Tecnología de la programación

Práctica 2.



Buenas Prácticas de Prógramacion: En este documento se explica en que consiste la práctica que se ha realizado, su alcance, sus objetivos, las soluciones dadas, etcétera…

2013

Juan Luis Pérez Valbuena, Daniel Serrano Torres, Juan Carlos Marcos González y Álvaro Quesada Pimentel

23/01/2013

Índice

[**1.** **Requisitos** 2](#_Toc346703472)

[**2.** **Objetivos** 2](#_Toc346703473)

[**3.** **Resumen** 2](#_Toc346703474)

[**4.** **Abstract** 3](#_Toc346703475)

[**6.** **Sugerencias** 3](#_Toc346703476)

[**7.** **Introducción** 3](#_Toc346703477)

[**8.** **Objetivos que se alcanzarán** 4](#_Toc346703478)

[**9.** **Relación con la docencia cursada** 5](#_Toc346703479)

[**10.** **Viabilidad** 5](#_Toc346703480)

[**11.** **Estado del arte y fundamentación teórica** 5](#_Toc346703481)

[**12.** **Propuestas de solución** 6](#_Toc346703482)

[**13.** **Conclusión** 7](#_Toc346703483)

[**15.** **Bibliografía** 7](#_Toc346703484)

**PRÁCTICA DOS**

Buenas Prácticas de Programación

**Fecha de entrega:** 24/1/2013

1. **Requisitos**

* Para realizar esta práctica es necesario conocer los factores de calidad del software
* Es necesario conocer el lenguaje Java así como tener conocimientos de la herramienta eclipse
* No es estrictamente necesario pero si ayuda conocer lo que hace el código para así poder corregirlo con seguridad

1. **Objetivos**

**General**:

El objetivo de la práctica es comprobar lo costoso que es comprender y mantener un código sin ningún comentario y sin variables autodocumentadas. Por eso se pide que se modifique el código para que cumpla los factores de calidad del software, en esta práctica se pide en particular dos factores:

-**Mantenibilidad**: La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las existentes, depurar errores, o mejorar el rendimiento.

-**Legibilidad**: La legibilidad se refieres al modo en que se estructura la información (flujo de datos). En otras palabras, que esté claramente documentado, sangrado, sin errores, y que facilite su rápido entendimiento.

**Particulares**:

* Conseguir que compile y que funcione.
* Crear una batería de pruebas que testeen que el código funcione correctamente.

1. **Resumen**

A través de un código de poca calidad se ha transformado en una aplicación de calidad mediante los factores de calidad del Software. Se ha realizado clases fácilmente escalables y reutilizables en otras aplicaciones. Los comentarios son copiosos y la documentación, extensa. Se ha realizado una validación y verificación exhaustiva.

1. **Abstract**

Though a poorly quality code we have done a quality application though software quality factors. Classes has been made easily scalable and reusable in other applications. Comments are copious and documentation, extensive. We have done an exhaustive validation of the results and verification.

1. **Observaciones**

La práctica requiere mucho tiempo para poder entender que es lo que hace cada parte del código ya que está escrito de una manera antinatural y sin ninguna legibilidad, además el nombre de las variables no ayudan en nada, y que se haya modificado algunas palabras insertando espacios entre medias solo provoca un estado de exasperación hacia la persona que lo haya hecho.

1. **Sugerencias**

Pocas sugerencias se pueden hacer, ya que el objetivo de la práctica es exasperar al alumno, y eso lo consigue perfectamente.

Como única sugerencia seria que se adjuntara el fichero que usa la práctica, también que se aclararan mejor los objetivos que se deben de alcanzar para dicha práctica.

1. **Introducción**

Dado un código aportado por el profesor, el cual no contiene ningún factor de calidad del software, y en caso de contenerlo, es en muy poca extensión. Se pretende acondicionar lo máximo posible este código, aplicándole los factores de calidad del software para que pueda ser llevadero y comprensible para el próximo usuario que tenga que mantenerlo o ampliarlo.

Los factores de calidad del software sirven para tener una serie de reglas o pautas para saber si nuestra aplicación está realmente bien hecha.

Estos factores son:

1. **Corrección:**

Es la capacidad para realizar con exactitud sus tareas, tal y como se definen en las especificaciones.

1. **Robustez:**

Es la capacidad de los sistemas de software de reaccionar apropiadamente ante condiciones excepcionales.

1. **Extensibilidad:**

Es la facilidad de adaptar los productos de software a los cambios de especificación.

1. **Reutilización:**

Es la capacidad de los elementos de software de servir para la construcción de muchas aplicaciones diferentes.

1. **Compatibilidad:**

Es la facilidad de combinar unos elementos de software con otros.

1. **Eficiencia**

Es la capacidad de un sistema software para exigir la menor cantidad posible de recursos hardware.

1. **Portabilidad (transportabilidad)**

Es la facilidad de transferir los productos software a diferentes entornos hardware y software.

1. **Facilidad de uso:**

Es la facilidad con la cual personas con diferentes formaciones y aptitudes pueden aprender a usar los productos software y aplicarlos a la resolución de problemas.

1. **Integridad:**

Es la capacidad de un sistema para protegerse de elementos que no tengan derecho de acceso.

1. **Verificabilidad:**

Que la aplicación sea fácilmente verificable.

1. **Mantenibilidad:**

La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse.

1. **Objetivos que se alcanzarán**

* Comprobar por nosotros mismos lo difícil y molesto que es mantener un código sin legibilidad.
* Aprender métodos y herramientas que se utilizan para aumentar la legibilidad y mantenibilidad de un código.

1. **Relación con la docencia cursada**

* Esta práctica pone a prueba nuestros conocimientos de factores de calidad del software.
* También se tiene que conocer las formas que utilizamos para aumentar la legibilidad de un código.

1. **Viabilidad**

Esta práctica resulta viable dado que su envergadura y complejidad es menor, si bien todos los conceptos y conocimientos necesarios para realizarla eran nuevos y ha sido necesario que comprenderlos y conseguir utilizarlos correctamente.

1. **Estado del arte y fundamentación teórica**

Esta aplicación fue publicada por la universidad Roskildensis, universidad que se encuentra en Dinamarca. En la aplicación se da solución al problema del viajero, que consiste en recorrer una serie de puntos pasando por todos ellos en el menor tiempo posible. Para ello se fijan en las hormigas.

Las hormigas particularmente eficientes en un cosa, encontrar la ruta más corta entre dos puntos. Esto lo consiguen de la siguiente manera: Las hormigas van moviéndose al azar hasta que encuentran un punto de interés, supongamos comida, cuando vuelven al hormiguero van dejando una sustancia química a la que se le conoce como feromona, lo que les ayudara a encontrar el camino de vuelta hacia la comida. Cuando otras hormigas encuentran el camino de feromonas, estas lo seguirán, cada vez de una manera menos errática. Debido a que la feromona es una sustancia muy volátil, se necesita un flujo constante de hormigas para mantener el camino. Esto significa que si un camino más corto existe, el poder de las feromonas será más fuerte, ya que las hormigas lo atravesaran en un periodo más corto de tiempo. Después de un periodo corto de tiempo la mayoría de las hormigas seguirán el camino más corto, ya que las feromonas serán más fuertes en el.

Alberto Colorni, Maniezzo Vittorio y Marco Dorigo fueron los primeros en llegar con un algoritmo que simula el comportamiento de las hormigas, llamándolo Ant Colony System (Sistema de la colonia de hormigas) ACS, conocido comúnmente ACO, la O es por las optimizaciones que se hacen diariamente a la ACS original. Como ACO contiene métodos generales para la resolución de distintos problemas, puede ser visto como una colección de métodos heurísticos que se pueden aplicar a un gran número de problemas. ACO es parte de la ciencia “Inteligencia de enjambre”, que se basa en la utilización de pequeñas unidades independientes (por ejemplo: abejas, hormigas, aves) para resolver problemas. Solas estas unidades son tontas, pero cuando se les permite comunicarse entre sí y reaccionar a sus respuestas, son capaces de resolver problemas complejos.

1. **Propuestas de solución**

* Soluciones
  1. Primera solución:

La primera solución que se nos ocurrió fue la de comprender que hacia el código para así hacer una versión nuestra de ese código que cumpliera con los factores de calidad del software. En otras palabras volver a hacer la aplicación desde el principio.

* 1. Segunda solución:

La segunda solución fue la de modificar el código que ya estaba hecho para que cumpliera los factores de calidad del software. Para ello se analizaría el código e se intentaría conocer los que hace, para así poder cambiar poco a poco el nombre de las variables y métodos por nombres más significativos

Según se fuera comprendiendo, se dividiría este en clases más concretas, con sus respectivos métodos y atributos.

El resultado final sería:

- Un código sangrado, con mayor legibilidad que el anterior.

- Clases reutilizadas de prácticas anteriores para esta.

- Comentarios añadidos.

- Paquetes y clases nuevas.

- Algunas variables sin renombrar por desconocer su uso.

* Desestimaciones: Se descartó la primera solución, ya que requería un tiempo bastante considerable así como saber exactamente como funciona la aplicación para poder hacer nuestra versión.
* Solución escogida: Se eligió la segunda solución, porque el tiempo que requería era menor, además no era necesario saber exactamente que hacia cada parte del código, eso si el modificar un código ya hecho y con poca mantenibilidad y escalabilidad es tedioso y peligroso. Aun así en nuestra opinión nos pareció la manera más práctica y efectiva de hacerlo.

1. **Conclusión**

Al realizar esta práctica hemos comprendido mejor la importancia de los factores de calidad del software, y él porque es necesario llevarlos a cabo correctamente.

También se ha visto lo frustrante que es llevar a cabo el mantenimiento de un código con una baja legibilidad y porque es básico que se lleven a cabo sencillos pasos como son el sangrado o los comentarios.

1. **Líneas Futuras**

De cara al futuro se tendrá muy en cuenta los factores de calidad del software, y la importancia de ellos, así como la gran importancia que tiene legibilidad del código.

1. **Bibliografía**

**Estructura del documento actual:**

* Transparencias sobre la estructura de las prácticas, del profesor Francisco Javier Crespo.

**Programación orientada a objetos:**

* <http://www.elguille.info/colabora/NET2005/Percynet_ConstruyendoSoftCalidad.htm>
* PDF sobre la ACS de la universidad de Roskildensis.