9.

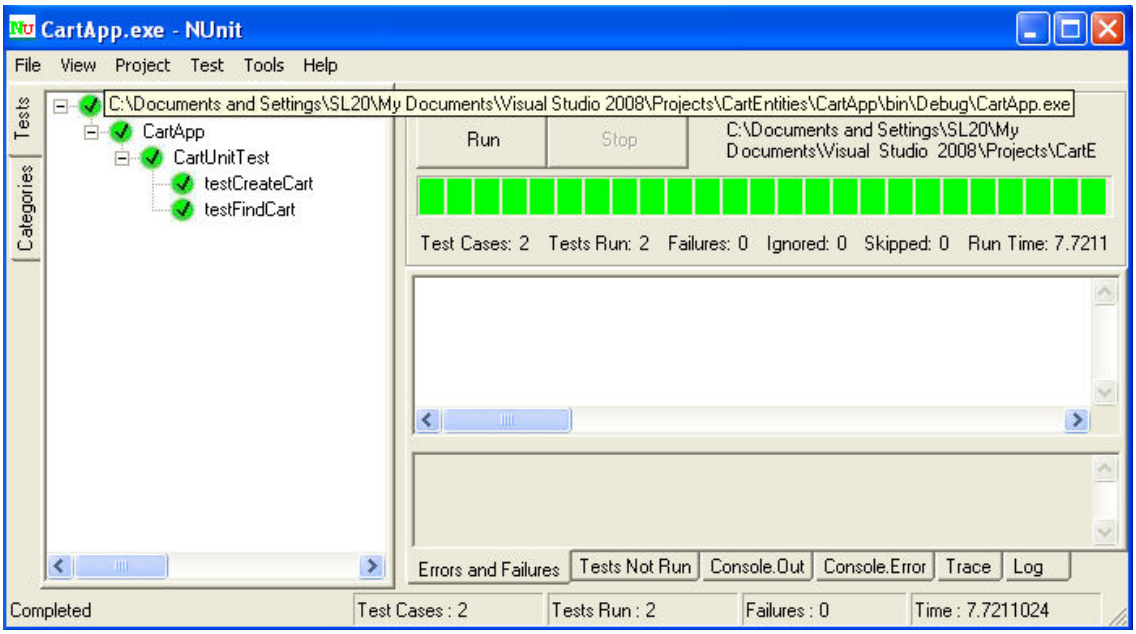

Figure 11. Running the unit test class in Figure 10 using the NUnit GUI application.