# Introducción

Este documento describe reglas y recomendaciones para el desarrollo de aplicaciones utilizando el lenguaje C# en la empresa “WebSoft”.

Específicamente, este documento contiene:

* Convenciones de nomenclatura
* Estilo de codificación
* Uso del lenguaje
* Diseño de modelos de objetos.

## Objetivo

El principal objetivo de la utilización de estándares de codificación, es institucionalizar buenas prácticas y recomendaciones de diseño, para lograr mayores niveles de calidad en los productos de software desarrollados.

La utilización de estándares debe mostrar efectos positivos sobre:

* Disminución de errores / bugs, especialmente en aquellos difíciles de individualizar.
* Mantenimiento del código estructurado de acuerdo a las guías de diseño.
* Mantenimiento del código escrito de acuerdo a un estilo estandarizado.
* Performance de la aplicación desarrollada con prácticas eficientes del lenguaje.

## Alcance

Este documento aplica únicamente al lenguaje C# y .Net . No se incluyen referencias o prácticas relacionadas al uso de las bibliotecas de clases de .Net Framework. Se incluyen algunos comentarios acerca de prácticas comunes y problemas conocidos de C#, las cuales se describen en forma breve.

Dado que las prácticas recomendadas en este manual se aplicarán en forma institucional, se han incluido normas que describen el uso de llaves ( {} ) y tabulaciones, para lograr una lectura uniforme del código generado en “WebSoft”.

## Condiciones de uso de este documento

Una regla puede romperse sólo ante razones justificadas, discutidas, con previa autorización del responsable del producto, y en caso que no pueda aplicarse ninguna alternativa razonable. El autor de la excepción, obligatoriamente debe documentar el código explicando la causa de la violación de la regla.

Las preferencias personales no se consideran una razón justificada.

## Fuentes de información consultadas

Basado en: Coding Standard C# - Phillips Medical Systems, C# Coding Standards - Lance Hunt, Microsoft Naming Guidelines - Microsoft Corp.

## Estilos de codificación

| *Código* | ***Estilo*** |
| --- | --- |
| Archivo del código fuente (Source code) | Un archivo por Namespace y un archivo por clase. |
| Llaves **{}** | Siempre en una nueva línea. Utilizar paréntesis cuando sea opcional. |
| Tabulación (Indent) | Utilizar siempre tabs de 4 caracteres. No utilizar espacios en blanco. |
| Comentarios (Comments) | Utilizar “//”. No utilizar “/\*…\*/”. |
| Variables | Una variable por cada declaración. |
| Tipos de datos nativos (Native Data Types) | Priorizar el uso de tipos nativos de C# ante tipos del CTS.  Ejemplo: int en lugar de Int32. |
| Enumerados (Enums) | Evitar cambiar el tipo de dato predeterminado. |
| Tipos genéricos (Generic types) | Utilizar preferiblemente tipos genéricos ante tipos estándar o strong-typed classes. |
| Propiedades (Properties) | No utilizar prefijos Get / Set. |
| Métodos (Methods) | Utilizar un máximo de 7parámetros. |
| Base / This | Utilizar solo en **constructors** o dentro de **override**. |
| Condiciones ternarias (Ternary conditions) |  |
| For each | No modificar tipos enumerados dentro de una sentencia foreach. |
| Condicionales (Conditionals) | Evitar la evaluación de condiciones booleanas contra valores true o false.  No realizar asignaciones embebidas.  No utilizar invocación de métodos embebidos. |
| Manejos de excepciones (Exception Handling) | No utilizar excepciones para el control de flujo.  Utilizar throw; NO throw e; cuando se redirije.  Sólo utilizar **catch** cuando es posible manejar la excepción.  Utilizar validaciones para evitar la ocurrencia de excepciones. |
| Eventos (Events) | Siempre debe validarse el valor **null** antes de la invocación. |
| Bloqueo (Locking) | Utilizar lock() en lugar de Monitor.Enter()  No utilizar lock en un tipo, por ejemplo: lock(typeof(MyType));  Evitar el uso de lock ante un this, por ejemplo: lock(this);  Preferiblemente, utilizar lock en objetos declarados como private, por ejemplo: lock(myVariable); |
| Dispose() / Close() | Utilizarlo siempre que sea possible. |
| Finalizers | Evitar la implementación de estos métodos.  Utilizar la sintaxis del destructor de C#. Ejemplo: ~MyClass() {…}  Nunca debe definirse un método llamado Finalize(); |
| AssemblyVersion | Incrementar en forma manual. |
| ComVisibleAttribute | Debe estar con el valor false para todos los assemblies. |

1. **Convenciones y Estándares de Nombres**

**1.** Usa palabras entendibles y descriptivas para nombrar a las variables. No uses abreviaciones.

Correcto:

string direccion;

int salario;

Incorrecto:

string nom;

string domic;

int sal;

2. No uses nombres de variables de un solo caracter como i, n, s etc. Usa nombres como indice, temp

Una excepción en este caso podría ser las variables usadas para iteraciones en los ciclos:

for ( int i = 0; i < cuantos; i++)

Si la variable es usada solo como un contador para una iteración y no es usada en ningún otro lado dentro del ciclo, mucha gente le agrada nombrar la variable con un solo carácter (i) en vez de usar un nombre adecuado distinto.

**3.** No uses guiones bajos (\_) para nombres de variables locales.

**4.** Todas las variables globales deben usar el prefijo de guión bajo (\_) de tal forma que puedan ser identificadas de otras variables locales.

No uses palabras reservadas para nombres de variables.

**5.** Usa el prefijo “Es” ó “Is” para variables de tipo boolean o prefijos similares.

Ejemplo:

private bool EsValido

private bool IsActivo

La propuesta de “Is” , más aceptada, viene dada a que es coherente con las propiedades de uso global de .NET , por ejemplo: objeto.IsEmpty(); se agrupa en el IntelliSense de una forma más coherente

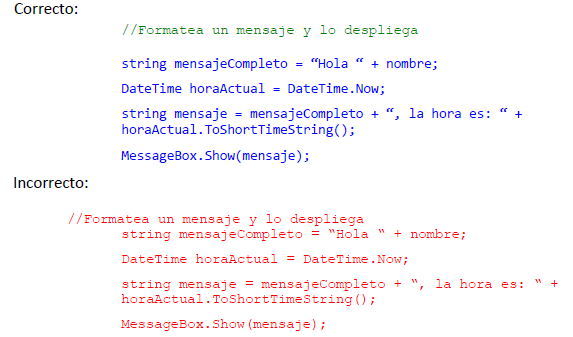
**6.** Usa un prefijo apropiado para cada elemento de la interfaz gráfica. Una lista breve es dada a continuación. Dado que .NET tiene una gran cantidad de controles, tu puedes lograr una lista completa de prefijos estándares para cada control (incluyendo controles de terceras partes) que estés usando.

|  |  |
| --- | --- |
| **Control** | **Prefijo** |
| Label | lbl |
| TextBox | txt |
| DataGrid | dtg |
| Button | btn |
| ImageButton | Imb |
| Hyperlink | hlk |
| DropDownList | ddl |
| ListBox | lst |
| DataList | dtl |
| Repeater | rep |
| Checkbox | chk |
| CheckboxList | cbl |
| RadioButton | rbt |
| RadioButtonlist | rbl |
| Image | img |
| Panel | pan |
| PlaceHolder | phd |
| Table | tbl |
| Validators | val |

1. **Sangría y Espaciamiento**

**1**. Usa TAB para la sangría. No uses ESPACIOS. Define el tamaño del Tab de 4 espacios.

2. Los comentarios deben estar al mismo nivel que el código (usar el mismo nivel de sangría).



**3.** Las llaves ({}) deben estar en el mismo nivel que el código fuera de las llaves.

if ( … )

{

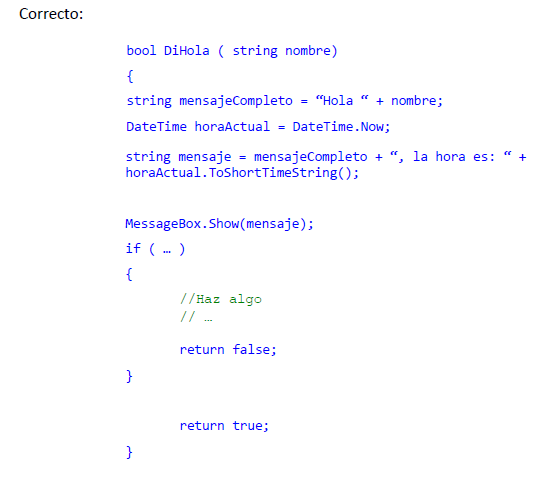
//Haz algo

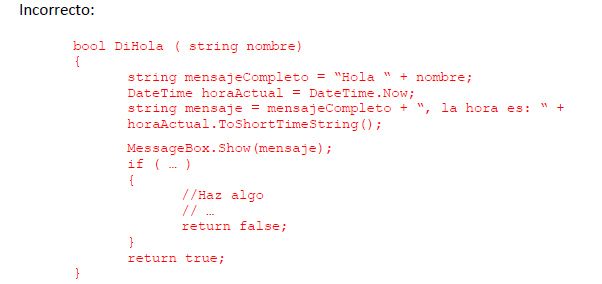
// …

return false;

}

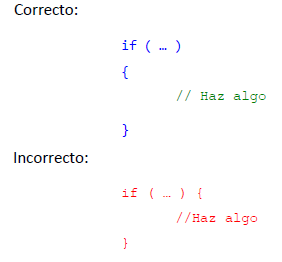
**4.** Usa una línea en blanco para separar un grupo lógico de código



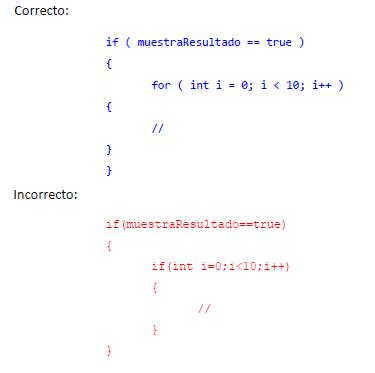


**5.** Debe haber una y solo una línea en blanco entre cada método dentro de las Clases.

**6.** Las llaves ({}) deben estar en una línea separada y no en la misma línea del if, for etc.



**7.** Usa un espacio simple antes y después de los paréntesis y los operadores.



**8**. Usa #region para agrupar piezas de código juntas. Si tu usas una agrupación apropiada usando #region, la página debe verse como a continuación cuando todas las definiciones estén cerradas.

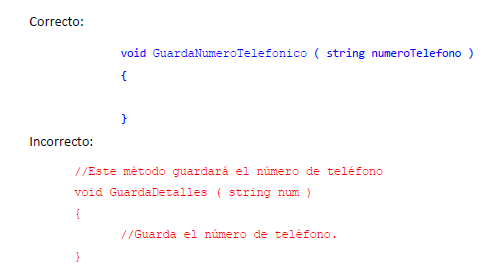


**9.** Mantén privadas las variables globales, las propiedades y los métodos en la parte superior del archivo y los elementos públicos en la parte inferior.

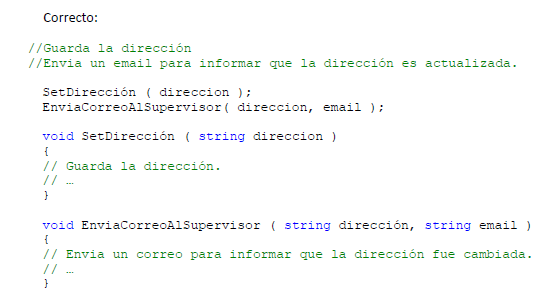
1. **Buenas prácticas de programación**

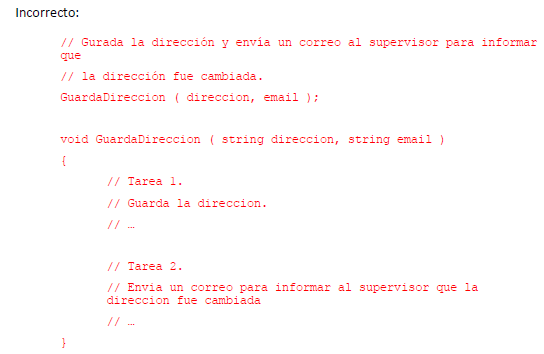
**1.** Evita escribir métodos muy largos. Un método debe típicamente tener entre 1 a 25 líneas de código. Si un método tiene más de 25 líneas de código, debes considerar refactorizarlo en métodos separados.

**2.** El nombre de los métodos debe decir lo que hace. No uses nombres engañosos. Si el nombre del método es obvio, no hay necesidad de documentación que explique qué hace el método.



**3.** Un método debe tener solo ‘una tarea’. No combines más de una tarea en un solo método, aún si esas tareas son pequeñas.





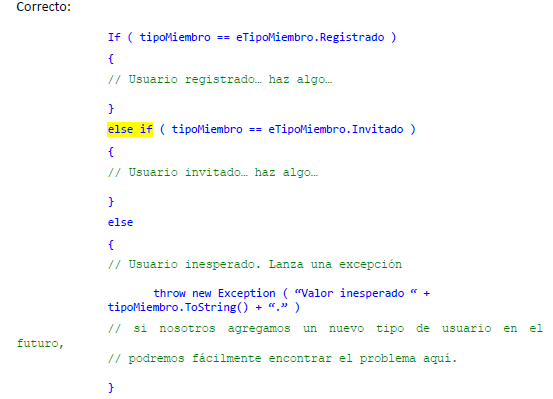
**4.**Usa los tipos específicos de C# o VB.NET (alias), en vez de los tipos definidos en el espacio de nombres System.

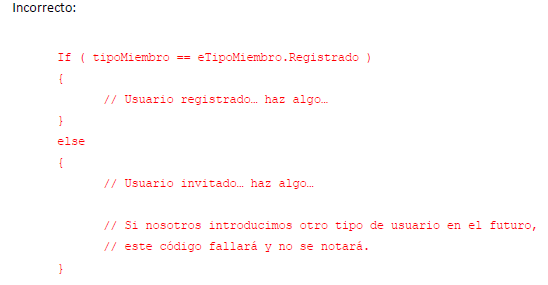
int edad; (no Int16)

string nombre; (no String)

object infoContrato; (no Object)

**5.** Siempre verifica valores inesperados. por ejemplo, si estas usando un parámetro con 2 posibles valores, nunca asumas que si uno no concuerda entonces la única posibilidad es el otro valor.



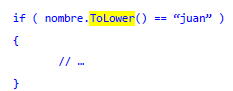


**6.** No incrustes números en el código. En vez de eso usa constantes. Declara constantes en la parte superior del archivo y úsalas en tu código.

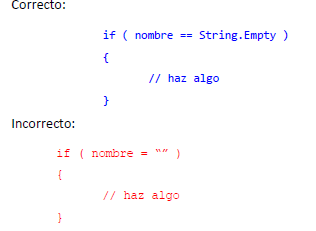
Sin embargo, usar constantes tampoco es recomendado . Se debe usar las constantes en el archivo de configuración o en la base de datos, de tal forma que las puedas cambiar posteriormente. Declara los valores como constantes solo si tu estas totalmente seguro de que este valor nunca necesitará ser cambiado.

**7.** No incrustes cadenas de texto en el código. **Usa archivos de recursos**.

8. Convierte las cadenas de texto a minúsculas o mayúsculas antes de compararlas. Esto asegurará que la cadena coincida.



**9.** Usa String.Empty en vez de “”.



**10.** Evita usar variables globales. Declara variables locales siempre que sea necesario y pásalas a otros métodos en vez de compartir una variable global entre métodos. Si compartes una variable global entre métodos, te será difícil rastrear qué método cambia el valor y cuando.

**11.** Usa enum dondequiera que sea requerido. No uses números o cadenas para indicar valores discretos.

**12.** No definas las variables globales públicas o protegidas. Mantenlas privadas y expón Propiedades públicas/protegidas.

**13.** Las rutinas que controlan los eventos (event handlers) no deben contener el código que ejecuta la acción requerida. En vez de ello, llama a otro método desde la rutina controladora.

ButtonVentas\_Click(Object sender, EventArgs e) Handles Button1.Click

{

GetVentasPorMes();

}

**14.** No invoques programáticamente el evento Click de un botón para ejecutar la misma acción que has escrito en el evento. En vez de ello, llama el mismo método que es invocado desde la rutina controladora del evento click.

**15.** Nunca incrustes en el código rutas o letras de dispositivos. Obtén la ruta de la aplicación programáticamente y usa rutas relativas a ella.

**16.** Nunca asumas que tu código se ejecutará desde el disco “C:”. No puedes saber si algunas usuarios lo ejecutan desde la red o desde “Z:”.

**17.** En el arranque de la aplicación, ejecuta una clase de “auto verificación” y asegúrate que todos los archivos requeridos y las dependencias están disponibles en las ubicaciones esperadas. Verifica las conexiones a la base de datos en el arranque, si es requerido. Dale un mensaje amigable al usuario en caso de algún problema.

**18.** Si el archivo de configuración requerido no se encuentra, la aplicación debe ser capaz de crear uno con valores predeterminados.

**19.** Los mensajes de error deben ayudar al usuario a resolver el problema. Nunca muestres mensajes de error como “Error en la Aplicación”, “Hay un error…” etc. En vez de ello da mensajes específicos como “Fallo al actualizar la base de datos”, sugiere lo que el usuario debe realizar: “Fallo al actualizar la base de datos. Por favor asegúrate de que la cuenta y la contraseña sean correctos”.

**20.** Cuando despliegues mensajes de error, adicionalmente a decirle al usuario qué está mal, el mensaje debe también decirle lo que el usuario debe hacer para resolver el problema. En vez de un mensaje como “Fallo al actualizar la base de datos.”, sugiere lo que el usuario debe hacer: “Fallo al actualizar la base de datos. Por favor asegúrate de que la cuenta y la contraseña sean correctos.”.

**21.** Muestra mensajes cortos y amigables al usuario. Pero registra el error actual con toda la información posible. Esto ayudará mucho a diagnosticar problemas.

**22.** Usa ficheros de recursos (.resx) para todos los literales de la aplicación. Nos favorece el cambio idiomático, cambio de literales rápido, unificación de mensajes de eventos, etc. En desarrollos muy compartidos, pensar en encapsular en una/s librería/s independiente/s los literales y mensajes para que pueda ser fácilmente compartido por las demás librerías.

¿y por qué no hacer un sistema basado XML ó Tablas en BBDD?... nadie te lo impide. Pero ten en cuenta que un fichero .resx es un sistema “específico” de .NET ideado para esta función y por tanto muy optimizado y “precompliado” (por lo tanto más rápido de acceder). XML es más lento y en cadenas largas muy poco optimo.

**23.** No guardes más de una clase en un solo archivo.

**24.** Evita tener archivos muy grandes. Si un solo archivo tiene más de 1000 líneas de código, es un buen candidato para refactorizar. Divídelos lógicamente en dos o más clases.

**26.** Evita pasar muchos parámetros a un método. Si tienes más de 4~5 parámetros, es un buen candidato para definir una clase o una estructura. Lo contrario destroza el consumo en memoria, más facilidad de corrupción de datos, más castigo a los ciclos del procesador... etc

**27.** Si tienes un método que retorna una colección, devuelve una colección vacía en vez de null, si no hay datos que retornar. Por ejemplo, si tienes un método que retorna un ArrayList, siempre retorna un ArrayList válido. Si no tienes elementos que devolver, entonces retorna un ArrayList válido con 0 elementos. Esto hará fácil para la aplicación que llama al método verificar solamente la propiedad “Count” en vez que hacer verificaciones adicionales para “null”.

1. **Arquitectura**

**1.** Siempre usa una arquitectura multicapa (N-Tier).

**2.** Nunca accedas a la base de datos desde las páginas de la interfaz gráfica. Siempre ten un conjunto de clases de capa de acceso a datos la cual ejecute todas las tareas relacionadas con la base de datos.

**3.** Usa las sentencias try-catch en tu capa de acceso a datos para atrapar todas las excepciones de la base de datos. Este controlador de excepciones debe registrar todas las excepciones desde la base de datos. Los detalles registrados deben incluir el nombre del comando siendo ejecutado, nombre del procedimiento almacenado, parámetros, cadena de conexión usada etc. Después de registrar la excepción, debe de re lanzarse para que la otra capa en la aplicación la atrape y tome las acciones apropiadas.

**Referencias:**

 Aplicación de criterios de usabilidad WEB

o http://www.canaldenegocio.com/docs/AplicaciondecriteriosdeusabilidadWEB.pdf

 Bases para la implementación de un UI WinForms

o http://www.canaldenegocio.com/docs/BasesdeUI.pdf

 Informe APEI de Usabilidad.

o http://www.canaldenegocio.com/docs/informeapeiusabilidad.pdf