CALIDAD DE LOS PROGRAMAS

AUTOR: WALTER MADRIGAL CHAVES

NOVIEMBRE: 2020

San Marcos

Contenido

Introducción

Los Sistemas de Software son cada vez más importantes en la sociedad actual y aumentan rápidamente en tamaño y complejidad. Desarrollar software de calidad, basado en estándares, con funcionalidad y rendimiento ajustado a las necesidades de las empresas, son aspectos fundamentales para asegurar el éxito del producto software.

El termino de calidad aplicado al desarrollo de software no solo se refiere a que la interfaz del programa sea agradable, ni tampoco a que el sistema cumpla con su objetivo, este término es más complejo y analiza más variables que las comentadas, los detalles como documentación, calidad del código, rendimiento, eficiencia, seguridad y aplicación de mejores prácticas, siempre serán tomadas en cuenta para definir este criterio.

En la siguiente lectura se analizarán las mejores prácticas en el desarrollo del software, se abarcarán temas como utilización de nombres significativos, disposiciones de texto, documentación interna, modificadores de acceso, entre otros. Recuerden que no solo es crear un sistema es hacerlo bien.

Calidad de software

La calidad del software es el grado de desempeño de las principales características con las que debe cumplir un sistema computacional durante su ciclo de vida, dichas características de cierta manera garantizan que el cliente cuente con un sistema confiable, lo cual aumenta su satisfacción frente a la funcionalidad y eficiencia del sistema construido.

Es posible afirmar que un software es de calidad no solo cuando cumple correctamente la funcionalidad solicitada y le aporta valor al cliente que la usa, para considerar un software de calidad se tienen que evaluar muchos factores como, por ejemplo, si el costo de su mantenimiento es bajo, si la dificultad para introducir nuevos requerimientos también es baja y si es eficiente en la ejecución de sus procesos.

Es posible dividir la calidad del software en tres niveles, el primer nivel es el de usabilidad, en este nivel un software es de calidad si su funcionamiento es adecuado, si es amigable, intuitivo, si su apariencia es bonita, si hay una buena distribución de objetos y colores, por lo general es el usuario final es quien lo determina.

Nivel medio, enfocado a la funcionalidad, aquí se evalúa si el sistema hace lo que le corresponde o para lo que fue creado. Los usuarios expertos son los encargados de determinar este apartado.

El tercer nivel es el de funcionamiento, que lo que hace lo haga bien, el rastreo es más minucioso, se toman parámetros como rendimiento, eficiencia, calidad del código, entre otras, el personal experto es el encargado de realizar pruebas y verificar la aplicación de buenas prácticas.

Según (Gómez Blanes, 2020) muchos desarrolladores, por falta de experiencia o por presiones en tiempo e inclusive ausencia de una disciplina metodológica, se quedan atascados en el primer aspecto de la calidad, en donde únicamente

se debe cumplir con la usabilidad y punto.

Algunos puntos básicos para determinar si una aplicación es de calidad o no, son:

El código es simple: es importante desarrollar soluciones sencillas a problemas complejos.

El código legible: El código debe ser fácil de leer por cualquier miembro del equipo y debe poder ser asumido con facilidad por cualquier nuevo miembro que se incorpore.

El diseño y el código es homogéneo y coherente: se debe mantener el diseño seguido en toda la solución y estar alineada con el resto del código de la aplicación en estilo, uso de librerías externas, normas consensuadas de hacer las cosas, etc.

Pocos errores: El desarrollador dedica la mayor parte de su tiempo a añadir nueva funcionalidad, no a corregir bugs. Si se pasa mucho tiempo detectando o corrigiendo errores, entonces es señal que la aplicación se aleja de la definición de calidad.

No errores críticos: los errores menores pueden ser soportados y mitigados, pero un error grande, ya sea de concepto o de aplicación no puede darse.

Buenas prácticas de programación

Para (Gómez Blanes, 2020) el concepto de buenas prácticas se refiere a una experiencia que se ha implementado con resultados positivos, siendo eficaz y útil en un contexto concreto, contribuyendo al afrontamiento, regulación, mejora o solución de problemas y/o dificultades que se presenten en el trabajo diario de las personas en los ámbitos clínicos, de la gestión, satisfacción usuaria u otros, experiencia que pueden servir de modelo para otras organizaciones.

A nivel de desarrollo de sistemas las buenas prácticas son consideradas como un conjunto de reglas que se deben de seguir con el fin de estandarizar y mejorar la calidad del software. Algunas de estas reglas aplicadas a las sintaxis propias del lenguaje son:

Nombres significativos.

Al momento de crear elementos como variables, métodos, funciones, clases, etc. es muy importante hacer uso de nombres que representen la acción o función que el elemento va a fungir en el entorno del programa. Esto ayudará a que su código más legible y por consiguiente más fácil de entender.

Convenciones de escritura de nombres

Las convenciones de escritura de nombres son un conjunto de reglas para la elección de la secuencia de caracteres que se utilizan para nombrar elementos, estas reglas no son obligatorias, pero si recomendadas, existen muchas variantes y cada lenguaje de programación tiene sus propias nomenclaturas. A pesar de que la oferta es variada, la clave es apropiarse de una en específico para aplicarla en todo el proyecto.

Las dos principales ventajas de utilizar una sola convención son:

- Disminuir el esfuerzo requerido para leer y entender el código fuente.
- Mejorar la apariencia del código fuente (por ejemplo, al no permitir nombres excesivamente largos o abreviaturas poco claros).

Variables:

- Los nombres de las variables deben ser cortos pero significativos.
- No debería comenzar con un guion bajo (‘_’), números y caracteres especiales.
- Si es una sola palabra deberá ir en minúscula.

- Si está formada por más de una palabra, la primera letra de la segunda palabra estará en mayúscula.
- Pueden usar separadores como guiones bajos.
- Ejemplos: salario, salarioTotal, salario_total

Constantes:

- El nombre debe ir con todas sus letras en mayúsculas y las palabras separadas por el signo de guion bajo
- Ejemplos: IVA, COMISION_TEC

Clases:

- La primera letra debe ir en mayúsculas.
- Se recomienda usar sustantivos y en singular.
- Si está compuesta por más de una palabra usar pascal case (cada palabra comienza con mayúsculas).
- Usar nombres simples y descriptivos.
- Usar palabras completas, evitar acrónimos y abreviaturas en caso de no ser representativos.
- Ejemplo: VentasFisicas, VentasVirtuales

Métodos

- Deben comenzar con infinitivos.
- Notación camel case (primera palabra en minúsculas y las siguientes con la primera letra en mayúsculas).
- Se recomienda asociar el nombre del método con la acción que ejecutar.
- Ejemplo: insertar (), insertarSalario ().

Tabulación y espacios

La tabulación consiste desplazar hacia la derecha los bloques de código con el fin estructurarlo de una forma determinada y a su vez dar orden. Por lo general la primera línea de código no lleva espacio o tabulación, está inicia un bloque de contenido que va dentro del signo de llaves, este contenido forma nuevos sub-bloques, estos se van desplazando hacia la derecha para indicar esa subordinación.

A nivel práctico, la diferencia entre el uso de espacios o tabulaciones es nula. Cuando el código pasa por el compilador previo a su ejecución, la computadora interpreta de igual forma ambos formatos, sin embargo, existen diferencias técnicas que marcan la diferencia entre el uso de tabulaciones y espacios:

- **Precisión:** Una tabulación no es más que un conjunto de espacios agrupados. Por norma general, este conjunto suele ser de 8 caracteres, pero puede variar. cuando un mismo fichero de código se abre en dos máquinas diferentes, la apariencia del código puede ser diferente. En cambio, el uso de espacios no conlleva este problema: un espacio siempre ocupa el mismo “espacio” y asegura que el código se visualiza de la misma forma en todas las máquinas.

- **Comodidad**: En el caso de las tabulaciones, basta con pulsar la tecla de tabulación una única vez para estructurar correctamente el código. En el caso de los espacios, es necesario pulsar varias veces la misma tecla para lograr la estructura deseada.

- **Almacenamiento:** El uso de tabulaciones también reduce el tamaño el fichero final, mientras que el uso de espacios lo aumenta. Lo mismo sucedería con el uso de espacios en lugar de saltos de línea.

Delimitadores de bloques

Los delimitadores de bloques son necesarios, no utilizarlos significaría un error al momento de compilar. En java los bloques se delimitan con los símbolos {}, estos se pueden utilizar para delimitar una clase, una función, bucle, un método, entre otras cosas.

Líneas en blanco

Son simples espacios que se colocan entre los bloques de código, su único objetivo es dar una visualización más sencilla al código. Estas líneas son optativas.

Documentación interna.

Consiste en agregar comentarios al código que brinden suficiente información para explicar lo que hace cada elemento programado, con esto mejoramos la transferencia de conocimiento y el mantenimiento del sistema. Agregar documentación interna no es obligatorio, pero si es una muy buena práctica acostumbrase a hacerlo. Cada lenguaje de programación tiene sus propias sintaxis para crear comentarios, por ejemplo, en java existen tres maneras de crear comentarios:

La primera es cuando la línea de comentario solo ocupa una línea de código, en este caso hay que anteponer dos líneas inclinadas (//) antes del texto.

```
// Comentario de una línea
```

En el caso de realizar un comentario de más de una línea se debe empezar por una barra inclinada y un asterisco (/*) y finalizar a la inversa, asterisco y barra inclinada (*/). El código nos quedará de la siguiente forma:

```
/* Comentario
de varias
```

```
líneas */
```

El último caso son los comentarios para la herramienta de documentación JavaDoc. En este caso, antes del comentario pondremos una barra inclinada y dos asteriscos (/**) y finaliza con un asterisco y una barra inclinada (*/)

```
/** Comentario para JavaDoc */
```

Orden de las declaraciones

No existe un estándar definido para dar orden a las declaraciones, pero la siguiente distribución es una forma aceptada para organizarlas:

- La sentencia package debe ser la primera sentencia del archivo.
- La sentencia import debe seguir a las sentencias package.
- Las declaraciones de clase deberían organizarse. Esto debería hacerse de la siguiente manera:
    - Documentación de la Clase/Interfaz.
    - Sentencia class o interfaz.
    - Variables de clase (estáticas) en el orden public, protected, package (sin modificador de acceso), privadas.
    - Variables de instancia en el orden public, protected, package (sin modificador de acceso), private.
    - Constructores.
    - Métodos (sin orden específico).

Modificadores de acceso

Los modificadores son los que permiten a una clase, constructor, constante, variable, método o cualquier otro objeto a determinar quién accede a sus datos. existen cuatro diferentes:

- **Default o package-private:** se aplica cuando no se define ninguno,

serán accesibles dentro del mismo paquete.

- **Public:** Los elementos declarados bajo este modificador serán accesibles desde cualquier parte del programa.
- **Protected:** Estos elementos se pueden acceder desde la misma clase o en una clase heredada.
- **Private:** es el modificador más restrictivo y especifica que los elementos que lo utilizan sólo pueden ser accedidos desde la clase en la que se encuentran. Es importante destacar que private convierte los elementos en privados para otras clases, no para otras instancias de la clase.

En cuanto este tema, las mejoras prácticas indican que se debe restringir en la medida posible el acceso a los elementos, por lo que es recomendable declarar todos los atributos como private, y cuando necesitemos consultar su valor o modificarlo, utilicemos los métodos get y set.

Pruebas de programas

Realizar pruebas a los programas es una muy buena práctica de aplicar antes de entregar un proyecto, esto ayuda a ejercitar hondamente al sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de los sistemas de información con los que se comunica.

Existen muchos tipos de prueba, estos utilizan diferentes formas para evaluar y medir los diferentes elementos que componen un sistema, algunos de ellos son:

Pruebas unitarias

Este tipo de pruebas desgrana lo más posible el código, hasta un punto en que no se pueda desglosar más. A nivel de funciones y métodos las pruebas verifican que su nombre o identificador sea adecuado, que los nombres y tipos

de los parámetros sean correctos, y así mismo el tipo y valor de lo que se devuelve como resultado.

Con la finalidad de evaluar únicamente el código, al aplicar este tipo de pruebas se recomienda que las funcionalidades no tengan ningún tipo de dependencia de servicios externos, se suele reemplazar los llamados a APIs y servicios externos por funcionalidad que los imite (para que no exista interacción que vaya más allá de la unidad que está siendo probada).

Pruebas de integración

Este tipo de pruebas confirman que los diferentes módulos y servicios funcionen en armonía cuando trabajan en conjunto. Prueban la interacción con una o múltiples bases de datos y verifican que los servicios operen de la forma esperada.

Las pruebas de integración por lo general se realizan después de superadas las pruebas unitarias, son de mayor costo, ya que requieren que más partes de la aplicación se configuren y se encuentren en funcionamiento.

Pruebas de funcionalidad

Las pruebas funcionales se concentran en los requerimientos de negocio. Verifican la salida o resultados de una acción, sin prestar atención a los estados intermedios del sistema mientras se lleva a cabo la ejecución.

Suele haber confusión entre "integration tests" y "functional tests", debido a que ambas requieren que múltiples componentes interactúen entre sí. La diferencia radica en que una prueba de integración puede verificar que las consultas a una base de datos se ejecuten correctamente, mientras que una prueba funcional esperaría mostrar un valor específico a un usuario, en concordancia a lo definido por los requerimientos del producto.

End-to-end tests

En español se llaman pruebas de punto a punto, su función es simular el comportamiento de los usuarios con el software, en un entorno de aplicación completo. Este tipo de pruebas comprueban que los flujos que sigue un usuario trabajen como se planteó.

Por su complejidad y valor es recomendable tener unas pocas pruebas end-to-end, que resulten claves para nuestra aplicación, y confiar en mayor medida en las pruebas a bajo nivel como pruebas unitarias y pruebas de integración.

Pruebas de regresión

Estas pruebas verifican un conjunto de escenarios que sucedieron correctamente en el pasado, para asegurar que continúen así. Es importante no agregar nuevas características a la prueba de regresión hasta que las pruebas de regresión actuales pasen.
Una falla en una prueba de regresión puede significar que una nueva funcionalidad ha afectado otra funcionalidad que era correcta en el pasado.

Smoke testing

Este tipo de pruebas verifican la funcionalidad básica y explicita de una aplicación. Son pruebas rápidas de ejecutar y su objetivo es asegurar que las características más importantes del sistema funcionan como se espera.
Los smoke tests pueden ser muy útiles en los siguientes escenarios:

- Después de construir una nueva versión de nuestra aplicación.
- Después de un proceso de deployment, para asegurar que la aplicación está funcionando adecuadamente en el nuevo entorno desplegado.

Pruebas de aceptación

Son pruebas formales, que se emplean para verificar si un sistema cumple con

los requerimientos del negocio. Para que estas pruebas se den es necesario que el software se encuentre en funcionamiento y es el solicitante quien las aprueba.

Pruebas de rendimiento

En las pruebas de rendimiento simulan un ambiente de alta carga al sistema. Su objetivo es ver tiempos de respuestas, en consultas y procesos. Las pruebas de rendimiento son, por su naturaleza, bastante costosas de implementar y ejecutar, pero pueden ayudarnos a comprender si nuevos cambios van a degradar nuestro sistema (como hacerlo más lento o aumentar su consumo de recursos).

Conclusiones y recomendaciones

- Las buenas prácticas en programación, a pesar de que no son

obligatorias su aplicación, deberían tener un papel preponderante durante la escritura de código, es importante que los programadores se acostumbres a aplicarlas en todos sus proyectos y a si aprovechar todas las ventajas que acarrean.

- El tiempo invertido en aplicar las buenas prácticas, suele parecer mucho durante el proceso de construcción del proyecto, sin embargo, ese tiempo será retribuido en los mantenimientos y mejoras que se le pueden hacer al sistema.

- Así como es importante verificar que los usuarios pueden usar nuestra aplicación, es igual de importante verificar la calidad interna del sistema, su rendimiento, seguridad, capacidad, eficiencia, entre otros.

- Las pruebas son herramienta súper importantes que los programadores tienen antes de poner en producción un sistema, es mejor descubrir problemas durante las pruebas, en donde todavía hay tiempo de corregirlas, que en producción donde las consecuencias serían más grandes.

Referencias bibliográficas

Buriticá, O. T. (2017). *Lógica de programación.* Bogotá: Ediciones de la U.
Gómez Blanes, R. (01 de 12 de 2020). *Hub de libros*. Obtenido de

https://www.rafablanes.com/blog/elpdpa-que

Herrera Morales, J., Gutiérrez Posada, J., & Pulgarín Giraldo, R. (2017). *Introducción a la lógica de programación.* Armenia: ElIZCOM S.A.A.

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica