Estándar de Programación .Net

1. Variables

1.1 Se deberá utilizar la notación **lowerCamelCase** la cual se especifica como un conjunto de palabras compuestas de las cuales la primera palabra empieza todo en minúsculas, las palabras posteriores iniciarán con su primera letra en mayúscula y el resto en minúscula.

Ejemplo:

Incorrecto: ColorFondo, _nombreUsuario, tIpoGanado

Correcto: colorFondo, nombreUsuario, tipoGanado

1.2 No utilizar prefijos para las variables para hacer distinciones, como tipos de datos.

Ejemplo:

Incorrecto: sCodigo, iContador, dPago

Correcto: codigo, contador, pago

1.3 No utilizar caracteres especiales en las variables (Acentos, Ñ, etc.)

Ejemplo:

Incorrecto: organización, código, años

Correcto: *organizacion*, codigo, anios(edad)

2. Métodos

2.1 Se deberá utilizar la notación **PascalCase** la cual se especifica como un conjunto de palabras compuestas donde cada palabra deberá ser de tipo capital, es decir, la primera letra de cada palabras deberá estar en mayúsculas y el resto en minúsculas.

Ejemplo:

Incorrecto: obtenerTiposGanado, OBTENERusuarioPorID, _GuardarCorral

Correcto: ObtenerTiposGanado, ObtenerUsuarioPorId, GuardarCorral

2.2 No utilizar más de dos parámetros en un método, en caso de que se requiera, generar una estructura o clase la cual permita encapsular todos los parámetros necesarios y enviar dicha estructura o clase como único parámetro. El manejo de muchos parámetros hace más difíciles de leer los métodos.

Ejemplo:

Incorrecto:

```
public decimal CalcularPesoPromedioTipoGanado(TipoGanado tipo, int cantidadGanado, decimal pesoMinimo, decimal pesoMaximo)
{
    //Logica del metodo
}
```

Correcto:

```
ParametroPesoPromedioTipoGanado parametroPesoPromedio = new
ParametroPesoPromedioTipoGanado();
parametroPesoPromedio.TipoGanado = tipoGanado;
parametroPesoPromedio.CantidadGanado = cantidadGanado;
parametroPesoPromedio.PesoMinimo = pesoMinimo;
parametroPesoPromedio.PesoMaximo = pesoMaximo;
decimal pesoPromedio = CalcularPesoPromedioTipoGanado(parametroPesoPromedio);

public decimal CalcularPesoPromedioTipoGanado(ParametroPesoPromedioTipoGanado
parametroPesoPromedioTipoGanado)
{
        //Logica del metodo
}
```

2.3 Generar métodos de responsabilidad única, métodos que cumplan una sola función, y de ser necesario, llamar a otros métodos de responsabilidad única, para evitar métodos multitarea, de esta manera será más fácil dar nombre a un método e identificarlo para mantenimiento. Por lo tanto es muy importante evitar escribir métodos con más de 45 líneas de código, la finalidad es mantener un código legible, fácil de identificar, que se pueda leer en un solo bloque de pantalla, y que no sea necesario hacer mucho desplazamiento para leerlo.

Ejemplo:

Incorrecto:

```
public decimal CalcularTotal(decimal cantidad, decimal precio)
{
   decimal subTotal = cantidad*precio;
   decimal iva = subTotal*Configuracion.PorcentajeIVA;
   return subTotal*iva;
}
```

Correcto:

```
public decimal CalcularTotal(decimal cantidad, decimal precio)
{
   decimal subTotal = cantidad*precio;
   decimal iva = CalcularIVA(subTotal);
   return subTotal*iva;
}

public decimal CalcularIVA(decimal importe)
{
   return importe*Configuracion.PorcentajeIVA;
}
```

3. Clases

3.1 Se deberá utilizar la notación **PascalCase** al igual que en la nomenclatura para métodos.

Ejemplo:

Incorrecto: alumno, Tipoganado, _Cliente
Correcto: Alumno, TipoGanado, Cliente

3.2 Evitar tener clases muy grandes con métodos que no son exclusivamente de la misma, en caso de necesitar métodos que no son exclusiva de la clase, estos se deberán invocar de su clase correspondiente, procura crear clases donde sus métodos estén orientados a un fin específico, esto nos permitirá hacer un mejor reusó de las mismas. Al delimitar las funciones de una clase permite el reusó de una manera más eficiente y a su vez facilita el mantenimiento al existir un punto común donde se pueda corregir un problema.

Ejemplo:

Incorrecto:

```
public class ControlEscolar
{
  private CapaDatos.AccesoDatos.AlumnoDataAccess alumnoDA = new AlumnoDataAccess();

  public void AltaAlumno(CapaBase.Entidades.Alumno alumno)
  {
    int idAlumno = alumnoDA.GuardarAlumno(alumno);

    if (idAlumno <= 0)
      throw new CapaBase.ExcepcionBase(ObtenerMensajeRecurso("ExcepcionAltaAlumno"));
  }

  public string ObtenerMensajeRecurso(string mensajeId)
  {
    return string.Empty;
  }
}
```

Correcto:

```
public class MensajesRecurso
{
  public string ObtenerMensajeRecurso(string mensajeId)
  {
    return string.Empty;
  }
}
```

```
public class ControlEscolar
{
   private CapaDatos.AccesoDatos.AlumnoDataAccess alumnoDA = new
CapaDatos.AccesoDatos.AlumnoDataAccess();
   private CapaBase.Utilerias.MensajesRecurso mensajeRecursos = new
CapaBase.Utilerias.MensajesRecurso();

   public void AltaAlumno(CapaBase.Entidades.Alumno alumno)
   {
      int idAlumno = alumnoDA.GuardarAlumno(alumno);

      if (idAlumno <= 0)
          {
             string  mensaje = mensajeRecursos.ObtenerMensajeRecurso("ExcepcionAltaAlumno");
         throw new CapaBase.ExcepcionBase(mensaje);
          }
   }
}
```

3.3 Utiliza siempre y correctamente los modificadores de acceso, no es necesario marcar todo como público cuando un elemento será de uso interno de la clase, ni es necesario marcarlo como protected, si no existirá una herencia que haga uso de este elemento. Una clase debe exponer propiedades únicamente, las variables deben permanecer como elemento de uso exclusivo por la clase por medio del ámbito privado a excepción de los eventos y los Delegados, la correcta definición de ámbitos permite a la clase darle un uso más consistente manejando abstracción en los elementos que se requiere.

Modificador de Acceso	**Uso**
private	Se utiliza cuando se requiere que el miembro solo sea de uso exclusivo de la clase donde se declara, normalmente se utiliza en variables de control, o elementos que serán usados durante todo el ámbito de la clase.
public	Se utiliza cuando el miembro sea accesado por cualquier otro miembro independientemente si es parte del ensamblado o no, normalmente se utiliza en propiedades o métodos que necesitan ser invocados desde cualquier capa.
protected	Se utiliza cuando se requiere que un miembro solo sea conocido por la clase que lo declara y sus derivados, normalmente se utiliza cuando necesitas establecer un control entre el padre y sus hijos, o

Modificador de Acceso	Uso
	cuando quieres que algunos métodos solo sean utilizados por sus hijos.
internal	Se utiliza cuando un miembro solo debe ser accesado por otros elementos dentro del mismo ensamblado y se quiere evitar que otros elementos que no son parte del ensamblado lo utilicen, normalmente es usado para abstracción de capas, para respetar el canal de comunicación, o para que clases con fines técnicos específicos no sean utilizados por otras capas.
external	Se utiliza cuando se requiere hacer un enlace entre un miembro de una clase y una acción que es parte de un componente externo del tipo API, un ejemplo seria la llamada de un método del Kernel del sistema operativo como el inicio de sesión para suplantación de identidad a demanda.

Ejemplo:

incorrecto:

```
public class ControlEscolar
{
  ...

  public bool permiteGuardarAlumno;
  bool permiteModificarAlumno;

  public ControlEscolar()
  {
    permiteGuardarAlumno = Configuracion.PermiteGuardarAltasAlumno;
    permiteModificarAlumno = Configuracion.PermiteModificacionAltasAlumno;
  }

  public void AltaAlumno(CapaBase.Entidades.Alumno alumno)
  {
    int idAlumno = 0;
    if (permiteGuardarAlumno)
    {
        idAlumno = alumnoDA.GuardarAlumno(alumno);
    }
```

```
        ...
    }
}
```

correcto:

```
public class ControlEscolar
{
    ...

    private bool permiteGuardarAlumno;
    private bool permiteModificarAlumno;

    public ControlEscolar()
    {
        permiteGuardarAlumno = Configuracion.PermiteGuardarAltasAlumno;
        permiteModificarAlumno = Configuracion.PermiteModificacionAltasAlumno;
    }

    public void AltaAlumno(CapaBase.Entidades.Alumno alumno)
    {
        int idAlumno = 0;
        if (permiteGuardarAlumno)
        {
            idAlumno = alumnoDA.GuardarAlumno(alumno);
        }

            ...
    }
}
```

4. Interfaces

4.1 Se deberá utilizar el prefijo "I" (i latina) antes del nombre de la interface. Esta es más que nada un estándar para poder identificar las interfaces. Cuando se definan interfaces, se deberá definir una clase controladora que les de uso.

Ejemplo:

Incorrecto: interfacePaginador, paginadorInterface, I_Paginador, iPaginador

Correcto: IPaginador, IIntegrador

5. Generales

Los siguientes puntos se deberán cuidar para las nomenclaturas antes mencionadas.

5.1 Se recomienda evitar el uso de palabras como “de, la, las, los”, etc., esto permitirá tener nombres de variables más cortos, solo en caso de que sea necesario para el entendimiento de un nombre de variable se podrán utilizar.

Ejemplo:

incorrecto: ObtenerTipoDeGanado, factorDeRiesgo
correcto: ObtenerTipoGanado, factorRiesgo

5.2 Usar palabras que sean descriptivas y que dejen en claro el uso, acción o representación de la variable, clase o método. El nombre debe ser de responsabilidad única y no deberá denotar más de dos acciones.

Ejemplo:

incorrecto:

```
public TipoGanado Obtener(int id)
{
  //Escribir logica aquí
}

public TipoGanado ValidarYObtenerGanado(int id)
{
  //Escribir logica aquí
}
```

correcto:

```
public TipoGanado ObtenerTipoGanadoPorID(int id)
{
  //Escribir lógica aquí
}

public bool ValidarGanado(int id)
{
  //Escribir lógica aquí
}

public TipoGanado ObtenerGanado(int id)
{
  //Escribir lógica aquí
}
```

5.3 Evitar usar nombres de métodos genéricos para funciones específicas, solo si el método tiene una definición genérica utilizar un nombre genérico.

Ejemplo:

incorrecto:

```
public TipoGanado ObtenerPorID(int id)
{
   //Escribir lógica aquí
}
```

correcto:

```
public TipoGanado ObtenerTipoGanadoPorID(int id)
{
   //Escribir lógica aquí
}

public T ObtenerPorID<T>(int id)
{
   //Escribir lógica aquí
}
```

5.4 No utilizar abreviaciones, a excepción de cuando el nombre exceda los 30 caracteres de longitud, en este caso se abreviará en las palabras que tengan una clara abreviatura. Los nombres muy largos producen confusiones cuando existen varios métodos parecidos.

Ejemplo:

incorrecto: noEmpleado, ObtenerVerXTipoGanado

correcto: numeroEmpleado, ObtenerVerificacionXTipoGanado

5.5 Utilizar las llaves de ámbito para todo tipo de bloques, es decir, colocar la llave que abre '{' al inicio del bloque y la llave que cierra '}' al final del bloque, esto para todos los diferentes tipos de sentencias dentro de un ámbito como, if, for, foreach, switch, etc.

Ejemplo:

incorrecto:

```
if (idAlumno <= 0)
   throw new
CapaBase.ExcepcionBase(mensajeRecursos.ObtenerMensajeRecurso("ExcepcionAltaAlumno"));

for (int i = 0; i < cabezas.Count; i++)
   generarSalidaGanado(cabeza[i]);
```

correcto:

```
if (idAlumno <= 0)
{
```

```
   string mensaje = mensajeRecursos.ObtenerMensajeRecurso("ExcepcionAltaAlumno");
   throw new CapaBase.ExcepcionBase(mensaje);
}

for (int i = 0; i < cabezas.Count; i++)
{
   var cabeza = cabezas[i];
   if (cabeza != null)
   {
      generarSalidaGanado(cabeza);
   }
}
```

5.6 Siempre hay que realizar validaciones de tipos de datos donde se pueda provocar una excepción, es decir, validar valores nulos si un elemento va ser utilizado en alguno de sus miembros, validar valores numéricos donde se pueda provocar errores de división por 0 o desbordamiento, longitud de colecciones donde se vaya a utilizar algún elemento de la misma.

Ejemplo:

incorrecto:

```
for (int i = 0; i < cabezas.Count; i++)
   generarSalidaGanado(cabeza[i]);

decimal cabezas = obtenerCabezasGanadoPorTipoGanado(tipoGanado);
decimal pesoPromedio = pesoTotal/cabezas;
```

correcto:

```
var cabeza = cabezas[i];
if (cabeza != null)
{
   generarSalidaGanado(cabeza);
}

decimal cabezas = obtenerCabezasGanadoPorTipoGanado(tipoGanado);
decimal pesoPromedio = 0;
if (cabezas > 0)
{
   pesoPromedio = pesoTotal/cabezas;
}
```

5.7 Utiliza las constantes que el framework ofrece en vez de constantes de código duro, es decir, puedes utilizar String.Empty en vez de una cadena vacía “”, o Math.PI en vez de 3.141592, las constantes ofrecidas por el .Net vienen optimizadas ya que son identificadas por el framework, en cambio un numero o cadena para el framework implica un cálculo.

Ejemplo:

incorrecto:

```
string nombreCompleto = "";
double pi = 3.141592;

double area = pi*(radio*2);

if (nombreCompleto == "")
{
  ...
}
```

correcto:

```
string nombreCompleto = string.Empty;

double area = Math.PI*Math.Pow(radio, 2);

if (string.IsNullOrWhiteSpace(nombreCompleto))
{
  ...
}
```

5.8 Utiliza valores de enumeradores en vez de valores constantes, de esta manera es más legible lo que se quiere hacer, y en caso de que un valor cambio, solo cambiaria en el enumerador y no en todo el código donde haya sido utilizado.

Ejemplo:

incorrecto:

```
if (alumno.Estatus == 0)
{
  ...
}
```

correcto:

```
if (alumno.Estatus == CapaBase.Enumeradores.Estatus.Inactivo)
{
  ...
}
```

5.9 No utilizar código duro (o fijo) para las diferentes accesos de entrada y salida o escritura y lectura, siempre utiliza variables que se puedan obtener por una configuración y que no impliquen una compilación innecesaria cuando el acceso cambie, un sistema debe ser susceptible al cambio cuando la dinámica del negocio cambie, y no cuando una ruta o valor cambie.

Ejemplo:

incorrecto:

```
var rutaPolizas = "\\\\Servidor\\Contabilidad\\Polizas";
var urlServicios = "http://portal.sukarne.com/servicios";
var unidadRespaldo = "F:";
```

correcto:

```
var rutaPolizas = Configuracion.RutaPolizas;
var urlServicios = Configuracion.UrlServicios;
var unidadRespaldo = Configuracion.UnidadRespaldos;
```

5.10 Declare variables en el ámbito donde serán utilizadas, en caso de que una variable esté condicionada a ser asignada declárela antes de hacer la asignación, utilice todas las variables que declare. Cuando declare una variable siempre asígnele un valor de inicio, si la variable es declarada en una clase utilice el constructor para asignar valor, si tiene más de un constructor, genere un método de inicialización que sea llamado por cada constructor, en caso de ser una clase esta deberá ser iniciada con una instancia de la clase o en su defecto con null. Cuando declare variables escriba una declaración por línea de código.

Ejemplo:

incorrecto:

```
public class ClaseFOO
{
   private bool puedeGuardar;
   private ClaseDos claseDos = null;
   private ClaseTres claseTres;

   public ClaseFOO()
   {

   }

   public void MetodoUno()
   {
      decimal pesoPromedio;
      puedeGuardar = Configuracion.PuedeGuardar;
      if (puedeGuardar)
      {

      }
   }

   public void MetodoDos()
```

```
  {
    ...
  }
}
```

correcto:

```
public class ClaseFOO
{
  private ClaseDos claseDos;
  private ClaseTres claseTres;

  public ClaseFOO()
  {
    claseDos = null;
    claseTres = new ClaseTres();
  }

  public void MetodoUno()
  {
    decimal pesoPromedio;
    bool puedeGuardar = Configuracion.PuedeGuardar;
    if (puedeGuardar)
    {
      claseDos = new ClaseDos();
      claseDos.Accion();
      claseTres.Accion();
    }
  }

  public void MetodoDos()
  {
    if (Configuracion.EsFactible)
    {
      claseDos = new ClaseDos();
      claseDos.Accion();
    }
    claseTres.Accion();
  }
}
```

5.11 Evita usar operadores ternarios para operaciones complejas, ya que los operadores ternarios son poco legibles, puede causar confusión al momento de interpretar lo que se requiere hacer en un método determinado.

Ejemplo:

incorrecto:

```
var calcularFoo = (valor1 > 0 ? valor1 : valor2 ?? 2);
var puedeAccion = (valor3 ?? valor4) ? valor5 > 100 ? true : false : valor6;
```

correcto:

```
decimal calcularFoo = valor1;
if (valor1 <= 0)
{
   calcularFoo = valor2 ?? 2;
}

bool puedeAccion = false;
if (valor3 ?? valor4)
{
   if (valor5 > 100)
   {
      puedeAccion = true;
   }
}
else
{
   puedeAccion = valor6;
}
```

5.12 Usa la clase StringBuilder o StringFormat para concatenar cadenas. Estas están optimizadas para esta función además de que son más fáciles de dar mantenimiento.

Ejemplo:

incorrecto:

```
string mensajeCompleto = "";
if (string.IsNullOrWhiteSpace(nombre))
{
   mensajeCompleto += mensajeRecursos.ObtenerMensaje("NoNombre");
}
if (string.IsNullOrWhiteSpace(apellidoPaterno))
{
   mensajeCompleto += mensajeRecursos.ObtenerMensaje("NoApellidoPaterno");
}
if (string.IsNullOrWhiteSpace(apellidoMaterno))
```

```
{
   mensajeCompleto += mensajeRecursos.ObtenerMensaje("NoApellidoMaterno");
}

string nombreCompleto = apellidoPaterno + " " + apellidoMaterno + " " + nombre;
```

correcto:

```
StringBuilder mensajeError = new StringBuilder();
if(string.IsNullOrWhiteSpace(nombre))
{
   mensajeError.Append(mensajeRecursos.ObtenerMensaje("NoNombre"));
}
if (string.IsNullOrWhiteSpace(apellidoPaterno))
{
   mensajeError.Append(mensajeRecursos.ObtenerMensaje("NoApellidoPaterno"));
}
if (string.IsNullOrWhiteSpace(apellidoMaterno))
{
   mensajeError.Append(mensajeRecursos.ObtenerMensaje("NoApellidoMaterno"));
}
string mensajeCompleto = mensajeError.ToString();

string nombreCompleto = string.Format("{0} {1} {2}", apellidoPaterno, apellidoMaterno,
nombre);
```

5.13 No usar querys Dinámicos (Código Duro de SQL), ya que las tablas o sus campos pueden cambiar, una relación para efecto de información, y en ese caso se necesitara recompilar la solución y generar una nueva publicación. Para esto hacer llamados a SP's o si se necesita dinamismo cambiar a esquemas LinQ con EntityFramework o algún otro ORM que implemente la interfaz IQueryable.

Ejemplo:

incorrecto:

```
StringBuilder query = new StringBuilder("select a.codigo as codigoAlumno, a.nombre,
a.apellidoPaterno, a.apellidoMaterno, g.grupoId, g.codigo, m.materiaId, m.Descripcion from
Alumno a ");
query.AppendLine("inner join GrupoAlumno ga on a.alumnoId = ga.alumnoId");
query.AppendLine("inner join Grupo g on ga.grupoId = g.grupoId");
query.AppendLine("inner join Materia m on g.materiaId = m.materiaId");

if (alumnoID > 0 || grupoID > 0 || materiaID > 0)
{
   query.AppendLine("where");
```

```
   bool yaTieneCondicion = false;
   if (alumnoID > 0)
   {
      query.AppendFormat("alumnoId = {0}\r", alumnoID);
      yaTieneCondicion = true;
   }
   if (grupoID > 0)
   {
      if (yaTieneCondicion) query.Append("and ");
      query.AppendFormat("grupoId = {0}\r", grupoID);
      yaTieneCondicion = true;
   }
   if (materiaID > 0)
   {
      if (yaTieneCondicion) query.Append("and ");
      query.AppendFormat("materiaID = {0}\r", materiaID);
      yaTieneCondicion = true;
   }
}

var alumnos = alumnoDA.EjecutarConsulta(query.ToString());
```

correcto:

```
var alumnos = alumnoDA.ObtenerAlumnos(alumnoId, grupoId, materiaId);

var query = from a in dataAccessor.Alumnos
         join ga in dataAccessor.GrupoAlumnos on a.AlumnoId equals ga.AlumnoId
         join g in dataAccessor.Grupos on ga.GrupoId equals g.GrupoId
         join m in dataAccessor.Materias on g.MateriaId equals m.MateriaId
         select new
         {
            codigoAlumno = a.Codigo,
            a.Nombre,
            a.ApellidoPaterno,
            a.ApellidoMaterno,
            g.GrupoId,
            g.Codigo,
            m.MateriaId,
            m.Descripcion
         };
if (alumnoID > 0)
{
```

```
    query = query.Where(e => e.AlumnoId == alumnoId);
}
if (grupoID > 0)
{
    query = query.Where(e => e.GrupoID == grupoID);
}
if (materiaID > 0)
{
    query = query.Where(e => e.MateriaID == materiaID);
}
var alumnos = query.ToList();
```

5.14 Evita instanciar e utilizar los miembros de una clase en una sola línea, hay que hacerlo por pasos, primero se realiza la instancia del objeto y posteriormente se utilizan los miembros. Ya que esto se presta a malas prácticas como en la inicialización de objetos hacer este tipo de llamadas para asignarle valor a una propiedad lo que produce un desempeño bajo en la aplicación.

Ejemplo:

incorrecto:

```
var alumnos = new AlumnosDataAccess().ObtenerTodosAlumnos();
```

correcto:

```
var alumnoDA = new AlumnosDataAccess();
List<Alumno> alumnos = alumnoDA.ObtenerTodosAlumnos();
```

5.15 Utiliza var para el tipo de datos que este implícito al lado derecho. Ya que el tipo var es implícito en ocasiones puede generar conflictos en programadores menos experimentados, por eso es recomendable en ambientes donde la interpretación del tipo de dato sea clara. Un error común es que al ser implícito el tipo de datos se pudiera generar un error no correcto, es decir, el compilador puede marcar como error la inexistencia de una propiedad por haber cambiado el tipo resultante de un método invocado, cuando el error real es que el tipo de dato esperado no es el deseado.

Ejemplo:

incorrecto:

```
var usuario = getUsuario(id);
```

correcto:

```
var usuario = new Usuario();
usuario = getUsuario(id);
```

5.16 Poner comentarios en todos los métodos y declaraciones de propiedades, esto es importante para que otros programadores puedan saber con claridad lo que hace un método, y para que con generadores de documentación automática se pueda generar un manual técnico, que sirva de apoyo a todos los desarrolladores, es importante usar en declaraciones de métodos, propiedades,

clases comentarios de tipo "///" y para comentar acciones entre líneas de código usar "//", evitar el uso de los comentarios multilineas "/* ... */"

Ejemplo:

incorrecto:

```
public void Malo()
{
   /* Este metodo es para explicar las malas practicas
    * que se cometen en el desarrollo de aplicaciones
    */
}
```

correcto:

```
/// <summary>
/// Determina buenas practicas de desarrollo
/// </summary>
public void Bueno()
{
   // Este metodo es para explicar las buenas practicas
   // que se deben de seguir al momento de desarrollar aplicaciones.
}
```

5.17 Evitar poner comparaciones redundantes, solo se le da carga extra al compilador y no es anda optimo hacer una comparación "true == true".

Ejemplo:

incorrecto:

```
bool puedeAccesar = true;
if (puedeAccesar == true)
{
   ...
}
```

correcto:

```
bool puedeAccesar = true;
if(puedeAccesar)
{
   ...
}
```

5.18 Utilizar el "Object Initializers" para simplificar los llenados de los objetos.

Ejemplo:

```
var alumno = new Alumno()
        {
          AlumnoId = alumnoId,
```

```
            Nombre = nombre,
            ApellidoPaterno = apellidoPaterno,
            ApellidoMaterno = apellidoMaterno,
            Codigo = codigo
        };
```

5.19 Evitar usar namespaces en código que ya haya sido declarado como directiva. Si los elementos del namespace van a ser muy poco utilizados evitar declarar el namespace con la directiva using, si los elementos del namespace van a ser muy utilizados declarar el namespaces con la directiva using, si el nombre de una clase es ambiguo entre dos namespaces utilizar alias en las declaraciones using.

Ejemplo:

incorrecto:

```
using System.Text;

namespace CapaNegocio.Negocio
{
   public class ClaseFOO
   {
      private System.Text.Encoding encoding;
      public void Malo()
      {
         encoding = System.Text.Encoding.UTF8;
      }
```

correcto:

```
using System;
namespace CapaNegocio.Negocio
{
   public class ClaseFOO
   {
      private System.Text.Encoding encoding;
      public void Malo()
      {
         encoding = System.Text.Encoding.UTF8;
      }
/*****************************************************************************/
using System;
using System.Text;
namespace CapaNegocio.Negocio
{
   public class ClaseFOO
```

```
{
    private Encoding encoding;
    public void Malo()
    {
        encoding = Encoding.UTF8;
    }
```

5.20 Abrir transacciones en la capa de Lógica de negocios con TransactionScope, así como terminarlas siempre al final, de esta manera aseguramos la transaccionabilidad de un conjunto de acciones.

```
internal class EmbarqueBL
{
  internal int GuardarEmbarque(EmbarqueInfo embarqueInfo)
  {
    try
    {
      var embarqueDAL = new EmbarqueDAL();
      using (var transaction = new TransactionScope())
      {
        //Persistencia de datos

        foreach (var costo in listaCostos)
        {
                //Persistencia de datos
        }

        //Persistencia de datos en otros metodos del DAL relacionado u otros BL

        transaction.Complete();
        return resultadoOperacion;
      }
    }
    catch (ExcepcionGenerica)
    {
      throw;
    }
    catch (Exception ex)
    {
      throw new ExcepcionDesconocida(MethodBase.GetCurrentMethod(), ex);
    }
  }
```

```
}
```

6. Interfaz de usuario

Los siguientes puntos están enfocados a la interfaz de usuario, se entiende por interfaz de usuario al código de la pantalla visual, ya sea .xaml (WPF) o .aspx (Web), y su vista de código asociado, .xaml.cs|.xaml.vb (WPF) y .aspx.cs|.aspx.vb (Web), de igual manera son los controles de usuario y componentes involucrados en la presentación grafica del sistema.

6.1 Para los controles utilizar un prefijo que relaciona al tipo de control utilizado, para esto utilizar la siguiente tabla de prefijos de controles.

Prefijo	Tipo de Control	Ejemplo
txt	TextBox	txtNombreUsuario
lbl	Label	LblNombreCompleto
dg	DataGrid	dgListaEmpleados
btn	Button	btnGuardarEmpleado
gpb	GroupBox	gpoDatosAdministracion
stp	StackPanel	stpSeccionGeneral
img	Image	imgLogotipo
lst	ListBox	lstTiposGanado
uc	<cualquier control de usuario>	ucPedimentoEspecial
cbo	ComboBox	cboSexo
tbl	TextBlock	tblMensajePrimario
dtp	DatePicker	dtpFechaIngreso
ckb	CheckBox	ckbEsImpuesto
grd	Grid	grdPrincipal

Prefijo	Tipo de Control	Ejemplo
tgb	ToggleButton	tgbPermiteEdicion

6.2 Utilizar archivos de recursos para todos aquellos textos que serán desplegados en la interfaz de usuario independientemente su presentación, (mensaje, texto, etiqueta, ventana emergente, etc.).

incorrecto:

```
[WPF]
<Label Content="Salida:" />

SkMessageBox.Show("Error al generar la salida.");

[Web]
<asp:Label ID="lblTitulo" runat="server">Salida:</asp:Label>
```

correcto:

```
[WPF]
<Label Content="{x:Static resx:Resources.SalidaIndividualGanado_lblSalida}" />

SkMessageBox.Show(Properties.Resources.SalidaIndividualGanado_MensajeErrorConsultarCausa
Salida);

[Web]
<asp:Label ID="lblTitulo" runat="server" meta:resourcekey="lblTituloResource1"></asp:Label>
```

6.3 En eventos declarados no escribir código, hacer llamada a una función que pueda ser reutilizada desde otros métodos.

incorrecto:

```
private void btnCancelar_Click(object sender, RoutedEventArgs e)
{
   MessageBoxResult respuestaInteface =
SkMessageBox.Show(Application.Current.Windows[ConstantesVista.WindowPrincipal],
      Properties.Resources.Cancelarcaptura_CorteGanado,
      MessageBoxButton.YesNo, MessageImage.Warning)
   if (respuestaInteface == MessageBoxResult.Yes)
   {
      _pesoTomado = false;
      Limpiar();
   }
}
```

correcto:

```
private void btnCancelar_Click(object sender, RoutedEventArgs e)
```

```
{
   CancelarSalidaIndividulaGanado();
}

private void CancelarSalidaIndividulaGanado()
{
   if (SkMessageBox.Show(Application.Current.Windows[ConstantesVista.WindowPrincipal],
         Properties.Resources.Cancelarcaptura_CorteGanado,
         MessageBoxButton.YesNo, MessageImage.Warning) == MessageBoxResult.Yes)
   {
      _pesoTomado = false;
      Limpiar();
   }
}
```

6.4 Para los proyectos web hacer uso de System.Web.Optimization del FrameWork 4 en adelante, esto funciona para poder agrupar todas las llamadas a los recursos estáticos (Js, css, ascx, etc.)
Ya que si se agregan manual cada uno de los recursos la maquina cliente, hace una petición por cada archivo, logrando así que se consuma más uso de red para descargar cada uno de los archivos, con Optimization, todos los recursos se agrupan en una sola petición, y a su vez son minificados, para que no consuman mucho uso de red.

incorrecto:

```
<head id="headEvaluacion" runat="server">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>SIAP - SuKarne <%: ConfigurationManager.AppSettings["version"] %></title>

    <link href="../assets/plugins/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <link href="../assets/plugins/bootstrap/css/bootstrap-responsive.min.css" rel="stylesheet" />
    <link href="../assets/plugins/font-awesome/css/font-awesome.min.css" rel="stylesheet" />
    <link href="../assets/css/style-metro.css" rel="stylesheet" />
    <link href="../assets/css/style.css" rel="stylesheet" type="text/css" />
    <link href="../assets/css/style-responsive.css" rel="stylesheet" />
    <link href="../assets/plugins/bootstrap/css/bootstrap-modal.css" rel="stylesheet" />
    <script src="../assets/plugins/jquery-1.7.1.min.js"></script>
    <script src="../assets/plugins/bootstrap-bootbox/js/bootbox.min.js"></script>
    <link href="../assets/css/media-queries.css" rel="stylesheet" />
    <link href="../assets/plugins/data-tables/DT_bootstrap.css" rel="stylesheet" />

    <script src="../assets/plugins/data-tables/jquery.dataTables.js"></script>
    <script src="../Scripts/EvaluacionPartida.js"></script>
    <script src="../Scripts/json2.js"></script>

    <script src="../assets/plugins/jquery-ui-1.10.1.custom.min.js"></script>
    <script src="../assets/plugins/bootstrap-modal/js/bootstrap-modal.js"></script>
    <script src="../assets/scripts/jquery-jtemplates.js"></script>
    <script src="../assets/plugins/data-tables/DT_bootstrap.js"></script>
    <script src="../assets/plugins/bootstrap-modal/js/bootstrap-modalmanager.js"></script>
    <script src="../assets/scripts/ui-modals.js"></script>
    <script src="../assets/scripts/app.js"></script>
```

correcto:

```
bundles.Add(new ScriptBundle("~/bundles/Scripts_Comunes").Include(
       "~/assets/plugins/jquery-1.7.1.min.js",
                          "~/Scripts/json2.js",
                          "~/assets/plugins/jquery-ui-1.10.1.custom.min.js",
                          "~/assets/plugins/bootstrap-modal/js/bootstrap-modal.js",
                          "~/assets/plugins/bootstrap/js/bootstrap.min.js",
                          "~/assets/plugins/bootstrap-bootbox/js/bootbox.min.js",
                          "~/assets/scripts/jquery-jtemplates.js",
                          "~/assets/plugins/bootstrap-modal/js/bootstrap-modalmanager.js",
                          "~/assets/scripts/ui-modals.js",
                          "~/assets/plugins/jquery.blockui.min.js",
                          "~/assets/scripts/app.js",
                          "~/assets/plugins/spin.js",
                          "~/assets/plugins/jquery.spin.js",
                          "~/assets/plugins/numericInput/jquery-numericInput.min.js",
                          "~/assets/plugins/jquery-linq/linq.js",
                          "~/assets/plugins/jquery-alphanumeric/jquery-alphanumeric.js",
                          "~/assets/plugins/jquery-inputmask/jquery.inputmask.bundle.min.js",
                          "~/assets/plugins/bootstrap-typeahead/typeahead.bundle.min.js",
                          "~/assets/plugins/jquery-timepicker/jquery.timepicker.min.js",
                          "~/assets/plugins/select2/select2.min.js",
                          "~/assets/plugins/data-tables/jquery.dataTables.min.js",
                          "~/assets/plugins/data-tables/DT_bootstrap.js",
                          "~/assets/plugins/jquery-slimscroll/jquery.slimscroll.min.js",
                          "~/assets/plugins/accounting/accounting.min.js"
                          ));
```

```
<asp:PlaceHolder ID="PlaceHolder1" runat="server">
    <%: Scripts.Render("~/bundles/Scripts_Comunes") %>
    <%: Styles.Render("~/bundles/Estilos_Comunes") %>
    <%: Scripts.Render("~/bundles/jscomunScript") %>
</asp:PlaceHolder>
```

6.5 No utilizar ni código JavaScript ni Estilos de CSS, en el HTML de la página, siempre se deben separar en archivo aparte.

Ejemplo:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>SIAP - SuKarne <%: ConfigurationManager.AppSettings["version"] %></title>

<asp:PlaceHolder ID="PlaceHolder1" runat="server">
    <%: Scripts.Render("~/bundles/Scripts_Comunes") %>
    <%: Styles.Render("~/bundles/Estilos_Comunes") %>
</asp:PlaceHolder>

<link href="../assets/plugins/font-awesome/css/font-awesome.min.css" rel="stylesheet" />

<style>
    .caja {
        border: 2px solid #c0c0c0;
        padding: 10px;
    }

    .seccionbotones {
        padding: 10px;
    }

    textarea {
        resize: none;
    }

    .tabladetectados {
        max-height: 240px;
        overflow: auto;
    }
</style>
```

6.6 En los proyectos de WPF, utilizar los controles contenedores para diseñar las pantallas (Grid, StackPanel, Canvas), nunca utilizar el Mouse para arrastrar los controles, ya que esto hace que sea muy complicado realizar cambios en el diseño además se tiene que configurar el Visual Studio para que se maneje una estructuración del código XAML.

Incorrecto:

```
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="656*"/>
    <ColumnDefinition Width="405*"/>
</Grid.ColumnDefinitions>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_Producto}" HorizontalAlignment="Left" Margin="35,20,0,0" VerticalAlignment=
"Top"/>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_CuentaProveedor}" HorizontalAlignment="Left" Margin="35,21,0,0"
VerticalAlignment="Top" Grid.Row="1"/>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_ClienteDivision}" HorizontalAlignment="Left" Margin="35,21,0,0"
VerticalAlignment="Top" Grid.Row="2"/>
<TextBox Name="txtProductoID" HorizontalAlignment="Left" Height="23" Margin="197,22,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top"
Width="120"/>
<TextBox Name="txtProductoDescripcion" HorizontalAlignment="Left" Height="23" Margin="331,22,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
Width="285"/>
<TextBox Name="txtCuentaProveedorID"  HorizontalAlignment="Left" Height="23" Margin="197,21,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
Width="120" Grid.Row="1"/>
<TextBox Name="txtCuentaProveedorDescripcion" HorizontalAlignment="Left" Height="23" Margin="331,21,0,0" TextWrapping="Wrap" VerticalAlignment
="Top" Width="285" Grid.Row="1"/>
<TextBox Name="txtClienteDivisionID" HorizontalAlignment="Left" Height="23" Margin="197,21,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
Width="120" Grid.Row="2"/>
<TextBox Name="txtClienteDivisionDescripcion" HorizontalAlignment="Left" Height="23" Margin="331,21,0,0" TextWrapping="Wrap" VerticalAlignment
="Top" Width="285" Grid.Row="2"/>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_Lote}" HorizontalAlignment="Right" Margin="0,18,291,0" VerticalAlignment="Top"
 Grid.Column="1"/>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_Cantidad}" HorizontalAlignment="Right" Margin="0,17,262,0" VerticalAlignment=
"Top" Grid.Column="1" Grid.Row="1"/>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_Importe}" HorizontalAlignment="Right" Margin="0,19,271,0" VerticalAlignment=
"Top" Grid.Column="1" Grid.Row="2"/>
<TextBox Name="txtLote" Grid.Column="1" HorizontalAlignment="Right" Height="23" Margin="0,20,67,0" TextWrapping="Wrap" VerticalAlignment="Top"
```

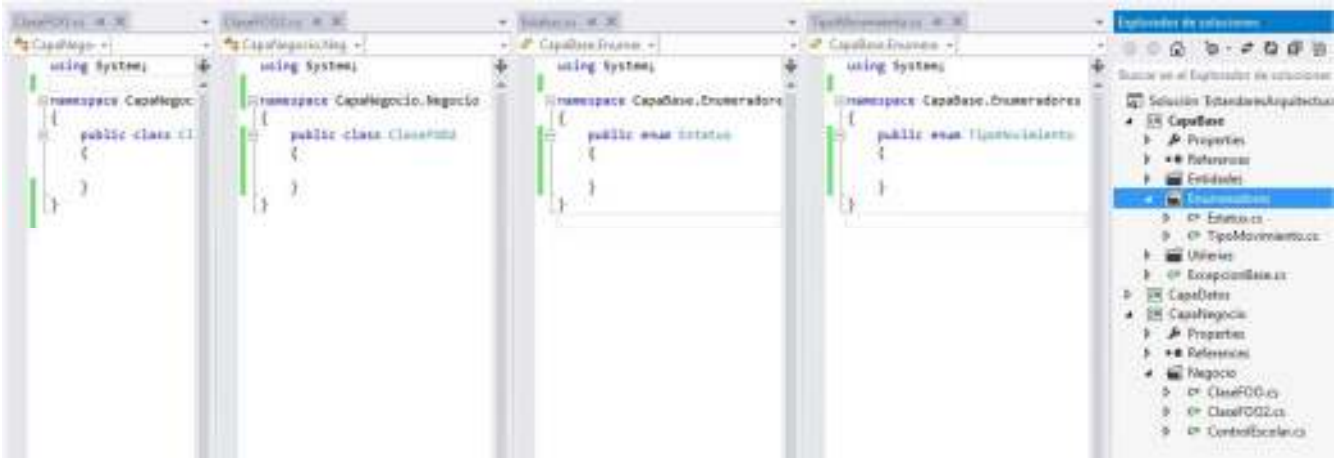
Correcto:

```
</Grid.ColumnDefinitions>
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_Producto}"
       HorizontalAlignment="Left"
       Grid.Row="0"
       Grid.Column="0"
       VerticalAlignment="Top" />
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_CuentaProveedor}"
       HorizontalAlignment="Left"
       Grid.Row="1"
       Grid.Column="0"
       VerticalAlignment="Top" />
<Label Content="{x:Static resx:Resources.CancelarMovimiento_Lbl_ClienteDivision}"
       HorizontalAlignment="Left"
       VerticalAlignment="Top"
       Grid.Row="2"
       Grid.Column="0" />
<TextBox Name="txtProductoID"
         HorizontalAlignment="Left"
         Height="23"
         Grid.Row="0"
         Grid.Column="1"
         TextWrapping="Wrap"
         VerticalAlignment="Top"
         Width="120" />
<TextBox Name="txtProductoDescripcion"
         HorizontalAlignment="Left"
         Height="23"
         Grid.Row="0"
         Grid.Column="2"
         TextWrapping="Wrap"
```

7. Solución y Proyectos

7.1 Deberá existir un archivo de proyecto por cada Clase, Enumerador, Interface, Estructura, Atributo, Excepción, por tal motivo no deberán de existir más de uno de estos elementos dentro de un archivo físico, de esta manera disminuirán los impactos dentro del control de código fuente, y será más fácil ubicar un elemento en la solución.

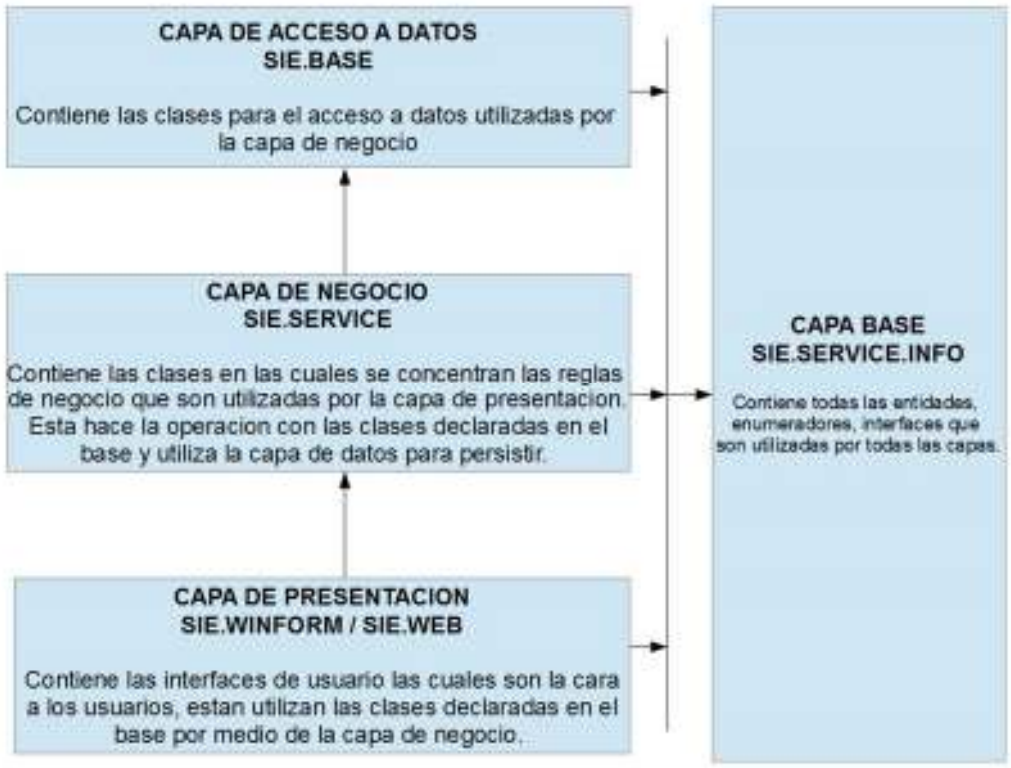
incorrecto:



correcto:

8. Arquitectura

8.1 La aplicación tiene una distribución a 4 capas, y cada capa tiene una función importante, es estrictamente obligatorio respetar esta distribución y por lo tanto colocar cada archivo dentro de su carpeta de proyecto en la capa adecuada.



8.2 La **capa base** contiene todas los elementos que son de uso común, y que necesitan ser conocidas por todas las capas, tales como entidades, enumeradores, atributos, extensiones, utilerías, excepciones, constantes, interfaces. Esta capa es visible para todas las demás capas.

8.3 La **capa de presentación** contiene todas las interfaces de usuario, que a su vez son la cara principal del sistema, en esta capa se deberán realizar validaciones de controles de usuario como entradas de datos validas, campos de capturas obligatorios, campos de rango válidos, en si todo lo que tiene que ver con la validación de la interfaz de usuario y se ven elementos como páginas web (html, aspx), servicios (asmx, svc, wcf), scripts (js, coffeeScript, typeScrypt), hojas de estilos (css, less), pantallas de aplicación (xaml, windows form), controles de usuario (ascx). Esta capa puede ver a la capa base para el uso de todos los elementos y utiliza a la capa de negocio para realizar las operaciones pertinentes. Esta estrictamente prohibido utilizar la capa de datos directamente desde esta capa.

8.4 La **capa de negocio** contiene todas las reglas de negocio tales como, validaciones de negocio, cálculos, formulas, flujos de procesos, interacción entre diferentes áreas/procesos de negocio. Deberá hacer uso exclusivo de las entidades definidas de la capa base, tanto para la recepción y respuesta de información desde la capa de presentación como el envío y recepción de datos en la capa de datos, esta capa recibe entidades con información de la capa de presentación y una vez después de ser validada y procesada deberá transmitir su respuesta a la capa de presentación o

de ser necesario enviar una petición a la capa de datos para la ejecución de alguna acción o consulta en el repositorio de datos. Esta capa usa los elementos de la capa base, y es el intermediario entre la capa de presentación y la capa de datos.

8.5 La **capa de acceso a datos** se encarga de consultar información y/o persistir información que es enviada de la capa de negocio, en esta capa van todos aquellos componentes necesarios para realizar la conexión con los diferentes repositorios de datos tales como bases de datos en sus diferentes motores, servicios web, archivos de datos. Esta capa hace uso de los elementos de la capa base y únicamente recibe y comunica información a la capa de negocio.

8.6 El diagrama de secuencia de comunicación entre capas es el siguiente:

Es importante respetar esta arquitectura ya que de no ser así, podrá en riesgo la escalabilidad, crecimiento, interoperabilidad, intercomunicación entre sistemas.

8.7 No hacer más de 2 llamadas en BD en un Ciclo, si no se requiere que la información sea validada en tiempo real, utilizar un método que regrese toda la información, y dentro del ciclo validar en base a lo cargado en memoria y no consultar en BD.

Ejemplo:

```
var lista = listaEntradaProductos.Select(entradaProducto =>
entradaProductoPl.ObtenerEntradaProductoPorId(entradaProducto.EntradaProductoId)).
ToList();
```

9. Navegadores permitidos

9.1 Para caso de las aplicaciones Web, se deberá cubrir la funcionalidad requerida al 100% y sin excepciones, bajo los siguientes navegadores permitidos:

- Google Chrome
- Mozilla Firefox
- Como nota aclaratoria, cualquier aplicación web deberá funcionar en ambos navegadores arriba mencionados.

10. Versionamiento del Código

- Solo subir código necesario, no subir archivos innecesarios.
- Detalle del proceso de versionamiento:
    1. SuKarne entrega branch a la fábrica.
    2. La fábrica deberá administrar de manera local (en algún servidor de ellos mismos) el código.
    3. La fábrica hace los desarrollos solicitados, ejecuta pruebas unitarias y hace sus correcciones.
    4. Una vez concluido el desarrollo ya includias pruebas unitarias, la fábrica deberá entregar a SuKarne un checklist de implementación (scripts, configuraciones) y el ejecutable de la aplicación.
    5. SuKarne monta en ambiente de calidad la aplicación y ejecuta pruebas integrales.
    6. Si se encuentran defectos, SuKarne regresa relación de defectos a la fábrica y el flujo regresa al paso 3.
    7. Cuando ya no existen defectos en la aplicación, SuKarne le solicita a la fábrica versionar todos los cambios relizados.
    8. La fábrica versiona en el repositorio de SuKarne todos los cambios ya sea de código y/o de base de datos. Es importante mencionar que el versionamiento es con subversion (SVN).
    9. De esta manera nos aseguramos que la fábrica entregará a SuKarne una sola revisión de SVN.