SOFTWARE: PRINCIPIOS, HERRAMIENTAS Y COMUNIDAD TECNOLOGICA.

José Alejandro Gómez Castro
e-mail: jgomezc@ucenfotec.ac.cr

RESUMEN: *Se mostraran herramientas básicas y muy prácticas para desarrolladores de software, tratando de incentivar su interés y modo de mantenerse activo o actualizado en el área tecnología.*

PALABRAS CLAVE: Fundamentos de programación, Estructura de datos, Algoritmos, Blogs.

1 INTRODUCCIÓN

Este documento analiza y comenta brevemente distintas formas de mantenerse activo y aprovechar los beneficios que nos ofrece el internet. Se pretende aconsejar al lector para incentivar sus intereses en función de su propio beneficio. Se tocaran temas como principios de diseño, blogs y comunidades activas, cursos en línea, herramientas, lecturas y facilidades públicas, gratis y al alcance de todos.

1.1 CARACTERÍSTICAS GENERALES

Tomar riesgos a la hora de competir por un trabajo, desarrollando software desconocido o bien poco amigable, e impulsar la audacia a la hora de resolver problemas en el área del software.

Reforzar conceptos esenciales de software que buscan seguir el camino a diseñar e implementar las mejores prácticas y nuevas posibilidades buscando un desarrollo más ingenieril.

Libros, artículos, ensayos e investigaciones acerca de fundamentos, metodologías, estructuras de datos y algoritmos son la base esencial de un diseño adecuado y eficaz.

2 TEXTO PRINCIPAL

2.1 Conceptos esenciales

Conceptos que te ayudaran a pensar en cómo diseñar una mejor arquitectura de software:

Realizar el diseño a la hora de la construcción, contamina la solución, perturbando la eficiencia del programa **[1]** (Ciclos de vida del software).

El enfoque que debe tener un diseño depende del paradigma que elijamos y sobre esa base se impulsa la implementación.

La herencia eleva el acoplamiento en un sistema.

Los 5 niveles de acoplamiento son:

1. **Nulo** – No significa que no va a ver cierta relación.
2. **Exportación** – Que los objetos se comuniquen a través de las interfaces.
3. **Apertura** – Se tiene acceso a los atributos públicos de los objetos con que se relaciona.
4. **Cobertura** – El mismo alcance que el nivel de apertura pero en ambas direcciones.
5. **Total**.

2.2 Principios de diseño

[2] Podemos ver estos principios como una parte fundamental y básica del software, encontramos en ellos gran sabiduría que aportan grandes desarrolladores desde los años 60, hasta nuevas implementaciones.

Por fuerza el diseño del software se mantiene implícito en su implementación, el texto anterior nos sugiere que, si el código es actualizado a su vez modifica el diseño, por eso es importante mantener e implementar las buenas prácticas para ser mejores arquitectos de software.

Modularidad

Subdividir nuestra aplicación en módulos para enfocarnos en los aspectos esenciales de cada elemento, esto nos permite reducir la complejidad del sistema y delegar responsabilidades apropiadamente. Se pretende así lograr desarrollar aplicaciones débilmente acopladas (vínculo entre módulos) y fuertemente cohesivas (Modularidad y delegación de responsabilidades).

Polimorfismo

El polimorfismo implica una jerarquía de clases, esto nos permite tener una clase padre y una o muchas clases hijas que pueden verse como el padre, lo que nos da la posibilidad de implementar un mismo método de maneras distintas, además de que el cliente no necesita saber específicamente el tipo de las clases hijas para poder comunicarse.

Definition DAT (Data Abstract Type)

Realizar una representación abstracta de forma gráfica o mediante la herramienta de UML o incluso con formalismo matemático. Además es necesario establecer condiciones sobre la estructura interna del TAD, es decir, definir las invariantes. También debemos definir y clasificar las operaciones según su función. Por ultimo debemos tomar en cuenta el manejo de errores para brindarle el apoyo a construir un mejor diseño.

Heuristicas

Uso de heurísticas, enfocándonos en los objetivos fundamentales que son encontrar algoritmos con buenos tiempos de ejecución y mejores soluciones.

Ley de Demeter

Una unidad solo debe tener conocimiento limitado de otras unidades y solo conocer a aquellas que estén relacionadas. La unidad solo debe enlazar con amigos nunca con extraños. Además solo debe enlazar con amigos inmediatos.

Sustitución de Liskov

Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas. Este principio dice que una clase derivada no debe modificar el comportamiento de la clase base.

DRY (Don't Repeat Yourself)

No escribir código duplicado.

Open/Close

Un módulo debe ser diseñado para estar cerrado para ser modificado pero abierto para ser extendido.

KISS (Keep It Simple Stupid)

En referencia al código fuente de un programa no significa tratar con la optimización desde el principio, sino que tratan de mantener un estilo de programación simple y directo, dejando la optimización en etapas posteriores del desarrollo.

DIP (Dependency Inversion Principle)

Las clases de alto nivel no deben depender de las clases de bajo nivel, ambos deben depender de sus abstracciones uy en definitiva tratar de invertir las dependencias.

YAGNI (You ain't gonna need it)

No debemos escribir código innecesario.

Boy Scout

Refactorización y mantenimiento del código.

Patrones de diseño:

Son un conjunto de normas y reglas que guían el proceso de transición de las entidades conceptuales a unidades con la capacidad de ser implementadas (clases).

Entre los principales síntomas de un mal diseño mencionamos los siguientes:

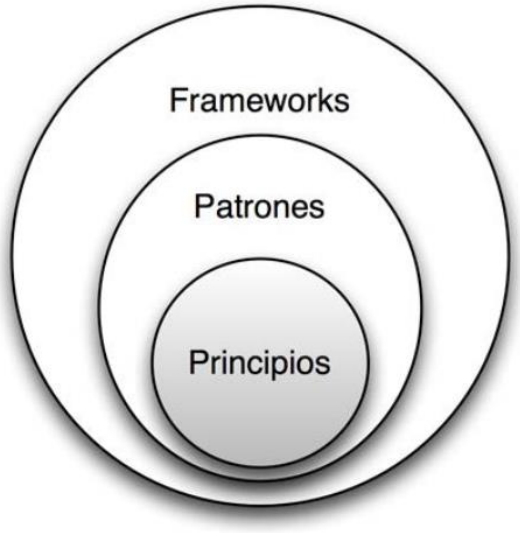
Usuario

- Falta de tiempo (Usuario)
- Evitar el uso de controles (Usuario y Software)
- Inaptitud de aprender (Usuario)
- Simplificación excesiva (Usuario y Software)
- Avaricia (Usuario)
- Orgullo (Usuario)
- Apatía (Usuario)

Sistema

- Seguridad (Software)
- Usabilidad (Software y Usuario)
- Portabilidad (Software)
- Robustez (Software)
- Rigidez (Software)
- Fragilidad (Software)
- Inmovilidad (Software)

- Viscosidad (Software)



2.3 Comunidades activas.

Una de las mejores maneras de mantenerse actualizado es siendo un usuario activo en los blogs de tecnología, programas relacionados, principios y buenas practicas, entre otros, hay muchas opciones. **[3]** El beneficio en ellas se centra en dos objetivos principales, el contenido del blog y la participación de sus seguidores.

Algunos ejemplos pueden ser:

ASP.NET Community Blogs
http://weblogs.asp.net/
http://odetocode.com/

DataBase Blogs
https://www.percona.com/blog/
https://www.topsqlblogs.com/

Developers Blogs
https://developers.googleblog.com/
http://www.vogella.com/tutorials/android.html
http://likecomtic.com/blog/

Hacking Blogs
http://siriushack.blogspot.com/
https://hacking-etico.com/
http://www.elladodelmal.com/

Una de las experiencias más educativas que nos demuestran que nosotros somos dueños de nuestro destino es el esfuerzo y dedicación que mostramos en un área determinada, internet está lleno de posibilidades para aprender nuevas y viejas tecnologías:

Cursos en línea
https://zenva.com/
https://www.edx.org/
https://www.patreon.com
http://www.newthinktank.com/
http://escuela.digital/
https://codigofacilito.com/
http://www.desarrolloweb.com/

Programados relacionados
https://teamtreehouse.com/library/the-treehouse-show
https://www.hackerrank.com/
https://cleancoders.com/

Libros y artículos recomendados

No Silver Bullet _Essence and Accident in Software Engineering **[4]**
The Gang of Four_Elements of Reusable Object-Oriented Software **[5]**
Algorithms_Fourth Edition **[6]**

2.4 Conclusión

La tecnología y en esencia el internet tiene para todos sus usuarios una enorme gama de posibilidades, es importante conocer acerca de grupos, entidades y comunidades que disponen su tiempo en ofrecernos profundas explicaciones con los elementos más efectivos de la educación. Por tanto navegar en internet de una manera responsable y orientada nos permite extraer las mejores experiencias y conocimientos de ella, no solo en el ámbito tecnológico, si no en sobre cualquier área de investigación, economía, entretenimiento, salud, ciencias, trabajo, actitud, etc.

3 REFERENCIAS

[1]
usr.code. (n.d.). *IMPLEMENTACION Y DEBUGGING : Ciclo de vida del Software.*

[2]
dev, G. (n.d.). Doce principios de diseño que todo desarrollador debería conocer.

[3]
Toptal. (n.d.). Toptal's Selection Of Best Developer Blogs. *developers*.

[4]
Jr, F. P. (1995). No Silver Bullet - Essence and accient in Software Engineering.

[5]
Erich Gamma, R. H. (n.d.). Design Patterns: Elemnts of Reausable Object-Oriented Software.

[6]

Robert Sedgewick, K. W. (n.d.). *Algorithms [Fourth Edition].*