UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

Proyecto *Sistema de Gestión de Configuración de Software*

Curso: *Gestión de la configuración de software*

Docente: *Mag. Ricardo Eduardo Valcárcel Alvarado*

Integrantes:

Villanueva Yucra, Josué	(2018000722)
Aguilar Pinto, Víctor Eleazar	(2017057405)
Chino Conde, Oswaldo	(2017057434)
Perez Vizcarra, Juan José	(2019063636)

Tacna – Perú
2023

RASGO

UNIVERSIDAD PRIVADA DE TACNA

Sistema de Gestión de Configuración de Software
Estándar de Programación

Versión *1.0*

INDICE GENERAL

Contenido

1. Objetivo 4
2. Declaración De Variables 4
   2.1. Descripción de la Variable. 4
   2.2. Variables de Tipo Arreglo 4
3. Definición de Controles 5
   3.1. Tipo de datos 5
   3.2. Prefijo para el Control 6
   3.3. Nombre descriptivo del Control 6
   3.4. Declaración de variables, atributos y objetos 6
   3.5. Declaración de clases 7
   3.6. Declaración de métodos 7
   3.7. Declaración de funciones 8
   3.8. Control de versiones de código fuente 8
4. Clases 9
5. Métodos, Procedimientos y Funciones definidos por el Usuario 9
6. Beneficios 13
7. Conclusiones 13

1. Objetivo

Reglamentar la forma en que se implementará el código fuente del proyecto, pasando, por las variables, controles, clases, métodos, ficheros, archivos y todo aquello que esté implicado en el código,

Mejorar y uniformizar a través de las reglas que se proponen, el estilo de programación que tiene cada programador.

- ✓ Los nombres de variables serán mnemotécnicos con lo que se podrá saber el tipo de dato de cada variable con sólo ver el nombre de la variable.
- ✓ Los nombres de variables serán sugestivos, de tal forma que se podrá saber el uso y finalidad de dicha variable o función fácilmente con solo ver el nombre de la variable.
- ✓ La decisión de poner un nombre a una variable o función será mecánica y automática, puesto que seguirá las reglas definidas por nuestro estándar.
- ✓ Permite el uso de herramientas automáticas de verificación de nomenclaturas.

Por tanto, se seguirán dichos patrones para un entendimiento legible del código y para facilitar el mantenimiento de este.

2. Declaración De Variables

2.1. Descripción de la Variable.

Nombre que se le asignará a la variable para que se le identifique y deberá de estar asociada al motivo para la cual se le declara, conectados por un sub- guion. Además, el cuerpo de un nombre de variable o procedimiento se debe escribir la primera letra en mayúsculas y el resto en minúsculas y debe tener la longitud necesaria menos a 30 caracteres para describir su funcionalidad.
Para nombres que se usen con frecuencia o para términos largos, se usara abreviaturas estándar para que los nombres tengan una longitud razonable.

```
Ejemplo: Solicitud_Cambios, Tarea_ECS , Tipo_Usuario
```

2.2. Variables de Tipo Arreglo

En el caso de las definiciones de arreglos de elementos se declarará la variable con el prefijo de “ar”, el cual nos dará entender que se trata de una variable del tipo arreglo la cual contendrá de cero a más datos, según el tamaño declarado.

```
Ejemplo: arCambios, arElementos, arUsuario
```

3. Definición de Controles

3.1. Tipo de datos

Para poder determinar el nombre de un control dentro de cualquier aplicación de tipo C# .net, se procede a identificar el tipo al cual pertenece y la función que cumple dentro de la aplicación.

TIPO	DESCRIPCION	EJEMPLO	RANGO VALORES
byte	Representa a un número entero real positivo. Usa 1 byte.	byte a=30; byte b=126;	0 a 255
sbyte	Representa a un número entero real. Usa 1 byte.	sbyte a = 30; sbyte b = 126;	-128 a 127
short	Representa a un número entero real. Usa 2 bytes.	short a = 20; short b = 300;	-32.768 a 32.767
ushort	Representa a un número entero real positivo. Usa 2 bytes.	ushort a = 20; ushort b = 300;	0 a 65.535
int	Representa a un número entero real. Usa 4 bytes. Es el mas usado	int a=-1; int b=5;	-2.147.483.648 a 2.147.483.647
uint	Representa a un número entero real positivo. Usa 4 bytes. Es el mas usado	uint a = 1; uint b = 5;	0 a 4.294.967.295
long	Representa a un número entero real. Usa 8 bytes.	long a = 40000000; long b = 646334578;	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
ulong	Representa a un número entero real positivo. Usa 8 bytes.	ulong a = 40000000; ulong b = 646334578;	0 a 18.446.744.073.709.551.615
float	Representa a un número de coma flotante. Hay que añadir un F al final. Usa 4 bytes.	float a = 4.55555F; float b = -1.8521F;	±1.401298E−45 a ±3.402823E+38
double	Representa a un numero de coma flotante. Usa 4 bytes.	double a = 5.5; double b = 30;	±4.94065645841246E−324 a
decimal	Se almacenan como enteros de 128 bits (16 bytes) con signo escalados mediante una potencia variable de 10.	decimal a = 5; decimal b = 30;	−7.92281625142643375935439503 35 a
char	Representa un numero, letra o simbolo según la tabla ASCII. Usa 2 bytes	char a='A'; char b='V';	'\u0000' a '\uFFFF'
bool	Solo puede contener true o false. Se usa para condiciones lógicas. Usa 2 bytes	bool p = true; bool a = false;	Verdadero o falso

3.2. Prefijo para el Control

El prefijo del control será determinado mediante tres caracteres que estarán conformados por las consonantes más representativas del control, es así, por ejemplo; el control Button, estará asociado al prefijo btn.

3.3. Nombre descriptivo del Control

Formado por la descripción de la función que lleva a cabo el control, esta debe ser descrita en forma específica y clara.

Tipo de control	Prefijo	Ejemplo
Label	`@Html.LabelFor(x => x.<NombreDeVariable>)`	`@Html.LabelFor(x => x.Nombre)`
TextBox	`@Html.TextBoxFor(x => x.<NombreDeVariable>, new { @class = "form-control" })`	`@Html.TextBoxFor(x => x.Nombre, new { @class = "form-control" })`
ValidaciónMessage	`@Html.ValidationMessageFor(x => x.<NombreDeVariable> , null, new { @class = "label label-danger" })`	`@Html.ValidationMessageFor(x => x.<NombreDeVariable> , null, new { @class = "label label-danger" })`
Button	`<button type="submit" class="btn btn-primary"><NombreDelBoton></button>`	`<button type="submit" class="btn btn-primary">GUARDAR</button>`

3.4. Declaración de variables, atributos y objetos

Se debe declarar una variable por línea, y en el caso de C# asp.net es necesario declarar el tipo de variable antes de una variable.

Título	Descripción
Sintaxis	[Nombre de la Variable] =”valor”
Descripción	Todas las variables o atributo tendrán una longitud máxima de 30 caracteres. El nombre de la variable puede incluir más de un sustantivo los cuales se escribirán separados mediante un sub-guión. Si se tuvieran variables que puedan tomar nombres iguales, se le agregará un número asociado (si está dentro de un mismo método será correlativo).
Observaciones	En la declaración de variables o atributos no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	nombre=”Id_proyecto”

3.5. Declaración de clases

Título	Descripción
Sintaxis	class [Nombre de Clase]
Descripción	El nombre de las clases tendrá una longitud máxima de 30 caracteres y las primeras letras de todas las palabras estarán en letra minuscula. Tipo se refiere a si la clase será: private, public o protected.
Observaciones	En la declaración de clases no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	**class** Docente: Indica una clase Docente

3.6. Declaración de métodos

Título	Descripción
Sintaxis	public [nombreMetodo]():
Descripción	El nombre del método constará hasta de 25 caracteres. La primera letra de la primera palabra del nombre será escrita en Mayúscula y las siguientes palabras empezarán con letra mayúscula.
Observaciones	En la declaración de métodos no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	public void Guardar(): Indica un método Guardar, en este caso puede ser de cualquier tipo

3.7. Declaración de funciones

Título	Descripción
Sintaxis	public [nombreDeLaClase] [nombreMetodo]():
Descripción	El nombre del método constará hasta de 25 caracteres. La primera letra de la primera palabra del nombre será escrita en Mayúscula y las siguientes palabras empezarán con letra mayúscula.
Observaciones	En la declaración de métodos no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	public Cronograma Obtener() Indica un método Obtener, en este caso puede ser de cualquier tipo

3.8. Control de versiones de código fuente

Cada modificación realizada será guardada en un repositorio, en este caso se utiliza GitHub.

Título	Descripción
Formato	NombreProyecto
Descripción	git clone “[linkRepositorio]” -> para clonar el código en su última versión git add . -> para agregar todos los cambios al directorio local de git git commit -m “[descripcionCommit]” -> para guardar el cambio en el código git push -u origin [rama] -> ejecutar este código para subir los cambios a la rama.

4. Clases.

El nombre de las clases debe ser auto descriptivo de manera que no se requiera, en lo posible, entrar al código de la función para saber qué es lo que realiza.

Nota:

- No se hará uso de los caracteres: Espacio en blanco " ", Carácter de subrayado "_".

4.1. Definicion de clases

```
public partial class Usuario
{
        ...
}
```

5. Métodos, Procedimientos y Funciones definidos por el Usuario.

El nombre de las funciones y procedimientos van a ser descriptivos, según el estándar de programación lo apropiado para esta etapa seria utilizar un verbo que describa la acción realizada seguidamente por un sustantivo, en este caso sería:

➔ GUARDAR

public ActionResult Guardar([NombreDeLaClase] model)

➔ AGREGAR

public ActionResult Agregar(int id = 0)

➔ LISTAR

public ActionResult Index([TipoDeVariable] [VariableRecibida])

➔ ELIMINAR

public ActionResult Eliminar([TipoDeVariable] [VariableRecibida])

Nota:

- No se hará uso de los caracteres: Espacio en blanco " ", Carácter de subrayado "_".

5.1. Definición de métodos CRUD en **Controller**

- **Método Guardar**

```
public ActionResult Guardar([NombreDeLaClase] model)
        {
            if (ModelState.IsValid)
            {
                model.Guardar();
                return Redirect("~/[NombreDeLaDireccion]/index");
            }
            else
            {
                return View("~/[NombreDeLaDireccion]/Agregar");
            }
        }
```

- **Método Agregar**

```
public ActionResult Agregar(int id = 0)
{
  //Realizar esto si proviene de una tabla relacional,sirve para identificar
 ViewBag.[NombreDeLaTablaRelacionada] = bj[NombreDeLaTablaRelacionada].Listar();

 return View(id == 0 ? new [NombreDeLaTabla]() : obj[NombreDeLaTabla].Obtener(id));

}
```

- **Método Listar**

```
public ActionResult Index(string criterio)
        {
            if (criterio == null || criterio == "")
            {
                return View(obj[NombreDeLaTabla].Listar());
            }
            else
            {
                return View(obj[NombreDeLaTabla].Buscar(criterio));
            }

                }
```

- **Método Eliminar**

```
public ActionResult Eliminar(int id)
        {
            obj[NombreDeLaTabla].[IDDeLaTabla]= id;
            obj[NombreDeLaTabla].Eliminar();
            return Redirect("~/[NombreDeLaDireccion] ");
        }
```

5.2. Definición de métodos CRUD en **Model**

- **Método Guardar**

```
public void Guardar()
        {
            try
            {
                using (var db = new ModelGCS())
                {
                    if (this. [IDDeLaTabla]> 0)
                    {
                        db.Entry(this).State = EntityState.Modified;
                    }
                    else
                    {
                        db.Entry(this).State = EntityState.Added;
                    }
                    db.SaveChanges();

                }
            }
            catch (Exception)
            {
                throw;
            }
        }
```

- **Método Agregar**

```
public [NombreDeLaTabla] Obtener(int id)
        {
            var [NombreDeLaTablaEnPlural] = new [NombreDeLaTabla]();
            try
            {
                using (var db = new ModelGCS())
                {
                    [NombreDeLaTablaEnPlural] = db.
[NombreDeLaTabla].Include("[NombreDeLaTablaRelacional]").Where(x => x. [IDDeLaTablaRelacional]==
id)
                                        .SingleOrDefault();
                }
            }
            catch (Exception)
            {
                throw;
            }

            return [NombreDeLaTablaEnPlural];
        }
```

- **Método Listar**

```
public List< [NombreDeLaTabla] > Listar()
        {
            var [NombreDeLaTablaEnPlural]= new List<Usuario>();
            try
            {
                using (var db = new ModelGCS())
                {
                                [NombreDeLaTablaEnPlural] = db. [NombreDeLaTabla].
    Include("[NombreDeLaTablaRelacional]").ToList();
                }
            }
            catch (Exception)
            {
                throw;
            }

            return [NombreDeLaTablaEnPlural];

        }
```

- **Método Eliminar**

```
public void Eliminar()
        {
            try
            {
                using (var db = new ModelGCS())
                {

                    db.Entry(this).State = EntityState.Deleted;

                    db.SaveChanges();

                }
            }
            catch (Exception)
            {
                throw;
            }
        }
```

- **Método Buscar**

```
public List<[NombreDeLaTabla]> Buscar(string criterio)
       {
           var [NombreDeLaTablaEnPlural]= new List<[NombreDeLaTabla]>();

           try
           {
               using (var db = new ModelGCS())
               {
                   [NombreDeLaTablaEnPlural] = db.
[NombreDeLaTabla].Include("[NombreDeLaTablaRelacional]").Where(x => x.
[CriterioDeLaTablaRelacional].Contains(criterio) .ToList();

               }
           }
           catch (Exception)
           {
               throw;
           }

           return [NombreDeLaTablaEnPlural];

       }
```

5.3. Definición de TryCatch

```
try
 {

 }
 catch (Exception)
 {
     throw;
 }
```

5.4. Definición de un For

```
int count = 0;
 for (int i = 0; i <= [NombreDelAlmacenDeDatosl].count; i++)
 {
    count = count + 1;
 }
```

5.5. Definición de un Foreach

```
@foreach (var m in Model)
        {

        }
```

6. Beneficios

- Es la mejor forma de preservar el conocimiento y la experiencia.
- Proveen una forma de medir el desempeño.
- Muestran la relación entre causas (acciones) y efecto (resultado).
- Suministran una base para el mantenimiento y mejoramiento de la forma de hacer el trabajo.
- Proporcionan una base para el entrenamiento.
- Proveen una base para diagnóstico y auditoria.
- Proveen medios para prevenir la recurrencia de errores
- Minimizan la variación.

7. Conclusiones

- Que los miembros del proceso participen en la estandarización.
- Que el personal involucrado reciba capacitación en el estándar.
- Que el estándar represente la forma más fácil, segura y mejor de hacer un trabajo.

https://docs.microsoft.com/es-es/dotnet/csharp/fundamentals/types/