.NET Framework

C# Introduction

C# is an object-oriented programming language that supports data encapsulation, inheritance, polymorphism, and method overriding. C# programming language was developed at Microsoft in the late 1990s headed by Anders Hejlsberg. It allows you to build various types of applications, such as web, windows, mobile, and cloud-based applications.

The C# has the following features:

- It allows you to develop both console applications and web applications.
- It supports garbage collection and automatic memory management.
- It includes native support for the Component Object Model (COM) and Windows-based applications.
- In C#, the values of the primitive data types are automatically initialized to zeros, and reference types like objects and classes are initialized to null.
- It produces portable code.
- Its programs execute in a secure, controlled runtime environment.
- It supports generics, partial types, and anonymous methods that expand the scope, power, and range of the language.
- It supports language-integrated query (LINQ) and lambda expressions. **LINQ** enables you to write database queries in C# programming while **lambda expressions** are used in LINQ expressions for efficient coding.

.NET Framework Structure

The **.NET framework** is a collection of tools, technologies, and languages that provides an environment to build and deploy different types of applications easily. It is a software development framework from Microsoft. The figure below describes the structure of the .NET framework and its components.

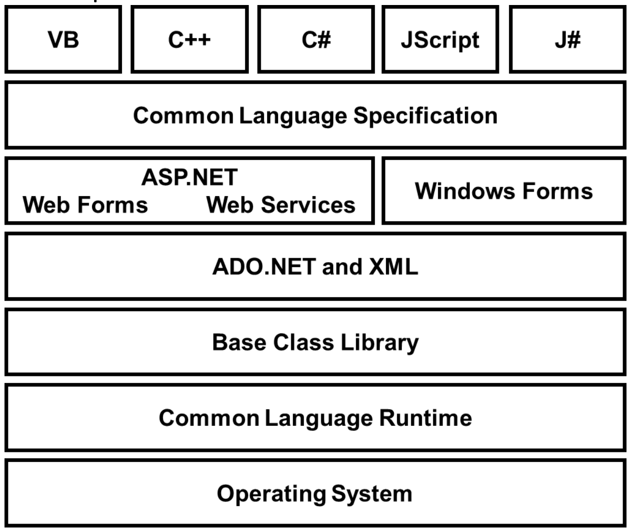

Figure 1. .NET framework structure (Harwani, 2015)

- The **operating system** controls tasks and manages system resources of the computer. The .NET framework runs on Microsoft Operating Systems (OS), but there are alternative versions that will work on other OS. The OS manages the hardware and software resources that supports the execution of .NET applications.
- **Common Language Runtime (CLR)** is a runtime environment of the .NET framework that manages the execution of the .NET code, enables debugging and exception handling, and makes programs portable. It converts (or compiles) the source code into an intermediate language called **Microsoft Intermediate Language (MSIL)**. The MSIL is a portable assembly code that con
. Then the compiler called **CLR Just-In-Time (JIT)** compiles the MSIL code into native code (machine-language code) before it is executed.
- The **.NET Base Class Library** contains the classes and interfaces used for building applications. The classes are organized as namespaces, and the developers can use them by including these namespaces in their programs. These classes or .NET objects can be used to build and interact with ASP.NET, Windows Forms, and/or ADO.NET and XML.
- **ADO.NET** provides access to relational databases and several data sources such as SQL Server and XML. The **XML** is the universal format for data on the Web. It allows developers to easily describe and deliver rich, structured data from any

application in a standard, consistent way.

- **ASP.NET** is a unified Web development model that includes the services necessary in building enterprise-class Web applications with a minimum of coding. **ASP.NET Web Forms** allows building of dynamic websites using a familiar drag-and-drop, event-driven model. **ASP.NET Web Services** extend the Web infrastructure to provide the means for software to connect to other software applications. The **Windows Forms** contain the graphical representation of any window displayed in the application.
- **Common Language Specification (CLS)** is a set of basic language features that ensures operability between the languages in the .NET environment. It is a subset of CTS. **Common Type System** is a formal specification that documents how types are declared, used, and managed so the CLR can use them. The CLS tells the guidelines to the compiler of each language.
- The .NET supports different programming languages. The VB, C++, C#, Jscript, and J# are languages supported by .NET.

<u>.NET supports the following features:</u>

- *Interoperability* – .NET includes a large library and supports several programming languages, allowing developers to write code in different languages.
- *Language independence* – The code in all the supported languages is compiled into Common Language Infrastructure (CLI) specifications, enabling the exchange of data types between programs developed in different languages.
- *AJAX* – .NET supports AJAX which is used by developers in creating highly responsive web applications with minimal effort.
- *Security* – .NET uses assembly for code sharing which allows only authorized categories of users or processes to call designated methods of specific classes.
- *Common Language Runtime*
- *Development for dynamic Web pages*
- *Base Class Library*
- *ADO.NET*
- *Web services*

Assembly Use

An **assembly** is a collection of types and resources that are built to work together and form a logical unit of functionality. Assemblies are the building blocks of the .NET framework applications and they form the fundamental unit of deployment version control, reuse, activation scoping, and security permissions. Applications in .NET are deployed as an assembly. Assemblies are compiled code in .NET that contains the code that CLR executes.

The entire .NET code on compilation is converted into an intermediate language code and is stored as an assembly. Assemblies can be in a form of either:

- **EXE (executable)**
- **DLL (Dynamic-Link Library)** is a module that contains functions and data that can be used by another module (application or DLL).

The assembly contains **metadata** that provides information along with a complete description of methods, types, and other resources. The metadata includes **Manifest**, which stores identification information, public types, and a list of other assemblies that are used in the current assembly. Each assembly has a **128-bit** version number that is presented as a set of four (4) decimal pieces: **Major.Minor.Build.Revision** *(Ex. 8.0.1.7)*. This concept of versioning enables the developers to install new versions of the components that are not backward compatible.

<u>There are two (2) types how an assembly is deployed: **private** and **shared**. Their differences are listed below:</u>

- Multiple applications can use a shared assembly. Only a single application can use a private assembly.
- A shared assembly has to be registered in GAC. Global Assembly Cache stores assemblies specifically designated to be shared by several applications on the computer. There's no need to register a private assembly since it is stored in the respective application's directory.
- A strong name has to be created for a shared assembly while it is not required for a private assembly.
- A shared assembly can have multiple versions that are not backward compatibility. A private assembly with multiple versions that are not backward compatible, the application stops working.
- By default, all assemblies created are considered private assemblies. When a private assembly is registered in GAC with a strong name, it becomes a shared assembly.

Program Structure of C#

A C# program consists of the following parts:

- Namespace declaration
- Class attributes
- Comments
- A class
- A Main method
- Class methods
- Statements and expressions

The following example console application shows the structure of a C# program:

Code Listing 1. Sample C# program

```
/*
This program prompts the users to enter their name and age then display them.
Author: Jess Diaz
*/

using System;
namespace ConsoleApp
{
	class GreetingProgram
	{
		static void Main (string[] args)
		{
			Console.Write("Enter your name: "); //this prints in console
			string name = Console.ReadLine(); //this reads the entered input by user as string
			Console.Write("Enter your age: ");
			int age;
			age = Convert.ToInt32(Console.ReadLine()); //this statement converts string to int
			Console.WriteLine("Hi! Your name is " + name + " and your age is " + age);
			Console.Write("\nPress any key to exit...");
			Console.ReadKey();
		}
	}
}
```

- The namespaces are used to organize classes in a project. The **`namespace`** declaration is a collection of classes. In Code Listing 1 sample program, the namespace **`ConsoleApp`** contains the class **`GreetingProgram`**. The code **`System`** is a `namespace` and one (1) of its class is **`Console`**. The **`using`** keyword is used to access the classes available in a `namespace`. The **`using namespace;`** (e.g., `using System;`) declaration must appear at the beginning of a source code. Every C# program must be save as the same with its class name with extension **.cs** (e.g., GreeetingProgram.cs).
- The statements enclosed in **`/*…*/`** and **`//`** are comments and ignored by the compiler. These are used to add notes to help programmers to easily read the program.
- Every C# application contains one (1) **`Main`** method. The `Main` method is the starting point of every application and it contains the statements that will be executed. In the sample program, the **`static void Main (string[] args) {}`** takes `string` array as argument. This method must be defined in the `class` as **`static`** method.
- The `Main` method contains the statements and expressions that will be executed when the program is compiled. All statements and expressions in C# ends with semicolon (`;`).

REFERENCES:

Deitel, P. and Deitel, H. (2015). *Visual C# 2012 how to program* (5th Ed.). USA: Pearson Education, Inc.

Gaddis, T. (2016). *Starting out with visual C# (*4th Ed.). USA: Pearson Education, Inc

Harwani, B. (2015). *Learning object-oriented programming in C# 5.0*. USA: Cengage Learning PTR.