Resumen y reflexión de Artículos relacionados con “Patrones de diseño” y “Arquitectura de Software”

Cristian Fernando Narváez Sánchez

Centro de la Industria, La empresa y los Servicios

Análisis y Desarrollo de Software

Jesús Ariel González Bonilla

Neiva, Colombia

15 de Noviembre de 2022

Tabla de contenido

Introducción

En este documento Word, se encontrará con distintos resúmenes, reflexiones y Diagramas de los artículos a investigar relacionados con patrones de diseño y arquitectura de software asignado por el instructor académico Jesús Ariel González Bonilla. Cada articulo cuenta con diferentes formas de aplicar patrones de diseño e implementar una arquitectura única y sencilla a proyectos formativos y laborales haciendo cada vez más flexible, reutilizable y escalable el código.

Articulo N°1: Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

Resumen:

El diseño de arquitecturas de software organiza componentes para cumplir con los requisitos del sistema. En el entorno de computación en la nube (Cloud Computing), las arquitecturas suelen dividirse en dos capas: una capa de aplicación que contiene las funcionalidades y otra de infraestructura que soporta estos recursos. Los arquitectos de software deben equilibrar atributos de calidad, como rendimiento y seguridad, mientras diseñan arquitecturas que se puedan desplegar en la nube.

Uno de los desafíos es la falta de patrones de diseño específicos para la nube y la mezcla de elementos de infraestructura y software, lo que dificulta la comprensión para diseñadores menos experimentados. Por eso, se emplean arquitecturas genéricas que encapsulan características comunes, lo que permite un diseño más flexible y fácil de adaptar a largo plazo.

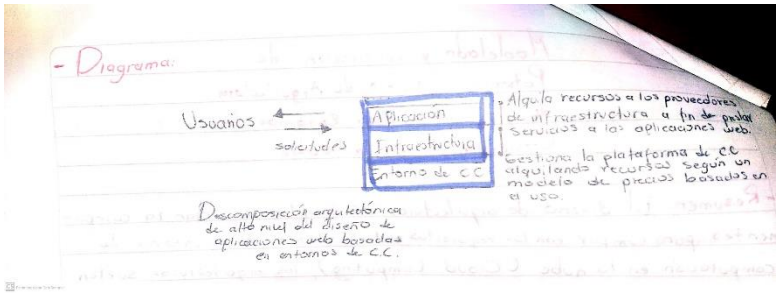
Los patrones arquitectónicos ayudan a resolver problemas comunes y a organizar los componentes de manera eficiente. En el contexto de la nube, se adaptan patrones tradicionales para guiar la distribución de componentes en la infraestructura. Estos patrones definen cómo organizar la arquitectura para satisfacer los requisitos específicos del software en entornos de nube.

Reflexión:

La arquitectura de software en la nube no solo representa la estructura técnica de un sistema, sino que también implica un enfoque estratégico para satisfacer necesidades de rendimiento, escalabilidad y flexibilidad. La nube brinda la oportunidad de desplegar aplicaciones sobre infraestructuras externas, reduciendo costos y optimizando recursos, pero también exige a los arquitectos considerar aspectos complejos, como la compatibilidad entre infraestructura y software.

Es clave que los arquitectos de software no solo sigan patrones conocidos, sino que tengan la flexibilidad para adaptarlos y resolver los desafíos específicos que puedan surgir. Al trabajar con arquitecturas genéricas, pueden construir bases sólidas que se ajusten a distintas soluciones, promoviendo diseños escalables y sostenibles. Este enfoque permite a los desarrolladores responder mejor a las necesidades de los usuarios en un entorno tecnológico en constante evolución.

Diagrama:



Bibliografía:

1) Albin, S. T. (2003). The art of software architecture: design methods and techniques. Hoboken: John Wiley & Sons. [Links]

2) Atkinson, C., & Kuhne, T. (2003). Model-driven development: a metamodeling foundation. IEEE software, 20(5), 36-41. [Links]

3) Barbacci, M., Klein, M., Longstaff, T., & Weinstock, C. (1995). Quality Attributes (CMU/SEI-95-TR-021). Pittsburgh: SEI, Carnegie Mellon University. [Links]

4) Bass, L., Clements, P., & Kazman, R. (2003). Software architecture in practice. Boston, MA: Addison-Wesley Professional.

5) Blas, M. J., Gonnet, S., & Leone, H (2015). Un Modelo para la Representación de Arquitecturas Cloud basadas en Capas por medio de la Utilización de Patrones de Diseño: Especificación de la Capa de Aplicación. In Proceedings of 3º Congreso Nacional de Informática / Sistemas de Información (CONAIISI), Buenos Aires. [Links]

Articulo N° 2: Desarrollo de una herramienta para el aprendizaje de patrones de diseño software

Resumen:

Este proyecto desarrolla una aplicación web educativa para facilitar el aprendizaje práctico de patrones de diseño orientados a objetos. La herramienta permite a profesores y formadores configurar actividades que los estudiantes pueden usar en la plataforma para identificar, aplicar e implementar patrones de diseño. Inicialmente, la herramienta soportará algunos patrones clave del libro de Gamma, aunque su estructura será extensible para añadir patrones nuevos en futuras versiones.

La aplicación sigue una arquitectura de tres capas:

1. **Capa de Presentación**: Proporciona la interfaz de usuario y se encarga de gestionar las peticiones y mostrar la información de manera visual y comprensible. Traduce las solicitudes en formato HTML y los datos de usuario en mensajes que la aplicación entiende.
2. **Capa de Negocio**: Realiza las operaciones lógicas principales, consulta datos cuando es necesario, y se comunica con la capa de presentación para devolver resultados. Aquí se aplican las reglas de negocio y lógica para responder a las interacciones de usuario y gestionar el flujo de datos entre capas.
3. **Capa de Datos**: Administra el almacenamiento y gestión de datos en la base de datos, permitiendo crear, actualizar y eliminar información de patrones y cuestionarios que los estudiantes utilizarán.

Para asegurar la calidad y flexibilidad del código, se incluirán pruebas unitarias que validen el funcionamiento, especialmente importante dado que será una herramienta de código abierto.

Reflexión:

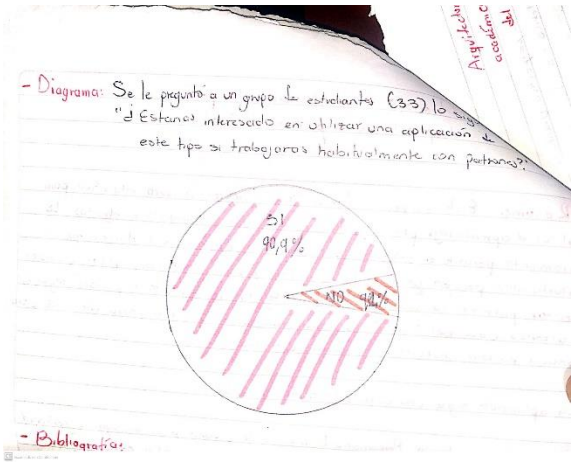
Este proyecto aborda un enfoque innovador en la enseñanza de patrones de diseño, ofreciendo una plataforma que puede cambiar la forma en que estudiantes y profesionales aprenden y aplican estos conceptos fundamentales en el desarrollo de software. La estructura de la aplicación, basada en una arquitectura de tres capas, permite una clara separación de responsabilidades, lo cual mejora la escalabilidad y facilita la futura ampliación de contenidos. Este aspecto es especialmente importante en un entorno en constante evolución, donde los patrones de diseño siguen siendo una herramienta esencial para abordar problemas de diseño complejos en programación orientada a objetos.

Por otro lado, el proyecto enfrenta desafíos propios de una iniciativa pionera. Al no contar con muchas herramientas de referencia, se requiere un análisis exhaustivo para prever problemas y establecer una base robusta. Además, la naturaleza extensible y de código abierto de la herramienta impulsa un trabajo de calidad en términos de pruebas y flexibilidad, permitiendo que desarrolladores de distintos niveles puedan aportar y expandir la funcionalidad de la plataforma sin comprometer la integridad del sistema.

En conclusión, esta herramienta no solo contribuirá al aprendizaje de patrones de diseño, sino que también fomentará la colaboración en la comunidad de desarrolladores y abrirá puertas a la mejora continua. A largo plazo, representa una oportunidad valiosa para evolucionar el conocimiento práctico en ingeniería de software y crear una cultura de aprendizaje accesible y adaptable.

Diagrama:

"¿Estarías interesado en utilizar una aplicación de este tipo si trabajaras habitualmente con patrones?"



Bibliografía:

1) J. Thompson, «Spring Framework 5: Beginner to Guru,» [En línea]. Available: https://www.udemy.com/course/spring-framework-5-beginner-to-guru/.

2) J. Thompson, «Testing Spring Boot: Beginner to Guru,» [En línea]. Available: https://www.udemy.com/course/testing-spring-boot-beginner-to-guru/.

3) Emprenderalia, «¿Qué es el Producto Mínimo Viable (MVP)?,» [En línea]. Available: https://www.emprenderalia.com/que-es-el-mvp-producto-viableminimo/.

4) IBM, «Test-driven development,» IBM, [En línea]. Available: https://www.ibm.com/garage/method/practices/code/practice_test_driven_develo pment/. [Último acceso: 2021].

5) E. Gamma, R. Helm, J. Ralph y V. John, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1994.

Articulo N° 3: Implementación de un framework para el desarrollo de aplicaciones web utilizando patrones de diseño y arquitectura MVC/REST

Resumen:

Este documento describe un proyecto cuyo objetivo es reemplazar un framework obsoleto para mejorar la gestión y distribución de contenidos audiovisuales en una empresa. Inicialmente, se introduce el lenguaje HTML como herramienta principal para la creación de documentos electrónicos en la web. Con el avance de la tecnología y la introducción de XML, se plantea la necesidad de crear sistemas más flexibles que puedan generar múltiples formatos de salida (como XHTML, RSS y otros), adaptándose a las diferentes plataformas y dispositivos.

El planteamiento del problema expone que el framework actual presenta dificultades debido a su alto acoplamiento modular y la necesidad de especialistas para su mantenimiento. Esto limita la capacidad de la empresa para ofrecer contenido en formatos modernos y en tecnologías como Microsoft Silverlight. El proyecto evalúa la viabilidad de modificar el framework existente, optar por frameworks de código abierto o crear uno nuevo que permita una separación clara entre la lógica de negocio, el acceso a datos y la capa de presentación.

El concepto de “patrón” es abordado como una solución a problemas recurrentes en ingeniería de software. Inspirado en la arquitectura, un patrón de software permite resolver problemas comunes de una forma repetible y adaptable. Estos patrones ayudan a formalizar soluciones efectivas para problemas técnicos complejos.

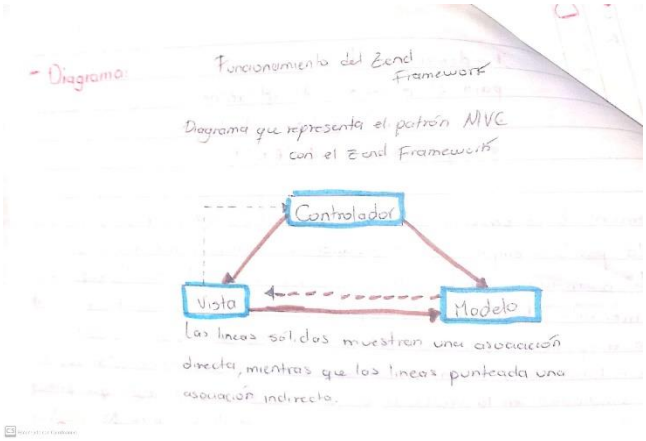
Finalmente, el proyecto selecciona el Zend Framework para implementar una arquitectura de capas, permitiendo a la empresa innovar en su oferta digital y afrontar nuevos desafíos sin comprometer la estabilidad de su sistema central. El nuevo framework permite una curva de aprendizaje rápida para desarrollar nuevos módulos y se ha probado en productos reales en línea, lo que demuestra su eficacia y potencial de crecimiento, apoyando a la empresa en su expansión y adaptación al mercado

Reflexión:

La renovación de un framework para adaptarse a los constantes cambios tecnológicos refleja una visión estratégica y proactiva, que resulta esencial en el ámbito de la tecnología actual. En este caso, la empresa se enfrenta al desafío de mejorar su estructura interna para responder a las demandas del mercado y asegurar una experiencia óptima para sus clientes. La implementación de un nuevo framework que separa la lógica de negocio, el acceso a datos y la presentación, no solo resuelve problemas de acoplamiento y mantenimiento, sino que sienta las bases para un crecimiento sostenible y una adaptabilidad continua.

Este enfoque subraya la importancia de los patrones de software como herramientas que ofrecen soluciones efectivas y probadas, permitiendo enfrentar problemas comunes de manera estructurada y eficiente. Al adoptar una infraestructura flexible y escalable, la empresa logra una ventaja competitiva, ya que puede integrar nuevas tecnologías sin comprometer la solidez de su sistema central.

Diagrama:



Bibliografía:

1) Patrones de Diseño Software. En: LIBRO: GOF Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides.

2) Patrón MVC en Sun [en línea] En http://java.sun.com/blueprints/patterns/MVC-detailed.html [consulta: 15 de Noviembre 2008]

3) Patron MVC en Cunningham & Cunningham, Inc. [en línea] En: http://c2.com/cgi/wiki y http://c2.com/cgi/wiki?ModelViewControllerAsAnAggregateDesignPattern [consulta: 20 de Noviembre 2008]

4) Wikipedia, definición Patrón MVC, [En línea] En: http://en.wikipedia.org/wiki/Model-view-controller [consulta: 20 de Noviembre 2008]

5) Wikipedia, definición Patrón REST, [En línea] En: http://en.wikipedia.org/wiki/REST [consulta: 3 de Diciembre 2008]

Articulo N° 4: Desarrollo de una arquitectura de software para el robot móvil Lázaro

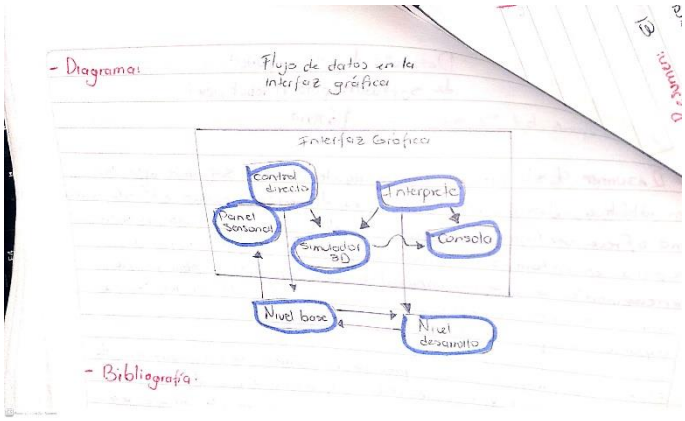
Resumen:

Este artículo explora arquitecturas de software aplicadas en robótica, destacando los enfoques deliberativo, reactivo e híbrido. Cada uno ofrece ventajas: la arquitectura deliberativa permite una actuación rápida en entornos conocidos, mientras que la reactiva responde eficazmente a cambios imprevistos. Las arquitecturas híbridas, que combinan ambos enfoques, han demostrado ser útiles en robots que requieren adaptabilidad. El estudio presenta un caso práctico: el desarrollo de una arquitectura de tres niveles para el robot móvil Lázaro, diseñado para exploración en terrenos irregulares. Esta arquitectura, implementada en C#, incluye una biblioteca para el control de actuadores, monitoreo de sensores y una interfaz de usuario que facilita la programación de instrucciones y el uso de simuladores 3D. A futuro, se propone ampliar los comportamientos autónomos del robot y mejorar su localización mediante sensores avanzados.

Reflexión:

Este artículo resalta la importancia de desarrollar arquitecturas de software flexibles y adaptables para robots, permitiéndoles actuar en entornos cambiantes y enfrentar situaciones imprevistas. Los distintos enfoques arquitectónicos—deliberativo, reactivo e híbrido—buscan equilibrar la planificación de tareas con la autonomía ante imprevistos. El caso de Lázaro muestra cómo una estructura escalable permite aprovechar los componentes del robot y facilita la integración de nuevos comportamientos. Este desarrollo impulsa la robótica hacia sistemas más inteligentes y alineados con las necesidades humanas.

Diagrama:



Bibliografía:

1) L. Bass, P. Clements y R. Kazman. "Software Architecture in practice". 3rd ed. Addison-Wesley Professional. 2013. [Links]

2) D. Nakhaeinia, S.H. Tang, S.B. Mohd Noor and O. Motlagh. "A review of control architectures for autonomous navigation of mobile robots". International Journal of Physical Sciences. Vol. 6 N° 2, pp. 169-174. 2011. [Links]

3) A. Ollero, A. Mandow, V. Muñoz and J. Gómez de Gabriel. "Control architecture for mobile robot operation and navigation". Robotics & Computer-Integrated Manufacturing. Vol. 11 N° 4, pp. 259-269. 1994. [Links]

4) R.C. Arkin. "Behavior-based Robot Navigation in Extended Domains". Journal of Adaptive Behavior. Vol. 1 N° 2, pp. 201-225. 1992. [Links]

5) M. Hank and M. Haddad. "Hybrid control architecture for mobile robots navigation in partially known environments". 11th International Conference on Informatics in Control, Automation and Robotics. Vienna, pp. 513-521. 2014. [Links]

Articulo N° 5: Arquitectura de software, esquemas y servicios

Resumen:

El artículo analiza la evolución de la arquitectura de software en respuesta a las crecientes necesidades de las aplicaciones empresariales, como la independencia de plataformas y la interoperabilidad en entornos distribuidos. Se destaca la transición de sistemas monolíticos a arquitecturas orientadas a servicios (SOA), que permiten la creación de aplicaciones modulares y escalables con bajo acoplamiento entre componentes. Los "esquemas" son definidos como conjuntos de clases unificadas que proporcionan funcionalidades comunes reutilizables, optimizando el desarrollo y el mantenimiento.

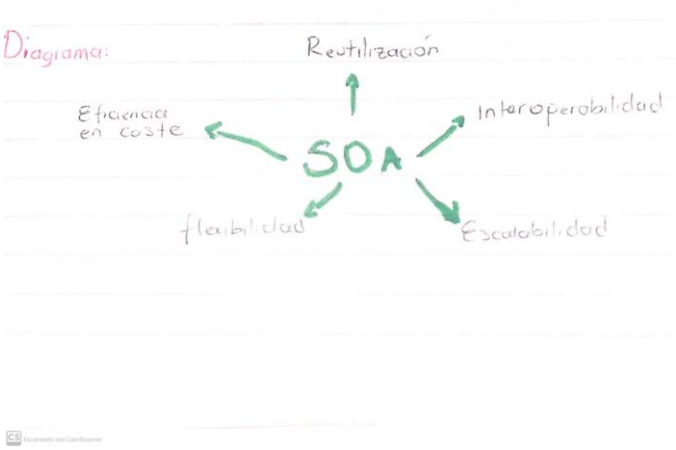
En una arquitectura orientada a servicios, cada funcionalidad se presenta como un servicio autónomo e independiente, comunicándose mediante mensajes, lo cual garantiza flexibilidad, escalabilidad y tolerancia a fallos. Sin embargo, SOA implica retos técnicos, como el manejo de comunicaciones no confiables y dependencias múltiples, que pueden hacer compleja la administración del sistema. La solución propuesta es centralizar procesos en servicios de negocio dedicados, lo que facilita el control y disminuye la interdependencia entre servicios.

El artículo concluye que el uso de esquemas y la orientación a servicios en la arquitectura de software mejoran la reutilización, la productividad y la adaptabilidad de los sistemas, permitiendo aplicaciones más flexibles y sostenibles en el tiempo.

Reflexión:

La arquitectura de software orientada a servicios representa un avance esencial para responder a las demandas modernas de flexibilidad y escalabilidad en aplicaciones empresariales. Al separar funciones en servicios independientes y reutilizables, se promueve una mayor adaptabilidad, facilitando la integración y el mantenimiento de sistemas complejos. Esta visión nos invita a repensar el desarrollo de software, enfocándonos en modularidad y bajo acoplamiento para crear soluciones que no solo respondan a las necesidades actuales, sino que sean sostenibles y evolutivas frente a futuros desafíos tecnológicos.

Diagrama:



Bibliografía:

1) CUESTA QUINTERO, C. E.: Arquitectura de software dinámica basada en reflexión. ETS, Informática. Valladolid, Universidad de Valladolid, 2002.

2) PARRA, JOSÉ DAVID: Hacia una arquitectura empresarial basada en servicios, MSDN, 2004.
3) JBOSS Inc. 2005. JEMS: The Open Source Platform for SOA. http://www.jboss.com/elqNow/elqRedir.htm?ref=/pdf/JEMSPlatformForSOA.pdf (2005-12-20).

Articulo N° 6: Arquitectura de software para el desarrollo de herramienta Tecnológica de Costos, Presupuestos y Programación de obra

Resumen:

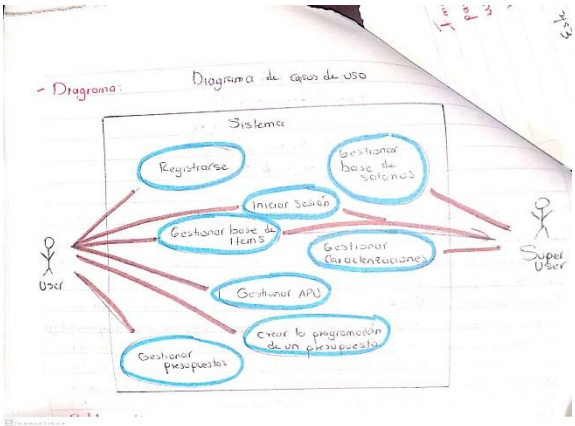
La introducción presenta las Tecnologías de la Información y las Comunicaciones (TIC) como esenciales para el desarrollo humano, la economía y la educación. Las TIC son cada vez más indispensables en la vida diaria y en la formación profesional, lo cual exige que las instituciones de educación superior adapten sus métodos de enseñanza a los avances tecnológicos. Esta adaptación es crucial para formar profesionales competentes que satisfagan las demandas del mercado laboral actual.

En el contexto de la Ingeniería Civil, el artículo resalta la importancia de herramientas tecnológicas para mejorar el aprendizaje, en especial en áreas como la gestión de costos y presupuestos de obra. Debido a que en el mercado existen herramientas comerciales costosas (como Procore o QuickBooks), el objetivo del trabajo es diseñar un software educativo asequible y didáctico que complemente el aprendizaje teórico y práctico en la Universidad Francisco de Paula Santander (UFPS). Este software apoyará el proceso de formación en la materia de “Costos, Presupuestos y Programación de obra” y promoverá el desarrollo de habilidades prácticas y el autoaprendizaje en los estudiantes de Ingeniería Civil.

Reflexión:

Este artículo subraya la importancia de incorporar herramientas tecnológicas en la educación superior, especialmente en Ingeniería Civil, donde las exigencias profesionales requieren competencias prácticas avanzadas. Proponer un software educativo específico para la gestión de costos y presupuestos en la UFPS es clave para mejorar la formación de los estudiantes, ofreciendo un aprendizaje práctico y accesible. Esta iniciativa no solo actualiza métodos de enseñanza, sino que también democratiza el acceso a herramientas esenciales para el desarrollo profesional.

Diagrama:



Bibliografía:

1) Almaguel, A., Alvarez, D., Pernía, L. A., Mota, G. J. y Coello, C.(2016). Software educativo para el trabajo con matrices. Revista Digital: Matemática, Educación e Internet, 16(2), 1-12. https://doi.org/10.18845/rdmei.v16i2.2525

2) Alonso, A. (2019). Software Educativo para la asignatura Introducción a la Ingeniería Civil [trabajo de grado. Facultad de Construcciones Universidad Central "Marta Abreu" de Las Villas, Santa Clara]. https://dspace.uclv.edu.cu/ handle/123456789/11672

3) Alvarado, M. E. y Finol, A. (2019). Diseño de un software para la enseñanza de Inglés Técnico en Ingeniería: enfoque teórico de su elaboración. Revista UCMaule, (57), 39-62. https://doi.org/10.29035/ucmaule.57.39

4) Báez-Pérez, A. A., Soto-Vergel, Á. J. y Herrera-Rubio, J. E. (2019). Enseñanza de sistemas de radiocomunicaciones terrestres con línea de vista mediante software educativo. Revista Educación En Ingeniería, 14(28), 78-87. https:// doi.org/10.26507/rei.v14n28.996

5) Benítez, C. M. y Camargo Pérez, J. F. (2020). Tipificación y análisis de precios unitarios de estructuras para la aducción de agua en vivienda de suelo rural para el Departamento de Cundinamarca [trabajo de grado. Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas, Bogotá]. https://repository.udistrital.edu.co/bitstream/ handle/11349/25575/CamargoPerezJairoFelipe2020. pdf?sequence=1&isAllowed=y

Articulo N° 7: Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte

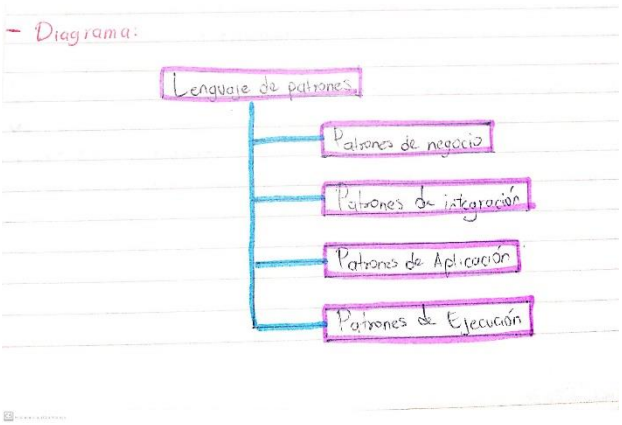
Resumen:

El artículo analiza la evolución de la Arquitectura de Software, destacando los lenguajes de patrones como herramientas clave para mejorar la calidad y mantenibilidad de los sistemas. Desde sus inicios en los años 50 hasta los avances en los 90, la Ingeniería de Software ha buscado optimizar la productividad y calidad del software a través de metodologías como la programación estructurada y la orientación a objetos. Los lenguajes de patrones permiten resolver problemas arquitectónicos de manera más eficiente, y el artículo concluye que es importante profundizar en las metodologías utilizadas para su diseño en futuros trabajos.

Reflexión:

La evolución de la Arquitectura de Software, particularmente a través de los lenguajes de patrones, refleja el crecimiento constante y la adaptación de la ingeniería del software a los desafíos del mundo tecnológico. En un entorno donde los sistemas de información se vuelven cada vez más complejos, la capacidad para diseñar soluciones eficientes y mantenibles es crucial. Los lenguajes de patrones no solo permiten resolver problemas arquitectónicos con mayor eficacia, sino que también brindan una base sólida para futuros avances en el campo. La reflexión aquí radica en que el aprendizaje continuo y la mejora de las metodologías son esenciales para garantizar que, a medida que la tecnología evoluciona, podamos construir sistemas cada vez más robustos y adecuados a las necesidades cambiantes de la sociedad.

Diagrama:



Bibliografía:

1) FAIRBANKS, George: Just Enough Software Architecture: A Risk Driven Approach. Bolder: Marshall & Brainerd, 2010 15 p

2) LÁZARO, María y MARCOS, Esperanza: Research in Software Engineering: Paradigms and Methods. Kybele Research Group, Rey Juan Carlos University, 2005

3) LÁZARO, María y MARCOS, Esperanza, Op. cit., p. 6.

4) ALEXANDER, Christopher: Notes on the Synthesis of Form, (1964).

Articulo N° 8: Implementación de una Arquitectura de Software guiada por el Dominio

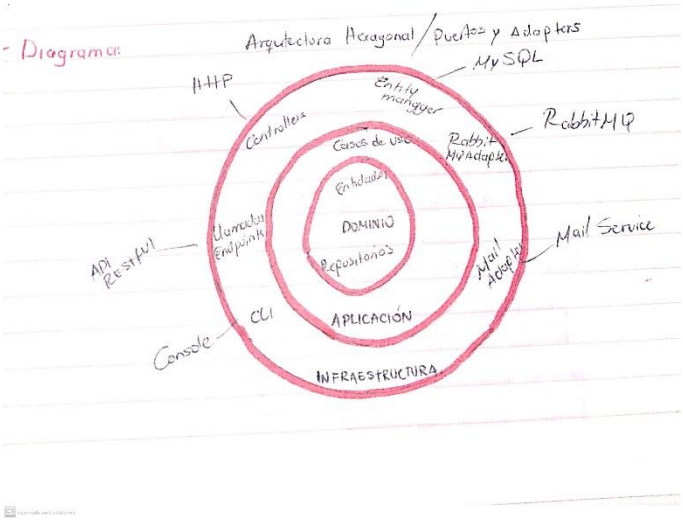
Resumen:

El diseño de software centrado en el dominio (DDD) propone una metodología que organiza el desarrollo de software en torno a un modelo de negocio, utilizando un conjunto de técnicas para gestionar la complejidad. DDD busca representar el negocio mediante "contextos delimitados" y un "lenguaje ubicuo", permitiendo una colaboración eficaz entre expertos en el dominio y desarrolladores. Además, se emplea la Arquitectura Limpia (o Hexagonal), que separa las responsabilidades en capas, manteniendo el núcleo del dominio independiente de tecnologías y otras interfaces. Este enfoque se valida con un caso de estudio que adapta una arquitectura de tres capas a una arquitectura hexagonal, mostrando la viabilidad y beneficios de este diseño centrado en el negocio.

Reflexión:

El Diseño Orientado al Dominio (DDD) nos invita a ver el desarrollo de software no solo como una cuestión técnica, sino como un proceso profundamente ligado al entendimiento del negocio. Al involucrar a expertos del dominio en el diseño del sistema, DDD ayuda a crear soluciones que no solo sean técnicamente viables, sino que también respondan con precisión a las necesidades y desafíos específicos del negocio. Esta colaboración genera un "lenguaje ubicuo" que une a todos los miembros del equipo, desde desarrolladores hasta stakeholders. Además, al aplicar la Arquitectura Limpia, se asegura que el sistema se mantenga flexible y adaptable, lo que permite a las organizaciones responder con agilidad a los cambios en el entorno. En resumen, DDD no es solo una metodología de desarrollo, sino una forma de enfocar el software como una herramienta que evoluciona de acuerdo con el contexto del negocio, lo que mejora la calidad, la comunicación y la efectividad en la entrega de valor.

Diagrama:



Bibliografía:

1) Vivas, L; Cambarieri, M: Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo. CACIC. (2013)

2) InfoQ, “Domain Driven Design and Development In Practice”, disponible en https://www.infoq.com/articles/ddd-in-practice/ [accedido: 01/03/2020].

3) Nair V: “Domain Driven Design. In: Practical Domain-Driven Design in Enterprise Java”. Apress, Berkeley, CA. (2019).

4) Eric Evans. Domain-Driver Design Reference. Definitions and Pattern Summaries. Domain Language, Inc. 2015

5) Martin Fowler, “Bounded Context” https://martinfowler.com/bliki/BoundedContext.html [accedido: 27/04/2020].

Articulo N° 9: Patrones de diseño GOF (The Gang of Four) en el contexto de procesos de Desarrollo de Aplicaciones Orientadas a la web

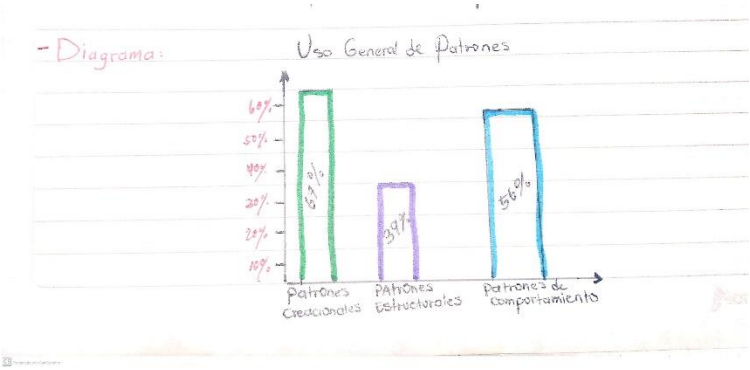
Resumen:

El articulo aborda la identificación y aplicación de patrones de diseño GOF (Gang of four) en procesos de desarrollo de Software, proporcionando una metodología especifica. Comienza estableciendo criterios de selección estrictos para elegir una muestra representativa de procesos de desarrollo en Colombia. Estos criterios incluyen la cantidad de desarrolladores, el uso de UML, la criticidad del sistema, el presupuesto, el modelo de requisitos, y el modelo de Calidad. La selección de los procesos se realiza en colaboración con expertos en desarrollo de software empresarial, asegurando un enfoque basado en estándares como CMMI, ISO 9001 y UML. Para determinar la muestra, se emplea la formula estadística para poblaciones infinitas, alcanzando un tamaño de muestra de 68 procesos.

Reflexión:

El articulo destaca la importancia de los patrones de diseño GOF en la estructuración y optimización de proyectos de desarrollo de software. Al definir un proceso metodológico, riguroso y basado en estándares, subraya como la adopción de estos patrones puede no solo mejorar la eficiencia del desarrollo, sino también incrementar la calidad y sostenibilidad del software producido. La colaboración con expertos y el uso de criterios específicos para seleccionar la muestra refuerzan la idea de que aplicar los patrones de diseño no debe ser una acción arbitraria, sino que debe alinearse con las necesidades y características del proyecto.

Diagrama:



Bibliografía:

1) Braude, E. Ingeniería del Software una Perspectiva Orientada a Objetos, 1a edición, 120-425. RA-MA EDITORIAL. Madrid, España (2003).

2) Cristian L. Vidal, Rodolfo F Schmal, Sabino Rivero y Rodolfo H. Villarroel. Extensión del Diagrama de Secuencias UML (Lenguaje de Modelado Unificado) para el Modelado Orientado a Aspectos. Revista Información Tecnológica. ISSN: 0718-0764. [En línea]. Vol. 23(6), (2012). http://www.scielo.cl/pdf/infotec/v23n6/art07.pdf. Acceso: Octubre 2012.

3) Engineering, S., & Committee, S. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers, 1-31 (1998).

4) Fuensanta, M. D. Marco Metodológico para la Mejora de la Eficiencia de Uso de los Procesos Software. Tesis de Doctorado, Universidad Carlos III de Madrid, Madrid, España (2010).

5) Gamma, E., Helm, R., Johnson, R., & Vlisides, J. Design Patterns Elements of Reusable Object Oriented Software. 1-358. Addison Wesley Longman Inc. USA (1998).

Articulo N° 10: Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el Momento Arquitectónico.

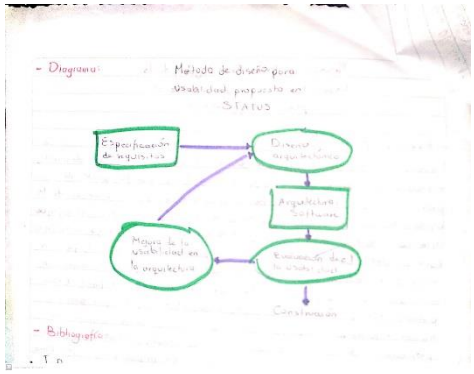
Resumen:

Este artículo presenta una aproximación para mejorar la usabilidad de los sistemas software desde las etapas iniciales del desarrollo, específicamente en el diseño arquitectónico. La propuesta, parte del proyecto Europec IST STATUS, busca anticipar la evaluación y mejora de la usabilidad antes de la construcción del sistema, a diferencia del enfoque tradicional que se centra en mejorar la usabilidad una vez que el sistema está completo. La investigación identifica patrones de usabilidad, que son mecanismos específicos (como “deshacer” o “alertas”) que se incorporan a la arquitectura para mejorar la usabilidad del producto final. Estos patrones son descritos con un enfoque arquitectónico, lo que permite aplicar mejoras en las primeras fases del diseño. Se relacionan con propiedades de usabilidad, como retroalimentación, control explícito del usuario y prevención de errores, que a su vez impactan atributos claves como la eficiencia, confiabilidad y satisfacción. La aproximación propuesta busca integrar estos patrones en la arquitectura para optimizar la usabilidad desde su diseño.

Reflexión:

El artículo refleja la importancia de la accesibilidad y la experiencia del usuario en el desarrollo de software, enfatizando cómo estas prácticas deben ser integradas desde el inicio de un proyecto. La accesibilidad no solo mejora la inclusión de personas con discapacidades, sino que también beneficia a todos los usuarios al hacer las interfaces más intuitivas y fáciles de usar. El diseño inclusivo, además, fomenta la empatía y la responsabilidad social de los desarrolladores, quienes al considerar diferentes contextos y necesidades logran crear productos más valiosos.

Diagrama:



Bibliografía:

1) J Bosch. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach, Pearson Education, Addison-Wesley, 2000.

2) P. Bengtsson, J. Bosh. Analyzing software architecture for modifiability. Journal of Systems and Software, 2000.

3) X. Ferré, N. Juristo, H. Windl, L. Constantine. Usability Basics for Software Developers. IEEE Software, vol 18 (11), p. 22 - 30

4) S. Uchitel, R. Chatley, J. Kramer and J. Magee. LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios. Proceedings of TACAS '03, Warsaw April 2003.

5) E. Folmer, J. Bosch. Usability patterns in Software Architecture. Aceptado en HCII 2003

Articulo N° 11: Arquitectura de software para el sistema de visualización médica Vismedic

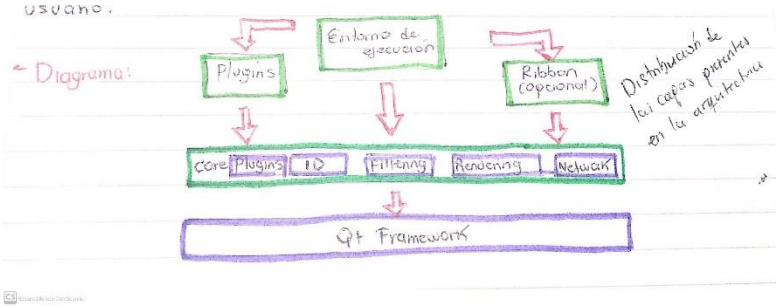
Resumen:

En los últimos años, la arquitectura de software ha tomado un rol fundamental para enfrentar problemas comunes en el desarrollo de software, apoyando la estrategia de negocio en organizaciones tecnológicas. Este trabajo propone una arquitectura mejorada para el sistema de visualización médica Vismedic, integrando los estilos arquitectónicos de componentes, capas y tuberías y filtros, con el objetivo de mitigar problemas de extensibilidad, reusabilidad y dependencias presentes en la ya mencionada arquitectura. Para fundamentar la propuesta, se analizaron conceptos clave de arquitecturas de software y las características arquitectónicas de herramientas en procesamiento y visualización de imágenes, como Voreen, VTK e ITK, además de la especificación OSGI para modularidad basada en componentes

Reflexión:

La propuesta arquitectónica para Vismedic subraya la importancia de un diseño modular y flexible en sistemas médicos, donde la precisión y la eficiencia son esenciales. Al integrar principios de componentes, capas y filtros, el sistema no solo logra una adaptabilidad y escalabilidad que facilitan su evolución, sino que también se vuelve más fácil de mantener y mejorar con el tiempo. Este enfoque modular es crucial en un entorno tan dinámico como el de la tecnología médica, en el que las necesidades cambian rápidamente y es esencial para garantizar la calidad y seguridad en los servicios de salud. Así, esta arquitectura ayuda a crear una base sólida que puede responder a nuevos desafíos y garantizar un mejor servicio al usuario

Diagrama:



Bibliografía:

1) SEI | CARNEGIE MELLON. Community Software Architecture Definitions. [Citado: enero 10, 2013] Disponible en: http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

2) ISO/IEC/IEEE 42010: Defining architecture. [Citado: enero 10, 2013]. Disponible en: http://www.iso-architecture.org/ieee-1471/defining-architecture.html Revista Cubana de Informática Médica 2016:8(1)75-86 http://scielo.sld.cu 86

3) Fielding R.T. Architectural styles and the design of network-based software architectures. Ph.D. dissertation. University of California, California, 2000.

4) Buschmann F. Pattern oriented software architecture: a system of patters. Ashish Raut, 1999. p. 25 - 26.

Articulo N° 12: Revisión sistemática sobre generadores de código fuente y patrones de arquitectura

Resumen:

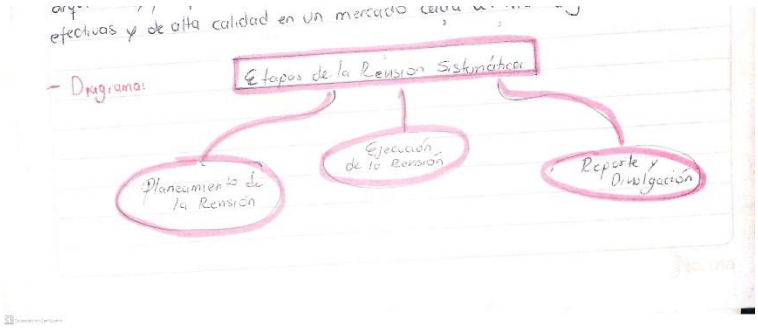
El desarrollo de software enfrenta desafíos que pueden retrasar la entrega o afectar la calidad del producto debido a la falta de organización y deficiencias en la integración de componentes. Los generadores de código fuente (GCF) ayudan a automatizar el código en aplicaciones de lógica repetitiva, lo cual optimiza la productividad y reduce tiempos de codificación. Además, los patrones de arquitectura son esenciales para estructurar y organizar aplicaciones en capas, facilitando la estandarización y el reúso del software.

Mediante una revisión sistemática, se identificaron patrones de arquitectura y herramientas clave en la generación de código para aplicaciones web, permitiendo a los desarrolladores elegir opciones adecuadas a sus ne cesidades. Los resultados confirman que una adecuada elección de herramientas y una buena arquitectura reducen el costo y tiempo de desarrollo, mejoran la estructura y facilitan el mantenimiento, logrando aplicaciones de alta calidad y funcionalidad.

Reflexión:

La automatización mediante generadores de código y la correcta elección de patrones de arquitectura son fundamentales para el desarrollo eficiente de software. Integrar estas herramientas y enfoques permite a los desarrolladores crear aplicaciones mas organizadas, mantenibles y escalables, optimizando tanto el tiempo de desarrollo como la calidad del producto final. Esto destaca la importancia de elegir bien las herramientas y la arquitectura, ya que, con ello, se potencia la capacidad de entregar soluciones efectivas y de alta calidad en un mercado cada vez más exigente.

Diagrama:



Bibliografía:

1) Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995. 13, 94, 155, p. 249.

Articulo N° 13: Una Teoría para el Diseño de Software

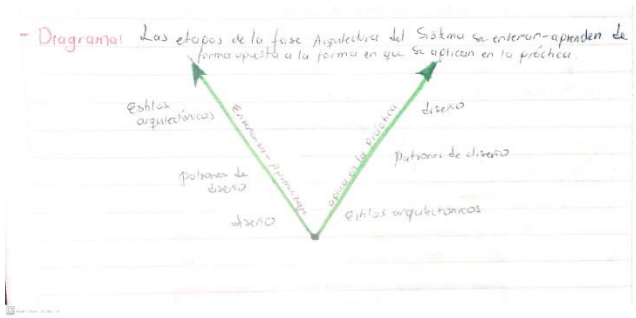
Resumen:

El diseño de software es una fase crucial del desarrollo donde se convierten los requerimientos en uan estructura técnica y funcional. Su objetivo es crear soluciones eficientes, escalables, y fáciles de mantener. Se divide en tres niveles: Diseño arquitectónico, que define la estructura global del sistema; diseño de patrones, que ofrece soluciones reutilizables a problemas comunes y diseño de patrones, que ofrece soluciones reutilizables a problemas comunes; y diseño detallado, que especifica como se implementaran los componentes. Un buen diseño garantiza calidad, rendimiento y seguridad, y facilita el mantenimiento y la expansión del sistema. Emplear herramientas y técnicas adecuadas en esta fase mejora la eficiencia y previene problemas a futuro, asegurando el éxito del proyecto.

Reflexión:

El articulo resalta la importancia del diseño de software como una frase fundamental en el desarrollo de sistemas tecnológicos. Reflexionando sobre ello, podemos concluir que un diseño solido no solo define la estructura de un sistema, sino que también influye en la eficiencia, la escalabilidad y la facilidad de mantenimiento del producto final. La creación de soluciones reutilizables y la anticipación de posibles problemas son clave para evitar complicaciones a largo plazo. Además, invertir tiempo en un diseño bien pensado permite optimizar recursos y asegurar que el software evolucione de manera controlada.

Diagrama:



Bibliografía:

1) B. P. Lientz and E. B. Swanson, Software Maintenance Management. Boston, MA, USA: AddisonWesley Longman Publishing Co., Inc., 1980.

2) F. P. Brooks, Jr., The mythical man-month (anniversary ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

3) J. R. McKee, “Maintenance as a function of design,” in AFIPS ’84: Proceedings of the July 9-12, 1984, national computer conference and exposition. New York, NY, USA: ACM, 1984, pp. 187–193.

4) F. DeRemer and H. H. Kron, “Programming-in-the-large versus programming-in-the-small,” in Proceedings of the 4. Fachtagung der GI Programmiersprachen. London, UK: Springer-Verlag, 1976, pp. 80–89. 13

5) D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” Commun. ACM, vol. 15, no. 12, pp. 1053–1058, 1972. [Online]. Available: https://doi.org/10.1145/361598.361623

Articulo N° 14: Perfiles UML para definición de Patrones de Diseño

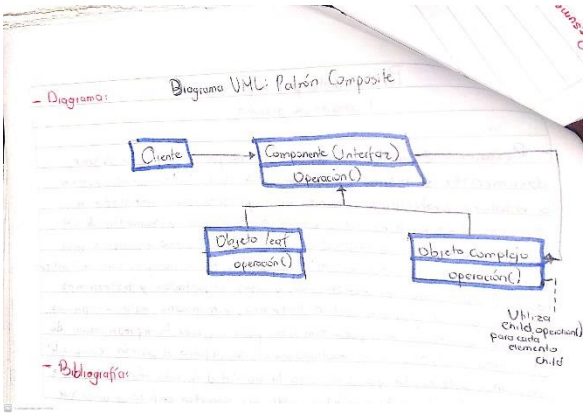
Resumen:

El articulo aborda el uso de perfiles UML para definir, documentar y visualizar patrones de diseño, los cuales ayudan a resolver problemas comunes en la programación orientado a objetos. Los perfiles UML extienden la sintaxis y semántica de UML, permitiendo a los desarrolladores crear un vocabulario especifico para patrones, algo que UML estándar no puede hacer con precisión. El articulo examina como los estereotipos, valores etiquetados y restricciones pueden usarse para modelar patrones, y menciona estudios previos que han utilizado enfoques similares para mejorar la representación de patrones en UML. Como resultado inicial, se definió el patrón “Composite” mediante perfiles, lo que demuestra la viabilidad de esta técnica. A futuro, se buscará evaluar herramientas UML que soporten perfiles y analizar su potencial para una arquitectura general de patrones de diseño.

Reflexión:

El articulo muestra el valor de los perfiles UML para representar patrones de diseño en el desarrollo de software, destacando como estos perfiles amplían la flexibilidad y precisión en la documentación de soluciones reutilizables. Refleja la importancia de los patrones como guías que los desarrolladores pueden seguir para resolver problemas recurrentes, y como los perfiles UML contribuyen a establecer un lenguaje común para ellos, lo que facilita la colaboración y comprensión en los equipos de trabajo. Sin embargo, el articulo también sugiere la necesidad de seguir investigando para integrar estos perfiles en herramientas UML, de manera que se haga mas accesible su aplicación en la práctica. Esta investigación nos lleva a reflexionar sobre la constante evolución de las metodologías en ingeniería de software, ya que siempre se buscan maneras de optimizar la claridad y eficacia en la fase de diseño, adoptando las herramientas existentes a los nuevos desafíos de desarrollo.

Diagrama:



Bibliografía:

1) ISO/IEC, Unified Modeling Language (UML), Version 1.5, International Standard ISO/IEC 19501.

2) E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns. Elements of Reusable ObjectOriented Software, Addison-Wesley. 1995.

3) UML 2.0, Infraestructure Specification, http://www.omg.org/technology/ documents, Agosto del 2004.

4) A. Le Guennec, G. Sunyé, J. Jézéquel, “Precise Modeling of Design Patterns”, In UML 2000, Vol. 1939 LNCS, 2000, pp. 482-496.

5) M. Fontoura, C. Lucena, “Extending UML to Improve the Representation of Design Patterns”, Journal of Object Technology, 2000.

Articulo N° 15: Herramienta para reúso de código JavaScript Orientado a Patrones de interacción

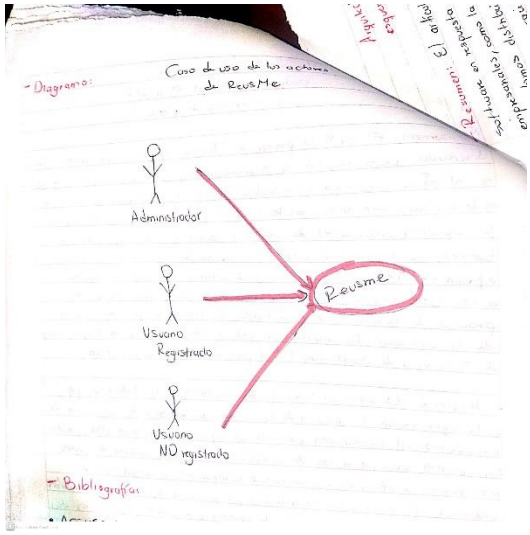
Resumen:

El artículo presenta "ReusMe", una herramienta web diseñada para ayudar a desarrolladores y diseñadores de interfaces al ofrecer código reutilizable en JavaScript para patrones de interacción comunes. Este sistema permite personalizar, imprimir y copiar componentes de interfaz como menús desplegables y botones, facilitando el diseño de aplicaciones web más usables y eficientes. Basada en patrones de interacción, "ReusMe" ayuda a los usuarios a evitar empezar desde cero, optimizando el proceso de desarrollo y permitiendo concentrarse en otras áreas de la aplicación. La investigación utilizó UML y el Proceso Unificado de Desarrollo de Software (PUD) para modelar su arquitectura.

Reflexión:

La reflexión sobre la herramienta "ReusMe" destaca su papel en la optimización del desarrollo de aplicaciones web mediante la reutilización de componentes. Al contar con patrones predefinidos y modelos UML, esta herramienta permite a los desarrolladores construir aplicaciones de manera más rápida y eficiente, aprovechando componentes ya diseñados y probados. Esto no solo ahorra tiempo y recursos, sino que también facilita la creación de interfaces de usuario consistentes, estandarizadas y de alta calidad. Además, "ReusMe" fomenta el trabajo en equipo y la colaboración, ya que los desarrolladores pueden acceder a una biblioteca común de recursos, reduciendo la duplicación de esfuerzos y asegurando que el diseño siga lineamientos claros y coherentes. Así, "ReusMe" no solo apoya el desarrollo ágil, sino que también impulsa la innovación y la mejora continua en el desarrollo web.

Diagrama:



Bibliografía:

1) ACEVES L. 2004. Reutilización de Código. [Documento en Línea]. Disponible: http://www.udem.edu. mx/udem/profesores/laceves/rad/m12arad.pdf. [Consulta: 2006, Noviembre 21].

2) ACOSTA E., ZAMBRANO N. 1999. Interacción Humano Computador: Fundamentos de Diseño. ND 99- 02. Lecturas en Ciencias de la Computación, ISS I316-6239. Escuela de Computación, U.C.V. Caracas, 1999.

3) ALVAREZ M. 2001. ¿Qué es Javascript?. [Documento en Línea]. Disponible: http://www.desarrolloWeb.com/ articulos/25.php. [Consulta: 2006, Noviembre 21].

4) ARAHAL M. 2003. Fundamentos de Informática para Ingeniería Aeronáutica. [Documento en Línea]. Disponible: http://www.esi2.us.es/~jaar/Datos/FIA/ fi 1aer04.pdf. [Consulta: 2006, Noviembre 21].

5) CHRISTOPHER A. 1964. An Introduction for Object-Oriented Designers. [Documento en Línea]. Disponible: http://g.oswego.edu/dl/ca/ca/ca.html. [Consulta: 2006, Septiembre 22].

Articulo N° 16: Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web

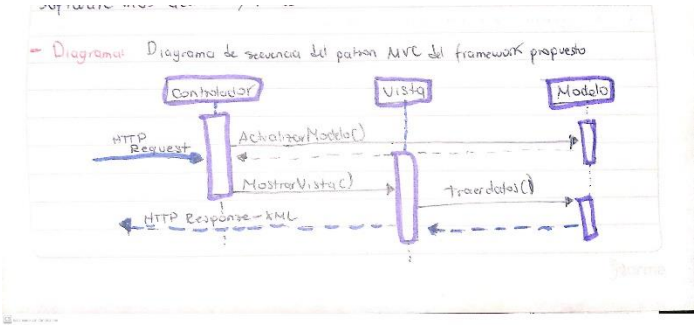
Resumen:

Este trabajo busca crear un framework para el desarrollo rápido de aplicaciones web, enfocado en minimizar costos y tiempos de desarrollo a través de un diseño reutilizable, basado en patrones de diseño y de interfaz. Este framework implementa el patrón MVC y utiliza PHP5 orientado a objetos y tecnologías como AJAX para facilitar la interacción en entornos web. La herramienta permite a los desarrolladores concentrarse en los requisitos funcionales específicos de cada aplicación, mientras reutilizan una estructura probada que favorece la escalabilidad, la usabilidad y el mantenimiento de aplicaciones web. Al compararlo con métodos de desarrollo tradicionales mediante modelos como COCOMO y COCOMO II, se observó una reducción significativa en el tiempo y esfuerzo requeridos. Además, se evaluó su facilidad de uso y aceptación por parte de programadores, obteniendo resultados positivos en usabilidad y eficiencia, lo que sugiere un aporte efectivo al diseño de arquitecturas de software reutilizables.

Reflexión:

El desarrollo de un framework para aplicaciones web facilita la creación rápida de soluciones personalizadas al aprovechar patrones de diseño y arquitecturas predefinidas. Esta práctica optimiza tiempos y recursos, permitiendo a los desarrolladores enfocarse en innovar y resolver problemas específicos, mientras reutilizan componentes que garantizan calidad y consistencia. La reutilización de código no solo hace el proceso más eficiente, sino que también impulsa la creación de software más accesible y robusto.

Diagrama:



Bibliografía:

1) Brooke John, “SUS - A quick and dirty usability scale”, http://www.usabilitynet.org/trump/documents/suschapt.doc, Fecha de acceso: 17/12/2008

2) CHARTE OJEDA FRANCISCO. (2004), “Programación PHP5”, Ed. Anaya Multimedia

3) DARIE CRISTIAN, BRINZAREA BOGDAN, CHERECHES-TOSA FILIP, BUCICA MIHAI, “Building Responsive Web Applications – AJAX and PHP”, March 2006, Ed. Packt Publishing

4) FERRÉ GRAU XAVIER, SÁNCHEZ SEGURA MARÍA ISABEL. “Desarrollo Orientado a Objetos”, ITBA (Instituto Tecnológico de Buenos Aires), Fecha de acceso: 11/07/07

5) GAMMA ERICH, HELM RICHARD, JOHNSON RALPH, VLISSIDES JOHN, "Patrones De Diseño", Ed. Pearson Educacion

Articulo N° 17: Introducción a la Arquitectura de Software

Resumen:

La arquitectura de software nació en los años 70 y 80 como respuesta a la necesidad de estructurar sistemas de forma ordenada antes de su implementación. Parnas impulsó el diseño modular para mejorar la calidad y mantenimiento del software, y en los 90, instituciones como Carnegie-Mellon formalizaron esta disciplina con métodos y patrones que permiten diseñar soluciones adaptables.

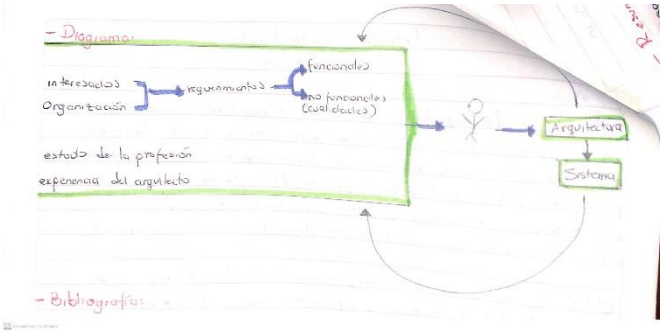
El "Ciclo de Negocio de la Arquitectura de Software" explica cómo los factores técnicos y de negocio influyen en su creación y evolución, y los requisitos no funcionales como la seguridad y el rendimiento son clave para asegurar la sostenibilidad del sistema. Una arquitectura sólida define estándares que optimizan el software actual y guían futuros desarrollos.

Reflexión:

La evolución de la arquitectura de software revela cómo la tecnología y la organización del desarrollo de sistemas avanzan para satisfacer las crecientes demandas de calidad y sostenibilidad. La modularidad y la formalización de patrones en el diseño arquitectónico permiten que el software no solo sea más eficiente y fácil de mantener, sino que también sea capaz de adaptarse y crecer con el tiempo, atendiendo necesidades cambiantes sin perder estabilidad.

Este enfoque no solo resuelve problemas técnicos, sino que también crea puentes entre los objetivos de negocio y las capacidades técnicas, convirtiendo la arquitectura en una disciplina estratégica. Los arquitectos de software, al integrar consideraciones como el rendimiento, la seguridad y la escalabilidad, establecen un entorno en el que cada componente contribuye al éxito del sistema completo.

Diagrama:



Bibliografía:

1) F. DeRemer and H. Cron, “Programming-in-the-large versus programming-in-the-small,” IEEE Transactions on Software Engineering, vol. 2, no. 2, pp. 80–86, 1976.

2) D. L. Parnas, “On the criteria to be used in descomposing systems into modules,” Communications of the ACM, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

3) ——, “Designing software for ease of extension and contraction,” IEEE Transactions on Software Engineering, vol. 5, no. 2, pp. 128–137, Mar. 1979.

4) D. L. Parnas, P. C. Clements, and D. M. Weiss, “The modular structure of complex systems,” IEEE Transactions on Software Engineering, vol. 11, no. 3, pp. 259–266, 1985.

5) B. Liskov and S.N. Zilles, “Programming with abstract data types,” ˜ SIGPLAN Notices, vol. 9, no. 4, pp. 50–60, 1974.

Articulo N° 18: Aplicaciones de patrones de Diseño para garantizar Alta flexibilidad en el Software

Resumen:

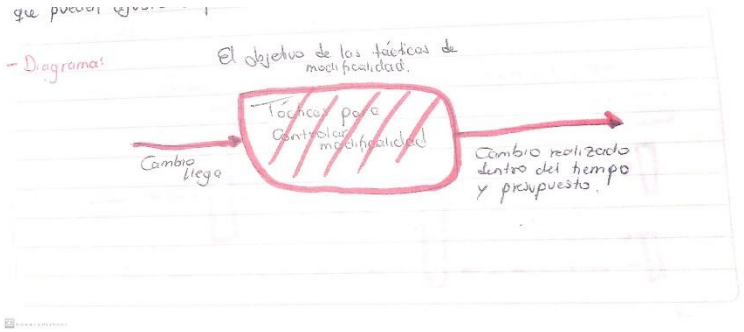
Este artículo analiza cómo el uso de patrones de diseño ayuda a desarrollar aplicaciones empresariales flexibles y adaptables a los cambios en las reglas de negocio. A través de los patrones Estrategia, Compuesto y Fábrica, se propone un modelo que permite agregar o modificar funcionalidades sin necesidad de hacer grandes cambios en la estructura general del software. Estos patrones ayudan a mantener el código modular, facilitando la extensión y modificación de las funcionalidades de la aplicación a medida que las necesidades del negocio evolucionan.

También se destaca el uso de frameworks que soporten la inyección de dependencias y la creación de objetos, ya que estos facilitan la separación de responsabilidades entre los componentes. Esto reduce el acoplamiento y mejora la cohesión, lo cual permite que el sistema sea más fácil de mantener, entender y reutilizar. La aplicación de estos principios y patrones de diseño contribuye a construir sistemas escalables y sostenibles, adaptados a un entorno empresarial dinámico donde los cambios son constantes.

Reflexión:

El artículo enfatiza cómo los patrones de diseño como Estrategia, Compuesto y Fábrica son fundamentales para crear aplicaciones empresariales sostenibles y de fácil mantenimiento. Estos patrones permiten una estructura flexible y modular, lo que facilita la adaptación y extensión del sistema sin generar grandes modificaciones en el código existente. Al combinar estos patrones con prácticas como la inyección de dependencias, se mejora la organización del código, se reduce el acoplamiento y se incrementa la escalabilidad. Esto resulta en aplicaciones más robustas que pueden ajustarse fácilmente a los cambios en los requisitos del negocio.

Diagrama:



Bibliografía:

1) Frank Buschmann, Regine Meunier, Hans Rohnert, PeterSornmerla, MichaelStal."PATTERN-ORIENTEDSOFTWAREARCHITECTURE, Volumen 1: A System of Patterns". John Wiley & Sons, 1996.

2) Len Bass, Paul Clements, Rick Kazman. “Software Architecture in Practice”. Third Edition, Addison Wesley, 2013.

3) ErichGamma."PatronesdeDiseño".Addison Wesley, 1995.

Articulo N° 19: Arquitectura de software académica para la comprensión del desarrollo de software en capas

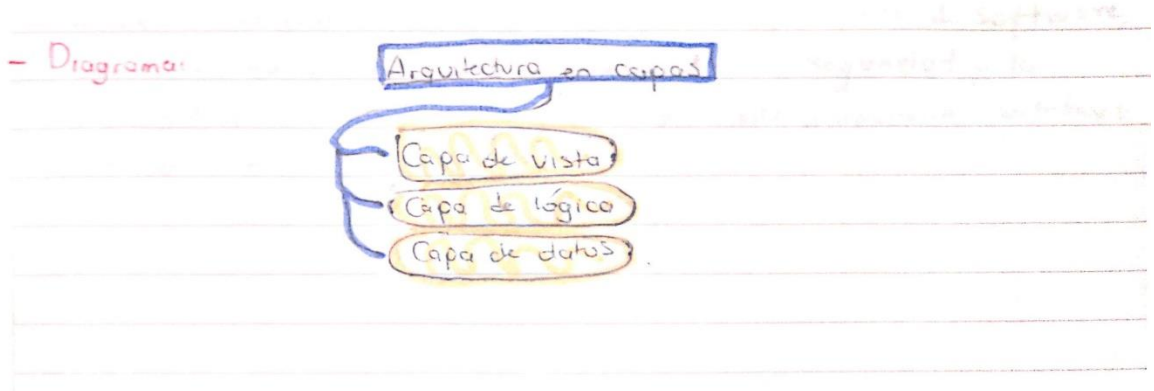
Resumen:

El desarrollo de software requiere considerar aspectos como el acceso a datos, interfaces, procesos funcionales, seguridad, y accesibilidad. Un diseño arquitectónico adecuado es clave para asegurar la flexibilidad, extensibilidad y facilidad de mantenimiento del sistema. Una arquitectura comúnmente utilizada es la arquitectura en capas, que divide el software en capas independientes que interactúan entre sí a través de interfaces. Esto permite que los cambios en una capa afecten mínimamente al resto del sistema. En este trabajo, se presenta un diseño arquitectónico basado en capas, destacando sus beneficios y aplicabilidad para desarrollar sistemas más complejos, con una estructura de siete capas que abordan diferentes aspectos del sistema, desde la interfaz gráfica hasta el acceso a datos.

Reflexión:

La arquitectura en capas es fundamental en el desarrollo de software, ya que permite organizar el sistema en unidades funcionales separadas, lo que facilita la modularización y la separación de responsabilidades. Este enfoque mejora la escalabilidad, el mantenimiento y la evolución del sistema, permitiendo que los cambios en una capa no afecten a las demás. Al dividir el software en capas como la vista, la lógica y los datos, se asegura una gestión eficiente de las interacciones y la persistencia de datos, promoviendo un desarrollo más ágil y flexible. En resumen, la arquitectura en capas optimiza la creación de sistemas robustos y fáciles de mantener.

Diagrama:



Bibliografía:

1) Braude E.J. (2003) Ingeniería de software - Una perspectiva orientada a objetos. Editorial Alfaomega, México.

2) Booch G. Cardacci D. (2013) Orientación a Objetos. Teoría y Práctica. Editorial Pearson Education. Buenos Aires. Argentina.

3) Gamma E. Helm R. Johnson R. Vlissides J. (2003) Patrones de diseño. Editorial Addison Wesley, Peason Education. Madrid.

4) Meyer B. (1999) Construcción de software orientado a objetos. Editorial Prentice Hall. Madrid. España.

5) Newkirk J.W. Vorontsov A.A. (2004) Test-Driven Development in Microsoft.NET. Editorial Microsoft Press. Redmond. Washington. EEUU.

Articulo N° 20: Módulo de recomendación de patrones de diseño para EGPat

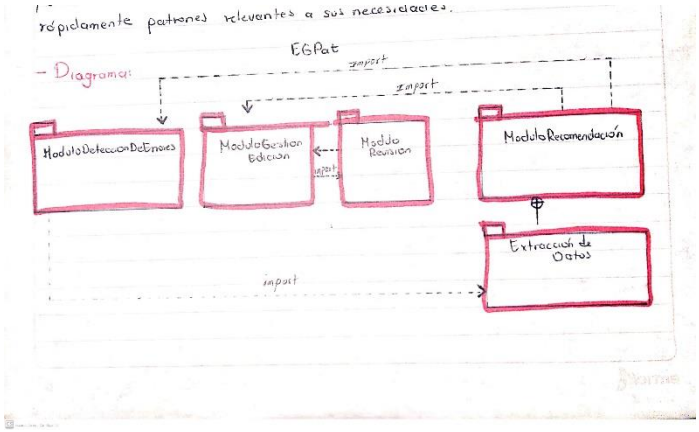
Resumen:

El diseño de recursos educativos digitales es crucial para aprovechar las tecnologías actuales, pero la variedad y complejidad de estos recursos pueden generar dificultades en la selección de patrones de diseño adecuados, especialmente para diseñadores inexpertos. EGPat, un entorno de gestión de patrones de diseño desarrollado por el Grupo de Tecnologías de Apoyo a la Educación (GITAE), busca resolver este problema, proporcionando acceso a patrones de diseño y recomendando los más adecuados para cada situación. Sin embargo, el proceso de selección aún era complicado, lo que llevó al desarrollo de un módulo de recomendación que utiliza técnicas de minería de texto y recuperación de información para sugerir patrones relevantes.

Reflexión:

El desarrollo del módulo de recomendación en EGPat resalta la importancia de aprovechar herramientas tecnológicas avanzadas para facilitar procesos complejos, como la selección de patrones de diseño en la creación de recursos educativos digitales. Este avance no solo optimiza la experiencia de los usuarios al reducir las barreras técnicas, sino que también democratiza el acceso a soluciones efectivas, permitiendo a diseñadores con diferentes niveles de experiencia encontrar rápidamente patrones relevantes a sus necesidades.

Diagrama:



Bibliografía:

1) ACM. (2013). *ACM Recommender Systems community*. PACM RecSys 2013. http://recsys.acm.org/recsys13/%0D [Links]
2) Alexander, C. (1977). *A Pattern Language* (2nd ed.). [Links]

3) Alexander, C., Dawes, M. J., & Ostwald, M. J. (2017). A Pattern Language : analysing , mapping and classifying the critical response. *City, Territory and Architecture*, 1-14. https://doi.org/10.1186/s40410-017-0073-1 [Links]
4) Alonso, G., Jos, F., Investigaci, P. D. E., & Formaci, P. D. E. D. (2020). *Guía metodológica para el éxito en el uso de las tecnologías digitales en educación para la mejora de los aprendizajes a través de los proyectos educativos europeos*. [Links]
5) Arteaga, Y., Terry, Y., & Vazquez, A. (2015). *Sistema de recomendación de patrones de diseño para Recursos Educativos Abiertos*. Universidad de las Ciencias Informáticas Facultad. [Links]

Articulo N° 21: Arquitectura de Software para el Soporte de Comunidades Académicas Virtuales en Ambientes de Televisión Digital Interactiva

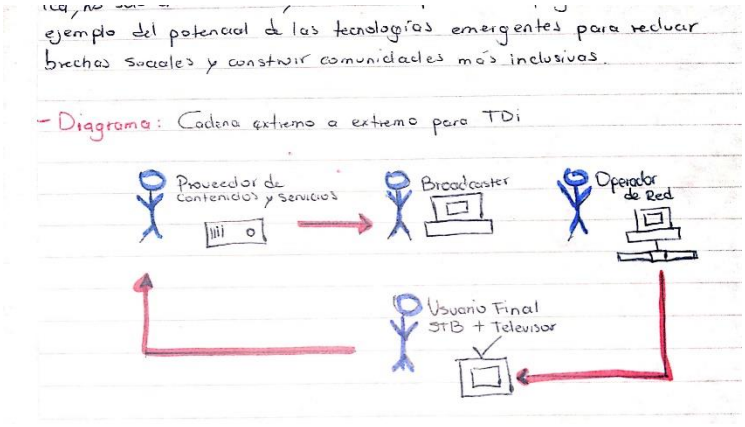
Resumen:

El artículo propone una arquitectura de software para soportar comunidades académicas virtuales (CAV) en el contexto de la televisión digital interactiva (TDi). Esta arquitectura integra aplicaciones de la Web 2.0 mediante un esquema de consumo de servicios REST-JSON, permitiendo la interactividad en ambientes de TDi, y soportando servicios como tablones de mensajes y salas de chat. Además, se presenta un análisis de proyectos similares en T-Learning y TDi, destacando el potencial de la televisión digital como herramienta educativa. La arquitectura es adaptable y extensible, facilitando su aplicación en contextos educativos y otros escenarios (como t-salud y t-gobierno).

Reflexión:

La integración de comunidades académicas virtuales en la televisión digital interactiva muestra cómo la tecnología puede democratizar el aprendizaje, expandiendo el acceso al conocimiento. Esta propuesta resalta la importancia de la interactividad y la interoperabilidad tecnológica, no solo en educación, sino también en salud y gobierno. Es un ejemplo del potencial de las tecnologías emergentes para reducir brechas sociales y construir comunidades más inclusivas.

Diagrama:



Bibliografía:

1) Aarreniemi, P., Modeling and Content Production of Distance Learning concept for Interactive Digital Television. Tesis Doctoral. Universidad de Tecnología de Helsinki, Finlandia. http://lib.tkk.fi/Diss/2006/isbn9512285428/ (2006).
2) Acevedo, C., Arciniegas, J., García X. y Perrinet, J., Proceso de Adaptación de una Aplicación de e-aprendizaje a t-aprendizaje, Revista Información Tecnológica. tecnol. v.21 n.6 La Serena 2010, Información Tecnológica Vol. 21(6), 27-36 (2010).

Articulo N° 22: Identificación y clasificación de patrones de diseño de servicios web para mejorar QoS

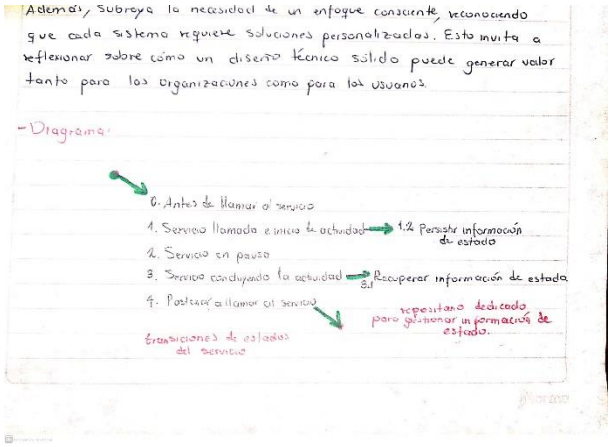
Resumen:

Esta tesis aborda el uso de patrones de diseño en servicios web para optimizar la calidad del servicio (QoS) en arquitecturas basadas en SOA y microservicios. Se propone un inventario de patrones que incluye su descripción, contexto de uso y beneficios asociados. A través de un análisis detallado, se examina cómo estos patrones influyen en aspectos clave como interoperabilidad, funcionalidad y eficiencia del sistema. Además, se evalúan los desafíos que surgen al implementar estos patrones y se presentan recomendaciones para su correcta integración en entornos empresariales. El estudio demuestra cómo la aplicación estratégica de patrones mejora la capacidad de respuesta, escalabilidad y mantenibilidad de los sistemas distribuidos, garantizando así soluciones más robustas y adaptadas a las necesidades actuales.

Reflexión:

Esta tesis resalta la importancia de los patrones de diseño en la creación de sistemas distribuidos eficientes y escalables, como los basados en SOA y microservicios. Aplicar estos patrones de manera estratégica no solo mejora la calidad del servicio, sino que también fomenta soluciones adaptables a las demandas actuales. Además, subraya la necesidad de un enfoque consciente, reconociendo que cada sistema requiere soluciones personalizadas. Esto invita a reflexionar sobre cómo un diseño técnico sólido puede generar valor tanto para las organizaciones como para los usuarios.

Diagrama:



Bibliografía:

1) S. Potts y M. Kopack, “Part 1: Introducing Web Services,” SAMS teach yourself web services in 24 hours. Indianapolis: Sam publishing, 2003, p. 11.

2) E. Marks y M. Werrell, “preface,” Executive's Guide to Web Services. New Jersey: Wiley & Sons, 2003, pp. XI - XII.

3) D. Box, J. Vadoss y K. Horrocks, “Service Orientated Architecture,” Service Orientated Architecture (SOA) in the Real World. Microsoft, 2007, p. 9. [E-book] Disponible: MSDN e-book.

4) T. Erl, “Understanding Layers with Services and Microservices,” Service-Oriented Architecture: Analysis and Design for Services and Microservices 2nd edition. New Jersey: Prentice Hall, 2017, p. 123. [E-book] Disponible: Amazon e-book.

Articulo N° 23: Desarrollo de sistemas de software con patrones de diseño orientado a objetos

Resumen:

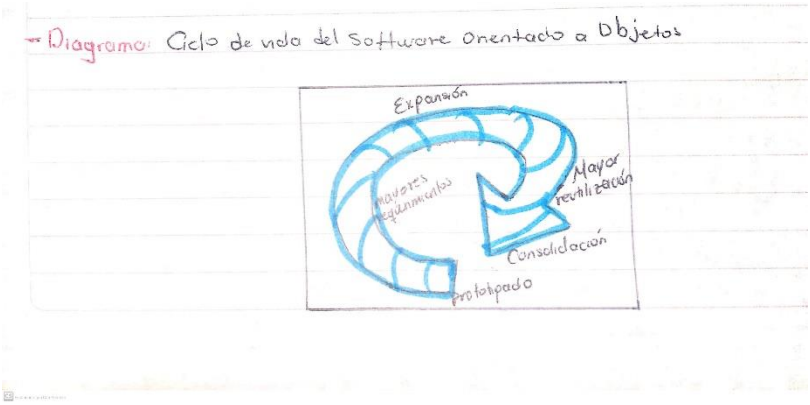
Este trabajo presenta la aplicación de patrones de diseño orientados a objetos para desarrollar una arquitectura de software en el sistema "INTRANET INDUSTRIAL" de la Facultad de Ingeniería Industrial de la UNMSM. Se analiza el impacto de usar patrones en el costo y tiempo de desarrollo en comparación con no utilizarlos. El estudio incluye definiciones teóricas sobre patrones de diseño y su clasificación, junto con la descripción detallada del sistema, sus módulos y herramientas utilizadas, como Rational XDE y Microsoft Visual Studio. Los resultados muestran que el uso de patrones, especialmente el Patrón Informador, genera un ahorro económico del 80% y reduce el tiempo de desarrollo en un 48%. Además, se destaca que la modularidad, reutilización y escalabilidad del sistema mejoran significativamente con la implementación de estos patrones.

Reflexión:

La implementación de patrones de diseño en el desarrollo de sistemas de software no solo representa una mejora técnica, sino también un cambio significativo en la forma en que concebimos y construimos soluciones tecnológicas. Este trabajo resalta cómo los patrones permiten simplificar procesos complejos, promover la reutilización y garantizar la escalabilidad, lo que resulta crucial en proyectos de gran envergadura como la "INTRANET INDUSTRIAL". Además, refleja la importancia de adoptar enfoques metodológicos que no solo optimicen costos y tiempos, sino que también fortalezcan la calidad y sostenibilidad del software a largo plazo.

La reutilización de estructuras lógicas no solo ahorra recursos, sino que establece un lenguaje común entre desarrolladores, facilitando la colaboración y el entendimiento en equipos multidisciplinarios. En un contexto donde los sistemas deben ser cada vez más escalables y flexibles, adoptar patrones arquitectónicos y de diseño no es solo una opción técnica, sino una necesidad estratégica para mantenerse competitivo en el desarrollo de software moderno.

Diagrama:



Bibliografía:

1) [ALE77] CHRISTOPHER ALEXANDER (1977) “A Pattern Language” (Oxford University Press 1977).

2) [ALE77-1] CHRISTOPHER ALEXANDER (1977) “EL Modo Intemporal De Construir” (Oxford University Press 1985)

Articulo N° 24: Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales

Resumen:

La arquitectura de software es clave en el desarrollo de sistemas empresariales eficientes, especialmente para herramientas como ERP y CRM, que optimizan procesos y gestión. Este análisis revisa frameworks arquitectónicos destacados:

1. **Arquitectura en capas:** organiza responsabilidades jerárquicamente, facilita mantenimiento y cambios aislados, pero puede ser redundante y menos eficiente en casos complejos.
2. **Cliente-servidor:** separa roles entre clientes que solicitan datos y servidores que responden, ideal para sistemas distribuidos y multiusuario.
3. **Arquitectura en tres capas:** divide funciones en presentación, lógica y datos, logrando modularidad y escalabilidad.

El uso adecuado de estos enfoques asegura aplicaciones alineadas con objetivos empresariales, mejorando recursos y rendimiento.

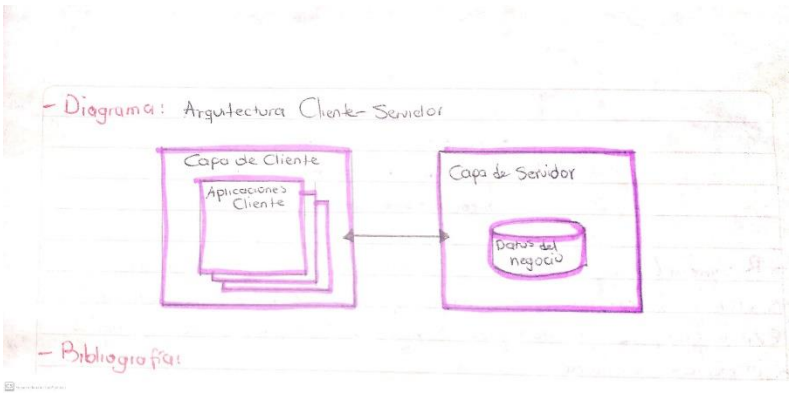
Reflexión:

La arquitectura de software es un componente esencial para el diseño y desarrollo de sistemas empresariales complejos como ERP (Planificación de Recursos Empresariales) y CRM (Gestión de Relaciones con Clientes). Elegir una arquitectura adecuada, como la arquitectura en capas, cliente-servidor o de tres capas, es crucial para garantizar que el sistema sea escalable, modular, y fácil de mantener y actualizar.

Estas arquitecturas permiten organizar el sistema separando responsabilidades, lo que reduce la complejidad, mejora la comunicación entre componentes y optimiza el rendimiento. Por ejemplo, en una arquitectura de tres capas, las funciones de presentación, lógica de negocio y acceso a datos están claramente diferenciadas, lo que facilita tanto el desarrollo como el mantenimiento.

Además, la arquitectura no solo responde a necesidades técnicas, sino también a objetivos empresariales estratégicos. Al diseñar un sistema alineado con los procesos y metas de la empresa, se garantiza que las aplicaciones puedan adaptarse y evolucionar conforme crece el negocio. En resumen, una arquitectura bien diseñada es un puente entre la tecnología y las necesidades empresariales, proporcionando un sistema flexible, robusto y preparado para el futuro.

Diagrama:



Bibliografía:

1) (informática), W. -A. (21 de 06 de 2014). Obtenido de http://es.wikipedia.org/wiki/Arquitectura_en_pipeline_(inform%C3%A1tica)

2) Abraham Sanchez Juárez - Arquitectura por capas, c.-s. T. (s.f.). http://www.joomag.com/. Obtenido de http://www.joomag.com/magazine/evidencia-ii-arquitectura-de-softwarevol1/0343642001389052240?page=5

Articulo N° 25: Atributos de Calidad y Arquitectura del Software

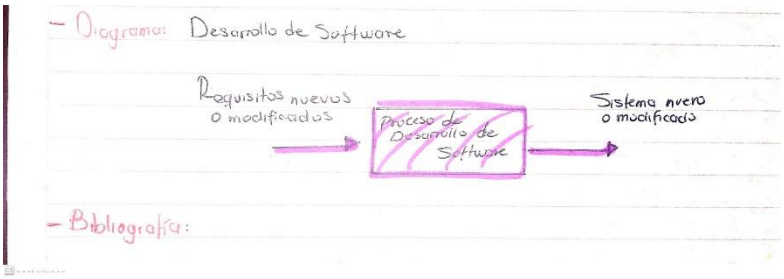
Resumen:

El articulo destaca la importancia de la arquitectura de software y sus atributos de calidad, que incluyen aspectos como la mantenibilidad, seguridad, rendimiento y portabilidad. Estos atributos deben ser equilibrados, ya que mejorar uno puede afectar negativamente a otro. La arquitectura se representa mediante diversas vistas, como módulos, componentes y conectores, que permiten analizar las cualidades estáticas y dinámicas del sistema. Además, se propone una metodología para documentar la arquitectura, que implica identificar los stakeholders, describir los requisitos de calidad y combinar vistas relevantes.

Reflexión:

El articulo resalta la complejidad y la importancia de diseñar una arquitectura de software que no solo cumplía con los requisitos funcionales, sino que también garantice la calidad del sistema en aspectos como la mantenibilidad, seguridad y rendimiento. A medida que se equilibran estos atributos, es fundamental comprender que cualquier cambio en la arquitectura para mejorar un aspecto puede tener repercusiones en otros, lo que hace necesario un enfoque estratégico y bien informado. Además, el articulo enfatiza la necesidad de una documentación detallada de la arquitectura, utilizando vistas que aborden las diversas facetas del sistema, lo cual facilita la toma de decisiones y permite un análisis profundo de los diferentes requisitos de calidad.

Diagrama:



Bibliografía:

1) Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice. SEI Series in Software Engineering. Addison-Wesley, 2 edition, 2003.
2) Frank Buschmann, Regine Meunier, Hans Rohnert, and Peter Sommerlad. Pattern Oriented Software Architecture: A System of Patterns. John Wiley & Son Ltd., August 1996.
3) Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. Documenting Software Architectures. Views and Beyond. SEI Series in Software Engineering. Addison Wesley, 2002.

4) Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. A practical method for documenting software architectures, May 2003. Submitted to the 2003 International Conference on Software Engineering.

Articulo N° 26: Arquitectura de Software en el Proceso de Desarrollo Ágil. Una Perspectiva Basada en Requisitos Significantes para la Arquitectura.

Resumen:

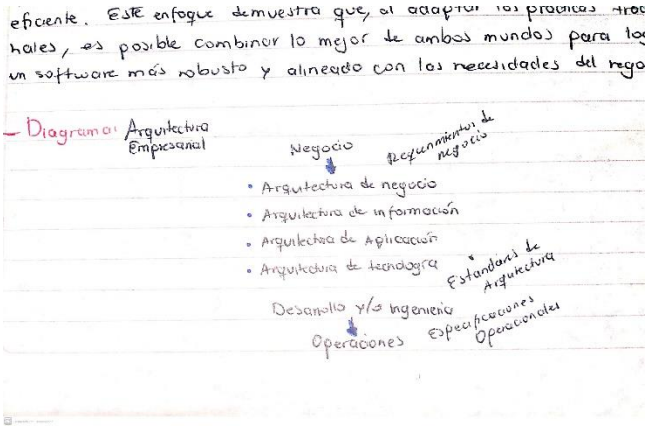
El artículo aborda la integración de la Arquitectura de Software (AS) con las Metodologías Ágiles (MA) a través de la identificación y captura de los *Requisitos Significantes para la Arquitectura* (RSA). Las MA, que se caracterizan por la flexibilidad y la adaptación constante de los requisitos durante el ciclo de vida del proyecto, entran en conflicto con la AS, que requiere una definición temprana y estable de los requisitos arquitectónicos debido a su impacto en el diseño y desarrollo del sistema. Los RSA son aquellos requisitos que tienen un efecto significativo en la arquitectura, y su identificación temprana es esencial para garantizar que la arquitectura cumpla con las necesidades del sistema.

El artículo propone un modelo para la identificación y captura de los RSA dentro de las MA, con el fin de integrar los aspectos arquitectónicos en el proceso de desarrollo ágil. Para ello, se plantean diversos enfoques, como el uso de marcos de trabajo (frameworks), el conocimiento del dominio y los objetivos de negocio, que ayudan a identificar los requisitos que deben ser considerados desde el inicio del proyecto. Al combinar estos enfoques, el artículo busca mejorar la colaboración entre la AS y las MA, permitiendo que ambos enfoques coexistan y se complementen para lograr un desarrollo de software más eficiente y alineado con los objetivos del negocio.

Reflexión:

La reflexión sobre la integración de la Arquitectura de Software (AS) con las Metodologías Ágiles (MA) nos muestra que, aunque en principio parecen incompatibles, pueden complementarse si se enfocan en los *Requisitos Significantes para la Arquitectura* (RSA). Estos requisitos esenciales permiten una arquitectura flexible que se adapta a los cambios sin perder estabilidad. La clave es encontrar un equilibrio entre la flexibilidad de las MA y la estabilidad necesaria en la AS, lo que fomenta una mayor colaboración entre los equipos y un desarrollo más eficiente. Este enfoque demuestra que, al adaptar las prácticas tradicionales, es posible combinar lo mejor de ambos mundos para lograr un software más robusto y alineado con las necesidades del negocio.

Diagrama:



Bibliografía:

1) Schwaber, K., Sutherland, J., “The scrum guide. The Definitive Guide to Scrum”.En: https://www.scrumguides.org/docs/scrum guide/v2017/2017-Scrum-Guide-US.pdf. Accedido el 20/ 02/2017.

2) Cohen, D., Lindvall, M., Costa, P., “An Introduction to Agile Methods” Pdf (2004) En: http://www.cse.chalmers.se/~feldt/course s/agile/cohen_2004_intro_to_agile_metho ds.pdf. Accedido el 20/02/2018

3) Breivold, H.P., Sundmark, D., Wallin, P. and Larsson, S., “What Does Research Say about Agile and Architecture?”, en Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA), USA, (2010).

4) Schwaber, K., Beedle M. “Agile Software Development with Scrum” 1st Edition. Prentice Hall PTR Upper Saddle River, NJ, USA 2001.

Articulo N° 27: Análisis comparativos de patrones de Diseño de Software

Resumen:

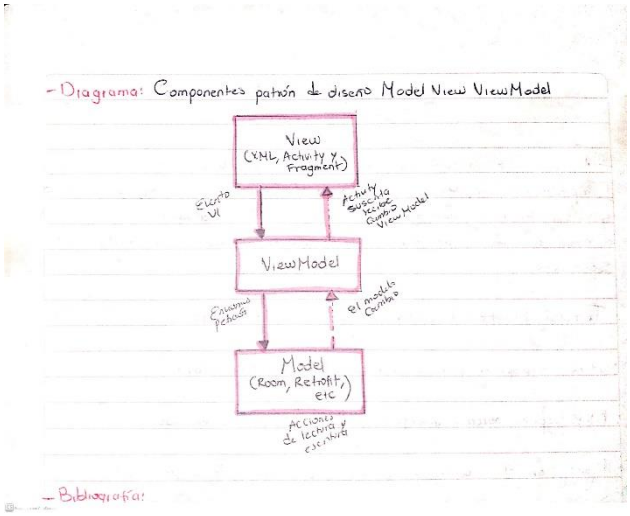
El artículo analiza cinco patrones de diseño de software, Template Method, MVC, MVP, Front Controller y MVVM, destacando sus características, ventajas y desventajas. Los patrones ayudan a organizar, reutilizar y mantener el código, mejorando la calidad del software.

- Template Method facilita la reutilización de código
- MVC es popular por su simplicidad y estructura clara
- MVP separa lógica e interfaz, pero consume mas recursos
- Front Controller centraliza peticiones, pero limita la escalabilidad
- MVVM mejora la mantenibilidad, aunque es complejo sin experiencia previa

Reflexión:

El artículo compara cinco patrones de diseño de software: Template Method, MVC, MVP, Front Controller y MVVM. Estos patrones son pilares fundamentales en el desarrollo de software, ya que proporcionan soluciones probadas para problemas recurrentes, optimizando tiempo y esfuerzo. Su correcta aplicación no solo mejora la calidad y modularidad del código, sino que también facilita el trabajo en equipo al establecer un lenguaje común entre desarrolladores. Sin embargo, es esencial recordar que ningún patrón es superior a otro; cada uno tiene un propósito específico y debe ser seleccionado con base en las necesidades del proyecto y la experiencia del equipo.

Diagrama:



Bibliografía:

1) Pressman, R. Ingeniería del Software: Un enfoque práctico (2010). McGrawHill.

2) E. Gamma, R. Helm, R. Johnson, y J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.

3) A.Shvets, Sumergete en los patrones de diseño. 2019.

4) J. E. McDonough, «Object-Oriented Design with ABAP», Object-Oriented Des. with ABAP, 2017.

Articulo N° 28: Evaluando la arquitectura de software

Resumen:

El artículo presenta una serie de métodos para evaluar la arquitectura de software en diferentes dominios, como sistemas financieros, médicos, o de tiempo real. Entre los métodos destacados están:

1. **ALMA (Architecture Level Modifiability Analysis)**, que analiza la facilidad de modificación en sistemas para estimar costos de mantenimiento, identificar riesgos o comparar arquitecturas según su adaptabilidad.
2. **PASA (Performance Assessment of Software Architecture)**, enfocado en medir el desempeño de la arquitectura mediante análisis de casos de uso críticos, anti-patrones y modelos de rendimiento.
3. **SNA (Survivable Network Analysis)**, desarrollado por el CERT, que evalúa la capacidad de supervivencia de un sistema ante ataques, fallas o accidentes, garantizando la continuidad de servicios esenciales.

Cada método utiliza técnicas basadas en escenarios, análisis detallados y la participación de arquitectos y equipos de desarrollo para optimizar el diseño, identificar problemas y proponer soluciones específicas. El artículo finaliza con una comparación de estos métodos, destacando su aplicabilidad y efectividad en distintos contextos.

Reflexión:

El artículo nos invita a reflexionar sobre la importancia de evaluar la arquitectura de software como un paso crítico en el desarrollo de sistemas robustos, flexibles y sostenibles. Cada método presentado ofrece enfoques especializados que abordan diferentes atributos de calidad, como la facilidad de modificación, el desempeño y la supervivencia ante fallas o ataques. Esto demuestra que no existe una solución universal, sino que cada contexto requiere herramientas y análisis adaptados a sus necesidades específicas.

La reflexión clave radica en cómo estos métodos fomentan una comprensión más profunda del sistema desde su concepción, lo que permite anticipar problemas, optimizar recursos y tomar decisiones informadas. Invertir tiempo en evaluar la arquitectura no solo mejora la calidad del software, sino que también reduce costos y riesgos a largo plazo.

Diagrama:

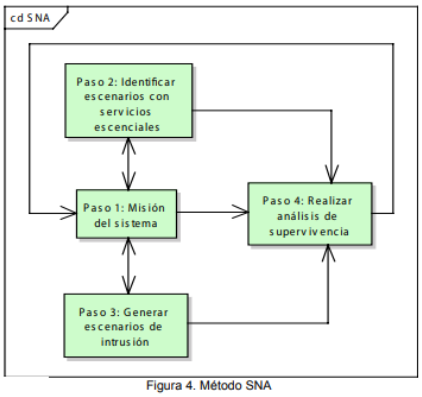


Figura 4. Método SNA

Bibliografía:

1) Bengtsson, PerOlof, Nico Lassing and Jan Bosch, Vliet, Hans van. "Architecture-Level Modifiability Analysis (Alma)." The Journal of Systems and Software 69 (2004): 129-147.

Articulo N° 29: Especificación de la arquitectura de software

Resumen:

El artículo aborda los componentes clave de la arquitectura de software y la importancia de los patrones arquitecturales para organizar sistemas complejos en un contexto de creciente demanda tecnológica. Destaca que una arquitectura efectiva combina métodos documentados, intuición y reutilización de soluciones previas. Se exploran diferentes tipos de patrones: arquitectónicos, que afectan la estructura global del sistema; de diseño, que definen relaciones entre subsistemas; y de lenguaje, específicos para implementar problemas usando características de programación. También se detallan categorías como estructuras organizadas, sistemas distribuidos, sistemas interactivos, y sistemas adaptables, enfatizando su contribución a la eficiencia, usabilidad y adaptabilidad del software. Finalmente, el artículo concluye que los patrones no son recetas mecánicas, sino herramientas que requieren creatividad, habilidades técnicas y experiencia para desarrollar arquitecturas innovadoras y sostenibles..

Reflexión:

La arquitectura de software no es simplemente un conjunto de reglas estáticas o recetas para seguir, sino un proceso dinámico que requiere creatividad, experiencia y un enfoque estratégico. Este artículo nos invita a reflexionar sobre la importancia de los patrones arquitecturales como herramientas que, lejos de simplificar de forma mecánica el diseño de sistemas complejos, potencian nuestra capacidad de resolver problemas. Nos recuerda que cada decisión arquitectónica debe ser consciente, considerando la estructura, los objetivos del sistema y las necesidades futuras de adaptabilidad. La integración efectiva de patrones no solo optimiza los procesos, sino que también impulsa la innovación y la sostenibilidad en un entorno tecnológico en constante evolución.

Diagrama:

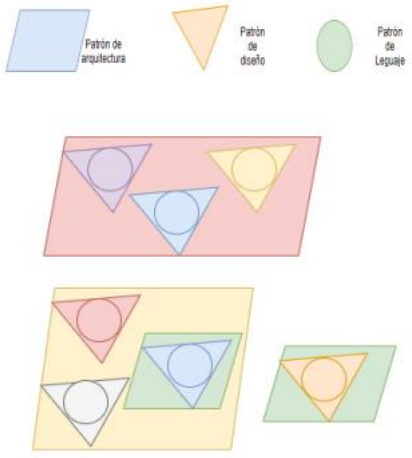


Ilustración 2: Ejemplos de sistemas de software con diferentes patrones.

Bibliografía:

1) E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Articulo N° 30: Evaluación de una Arquitectura de Software

Resumen:

Una arquitectura de software adecuada es esencial para cumplir la Resolución 4505 de 2012 del Ministerio de Salud y Protección Social de Colombia, que regula la validación y reporte de actividades en salud pública. La Secretaría de Salud de Villavicencio (SMSV) enfrenta desafíos con el manejo de grandes volúmenes de datos inconsistentes, afectando la toma de decisiones.

Se evaluaron arquitecturas mediante el método SAAM, priorizando la modificabilidad debido a los constantes cambios normativos. La arquitectura MVC fue seleccionada por su modularidad, bajo costo de actualización y escalabilidad. Esta estructura facilita la integración futura con múltiples dispositivos y permite a la SMSV adaptarse rápidamente a cambios.

El Sprint Cero permitió una evaluación temprana de la arquitectura, mejorando las decisiones de diseño. SAAM también favoreció la comunicación con los interesados y la anticipación de cambios, aunque es susceptible a sesgos en la formulación de escenarios. La implementación de MVC asegura un sistema más robusto y ajustable para cumplir las exigencias normativas

Reflexión:

La elección de una arquitectura adecuada demuestra cómo las decisiones de diseño en software impactan directamente en la capacidad de una organización para cumplir con normativas y manejar grandes volúmenes de datos. En este caso, la implementación de la arquitectura MVC resalta la importancia de priorizar la adaptabilidad y escalabilidad en sistemas que operan en entornos regulados y dinámicos.

El uso de metodologías como SAAM no solo guía la selección de una solución técnica eficiente, sino que también fomenta la colaboración entre los interesados, garantizando que el sistema no solo cumpla con las normativas actuales, sino que esté preparado para futuros cambios. Este enfoque subraya la relevancia de la planificación y evaluación en el desarrollo de software como herramientas clave para garantizar el éxito y sostenibilidad de proyectos en el tiempo.

Diagrama:

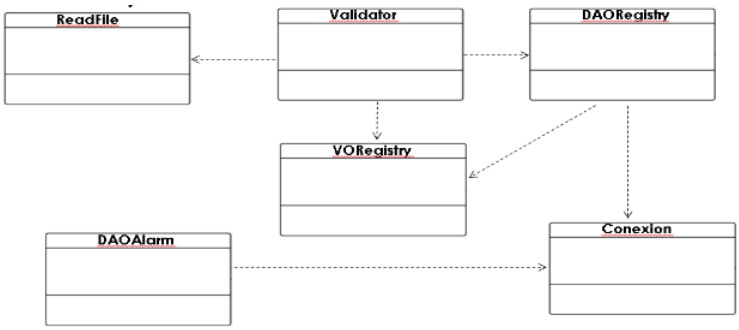


Figura 3. *Estructura básica del Modelo*

Bibliografía:

1) Salud y Protección Social. Ministerio [Internet], [Desconocido] Resolución 4505 de 2012, Colombia, 2012. 2004 [citado 12 septiembre 2019] disponible en https://www.minsalud.gov .co/sites/rid/Lists/BibliotecaDigital/RIDE/DE/DIJ/Resolucion-4505-de-2012.PDF

Articulo N° 31: Introducción a los patrones de diseño

Resumen:

Los patrones de diseño, originados en la arquitectura por Christopher Alexander, fueron adaptados a la programación orientada a objetos en los años 80 por Ward Cunningham y Kent Beck. En los 90, el grupo Gang of Four popularizó 23 patrones en su libro Design Patterns, los cuales son soluciones probadas a problemas recurrentes en el desarrollo de software.

Estos patrones son reutilizables y evitan reinventar soluciones. Su uso muestra la madurez del programador, aunque no deben ser aplicados a todos los problemas. Se clasifican en tres tipos: creacionales (relacionados con la creación de objetos), estructurales (sobre la relación entre clases) y de comportamiento (sobre la asignación de responsabilidades entre objetos).

Reflexión:

Los patrones de diseño son soluciones reutilizables para problemas comunes en el desarrollo de software, basadas en experiencias previas que han demostrado su efectividad. Su implementación permite evitar la necesidad de reinventar soluciones para cada problema, mejorando la calidad, la escalabilidad y el mantenimiento del código. Sin embargo, su uso requiere experiencia, ya que es crucial identificar el contexto adecuado para cada patrón. Aplicarlos de manera incorrecta puede generar más complejidad. Al emplearlos correctamente, se favorece la creación de sistemas robustos, flexibles y fáciles de gestionar, promoviendo mejores prácticas en la programación y una comunicación más clara entre los desarrolladores.

Diagrama:

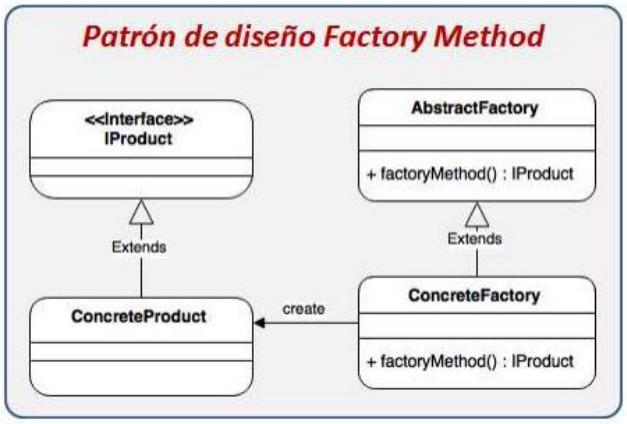


Ilustración 1: Estructura del patrón de diseño Factory Method

Bibliografía:

1) Escobar J [Internet]. [Lugar desconocido] Propuesta de un marco de trabajo de arquitectura para el aseguramiento de la calidad en el diseño de videojuegos serios orientados a la rehabilitación física de acuerdo a la Norma ISO/IEC/IEEE 42010.Escuela Politécnica Nacional, 2019 [Citado 5 feb 2019]. Disponible en: https://bibdigital.epn.edu. ec/bitstream/15000/20119/1/CD%209554.pdf

Articulo N° 32: Utilización de antipatrones y patrones en el análisis de software

Resumen:

El artículo aborda el uso de patrones y antipatrones en el análisis de software. Se destaca que los patrones son soluciones reutilizables a problemas comunes en el desarrollo de software, mientras que los antipatrones son soluciones incorrectas que generan consecuencias negativas. Los antipatrones sirven para identificar fallos y evitar alternativas ineficientes, mientras que los patrones documentan soluciones exitosas. El grupo GIDTSI ha trabajado en la metodología de modelado de procesos de negocio (BPMN) para facilitar la trazabilidad entre los modelos de negocio y los casos de uso del sistema. La investigación también incluye la aplicación de técnicas como “Model Checking” para validar los modelos de software. El uso de patrones de análisis, como los propuestos por Martin Fowler, acelera la definición del modelo abstracto y facilita el paso del análisis al diseño.

Reflexión:

El artículo subraya la importancia de los patrones y antipatrones en el análisis y desarrollo de software, brindando herramientas esenciales para mejorar la calidad del producto final. Mientras que los patrones de análisis proporcionan soluciones probadas que optimizan el proceso de modelado y diseño, los antipatrones alertan sobre errores comunes y malas prácticas que pueden tener consecuencias negativas. Este enfoque no solo ayuda a evitar errores, sino que también promueve una mejor comunicación y comprensión entre los miembros del equipo de desarrollo. La metodología propuesta por el grupo GIDTSI y el uso de técnicas como "Model Checking" refuerzan la necesidad de una validación rigurosa en las etapas tempranas del desarrollo, asegurando que el software final sea más eficiente y confiable.

Diagrama:

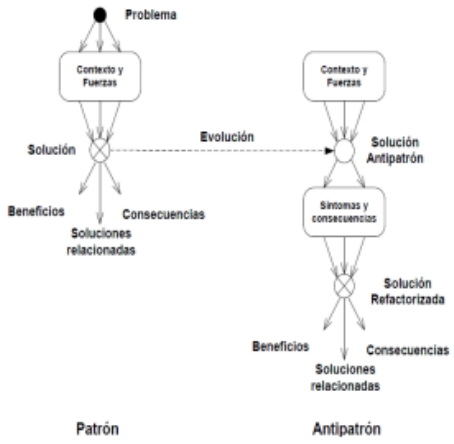


Figura 1. Comparación entre Patrón y Antipatrón

Bibliografía:

1) Marcelo Marciszack, Oscar Medina, Juan Pablo Fernández Taurant, Juan Carlos Moreno y Claudia Castro. “Implementación de Patrones en la Validación de Modelos Conceptuales”. WICC 2015

Articulo N° 33: Arquitectura de software para los Laboratorios Virtuales

Resumen:

El artículo propone una arquitectura de software (AS) para el desarrollo de laboratorios virtuales, con el objetivo de optimizar los tiempos de desarrollo y permitir la reutilización de componentes. Se discuten los conceptos clave de arquitectura y su importancia para la creación de sistemas funcionales y escalables. Se realiza un análisis de diferentes estilos arquitectónicos y patrones de diseño, aplicados al problema de los laboratorios virtuales.

Además, se identifican los requisitos funcionales y no funcionales del sistema, como la usabilidad, la fiabilidad y el soporte, y se detallan las restricciones de diseño, como el uso de C++ y las herramientas de desarrollo como MinGW y G++. Los requisitos de hardware y software también son especificados, y se establece una estrategia de implementación basada en componentes reutilizables, lo que facilita el desarrollo paralelo de múltiples laboratorios virtuales.

Reflexión:

La propuesta de una arquitectura de software para laboratorios virtuales destaca la importancia de una estructura bien definida para agilizar el desarrollo y facilitar la reutilización de componentes. En un mundo donde la eficiencia y la calidad son clave, contar con una AS sólida permite no solo cumplir con los requisitos funcionales, sino también garantizar la usabilidad, fiabilidad y sostenibilidad a largo plazo del sistema. La modularidad y la integración de componentes son esenciales para crear soluciones escalables y flexibles, adaptables a diversas necesidades educativas, lo que refuerza el valor de la arquitectura en el éxito de proyectos tecnológicos. Además, la implementación de estrategias claras de desarrollo y mantenimiento, junto con el uso de tecnologías como C++ y herramientas como OGRE3D, asegura una plataforma robusta y de alto rendimiento, capaz de ofrecer experiencias educativas ricas y dinámicas para los usuarios.

Diagrama:

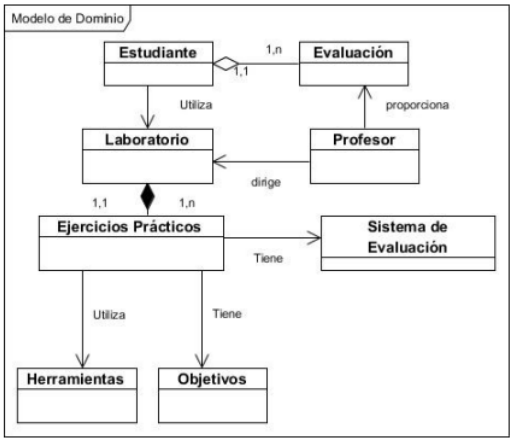


Figura 6. Modelo de dominio.

Bibliografía:

1) A Survey of Architecture Description Languages. Clements, Paul. Alemania : s.n., 1996. IWSSD '96 Proceedings of the 8th International Workshop on Software Specification and Design. p. 16. ISBN: 0-8186-7361-3.

Articulo N° 34: Arquitectura de software para sistema gestión de inventarios

Resumen:

El artículo analiza la importancia de la arquitectura de software en el desarrollo de sistemas modernos, centrándose en la creación de un sistema de gestión de inventarios y almacenes para empresas en Cuba. Señala que, debido a la obsolescencia de los sistemas existentes, es crucial implementar nuevas arquitecturas que sean más eficientes y adaptables a las necesidades actuales del entorno económico y tecnológico. El artículo propone una arquitectura modular, escalable y flexible, que pueda integrarse fácilmente con otros sistemas y que cumpla con los requisitos de confiabilidad, rendimiento y escalabilidad. Además, destaca la necesidad de que los desarrolladores cuenten con una visión clara del sistema a construir, lo que facilita la implementación y la evolución del mismo. También se hace énfasis en la importancia de contar con una base sólida de diseño y una planificación adecuada para garantizar la calidad del software a largo plazo.

Reflexión:

La reflexión destaca la importancia de una arquitectura de software bien diseñada, que no solo optimiza el rendimiento y la escalabilidad, sino que también facilita futuras modificaciones y adaptaciones. En el caso del sistema propuesto para gestionar inventarios en Cuba, se subraya la necesidad de crear soluciones modulares y flexibles que respondan a las condiciones locales y tecnológicas cambiantes. Este enfoque resalta que una planificación adecuada desde el inicio garantiza la durabilidad y relevancia del sistema a largo plazo.

Diagrama:

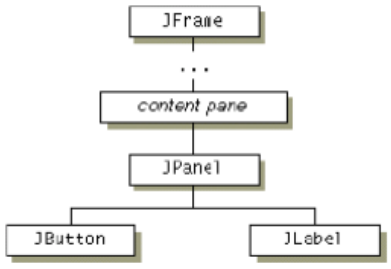


Fig. 12. Ejemplo de Jerarquía de clases de Swing

Bibliografía:

1) Medvidovic, N. y. R., D. S. (1997) Domains of Concern in Software Architectures and Architecture Description Languages. En USENIX Conf. on Domain-Specific Languages, Santa Barbara (Estados Unidos).