**ESTÁNDAR DE DESARROLLO**

Estándar de desarrollo

Medic Remote

Tacna, Junio 2020

V 1.0

**REVISION Y CONTROL DE CAMBIOS**

**Revisión y Versiones**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Nombres y Apellidos** |  |  | **Versión** |  |  | **Cargo** |  |  | **Fecha** |  |
|  |  |  |  |  |  |  |
|  |  |  | **Aprobada** |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | Roberto Zegarra Reyes |  |  | 1.0 |  |  | Gestor del Proyecto |  |  | 12/06/2020 |  |
|  | Marlon Xavier Villegas |  |  | 1.0 |  |  | Miembro del Proyecto |  |  | 12/06/2020 |  |

**Control de Cambios**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fecha** |  | **Autor** |  | **Versión** | **Descripción del cambio** |
|  |  | | |  |  |
| 12/06/2020 | Roberto Zegarra Reyes | | | 1.0 | Verificación de los capítulos de |
|  |  |  |  |  | Introducción y Generalidades del |
|  |  |  |  |  | estándar. |
| 12/06/2020 | Marlon Xavier Villegas | | | 1.0 | Estructuración de los capítulos del |
|  |  | documento |
|  |  |  |
| 12/06/2020 | Marlon Xavier Villegas | | | 1.0 | Se elaboran los capítulos 2.1 al 2.5 |
|  |  |  |
| 12/06/2020 | Marlon Xavier Villegas | | | 1.0 | Se elaboran los capítulos 2.6 al 2.7 y |
|  |  |  |

# Generalidades del Estándar

## Esencia de los Estándares

Los estándares son lineamientos, directrices y protocolos que se establecen con el propósito de normalizar la escritura del código que conforma un desarrollo de software, procurando la consistencia y reusabilidad del mismo. Emplear un lenguaje claro al definir cada uno de los elementos de software a través de convenciones permite mantener el código de manera eficaz y eficiente, así mismo agiliza el proceso de detección de errores toda vez que los estándares sean claros y se apliquen cabalmente.

Al considerar la selección del estándar de desarrollo se deben tener en cuenta aspectos técnicos como son la definición de variables por medio de una nomenclatura adecuada, la documentación requerida, las herramientas de verificación entre otros conceptos que serán objeto de estudio en este documento. Además de la selección del estándar se requiere el compromiso por parte de los desarrolladores de aplicar los lineamientos mencionados y cumplir con todas las especificaciones que sean solicitadas por la Dirección de Planeación y Sistemas.

Es necesario mencionar que cada desarrollo de software que se ha llevado a cabo sin aplicar ningún estándar en su construcción, requiere una dedicación especial de recurso humano y de tiempo debido a que se cuenta solamente con los conceptos aplicados arbitrariamente por el desarrollador a cargo; este comportamiento constituye una mala práctica en cuanto a construcción de software debido a que se limita la escalabilidad del sistema y su reusabilidad que son conceptos ampliamente utilizados al considerar aspectos como calidad y desempeño del software.

## Tipo de Dato

Todo lenguaje de programación posee una definición para cada tipo de dato que permite identificar el atributo que soporta y el tamaño de la información que puede contener. Los datos más comunes empleados por la mayoría de lenguajes de programación se denominan “Primitivos” estos tipos de datos son: Integer, Long, Boolean, String, Double, Object.

Es necesario tener cuidado al hacer uso del tipo de datos para definir un elemento como por ejemplo una variable, debido a que se puede hacer un uso indebido de la memoria cuando no se requiere; Por ejemplo para una variable que almacena un valor máximo de 99 no se requiere un tipo de dato ”Long” en su lugar es correcto definirla como “Integer”.

Asignar erróneamente un tipo de dato constituye una mala práctica y debe ser corregida para optimizar los bloques de código que se ejecuten. Esta directriz hace parte de la optimización del código que busca mejorar la aplicación del estándar.

# Implementación del Estándar

## Sangría

Como norma general se establecen 4 caracteres como unidad de sangría. Los entornos de desarrollo integrado (IDE) más populares, tales como Eclipse, NetBeans o Visual Studio .Net, incluyen facilidades para formatear código Java.

Como norma general se establecen 4 caracteres como unidad de sangría. Los entornos de desarrollo integrado (IDE) más populares, tales como Eclipse, NetBeans o Visual Studio .Net, incluyen facilidades para formatear código Java o .Net (C#).

## Longitud de Línea

La longitud de línea no debe superar los 80 caracteres por motivos de visualización

## División de Líneas

Cuando una expresión ocupe más de una línea, esta se podrá romper o dividir en función de los siguientes criterios,

•Tras una coma.

•Antes de un operador.

•Se recomienda las rupturas de nivel superior a las de nivel inferior.

•Alinear la nueva línea con el inicio de la expresión al mismo nivel que la línea anterior.

•Si las reglas anteriores generan código poco comprensible, entonces estableceremos tabulaciones de 8 espacios.

Ejemplos:

unMetodo(expresionLarga1, expresionLarga 2, expresionLarga 3,

expresionLarga 4, expresionLarga 5);

if ((condicion1 && condicion2)

|| (condicion3 && condicion4) ||!(condicion5 && condicion6))

{

unMetodo();

}

## Variables

* + 1. El nombre no podrá ser mayor a los 20 caracteres.
    2. El nombre no podrá constar de un simple carácter. La única excepción a esta regla es cuando se usan iteraciones (i, j, k).
    3. El nombre de una variable deberá ser descriptivo con respecto a su uso
    4. El nombre de una variable compuesto por más de una palabra deberá usar sub-guion, Ejemplo: "Usu\_Clave".

## Funciones

* + 1. El nombre de la función deberá ser descriptivo acorde a la tarea que realice.
    2. Las funciones deberán iniciar con mayúscula cada palabra descrita en ella, Ejm: MetodoListar()
    3. Las funciones deberán realizar una solo y específica acción no importando si solo se compone de una sola línea de código.

## Sentencias y Ciclos

* + 1. Las expresiones a ejecutar cuando una sentencia sea verdadera o cuando un ciclo se mantenga, deberán ser siempre del tipo compuestas, es decir siempre dentro de llaves. Incluso si no existe código a ejecutar:

if(condicion)

{

...;

}

while(condicion)

{

...;

}

for(cond.;cond.;cond.)

{

...;

}

* + 1. No se deben ejecutar operaciones dentro de las expresiones a evaluar en las sentencias de control y los ciclos.

Correcto

condicion -= 2;

if(condicion)

{

...;

}

Incorrecto

if(condicion -= 2)

{

...;

}

* + 1. Cada if o else if deberá tener su respectivo else, aunque no se ejecute nada dentro de este. Esto ayudará a tomar siempre en cuenta situaciones imprevistas.

if(condicion)

{

...;

}

else

{

/\*unexpected state\*/

}

* + 1. Se debe evitar la sentencia break dentro de todo ciclo, de requerirse romper el ciclo antes de su final, se deberá utilizar otros medios. Como una escritura a la variable contadora o centinela.

Correcto

for(i=0;i<5;i++)

{

if(condicion)

{

i = 5;

}

else

{

/\*unexpected state\*/

}

}

Incorrecto

for(i=0;i<5;i++)

{

if(condicion)

{

break;

}

else

{

/\*unexpected state\*/

}

}

## Comentarios

* + 1. Evitar comentarios anidados, pues no cumplen con el estándar ANSI C11.
    2. Escribir comentarios utilizando /\* <comentario> \*/ y evitar comentarios de tipo //
    3. Los comentarios deben ser descriptivos y no hacer eco de lo que explícitamente se lee en el código.
    4. No sobrecomentar el código. Un código bien escrito es un código implícitamente comentado.
    5. El encabezado de cualquier archivo de código deberá la licencia Beerware, asi como los datos de la persona que escribió el código.