Serpost
El Correo del Perú

NORMAS PARA EL

DESARROLLO

DE APLICACIONES

SD-N-005.02

I. OBJETIVO

La presente Especificación de Estándares – Desarrollo de Aplicaciones, tiene como objetivo establecer un marco general bajo el cual se pueda crear interfaces, objetos y variables, y nomenclaturas (codificación) de cualquier sistema que deba crearse por solicitud de cualquier área usuaria de la Empresa.

II. ALCANCE

El documento alcanza al Departamento de Sistemas de la Información de la Gerencia de Desarrollo Corporativo y todos los especialistas informáticos que participen directa o indirectamente, a tiempo parcial o completo en el desarrollo y/o mantenimiento de cualquiera de los aplicativos que forman parte de SERPOST.

III. BASE LEGAL

3.1. Decreto Legislativo N° 685 Ley de Creación de SERPOST S.A.

3.2. Estatuto Social de SERPOST S.A.

3.3. Resolución de Contraloría N° 320-2006-CG Normas de Control Interno

3.4. Norma Técnica Peruana "NTP-ISO/IEC 17799:2007 EDI. TECNOLOGÍA DE LA INFORMACIÓN. CÓDIGO DE BUENA PRÁCTICAS PARA LA GESTIÓN DE SEGURIDAD DE LA INFORMACIÓN"

3.5. Normativa interna.

IV. DEFINICIONES Y ABREVIACIONES

Se definen los siguientes términos para mejor interpretación del contenido de este documento:

a. **Net:** Plataforma de desarrollo de Microsoft.

b. **XML:** eXtensible Markup Language.

c. **JavaScript:** Lenguaje interpretado en el cliente, multiplataforma y orientado a eventos.

d. **C#:** Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

V. NORMAS

A) ESTÁNDARES PARA EL DISEÑO DE LAS INTERFACES DE USUARIO

Es importante mantener un estándar al diseñar las interfaces para el usuario final, de tal forma que se mantenga en lo posible uniformidad en la operatividad de los sistemas.

i. Principios de Codificación

Los estándares de codificación de SERPOST S.A. se guían por los siguientes principios:

a. El código debe ser legible para poder mantenerse.
b. Se debe hacer uso, siempre que sea posible, de la programación en el servidor de base de datos de tal forma que se optimice el uso los recursos de la red y se aproveche la capacidad de procesamiento del servidor de base de datos.
c. Debe evitarse el código específico para alguna plataforma y sólo hacerlo cuando sea estrictamente necesario.
d. Siempre que sea posible deben diseñarse y emplearse objetos reutilizables.

ii. Estándares para desarrollo general

a. Claridad y consistencia

Asegurar que la claridad, legibilidad y transparencia sean de suma importancia. El estándar de codificación apunta a asegurar que el código resultante sea fácil de entender y mantener, pero nada supera al código claro, conciso y que es su propia documentación.

b. Formato y estilo

Todo el código debería ser escrito utilizando cuatro espacios para la indentación. No utilizar tabulaciones. Es generalmente aceptado a través de Microsoft que las tabulaciones no deberían ser usados en los archivos de código – diferentes editores de texto utilizan espaciado distinto para reproducir las tabulaciones y esto causa inconsistencia en el formato.

Ejemplo C#:

```
public class Example
{
····public void Method()
····{
········return;
····}
}
```

c. Uso de librerías

No hacer referencia a librerías innecesarias, tampoco incluir namespaces innecesarios, ni referenciar dll que no van a ser usadas. Esto mejora el tiempo de compilación, reduce oportunidades para equivocaciones y mejora la legibilidad del archivo.

d. Variables globales

Mantener al mínimo las variables globales. Para utilizar variables globales de manera apropiada siempre pasarlas a las funciones a través de valores de parámetros. Nunca hacer referencia directa a variables globales dentro

de métodos o clases porque esto crea efectos colaterales que alteran el estado global sin que lo sepa quién invoca. Lo mismo va para las variables estáticas. Si se necesita modificar una variable global, debería hacerse como un parámetro de salida o devolviendo una copia de la variable.

iii. Formularios

En términos generales, todos los Formularios deben cumplir con las siguientes especificaciones:

a. Como título del formulario, debe colocarse siempre una descripción clara y breve del documento que se va registrar (en el caso de pantallas de capturas de datos) o el nombre de la consulta/reporte que se va obtener. El cono a colocarse en la parte superior izquierda del formulario, debe ser el de la Empresa.

b. Los controles incrustados en los formularios (sean estos para capturar o mostrar datos) deben estar ordenados y alineados guardando la simetría y estética necesaria.

c. Los controles que van a ser incrustados en un Formulario y que van a mostrar datos de un mismo genero (o referentes a un tema u objeto específico), deben ser agrupados en *frames* titulados adecuadamente.

d. No se debe pretender mostrar muchos datos en un formulario llenándolo de controles, pues se le resta funcionalidad. Lo más aconsejable es hacer uso de pestañas o de formularios anidados.

e. Hacer uso, siempre que sea posible, de las teclas funcionales, sobre todo en los formularios de captura de información (ingreso masivo de datos) en los que se necesita reducir los tiempos de operación.

f. Para tal efecto se define la siguiente relación entre las teclas funcionales y las operaciones que deberían realizar:

[F2]	=	Adicionar
[F3]	=	Modificar
[F4]	=	Grabar
[F5]	=	Actualizar
[F6]	=	Anular /Eliminar
[F7]	=	Buscar
[F8]	=	Imprimir
[ESC]	=	Salir /Cancelar
[ENTER]	=	Aceptar
[F9] – [F12]	=	Otras Funciones adicionales

iv. Formularios MDI

Para el diseño de los formularios MDI es necesario seguir los siguientes lineamientos:

a. Como título de los formularios deberá colocarse el nombre completo del sistema.

b. El fondo de pantalla a utilizarse en los formularios MDI es aquel determinado como estándar por el Departamento de Sistemas de la Información.

c. En cuanto al diseño de los menús de los formularios MDI, se deberá tener en cuenta la siguiente estructura:

- **Mantenimiento.-** En este menú se deberán colocar todas las interfaces para la captura y mantenimiento de datos.
- **Consultas /Reportes.-** En este menú se deberán ubicar todas las interfaces para obtener consultas y reportes para el usuario final.
- **Tablas.-** En este menú se deberán colocar todas interfaces para el mantenimiento de las tablas auxiliares del sistema.
- **Administración del Sistema.-** En este menú se ubicarán todos los módulos relacionados a la seguridad, configuración del sistema y otros parámetros.
- Se debe incluir una barra de herramientas con botones de acceso directo a las principales opciones del sistema.
- Se debe incluir tambien una barra de estado donde se muestre principalmente el usuario actual que inicio sesión en el sistema.

Debe tenerse en cuenta que estos estándares pueden acondicionarse según la naturaleza del sistema o módulo (por ejemplo, un módulo que no permita el manejo de las tablas del sistema, no mostrará dicha opción al usuario).

v. Formularios de Ayuda

Para el diseño de formularios de ayuda se deben seguir los siguientes lineamientos:

a. Se debe mostrar en la cabecera del *grid* donde se muestran los datos de las tablas auxiliares, el nombre de la entidad a la que pertenecen los datos.

b. Mostrar siempre los campos 'código' y 'nombre' de la entidad en el *grid.*

c. Para seleccionar un registro, bastará con dar doble clic sobre aquel o pulsar la tecla [ENTER].

d. Los formularios de ayuda(o selección) deberán ser modales y aparecerán siempre centrados en la pantalla.

B) ESTÁNDARES PARA LA NOMENCLATURA DE OBJETOS Y VARIABLES

i. Estándares para nombrar Controles

Para nombrar los controles empleados durante el diseño de las interfaces, se debe tener en cuenta qué información se va mostrar y capturar en los controles precedidos por un prefijo que nos permita identificar (al momento de programar o darle mantenimiento) el tipo de control al que se está haciendo referencia.

Prefijo + AAAA...A

El Anexo 1 muestra un listado de los prefijos a utilizar para nombrar a los principales objetos y los ejemplos de como emplearlos.

ii. Estándares para nombrar los objetos ADO

Para nombrar los objetos pertenecientes a la colección *ADO*, se debe especificar un nombre descriptivo que haga alusión a la función que va cumplir o al dato que va almacenar (según sea el caso), pero precedido por un prefijo determinado que identifica al objeto de manera clara.

El Anexo 2 muestra un listado de los prefijos a utilizar para nombrar a los principales objetos y los ejemplos de como emplearlos.

iii. Estándares para nombrar las variables

Para efectos de nombrar se debe tener en cuenta la siguiente estructura:

A+B+CCC...C

Dónde:

→ A = Carácter que identifica alcance de la variable.
→ B = Carácter que identifica Tipo de dato Genérico.
→ C = Nombre que identifica y describe el dato que va almacenar la variable.

Tipos de Alcance de las Variables

Alcance	Símbolo
Global	P
Módulo o Form	M
Local	L

Tipos de Datos Genéricos

Tipo de dato Genérico	Símbolo
Cadena de caracteres	C
Lógico	L
Numérico	N
Fecha	D
Objeto	O

iv. Estándares para nombrar objetos de Base de Datos

Todos los objetos de base de datos deben nombrarse sin utilizar caracteres especiales. Los nombres de objetos de base de datos sólo deben utilizar letras, números y subrayados y deben comenzar con una letra.

a. **Base de Datos.-** Para nombrar las Base de Datos se deberá elegir un nombre que describa lo mejor posible el tipo de información que va almacenar o en todo caso al área que administrará dicha información.
Ejemplo: 'CONTABILIDAD'

b. **Tablas.-** Para nombrar una tabla de una base de datos se debe seguir el siguiente criterio:

AAABBBB...B

Donde los tres (3) primeros dígitos deben ser una descripción corta del sistema con el cual interactúa y los siguientes una descripción lo más clara y concisa posible de la entidad cuyos datos vamos a almacenar en la tabla.
Ejemplo: 'SCTCUENTA'

c. **Campos.-** Para nombrar los campos de una tabla de una base de datos se debe seguir el siguiente criterio:

ABBBBCCC...C

Donde el primer carácter define el tipo de dato genérico que se guardará en el campo. Los cuatro (4) siguientes (del segundo al quinto dígito) es una cadena que representa una abreviatura de la tabla y los siguientes una descripción del dato a almacenar.

Los prefijos según el tipo de dato genérico son:

C → Carácter
N → Numérico
D → Fecha
L → Lógico

Ejemplos: 'CTRABCODIGO', 'DTRABFECHANAC', 'NTRABSUELDO'

d. **Indices.-** Para nombrar a los índices de una tabla se debe anteponer el prefijo 'IDX' seguido del nombre del campo a indexar.

"IDX" + AAABBBB...B

Ejemplo: 'IDXNOMBRE', 'IDXFECHA'

e. **Triggers.-** Para nombrar a los bloques asociados a una tabla se debe usar un criterio parecido al de las tablas pero anteponiéndosele el prefijo 'TR'.

"TR" + AAABBBB...B

Ejemplo: 'TRAUDITORIAVENTAS', 'TRSEGUIMIENTOENVIO'

f. **Vistas.-** Para nombrar a las Vistas se debe usar un criterio parecido al de las tablas pero anteponiéndosele el prefijo 'VW'.

"VW" + AAABBBB...B

Ejemplo: 'VWSCTPLANILLAPAGO'

g. **Constraints.-** Para nombrar constraints se debe seguir el siguiente criterio:

Llave primaria	PK_Tabla
Llave foránea	FK_Tabla_TablaPadre

Ejemplos:
'PK_LOGCOTIZACION',
'FK_ LOGCOTIZACION_PROVEEDOR'

h. **Procedimientos Almacenados.-** Para nombrar Procedimientos Almacenados se debe especificar una descripción breve del proceso que se va realizar precedida por el prefijo "PROC" y las siglas de la base de datos, , con una cantidad máxima de 30 caracteres

"PROC"+AAABBBB...BB

Ejemplo: 'PROSCTMAYORAUXILIAR'

i. **Paquetes o Packages.-** Son un conjunto de objetos agrupados en un paquete principalmente son Procedimientos Almacenados y se deben de nombrar el Paquete o Package por el prefijo "PCK" y el nombre del aplicativo con una cantidad máxima de 30 caracteres.

"PCK"+MM....MM

Ejemplo: 'PCKATENCIONSUNAT'

j. **Cursores.-** Para nombrar cursores dentro de un Procedimiento almacenado se debe tener en cuenta el siguiente criterio:

"CUR" + AAAA...A

El nombre del cursor siempre debe estar precedido por el prefijo "CUR" seguido de un nombre breve y conciso que referencia la información que va manipular.

Ejemplo: 'CURPENDIENTESDEPAGO'

k. **Tablas Temporales.-** Para nombrar tablas temporales tanto locales como globales dentro de un procedimiento almacenado, debemos especificar un

nombre que describa en forma clara y breve el tipo de información que se va almacenar en la tabla temporal precedido del prefijo "TMP".

Ejemplo: 'TMPCONSOLIDADO', 'TMPESTADISTICA'

l. **Web Services.-** Archivo que son utilizados por un servicio web para intercambio de información entre aplicaciones independientemente del lenguaje de desarrollo del sistema, para nombrarlos se debe de tomar en cuenta que no debe excederse de los 25 caracteres, anteponiendo el prefijo "WS" y una referencia al servicio que va a realizar

Ejemplo: 'WS_ExportaFacil', 'WS_SeguimientoEnvios'

C) ESTÁNDARES PARA LA CODIFICACIÓN

I. Declaración de variables e inicialización

Declarar las variables locales en el mínimo bloque que las pueda contener, en la mayoría de los casos inmediatamente antes de su uso.

Inicializar las variables cuando sean declaradas siempre que sea posible.

Declarar e inicializar/asignar las variables locales en una sola línea de ser posible. Esto reduce el espacio vertical y asegura que la variable no exista en un estado sin inicialización o en un estado que inmediatamente cambiara.

```
string usuario = User.Identity.Name;
DateTime fecha = DateTime.Now;
```

II. Enumerados

Utilizar enum para parámetros, propiedades y valores de retorno que representen una posible opción dentro de un conjunto específico de valores.

Utilizar enum antes que constantes estáticas. Un enum es una estructura con un conjunto de constantes estáticas. La razón para seguir este lineamiento es porque de esta manera se obtendrá soporte adicional del compilador y de reflexión si se define un enum en vez de una clase con constantes.

No utilizar un enum para conjuntos abiertos.

Proveer un valor de cero para enums simples. Considerar llamar al valor tal como "Ninguno" (None). Si tal valor no es apropiado para el enum en particular, el valor por defecto más común debería ser asignado a "Ninguno".

```
public enum Color
{
  White,
  Red,
  Green,
  Blue,
  Black
}
//.....
if (c < Color.Red || c > Color.Blue)
{
  throw new ArgumentOutOfRangeException(...);
}
```

III. Comentarios

Los comentarios deben resumir que es lo que una pieza de código ha sido diseñada para realizar y porque. No utilizar comentarios que solo repitan lo la línea de código hace a bajo nivel..

Los comentarios deberían estar indentados al mismo nivel que el código que describen.

En los comentarios se debe respetar la correcta escritura, ortografía y puntuación. No escribir todo en mayúsculas. El código es un documento redactado por un profesional.

D) ESTÁNDARES PARA DESARROLLO .NET

Para una mejor fluidez en el código, utilizar el idioma inglés en la escritura del código. Los comentarios deben ser redactados en español.

Utilizar nombres significativos para los distintos tipos, métodos, variables y clases.

No se deberían utilizar abreviaciones como parte del nombre de los identificadores. Utilizar términos completos, pero se aceptan abreviaciones solamente en el caso de que sean ampliamente reconocidas, por ejemplo Xml.

Solo utilizar caracteres alfanuméricos regulares en los nombres. .NET permite utilizar caracteres UNICODE en los nombres de identificadores pero solo se deberían utilizar letras [a-z] (sin tildes ni ñ) y números [0-9].

Nunca utilizar prefijos para indicar que la variable o parámetro es un objeto (oParameter).

a. Arreglos y colecciones

Se deben usar arreglos en métodos de bajo nivel para minimizar el consumo de memoria y maximizar el rendimiento. En el caso de interfaces públicas preferir utilizar colecciones.

No utilizar arreglos de solo lectura (readonly). El campo mismo es solo lectura y no puede ser modificado pero los elementos del arreglo si pueden ser cambiados. En su lugar se puede utilizar una colección de solo lectura (solo si los elementos son inmutables) o clonar el arreglo antes de entregarlo. Sin embargo el costo de clonar el arreglo puede ser prohibitivo

Utilizar Collection<T> o una subclase de Collection<T> para propiedades o valores de retorno que representen colecciones de lectura/escritura, y utilizar ReadOnlyCollection<T> o una subclase de esta para propiedades o valores de retorno de solo lectura.

No retornar una referencia nula (null) de un arreglo o colección. El valor nulo puede ser inesperado en el contexto, en su lugar devolver un arreglo o colección vacío.

b. Clases

Utilizar la herencia para expresar relaciones de tipo "A es un B".

```
public class Agenda : List<Actividad>
```

Utilizar interfaces para expresar relaciones de tipo "A puede realizar (hacer) B"

```
public class Ventana : IAutorizable
```

c. Campos

No utilizar campos de la instancia que sean públicos. Los campos públicos crean problemas de versión y no encapsulan los detalles de la implementación. En su lugar utilizar campos privados y exponerlos a través de propiedades o utilizar propiedades automáticas.

Utilizar campos public static readonly para instancias de objetos predefinidas.

Utilizar campos const para constantes que nunca van a cambiar.

No asignar instancias de tipos mutables a campos de solo lectura.

d. Propiedades

Crear propiedades de solo lectura si quien invoca la propiedad no debería ser capaz de modificar el valor.

No proveer propiedades de solo escritura. Si la propiedad de *get* no puede ser expuesta, utilizar un método que implemente la funcionalidad de *set* en su lugar. El nombre del método debería comenzar por Set seguido por lo que hubiera sido el nombre de la propiedad.

No se deberían lanzar excepciones desde el método *get* de una propiedad. Estos métodos deberían ser operaciones simples sin ninguna precondición. Si el método fuera a arrojar una excepción, considérese rediseñar la propiedad

para que sea un método. Esta recomendación no aplica para los indexadores. Los indexadores puedes arrojar excepciones debido a argumentos inválidos. Es válido y aceptable lanzar excepciones desde el método *set* de una propiedad.

e. Métodos

Colocar todos los parámetros de salida después de los parámetros por valor y por referencia (excluyendo al arreglo de parámetros (params)), inclusive si esto resulta en una inconsistencia en el ordenamiento de los parámetros entre sobrecargas.

Validar los argumentos entregados a los miembros públicos, protegidos o implementados explícitamente. Lanzar ArgumentException o una de sus subclases si la validación falla. Si un argumento nulo es entregado y el miembro no soporta argumentos nulos, lanzar ArgumentNullException. Si el valor del argumento se encuentra fuera del rango de valores definidos por el método invocado, lanzar ArgumentOutOfRangeException.

f. Eventos

Estar preparado para que se ejecute código arbitrario en el método de manejo del evento. Considerar colocar el código donde se levanta el evento dentro de un bloque try-catch para prevenir la finalización inesperada del programa debido a excepciones no controladas lanzadas desde los manejadores de eventos.

```
public class Metronome
{
  public event TickHandler Tick;
  public EventArgs e = null;
  public delegate void TickHandler(Metronome m, EventArgs e);
  public void Start()
  {
    while (true)
    {
      System.Threading.Thread.Sleep(3000);
      if (Tick != null)
      {
        try
        {
          Tick(this, e);
        }
        catch(Exception ex)
        {
          //Handle exception
        }
      }
    }
  }
}
```

No utilizar eventos en APIs de rendimiento sensible. Mientras que los eventos son más fáciles de entender y usar para muchos desarrolladores, son menos deseables que miembros virtuales desde la perspectiva de rendimiento y consumo de memoria.

g. Sobrescribiendo Dispose

Si se hereda de una clase base que implementa IDisposable, se debe implementar IDisposable también. Siempre llamar al método Dispose(bool) de la clase base también.

E) ESTÁNDARES PARA DESARROLLO JAVASCRIPT

Se va detallar las reglas del lenguaje:

a. Var

Siempre declarar las variables utilizando var.

Si no se especifica var, la variable es colocada en el contexto global, pudiendo hacer conflicto con valores existentes. Además si no existe declaración es difícil saber el alcance del contexto de la variable.

b. Constantes

Nunca utilizar la palabra reservada const debido a que no es soportada por Internet Explorer. Se pueden utilizar variables.

c. Punto y coma

Siempre utilizar punto y coma. No se debe depender de la inserción implícita porque puede causar problemas difíciles de depurar.

JavaScript requiere que las sentencias terminen con un punto y coma, excepto cuando supone que puede inferir con seguridad su existencia. El cerrar llaves no es suficiente para indicar el final de una sentencia. JavaScript nunca termina una sentencia si el siguiente elemento es infijo o un operador de llave.

d. Excepciones

Utilizar excepciones para el manejo de errores en lugar de usar códigos de error.

e. Características estándar

Siempre preferir características estándar de JavaScript. Este provee máxima portabilidad y compatibilidad.

f. Definición de métodos

Para asignar propiedades y métodos a un constructor debe ser de la siguiente manera:

```
Foo.prototype.bar = function() {
  /* ... */
};
```

g. eval()

Solamente utilizar eval() para deserialización.

h. with() {}

No utilizar with(), obscurece la semántica del código debido a que el objeto que se le pasa a with puede tener propiedades que colisionen con variables locales y puede cambiar drásticamente el significado del código.

i. this

Utilizar la palabra reservada this solo en constructores, métodos y construyendo cerraduras. La semántica de this puede ser intrincada. En algunos puntos se refiere al objeto global (en la mayoría de lugares), el contexto de quien lo invoca, un nodo en el árbol DOM, un objeto que acaba de ser creado o algún otro objeto. Porque es fácil equivocarse con esta palabra reservada, su uso debería estar limitado a donde se requiere.

j. Reglas de Estilo

- **Nombres:**

Identificador	Tipo	Estructura del nombre	Ejemplo
Función	camelCase	Verbo	function processItem() {...};
Variable	camelCase	Sustantivo	var customerId;

Clase	PascalCase	Sustantivo	project.MyClass = function () {} new project.MyClass();
Enumerado	PascalCase	Sustantivo Agregar el sufijo "Enum"	my.namespace.ColorEnum = { Red : 0,Green : 1, Blue : 2};
Métodos	camelCase	Verbo Los métodos privados deben comenzar con "_"	function _init() { ... };
Constantes simbólicas	Todo mayúsculas	Sustantivo	PI = 3.141592;
Propiedades	camelCase	Sustantivo Las propiedades privadas deben comenzar con "_"	_privateProperty = 'value'; publicProperty = 0;
Parámetros	camelCase	Sustantivo	Function f(paramN1, paramN2);
Métodos de Acceso	camelCase	Sustantivo o Adjetivo Deben comenzar con los prefijos "get" o "set" según el caso	getProperty(); setProperty(valu e);
Nombre del archivo	Todo en minúscula, separado por guiones "-"	Nombre del módulo o librería Debe terminar con la extensión .js	grid-helper.js

F) CONTROL DE CÓDIGO FUENTE

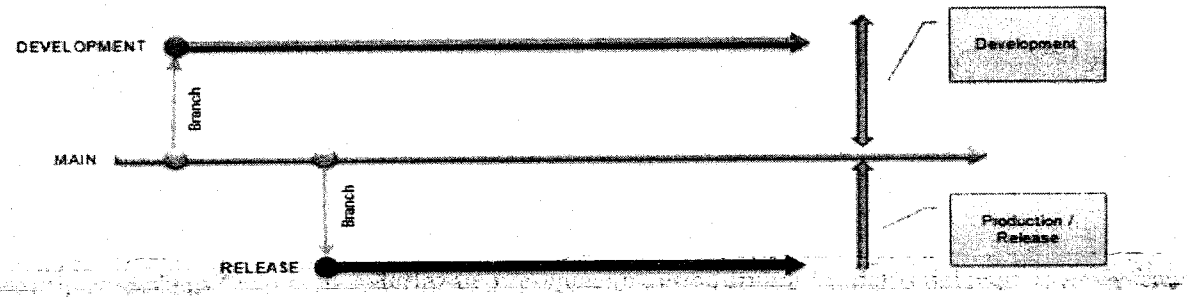
a. Software

El software de control de código fuente a usar Team Foundation Server, proporciona la funcionalidad de control de versiones de código fuente estándar.

b. Estructura de carpetas

La estructura se basa en el Plan Básico de la Guía de Bifurcación 2010. Los elementos clave de este plan incluyen:

- Dev: Para trabajar en la versión de desarrollo del producto.
- Main: Mantiene la última versión estable.
- Releases: Contiene las versiones entregables del producto.



c. **Políticas de Check In**

Considerar las siguientes políticas para los cambios en el código fuente:

i. **Check Out**

Antes de iniciar el desarrollo asegurarse de trabajar sobre la última versión del código fuente.

ii. **Check In**

- Verificar que la solución compile antes de hacer check in.
- Hacer un único check in que incluya todos los cambios realizados para implementar una funcionalidad. No hacer check in parciales que suba código que no compile en el servidor.
- Agregar comentarios que describan los cambios realizados.

G) ESTÁNDARES PARA DOCUMENTAR LOS PROGRAMAS

Para el desarrollo de programas se seguirán los siguientes lineamientos:

a. **Eventos**

- Se debe mantener el orden al programar los eventos haciendo uso de una adecuada tabulación cuando corresponda. Esto permitirá que los programas sean más legibles y de rápida comprensión.
- Es necesario realizar la documentación de los eventos en forma paralela a la programación de los mismos a través de los comentarios que se deben incluir siempre que sea posible y necesario en forma contigua o en la línea superior de la línea de código que se va documentar.

b. **Funciones y Procedimientos personalizados**

Son las funciones o procedimientos desarrollados por el programador con la finalidad de cumplir una tarea específica. Estos por lo general son públicos por lo que deberían ser reutilizables. Los criterios a seguir para lograr una programación de calidad son los siguientes:

- Es importante mantener el orden al programar haciendo uso de una adecuada tabulación cuando corresponde. Esto permitirá que los programas sean más legibles y de rápida comprensión.
- Se debe incluir siempre en la parte superior de la función o procedimiento un encabezado en forma de comentario. Este debe estar conformado por 3 secciones claramente definidas. La primera deberá definir el propósito o tarea que deberá cumplir la función o procedimiento, la segunda sección servirá para indicar las entradas necesarias o requeridas por la función y por último la tercera sección permitirá saber cuáles son las salidas que producirá.
- Es necesario realizar la documentación de las líneas de código de las funciones o procedimientos en forma paralela a la programación de los mismos a través de los comentarios que se deben incluir siempre que sea posible y necesario en forma contigua o en la línea superior de la línea de código que se va documentar.

c. **Procedimientos Almacenados.-**

Los criterios a seguir para realizar una buena programación al desarrollar un procedimiento almacenado son similares al caso anterior.

H) ESTÁNDARES PARA EL DISEÑO DE MENSAJES

Los mensajes son el medio de comunicación con los usuarios. Los mensajes típicos incluyen advertencias y mensajes de error, instrucciones breves y mensajes informativos que le comunican al usuario del estado de sus requerimientos. Las aplicaciones basadas en Formularios muestran mensajes en la ventana del usuario. Estos estándares le ayudan a escribir mensajes que sean breves, claros e informativos.

Un tipo de mensaje se clasifica como:

a. **Mensajes de Error.-** Muestran información sobre problemas presentados durante la operatividad del sistema.

b. **Mensajes de Advertencia.-** Muestran información solicitando reconfirmación para realizar cierta operación.

c. **Mensajes Informativos.-** Muestran información informando al usuario acerca de la operatividad del sistema.

I) ESTÁNDARES PARA TÍTULOS DE MENSAJES

Los siguientes estándares sugeridos proporcionan una convención consistente para los títulos de mensajes.

a. Los Títulos de mensajes se utilizan en los formularios y programas y nunca deben cambiar.

b. Escriba los nombres de mensajes con mayúsculas y minúsculas y siempre de la misma forma.

c. No utilice espacios en los nombres de mensajes.

d. Los títulos de mensajes no deben contener números de mensajes o prefijos

e. El criterio para la titulación de los mensajes es el siguiente:

- 'ERROR' para los mensajes de tipo Error;
- 'ADVERTENCIA' para los mensajes de tipo Advertencia; y
- 'ATENCION' para los mensajes de tipo Informativo.

J) ESTÁNDARES PARA LA AUDITORIA Y SEGURIDAD

Se debe tener en cuenta que estos estándares están referidos a la opción de Auditoría dentro de un módulo o programa, cualquiera sea la naturaleza de este, que se debe contemplar al momento de desarrollar las aplicaciones.

i. Ubicación de la opción de Auditoría

La opción de Auditoria debe contemplar lo siguiente:

a. Debe colocarse de preferencia en la opción 'Administración del Sistema' del menú principal del sistema.
b. Deben tener acceso solamente el personal de Sistemas designado para tal propósito.
c. Debe ser puesto en cada sistema que se desarrolle.

ii. Opciones Básicas de Auditoría

La opción de Auditoría debe poseer las siguientes opciones básicas:

a. Criterio de Búsqueda por rango de fechas.
b. Selección de Acción a auditar: Inserción (INSERT), Actualización (UPDATE), Eliminación (DELETE) o TODAS las anteriores.
c. Selección de tabla específica.
d. Selección de Módulo del Sistema a auditar.
e. Criterio de Búsqueda por usuario específico. En este caso debe mostrarse *código* y *nombre* del usuario
f. Debe tener los siguientes botones, como mínimo: CONSULTAR, SALIR e IMPRIMIR.
g. Todos los datos deben reflejarse en un *grid.*
h. Los datos obligatorios que se deben de almacenar en la tabla de auditoría:

i. Acción que se realizó Inserción (INSERT), Actualización (UPDATE), Eliminación (DELETE)
ii. Usuario del módulo que la ejecuto
iii. Nombre del equipo de donde se ejecuto
iv. Fecha y hora del servidor en que se realizó la acción
v. Usuario del Sistema Operativo que se logeo a la red
vi. Tabla que fue afectada
vii. La llave única del registro afectados

VI. ARQUITECTURA DE DESARROLLO CONVENCIONAL

Para el desarrollo de las nuevas aplicaciones de desarrollo se utilizará una arquitectura de 3 capas.

1. **Capa Presentación:**
La capa de presentación enviará mensajes a los objetos de esta capa de negocios o intermedia, la cual o bien responderá entonces directamente o mantendrá un diálogo con la capa de la base de datos, la cual proporcionará los datos que se mandarían como respuesta a la capa de presentación.
Podemos decir que es la que se presenta al usuario, llamada también formulario o interfaz de presentación, esta captura los datos del usuario en el formulario e invoca a la capa de negocio, trasmitiéndole los requerimientos del usuario, ya sea de

almacenaje, edición, o de recuperación de la información para la consulta respectiva.

2. **Capa Lógica de Negocios:**
La capa lógica de negocio es la responsable del procesamiento que tiene lugar en la aplicación, por lo cual contendrá objetos definidos por clases reutilizables que se pueden utilizar una y otra vez en otras aplicaciones (contienen la gama normal de constructores, métodos para establecer y obtener variables, métodos que llevan a cabo cálculos y métodos, normalmente privados, en comunicación con la capa de la base de datos).
Es en esta capa se recibirán los requerimientos del usuario y se enviaran las respuestas tras el proceso, a requerimiento de la capa de presentación, en realidad se puede tratar de varias funciones, por ejemplo, puede controlar la integridad referencial, otro que se encargue de la interfaz, tal como abrir y cerrar ciertos formularios o funcionalidades que tengan que ver con la seguridad, menús, etc.

3. **Capa de Acceso a Datos:**
Esta capa se encarga de acceder a los datos, se debe usar la capa de datos para almacenar y recuperar toda la información de sincronización del Sistema.
Es aquí donde se implementa las conexiones al servidor y la base de datos propiamente dicha, se invoca a los procedimientos almacenados los cuales reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio

Todas estas capas pueden residir en un único ordenador (no debería ser lo usual), pero es lo más frecuente. En sistemas complejos se llega a tener varios ordena-dores sobre los cuales reside la capa de datos, y otra serie de ordenadores sobre los cuales reside la base de datos.
Se recomienda que si el crecimiento de las necesidades o complejidad aumenta se debe separar en dos o más ordenadores, los cuales recibirán las peticiones del ordenador en que resida la capa de negocio. Esta recomendación es válida para la capa de negocios.

❖ **Relación de la Capas(Anexo 4)**

VII. ARQUITECTURA DE DESARROLLO NO CONVENCIONAL

Se cuenta actualmente con la programación en n capas que hace uso de la programación orientada a objetos; la cual consiste en separar el código fuente según el rol, responsabilidad y funcionalidad; por ende el desarrollo es más rápido, y resulta más fácil el darle mantenimiento al Sistema.

1. **Programación en N capas**
El estilo arquitectural en n capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan.

Cuanto más se aumenta el proceso operativo de la Empresa, las necesidades de proceso crecen hasta desbordar las máquinas. Es por ello que se separa la estructura de un programa en varias capas.

En adición podemos decir que actualmente la programación por capas es un estilo de programación en la que el objetivo principal es separar la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de negocios y ésta a su vez de la capa de presentación al usuario.

2. Objetivos de diseño de aplicaciones distribuidas

El diseño de una aplicación distribuida implica la toma de decisiones sobre su arquitectura lógica y física, así como sobre la tecnología e infraestructura que se emplearán para implementar su funcionalidad. Para tomar estas decisiones, debe tener un conocimiento claro de los procesos empresariales que realizará la aplicación (sus requisitos funcionales), así como los niveles de escalabilidad, disponibilidad, seguridad y mantenimiento necesarios (sus requisitos no funcionales, funcionales u operativos).

El objetivo consiste en diseñar una aplicación que:

- ✓ Solucione el problema empresarial para el que se diseña.
- ✓ Tenga en consideración la seguridad desde el principio, teniendo en cuenta los mecanismos adecuados de autenticación, la lógica de autorización y la comunicación segura.
- ✓ Proporcione un alto rendimiento y esté optimizada para operaciones frecuentes entre patrones de implementación.
- ✓ Esté disponible y sea resistente, capaz de implementarse en centros de datos de alta disponibilidad y redundantes.
- ✓ Permita la escalabilidad para cumplir las expectativas de la demanda y admita un gran número de actividades y usuarios con el mínimo uso de recursos.
- ✓ Se pueda administrar, permitiendo a los operadores implementar, supervisar y resolver los problemas de la aplicación en función del escenario.
- ✓ Se pueda mantener. Cada parte de funcionalidad debería tener una ubicación y diseño predecibles teniendo en cuenta distintos tamaños de aplicaciones, equipos con conjuntos de habilidades variadas y requisitos técnicos y cambios empresariales.
- ✓ Funcione en los distintos escenarios de aplicaciones y patrones de implementación.
- ✓ Las instrucciones de diseño que se ofrecen en los siguientes capítulos persiguen estos objetivos y explican los motivos para las decisiones de un diseño en particular siempre que sea importante para entender su fondo.

3. Conclusión

El estilo de programación en N capas se basa en segmentar un proyecto en varias partes para realizar una programación independiente en cada una de ellas.

- ✓ Facilita la reutilización de capas.
- ✓ Permite una mejor estandarización.
- ✓ El trabajo por parte de los analistas es complejo, pero al final se crea una arquitectura más fácil de comprender y de implementar.

En cuanto a la seguridad este estilo de programación es más fiable.

- ✓ Se puede elaborar componentes para cada capa, avanzando el desarrollo de manera independiente y por ende el global del Sistema puede desarrollarse más rápido.
- ✓ Ayuda mucho al programador de aplicaciones para dar mantenimiento al Sistema, dado que el problema que pudiera suscitarse es visto en la capa respectiva.
- ✓ Por ende los costos de mantenimiento tienden a ser menores.
- ✓ Dado los vertiginosos cambios en la dinámica de los negocios este estilo de programación provee que el Sistema sea escalable.

❖ El desarrollo de aplicaciones en el Departamento de Sistemas de Información se rige básicamente en tres capas; sin embargo, el programador bajo sustento, podrá efectuar el desarrollo basado en múltiples capas siempre y cuando se cuente con la infraestructura y los controles necesarios para su implementación.

VIII. RECOMENDACIONES:

A. Recomendaciones Generales de Programación

- ✓ Las rutinas que controlan los eventos no deben contener el código que ejecuta la acción requerida. En vez de ello, llama a otro método desde la rutina controladora.
- ✓ No invocar programáticamente los eventos Ejm: Click de un botón, para ejecutar el método contenido en el evento. En vez de ello, llama el mismo método que es invocado desde la rutina controladora del evento click.
- ✓ Registrar los errores de la aplicación con toda la información posible. Esto ayudará a diagnosticar problemas.
- ✓ No guardar más de una clase en un archivo.
- ✓ Evitar tener archivos muy grandes de más de 1000 líneas de código por archivo, estos son candidatos a dividirlos lógicamente en dos o más clases.
- ✓ Evita pasar muchos parámetros a un método, no más de 5 parámetros, en su lugar utilizar estructuras o clases entidad.
- ✓ En métodos que retornan colecciones, retornar colección vacía en lugar de retornar NULL.
- ✓ Para organizar los archivos no usar jerarquías de más de dos niveles y no más de 10 carpetas en la raíz, además de no más de 5 subcarpetas.
- ✓ Usar try catch en la capa de acceso a datos, para atrapar todas las excepciones de la base de datos, luego relanzarla para que la siguiente capa lo gestione.

B. Recomendaciones Generales de Acceso a datos

- ✓ La cadena de conexión, deberá estar especificado en un archivo de configuración encriptado.
- ✓ Todas las conexiones a fuentes de datos deben ser gestionados por la capa de

datos

- ✓ Abrir conexiones lo más tarde posible y cerrarlos tan pronto como sea posible
- ✓ Nunca se debe mantener conexiones abiertas permanentemente o por grandes períodos
- ✓ Toda conexión a la base de datos se debe realizar dentro de una sentencia
- ✓ Ejem: C#: using() ; BasicNet: Import
- ✓ Siempre debe utilizarse procedimientos almacenados para consultar o modificar los datos en la base de datos.
- ✓ Se debe realizar transacciones a través de una única conexión siempre que sea posible.
- ✓ Se debe evitar utilizar SQL Dinámico.
- ✓ Evitar en lo posible utilizar tablas temporales, de ser necesario utilizar variables tipo tabla.
- ✓ Evitar en lo posible almacenar archivos en la base de datos, en su lugar almacenarlo en algún tipo de repositorio para archivos y almacenar sólo enlaces
- ✓ Validar cada uno de los datos recibidos por la capa de datos, filtrar los caracteres no válidos
- ✓ Manejar adecuadamente los valores NULL
- ✓ Utilizar expresiones regulares para validar los datos complejos, en la capa de presentación.
- ✓ Identificar las excepciones que deben ser capturadas y manejadas en la capa de acceso a datos.
- ✓ Establecer la ubicación donde las excepciones son registradas y transformadas
- ✓ Se debe registrar un log con suficientes detalles y no revelar información confidencial.
- ✓ Se debe optar por una estrategia de CACHE, para el acceso a los datos cuya variabilidad sea muy poca.

IX. DISPOSICIÓN COMPLEMENTARIA

El presente documento deja sin efecto a las Normas para el Desarrollo de Aplicaciones (SD-N-005.01) aprobadas con fecha 10 de diciembre de 2010

X. APROBACIÓN

El presente procedimiento queda aprobado por Gerencia General y entra en vigencia a partir de la fecha de su suscripción.

Lima, 27 ENE. 2015

FRIBERG QUISPE GRAJEDA
Gerente General (e)
Serpost
El Correo del Perú

Anexo 1

Reglas de Codificación de Herramientas de Desarrollo C#

Existen dos tipos principales en el uso de mayúsculas y minúsculas, estos son:

- **CamelCase** – El primer término del nombre es siempre todo en minúsculas y para los siguientes términos que componen el nombre se utiliza mayúscula en la primera letra del término y el resto en minúsculas. No utilizar guion bajo (_) para separar términos.
- **PascalCase** – La primera letra de todos los términos que componen el nombre se encuentra en mayúscula y el resto en minúscula. No utilizar guion bajo (_).

La siguiente tabla describe las reglas para utilizar mayúsculas y minúsculas para los nombres de diferentes tipos de identificadores.

Identificado	Tipo	Estructura del nombre	Ejemplo
Clase	PascalCase	Sustantivo	public class ComplexNumber {...}
Namespace	PascalCase	Sustantivo No utilizar el mismo nombre para un namespace y un tipo contenido en el mismo.	namespace Microsoft.Sample.Windows7
Enum	PascalCase	Sustantivo Los enums marcados con el atributo Flags deben tener un nombre en plural, los demás en singular.	[Flags] public enum ConsoleModifiers { Alt, Contro l
Método	PascalCase	Verbo	public void Print() {...} public void ProcessItem() {...}
Propiedad Pública	PascalCase	Sustantivo o Adjetivo Los nombres de colecciones deben ser una frase en plural describiendo a los ítems de la colección y no un término en singular seguido de "List" o "Collection". Las propiedades con valores de verdadero o falso deben tener un nombre con una frase afirmativa. Opcionalmente se puede tener el prefijo "Is", "Can" o "Has" si hace más clara la intención de la propiedad.	public string CustomerName public ItemCollection Items public bool CanRead
Campos de Clase no públicos	_camelCase	Sustantivo o Adjetivo. Utilizar un guion bajo al inicio del nombre del campo.	private string _name;

Evento	PascalCase	Verbo
		Para eventos que tengan un concepto de antes y después utilizar tiempo pasado y presente.
		No utilizar prefijos o sufijos "Before" o "After" para indicar eventos pre y post.
			// A close event that is raised after the window is closed.
			public event WindowClosed
			
			// A close event that is raised before a window is closed.
			public event WindowClosing

Delegado	PascalCase	Agregar el sufijo "EventHandler" a los nombres de delegados que son usados en eventos. Agregar el sufijo "Callback" a los nombres de delegados que no son manejadores de evento. No agregar el sufijo "Delegate" a un delegado.	public delegate WindowClosedEventHandler
Interfaz	PascalCase con prefijo "I"	Sustantivo	public interface IDictionary
Constante	PascalCase si es pública; camelCase si es solo interna a la clase; Solamente utilizar todo en mayúsculas si es una abreviación de uno o dos caracteres.	Sustantivo	public const string MessageText = "A"; private const string messageText = "B"; public const double PI = 3.14159...;
Parámetro, Variable	camelCase	Sustantivo	int customerId;
Parámetro de tipo Genérico	PascalCase con prefijo "T"	Sustantivo Nombrar los parámetros de tipo generic con nombres descriptivos a menos que una sola letra sea suficiente para entender y un nombre más largo no agregara valor.	T, TItem, TPolicy
Recurso	PascalCase	Sustantivo Utilizar identificadores descriptivos en vez de cortos. Mantenerlos concisos cuando sea posible, pero no sacrificar legibilidad por espacio.	ArgumentExceptionInvalidName

Anexo 2

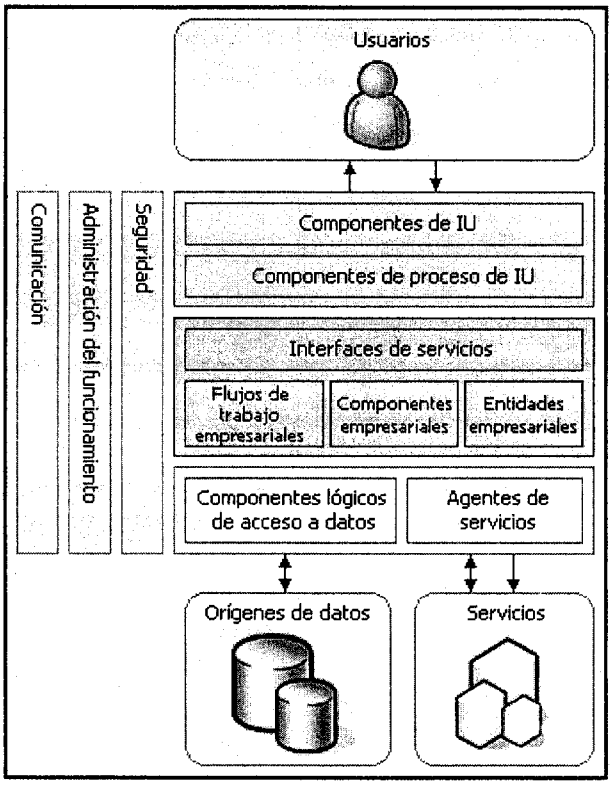
Objeto	Prefijo	Ejemplo
Animated button	Ani	AniMail
Check box	Chk	ChkAfecto
Combo box, drop-down list box	Cbo	CboIdioma
Command button	Cmd	CmdSalir
Common dialog	Dlg	DlgAbrirarchivo
Communications	Com	ComFax
Data-bound combo box	Dbcbo	DbcboMoneda
Data-bound grid	Dbgrd	DbgrdCuenta
Data-bound list box	Dblst	DblstMes
Data combo	Dbc	DbcAutor
Data grid	Dgd	DgdLibro
Data list	Dbl	DblAutor
Drive list box	Drv	DrvMipc
File list box	Fil	FilArchivos
Form	Frm	FrmProveedores
Frame	Fra	FraDatos
Graph	Gra	GraEstadistica
Grid	Grd	GrdPrecios
Image	Img	ImgEmpresa
Image combo	Imgcbo	ImgcboProducto
ImageList	Ils	IlsImagenes
Label	Lbl	lblTitulo
Line	Lin	LinVertical
List box	Lst	LstCodigos
ListView	Lvw	LvwDocumentos
MAPI message	Mpm	MpmMensaje
MAPI session	Mps	MpsSesion
MCI	Mci	MciVideo
Menu	Mnu	Mnuabrir
OLE container	Ole	Oleexcel
Option button	Opt	OptMasculino
ProgressBar	Prg	Prgavance
RichTextBox	Rtf	Rtffecha
Shape	Shp	ShpCircle
Slider	Sld	SldScale
Spin	Spn	SpnPaginas
StatusBar	Sta	StaFecha
SysInfo	Sys	SysMonitor
TabStrip	Tab	TabOpciones
Text box	Txt	TxtApellido
Timer	Tmr	TmrAlarma
Toolbar	Tlb	TlbAcciones
TreeView	Tre	Tremodulos
UpDown	Upd	UpdDireccion

Anexo 3

Objeto	Prefijo	Ejemplo
Command	AdoCmd	AdoCmdsql
Field	AdoFld	AdoFldDireccion
Parameter	AdoPrm	AdoPrmCodigo
Recordset	AdoRs	AdoRsProveedores
Connection	AdoCon	AdoConBDconta

Anexo 4

Relación de Capas



1. Capa Presentación:

1.1. Componentes de Interfaz de Usuario (UIC)

En aplicaciones de cliente inteligente se utilizan los componentes del namespace: System.Windows.Forms

En aplicaciones WEB, los componentes corresponden al namespace: System.Web.UI

1.2. Componentes de Proceso de Interfaz de Usuario (UPC)

Componentes personalizados, frameworks de navegación, librerías de funciones comunes, que faciliten el desarrollo de la interfaz de usuario. Estándares de Programación

2. Capa Lógica de Negocios

2.1. Componentes de Negocios (BC)

Son los componentes que implementan las reglas de negocio.

Se implementan en clases .NET que puedan utilizar los servicios de COM+ y/o Remoting.

2.2. Componentes de Workflow de Negocios (BW)

Son componentes que representan las actividades de un proceso de negocio.

Se implementan las clases .NET en conjunto con algún motor de flujos de trabajo u orquestador de procesos.

2.3. Entidades del Negocio (BE)

Son contenedores de datos. Representan las entidades persistentes del negocio. Se transportan entre las capas.

Pueden implementarse en clases .NET, DataSets tipados o DataSets sin tipar.

2.4. Interfaces de Servicio (SI)

La lógica de negocios se expone mediante servicios. Cada servicios posee una interfaz, es decir un protocolo que indica cómo debe ser invocado.
Puede implementarse utilizando Web Services o interfaces .NET.

3. Capa de Acceso a Datos

3.1. Componentes de Accesos a Datos (DAC)

Abstraen a la lógica de negocios de los detalles específicos de la persistencia. Estándares de Programación minimizan el impacto en caso de cambio de motor de base de datos o la representación de los datos.
Se implementan con ADO.NET y generalmente se construye un set de clases utilitarias para facilitar el desarrollo y abstraer la utilización de un proveedor específico (SQL Client, ORACLE Client, etc.)

3.2. Agentes de Servicio (SA)

Se encargan de manejar el acceso a aplicaciones y servicios externos.
Las tecnologías utilizadas son muy variadas y dependen de la aplicación externa.

Serpost
El Correo del Perú
Gerencia de Desarrollo Corporativo
27 ENE 2015 | 58
RECIBIDO
Hora: 17:19 Firma: