

Artículos

Carolina Martínez Cortes

SENA-Servicio Nacional de Aprendizaje

Centro de la Industria la Empresa y Servicio

Análisis y Desarrollo de Software

Ficha: 2694676

Neiva/Huila

ARTÍCULO 01.

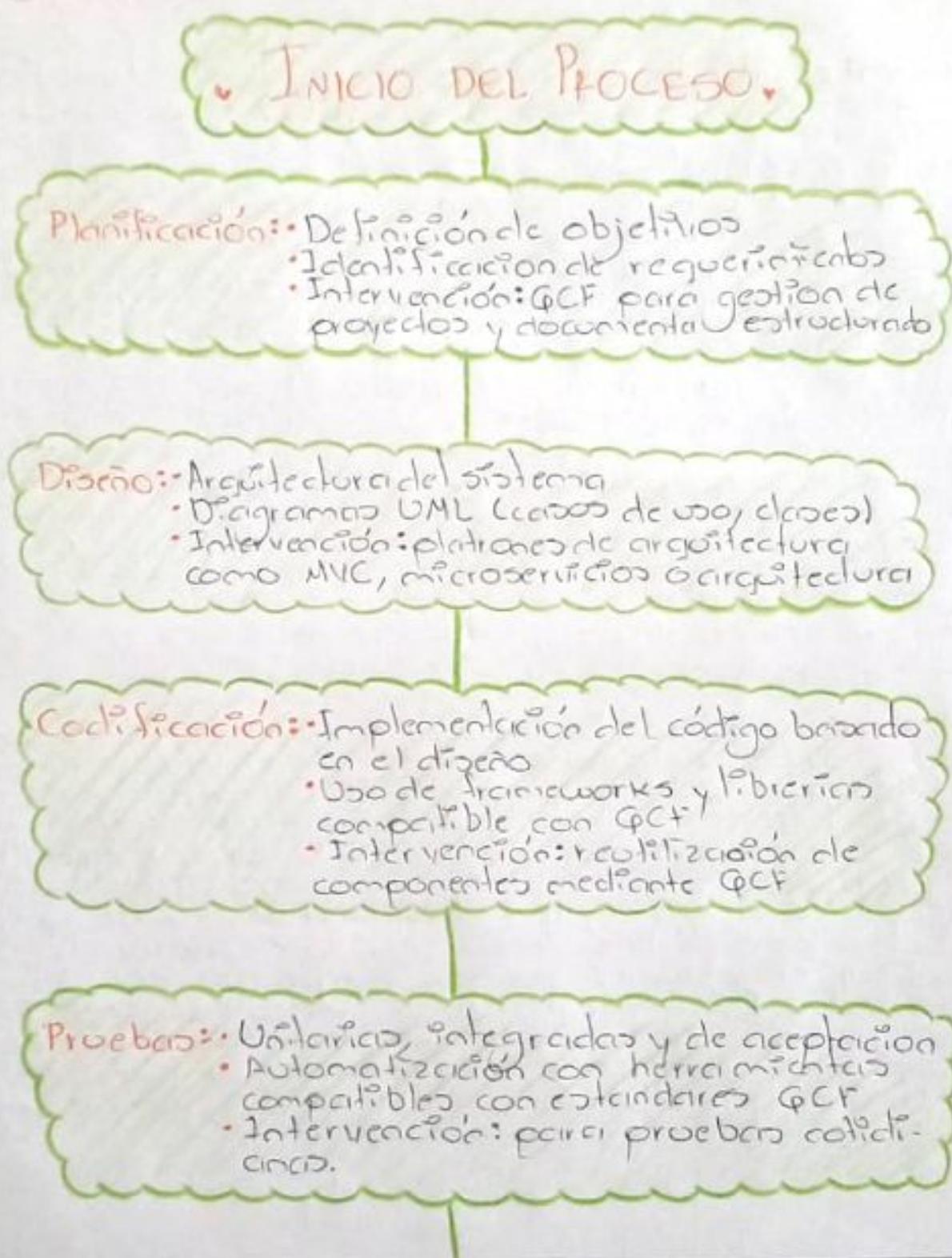
REVISIÓN SISTEMÁTICA SOBRE GENERADOR DE CÓDIGO FUENTE Y PATRONES DE ARQUITECTURA

Resumen: el artículo habla sobre la revisión sistemática sobre Generador de Código Fuente (GCF) y patrones de arquitectura en el desarrollo de software, específicamente orientado a aplicaciones web. Lo cual analiza la utilidad de los GCF como herramientas que automatizan la ejecución de código, permitiendo a los desarrolladores a reducir tiempos, costos y errores. La investigación también analiza la importancia de los patrones de arquitectura, que establecen estructuras para organizar el software de manera coherente, favoreciendo su mantenimiento, escalabilidad y calidad a largo plazo. El análisis que se obtuvo de esta investigación es que se identificaron las herramientas frameworks más empleados en los GCF, así como los lenguajes de programación y bases de datos más comunes en este contexto. La investigación también obtuvo la metodología, la cual abarca la planificación, ejecución, selección y evaluación de estudios previos, con el fin de reunir evidencia de los mejores prácticos y tendencias actuales en el uso de GCF y patrones de arquitectura.

Reflexión: ya que en el mundo de la tecnología se combina y sigue cambiando a toda velocidad, los generadores de código fuente (GCF) y los patrones de arquitectura se han vuelto una herramienta muy esencial e importante para los desarrolladores y arquitectos, ya que no solo aceleran el proceso de desarrollo, sino que también mejoran la calidad de desarrollo y productos. También hace más sencillo mantener y actualizar el software cuando surgen nuevas necesidades. Esta investigación destaca la importante que es mantenerse actualizado con estas herramientas, ya que las decisiones bien informadas pueden tener una gran diferencia en la eficiencia y durabilidad de un proyecto de software.

Bibliografía: Casas, M.R.H. (2020). Revisión sistemática sobre generadores de código fuente y patrones de arquitectura (Magister Thesis, Pontificia Universidad Católica del Perú)

Diagrama:



Mantenimiento:

- Corrección de errores
- Actualización y mejoramiento
- Intervención: que implica resolución de cambios y soporte

ARTÍCULO 02.

UNA TEORÍA PARA EL DISEÑO DE SOFTWARE

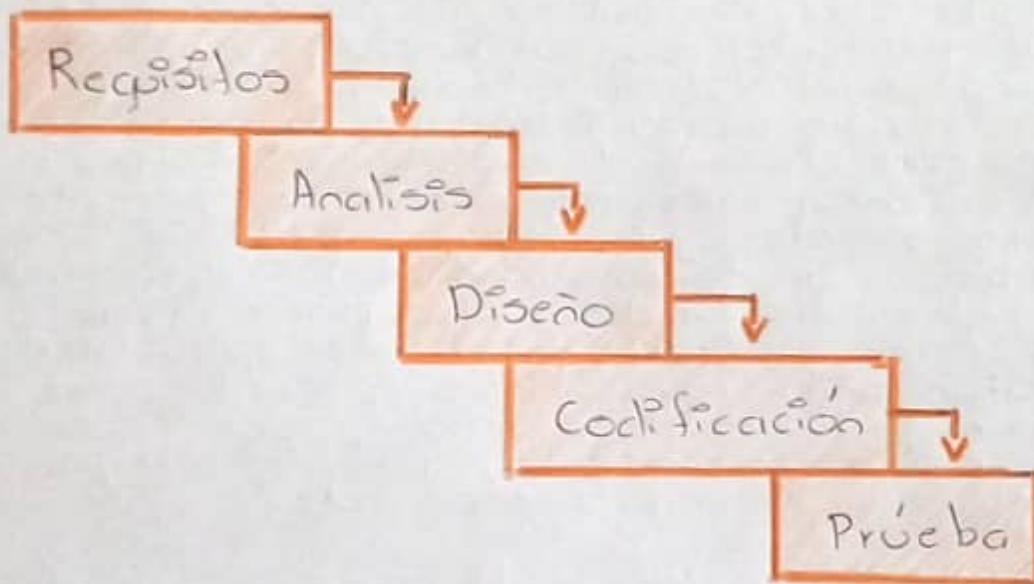
Resumen: este artículo habla sobre la teoría del diseño de software, el cual desempeña un papel muy importante en el ciclo de desarrollo, desde la planificación hasta el mantenimiento. La idea principal es dividir el software en componentes interconectados, con funciones y relaciones claras, lo que hace el sistema más comprensible y fácil de mantener. La teoría introduce el concepto de diseño "Diseño para el cambio", que busca anticipar futuras modificaciones, preparando cada componente para adaptarse sin afectar la integridad del sistema. Además, habla de un "diseño calculado", que mezcla intuición y reglas estructurales claras para crear componentes bien definidos. El diseño se estructura en tres niveles: un estilo arquitectónico general como cliente/servidor, patrones de diseño específicos para resolver problemas comunes, y finalmente, la definición detallada de cada componente.

Reflexión: esta teoría nos hace reflexionar sobre el rol esencial del diseño como un elemento estructurador del proceso de desarrollo de software. A través de la creación de una estructura anticipada y flexible, el diseño permite a los desarrolladores adaptarse a nuevas exigencias sin restructuring completamente el sistema. Evita de ver el diseño como una fase que consume tiempo, el enfoque sugiere que este es un ahorro de costos a largo plazo) ya que los cambios y el mantenimiento se pueden realizar con menores recursos. La teoría también destaca la importancia de establecer reglas de diseño en el diseño para los diferentes estructurales sean consistentes y reproducibles. Un enfoque lógico y calculado en el diseño proporcionando una guía que equilibra la creatividad

con la calendarización. También el uso de estilos arquitectónicos y patrones de diseño genera un lenguaje común entre los desarrolladores, facilitando el trabajo en equipo y reduciendo los problemas de integración y comunicación.

Bibliografía: Cristóbal, M. (2021). Una Teoría para el Diseño de Software

Diagrama:



ARTÍCULO 03.

PERFILES UML PARA DEFINICIÓN DE PATRONES DE DISEÑO

Resumen: el artículo habla sobre cómo los perfiles UML pueden extender el uso de UML (Lenguaje Unificado de Modelado) para mejorar la definición y visibilización de patrones de diseño. Los patrones de diseño son soluciones reutilizables en el desarrollo de software orientado a objetos, y el uso UML ayuda a modelar esas soluciones. Los perfiles UML se propone como una solución para superar estas limitaciones, ya que permiten personalizar la sintaxis y semántica de UML mediante estereotipos, etiquetas y restricciones. Esto no solo permite personalizar el lenguaje UML para dominios específicos, sino también para modelar patrones comunes en diferentes dominios. El uso de perfiles UML tiene múltiples beneficios: evita la necesidad de herramientas específicas para patrones y permite la creación de bibliotecas de patrones adaptadas, lo que facilita la reutilización y standardización de soluciones de diseño.

Reflexión: Los UML para definir patrones de diseño son expresivos y flexibles, lo cual permite adaptarse a las necesidades específicas de cada proyecto y ayuda a resolver problemas comunes en el desarrollo de software. Gracias a los perfiles UML, los desarrolladores pueden representar patrones de diseño complejos sin depender de herramientas externas, lo que hace el proceso más fluido y mejora la coherencia entre lo que se modela y lo finalmente se diseña. No solo ayuda a ahorrar tiempo sino que también facilita el trabajo en equipo ya que permite crear un vocabulario común y específico que mejora la comunicación entre los miembros. Este tipo de solución impulsa una forma de trabajo más dinámica y colaborativo, donde los desarrolladores pueden personalizar UML para realmente se ajuste a las necesidades de sus patrones de diseño.

Bibliografía: Gómez, A. G. Francisco, D. E., & Montalvo, G. A. (2006). Perfiles UML para la definición de patrones de Diseño. In VIII Workshop de Investigadores en Ciencias de la Computación

Diagrama: representa los componentes principales de un perfil UML, incluyendo estereotipos, valores elípticos, restricciones con ejemplos de cómo estos elementos son personalizados para el patrón "Composite".

Profile Composite Pattern

Stereotype Component

Stereotype Composite

Tagged Value: maxChildren: Integer

Constraint: self. children->size() >= 1

Stereotype Leaf

Tagged Value: ismutable: Boolean

Constraint: self. children->isEmpty()

ARTÍCULO 12.

ALQUINTECTURA DE SOFTWARE PARA EL DESARROLLO DE HERRAMIENTA TECNOLÓGICA DE COSTOS, PRESUPUESTOS Y PROGRAMACIÓN DE OBRA

Resumen: el artículo presenta el desarrollo de una arquitectura de software educativo para la materia de "Costos, presupuestos y programación de obra" en la carrera de Ingeniería Civil en la Universidad San Francisco de Paula de Santander. Este software está diseñado para apoyar el aprendizaje autónomo de los estudiantes, permitiéndoles gestionar sus conocimientos sobre los temas fundamentales en la construcción, como el análisis de precios unitarios (APU), la planificación de actividades y la optimización de recursos. La herramienta sigue una estructura lógica y sencilla, una metodología secuencial que abarca desde la creación de una lista de materiales hasta el cálculo de costos y la programación de tareas. Los desarrolladores utilizaron diagramas UML para representar procesos, como el diagrama de secuencia, que ilustra el flujo desde el registro del usuario hasta la creación de presupuestos. Esto, permite a los usuarios o estudiantes ver cómo funcionan las actividades de obra en un entorno simulando, facilitando la comprensión del impacto de cada decisión financiera y logística en un proyecto de construcción.

Reflexión: esta herramienta transforma el enfoque de la educación en ingeniería civil, combinando teoría y práctica de manera interactiva. Un ejemplo de esto es cómo permite gestionar costos directos como materiales y mano de obra e indirectos (gastos generales) simulando decisiones que los estudiantes deberían tomar en el campo profesional. Este tipo de aprendizaje autodirigido no solo facilita la adquisición de habilidades técnicas, sino que también desarrolla competencias críticas como la

ARTÍCULO 04.

PATRONES DE USABILIDAD: MEJORA DE LA USABILIDAD DEL SOFTWARE DESDE EL MOMENTO DE ARQUITECTÓNICO

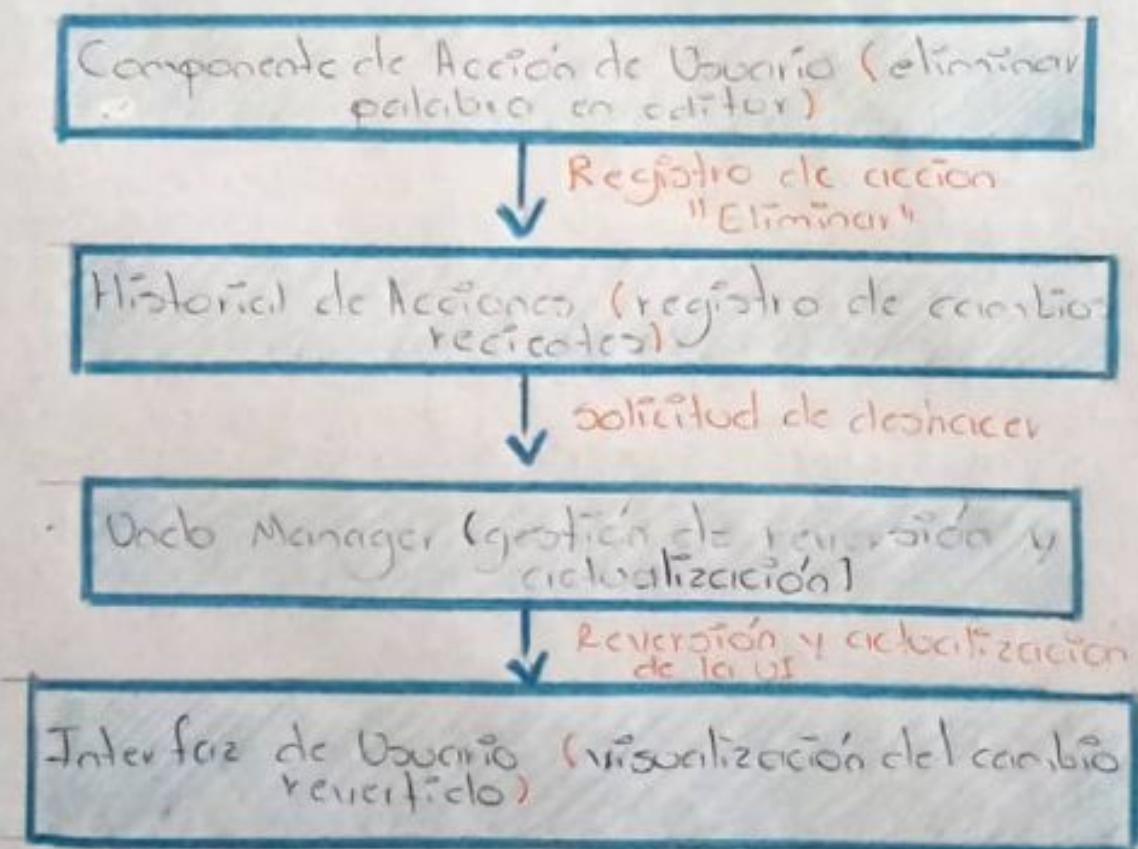
Resumen: el artículo aplica una metodología innovadora para mejorar la usabilidad en el desarrollo de software, introduciendo este criterio desde la fase final/inicial de diseño y no solo al final del proceso. En el marco de proyecto europeo STATUS, se propone incorporar patrones de usabilidad en la arquitectura del sistema, lo que permite optimizar la interacción del usuario desde el inicio. A diferencia de los enfoques tradicionales, que suelen enfocarse en la interfaz gráfica, STATUS sugiere patrones que afectan la estructura interna del software. Estos patrones incluyen funciones claves como "deshacer", "cancelar" y soporte para "múltiples documentos" y se organizan en atributos, propiedades y patrones. Los atributos, como la facilidad de aprendizaje y la satisfacción del usuario, se descomponen de propiedades específicas cuales son implementadas al traves de patrones arquitectónicos. El enfoque implica un proceso inductivo en el que los desarrolladores construyen modelos arquitectónicos iniciales y los modifícan para incluir estos patrones. Posteriormente, estos patrones se validan mediante pruebas en proyectos de redes simulaciones y en escenarios específicos. Este enfoque garantiza que la usabilidad sea un componente estructural del sistema desde el diseño inicial, reduciendo la necesidad de etapas posteriores y promoviendo una arquitectura que optimice la experiencia final del usuario final.

Reflexión: el enfoque de STATUS para mejorar la usabilidad desde la arquitectura del software representa un avance importante respecto a los métodos convencionales. Esta metodología permite anticiparse a problemas de usabilidad que a menudo se identifican solo al final de desarrollo, lo que genera un proceso más eficiente y menos costoso.

Además, recalla la importancia de integrar la usabilidad como un componente estructural, y no únicamente como un elemento de la interfaz gráfica, promoviendo un sistema intuitivo y fácil de usar desde el nacimiento del software. Al trabajar con patrones arquitectónicos, STATUS crean una relación directa entre la estructura del sistema y la experiencia del usuario, lo que facilita diseños más robustos que no requieren ajustes de ultimanento. Esta visión arquitectónica de la usabilidad también favorece la satisfacción y la eficiencia del usuario, ya que permite que las interacciones con el sistema sean más fluidas y menos propensas al error.

Bibliografía: Moreno, A.M., & Sanchez-Segura, M. (2003). Noviembre. Patrones de usabilidad del software desde el momento arquitectónico. In JISBII (p.p 117-126)

Diagrama: Patrón de usabilidad "Deshacer" (Undo) en una aplicación de edición



ARTÍCULOS 05

ARQUITECTURA DE SOFTWARE ESQUEMAS Y SERVICIOS

Resumen: La arquitectura orientada a servicios (SOA) es un enfoque de diseño que permite la creación de aplicaciones distribuidas, donde los servicios independientes se comunican entre sí a través de interfaces bien definidas. A diferencia de los componentes monolíticos, los servicios en SOA tienen una menor granularidad y están diseñados para ser reutilizables, autónomos y de fácil integración. Cada servicio tiene su propia lógica y bases de datos, lo que permite una alta escalabilidad y fiabilidad. Estos servicios se comunican mediante protocolos abiertos, lo que facilita la interoperabilidad entre diferentes plataformas y lenguajes de programación. La implementación del SOA presenta retos, como la gestión de la comunicación entre los servicios, el tiempo de respuesta y el tamaño de los mensajes.

Reflexión: La evolución de la arquitectura ha llevado un gran cambio en cómo se diseñan y desarrollan las aplicaciones. Mientras que en la arquitecturas monolíticas eran más fáciles de desarrollar inicialmente, a medida que las aplicaciones crecen y los requerimientos cambian, estos se vuelven más difíciles de mantener y adaptar. En contraste, SOA promueve la flexibilidad al permitir que los servicios sean independientes y se comuniquen entre sí a través de interfaces abiertas. La integración de múltiples servicios puede ser compleja y la comunicación entre ellos debe ser eficiente y confiable para evitar latencias y problemas de sincronización. A pesar de estos retos, SOA permite una mayor adaptabilidad y capacidad de responder rápidamente y efectivamente para el desarrollo de aplicaciones modernas.

Bibliografía: Romero, P.H (2006). Arquitectura de software, esquemas y servicios. Ingeniería Industrial 27(1). 1.

Diagrama:

Arquitectura Tradicional	Arquitectura Orientada a Servicios (SOA)
<ul style="list-style-type: none">• Componentes monolíticos.• Alto acoplamiento• Escalabilidad limitada• Integración difícil• Flexibilidad baja	<ul style="list-style-type: none">• Servicios autónomos• Comunicación independiente• Bajo acoplamiento• Escalabilidad alta• Flexibilidad alta

ARTÍCULO 06.

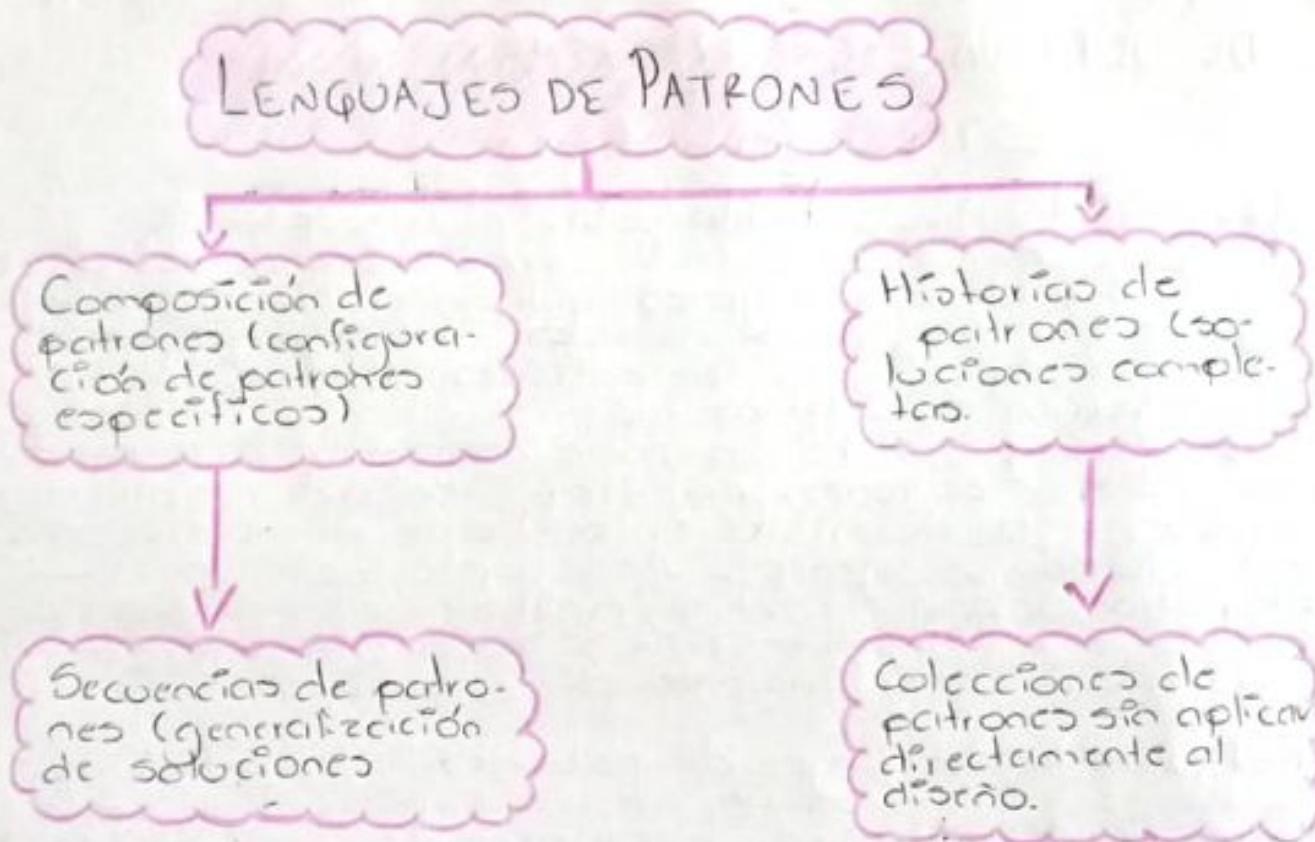
LENGUAJES DE PATRONES DE ARQUITECTURA DE SOFTWARE: UNA APROXIMACIÓN AL ESTADO DEL ARTE

Resumen: el artículo habla sobre el concepto de "Lenguaje de Patrones" en la arquitectura de software. Desde sus orígenes en la ingeniería de software, pasando por la especificación de estos lenguajes en dominios específicos, los patrones han facilitado la construcción de sistemas más robustos y escalables. Estos lenguajes, definidos como conjuntos interconectados de patrones, permiten abordar problemas comunes en el desarrollo de software de manera más estructurada. Un ejemplo de su aplicación son el uso de patrones para resolver problemas de autenticación en sistemas de información o para estructurar la arquitectura de aplicaciones de e-business.

Reflexión: los lenguajes de patrones de arquitectura de software representan un avance significativo en la ingeniería de software, pues permiten simplificar el proceso de diseño y mejorar la calidad de los sistemas. La capacidad de aplicar patrones establecidos a diferentes dominios, como seguridad o e-business, demuestra su flexibilidad y utilidad en contextos diversos. No obstante, la dependencia de patrones también puede llevar a una falta de innovación si se aplica de manera rígida. Es esencial que los desarrolladores mantengan un equilibrio entre la reutilización de patrones existentes y la creatividad en el diseño de soluciones únicas.

Bibliografía: Jiménez-Torres, V.H., Tello-Borja, W. & Ríos-Palma, J.I. (2014). Lenguaje de patrones de software: una aproximación al estado del arte. *Scientia et Technica*, 19(4), 371-376.

Diagrama que describe las categorías de los lenguajes de patrones de arquitectura y como se organizan para facilitar la resolución de problemas.



ARTÍCULO 07

INTRODUCCIÓN DE LA ARQUITECTURA DE SOFTWARE

Resumen: los estilos arquitectónicos son un concepto fundamental para definir la estructura general de un sistema, más allá de los patrones de diseño, ya que describen sistemas complejos y no sólo partes de estos. Un estilo arquitectónico se caracteriza por los tipos de componentes, patrones de interacción y las estructuras que se permiten, influyendo la forma en que el sistema se construye y se implementa. La arquitectura, según los trabajos de Shaw y Garlan, es una parte esencial que guía la construcción de software, y su éxito puede establecerse como un indicador dentro de una organización o incluso en la industria. Además los requerimientos no funcionales, como el desempeño y la seguridad, deben ser considerados, desde el inicio del proyecto para evitar problemas costosos durante las etapas finales del desarrollo.

Reflexión: el ciclo de la arquitectura debe ser un proceso continuo que no sólo se basa en los requerimientos funcionales, sino que también toma en cuenta los no funcionales, como la seguridad, el rendimiento y la escalabilidad, son a menudo ignorados en las primeras fases, lo que genera complicaciones costosas en etapas avanzadas del desarrollo. Es fundamental tener una visión integral desde el principio ya que estos aspectos pueden afectar todo el sistema y no sólo componentes aislados. En este sentido, la arquitectura debe ser diseñada de forma que sea adaptable y capaz de abordar estos desafíos a medida que el sistema crece.

Bibliografía: Cristián, M (2008). Introducción a la arquitectura de software. Rescarch - Gate [online]. Recuperado de: <https://www.rescarchgate.net/publication/251932352>. Introducción a la arquitectura de software

Diseño de arquitectura

Arquitectura	Detalle
• Nivel de abstracción alto	• Nivel de detalle
• Define componentes y relaciones principales de los componentes definidos en la arquitectura	• Refina la implementación de los componentes definidos en la arquitectura
• Enfoque en conectores: Se coloca en la disposición externa de los componentes	
• Considera todos los requerimientos funcionales como no funcionales	• No se ocupan tanto de los requerimientos no funcionales

ARTÍCULO 08.

ATRIBUTOS DE CALIDAD Y ARQUITECTURA DE SOFTWARE

Resumen: la arquitectura de software es crucial para lograr un sistema que no solo sea funcional, sino que también cumpla con atributos de calidad esencial como la seguridad, la disponibilidad, la reusabilidad y la interoperabilidad. Estos atributos son difíciles de equilibrar, ya que mejorar uno podría afectar negativamente a otro. Para abordar este desafío, la arquitectura debe ser definida desde varias perspectivas, con vistas que analicen los características críticas, dinámicas y de interacción con el entorno. Los estilos arquitectónicos influyen significativamente en la calidad del sistema, ya que ciertos estilos priorizan cualidades como la modularidad o la seguridad, pero pueden sacrificar otras como la performance.

Reflexión: es que el proceso de diseñar una arquitectura de software se comienza a la creación de un mapa detallado que gula el futuro del sistema. Las decisiones tomadas durante la fase de diseño no sólo son fundamentales para el éxito del proyecto sino que también determinan la capacidad de evolución y adaptación del sistema ante nuevos desafíos. El uso de diferentes vistas arquitectónicas y la selección de estilos adecuados permitiendo a los desarrolladores tomar decisiones informadas que optimicen el rendimiento, la seguridad y otros atributos críticos. Sin embargo, este proceso no está exento de riesgos, ya que incluso una pequeña modificación en la arquitectura puede tener efectos a largo plazo en la funcionalidad del sistema.

Bibliografía: Bautista, M.C. (2005) Atributos de calidad y arquitectura del software

Diagrama: proceso de documentación de la arquitectura

Jerarquizar la importancia de los vistos para cada Stakeholder



Elegir vistos relevantes (elegir los más representativos)



Combinar y documentar vistos
(crear un documento único)



Documentar información común
(detalles que se aplican varias vistos)

ARTÍCULO 09

ARQUITECTURA DE SOFTWARE PARA EL SOPORTE DE COMUNIDADES ACADÉMICAS VIRTUALES EN AMBIENTES DE TELEVISIÓN DIGITAL INTERACTIVA

Resumen: El artículo habla sobre el diseño de una arquitectura de software para soportar comunidades académicas virtuales en entornos de televisión digital interactiva (TDI). Integrando servicios de e-learning con tecnologías Web 2.0, la arquitectura usa un esquema REST-JSON para mejorar la flexibilidad y la interoperabilidad. Los principales componentes incluye un directorio de servicios, un medidor de canales y set-top box que facilita la interacción del usuario. Estos servicios permiten la participación en tiempo real a través de foros, chats y otros servicios, tanto asociados al contenido televisivo. La propuesta se valida a través de aplicaciones educativas, como el curso AgroEDTV, buscando ampliar la difusión de la televisión interactiva como una herramienta clave en la educación específicamente en regiones con limitaciones de acceso a Internet.

Reflexión: La implementación de comunidades académicas virtuales en entornos de televisión digital interactiva (TDI) destaca por su potencial para cambiar la forma en que se accede a la educación. La arquitectura propuesta no solo optimiza la interactividad de los usuarios, sino que también hace posible una educación más flexible y accesible al integrar servicios como foros y chats en tiempo real. Al utilizar tecnologías como REST-JSON, se facilita la expansión de esta arquitectura hacia otras aplicaciones, incluyendo áreas como e-commerce y social. Esta flexibilidad es crucial para hacer frente a los desafíos educativos actuales y aprovechar las ventajas de la televisión interactiva en un mundo cada vez más digital.

Bibliografía: Campo, W.V. Chanchi, G.E & Arciniega J.L (2003) Arquitectura de software para el soporte de comunicaciones académicas virtuales en ambientes de la televisión digital interactiva.. Formación universitaria, 6(2) 03-46.

Diagrama: ilustrar los componentes claves de la arquitectura propuesta, como el directorio de servicios virtuales, el medidor, el canal de retorno y el set-top box (STB)



ARTÍCULO 10.

PATRONES DE DISEÑO GOF (THE GANG OF FOUR) EN EL CONTEXTO DE PROCESO DE DESARROLLO DE APLICACIONES ORIENTADAS A LA WEB

Resumen: en el contexto de desarrollo de aplicaciones orientadas a la Web, los patrones de diseño GOF han demostrado ser una herramienta importante para mejorar la calidad del software, si embargo, su implementación sigue siendo limitada debido a factores como la falta de conocimiento y experiencia entre los desarrolladores. Los patrones adicionales, como Singleton y Factory Method, fueron aplicados en los proyectos encuestados, seguidos por los patrones de comportamiento como Iterator y Template Method. En los proyectos seleccionados se observó que la aplicación de estos patrones es parcial, y algunos patrones como abstract, Factory, y observer que se integraron mejor en marcos de trabajo que se utilizaron de manera predeterminada. En general, la investigación confirmó que los patrones de diseño son esenciales en el desarrollo de software de alta calidad, pero también destaca que para los desarrolladores pueden usarlo adecuadamente, es necesario un mayor enfoque en la capacitación y el conocimiento práctico.

Reflexión: aunque los patrones de diseño GOF proporcionan soluciones efectivas a problemas comunes en el desarrollo de software, su aplicación en el desarrollo web es aún limitada. Esto se debe en parte a la alta abstracción de los patrones, lo que hace que algunos desarrolladores, especialmente los novatos, tengan dificultades para entender cómo aplicarlos. Además, los patrones de diseño requieren un nivel de experiencia que solo se adquiere con el tiempo. Es fundamental que los desarrolladores no solo aprendan sobre los patrones, sino que los apliquen en proyectos

reales para fortalecer su comprensión. También es necesario que la industria del software se enfoca en simplificar la implementación de estos patrones para que a uno sea más accesible y puede ser aplicado integrando de manera efectiva en el proceso de desarrollo de aplicaciones web.

Bibliografía: Guerrero, C.A. Suárez, J.M & Gutiérrez, L.E (2013) Patrones de diseño GOF (The Gang of Four) en el contexto de procesos de desarrollo de aplicaciones orientados a la web. Información tecnológica 24(3) 103-114.

Diagrama: Los categorías de los patrones de diseño GOF en el desarrollo web.

Categoría	Descripción	Ejemplos	Aplicación en Proyectos
Patrones Creadionales	Controlan la creación de objetos	Singleton, Factory Method	67% de los proyectos
Patrones Estructurales	Organizar las relaciones entre clases y objetivos	Decorator, Facade	30% de los proyectos
Patrones de Comportamiento	Gestiona la interacción y distribución de responsabilidades	Iterator, Template, Method	67% de los proyectos
Resumen	Aplicación general de patrones GOF en proyectos web		

ARTÍCULO 11

INTEGRACIÓN DE ARQUITECTURA DE SOFTWARE EN CICLO DE VIDA DE LAS METODOLOGÍAS AGILES: UNA PERSPECTIVA BASADA EN REQUISITOS

Resumen: la investigación se centra en la integración de los RSA en el contexto de las MA, explorando su potencial impacto en la AS. Se observa que los RSA, aunque un complejo de identificar, son cruciales para una arquitectura sólida y adecuada en proyectos ágiles. La especificación de estos requisitos se basa en la evaluación de necesidades específicas del sistema y en la interacción de diversos stakeholders. Se proponen métodos como entrevistas y análisis de objetivos comerciales para detectar estos requisitos, aunque se reconoce que los RSA pueden variar según el contexto y pueden ser subjetivos, relativos e interrelacionados entre sí. El estudio tiene como objetivo validar la viabilidad de integrar los RSA en las MA, analizando los beneficios de esta integración a través de ejemplos empíricos en teoría y en proyectos de investigación, y contribuyendo con nuevas estrategias para el desarrollo ágil de sistemas.

Reflexión: en los proyectos de desarrollo de software, la colaboración entre las metodologías ágiles y la arquitectura de software puede ofrecer un modelo de desarrollo equilibrado. La identificación temprana de RSA no solo asegura una arquitectura sólida, sino que también aporta una guía estratégica que permite adaptarse a los cambios heterogéneos de las metodologías ágiles. Esto nos hace pensar la importancia de los requisitos no funcionales como un adecuado interpretación puede determinar el éxito del sistema final. En un contexto donde la adaptabilidad y la estructura sólida parecen contradicen, la "Arquitectura Ágil" aparece como una solución viable para integrar estos conflictos y facilitar un desarrollo de software y alineado con las expectativas del cliente.

Diagnóstico: proceso de identificación y captura de RSA.

Bibliografía: Navarro, M.E., Moreno, M.P., Arciada, J., Párra, L., Rueda, J. & Pintor, J.C. (2017, Septiembre). Integración de arquitectura de software en el ciclo de vida de las metodologías. In XIX workshop de investigadores en ciencias de la computación (WICCA 2017) ITBA, Buenos Aires)

Aspectos de los RSA	Descripción
Tipos de requisitos	Se divide en funcionales y no funcionales. Los RSA suelen ser no funcionales, pero pueden incluir funcionales claves.
Impacto en arquitectura	Los RSA afectan profundamente la estructura del sistema; su ausencia puede modificar radicalmente la arquitectura.
Características de los RSA	son objetivos, relativos e interacción entre ellos, lo cual dificulta su identificación y tratamiento en MA
Desafíos en MA	En MA, los RSA suelen percibirse por ello o expresarse de informalmente, lo cual puede resultar en una arquitectura inadecuada
Estrategias para captura	Incluye métodos de entrevistas, análisis de objetivos de negocio y categorización de decisiones arquitectónicas.
Evaluación y Validación	Es crucial para asegurar que la arquitectura cumple con los RSA identificados y las expectativas de los partes interesadas

ARTÍCULO 12.

ALQUITECTURA DE SOFTWARE PARA EL DESARROLLO DE HERRAMIENTA TECNOLÓGICA DE COSTOS, PRESUPUESTOS Y PROGRAMACIÓN DE OBRA

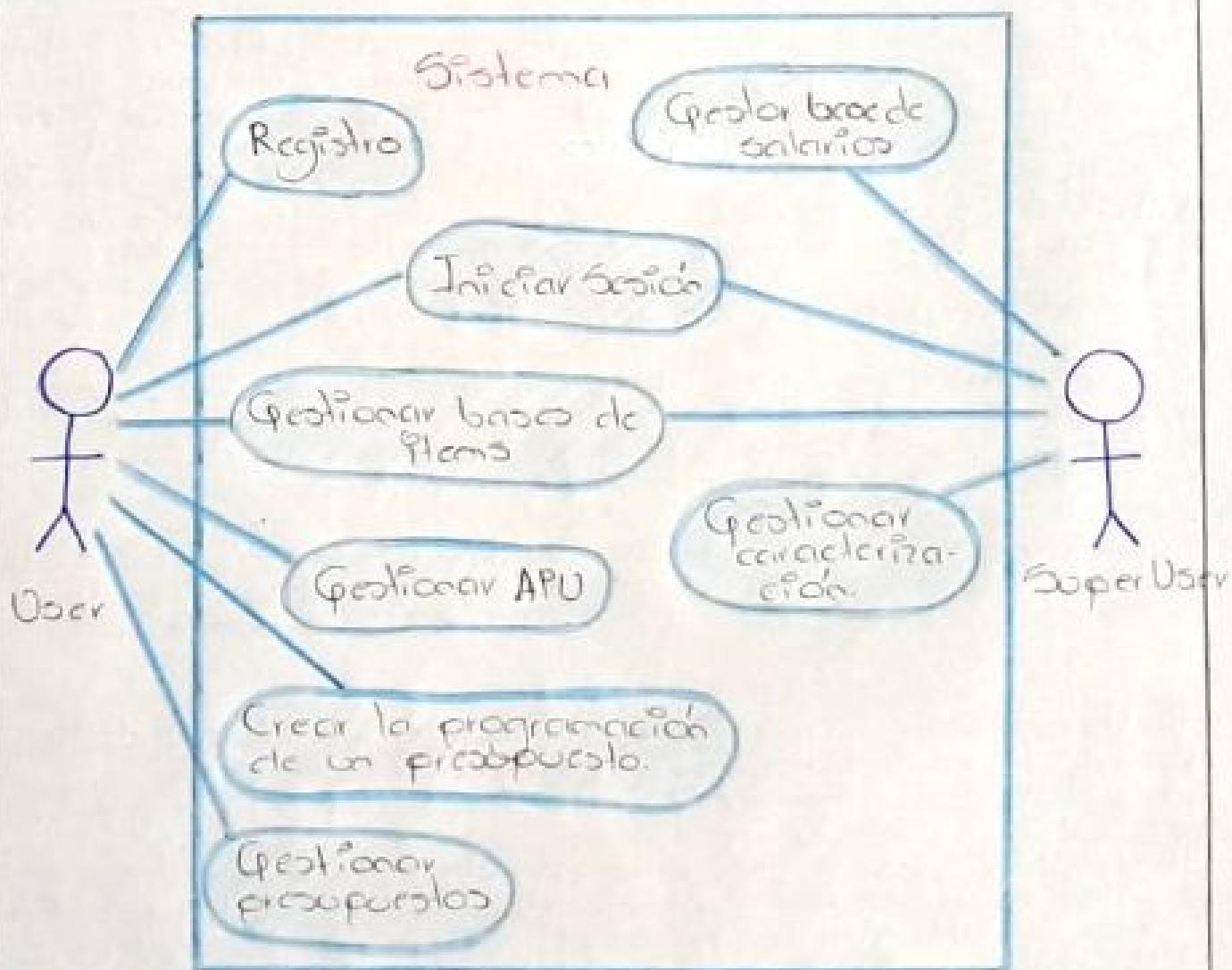
Resumen: El artículo presenta el desarrollo de una arquitectura de software educativo para la materia de "Costos, presupuestos y programación de obra" en la carrera de Ingeniería Civil en la Universidad de San Francisco de Paula de Santaander. Este software está diseñado para apoyar el aprendizaje autónomo de los estudiantes, permitiéndoles gestionar sus conocimientos sobre los temas fundamentales en la construcción, como el análisis de precios unitarios (APU), la planificación de actividades y la optimización de recursos. La herramienta sigue una estructura lógica y utiliza una metodología secuencial que abarca desde la creación de una lista de materiales hasta el cálculo de costos y la programación de tareas. Los desarrolladores utilizaron diagramas UML para representar procesos, como el diagrama de secuencia que ilustra el flujo desde el registro del usuario hasta la creación de presupuestos. Esto permite a los usuarios o estudiantes ver cómo funcionan las actividades de obra en su entorno sin embargo facilitando la comprensión del impacto de cada decisión financiera y logística en un proyecto de construcción.

Reflexión: Esta herramienta transforma el enfoque de la educación en ingeniería civil, combinando teoría y práctica de manera interactiva. Un ejemplo de esto es cómo permite gestionar costos directos como materiales y mano de obra e indirectos (gastos generales) simulando decisiones que los estudiantes deberán tomar en el campo profesional. Este tipo de aprendizaje cíclico no solo facilita la adquisición de habilidades técnicas, sino que también desarrolla competencias críticas como la

lomo de decisiones y de ese software se alinea con la necesidad de que los futuros ingenieros se adapten a un mundo en constante avance tecnológico donde el dominio de la herramienta digital es cada vez más esencial.

Bibliografía: Cárdenas-Gutiérrez, J.A., Barrios Monroy, E.J. & Molina Salazar, L. (2022) Arquitectura de software para el desarrollo de herramientas tecnológicas de costos, presupuesto y programación de obra II. Revista de Investigaciones (17 y 18), 83, 309-00

Diagrama: Caso de uso del registro e inicio de sesión del aplicativo



ARTÍCULO 13.

ARQUITECTURA DE SOFTWARE PARA EL SISTEMA DE VISUALIZACIÓN MÉDICA VISMEDIC

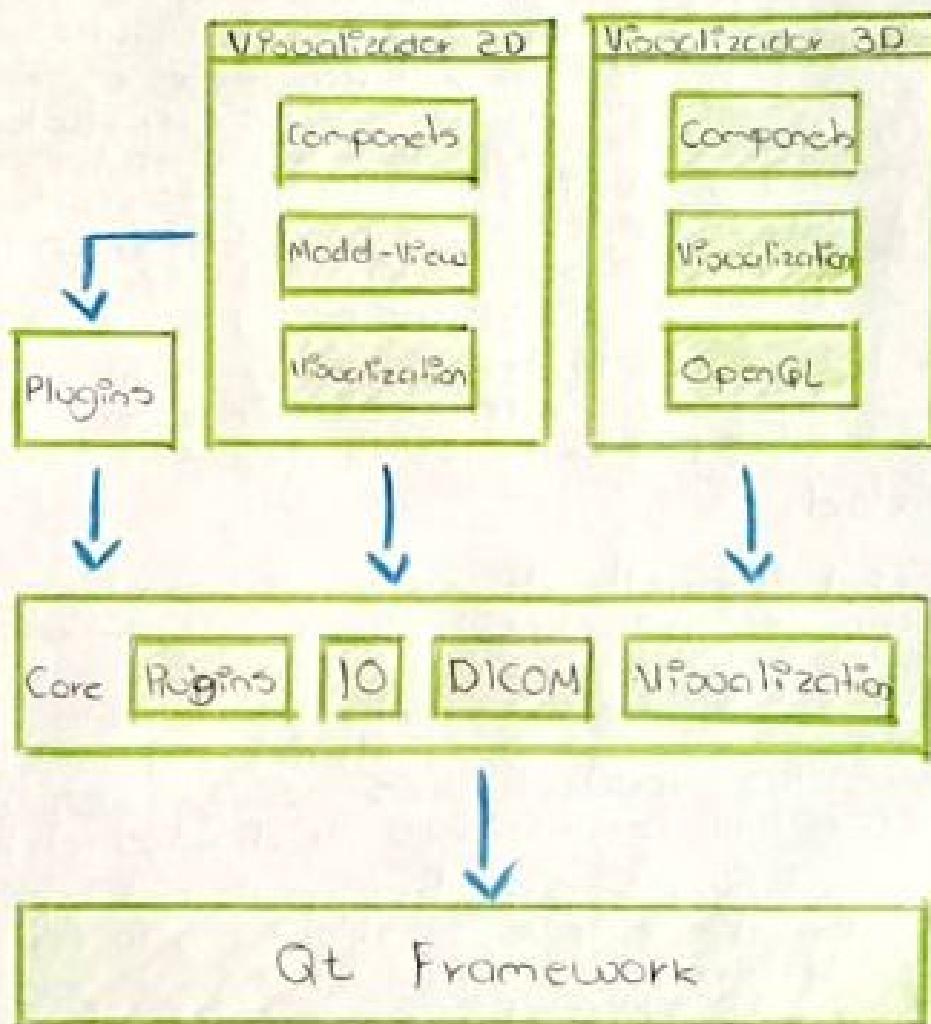
Resumen: el artículo aborda la propuesta de una nueva arquitectura de software para el sistema de visualización médica Vismedic, orientada a resolver problemas de extensibilidad, reusabilidad y dependencia en su diseño actual. La propuesta combina tres estilos arquitectónicos: arquitectura basada en capas, arquitectura basada en componentes y tuberías y filtros, integrando el uso de plug-ins para extender funcionalidades. La validación se llevó a cabo mediante la técnica de evaluación basada en prototipos y el método ATAM (Architecture Trade-off Analysis Method), concluyendo la nueva arquitectura mejoró atributos como portabilidad, mantenibilidad, accesibilidad y reusabilidad.

Reflexión: el desarrollo de sistemas complejos, como el de Vismedic, exige arquitecturas que permiten flexibilidad y adaptación a cambios sin comprometer su estabilidad. La propuesta del artículo demuestra que la adopción de enfoques híbridos y modulares, como los estilos arquitectónicos mencionados, es esencial para superar las limitaciones de diseños cíclicos. Esto resalta la importancia de anticipar futuros requisitos durante la fase de diseño y la relevancia de incluir técnicas de evaluación para validar decisiones arquitectónicas. En esencia, planificar y diseñar con un enfoque adaptable no solo mejora la eficiencia del desarrollo, sino que también reduce el esfuerzo requerido para realizar modificaciones al sistema.

Bibliografía: Rodríguez Peña, A.D., & Silvia Rojas, L.G. (2016). Arquitectura de software para el sistema de visualización médica Visomedic. Revista Cubana de Informática Médica, 8 (1) 75-86.

Diagrama:

Arquitectura actual del proyecto Visomedic



ARTÍCULO 14.

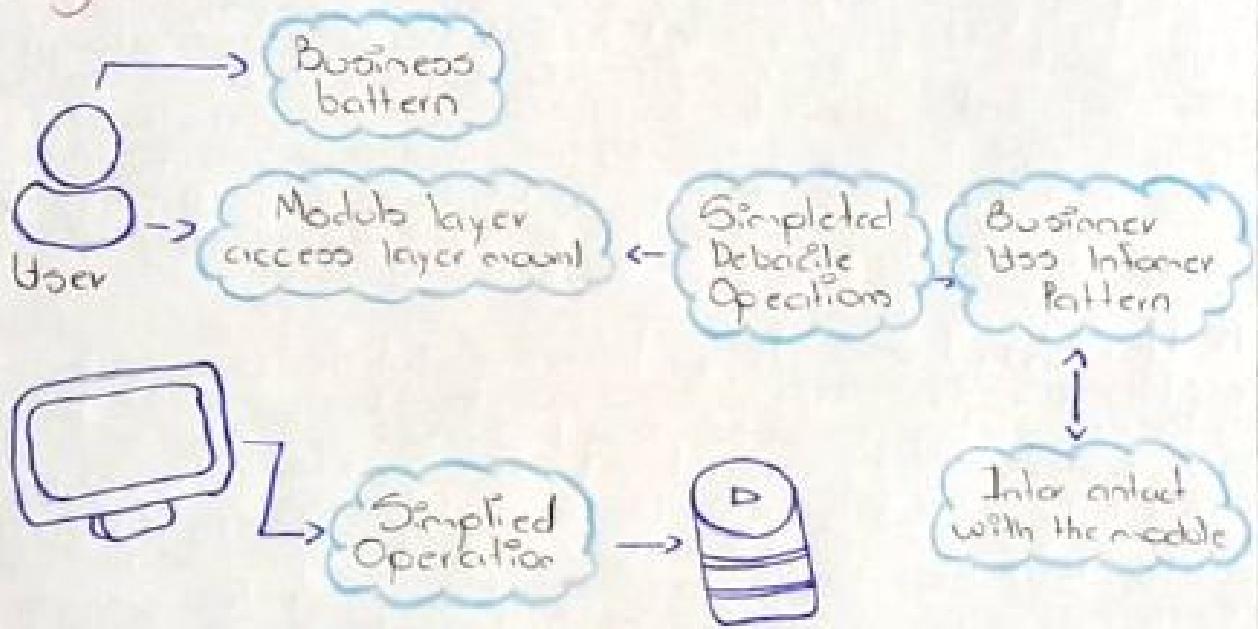
Desarrollo de sistemas de software con patrón de diseño orientado a objetos

Resumen: el proyecto "Intranet Industrial" desarrollo para la facultad de Ingeniería Industrial de la UNMSM aplica patrones de diseño para resolver problemas comunes en sistemas complejos. Usando el patrón informador y el sensor, se mejoró la gestión de operaciones con bases de datos y el manejo de errores en toda la aplicación. Además, la arquitectura de tres capas permitió separar la presentación lógica de negocio y los datos, garantizando modularidad y escalabilidad. La metodología RUP, junto con herramientas como UML y modelamiento visual, favoreció un desarrollo iterativo que redujo riesgos y optimizó recursos. Finalmente el impacto económico y en plazos se viviloó comparando el uso y el no uso de patrones, demostrando ahorros significativos en ambas métricas.

Reflexión: el uso de patrones de diseño no solo aborda problemas técnicos, sino que también mejora la calidad del producto final. Un sistema desarrollado con patrones es más fácil de mantener, escalar y adaptar, beneficiando tanto a los desarrolladores como a los usuarios. Además, metodologías estructuradas como RUP ayudan a gestionar mejor los proyectos asegurando que se cumplen los objetivos iniciales de forma eficiente y confiable. Este ejemplo resalta la importancia de seguir estandares y prácticas comprobadas en el desarrollo de software.

Bibliografía: Lazo, L. & Paul, J (2004). Desarrollo de sistemas de software con patrones de diseño orientado a objetos

Diagram:



ARTÍCULO 15.

ARQUITECTURA DE SOFTWARE EN EL PROCESO DE DESARROLLO ÁGIL: UNA PERSPECTIVA BASADA EN REQUISITOS PARA LA ARQUITECTURA

Resumen: los requisitos significativos para la arquitectura (R_SA) son elementos esenciales que influyen directamente en la estructura, diseño y calidad de un sistema de software. Estos requisitos pueden ser naturaleza funcional o no funcional. Sin embargo, su impacto va más allá de lo evidente: si no son identificados y considerados adecuadamente, pueden derivar en una arquitectura débil o insuficiente.

Incapaz de satisfacer los objetivos clave del sistema. Una de las mayores retos de los R_SA radica en su subjetividad, pues su relevancia depende del contexto y las perspectivas de quienes los interpretan.

Además, existen interacciones entre sí, donde priorizar uno puede beneficiar o perjudicar a otros. El artículo aborda esta problemática proponiendo estrategias como el uso de frameworks en enfoque en conocimiento del dominio y la alineación con objetivos empresariales para integrar los R_SA de manera eficiente en los ciclos ágiles. Este enfoque no solo permite gestionar cambios sin comprometer la arquitectura, sino que también asegura que atributos clave como rendimiento, seguridad y accesibilidad sean respetados a lo largo del proyecto.

Conclusiones: Identificar y trabajar con los R_SA es un paso fundamental en la construcción de sistemas de software sostenibles y adaptables. En lugar de ver la arquitectura como un elemento estático que limita el cambio, este enfoque la transforma en un hábito, capaz de responder a los demandas de un mercado dinámico. El artículo destaca la importancia de adoptar estrategias claras y flexibles para gestionar los R_SA.

como basarse en el conocimiento del entorno o los metodos del negocio. Esto asegura un equilibrio entre flexibilidad y estabilidad, los conceptos aparentemente opuestos pero complementarios para el éxito en metodologías ágiles.

Bibliografía: Nuñez, M. E., Moreno, M.P., Arredondo, J., Peirce, L., & Roedel, J. F. (2018). Arquitectura de software en el proceso de desarrollo ágil: una propuesta basada en requisitos significativos para la arquitectura. In XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste)

Diccionario:

Método	Descripción	Ventajas	Desafíos
Framework	Basados en cambios empíri- cos con expertos	Estandarización y robustez	Requiere ex- pertos califi- cados
Conocimiento del Dominio	Entendimiento profundos del entorno y supo- rtaciones	Adaptabilidad a contextos específicos	Limitado por el alcance del conocimiento
Objetivos del Negocio	Priorizaciones empresariales y atributos de calidad	Relación directa con el valor del negocio	Puede ser objeti- vo encierto industrial

ARTÍCULO 16.

DESARROLLO DE UNA HERRAMIENTA PARA EL APRENDIZAJE DE PATRONES DE DISEÑO SOFTWARE

Resumen: la herramienta se desarrolla en un contexto educativo y profesional, para simplificar la comprensión y aplicación de patrones de diseño con metodologías ágiles como Scrum, el proyecto prioriza la creación de un producto mínimo viable que evoluciona iterativamente.

Destacados del desarrollo:

- Uso de encuestas para identificar necesidades de los usuarios.
- Priorización de características como explicaciones claras y ayudas visuales.
- Implementación de una arquitectura de tres capas (representación, negocio, datos).

Resultados esperados:

Con una interfaz intuitiva y un contenido adaptable se espera que docentes y alumnos puedan personalizar y ampliar el uso de la herramienta según sus necesidades.

Reflexión: La construcción de este herramienta no solo plantea un avance en el ámbito académico, sino que también introduce un enfoque innovador en la creación de herramientas educativas. Las metodologías ágiles permiten adaptarse a cambios y mejorar la alineación con las necesidades y realidades del usuario.

Bibliografía: Ferrandis Horri A. (2021) Desarrollo de una herramienta para el aprendizaje de patrones de diseño software (Doctoral dissertation) Universitat Politècnica de València.

Dicograma:

Aspecto	Fortalezas	Debilidades
Educativo	Explicaciones detalladas, curriculares.	Requiere formación previa en patrones.
Técnico	Uso de tecnologías modernas y ágiles	Complejidad en la implementación.
Accesibilidad	Aplicación web multiplataforma	Dependencia de conexión a internet
Escalabilidad	Extensible y modular	Limitada a patrones preconfigurados

ARTÍCULO 17.

APLICACIONES DE PATRONES DE DISEÑO PARA GARANTIZAR ALTA FLEXIBILIDAD EN SOFTWARE

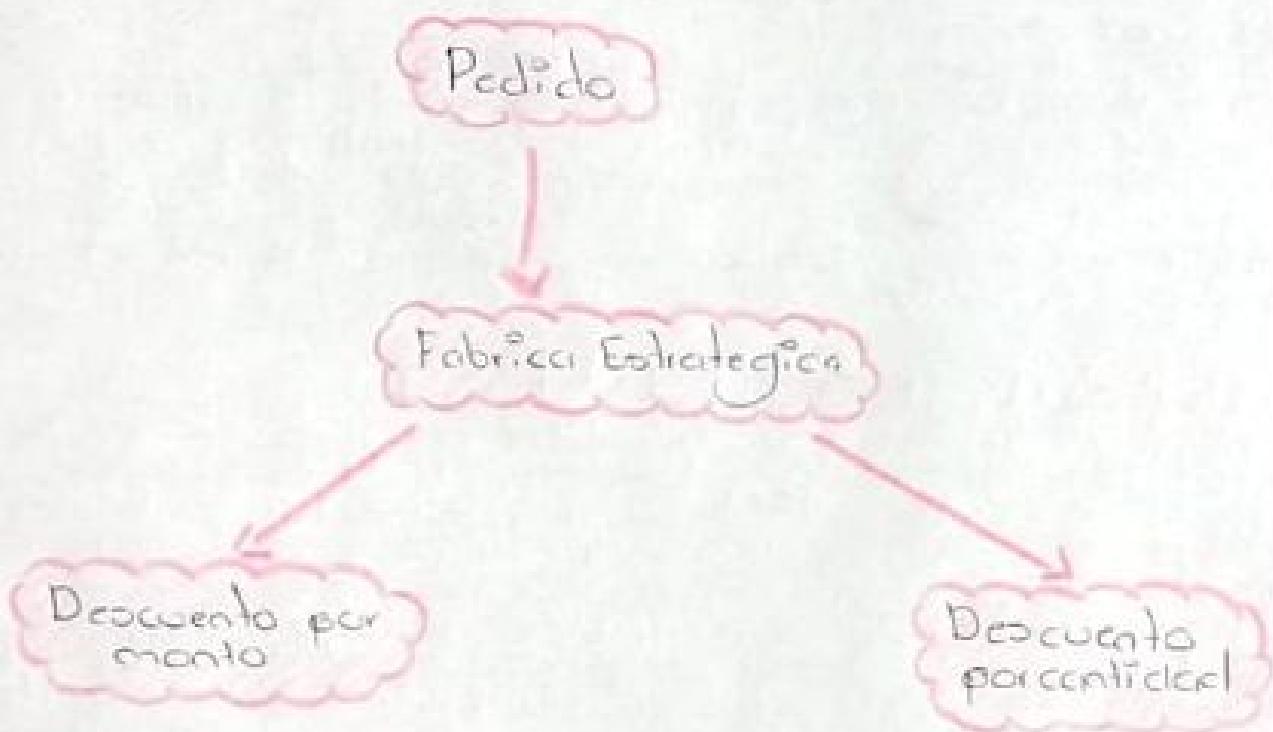
Resumen: El artículo analiza cómo los patrones de diseño permiten desarrollo de aplicaciones empresariales flexibles frente a cambios constantes en sus reglas de negocio. A través del caso de un sistema de pedidos, se implementan patrones como estrategias, composito y fábrica para estructurar el diseño y gestionar reglas de desarrollo variables. La clase principal "pedido" delega las operaciones relacionadas con los descuentos a las estrategias específicas garantizando alta coherión. Por parte, la fábrica se encarga de instanciar las estrategias necesarias promoviendo bajo acoplamiento. Este enfoque respeta el principio abierto-cerrado, permitiendo añadir nuevos reglas de negocio sin alterar el código existente.

Reflexión: La flexibilidad del software no solo depende de la tecnología utilizada, sino también de un diseño sólido fundamentado en patrones reconocidos. Este artículo ilustra cómo los patrones de diseño, combinados con principios como el abierto-cerrado, garantizan que las aplicaciones sean sostenibles y escalables a lo largo del plazo. En el caso de sistemas de pedidos, el modulabilidad logrado mediante el uso de patrones de estrategia y composito permite manejar la complejidad sin comprometer la coherencia del sistema.

Bibliografía: Escalante, L.C. (2014) aplicación de patrones de diseño para garantizar alta flexibilidad en el software. Tecnología y Desarrollo (Trujillo) 12(1), 77-82

Diagramas:

Diagrama de relación entre pedido y fábrica



ARTÍCULO 18.

HERRAMIENTA PARA REUSO DE CÓDIGO JAVASCRIPT ORIENTADO A PATRONES DE INTERACCIÓN

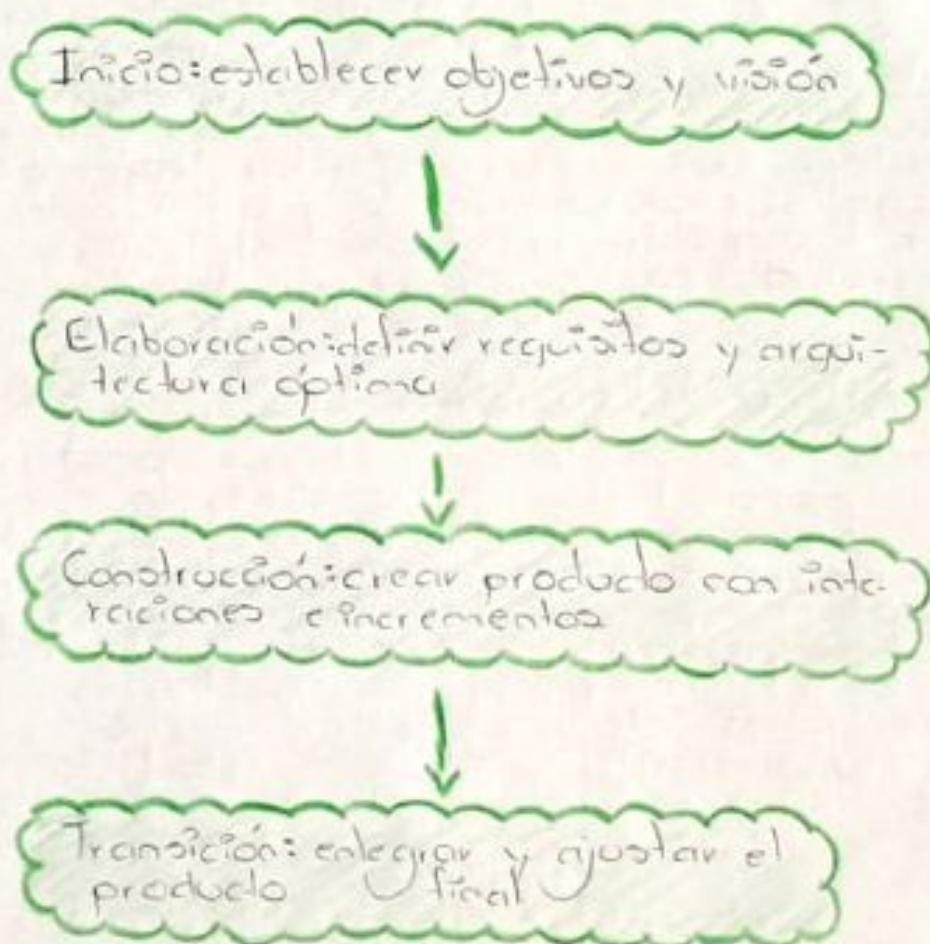
Resumen: el proyecto ReusMe ofrece resultados significativos para el desarrollo web, destacando su capacidad para generar interfaces sencillas y eficientes mediante la reutilización de código Javascript basado en patrones de interacción. Los patrones incluyen soluciones específicas adaptadas a problemas comunes, como botones, cuadros desplegables y otros componentes de interfaz. ReusMe permite personalizar esas soluciones, ofreciendo flexibilidad y control al usuario final. Esto representa una ventaja respecto a heredamientos tradicionales, que suelen limitar las opciones de personalización. Además la plataforma asegura ahorro de tiempo y recursos al evitar que los desarrolladores comience desde cero en cada proyecto, coincidiendo con la eficiencia.

Reflexión: la reutilización de código, como lo propone ReusMe, representa un enfoque sostenible y práctico para el desarrollo web, que prioriza la eficiencia sin sacrificar la calidad. Al emplear patrones de interacción reutilizables, los desarrolladores pueden resolver problemas comunes de diseño de manera más rápida y efectiva, reduciendo el tiempo y los costos asociados con la creación de aplicaciones desde cero. Este modelo fomenta una comunidad de desarrolladores más colaborativa, donde las soluciones probadas y exitosas se comparten y se adaptan a diferentes contextos. Sin embargo también plantea la necesidad de mantener altos estándares de documentación y pruebas para asegurar que si código reutilizable sea confiable y funcional.

Bibliografía: Benigno, G., Antonelli, O., & Masucci, Y. (2009). TOOL FOR REUSE OF JAVA-SKIRT CODE ORIENTED TO INTERACTION PATTERNS. *SABER Multidisciplinary Journal of the Research Council of the University of the East*, 21(1), 60-69.

Diagrama:

Ciclo de desarrollo según PUD en ReuseMe



ARTÍCULO 19

ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB

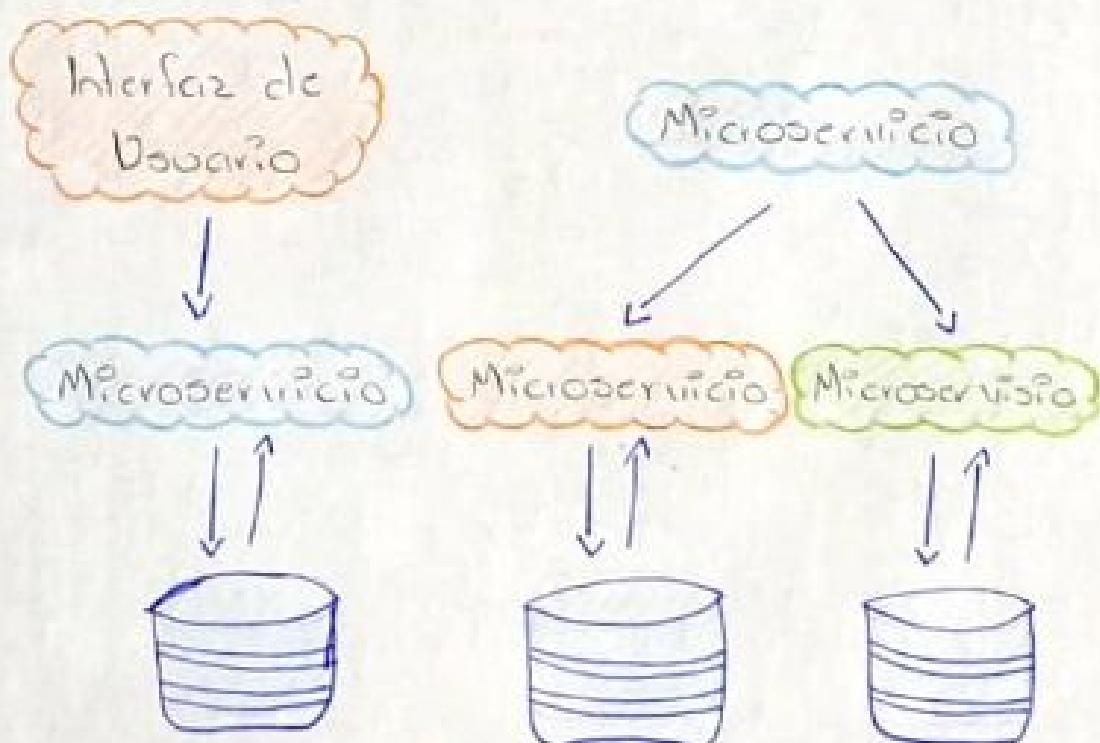
Resumen: el texto describe cómo las arquitecturas monolíticas limitan la capacidad de los organizaciones para mantener y actualizar aplicaciones web complejas. Este enfoque, que centraliza todo la funcionalidad en un solo bloque, genera problemas como tiempos prolongados para implementar actualizaciones, altas tasas de fallas y dificultades para escalar sistemas. La arquitectura monolítica también incrementan los costos asociados al mantenimiento y dificulta la adopción de nuevas tecnologías. En respuesta, se plantea la implementación de una arquitectura de microservicios, que permite desarrollar y desplegar componentes de forma independiente. Cada microservicio tiene su propia lógica de negocio y base de datos, lo que facilita el mantenimiento y la implementación de cambios sin afectar el resto del sistema.

Reflexión: el documento subraya que la elección de una arquitectura de software es una decisión clave que afecta directamente la eficiencia, escalabilidad y mantenimiento de los sistemas. Las limitaciones de las aplicaciones monolíticas, especialmente en el sector público, dificultan la agilidad organizacional y aumentan los riesgos operativos. La arquitectura de microservicios ofrece una alternativa moderna que permite descomponer las aplicaciones en componentes separados y autónomos, mejorando la flexibilidad y el tiempo de respuesta ante cambios. Sin embargo, este modelo también introduce nuevos desafíos, como la seguridad y la complejidad de la gestión de redes distribuidas.

Bibliografía: López, D & Mayor, E (2014)

Arquitectura de software basada en microservicios para el desarrollo de aplicaciones web

Diagrama:



ARTÍCULO 20

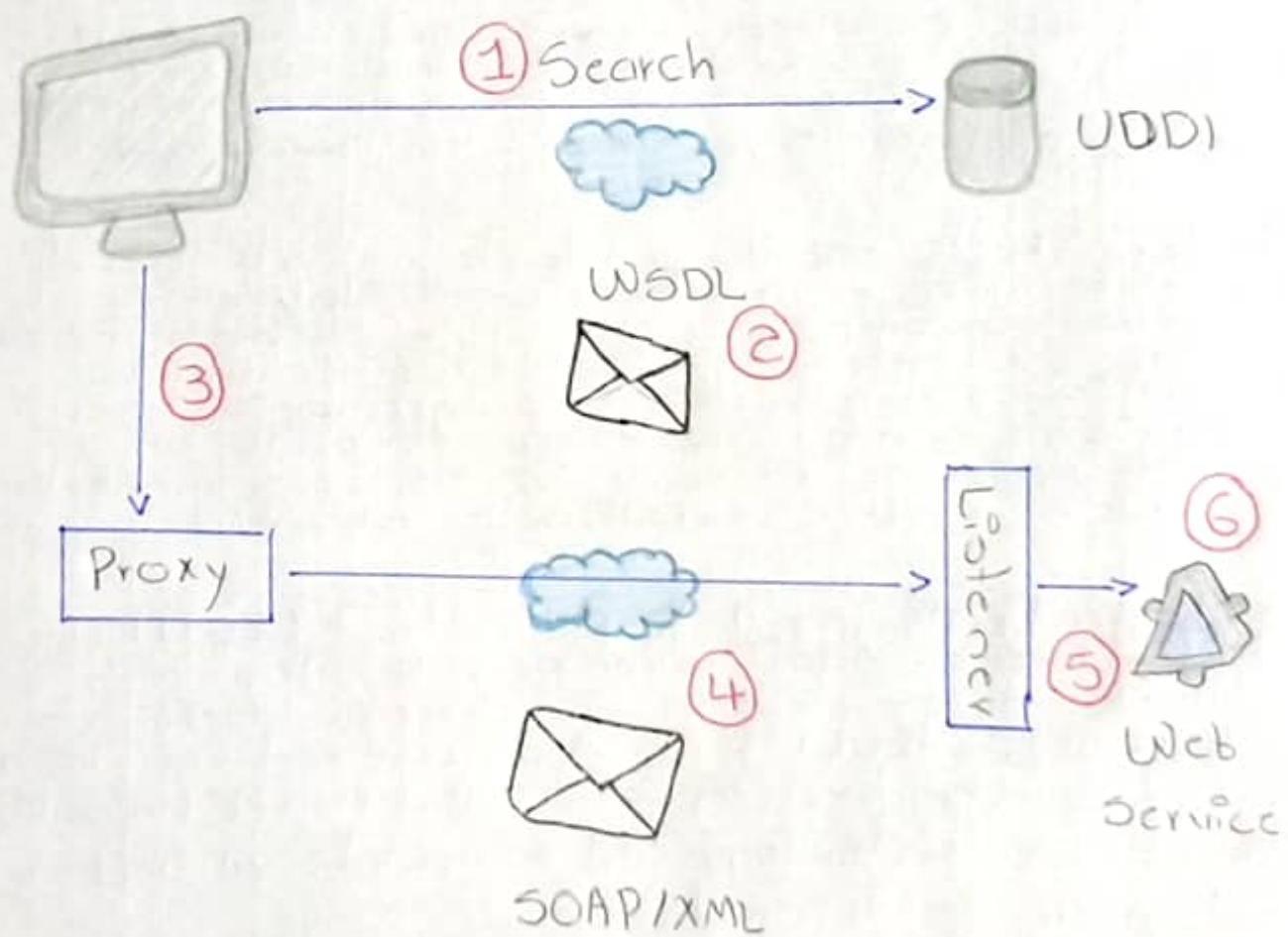
UNA ARQUITECTURA DE SOFTWARE PARA LA INTEGRACIÓN DE OBJETOS DE APRENDIZAJE BASADA EN SERVICIOS WEB

Resumen: el artículo presenta una arquitectura de software para integrar repositorios de objetos de aprendizaje (LOA) y sistemas de gestión de aprendizaje (LMS) mediante tecnologías web como SOA (arquitectura orientada a servicios). Esto mejora la interoperabilidad, reutilización e integración de recursos educativos. SCORM es el estándar clave que garantiza que los objetos educativos puedan transferirse entre sistemas. La arquitectura propuesta incluye capas funcionales como diseño, búsqueda y almacenamiento, y permite a los desarrolladores crear cursos combinando objetos de aprendizaje de múltiples repositorios distintos.

Reflexión: este modelo resalta la importancia de la colaboración en la educación digital, eliminando barreras tecnológicas. Sin embargo, su implementación requiere una inversión inicial significativa y conocimientos avanzados en programación. Reflexionar sobre cómo estos arquitecturas pueden aplicarse en contextos menos desarrollados rural o crudos para democratizar la educación.

Bibliografía: Rojas, M., & Montaña, J. (2011). Una arquitectura de software para la integración de objetos de aprendizaje basada en servicios web. In Ninth L'ACCEI Latin American and Caribbean Conference Engineering for a Smart Planet Innovation Information Technology and Computational Tools for Sustainable Development.

Proceso y tecnologías de los servicios web



ARTÍCULO 21.

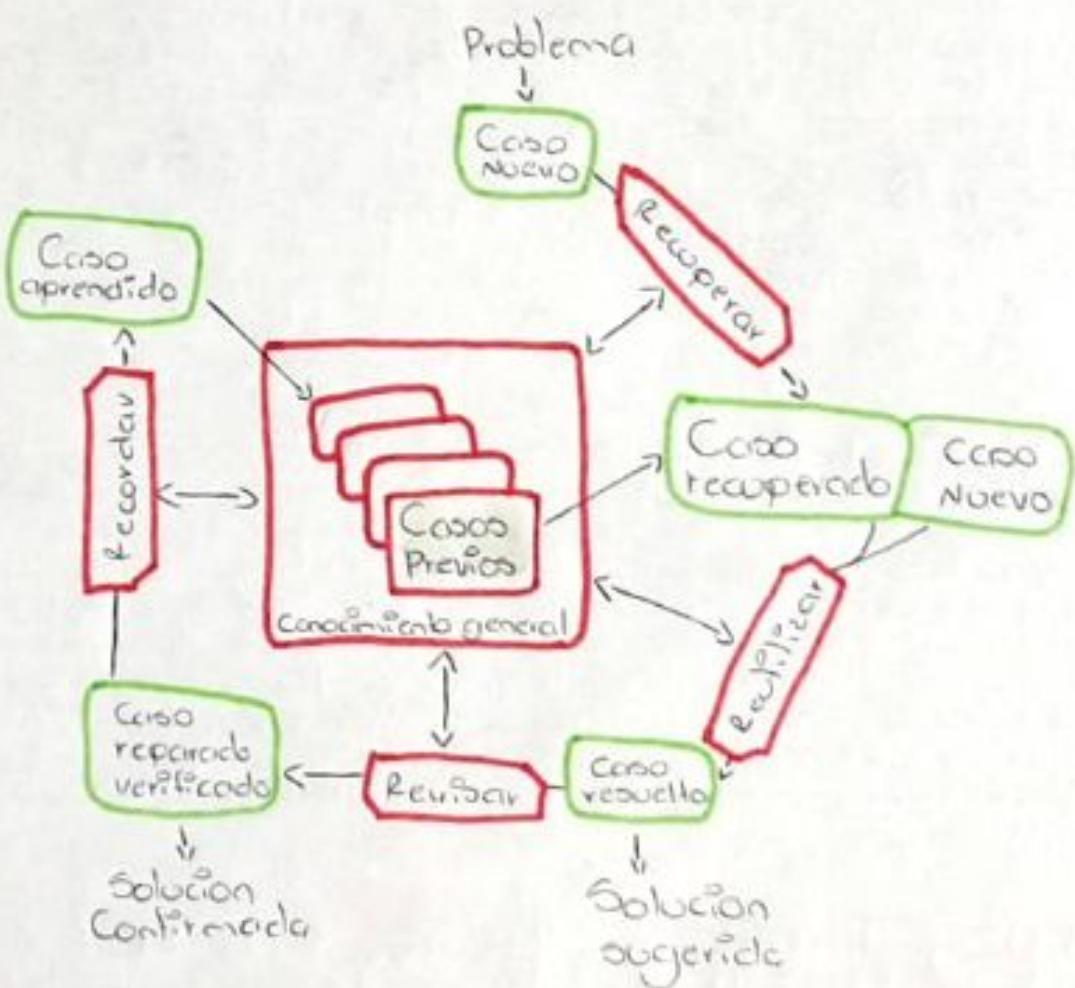
MÓDULO DE RECOMENDACIÓN DE PATRONES DE DISEÑO PARA EGPAT

Resumen: el avance de las tecnologías ha facilitado el acceso a recursos educativos mediante plataformas digitales. Sin embargo, los diseñadores enfrentan problemas en la creación de recursos que cumplen sus objetivos, debido al desconocimiento o la dificultad para acceder a patrones de diseños adecuados. Estos patrones, almacenados en repositorios, son soluciones probadas que mejoran la estructura, reutilización y efectividad de los recursos educativos. El grupo de tecnologías de apoyo a la educación (GITA) desarrolló un entorno, EGPAT, que permite la gestión y consumo de patrones de diseño además de recomendar los más adecuados para problemas específicos mediante minería de texto y técnicas de razonamiento basado en reglas.

Reflexión: esta investigación destaca cómo la tecnología puede democratizar el acceso a herramientas avanzadas, reduciendo la brecha técnica entre expertos y novatos. EGPAT refleja una solución innovadora que no solo mejora la eficiencia en el diseño, sino que también fomenta la creación de materiales educativos más inclusivos y adaptados a diversas necesidades. Sin embargo, para maximizar su impacto se deben abordar barreras como la accesibilidad y el acceso multilingüe.

Bibliografía: Alfonso Azcurry, F & Llull Cepedas (2021). Módulo de recomendación de patrones de diseño para EGPAT. Revista Cubana de Ciencias Informáticas, 15(2), 118-137

Diagrama:



ARTÍCULO 22.

ANÁLISIS COMPARATIVO DE PATRONES DE DISEÑO DE SOFTWARE

Resumen: el artículo presenta un análisis comparativo de cinco patrones de diseño de software: Template Method, Model-View-Controller (MVC), Model-View-Presenter (MVP), Model-Front Controller y Model-View-ViewModel (MVVM). Se destacan sus características, ventajas y desventajas, abordando aspectos como la modularidad, reutilización del código y facilidad para pruebas unitarias. La investigación concluye que no existe un patrón superior, ya que cada uno es útil para resolver problemas específicos. Los patrones son ejemplos para estructurar aplicaciones robustas y facilitar el mantenimiento del software.

Reflexión: este análisis resalta la importancia de seleccionar patrones de diseño adecuados según el contexto del proyecto. Si bien los patrones son herramientas potentes para modularidad y escalamiento, también exigen un conocimiento técnico fuerte a los desarrolladores al evaluar cuidadosamente las necesidades del sistema antes de decidir qué patrón adoptar, optimizando así los recursos y evitando sobreingeniería.

Bibliografía: Muñoz, O. D. Q., Lorca, N. P. L. & Valenzuela, M. V. B. (2022). Análisis comparativo de patrones de diseño de software. Boletín del conocimiento. Revista científica profesional, 7(7) 2146-2163.

Diagramas:

Métrica	simple metodo	Modelo w-frente muy	Modelo view- muy	Modelo front muy	Modelo view-Model muy
Cooperabilidad de aplicación	Medio	Alto	Mto	Bajo	Bajo
Montabilidad de aplicación	Medio	Alto	Mto	Alto	Alto
Acoplamiento con los módulos	Medio	Bajo	Bajo	Bajo	Bajo
Facilidad de implementación	Mto	Bajo	Bajo	Bajo	Bajo
Facilidad para testing	Mto	Alto	Mto	Mto	Alto
Comunicabilidad con multiparadigmas de programación	Bajo	Bajo	Bajo	Bajo	Bajo
Montabilidad de aplicación	Medio	Alto	Alto	Alto	Alto

ARTICULO 23.

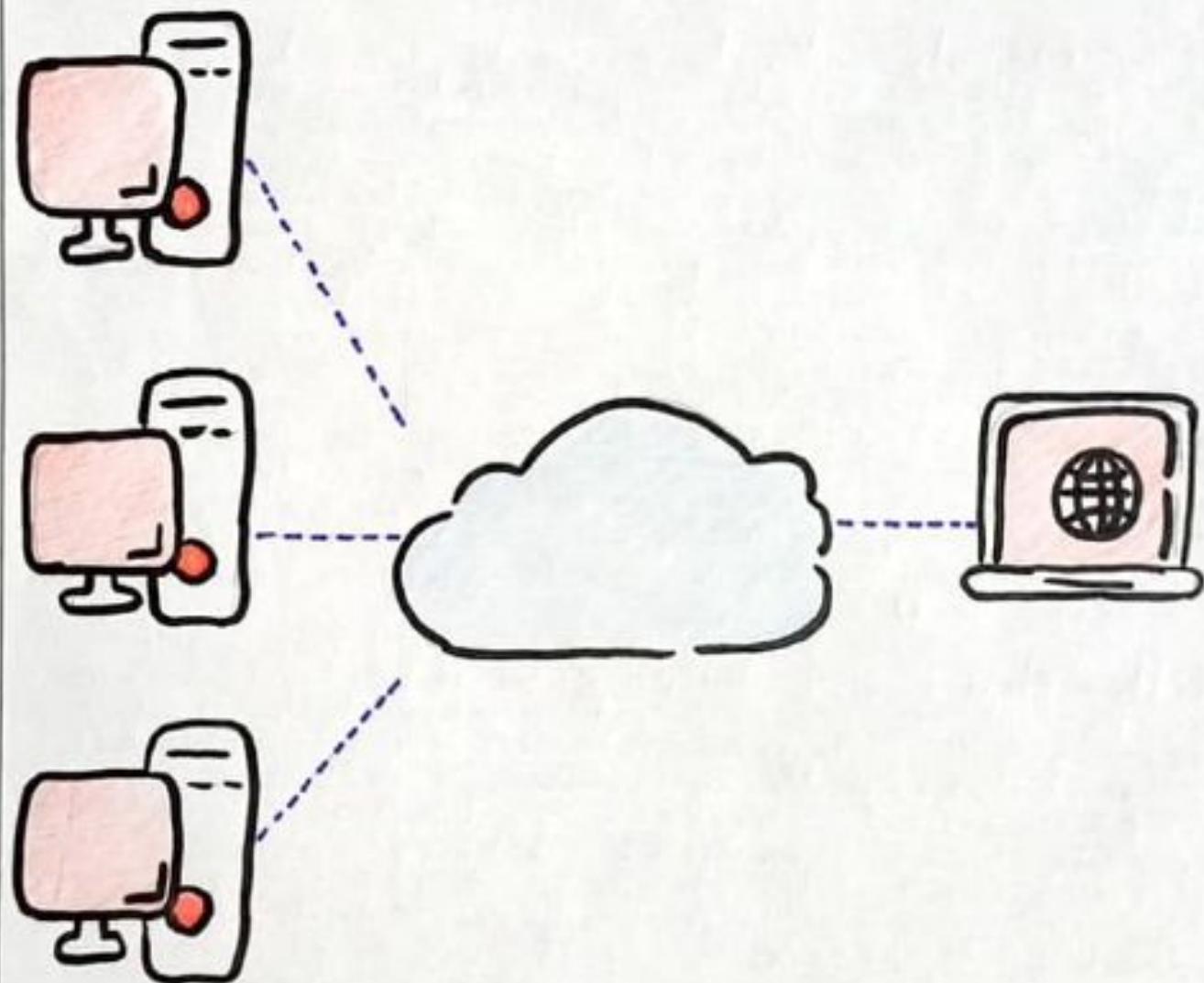
MODULO Y VERIFICACION DE PATRONES DE DISEÑO DE ARQUITECTURA DE SOFTWARE PARA ENTORNOS DE COMPUTACION EN LA NUBE

Recientemente se ha presentado un entorno de diseño específico que apoya en la creación de arquitecturas de software destinadas a aplicaciones web en la nube. El entorno se basa en un metamodelo de componentes arquitectónicos que establece un conjunto de elementos recurrentes y patrones específicos en sistemas cloud. Alentando la modularización del diseño, incorporando una herencia entre la verificación gráfica que asegura la coherencia de los patrones de diseño. Al establecer el proceso de diseño en módulos específicos, esta herencia permite a los arquitectos generar diseños que mantienen la calidad y la alineación con los requisitos del sistema, en particular los no funcionales (escalabilidad y seguridad).

Reflexión: La computación en la nube tiene nuevos desafíos al diseño de software debido a la necesidad de integrar sistemas en infraestructuras compartidas, gestionar recursos distribuidos y adaptarse a los entornos cambiantes. Este entorno permite a los arquitectos abstraerse de complejidades, guiandolos hacia la creación de arquitecturas estandarizadas y consistentes. Los módulos de diseño y verificación ofrecen un marco de referencia para asegurar que cada componente responde a los principios de calidad y diseño establecidos, optimizando el trabajo arquitectónico y reduciendo el riesgo de errores. Este enfoque modular y verificado es fundamental en contextos de alta demanda, donde se requiere tanto escalabilidad como flexibilidad.

para adaptarse a las demandas del negocio y a la tecnología en constante evolución

Bibliografía: Blas, M. J. Leone, H. D. & Gómez, J. M.
(2019) Modelado y Verificación de patrones de diseño de arquitecturas de software para entornos de Computación en la Nube



ARTÍCULO 24.

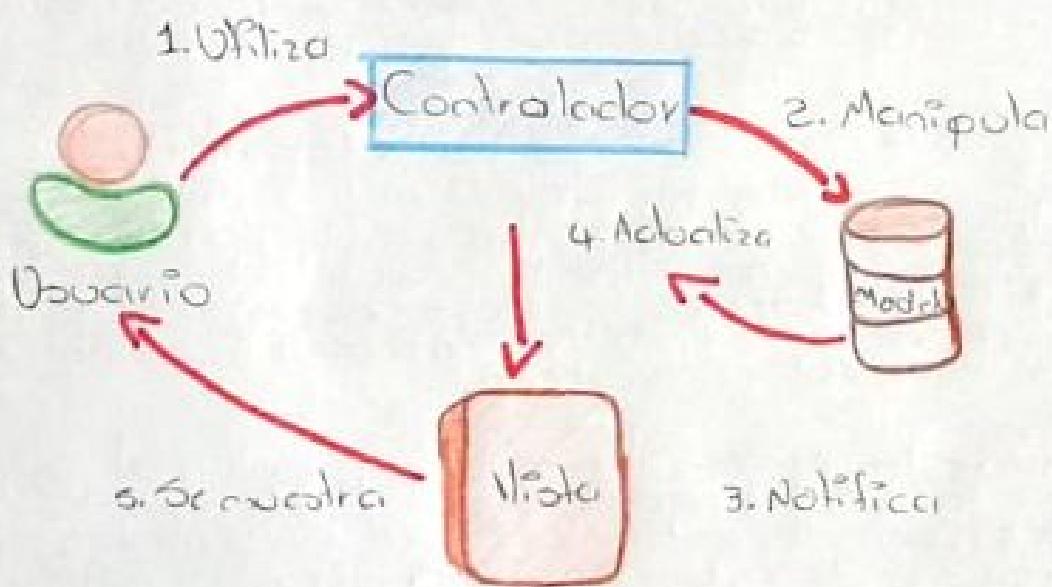
ARQUITECTURA SOFTWARE REUTILIZABLE BASADA EN PATRONES DE DISEÑO Y PATRONES DE INTERACCIÓN, PARA EL DESEARROLLO RÁPIDO DE APLICACIO- NES WEB

Resumen: el documento propone una arquitectura de software reutilizable basada en patrones de diseño y patrones de interacción, enfocada en acelerar el desarrollo de aplicaciones web multiplataforma. El framework busca solucionar problemas comunes del diseño y la interacción mediante estandares abiertos como XML y tecnologías modernas como PHP, MySQL y Ajax. Una de sus características principales es la implementación del patrón MVC, que separa las capas de presentación lógica de negocio y control para facilitar la escalabilidad y mantenibilidad. Además, apoya puntos calientes y congelados, render persistente, personalización y rotación, respectivamente.

Reflexión: esta propuesta es un ejemplo del tipo que impone la reutilización de componentes y la adopción de estándares en la ingeniería del software. Permite a los desarrolladores enfocarse en los requisitos específicos de cada proyecto sin reinventar procesos básicos. Sin embargo, el éxito de este tipo de framework depende de la calidad de la documentación y la flexibilidad para adaptarse a contextos variados. Es fundamental que los equipos de desarrollo también invierten en comprender más herramientas para maximizar su potencial.

Bibliografía: Muñoz, M. M (2010). Arquitectura de software reutilizable basada en patrones de diseño y patrones de interacción para el desarrollo rápido de aplicaciones web.

Diagramas



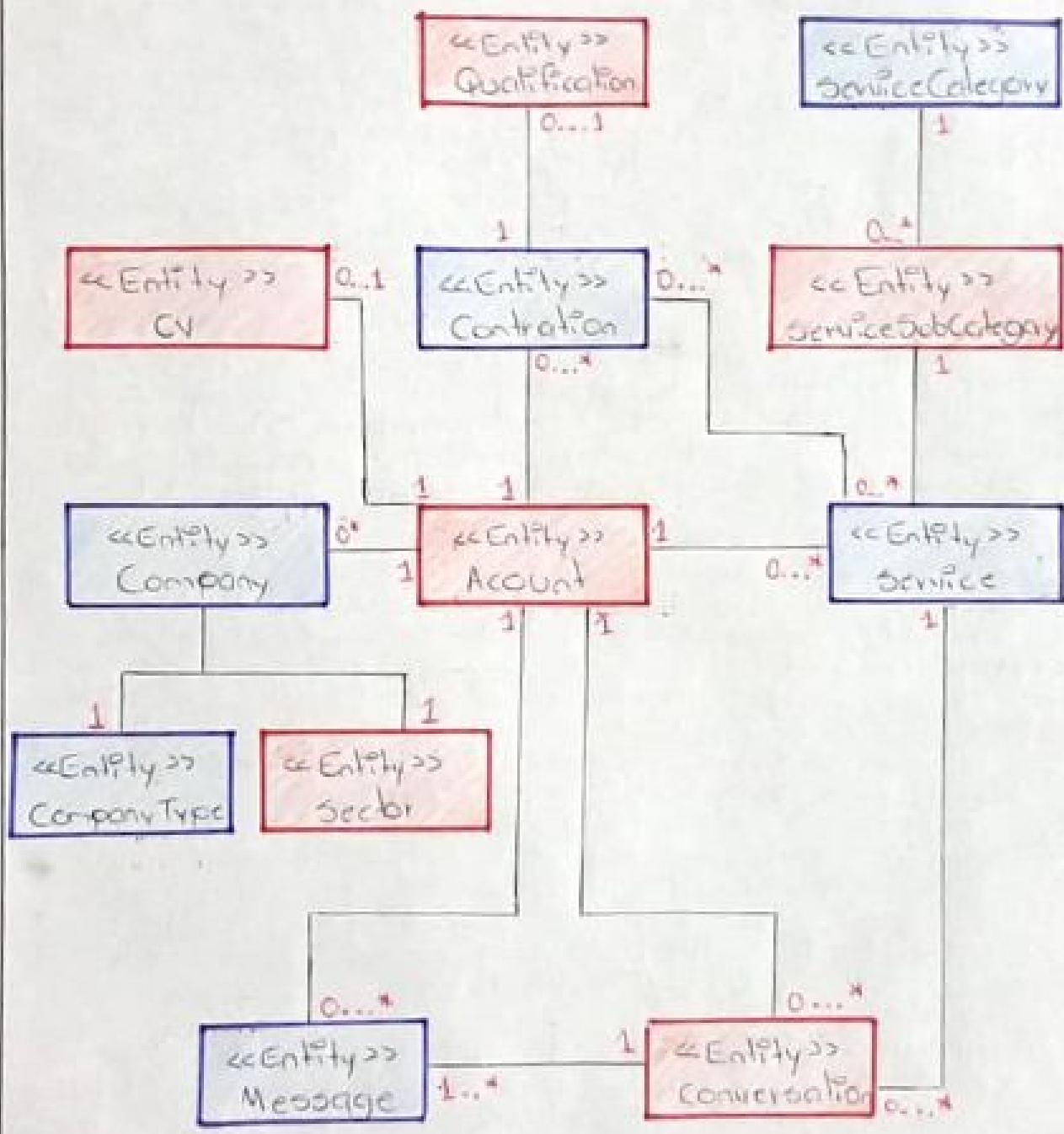
ARTICULO 25.

IMPLEMENTACIÓN DE UNA ARQUITECTURA DE SOFTWARE GUIADA POR EL DOMINIO

Resumen: la arquitectura hexagonal, también conocida como arquitectura de puertos y adaptadores, es un modelo estructural de software que promueve el aislamiento de la lógica de negocio del sistema respecto de sus interacciones externas (como bases de datos, interfaces de usuario y API). En esta arquitectura, el núcleo del negocio es protegido y no depende de tecnologías, sino, que interacciona con el exterior a través de "puertos" (interfaces y adaptadores) (implementaciones específicas de esas interfaces). Esto permite que el software mantenga su funcionalidad principal (aislada), aunque se cambien herramientas, bases de datos o interfaces de usuario externas. La arquitectura hexagonal es ideal para sistemas que requieren adaptabilidad y resistencia al cambio, permitiendo un desarrollo sostenible y modular.

Reflexión: En un ecosistema donde la tecnología cambia con rapidez, es crucial que el software evolucione su arquitectura sin volverse obsoleto. Una arquitectura hexagonal garantiza que la lógica de negocio se mantenga independiente de la forma en que las actualizaciones o cambios en herramientas, externos, como frameworks o motores de bases de datos, se implementan o funcionan. Esto permite a los equipos de desarrollo enfocarse en mejorar y optimizar la funcionalidad del negocio, en lugar de invertir tiempo en reestructuraciones complejas cada vez que un componente tecnológico queda desactualizado. A la larga, esto favorece un desarrollo más ágil y adaptable.

Bibliografía: Combariza, M., Díaz, F., & García Martínez N. (2020). Implementación de una arquitectura de software guiada por el dominio. In XLI Simposio Argentino de Ingeniería de Software (SAIS 2020) JNIO 49 (Modelo Virtual).



ARTICULO 26.

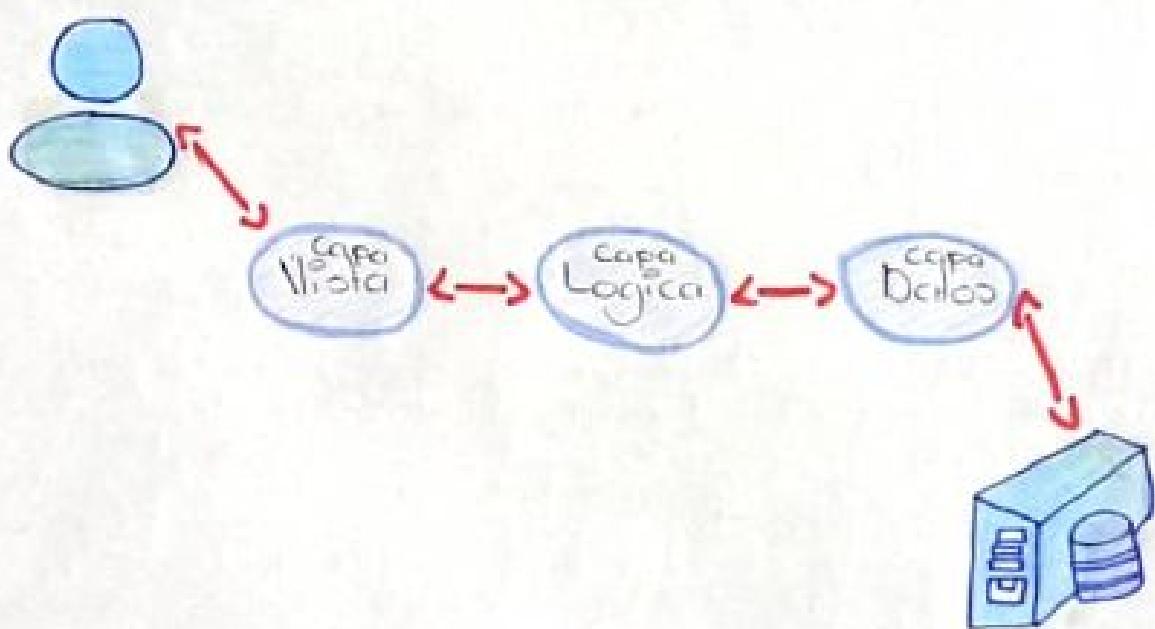
ARQUITECTURA DE SOFTWARE ACADÉMICA PARA LA COMPRESIÓN DEL DESARROLLO DE SOFTWARE EN CAPAS

Resumen: La arquitectura en capas es una estrategia ampliamente adoptada en el desarrollo de software debido a su capacidad para segmentar responsabilidades en componentes independientes. Cada capa tiene funciones específicas: la capa de presentación gestiona la interacción con los usuarios y la captura de datos, la capa lógica complementa las reglas de negocio, y la capa de datos se encarga de almacenar y recuperar información en estructuras persistentes. Este modelo permite una integración eficiente y reduce significativamente los efectos colaterales al realizar cambios, ya que cada capa interactúa únicamente mediante interfaces definidas, especialmente útil en sistemas que requieren flexibilidad mantenimiento sencillo y extensibilidad a lo largo del tiempo. El uso de este modelo asegura que los cambios en una capa no afecten drásticamente a las demás, promoviendo así la escalabilidad y la estabilidad del sistema.

Reflexión: La arquitectura en capas refleja un enfoque profesional para el diseño de sistemas priorizando la claridad, la modularidad y la separación de responsabilidades. En proyectos a largo plazo, permite incorporar nuevas funcionalidades sin comprometer la estabilidad existente. Sin embargo, su éxito depende de un diseño sólido y del cumplimiento riguroso de los principios de independencia entre capas. Adoptar esta arquitectura implica comprometerse con una metodología más estructurada, que a menudo requiere un esfuerzo adicional en las fases iniciales del proyecto, pero con grandes recompensas a futuro.

Bibliografía: Leydier, D. G. (2015) Arquitectura de software orientado para la compresión del diseño. No. de software en capas (No. 574). Serie documentos de trabajo.

Diagrama:



ARTÍGULO 27.

DESARROLLO DE UNA ARQUITECTURA DE SOFTWARE PARA EL ROBOT MÓVIL LÁZARO

Resumen: el robot Lázaro, diseñado para terrenos inexplorados, implementa una arquitectura de software basada en tres niveles principales:

1. Nivel Básico: responsable de gestionar sensores y actuadores a bajo nivel, operando ya procesos secuenciales o multiproceso.

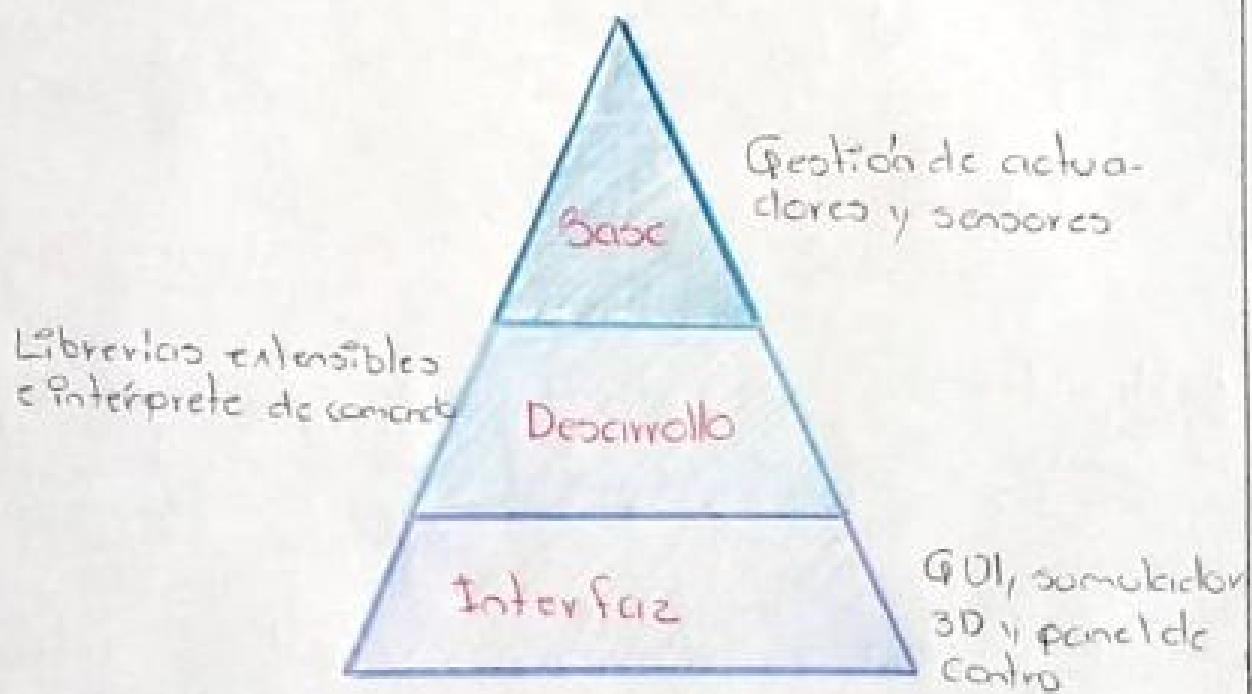
2. Nivel de Desarrollo: permite extender capacidades del robot mediante nuevos librerías y componentes programados, incluyendo un intérprete de comandos para facilitar la integración de algoritmos.

3. Nivel de Interfaz: brinda al usuario una GUI con simulador 3D, panel de control directo, y herramientas para programar y monitorear el comportamiento del robot.

Reflexión: este arquitectura resalta como un diseño modular y escalable facilita la integración entre el hardware y software, adaptándose a las necesidades cambiantes de los usuarios. Además, la inclusión de herramientas de simulación y programación accesibles mejora la eficiencia en la operación y experimentación del robot. Sin embargo, la dependencia de un ordenador remoto y costosos módulos inalámbricos podría representar limitaciones en entornos más complejos o autónomos.

Bibliografía: Gómez, J. M., Gil, A. C. & Sánchez, E. A. (2018) Desarrollo de una arquitectura de software para el robot móvil Lázaro. Ingeniería y Tecnología Chilena de la Ingeniería, 26(3), 376-390

Diagrama:



ARTÍGULO 28.

LINEAMIENTOS PARA EL DISEÑO DE APLICACIONES WEB SOORTADAS EN PATRONES

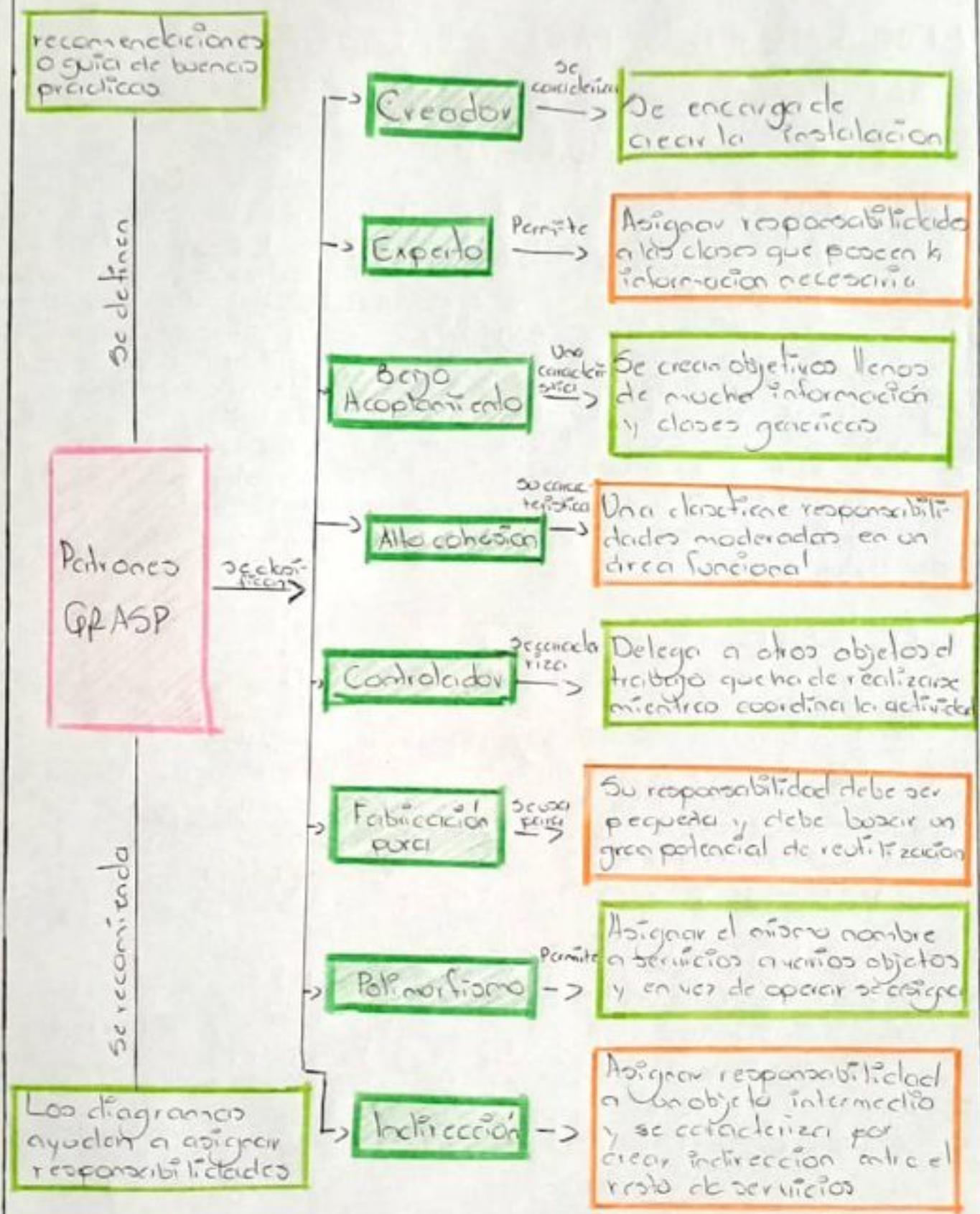
GRASP

Resumen: el artículo destaca la importancia de los patrones de diseño GRASP (General Responsibility Assignment Software Patterns) como herramienta para mejorar el diseño y desarrollo de aplicaciones web orientadas a objetos. Estos patrones ayudan a asignar responsabilidades en el diseño promoviendo buenas prácticas como alta cohesión y bajo acoplamiento. Además, se enfoca en el aprendizaje de conceptos fundamentales del análisis, diseño y programación orientada a objetos (POO) permitiendo a los estudiantes de ingeniería desarrollar aplicaciones flexibles, reutilizables y con un vocabulario común.

Reflexión: los patrones GRASP ofrecen un marco conceptual sólido para diseñar software orientado a objetos. Al enfocarse en asignar responsabilidades claramente claras, no solo se mejora la calidad del código, sino que también se facilita el mantenimiento y la evolución de las aplicaciones. Sin embargo, su implementación requiere una comprensión adecuada de los fundamentos POO, lo que implica una curva de aprendizaje significativa para los estudiantes.

Bibliografía: Ortega, G. A. M. (2021). Lineamientos para el diseño de aplicaciones web soportadas en patrones GRASP. *Ciencia e Ingeniería: Ensayos de Investigación Interdisciplinaria en Biotecnología y Desarrollo Sostenible, Ciencia, Tecnología e Innovación y Procesos Productivos Industriales*.

Diagramas:



ARTICULO 29.

LA ARQUITECTURA DE SOFTWARE EN EL PROCESO DE DESARROLLO: INTEGRANDO MDA AL CICLO DE VIDA EN ESPIRAL

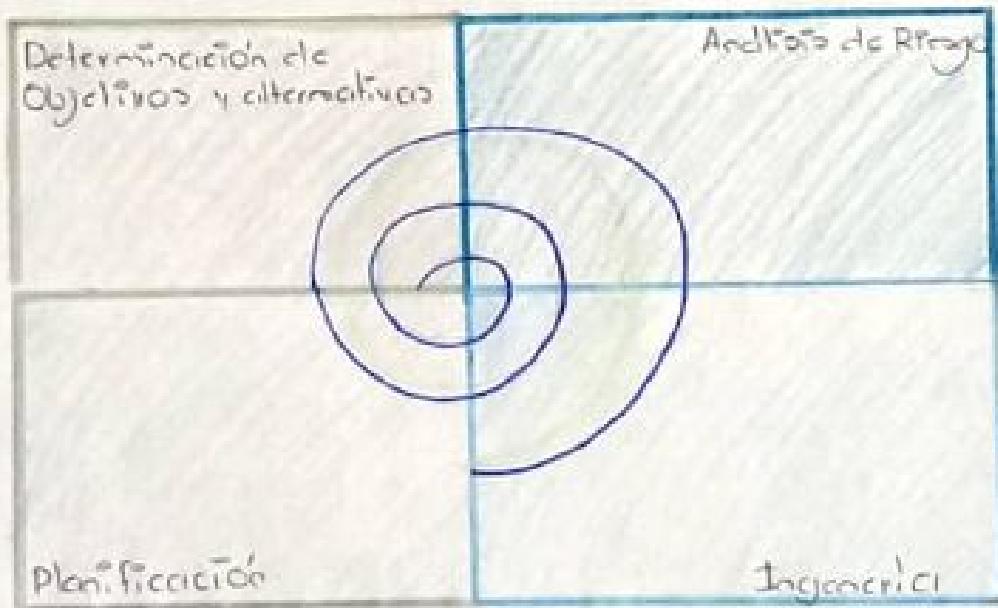
Resumen: el artículo explora la integración de la arquitectura dirigida por modelos (MDA, Model Driven Architecture) al ciclo de vida en espiral propuesto por Boehm abordando su impacto en los distintos pasos del desarrollo de software. MDA se basa en transformar modelos de alto nivel, como los CIM (Modelos Independientes del Componente) y DIM (modelos Independientes de la Plataforma), en modelos específicos de implementación (PSM) y en código fuente mediante transformaciones autorretroactivas. Este enfoque fortalece la trazabilidad, la separación de responsabilidades y la portabilidad del software. Por otro lado, el ciclo en espiral prioriza la gestión de riesgo en un proceso iterativo, como fases de planificación, análisis de riesgo, ingeniería y evaluación del cliente. La propuesta del artículo combina ambos enfoques permitiendo que MDA optimice la trazabilidad y minimice los riesgos asociados a cambios tecnológicos y de requisitos.

Reflexión: La fusión de MDA y el ciclo de vida en espiral representa una solución robusta para abordar el creciente complejidad del desarrollo de software. Al enfocarse en la trazabilidad y el análisis iterativo, se mejora la calidad y la adaptabilidad del producto final. Sin embargo, implementar estos métodos logísticos puede resultar desafiante debido a la necesidad de herramientas específicas y habilidades técnicas avanzadas.

Bibliografía: Menurio, M. S. & Schenckler, E. (2013)

La arquitectura de software en el proceso de desarrollo: integrando MDA al ciclo de vida en espíritu Archivo de la Revista Latinoamericana de Ingeniería de Software 4(4) 312 - 346

Diagrama:



ARTÍGULO 30.

PLANOS ARQUITECTÓNICOS: EL MODELO DE "4+1" VISTAS DE LA ARQUITECTURA DEL SOFTWARE.

- Resumen:** el modelo "4+1" no presenta vistas independientes, sino que estas están interconectadas para garantizar la coherencia entre los aspectos funcionales y no funcionales del sistema:
- * De la vista lógica a la de procesos: las clases metálico identificadas en la lógica se mapean en procesos, garantizando concurrencia adecuada.
 - * De proceso a físico: los procesos se asignan a nodos físicos, asegurando un rendimiento óptimo y tolerancia a fallos.
 - * De lógica a desarrollo: las clases se agrupan en subsistemas y capas según los reglas de diseño, como minimizar dependencias entre módulos.

Reflexión: los escenarios son una herramienta clave en este modelo porque integran todas las vistas y permiten validar que la arquitectura cumpla con los requisitos funcionales y no funcionales. Sirven como pruebas iniciales de las decisiones de diseño y como guía para identificar posibles inconsistencias o áreas de mejora. Además, los escenarios promueven la colaboración entre stakeholders y que reflejan situaciones reales de uso lo que fomenta un entendimiento común del sistema. En proyectos grandes, los escenarios sirven para resaltar el abordar elementos críticos desde los primeros frisos del diseño.

Bibliografía: Fruchter, P. (1995) Plurales arquitectónicos: el modelo UML visto de la arquitectura del software. IEEE Software, 12 (6), 42-50.

Nodo Físico	Procesos Asignados	Conexiones
Disco	Pro	Comunicación con
Servicio	Proceso	Con
C.	Pag	Comun

ARTICULO 31.

INTRODUCCIÓN A LA REINGENIERÍA DE SOFTWARE MEDIANTE PATRONES DE DISEÑO

Resumen: El artículo explora como la reingeniería puede transformar sistemas heredados en software moderno y funcional. Explica que el comienzo con la ingeniería inversa para recuperar diseño y datos relevantes. Luego, el análisis y diseño identifican las necesidades del usuario y los ajustes necesarios en el sistema. Finalmente, la ingeniería avanzada implementa cambios eficiente la modularización y pruebas para garantizar calidad y mantenibilidad. Se destacan los heredamientos Columbus y Mojarra para automatizar y mejorar este proceso. Columbus convierte el código fuente en diagramas UML, mientras que Mojarra aplica máscaras de patrones para identificar y optimizar el diseño existente.

Reflexión: El uso de patrones en la reingeniería evidencia la importancia de construir software utilizable y bien estructurado. Identificar y aplicar estos patrones en sistemas heredados permite mantener consistencia en el desarrollo. Además, los refacturings juegan un papel crucial al preservar el comportamiento del software original mientras se realizan mejoras. Esto muestra que la reingeniería no solo soluciona problemas técnicos, sino que también promueve una cultura de mejoramiento en la ingeniería de software.

Bibliografía: Contreras, T. B. Introducción al
Engineering de Software Mediante
Patrones de Diseño.

Colombus: extrae modelos de clase
en UML desde el código
fuente de C++

Maia: analiza diagramas UML
para encontrar patrones
mediante restricciones de roles

Routledge: obtención de patrones re-
utilizables que optimizan diseño
y funcionalidad

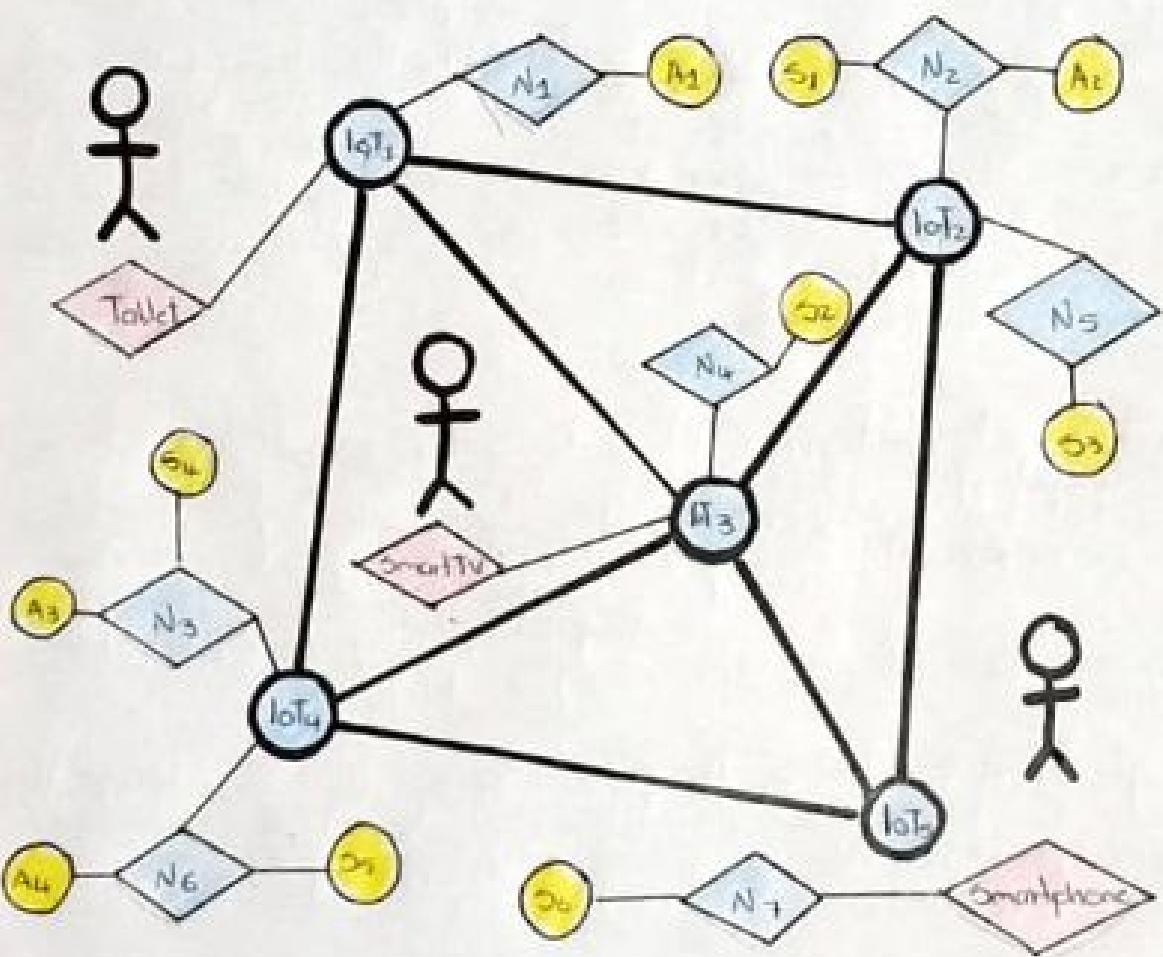
ARTICULO 32.

USO DE PATRONES DE DISEÑO Y METAPROGRAMACIÓN PARA CONSTRUIR APIs DE IoT USANDO C++

Resumen: El trabajo analiza la implementación de una API para la capa de percepción de sistemas IoT. Se emplearon patrones como programación orientada a objetos y metaprogramación para optimizar el rendimiento y facilitar el desarrollo de aplicaciones. Se realizaron tres iteraciones que incluyeron prototícos básicos hasta el uso crudo de templates. El uso resultado es una API eficiente, modular y capaz de adaptarse a diversos entornos, manteniendo la aplicabilidad de otras técnicas en sistemas embedidos.

Reflexión: Si enfocar el enfoque práctico y participativo de la investigación -acción fue clave para el éxito del proyecto. Este método permite identificar problemas prácticos y buscar soluciones en el campo real evitando así la importancia de teorizar la investigación y práctico en el desarrollo de tecnología IoT.

Bibliográfico: Peñache A, Escobar J, & Trujillo E. Uso de patrones de diseño y metaprogramación para construir APIs de IoT usando C++.



APÉNDICULO 33.

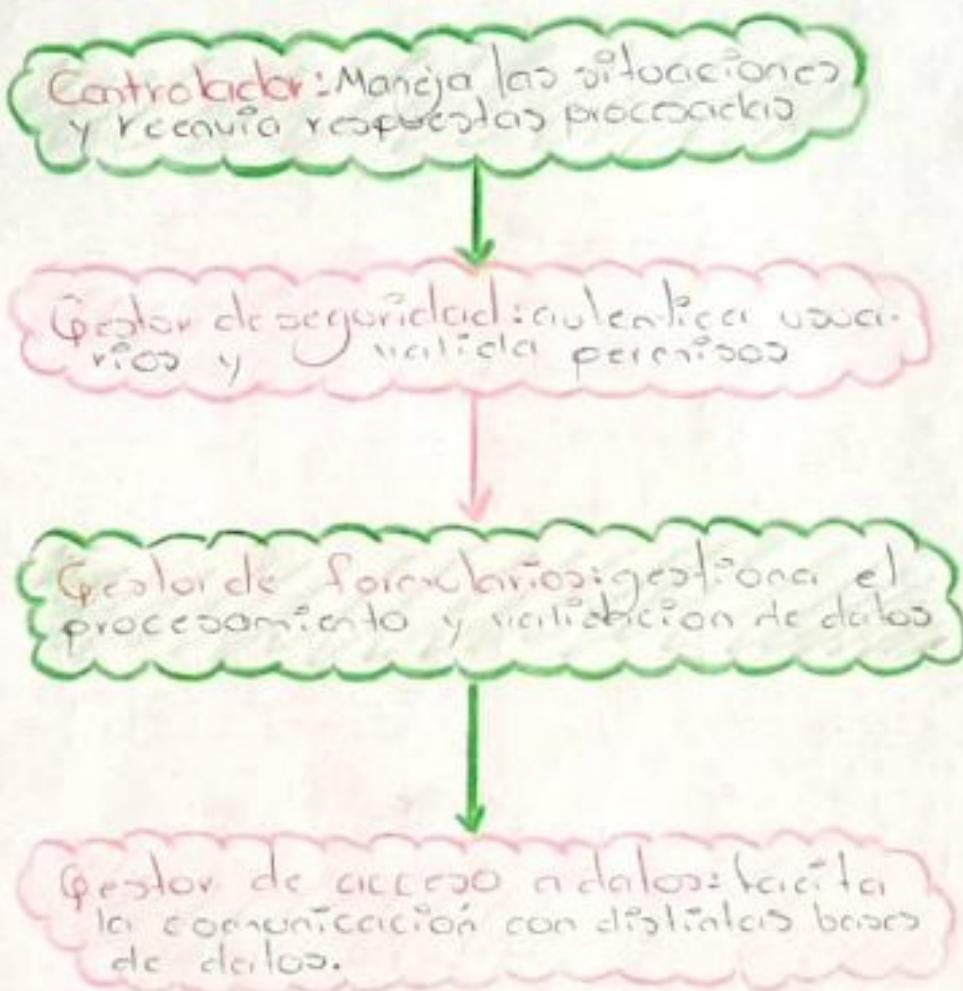
DISEÑO DE FRAMEWORK WEB PARA EL DESARROLLO DINÁMICO DE APLICACIONES

Resumen: el framework propuesto sigue un arquitectura cliente-servidor distribuida para, ejecutarse tanto en intranet como en entornos de internet, utilizando HTTP como protocolo base. Incorpora componentes como: un controlador control para gestionar subsistemas, un gestor de seguridad que valida roles y permisos, un gestor de formularios para operaciones CRUD y gestor de acceso a los datos que abstracta la interacción con diversas bases de datos. Se utilizan tecnologías como Apache PHP, MySQL y librerías como MongoDB para garantizar escalabilidad, rendimiento y seguridad. Además se emplearon herramientas como Kendo Grid y Subversive para facilitar el desarrollo colaborativo y la gestión de versiones. El resultado es un entorno que permite la personalización y adaptación a diferentes proyectos.

Reflexión: la capacidad de reutilizar componentes a través de un framework no solo optimiza los recursos, sino que también contribuye al mantenimiento y evolución de proyectos. Esto, permite abordar problemas comunes como soluciones probadas, lo que incrementa la calidad y la seguridad del software. Al concentrarse en la redacción modular, los frameworks se convierten en herramientas esenciales para garantizar la escalabilidad y compatibilidad de los sistemas, respondiendo a los demandas de un mercado cada vez más competitivo.

Bibliografía: Villobos, C. M., Sanchez, G. D. C. & Gutierrez, D. A. B. (2010). Diseño de framework web para el desarrollo dinámico de aplicaciones. *Scientia et Technica*, 16(44), 178-183.

Diagramas:



ARTÍCULO 34.

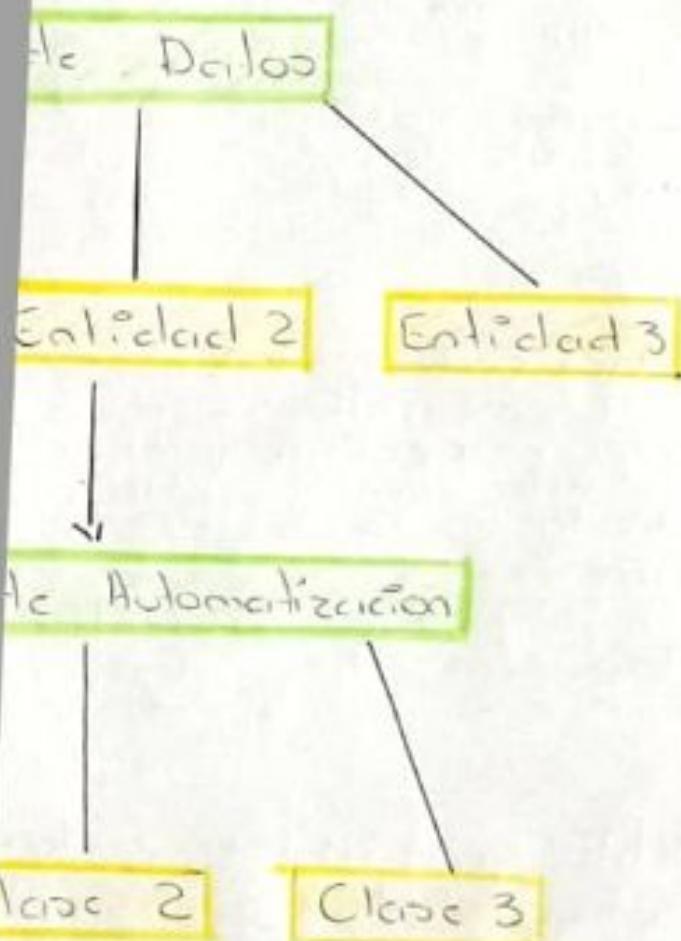
CONSTRUCCIÓN DE UNA LIBRERÍA PARA GENERACIÓN AUTOMÁTICA DE CLASES

PHP

Resumen: el artículo aborda la construcción de una librería para la generación automática de clases PHP basada en patrones de diseño utilizando el proceso personal de software CSD. El objetivo principal es optimizar el diseño de entidades de bases de datos hacia clases en PHP para mejorar la eficiencia, prevenir errores humanos y garantizar la estandarización del código. Esto facilita la integración con tecnologías orientadas a objetos como los ORMs, optimizando el manejo de datos y permitiendo un enfoque en reglas de negocio.

Reflexión: este enfoque automatizado es esencial en el desarrollo moderno ya que reduce la carga respectiva y el riesgo de errores. Además, suministra prácticas de codificación robustas y escalables. Sin embargo, el éxito de herramientas como esta depende de su capacidad para adaptarse a proyectos diversos y de mantener actualizadas otras, algo que el uso de patrones de diseño ayuda a lograr.

Bibliografía: Torres Morcillo, J.L. (2014). Construcción de una librería para generación automática de clases PHP basada en patrones de diseño utilizando PDP.



ARTICULO 35.

ESTILOS Y PATRONES EN LA ESTRATEGIA DE ARQUITECTURA DE MICROSOFT

Resumen: el documento explora los estilos arquitectónicos en software y su importancia en la estrategia de Microsoft. Los estilos arquitectónicos son estructuras de alto nivel que describen la organización de sistemas mediante componentes, relaciones y restricciones. Ejemplos incluyen arquitecturas en capas, cliente-servidor, tubería-filtro y MEC. Estos estilos permiten utilizar patrones y optimizar procesos de desarrollo, vinculando teoría con práctica y adaptándose a distintas necesidades.

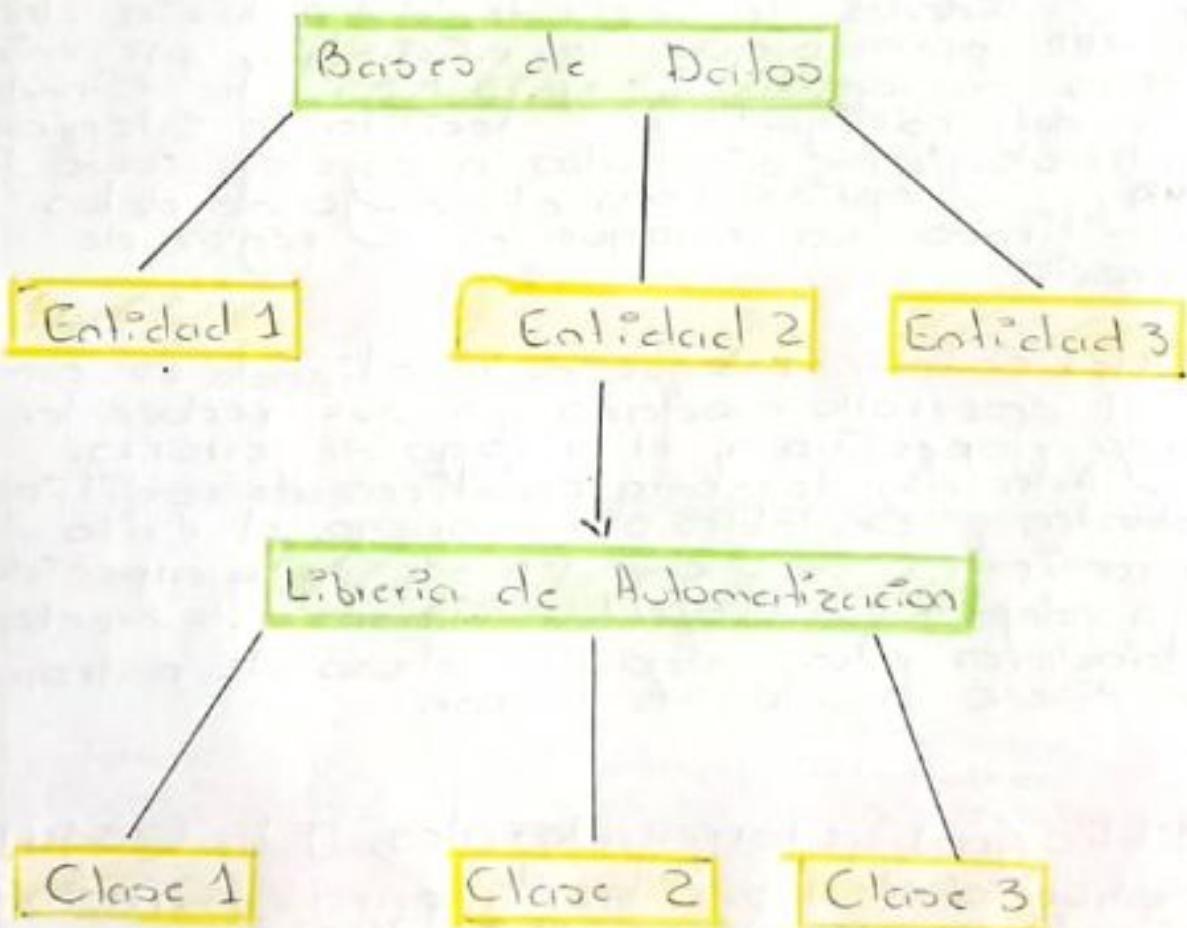
Reflexión: los estilos arquitectónicos representan un marco esencial para estructurar sistemas, fortaleciendo claridad, reutilización y mantenimiento. Este enfoque abstracto sirve como puente entre teoría y práctica, ayudando a arquitectos y desarrolladores a diseñar sistemas escalables y robustos. Sin embargo, su elección requiere un análisis detallado del problema y del contexto tecnológico, ya que cada estilo tiene ventajas y limitaciones específicas.

Bibliografía: Reynoso, C & Kicillof, N (2004) Estilos y Patrones en la estrategia de arquitectura de Microsoft.

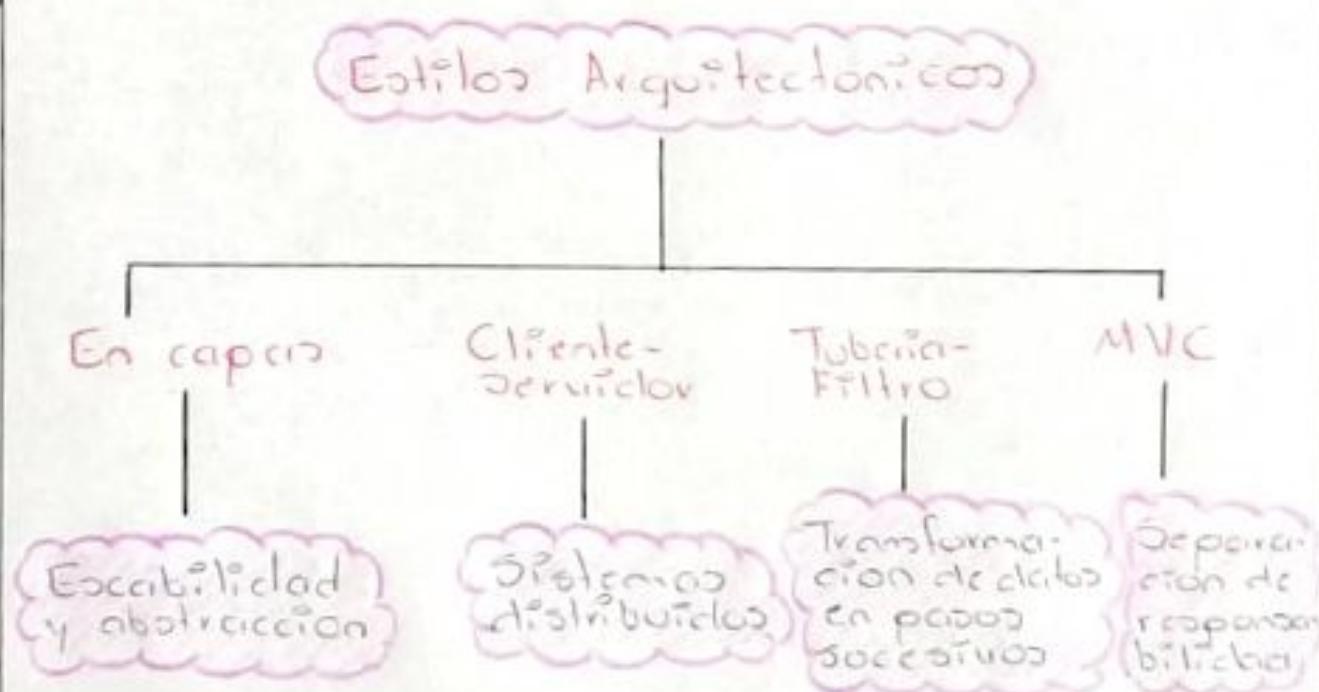
ARQUITECTURA

Arquitectura para un sistema de
gestión de información en la base de datos

Diagrama:



Diagramas:



ARTICULO 36

PROPIUESTA DE ARQUITECTURA PARA EL PRODUCTO "MIS MEJORES CUENTOS"

Resumen: el artículo presenta la propuesta de una arquitectura de software para el producto educativo "Mis mejores cuentos" diseñado para digitalizar cuentos infantiles tradicionales. Su objetivo principal es proporcionar una solución eficiente para niños de 3 a 5 años, optimizando atributos como integrabilidad, rendibilidad y portabilidad mediante el uso de herramientas libres y patrones arquitectónicos modernos de incluye la selección de tecnologías, patrones como MVC y una estructura modular basada en capas.

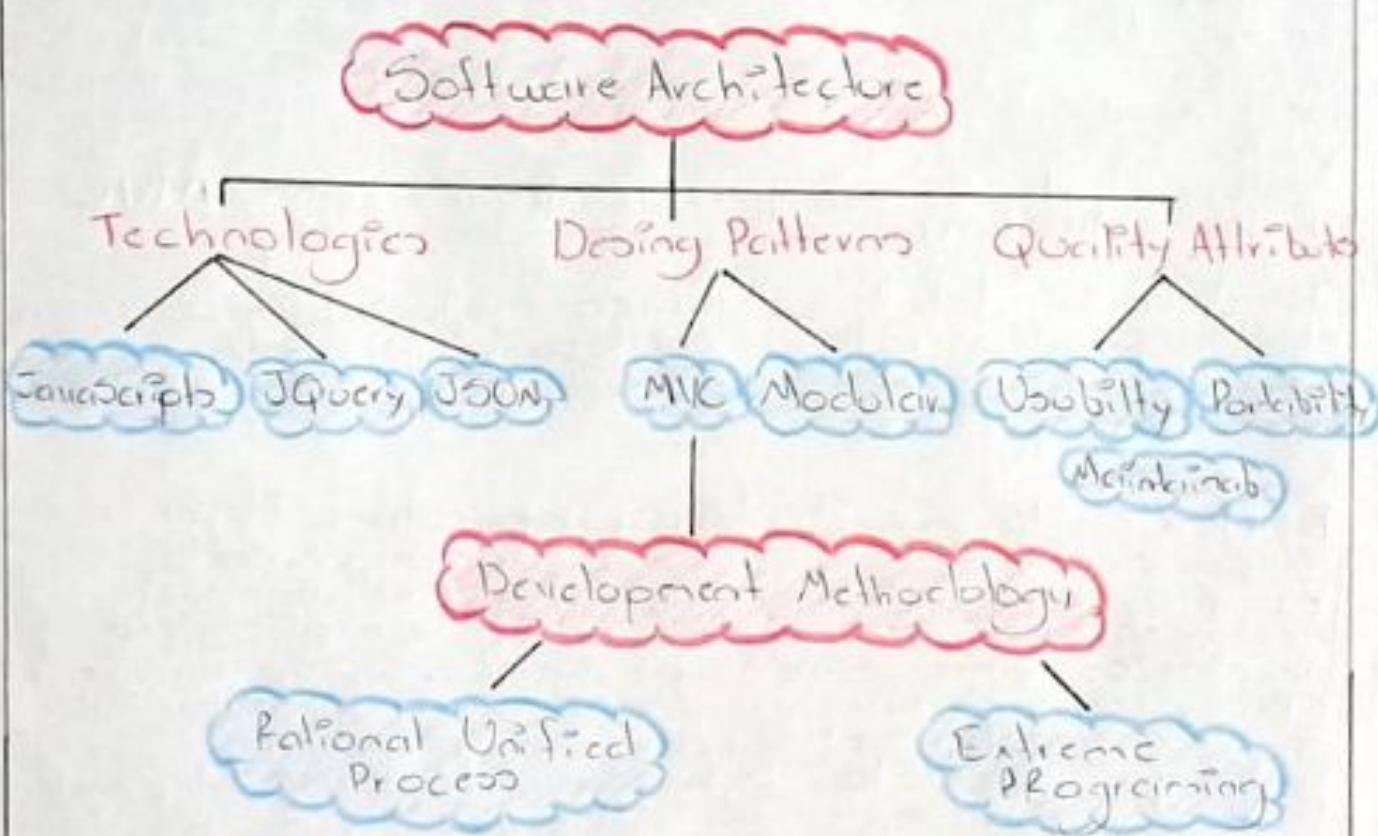
También se detalla un marco metodológico para el diseño, desarrollo y evaluación de esta arquitectura, con énfasis en la calidad y la usabilidad del proyecto.

Reflexión: la arquitectura propuesta refleja la importancia de un diseño bien estructurado en proyectos educativos, al especialmente al considerar la accesibilidad y simplicidad necesarios para niños pequeños. Su herramienta libre y establecida ciertos destaca la función de democratizar el acceso al aprendizaje. Sin embargo, el éxito de la arquitectura depende de la implementación adecuada y de mantener un equilibrio entre los recursos tecnológicos y las necesidades del usuario final.

Bibliografía: Jiménez, L. B. (2011) Título:

Propuesta de arquitectura de software para el producto "Mis mejores cuentos" (Doctoral Dissertation. Universidad de las Ciencias Informáticas)

Diagrams:



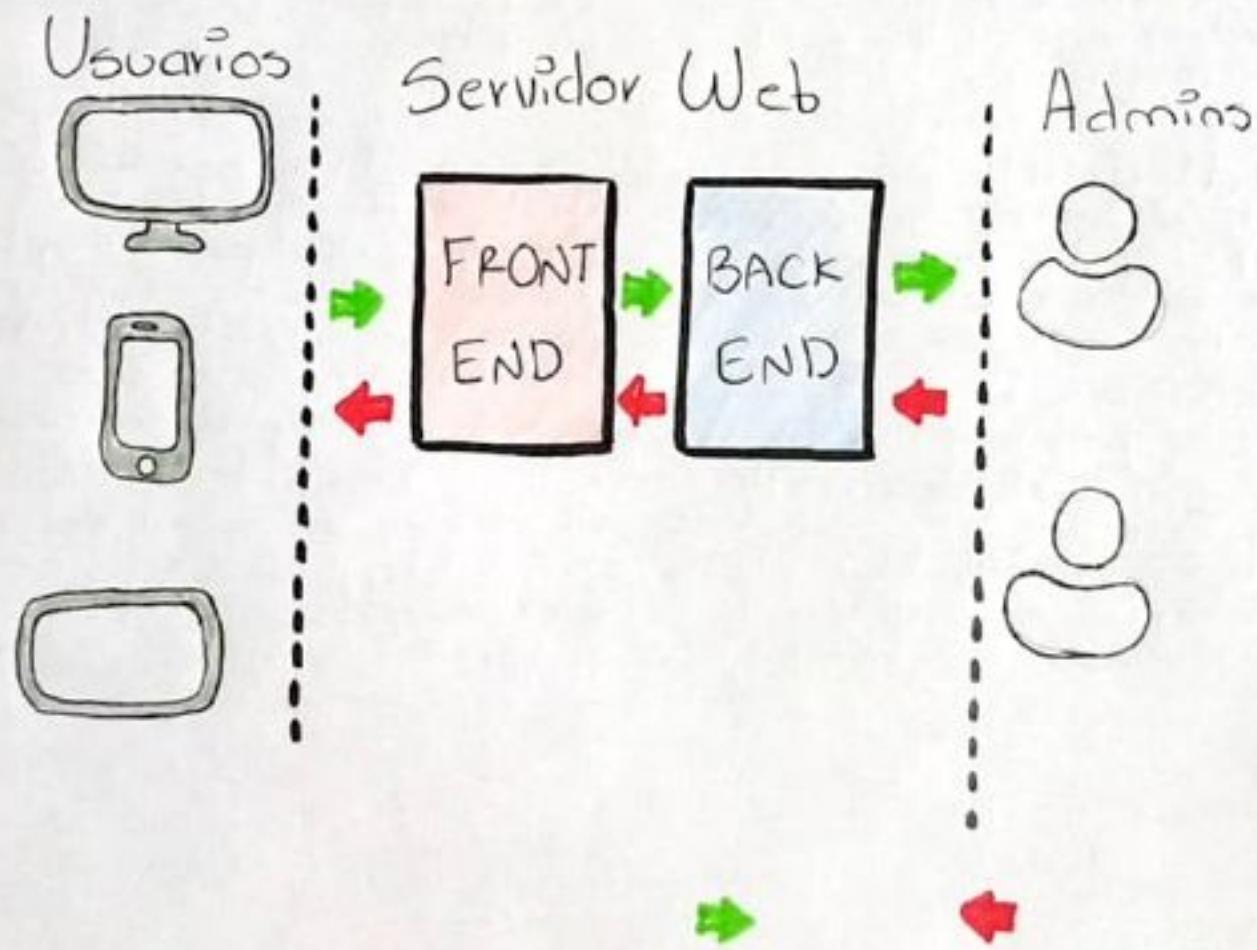
ARTICULO 37.

ARQUITECTURA DE PROGRAMACIÓN WEB: BACKEND

Resumen: el desarrollo es esencial para crear aplicaciones web dinámicas que permiten interactividad y gestión eficiente de recursos en el servidor. Este desarrollo se centra en las tareas que no son visibles para el usuario final, como la interacción con bases de datos y ejecución lógica de negocio. El texto detalla la ejecución de lenguajes, como PHP, Python, Ruby y Java así como frameworks destacados como Laravel, Django y Spring. También se exploran novedades tendenciales, como la programación para el Internet de las cosas (IoT), el uso de contenedores (Docker) y la implementación de microservicios.

Reflexión: la programación backend representa el núcleo de la funcionalidad de cualquier aplicación web moderna. Su capacidad para adaptarse a los rápidos cambios tecnológicos es crucial para facilitar las necesidades crecientes del usuario y empresa. La tendencia hacia el uso de lenguajes más eficientes y tecnologías emergentes como el desarrollo servido o back-end practices que en un futuro en el que backend será más modular, flexible y escalable. Además, el enfoque en mejorar la experiencia mediante interfaces nativas y servicios distribuidos.

Bibliografía: Montañez, Sánchez, M. (2020). Arquitectura de programación web: Backend



ARTÍCULO 38 •

INTEGRANDO LA INGENIERÍA DE SEGURIDAD EN UN PROCESO DE INGENIERÍA SOFTWARE

Resumen: el artículo analiza cómo las prácticas de ingeniería de seguridad no suelen integrarse adecuadamente en el proceso de desarrollo de software, lo que resulta en sistemas menos seguros. Propone una metodología para incorporar la seguridad desde los etapas iniciales del desarrollo, utilizando extensiones del lenguaje UML extendido de Modelado (UML) y herramientas automatizadas. Este enfoque busca garantizar que los modelos de seguridad sean consistentes a lo largo de ciclo de vida del software y promover el uso de patrones de seguridad que sean consistentes a lo largo de vida del software y promover el uso de patrones de seguridad para resolver problemas recurrentes.

Reflexión: el trabajo destaca la importancia de abordar la seguridad como un componente esencial del desarrollo de software, no como un complemento. Este enfoque mejora la confiabilidad del sistema y ayuda a reducir la desconfianza de los usuarios hacia las tecnologías actuales. Sin embargo, la implementación de esta metodología requiere un cambio actual en la industria y un esfuerzo significativo para capacitar a los desarrolladores en técnicas de seguridad, así como en el uso de herramientas avanzadas como UML extendido.

Bibliografía: Mañó, A., Rey, D., Sanchez F & Vingue M. I (2004). Integrando la ingeniería de seguridad en un proceso de ingeniería. VIII Reunión Española de Criptología y Seguridad de la Información RECSI 4, 383-392

Diagrama:

