Comparing .NET with Java

Barry Cornelius
Computing Services, University of Oxford
Date: 29th October 2002; first created: 12th January 2002
http://users.ox.ac.uk/~barry/papers/
mailto:barry.cornelius@oucs.ox.ac.uk

1 Introduction

Microsoft have hit back: having fallen out with Sun over Java, they have now developed a rival product. Whereas the Java technology has produced a single language that is portable across many platforms, Microsoft's .NET Framework provides a number of languages that interoperate, initially only for most varieties of Microsoft Windows.

Microsoft also have an IDE called Visual Studio.NET. This product not only makes it easy to produce code for standalone programs (such as console applications and windows forms applications) but also makes it easy to produce code that can be executed by IIS (Microsoft's web server software). The latter possibility not only allows the creation of dynamically generated WWW pages, but also the ability to offer *web services*, i.e., to provide methods that can be called by external clients.

This paper:

- considers the key aspects of the .NET Framework;
- compares .NET languages (such as C# and Visual Basic.NET) with Java;
- demonstrates the use of these languages to produce standalone programs, web form applications and web services.

2 The .NET Framework

2.1 Overview of the .NET Framework

Put simply, the .NET Framework consists of three aspects:

1. the Common Language Runtime (CLR);
2. a comprehensive set of class libraries;
3. class libraries associated with particular kinds of applications:
   (a) console applications;
   (b) windows forms applications;
   (c) web form applications;
   (d) web services.

2.2 The Common Language Runtime

In the past, compiler writers have put code to support the execution of programs into a runtime system. Instead of providing a different runtime system for each programming language, the .NET Framework provides a runtime system that is used by all of the languages that are targetted at the .NET Framework. This is called the *Common Language Runtime* (or *CLR*). Code that targets the CLR is called *managed code*.

Microsoft are providing .NET compilers for several programming languages: Managed C++, Visual Basic.NET, JScript and C#. In addition, other people/companies are producing .NET compilers for other languages including COBOL, Eiffel, Fortran, Haskell, ML, Perl, Python, Scheme and Smalltalk.

A .NET compiler writer can rely on the CLR for a large number of tasks, including:

- creating new types;
- creating and initializing of objects;
- tracking references to objects and providing garbage collection;
- handling the calling of methods (including virtual methods);
- managing the access to array elements;
- providing support for exceptions and exception handling.

All of the .NET languages have compilers that generate instructions coded in an intermediate language called *MSIL* (or *IL*). A file containing MSIL instructions can be run on any platform so long as the operating system for that platform hosts the CLR engine. Currently, a CLR engine is available for Windows XP, Windows 2000, Windows NT 4.0, Windows 98 and Windows Me.

There is a project called *Mono* that is building an open-source implementation of the .NET Framework, and Microsoft are working on an implementation for FreeBSD.

2.3 The Common Type System

Besides providing the functionality normally expected from a runtime system, the CLR also defines a *Common Type System* (*CTS*). Thai and Lam say: 'The CLR provides full support for object-oriented concepts (such as encapsulation, inheritance, and polymorphism) and class features (such as methods, fields, static members, visibility, accessibility, nested types, and so forth). In addition, the CLR supports new features that are nonexistent in many traditional object-oriented programming languages, including properties, indexers and events.'

For efficiency reasons, the CTS has value types as well as reference types. So an `int` or a value of some struct type will be stored on the stack or inline, whereas an instance of some class type will be stored on the heap (and pointed to by some variable). However, any value (that is of some value type) can automatically be wrapped into an object by a process known as *boxing* (which will be considered later).

The aspects of the CTS which must be supported by all .NET languages is defined by the *Common Language Specification* (*CLS*). The CLS says that each language must provide value types (primitive types, struct types, enumerations) and reference types (class types, interface types, array types, delegate types).

2.4 The primitive types

As well as providing a type system that is common to all .NET languages, the CLR also provides a set of primitives types that is common to all .NET languages. The .NET primitive types include:

	size	*C#*	*Visual Basic.NET*
`System.Boolean`	8	`bool`	`Boolean`
`System.Byte`	8	`byte`	`Byte`
`System.Int16`	16	`short`	`Short`
`System.Int32`	32	`int`	`Integer`
`System.Int64`	64	`long`	`Long`
`System.Float`	32	`float`	`Single`
`System.Double`	64	`double`	`Double`
`System.Char`	16	`char`	`Char`
`System.Decimal`	128	`decimal`	`Decimal`

2.5 Language interoperability

Given that the compilers for each .NET language generate the same intermediate language, use the same runtime, build the same kind of types and use the same primitive types, the .NET Framework makes it easy to build programs where the code is written in different .NET languages.

For example, a Visual Basic.NET programmer can create a class that derives from a C# class and overrides some of its virtual methods; or a C# programmer can handle an exception thrown by a method being applied to an object of an Eiffel class; and so on.

2.6 Tool support

Because there is a CLR, debuggers can support programs where the code has been written in different .NET languages, and IDEs (such as Visual Studio.NET) can use the CLR to provide information to the programmer. For example, if you have declared some variable:

```
ArrayList tArrayList;
```

then, when you start to type the code to apply a method to `tArrayList`:

```
tArrayList.
```

as soon as you type the dot, the IDE can provide you with a pop-up window displaying a list of methods that can be applied to `tArrayList`.

2.7 Deployment

One of the problems with Windows applications is that they can be difficult to install. Besides providing the files, the application may want to change the registry or provide shortcuts. It is also difficult and sometimes impossible to uninstall applications. With .NET, all the code and any information needed to run an application are provided in a collection of files. In order to install an application, you just need to create a directory containing these files, and removing this directory uninstalls the application.

2.8 Versioning

One of the main problems with installing a Windows application is that the installation may overwrite a DLL used by some other application, and overwriting it causes the other application no longer to work. This is because the two applications require different versions of the DLL file. Because, in .NET, DLLs can be signed with a public key and a version number, it is possible for the cache of DLLs to have more than one DLL with the same name.

3 Moving to the .NET Framework

3.1 Introducing C# and Visual Basic.NET

As mentioned earlier, there are a number of languages that can be used with the .NET Framework. In designing their new language C#, Microsoft have produced a language that is similar to the Java programming language borrowing some ideas from C++ and some from Visual J++. So a Java programmer will have little difficulty in adopting C#.

In order to transform Visual Basic into a .NET language, many aspects of Visual Basic have been changed or removed, and many new features have been added. The result, called *Visual Basic.NET*, is a language that provides full support for object-oriented programming.

3.2 Similarities with Java

Here is a list of some of the ways in which the languages C# and Visual Basic.NET are similar to the Java programming language:

- there are two main kinds of types: value types and reference types;
- the value types include primitive types (e.g., char, int, double) that have sizes that are determined by the definition of the language;
- the reference types include class types, interface types and array types;
- classes are organized into a class hierarchy that has a root class (called `System.Object`) and that supports single inheritance;
- a class may implement one or more interfaces;
- an abstract class can be used to represent the common aspects of a set of classes;
- a class can provide one or more constructors that are used to create objects;
- overriding is possible, i.e., a method of a class may have the same name as a method of its superclass;
- you can control the visibility of a class and of the members of the class making them private or public or only accessible to subclasses, and so on;
- there are no global functions or global variables;
- classes derived from `System.Exception` can be used to represent exceptions which can be thrown using a throws statement and handled using a try statement (which has one or more catch clauses and possibly a finally clause).

3.3 Differences from Java

We now look at some of the ways in which C# and Visual Basic.NET are different from Java.

3.3.1 Types

As mentioned earlier, the .NET languages provide two kinds of types: *value types* and *reference types*.

There are two kinds of value types: *struct types* and *enumeration types*.

The declaration of a struct type is similar to that of a class type. For example, it can declare fields, constructors and methods. However, there is a big difference: a variable that is of a class type points to an object of that class whereas a variable that is of a struct type stores the value itself. So assignment between two struct variables causes copying of a struct value to take place rather than making a variable point to the same object as some other variable.

In C# and Visual Basic.NET, each of the keywords used for the primitive types (e.g., `int` in C# and `Integer` in Visual Basic.NET) is just an alias for a struct type (e.g., `System.Int32`).

Both C# and Visual Basic.NET allow the declaration of enumeration types. These are also value types.

As in Java, a reference type is a class type, an interface type or an array type. A .NET language has one other kind of reference type: the *delegate type* (which will be considered later).

As well as value types and reference types, C# also has *pointer types*. These can only be used in code marked as `unsafe`. So, we will ignore pointer types.

As mentioned earlier, in some contexts, *boxing* occurs: this means that an object is automatically created from a value (that is of some value type). So a value can be used in contexts where an object is expected:

```
tArrayList.Add(27);
```

The reverse process (called *unboxing*) is also possible:

```
int tValue = (int) tArrayList[0];
```

Automatic boxing and unboxing are not present in Java. Instead, in Java, you have to do these tasks explicitly using the wrapper types.

3.3.2 Methods

In Java, all parameters are *value parameters*. A value parameter acts like a local variable of the method whose initial value is the value of the argument used in the call. C# has value parameters, *ref parameters* and *out parameters*, whereas Visual Basic.NET has `ByVal` and `ByRef` parameters. If a method has a ref/out/ByRef parameter, then that parameter represents the same variable as the variable given as the argument. In C#, a *params parameter* may appear as the last parameter of a parameter list: it permits a variable number of arguments.

In C#, by default, methods are non-virtual. If you want a subclass to override a method declared in a superclass, both method declarations must be flagged to indicate this (using the `virtual` keyword in the superclass and the `override` keyword in the subclass). The method declaration in the subclass can instead use the `new` keyword if instead you want the subclass to hide the method declared in the superclass.

Although it uses different keywords (`Overridable`, `Overrides` and `Shadows`), the same possibilities for overriding and shadowing methods of a superclass exist in Visual Basic.NET.

3.3.3 Statements

The statements of C# are very similar to those of Java. However, C# also has a `foreach` statement which permits the code to visit each element of a collection. And there are two changes to the `switch` statement: in C#, (a) the selecting expression and the case labels may be strings, and (b) you have to indicate explictly any fall through from one arm of a switch to the next.

3.3.4 Properties, indexers and operator overloading

Besides constructors, fields and methods, in .NET languages, a class type (or a struct type) may also declare *properties*. Properties should be used instead of providing `get` and `set` methods. In the following example (coded in C#), the class `Point` is providing a property called `X`:

```
public class Point
{
    private int iX;
    ...
    public Point(int pX, int pY)
    {
        iX = pX;  iY = pY;
    }
    public int X
    {
        get { return iX;   }
        set { iX = value;  }
    }
    ...
```

```
}
...
Point tPoint = new Point(100, 200);
int  tX = tPoint.X;    // uses get
tPoint.X = 150;        // uses set
tPoint.X++;            // uses get and set
```

In C#, a class type (or a struct type) may declare *indexers* and/or *operators*. In the same way that a property provides access to one value, an indexer provides access to an array of values. An operator declaration permits a class (or a struct) to define new meanings for existing operators (such as ==, <, +, ++).

3.3.5 Delegates

In .NET languages, a *delegate* is a type-safe function pointer. Although the following code is given in C#, it could instead have been provided in Visual Basic.NET.

Here is the declaration of a delegate type called `Analyse`, and a delegate variable called `tAnalyse`:

```
delegate int Analyse(string s);
Analyse tAnalyse = new Analyse(StringLength);
```

where `StringLength` is declared as:

```
private static int StringLength(string pString)
{
    return pString.Length;
}
```

The variable `tAnalyse` now contains a pointer to the `StringLength` method.

A method can be written in terms of a parameter that is a delegate:

```
private static void iProcess(Analyse pAnalyse)
{
    string tString = Console.ReadLine();
    int tInt = pAnalyse(tString);
    Console.WriteLine(tInt);
}
```

and the particular function that is to be called can be supplied as an argument when this method is called:

```
iProcess(tAnalyse);
```

If a delegate is a `Sub` in Visual Basic.NET or its return type is void in C#, a delegate variable can be assigned a value that represents (not just one method but) a list of methods to be called.

3.3.6 Events

If the declaration of a delegate field of a class includes the keyword `event` in C# or `Event` in Visual Basic.NET, then clients of the class are restricted in the ways in which they can access the field: a client can only add new methods (or remove methods that they previously added).

One common use of events is for registering methods to be executed when an event such as a click of a button of a GUI occurs. The class `Button` (from the `System.Windows.Forms` namespace) has a field called `Click`:

```
public event EventHandler Click;
```

Suppose we have an instance of this class:

```
private Button iAddButton = new Button();
```

then a method can be added to this Button's `Click` field using:

```
iAddButton.Click += new EventHandler(iHandleClick);
```

in C# or:

```
AddHandler iAddButton.Click, AddressOf iHandleClick
```

in Visual Basic.NET. This assumes the existence of a method having the header:

```
protected void iHandleClick(object sender, System.EventArgs e)
```

in C# or:

```
Protected Sub iHandleClick(ByVal sender As Object, ByVal e As System.EventArgs)
```

in Visual Basic.NET.

3.3.7 Exception handling

Although C# and Visual Basic.NET have similar constructs as Java for throwing and catching exceptions, there is one major difference. In Java, the header of a method declaration must contain a throws clause if the code of the method does not handle a checked exception that can be thrown by the code of the method. In C# and Visual Basic.NET, it is not possible to do this. One consequence is that, if you wish a method to catch all the exceptions that its code can throw, you cannot use the compiler to check whether you are doing this.

3.4 Not just the language

Although coding in C# (or Visual Basic.NET) is easy for a Java programmer, there is one big problem: the class libraries of the .NET Framework are quite different from the APIs of Java. So the biggest challenge in moving from Java to the .NET Framework is learning the details of another set of class libraries.

4 Four kinds of .NET applications

As mentioned earlier, the .NET Framework (and Microsoft's Visual Studio.NET) makes it easy to produce console applications, windows forms applications, web form applications and web services. Examples of each of these will now be given. Although these examples are coded using C#, each of these examples could just as easily have been coded in any other .NET language.

4.1 Console applications

Here is a program that reads in a temperature given in degrees Centigrade and outputs the corresponding value in degrees Fahrenheit:

```
using System;
namespace ConsoleConvert
{
    class Class1
    {
        [STAThread]
        static void Main()
        {
            Console.Write("Centigrade value: ");
            string tCentigradeString = Console.ReadLine();
            double tCentigrade = double.Parse(tCentigradeString);
            double tFahrenheit = 32 + tCentigrade*9/5;
            Console.WriteLine("Fahrenheit value: " + tFahrenheit);
        }
    }
}
```

4.2 Windows Forms applications

The .NET Framework includes a number of classes that can be used to provide an application driven by a GUI. Some of these classes are used in the following program:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace MyWindowsConvert
{
    public class Form1 : Form
    {
        private TextBox textBox1;
        private Button button1;
        private Label label1;
        public Form1()
        {
            textBox1 = new TextBox();
            textBox1.Location = new Point(64, 32);
            textBox1.Size = new Size(120, 20);
            Controls.Add(textBox1);
            button1 = new Button();
            button1.Location = new Point(64, 64);
            button1.Size = new Size(120, 20);
            button1.Text = "Get Fahrenheit";
            button1.Click += new EventHandler(button1_Click);
            Controls.Add(button1);
            label1 = new Label();
            label1.Location = new Point(64, 104);
            label1.Size = new Size(120, 20);
            Controls.Add(label1);
            Text = "MyWindowsConvert";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            double tCentigrade = double.Parse(textBox1.Text);
```

```
            double tFahrenheit = 32 + tCentigrade*9/5;
            label1.Text = tFahrenheit.ToString();
        }
        public static void Main()
        {
            Form1 tForm1 = new Form1();
            Application.Run(tForm1);
        }
    }
}
```

The class `Form1` is derived from `System.Windows.Forms.Form`. Its constructor creates a `TextBox` object, a `Button` object and a `Label` object. It adds each of these to the `Controls` property of the `Form`.

A `Button` object has an `Event` field called `Click` and `Form1`'s constructor uses `+=` to add a method called `button1_Click` to the list of methods awaiting each click of the button.

When the button is clicked, the `button1_Click` method is executed. This takes the string stored in the `TextBox`, converts it to a double, produces the corresponding value in Fahrenheit, and stores this as a string in the `Label`.

The class `Form1` has a `Main` method. When this program is run, it creates a `Form1` object and passes this an argument to `Application`'s `Run` method. The program will terminate when the close button of the form is clicked.

Although it is possible to produce the above program using any text editor, Visual Studio.NET has a wizard that can be used to generate a Windows Forms application in C# (or VB.NET). If this is used, you can generate the program by dragging a TextBox, a Button and a Label from the ToolBox onto the form and then adding the C# code to respond to a click of the button. Although the code Visual Studio.NET generates is more verbose than that given above, most of it is reasonably easy to understand.

4.3 Web Form applications

It is also possible to use Visual Studio.NET to generate a web equivalent of the above program. The appropriate wizard can be used to produce (in a file called `WebForm1.aspx`) a WWW page containing the form:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
         AutoEventWireup="false" Inherits="WebFormConvert.WebForm1" %>
...
<HTML>
    ...
    <form id="Form1" method="post" runat="server">
        <asp:TextBox id="TextBox1"
             style="-INDEX: 101; LEFT: 103px; POSITION: absolute; TOP: 50px"
             runat="server">
        </asp:TextBox>
        <asp:Button id="Button1"
             style="-INDEX: 102; LEFT: 117px; POSITION: absolute; TOP: 96px"
             runat="server" Text="Get Fahrenheit"></asp:Button>
        </asp:Button>
        <asp:Label id="Label1"
             style="-INDEX: 103; LEFT: 152px; POSITION: absolute; TOP: 144px"
             runat="server">
        </asp:Label>
    </form>
    ...
</HTML>
```

together with a file (called `WebForm1.aspx.cs`) containing the code that is to be executed when the user clicks on the button:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
namespace WebFormConvert
{
    public class WebForm1 : Page
    {
        protected TextBox TextBox1;
        protected Button Button1;
        protected Label Label1;
        private void Page_Load(object sender, EventArgs e)
        {
        }
        override protected void OnInit(EventArgs e)
        {
```

```
            InitializeComponent();
            base.OnInit(e);
        }
        private void InitializeComponent()
        {
            this.Button1.Click += new System.EventHandler(this.Button1_Click);
            this.Load += new System.EventHandler(this.Page_Load);
        }
        private void Button1_Click(object sender, EventArgs e)
        {
            double tCentigrade = double.Parse(TextBox1.Text);
            double tFahrenheit = 32 + tCentigrade*9/5;
            Label1.Text = tFahrenheit.ToString();
        }
    }
}
```

Note that the HTML instructions (for the WWW page) and the C# code are in separate files. This separation means that the two files could be looked after by different people.

The WWW page can be visited using a URL which is something like:

```
http://machine.site.ac.uk/WebFormConvert/WebForm1.aspx
```

This is an ASP.NET page, and so `machine.site.ac.uk` would need to be running Microsoft's Internet Information Server (IIS) as a web server.

4.4 Web Services

4.4.1 What is a Web Service?

Instead of providing a WWW page that has a form to convert a temperature, we could provide a *Web Service*. A Web Service consists of one or more methods (written in a .NET language) that can be invoked by some external site. The external site can pass arguments to the method, and the method returns a result to the external site.

External sites submit requests through a web server to the method via *SOAP*, *HTTP GET* or *HTTP POST*. These requests include both the name of the method to be executed and any arguments to be passed to the method.

Here is an example of what an external site would send to the web server if SOAP were used:

```
POST /barry.cornelius/moving/WebServerConvert/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: XXXX
SOAPAction: "http://www.dur.ac.uk/barry.cornelius/webservices/ToFahrenheit"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ToFahrenheit xmlns="http://www.dur.ac.uk/barry.cornelius/webservices/">
      <pCentigrade>0</pCentigrade>
    </ToFahrenheit>
  </soap:Body>
</soap:Envelope>
```

The web server of the external site passes back the result coded using XML. Here is an example:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: YYYY

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ToFahrenheitResponse xmlns="http://www.dur.ac.uk/barry.cornelius/webservices/">
      <ToFahrenheitResult>32</ToFahrenheitResult>
    </ToFahrenheitResponse>
  </soap:Body>
</soap:Envelope>
```

4.4.2 Providing a Web Service

Visual Studio.NET provides a wizard that generates the files necessary to provide a Web Service. All the programmer has to do is to provide the code of each method flagging each method with a `WebMethod` attribute. An example is:

```
[WebService]
public class Service1 : System.Web.Services.WebService
{
    ...
    [WebMethod]
    public double ToFahrenheit(double pCentigrade)
    {
        return 32 + pCentigrade*9/5;
    }
}
```

Note that the programmer does not have to decode the incoming HTTP requests or generate any XML: all of this is handled elsewhere.

4.4.3 Accessing a Web Service

Visual Studio.NET also provides a wizard that generates a proxy class that can be used to provide access to a Web Service on an external site. You just supply the wizard with the URL of the Web Service; it queries the Web Service in order to discover the methods that the Web Service provides together with their signatures and return types; and then it automatically produces the code of a class that has similar methods.

A programmer wishing to use a Web Service can use this proxy class. Here is an example of a call of a method of a proxy class:

```
Proxy.Service1 tService1 = new Proxy.Service1();
double tFahrenheit = tService1.ToFahrenheit(tCentigrade);
```

And here is an example of a proxy class:

```
public class Service1 : System.Web.Services.Protocols.SoapHttpClientProtocol
{
    ...
    public double ToFahrenheit(double pCentigrade)
    {
        object[] results = this.Invoke("ToFahrenheit", new object[]{pCentigrade});
        return (double) results[0];
    }
    ...
}
```

When a method of the proxy class is called, the code of the method sends the appropriate SOAP request to the external site; decodes the XML that is returned by the external site; and returns an appropriate value to the caller of the method.

Once again, the programmer does not have to do the complicated bits: formulating a SOAP request and decoding the XML that is returned by the Web Service.

If you have not got Visual Studio.NET, you can instead generate a proxy class with a tool (called `wsdl.exe`) that is part of the .NET Framework (which is free). This tool lives in the directory:

```
C:\Program Files\Microsoft.NET\FrameworkSDK\Bin
```

5 Conclusions

Microsoft are putting a lot of work into the .NET Framework: it is going to be important.

If you have not switched to Java, you may want to consider C# or VB.NET instead: both languages are object-oriented languages that are as sophisticated as Java. Java progammers will find it very easy to use C# as the languages are similar.

This paper has given some examples of C# code. In particular, it has demonstrated that it is easy to produce code that can be executed by a web server when a browser visits a WWW page. Compared with ASP, ASP.NET pages can be executed faster and can be more sophisticated.

And this paper has also indicated that we can provide a web server with a Web Service, a set of methods that can be invoked by some other site. These Web Services rival the facilities offered by CORBA and Java's Remote Method Invocation (RMI).

6 References

1. Ben Albahari, 'A Comparative Overview of C#',
   http://genamics.com/developer/csharp_comparative.htm

2. Ben Albahari, Peter Drayton and Brad Merrill, 'C# Essentials (2nd edition)',
   O'Reilly, 2002, 0-596-00315-3.

3. Barry Cornelius, 'A Taste of C#',
   http://www.dur.ac.uk/barry.cornelius/papers/a.taste.of.csharp/

4. Barry Cornelius, ‘Web Services’,
http://www.dur.ac.uk/barry.cornelius/papers/web.services/

5. Harvey Deitel, Paul Deitel, J. Listfield, T. R. Nieto, Cheryl Yaeger, Marina latkina, ‘C# How to Program’,
Prentice Hall, 2002, 0-13-062221-4.

6. Eric Gunnerson, ‘A Programmer’s Introduction to C# (2nd edition)’,
Apress, 2001, 1-893115-62-3

7. Billy Hollis and Rockford Lhotka, ‘VB.NET Programming (With the Public Beta)’,
Wrox Press, 2001, 1-861004-91-5. Although this book is still useful, it has not been updated for Beta 2.

8. Mark Johnson, ‘C#: A language alternative or just J–?’,
http://www.javaworld.com/javaworld/jw-11-2000/jw-1122-csharp1.html

9. Microsoft, ‘C# Language Specification’,
http://msdn.microsoft.com/vstudio/nextgen/technology/csharpdownload.asp

10. Microsoft, ‘Visual Studio.NET and .NET Framework Reviewers Guide’,
http://msdn.microsoft.com/vstudio/nextgen/evalguidedownload.asp

11. Microsoft, ‘Visual Basic .NET Upgrade Guide’,
http://msdn.microsoft.com/vbasic/technical/upgrade/guide.asp

12. Microsoft (MSDN UK), ‘Visual Studio.NET Guided Tour’,
http://www.microsoft.com/uk/msdn/vstudiotour/tour.htm

13. ‘The Mono project’,
http://www.go-mono.com/

14. Dare Obasanjo, ‘A comparison of Microsoft’s C# programming language to Sun Microsystem’s Java programming language’,
http://www.25hoursaday.com/CsharpVsJava.html

15. Thuan Thai and Hoang Q. Lam, ‘.NET Framework Essentials (2nd edition)’,
O’Reilly, 2002, 0-596-00302-1.