***USO DE PATRONES DE DISEÑO: UN CASO PRÁCTICO***

Los patrones de diseño brindan soluciones a una serie de problemas comunes que se presentan en el desarrollo de software. Algunas soluciones son: facilitan la expansión del software, reduce la complejidad del código y facilitan el mantenimiento. En este articulo nos informa sobre la importancia de aplicar patrones en nuestro software el cual fue puesto a prueba en un proyecto de software sobre un simulador de un procesador multinúcleo, el cual inicialmente tiene restricciones que lo hacen muy simple, al aplicar los patrones de diseño se pudo extender la capacidad del procesador para simular una mayor diversidad de arquitectura.

Patrones aplicados para el proyecto:

* ***Fabrica (Factory):*** Su rol en el proyecto es crear un objecto que encapsule y se encargue de la creación del objecto.
* ***Singleton:*** Su rol define un tipo de objecto que posee como uno de su miembro una única instancia de si mismo.
* ***Indirección:*** Su rol en el proyecto en crear un objecto que maneje la comunicación entre los dos componentes
* ***Observador:*** consiste en definir una interfaz que implementa todos los objetos que estén interesados en un evento. El objeto que genere el evento avisará, por medio de esta interfaz, a todos estos componentes que el evento ha ocurrido.

***El patrón de arquitectura n-capas con orientación al dominio como solución en el diseño de aplicaciones empresariales.***

El articulo nos habla sobre un modelo de software aplicando el patrón de arquitectura N-Capas con orientación al dominio y el patrón de diseño fabrica Abstracta. El propósito de aplicar estos patrones en el diseño de Aplicaciones Empresariales complejas es para garantizar que sean flexibles y Fácilmente mantenibles.

***Patrón capaz:*** El patrón arquitectónico capas nos ayuda con la estructuración de las aplicaciones que se pueden descomponer en subtareas en la que cada grupo esta en un nivel particular de abstracción. Este patrón cuenta con extensiones como : ***Patrón N-Capas con Orientación al Dominio, capas de presentación, capas de servicios distribuidos, capas de aplicación, capas del dominio, capa de presentación de datos.***

***Patrón fabrica abstracta:*** una fabrica abstracta es una clase que proporciona una interfaz para producir una familia de objectos.

***Las clases que participan en este patrón son:***

***Fabrica Abstracta:*** declara una interfaz para operaciones que crean objectos productos abstractos.

***Fabrica concreta:*** implementa las operaciones para crear objectos productos concretos

***Producto Abstracto:*** declara una interfaz para un tipo de objeto producto

***Producto concreto:*** define un objecto producto para que sea creado por la fábrica correspondiente implementando la interfaz producto abstracto

***Modelado y Verificación de Patrones de Diseño de Arquitectura de Software***

Este artículo nos habla sobre el diseño integral que permite formular diseños de arquitectura de software destinadas a la representación de aplicaciones web. Este entorno abstrae los principales problemas identificados a diseño de calidad. Para este se utiliza como base un metamodelo ***(un modelo que describe como se construye otros modelos)*** para componentes arquitectónicos que identifican un conjunto de elementos comúnmente utilizados en ciertas arquitecturas. Sobre el modelo se construye una herramienta de instanciación grafica que se complementa con la verificación de patrones de diseño a fin garantizar su correcta Aplicación.

Se ha convertido rápidamente en una de las estrategias de solución tecnológica más populares e influyentes del mundo actual, en este trabajo se han especificado los lineamientos requeridos para la construcción de un entorno de trabajo que facilite la tarea de diseño arquitectónico. Dicho entorno queda definida como un metamodelo, el cual describe un conjunto de elementos obtenidos a partir del estudio de múltiples patrones de diseño.

***APLICABILIDAD DE LA ARQUITECTURA MVC EN UNA APLICACIÓN WEB (WebApps)***

Este articulo nos habla sobre el nacimiento de la ingeniería de software las empresas han estado buscado cada día mas desarrolladores de calidad dentro de su presupuesto. A causa de las deficiencias encontradas en el desarrollo, se han creado nuevas arquitecturas. Como el suso de patrones de arquitectura han sido utilizados fuertemente en las empresas entre ellos nos encontramos con el patrón ***MVC*** (***Modelo-Vista-Controlador)*** Es uno de los que mas se destaca. Con esté patrón se buscan abordar ventajas y desventajas, la facilidad en el desarrollo, la alta mantenibilidad, el alto aprovechamiento de código, para las empresas poder alcanzar estos objetivos, hacen un alto levantamiento bibliográfico donde se encarga de recolectar la mayor información sobre el patrón ***MVC*** y llegar comprenderlo. Para lograr esto se desarrolló una ***aplicación web*** utilizando el ***Framewok Asp.Net MVC 4.0,*** lo que permite demostrar las facilidades en el mantenimiento y la prueba del sistema. En el proceso de modelado el patrón MVC requiere más tiempo para analizar y modelar el sistema, lo que nos recomienda para proyectos pequeños, pero muy esencial para proyectos grandes cuyo control y organización del desarrollo se obtienen de manera eficiente y satisfactoria.

***PAQUETE JAVA PARA LA INTEGRACIÓN DE HERRAMIENTAS DE PRUEBAS DE SOFTWARE BASADO EN PATRONES DE SOFTWARE***

Este artículo se centra como objetivo diseñar y construir un adaptador que permita la integración de tecnologías y herramientas de pruebas basado en patrones de software la cual facilita el desarrollo de aplicaciones de automatización.

Los patrones de diseño de arquitectura son soluciones para problemas recurrentes en el desarrollo, al aplicar un patrón incorrectamente este puede llegar a generar problemas de rendimiento, seguridad o mantenibilidad.

Al verificar que el patrón este correctamente implementado es muy fundamental para garantizar la calidad del software.

***Unas de las metodologías propuestas son:***

***Modelado formal del patrón:*** El uso de diagramas y especificaciones formales para describir la estructura y el comportamiento esperado uno de los ejemplos que nos da es el patrón **Layered Architecture** modelado en UML.

***Trasformación del modelo en un formato verificable :***  la conversión de diagramas a un lenguaje formal como Alloy para su verificación automática

***Verificación de propiedades:*** Define si el patrón cumple con ciertas propiedades como responsabilidad única, inversión de dependencias etc..

Esta metodología nos permite detectar errores en implementaciones que a simple vista parecen correctas, la formalización ayuda a tener documentación clara y no ambigua del patrón, la verificación automática nos ayuda a la reducción de riesgos y asegura la calidad desde las primeras etapas.

***Framework para la generación del BACK-END de una aplicación a partir de objetos JSON (Generador JSON-NODEJS) bajo principios SOLID-(GCAE).***

Esta investigación nos empieza a hablar sobre el crecimiento acelerado de las empresas para adaptarse a las nuevas realidades y poder subsistir en el mercado, para un usuario final la inmediatez a la información juega un papel importante, y ahí es donde la calidad del servicio que la empresa ofrece gana protagonismo este a llegado a dar lugar a una exigencia calidad del software y la existencias de múltiples problemas que afectan a la calidad, dando a conocer este problema esta investigación trata sobre un proyecto que propone un software generador de código en entorno empresariales llegando a la utilización de estrategias de calidad como lo son los principios ***SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversión)*** cuyo objetivo es en la contribución de la calidad del software y promover una estandarización en la arquitectura del software que incide en la etapa del diseño y la implementación del software, favoreciendo a la construcción e software de calidad y así evita una dependencia total de programadores.

***Como mantener el patrón modelo vista controlador en una aplicación orientada a la web***

Este artículo nos inicia hablando sobre la actualidad en el desarrollo de software orientada a la web, ya es impredecible la actualización de patrones de diseño que permitan el mejoramiento de la calidad del software otorgando a las aplicaciones características que les permita ser fácilmente mantenibles, flexibles y evolutivas.

Uno de los patrones básicos que nos mencionan para el desarrollo de aplicaciones orientadas a la web es la modelo vista controlador ***(MVC)*** el cual nos permite hacer una separación en tres partes graficas de la aplicación y los procesos de esta misma, Este articulo presenta la actualización del patrón Modelo Vista Controlador en el desarrollo de las aplicaciones orientadas a la web y como este a su ves se de adaptar al constante cambio de tecnologías. Se hace una mirada a la evolución del desarrollo web haciendo énfasis en la influencia de las tecnologías como AJAX, FLEX y Openlaszlo. Finamelote nos menciona sobre la posibilidad de la unificación de todas las tecnologías para llegar a la creación de aplicaciones web que funcionen completamente del lado del cliente, y que todos los procesos complejos actuales de la arquitectura Cliente-Servidor se reduzcan a la simple toma de información que se encuentran en servidores de bases de datos.

***Principio SOLID en la automatización de pruebas de software para interfaces de Usuarios de web con selenium webDriver y java***

Este articulo esta centrado en analizar de aplicaciones de los principios solid automatización de pruebas de interfaces web usando Selenium WebDriver y java, este con el fin de lograr código mas comprensible, mantenible y escalable. Nos explica cada principio como Responsabilidad ***Única, Abierto/Cerrado, Sustitución de Liskov, Segregación de Interfaces e Inversión de Dependencias*** también nos dice de como podemos adaptarlos a Frameworks de pruebas, identificando violaciones comunes de estos y proponiendo refactorizaciones. También nos menciona algunas herramientas como el ***patrón (POM) Page Object Model y Factory*** para la separación de responsabilidades y centralizar cambios de la interfaz. Se abordan casos prácticos como la separación de objectos y acciones en clases distintas, adaptar menú según el rol, la división de interfaces para evitar dependencias innecesarias, también nos menciona el uso de java para la versatilidad de selenium. Los resultados muestran que aplicar solid mejora la calidad del código, también facilita la adaptación a cambios sin comprender la estabilidad, esto nos garantiza la reducción de costo en mantenimiento del software asegurando pruebas mas eficientes y un desarrollo continuo de calidad.

***Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web***

Este articulo se presenta el análisis de identificación de patrones de diseño definidos por ***THE GANG OF FOUR (GOF)*** en procesos de desarrollo de software orientados a la web. Inicialmente se construye un conjunto de criterios para evaluar y seleccionar procesos de desarrollo formales de gran envergadura. Se establece en Tamaño de la muestra para aplicar los criterios con estricto rigor metodológico, se inicia la inspección del código fuente para identificar el uso de patrones de diseño llevado a cabo un proceso que nos permite la identificación de los patrones de diseño que son utilizados por expertos del área de la ingeniería del software. Los resultados nos permiten llegar a la conclusión que en el sector productivo los patrones de diseño han sido aplicados, mas sin embargo su uso es reducido por falta de conocimiento de la existencia de estos patrones o por falta de experiencia parta lograr su correcta utilización.

***Análisis de patrones de resiliencia en una arquitectura basada en microservicios***

Esta tesis explora el enfoque **Domain-Driven Design (DDD)** aplicado al desarrollo de aplicaciones empresariales, analizando su efectividad y los retos que presenta en la práctica. El autor parte de la premisa de que muchos proyectos de software fracasan porque el diseño no refleja con precisión el dominio del negocio. Para abordarlo, ***DDD*** propone trabajar de forma colaborativa entre expertos del negocio y desarrolladores, construyendo un **modelo de dominio** claro y compartido.

La investigación combina un marco teórico con un caso práctico, en el que se diseña y desarrolla una aplicación empresarial siguiendo los principios de ***DDD***: creación de un **lenguaje ubicuo** para evitar ambigüedades, modelado de entidades, agregados, repositorios y servicios de dominio. También se evalúan patrones estratégicos como **Bounded Context** **y** **Context Mapping** para organizar mejor las responsabilidades y dependencias del sistema.

El trabajo concluye que DDD favorece la alineación entre el software y las necesidades reales del negocio, mejora la comunicación en el equipo y facilita la evolución del sistema. También requiere una curva de aprendizaje significativa y una inversión de tiempo inicial mayor. El autor considera que sus beneficios en calidad, mantenimiento y escalabilidad superan ampliamente las dificultades iniciales.

***PROPUESTA DE PATRÓN DE DISEÑO DE SOFTWARE ORIENTADO A PREVENIR LA EXTRACCIÓN AUTOMATIZADA DE CONTENIDO WEB.***

Este articulo no habla sobre el proceso de la recolección de información de uno o más sitios web de manera automatizada, emulando la interacción entre un usuario y un servidor, estos se basan en el análisis de estructuras HTML y nos indica que no requiere de la autorización de propiedades.

El articulo nos habla sobre el uso de las estructuras repetitivas o plantillas, que facilitan la funcionalidad de un programa informático que extrae contenido web, dicha intrusión genera un incremento considerable en el uso de recursos, considerando la permanente ejecución de instrucciones para obtener tanto contenido como sea posible.

Este con la finalidad de reducir la vulnerabilidad de los sitios web frente a procesos de extracción de contenido masivo, En el artículo se planteó un patrón de diseño de software tomando como referencia el patrón ***Témplate View de Martin Fowler***, al cual se le agrego una capa de aleatorización que permita generar estructura HTML no predecibles.

Este se lleva a cabo mediante una aplicación de una herramienta de extracción de contenido a un sitio web de prueba, cuya capa de presentación se desarrollo tomando en cuenta el patrón de diseño que fue propuesto, se logró verificar una reducción considerable de la calidad de datos extraídos.

***Factory Design Pattern***

Este trabajo explica de forma clara el ***patrón de diseño Factory***, uno de los más usados en la programación moderna, especialmente en Java. El autor parte recordando que los patrones de diseño son soluciones reutilizables a problemas comunes y que el Factory pertenece al grupo de **patrones creacionales**, cuyo objetivo es simplificar y estandarizar la creación de objetos, reduciendo el acoplamiento y mejorando la escalabilidad.

El patrón Factory define una **interfaz para crear objetos** en una clase base, permitiendo que las subclases decidan qué tipo de objeto instanciar. Esto facilita agregar nuevos productos sin modificar el código existente, centralizando la lógica de creación y mejorando el mantenimiento. El documento presenta ejemplos prácticos, como un sistema de logística que maneja camiones y barcos, y un generador de figuras geométricas (círculo, rectángulo, cuadrado), ilustrando cómo el patrón permite extender funcionalidades sin romper la estructura.

En la revisión de trabajos previos, se mencionan ventajas como personalización y reutilización, así como retos como el aumento de la complejidad al requerir más clases. El autor concluye que el patrón Factory es ideal para aplicaciones que necesitan flexibilidad y crecimiento a largo plazo, siempre que se use con criterio para no sobre complicar el diseño.

***Generación Automática de Interfaces a Partir de Patrones Estructurales de Tareas***

Este articulo nos habla sobre técnicas de especificación de requisitos de interacción basada en construir un Modelo de Tareas usando patrones estructurales de tareas, a partir de estas se puede derivar automáticamente el Modelo de presentación. Ya que esta técnica mejora el proceso de desarrollo que soportado por la tecnología ONME, permite generar automáticamente el código de la aplicación a partir de los modelos de presentación, objectos, dinámico y funcional.

Esta experiencia nos indica que los diagramas CTT pueden llegar a convertirse en ilegibles para aplicaciones medianas. Para esto usa el proceso de generación de los CTT debe ser oculto para el analista. Se propuso como solución el uso de bocetos ***(SKETCHES)*** para generar los diagramas. Gracias a estas reglas los analistas están limitados a utilizar las reglas existentes. Esta limitación afecta a la construcción de los bocetos.

***Patrones De Alexander a la Tecnología de Objetos***

Este articulo nos infirma sobre El concepto de patrones como elemento reutilizable de experiencia y conocimiento ha calado profundamente en el área del desarrollo de aplicaciones software, teniendo mas activa la comunidad de orientación de objecto.

De hecho, nos indica que deriva el termino patrón software y mas concretamente el de patrón de diseñó para hacer referencia al uso de patrones en ***el diseño orientado a objecto (DOO).***

objetivo familiarizar al lector con el uso de patrones en el diseño de sus aplicaciones software, consiguiendo un mayor grado de fiabilidad y flexibilidad en sus diseños al reutilizar estructuras ya probadas en otros contextos, siendo además los sistemas software construidos más sencillos de comprender y mantener al estar documentados sobre la base de un conjunto de patrones ampliamente conocidos y difundidos.

No obstante, a lo indicado antes de empezar a trabajar con los patrones en el campo de ***diseño orientado a objecto*** ***(DOO)*** y de cara para evitar problemas, mitos o creencias en panaceas que den soluciones a todos los males del software, nos da una primera aproximación a los patrones en la que se explica su origen, su evolución hacia el mundo del software, su significado, la terminología que introducen, así como un conjunto de referencias.

***Guía práctica para el uso de patrones de diseño en el desarrollo de software***

Este articulo presenta un estudio sobre la ***arquitectura Onion*** aplicada al desarrollo de software, destacando su combinación con el enfoque ***Domain-Driven Design (DDD)*** para lograr sistemas más mantenibles, escalables y con bajo acoplamiento. La arquitectura Onion organiza el código en capas concéntricas donde las dependencias siempre apuntan hacia el núcleo, que contiene el dominio y las reglas de negocio. Las capas externas manejan infraestructura, interfaces y acceso a datos, permitiendo reemplazar tecnologías sin afectar el núcleo.

El trabajo explica que integrar DDD con Onion Architecture refuerza la separación de responsabilidades y la claridad en el modelo de dominio. El caso de estudio descrito es el desarrollo de un sistema de gestión de dormitorios universitarios (MyITS Dorm), implementado siguiendo estas prácticas. Se detalla cómo la aplicación se diseñó dividiendo las funciones en capas y respetando la dirección de dependencias, lo que facilitó pruebas unitarias y la incorporación de cambios.

Entre las ventajas encontradas se incluyen un código más limpio, facilidad para realizar pruebas y adaptabilidad tecnológica. Las desventajas señaladas son la mayor complejidad inicial y la necesidad de experiencia previa en diseño orientado al dominio. El estudio concluye que esta combinación de enfoques es especialmente útil en proyectos de gran escala que requieren evolución a largo plazo.

***Buenas prácticas de diseño de software con SOLID***

Este articulo propone usar técnicas DCU para diseñar una aplicación web de gestión de identidad y datos personales online. Tomando referencia el prototipo “Solid Community” se pretende su accesibilidad y experiencia de uso general, además de corregir errores de usabilidad que evitan que la aplicación pueda ser usada por cualquier usuario sin concomimientos técnicos.

Según la investigación que se realizo los participantes en las entrevistas definieron identidad online de manera informal, aquellos datos de carácter personal que se encuentran públicos en la web. De manera que ofrece una aplicación web que sea capaz de reunir datos, que permita gestionarlos y tenerlos actualizados, y controlar el acceso a los mismo, cumple el objetivo general planteado.

También nos menciona que se logró refutar la hipótesis planteada del proceso DCU en el diseño y desarrollo del proyecto solid ya que la satisfacción subjetiva de los usuarios fue mucho mayor al usar la solución propuesta por este trabajo, en comparación con la exploración del POD de referencia “SOLID COMMUNITY” se puede afirmar que parte de los objetivos particulares que pasaban por mejorar su usabilidad y accesibilidad, también se cumple.

***An Overview of Structural Design Patterns in Object-Oriented Software Engineering***

Este documento explica qué son los **patrones de diseño** en la ingeniería de software y cómo ayudan a resolver problemas comunes en el desarrollo. Son soluciones ya probadas que facilitan crear software más organizado, fácil de mantener y ampliar. Los patrones se dividen en tres grupos principales: creacionales, estructurales y de comportamiento.

***Adapter***: permite que dos cosas que no son compatibles trabajen juntas.

**Bridge**: separa la abstracción de la implementación para que evolucionen de forma independiente.

***Composite***: trata igual a objetos individuales y a grupos de objetos.

***Decorator***: añade funciones extra a un objeto sin cambiar su estructura.

***Façade***: crea una interfaz sencilla para usar sistemas complejos.

***Flyweight***: ahorra memoria compartiendo datos comunes entre objetos.

***Proxy***: actúa como intermediario para controlar el acceso a un objeto.

Aunque estos patrones son muy útiles, en el articulo advierte que hay que usarlos con cuidado, ya que aplicarlos sin necesidad puede complicar innecesariamente el código.

***Implementación de un framework para el desarrollo de aplicaciones web utilizando patrones de diseño y arquitectura MVC/RES***

Esta tesina aborda el uso de **metodologías ágiles** en el desarrollo de software, haciendo especial énfasis en ***Scrum*** como marco de trabajo para mejorar la gestión y la calidad de los proyectos. El documento inicia explicando cómo el desarrollo de software ha pasado de modelos tradicionales, como en **cascada**, a enfoques más flexibles que permiten adaptarse a cambios y entregas rápidas.

Se describe Scrum como un método basado en iteraciones cortas **(sprints),** con roles bien definidos ***(Product Owner, Scrum Master y Equipo de Desarrollo)*** y eventos clave como la planificación, las reuniones diarias, la revisión y la retrospectiva. También se explica el uso del ***Product Backlog*** *y* ***Sprint Backlog*** para organizar y priorizar tareas.

La autora incluye un caso práctico en el que se implementa Scrum en un proyecto real, mostrando cómo esta metodología favorece la comunicación entre los miembros del equipo, la identificación temprana de problemas y la mejora continua.

Entre las conclusiones, se destaca que Scrum permite una mejor adaptación a los cambios del cliente, entrega valor de forma más rápida y fomenta la colaboración. No obstante, también se reconoce que su correcta aplicación requiere compromiso, disciplina y experiencia en el trabajo en equipo.

***Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web***

En este Articulo se analiza cómo se identifican los **Patrones de Diseño** propuestos por The Gang of Four (GOF) dentro de procesos de desarrollo de software para la web. Primero, se definieron criterios para evaluar y seleccionar procesos de desarrollo grandes y formales. Luego, se determinó el tamaño de la muestra y se aplicaron estos criterios de forma rigurosa.

Posteriormente, se revisó el código fuente para detectar si se usaban patrones de diseño y se siguió un proceso para identificar cuáles eran aplicados por expertos en ingeniería de software.

Los resultados muestran que, aunque en el sector productivo sí se utilizan estos patrones, su uso todavía es limitado. Esto se debe principalmente a que muchos desarrolladores desconocen su existencia o no tienen la experiencia suficiente para aplicarlos correctamente.

***Análisis comparativo de Patrones de Diseño de Softw***

Este artículo explica que los **patrones de diseño** son herramientas que ayudan a resolver problemas comunes en el desarrollo de software, evitando duplicar código y facilitando su reutilización. El estudio se enfoca en analizar la estructura, componentes, ventajas y desventajas de cinco patrones específicos*:* ***Template Method****,* ***Model-View-Controller (MVC)****,* ***Model-View-Presenter (MVP)****,* ***Front Controller*** *y* ***Model-View-ViewModel (MVVM)****.*

La investigación se basó en una revisión bibliográfica de fuentes científicas y en el uso de métricas para comparar los patrones. El análisis reveló que no existe un patrón que sea el mejor en todos los casos; cada uno tiene un objetivo específico, y es el desarrollador quien debe decidir cuál se ajusta mejor a la solución que necesita implementar.