**TSP - ESTANDARES DE CALIDAD**

**Layne Granados**

**Claudia María Vargas García**

**Rafael Andrés Ortega Pabón**

**Fabián Eduardo Becerra Pérez**

**Oscar Fabián Velásquez Jaime**

**CONCEPTOS AVANZADOS DE INGENIERÍA DE SOFTWARE**

**LUIS BENAVIDES**

**UNIVERSIDAD DE LOS ANDES**

**DEPARTAMENTO INGENIERÍA DE SISTEMAS**

**ESPECIALIZACIÓN CONSTRUCCIÓN DE SOFTWARE**

**CONTROL DE VERSIONES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Realizado Por** | **Fecha** | **Motivo** |
| **1.0** | Líder Calidad | 28/03/2014 | Versión Inicial Estrategia |

Contenido

[1. Estándar de Diseño 4](#_Toc383886486)

[2. Estándar de Codificación 4](#_Toc383886487)

[3. Estándares y reglas para manejo de GibHub 8](#_Toc383886488)

[4. Estándar de Documentación 8](#_Toc383886489)

[5. Estándar para Integración de Código 8](#_Toc383886490)

[6. Estándar para Manejo de Errores 9](#_Toc383886491)

[7. Estándar QUAL-Criterios de Calidad 9](#_Toc383886492)

# Estándar de Diseño

|  |  |
| --- | --- |
| **Responsabilidad simple** | Cada clase definida debe tener una finalidad sencilla y concreta. No asignar actividades o funcionalidades a clases que no contienen esa responsabilidad. |
| **Segregación de interfaces** | Definir interfaces con una finalidad concreta. Si la interfaz tiene varias funcionalidades que le agregan complejidad en su entendimiento y mantenimiento, es mejor definir varias interfaces pero cada una con un solo objetivo y así evitar posibles errores a futuro. |
| **Alta cohesión y bajo acoplamiento** | Realizar un diseño de tal manera que se cumpla los principios de alta cohesión y bajo acoplamiento en código. |
| **Funciones reutilizables** | Definir una clase que contenga funciones reutilizables para evitar código duplicado |
| **Nombre de clases** |  |

# Estándar de Codificación

|  |  |
| --- | --- |
| **Propósito** | Guía para implementar programas en JAVA |
| **Encabezados de los programas** | Todos los programas tendrán un encabezado descriptivo. Este encabezado se utilizará en el main o en la clase principal. |
| **Formato del Encabezado** | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*@see [Número de programa:] \*/  /\*@author: \*/  /\*@see [Fecha de desarrollo del programa:] \*/  /\*@version: \*/  /\*@see [Descripción: ] \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| **Identificadores** | Los nombres de las variables deben ser significativos. No se debe usar nombres de variables de una sola letra. Ejemplo: int p; |
| **Ejemplo de identificador** | Int numeroLineasLogicas; |
| **Comentarios** | * Los comentarios utilizados en los diferentes bloques de código deben explicar el funcionamiento y el propósito de este. * Comentarios de la declaración de las variables deben explicar su objetivo y funcionamiento. * En el código se debe documentar las clases, funciones y variables declaradas si su nombre por sí solo no es lo suficientemente claro para entender el funcionamiento de estas. * Cada línea de comentario debe iniciar con “/\*” o “//” * Los comentarios para documentar funciones, clases, constructores, entre otros, deben tener formato de javadoc y hacer uso de las etiquetas preestablecidas. * Se deben comentar todas las funciones y métodos con el objetivo de poder contabilizarlas en el programa psp0.1 * Los comentarios para las funciones debe tener la siguiente estructura:   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  /\*@función \*/  /\*@param \*/  /\*@return \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   * Los comentarios para los métodos debe tener la siguiente estructura:   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  /\*@método \*/  /\*@param \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   * Los comentarios para documentar las clases deben tener la siguiente estructura   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  /\*@author\*/  /\*@version \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| **Ejemplo de comentarios bien realizados** | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  /\*@función: Calcula la media de un listado de números reales \*/  /\*@param: Lista que contiene números reales \*/  /\*@return: El valor de la media del listado de los números reales \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| **Ejemplo de comentarios mal realizados** | /\* La función devuelve un número real\*/ |
| **Major Sections** | Para mejorar la organización de código en las clases y realizar aclaraciones, se utilizará “Major Sections”.  Para distinguir las secciones se utilizará 2 slash, 1 espacio y 8 asteriscos |
| **Example** | // \*\*\*\*\*\*\*\* Métodos de la clase  Public int getId(){  Return this.id;  } |
| **Espacios en blanco** | Utilizar espacios en blanco para separar bloques de código y hacer más fácil su lectura. |
| **Sangría** | * El código debe estar tabulado para facilitar su lectura * Todas las sentencias de iteración, selección deben utilizar corchetes. Los corchetes de apertura deben estar en la misma línea física que la sentencia de iteración, selección o de la declaración de funciones. |
| **Ejemplo de Sangría** | For(int i=0; i<cantidad\_lineas.size(); i++){  Valor=valor+cantidad\_lineas.get(i);  } |
| **Formato para declarar variables** | * Las variables se declararán en letra minúscula y si contienen más de una palabra, la primera letra de la segunda palabra comenzará en mayúscula. * Las constantes siempre se declararán en mayúsculas y si contienen más de una palabra, se separarán por medio de un subguión. |
| **Ejemplo de formato para declarar variables** | Int areaRectangulo;  static final int NUMERO\_MESES = 12; |

# Estándares y reglas para manejo de GibHub

* Antes de realizar commit, se debe actualizar los archivos de código, compilarlo y comprobar que este se ejecute correctamente.
* Al actualizar un fichero, incluir un comentario significativo que explique las modificaciones realizadas.

# Estándar de Documentación

Utilizar el estándar de Javadoc para documentar las clases del programa en java.

# Estándar para Integración de Código

* El software del proyecto se manejará a través del repositorio GibHub
* Para la asignación de código a desarrollar, se debe tratar de asignar funcionalidades independientes a los diferentes desarrolladores para facilitar el proceso de codificación.
* Cuando se desee modificar un código de otro desarrollador, se debe comunicar previamente la modificación al autor del código, para validar si el cambio no afecta otras funcionalidades del programa.
* Si un programador considera necesario modificar el diseño establecido para el desarrollo de una funcionalidad, estas modificaciones deben ser socializadas por todo el equipo, para actualizar el diseño y/o los formatos correspondientes.

# Estándar para Manejo de Errores

* Utilizar log4j para el manejo de errores
* En el log, imprimir información significativa y evitar imprimir información que ya se puede visualizar a través de la traza.
* Capturar los errores mediante try catch. En lo posible, solo utilizar un try y en el catch trazar todas las excepciones. Los catch deben ser ordenados desde el más específico al más general.
* Registrar en los formatos correspondientes de la metodología TSP, los errores detectados

# Estándar QUAL-Criterios de Calidad

|  |  |  |
| --- | --- | --- |
| **Medida** | **Objetivo** | **Comentarios** |
| **Porcentaje de defectos detectados** | | |
| Pruebas unitarias | >60% |  |
| Pruebas de integración | >70% |  |
| Pruebas de sistema | >90% |  |
| **Defectos/KLOC** | | |
| Total de defectos inyectados | 100-200 |  |
| Pruebas Unitarias | <10 |  |
| Construcción e Integración | <0.5 |  |
| Pruebas del sistema | <0.2 |  |
| **Ratio de defectos** | | |
| Defectos en revisión de diseño/Defectos en Pruebas Unitarias. | >2 |  |
| Defectos en revisión de código/Defectos en Pruebas Unitarias. | >2 |  |
| **Ratio en tiempos de desarrollo** | | |
| Inspección de requerimientos/tiempo de requerimientos | >0.25 |  |
| Inspección HLD/Tiempo HLD | >0.5 | HLD=Diseño de alto nivel |
| DLD/Tiempo de codificación | >1 | DLD=Diseño detallado |
| Revisión DLD/Tiempo DLD | >0.5 |  |
| Revisión de código/Tiempo de código | >0.5 |  |
| **Revisión e Inspección** | | |
| Páginas de requerimientos/hora | <2 |  |
| Páginas de HLD/hora | <5 |  |
| Código LOC/hora | <200 |  |
| **Ratio de Inyección de defectos** | | |
| Defectos de requerimientos inyectados/hora | 0.25 |  |
| Defectos HLD/hora | 0.25 |  |
| Defectos DLD/hora | 2 |  |
| Defectos código/hora | 4 |  |
| Defectos pruebas unitarias/hora | 0.2 |  |
| **Ratio de remoción de defectos** | | |
| Defectos removidos en la Inspección de requerimientos/hora | 0.5 |  |
| Defectos removidos en la inspección HLD/hora | 0.5 |  |
| Defectos revisión DLD/hora | 2 |  |
| Defectos Inspección DLD/hora | 0.5 |  |
| Defectos inspección de código/hora | 6 |  |
| Defectos de inspección de código/hora | 1 |  |
| **Fase Yield** | | |
| Inspección de requerimientos | -70% |  |
| Revisión de diseño e inspecciones | -70% |  |
| Defectos de pruebas unitarias/KLOC | -90% |  |
| Pruebas de construcción, integración y sistema< 1  defectos/KLOC | -80% |  |
| **Proceso Yield** | | |
| Antes de pruebas unitarias | >80% |  |
| Antes de construcción e integración | >93% |  |
| Antes de pruebas de sistema | >95% |  |