Artículos

Carolina Martinez Cortes

SENA – Servicio Nacional de Aprendizaje

Centro de la Industria la Empresa y los Servicios

Análisis y Desarrollo de Software

Ficha: 2694667

Neiva / Huila

Tabla de Contenido

Biografía

Soy Carolina Martínez Cortes, estudiante del Tecnólogo en Análisis y Desarrollo de Software en el Servicio Nacional de Aprendizaje (SENA). A lo largo de mi formación, he trabajado en proyectos que integran el desarrollo de aplicaciones móviles y sistemas backend, utilizando tecnologías como Java, Kotlin, Spring Boot y Android Studio. Me especializo en el análisis y desarrollo de software, con habilidades en la creación de interfaces intuitivas, gestión de bases de datos y programación orientada a objetos.

Además, he cultivado competencias en organización, responsabilidad y liderazgo, lo que me permite trabajar de manera efectiva en equipo y asumir desafíos con confianza. Mi objetivo es seguir aprendiendo y aportar soluciones tecnológicas que impacten de manera positiva en el entorno profesional y social.

Artículo 01: Revisión sistemática sobre generadores de código fuente y patrones de arquitectura.

Resumen

El documento es una revisión sistemática sobre Generadores de Código Fuente (GCF) y patrones de arquitectura en el desarrollo de software, especialmente orientado a aplicaciones web. La autora busca analizar la utilidad de los GCF como herramientas que automatizan la creación de código, permitiendo a los desarrolladores reducir tiempos, costos y errores. Estas herramientas generan automáticamente el código para operaciones comunes, como interfaces de usuario y conexiones a bases de datos, que generalmente comparten una lógica común. Al automatizar estas tareas repetitivas, los GCF permiten que los desarrolladores se enfoquen en los aspectos específicos y más complejos de sus aplicaciones. La investigación también enfatiza la importancia de los patrones de arquitectura, que establecen estructuras para organizar el software de manera estandarizada, favoreciendo su mantenimiento, escalabilidad y calidad a largo plazo. Los patrones arquitectónicos, como MVC (Modelo-Vista-Controlador) o MVVM (Modelo-Vista-ViewModel), encapsulan las relaciones y responsabilidades de cada componente del software. Esto permite que los desarrolladores no solo mantengan una coherencia y organización en el código, sino que también puedan reutilizar partes significativas del

sistema y asegurar que el software sea fácilmente adaptable a nuevos requerimientos o tecnologías. A través de un análisis exhaustivo de la literatura y de estudios previos, la revisión identifica las herramientas y frameworks más empleados en los GCF, así como los lenguajes de programación y bases de datos más comunes en este contexto. La metodología de investigación se basa en un proceso de revisión sistemática que abarca la planificación, ejecución, selección y evaluación de estudios primarios, con el fin de reunir evidencia de las mejores prácticas y tendencias actuales en el uso de GCF y patrones de arquitectura.

Reflexión

En un mundo donde la tecnología cambia a toda velocidad, cada vez se necesitan aplicaciones más rápidas y sofisticadas. En este contexto, los Generadores de Código Fuente (GCF) y los patrones de arquitectura se han vuelto herramientas esenciales para desarrolladores y arquitectos, ya que no solo aceleran el proceso de desarrollo, sino que también mejoran la calidad de los productos. Elegir las herramientas y patrones correctos permite cumplir con lo que el cliente espera ahora, pero también hace más sencillo mantener y actualizar el software cuando surgen nuevas necesidades. Este estudio destaca lo importante que es mantenerse actualizado con estas herramientas, ya que tomar decisiones bien informadas puede marcar 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* (Master's thesis, Pontificia Universidad Catolica del Peru (Peru)).

https://www.proquest.com/openview/03dd8f10be19360f9fbd034696c5530a/1?pq-origsite=gscholar&cbl=18750&diss=y

Diagrama

Artículo 02: Una teoría para el diseño de software.

Resumen

Este documento presenta una teoría sobre el diseño de software, destacando su papel esencial 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. Invertir en un diseño estructurado reduce significativamente los costos de mantenimiento, que pueden representar hasta el 67% de los costos del proyecto.

La teoría introduce el concepto de "Diseño para el Cambio," que busca anticiparse a 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. Históricamente, el diseño de software ha evolucionado de una etapa opcional a una práctica crucial, que organiza el desarrollo, facilita la adaptación y asegura la calidad del sistema a largo plazo.

Reflexión

Esta teoría invita a 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 reestructurar completamente el sistema. En lugar 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. El principio de "Diseño para el Cambio" es particularmente relevante en un contexto de rápida evolución tecnológica, donde las demandas de software cambian constantemente. La teoría también destaca la importancia de integrar reglas de cálculo en el diseño para que las decisiones estructurales sean consistentes y reproducibles.

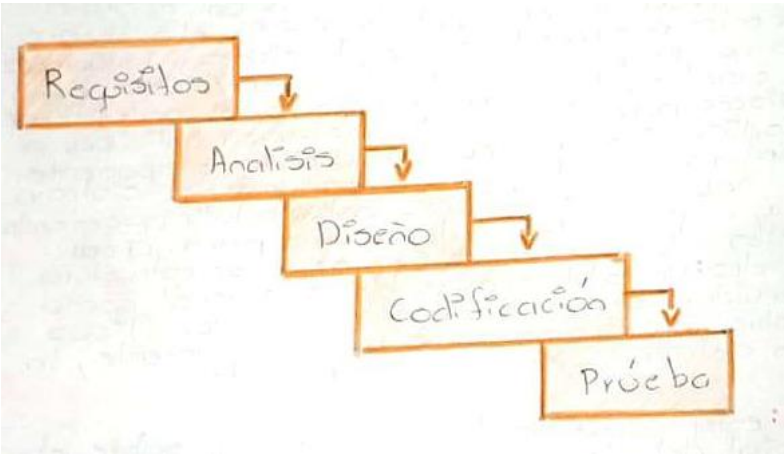
Un enfoque metodológico y calculado en el diseño proporciona una guía que equilibra la creatividad con la estandarización. Además, el uso de estilos arquitectónicos y patrones de diseño genera un lenguaje común entre los desarrolladores, facilitando el trabajo en equipo y reduciendo los problemas de integración y comunicación. En última instancia, esta teoría subraya que un diseño adecuado no solo mejora la calidad del software y la experiencia del usuario, sino que también protege la inversión en tecnología, al crear sistemas que puedan evolucionar sin necesidad de ser completamente reconstruidos.

Bibliografía

Cristiá, M. (2021). Una Teoría para el Diseño de Software.

https://www.fceia.unr.edu.ar/ingsoft/intro-diseno.pdf

Diagrama



Artículo 03: Perfiles UML para definición de Patrones de Diseño.

Resumen

El documento explora cómo los perfiles UML pueden extender el uso de UML (Lenguaje Unificado de Modelado) para mejorar la definición y visualización de patrones de diseño. Los patrones de diseño son soluciones reutilizables en el desarrollo de software orientado a objetos, y el uso de UML ayuda a modelar estas soluciones. Sin embargo, los

autores destacan que UML, en su forma estándar, a menudo carece de expresividad para representar patrones de diseño de manera precisa.

Los perfiles UML se proponen como una solución para superar estas limitaciones, ya que permiten ampliar la sintaxis y semántica de UML mediante estereotipos, etiquetas y restricciones. Esto no solo permite personalizar el lenguaje para dominios específicos, sino también para modelar patrones comunes en diferentes dominios. Los autores ejemplifican este enfoque mediante el patrón “Composite” de diseño estructural, definiendo estereotipos como “Component,” “Leaf,” y “Composite” y usando restricciones en OCL (Object Constraint Language) para especificar relaciones entre estos elementos.

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 estandarización de soluciones de diseño. Los autores concluyen que esta técnica es efectiva y sugieren una futura investigación en la evaluación de herramientas UML que soporten perfiles y permitan una mejor especificación de patrones.

Reflexión

Usar perfiles UML para definir patrones de diseño es una forma innovadora y práctica de superar las limitaciones que tiene UML para modelar ciertos aspectos del software. Con este enfoque, UML se vuelve más expresivo y flexible, permitiendo que se adapte a las necesidades específicas de cada proyecto y ayudando 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 que finalmente se diseña.

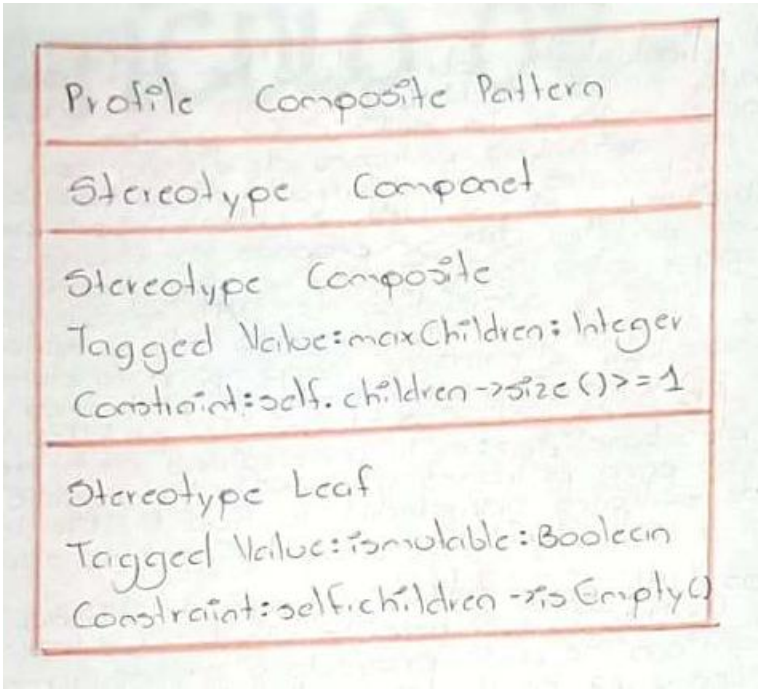
Este enfoque lleva el modelado de software un paso más allá, haciéndolo menos rígido y más adaptable a los requerimientos particulares de cada proyecto. No solo ayuda a ahorrar tiempo, sino que también facilita el trabajo en equipo, ya que permite crear un vocabulario visual común y específico que mejora la comunicación entre los miembros. En última instancia, este tipo de solución impulsa una forma de trabajo más dinámica y colaborativa, donde los desarrolladores pueden personalizar UML para que realmente se ajuste a las necesidades de sus patrones de diseño.

Bibliografía

Garis, A. G., Riesco, D. E., & Montejano, G. A. (2006). Perfiles UML para definición de Patrones de Diseño. In *VIII Workshop de Investigadores en Ciencias de la Computación*.

https://sedici.unlp.edu.ar/bitstream/handle/10915/20784/Documento_completo.pdf?sequence=1&isAllowed=y

Diagrama



Artículo 04: Arquitectura 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 Francisco de Paula Santander. Este software está diseñado para apoyar el aprendizaje autónomo de los estudiantes, permitiéndoles gestionar sus conocimientos sobre 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 estudiantes ver cómo funcionan las actividades de obra en un entorno simulado, facilitando la comprensión del impacto de cada decisión financiera y logística en un proyecto de construcción.

Reflexión

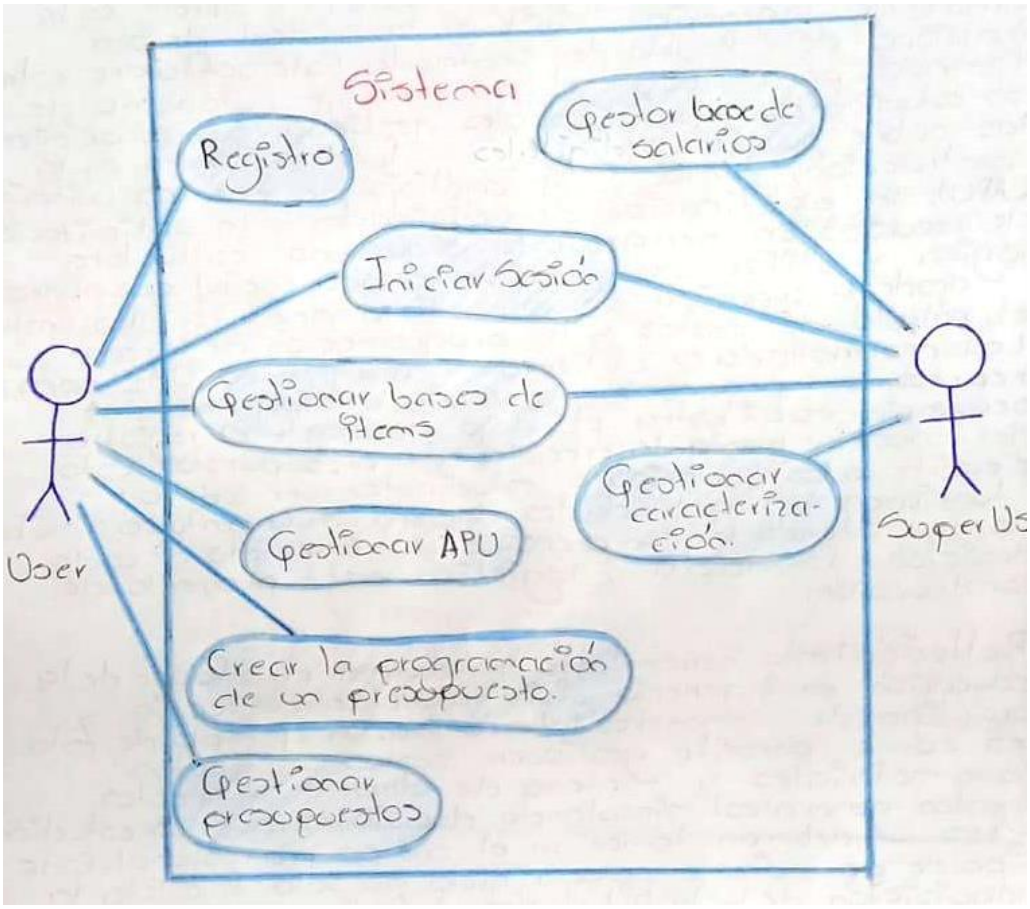
Esta herramienta transforma el enfoque de la educación en Ingeniería Civil, combinando teoría y práctica de manera interactiva. Un ejemplo de esto es cómo permite gestionar costos directos (como materiales y mano de obra) e indirectos (gastos generales), simulando decisiones que los estudiantes deberán tomar en el campo profesional. Este tipo de aprendizaje autodirigido no solo facilita la adquisición de habilidades técnicas, sino que también desarrolla competencias críticas como la toma de decisiones, el pensamiento lógico y la autonomía. La arquitectura de este software se alinea con la necesidad de que los futuros ingenieros se adapten a un mundo en constante avance tecnológico, donde el dominio de herramientas digitales es cada vez más esencial.

Bibliografía

Cárdenas-Gutiérrez, J. A., Barrientos-Monsalve, E. J., & Molina-Salazar, L. (2022). Arquitectura de software para el desarrollo de herramienta Tecnológica de Costos, Presupuestos y Programación de obra. *I+ D Revista de Investigaciones*, *17*(1), 89-100.

https://www.redalyc.org/journal/5337/533774788007/533774788007.pdf

Diagrama



Artículo 05: 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 interfaces externas (como bases de datos, interfaces

de usuario y APIs). En esta arquitectura, el núcleo del negocio está protegido y no depende de tecnologías específicas, sino que interactúa con el exterior a través de "puertos" (interfaces) y "adaptadores" (implementaciones específicas de esas interfaces). Esto permite que el software mantenga su funcionalidad principal inalterada, 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

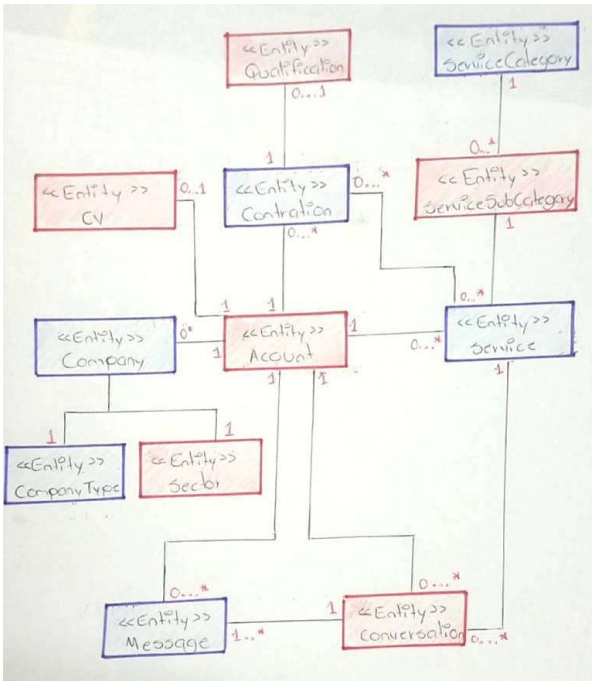
En un ecosistema donde la tecnología cambia con rapidez, es crucial que el software mantenga su estructura sin volverse obsoleto. La arquitectura hexagonal garantiza que la lógica de negocio se mantenga independiente, de forma que las actualizaciones o cambios en herramientas externas, como frameworks o motores de bases de datos, no interrumpan su funcionamiento. 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, este modelo favorece un desarrollo más ágil y adaptable.

Bibliografía

Cambarieri, M., Difabio, F., & García Martínez, N. (2020). Implementación de una arquitectura de software guiada por el dominio. In *XXI Simposio Argentino de Ingeniería de Software (ASSE 2020)-JAIIO 49 (Modalidad virtual)*.

https://sedici.unlp.edu.ar/bitstream/handle/10915/115198/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Diagrama



Artículo 06: Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube.

Resumen

Este artículo presenta un entorno de diseño especializado que asiste 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 comunes y patrones utilizados en sistemas cloud. Además de facilitar el modelado de arquitecturas complejas, este entorno incorpora una herramienta de verificación gráfica que asegura la correcta aplicación de los patrones de diseño. Al estructurar el proceso de diseño en módulos específicos, esta herramienta permite a los arquitectos generar diseños que mantienen la calidad y la alineación con los requisitos del sistema, en particular los no funcionales (como la escalabilidad y la seguridad).

Reflexión

La computación en la nube trae nuevos desafíos al diseño de software debido a la necesidad de integrar sistemas en infraestructuras compartidas, gestionar recursos distribuidos y adaptarse a estándares cambiantes. Este entorno permite a los arquitectos abstraer estas complejidades, guiándolos hacia la creación de arquitecturas estandarizadas y consistentes. Los módulos de diseño y verificación ofrecen un marco de referencia para

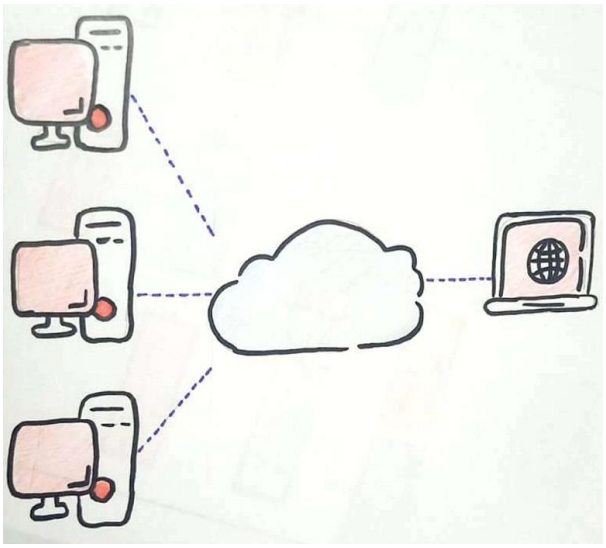
asegurar que cada componente cumple con los principios de calidad y diseño establecidos, optimizando el trabajo arquitectónico y reduciendo el riesgo de errores. Este enfoque modular y validado es fundamental en contextos de alta demanda, donde se requiere tanto escalabilidad como flexibilidad para adaptarse a las demandas del negocio y a la tecnología en constante evolución.

Bibliografía

Blas, M. J., Leone, H. P., & Gonnet, S. M. (2019). Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube.

https://ri.conicet.gov.ar/bitstream/handle/11336/125130/CONICET_Digital_Nro.8053e909-ab36-4e05-a990-4d92bb067f45_A.pdf?sequence=2&isAllowed=y

Diagrama



Artículo 07: Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico.

Resumen

El artículo analiza una metodología innovadora para mejorar la usabilidad en el desarrollo de software, introduciendo este atributo desde la fase inicial de diseño y no solo al final del proceso. En el marco del 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 centrarse en la interfaz gráfica, STATUS sugiere patrones que afectan la estructura interna del software. Estos patrones incluyen

funciones clave como "deshacer," "cancelar," y soporte para "múltiples idiomas," y se organizan en atributos, propiedades y patrones. Los atributos, como la facilidad de aprendizaje y la satisfacción del usuario, se descomponen en propiedades específicas, tales como retroalimentación y control, las cuales son implementadas a través de patrones arquitectónicos. El enfoque implica un proceso inductivo en el que los desarrolladores construyen modelos arquitectónicos iniciales y los modifican para incluir estos patrones. Posteriormente, estos patrones se validan mediante pruebas en proyectos reales, en 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 cambios en etapas posteriores y promoviendo una arquitectura que optimice la experiencia 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 antes se identificaban solo al final del desarrollo, lo que genera un proceso más eficiente y menos costoso. Además, resalta la importancia de integrar la usabilidad como un componente estructural, y no únicamente como un elemento de la interfaz gráfica,

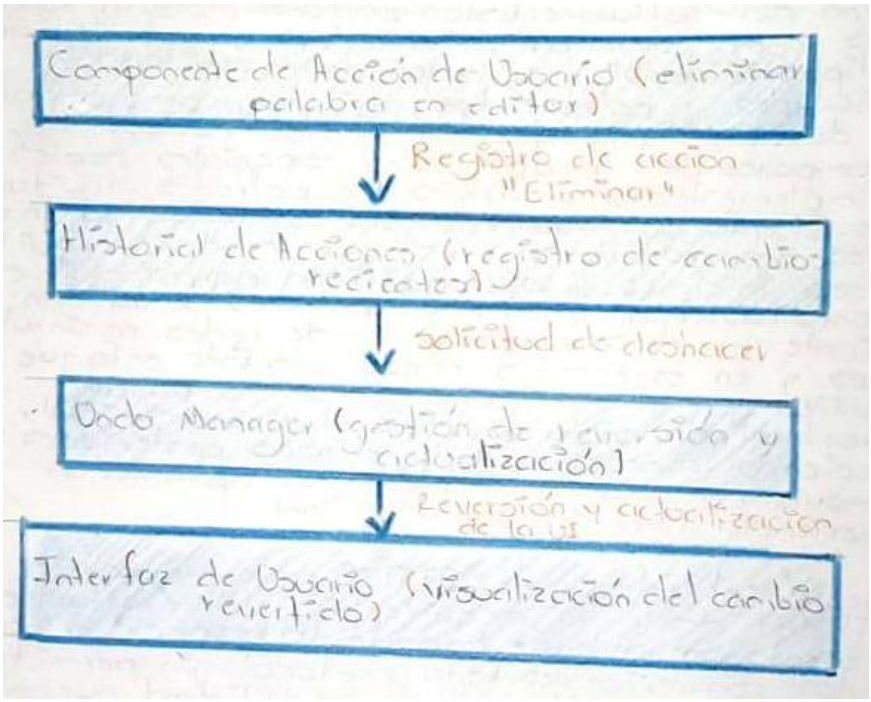
promoviendo un sistema intuitivo y fácil de usar desde el núcleo del software. Al trabajar con patrones arquitectónicos, STATUS crea una relación directa entre la estructura del sistema y la experiencia del usuario, lo que facilita diseños más robustos que no requieren ajustes de último momento. Esta 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 a errores.

Bibliografía

Moreno, A. M., & Sánchez-Segura, M. (2003, November). Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el Momento Arquitectónico. In *JISBD* (pp. 117-126).

https://www.researchgate.net/profile/Ana-Moreno-56/publication/221595496_Patrones_de_Usabilidad_Mejora_de_la_Usabilidad_del_Software_desde_el_Momento_Arquitectonico/links/0deec51d6997037422000000/Patrones-de-Usabilidad-Mejora-de-la-Usabilidad-del-Software-desde-el-Momento-Arquitectonico.pdf

Diagrama



Artículo 08: Arquitectura del 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 base 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 de SOA presenta retos, como la gestión de la comunicación entre los servicios, el tiempo de respuesta y el tamaño de los mensajes.

Reflexión

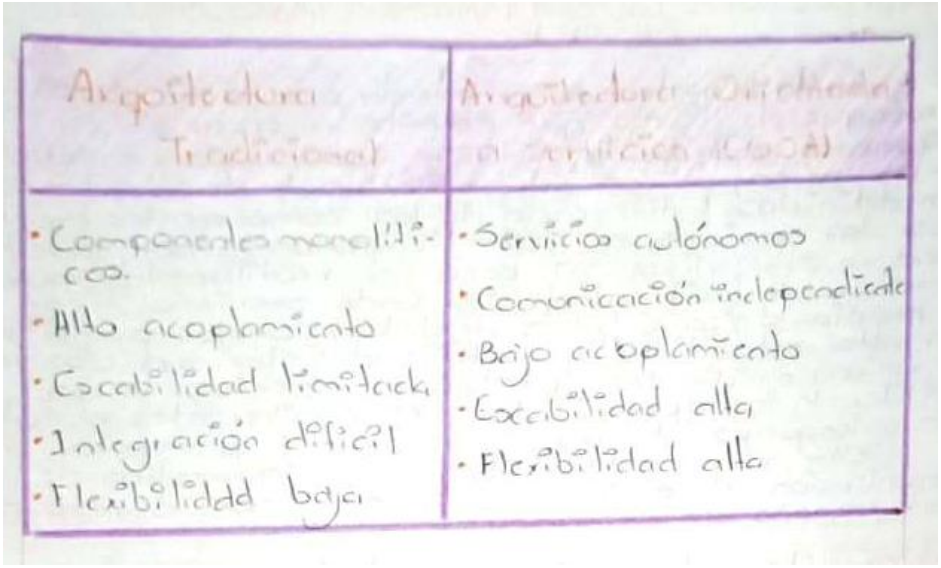
La evolución hacia arquitecturas orientadas a servicios refleja un cambio significativo en cómo se diseñan y desarrollan las aplicaciones. Mientras que las arquitecturas monolíticas eran más fáciles de desarrollar inicialmente, a medida que las aplicaciones crecen y los requerimientos cambian, estas 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. a 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, capacidad de responder rápidamente y atractiva para el desarrollo de aplicaciones modernas.

Bibliografía

Romero, P. Á. (2006). Arquitectura de software, esquemas y servicios. *Ingeniería Industrial*, *27*(1), 1.

https://dialnet.unirioja.es/servlet/articulo?codigo=4786655

Diagrama



Artículo 09: Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte.

Resumen

Este articulo habla sobre el concepto de "Lenguajes de Patrones" en la Arquitectura de Software, destacando su impacto en la mejora de la calidad y mantenibilidad del software. Desde sus orígenes en la ingeniería de software, pasando por la creación de patrones de diseño en la arquitectura de software, hasta la aplicación 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 escenarios 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

Jimenez-Torres, V. H., Tello-Borja, W., & Rios-Patiño, J. I. (2014). Lenguajes de patrones de arquitectura de software: una aproximación al estado del arte. *Scientia et technica*, *19*(4), 371-376.

https://www.redalyc.org/pdf/849/84933912003.pdf

Diagrama

.

Artículo 10: Introducción a 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 completos 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, influenciando 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 estándar 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

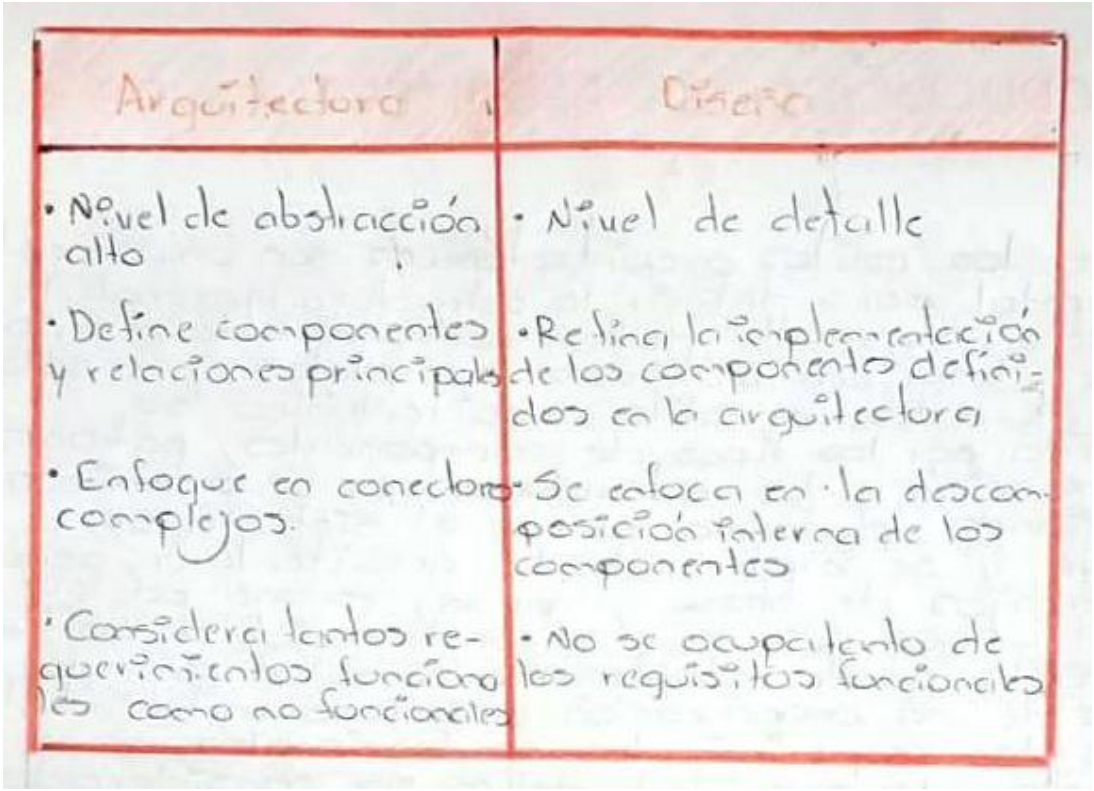
El ciclo de la arquitectura debe ser un proceso continuo que no solo se basa en los requerimientos funcionales, sino que también toma en cuenta los no funcionales desde el inicio. Los requerimientos no funcionales, como la seguridad, el rendimiento y la escalabilidad, son a menudo ignorados en las primeras 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 solo 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

Cristiá, M. (2008). Introducción a la Arquitectura de Software. *Research-Gate.[Online]. Recuperado de: https://www. researchgate. net/publication/251932352 Introduccion a la Arquitectura de Software*.

https://www.fceia.unr.edu.ar/~mcristia/Introduccion_a_la_Arquitectura_de_Software.pdf

Diagrama



Artículo 11: 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 esenciales 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 las características estáticas, 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 modificabilidad o la seguridad, pero pueden sacrificar otras, como la performance.

Reflexión

Es que el proceso de diseñar una arquitectura de software se asemeja a la creación de un mapa detallado que guía el futuro del sistema. Las decisiones tomadas durante la fase de diseño no solo 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 permiten a los desarrolladores tomar decisiones

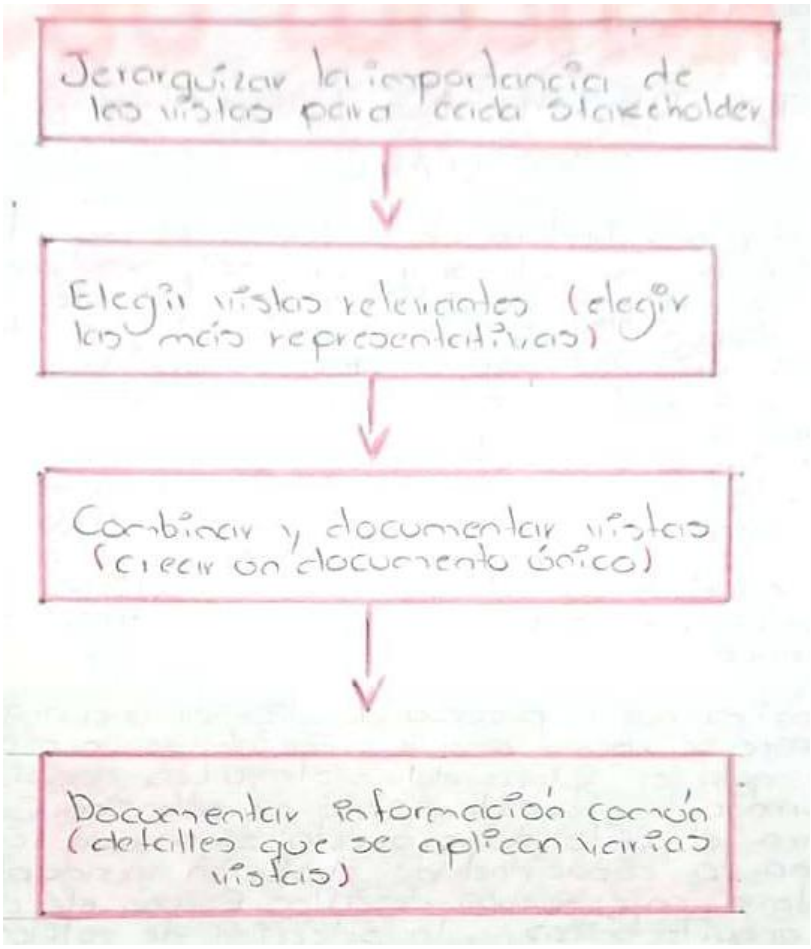
informadas que optimicen el rendimiento, la seguridad y otros atributos críticos. Sin embargo, este proceso no está exento de riesgos, ya que incluso una pequeña modificación en la arquitectura puede tener efectos a largo plazo en la funcionalidad del sistema.

Bibliografía

Bastarrica, M. C. (2005). Atributos de Calidad y Arquitectura del Software.

http://www.grise.upm.es/rearviewmirror/conferencias/jiisic04/Tutoriales/tu4.pdf

Diagrama



Artículo 12: Arquitectura de Software para el Soporte de Comunidades Académicas Virtuales en Ambientes de Televisión Digital Interactiva.

Resumen

El artículo aborda 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 incluyen un directorio de servicios, un mediador de canales y un 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 como no asociados al contenido televisivo. La propuesta se valida a través de aplicaciones educativas, como el curso AgroEDiTV, y busca ampliar la difusión de la televisión interactiva como una herramienta clave en la educación, especialmente 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 t-comercio y t-salud. Esta flexibilidad es crucial para hacer frente a los desafíos educativos actuales y

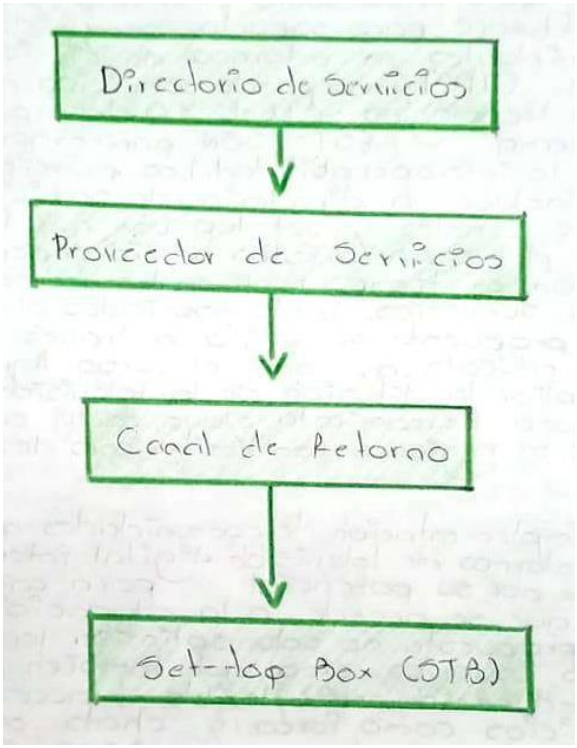
aprovechar las ventajas de la televisión interactiva en un mundo cada vez más digital.

Bibliografía

Campo, W. Y., Chanchí, G. E., & Arciniegas, J. L. (2013). Arquitectura de software para el soporte de comunidades académicas virtuales en ambientes de televisión digital interactiva. Formación universitaria, 6(2), 03-14.

https://www.scielo.cl/scielo.php?pid=S0718-50062013000200002&script=sci_arttext

Diagrama



Artículo 13: Integración de Arquitectura de Software en el Ciclo de Vida de las Metodologías Ágiles. 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 complejos de identificar, son cruciales para una arquitectura sólida y adecuada en proyectos ágiles. La identificación 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 suelen ser subjetivos, relativos e interactuar 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 tesis y 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

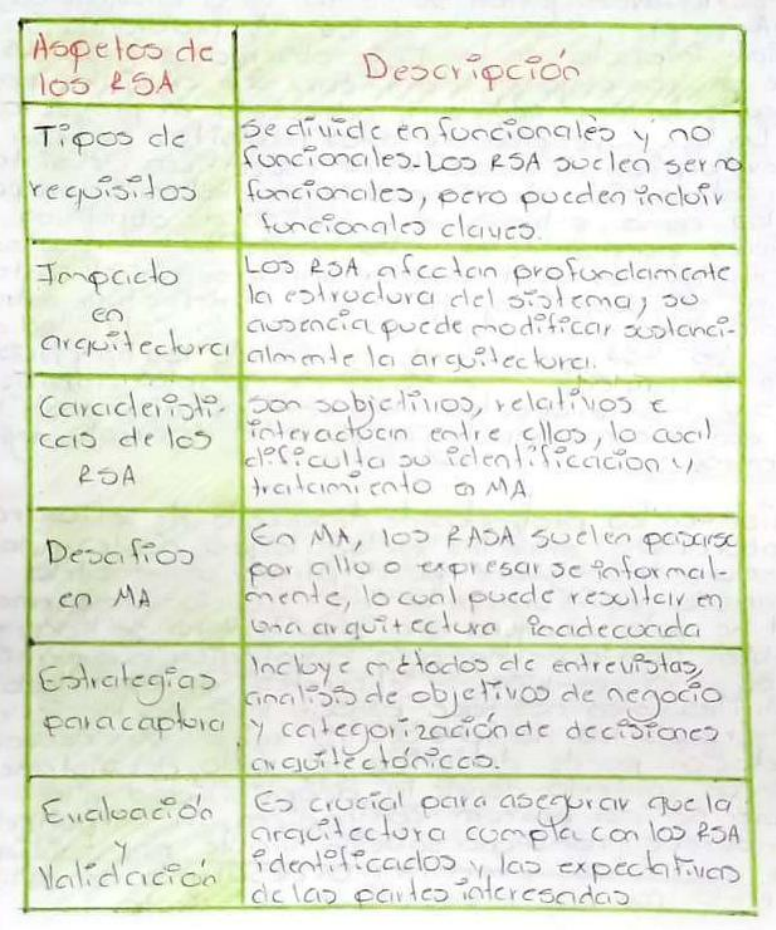
desarrollo equilibrado. La identificación temprana de RSA no solo asegura una arquitectura sólida, sino que también aporta una guía estratégica que permite adaptarse a los cambios inherentes de las metodologías ágiles. Esto nos hace repensar la importancia de los requisitos no funcionales y cómo su adecuada interpretación puede determinar el éxito del sistema final. En un contexto donde la adaptabilidad y la estructura sólida parecen contradecirse, la "Arquitectura Ágil" aparece como una solución viable para mitigar estos conflictos y facilitar un desarrollo de software sostenible y alineado con las expectativas del cliente

Bibliografía

Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (2017, September). Integración de arquitectura de software en el ciclo de vida de las metodologías ágiles. In XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017, ITBA, Buenos Aires).

https://sedici.unlp.edu.ar/bitstream/handle/10915/62077/Documento_completo.pdf?sequence=1

Diagrama



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 sustancialmente la arquitectura.
Características de los RSA	son subjetivos, relativos e interactúan entre ellos, lo cual dificulta su identificación y tratamiento en MA
Desafíos en MA	En MA, los RASA suelen pasarse por alto o expresarse 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 cumpla con los RSA identificados y las expectativas de las partes interesadas

Artículo 14: Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web.

Resumen

En el contexto del desarrollo de aplicaciones orientadas a la web, los patrones de diseño GOF han demostrado ser una herramienta importante para mejorar la calidad del software, sin embargo, su implementación sigue siendo limitada debido a factores como la

falta de conocimiento y experiencia entre los desarrolladores. Los patrones creacionales, como Singleton y Factory Method, fueron los más aplicados en los proyectos analizados, seguidos por los patrones de comportamiento como Iterator y Template Method. En los proyectos seleccionados, se observó que la adopción de estos patrones es parcial, y algunos patrones como Abstract Factory y Observer se integran mejor en marcos de trabajo que se utilizan de manera predeterminada. En general, la investigación confirma que los patrones de diseño son esenciales en el desarrollo de software de alta calidad, pero también destaca que para que los desarrolladores puedan usarlos 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 adopción 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 y cuándo 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 enfoque en simplificar

la enseñanza de estos patrones para que su uso sea más accesible y pueda integrarse 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 Orientadas a la Web. Información tecnológica, 24(3), 103-114.

https://www.scielo.cl/pdf/infotec/v24n3/art12.pdf

Diagrama

Categoría	Descripción	Ejemplos	Aplicación en Proyectos
Patrones Creacionales	Controlan la creación de objetivos	Singleton, Factory Method	67% de los proyectos
Patrones Estructurales	Organizar las relaciones entre clases y objetivos	Decorator, Facade	50% de los proyectos
Patrones de Comportamiento	Gestionan la interacción y distribución de responsabilidades	Iterator, Template, Method	67% de los proyectos
Resumen	Aplicación general de patrones GOF en proyectos web		

Artículo 15: 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 plugins 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 que la nueva arquitectura mejora atributos como portabilidad, mantenibilidad, escalabilidad y reusabilidad.

Reflexión

El desarrollo de sistemas complejos, como el de Vismedic, exige arquitecturas que permitan 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 acoplados. Este caso subraya la importancia de anticipar futuros requisitos durante la fase

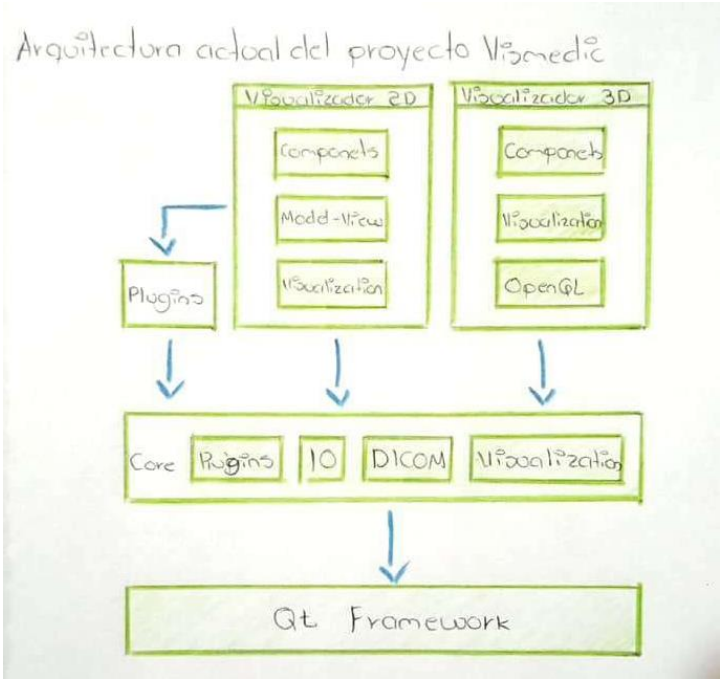
de diseño y la relevancia de incluir técnicas de evaluación para validar decisiones arquitectónicas. En esencia, planificar y diseñar con un enfoque adaptable no solo mejora la eficiencia del desarrollo, sino que también reduce el esfuerzo requerido para escalar o modificar el sistema.

Bibliografía

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

http://scielo.sld.cu/pdf/rcim/v8n1/rcim06116.pdf

Diagrama



Artículo 16: Desarrollo de sistemas de software con patrones de diseño orientado a objetos.

Resumen

El proyecto "Intranet Industrial" desarrollado 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, la 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 evaluó comparando el uso y 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 diseñado 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 cumplan los objetivos iniciales de forma eficiente y confiable. Este

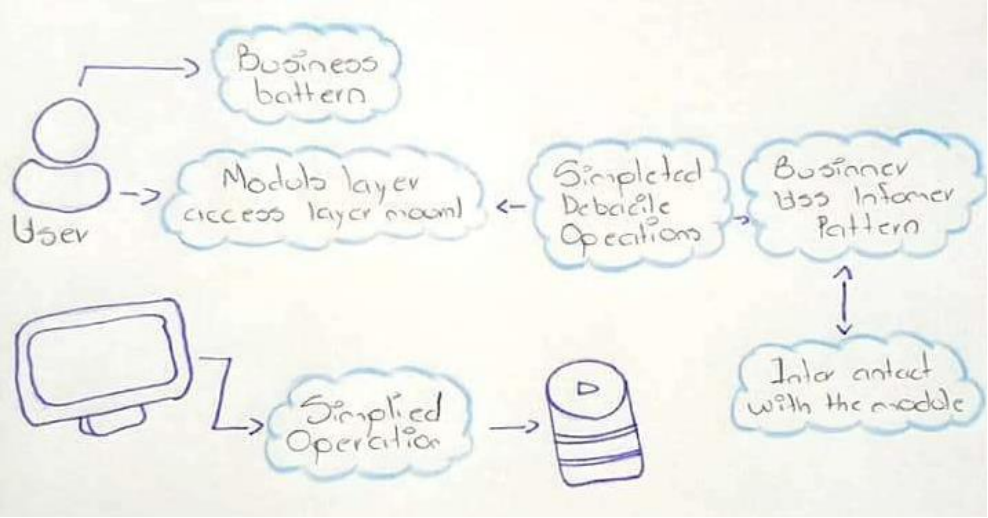
ejemplo resalta la importancia de seguir estándares y prácticas comprobadas en el desarrollo de software.

Bibliografía

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

https://cybertesis.unmsm.edu.pe/backend/api/core/bitstreams/92b01647-3651-4e75-9fd1-b00cb53826b1/content

Diagrama



Artículo 18: Arquitectura de software en el proceso de desarrollo ágil: una perspectiva basada en requisitos significantes para la arquitectura.

Resumen

Los **Requisitos Significantes para la Arquitectura (RSA)** son elementos esenciales que influyen directamente en la estructura, diseño y calidad de un sistema de software. Estos requisitos pueden ser de 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 ineficiente, incapaz de satisfacer los objetivos clave del sistema. Uno de los mayores retos de los RSA radica en su subjetividad, pues su relevancia depende del contexto y las perspectivas de quienes los interpretan. Además, suelen interactuar entre sí, donde priorizar uno puede beneficiar o perjudicar a otros. El artículo aborda esta problemática proponiendo estrategias como el uso de **frameworks**, el enfoque en **conocimiento del dominio** y la alineación con **objetivos empresariales** para integrar los RSA 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 escalabilidad sean respetados a lo largo del proyecto.

Reflexión

Identificar y trabajar con los RSA 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 habilitador, capaz de responder a las demandas de un mercado dinámico. El artículo destaca la importancia de adoptar estrategias claras y flexibles para gestionar los RSA, como basarse en el conocimiento del entorno o las metas del negocio. Esto asegura un equilibrio entre flexibilidad y estabilidad, dos conceptos aparentemente opuestos pero esenciales para el éxito en metodologías ágiles.

Bibliografía

Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., & Rueda, J. R. (2018). Arquitectura de software en el proceso de desarrollo ágil: una perspectiva basada en requisitos significantes para la arquitectura. In *XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste)*..

Diagrama

Método	Descripción	Ventajas	Desafíos
Framework	Basados en estudios empíricos con expertos	Estandarización y robustez	Requiere expertos calificados
Conocimiento del Dominio	Enfocado en propiedades del entorno y suposiciones	Adaptabilidad a contextos específicos	Limitado por el alcance del conocimiento
Objetivos del Negocio	Priorizar metas empresariales y atributos de calidad	Relación directa con el valor del negocio	Puede ser objetivo en ciertas industrias

Artículo 19: 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:

1. Uso de encuestas para identificar necesidades de los usuarios.
2. Priorización de características como explicaciones claras y ayudas visuales.

3. Implementación de una arquitectura de tres capas (presentación, 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

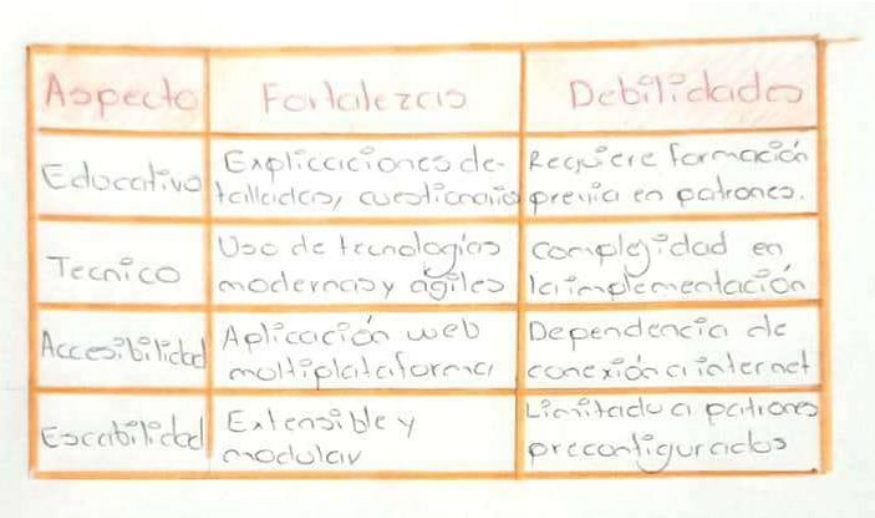
La construcción de esta herramienta no solo plantea un avance en el ámbito académico, sino que también introduce un enfoque innovador en la creación de herramientas educativas. Las metodologías ágiles permiten adaptarse a cambios y mejoran la alineación con las necesidades reales del usuario.

Bibliografía

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

https://riunet.upv.es/bitstream/handle/10251/174122/Ferrandis%20-%20Desarrollo%20de%20una%20herramienta%20para%20el%20aprendizaje%20de%20patrones%20de%20diseno%20software.pdf?sequence=1&isAllowed=y

Diagrama



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

Artículo 20: Aplicación de patrones de diseño para garantizar alta flexibilidad en el software.

Resumen

El artículo analiza cómo los patrones de diseño permiten desarrollar 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 estrategia, compuesto y fábrica para estructurar el diseño y gestionar reglas de descuento variables. La clase principal "Pedido" delega las operaciones relacionadas con los descuentos a estrategias específicas, garantizando alta cohesión. Por su parte, la fábrica se encarga de instanciar las estrategias necesarias, promoviendo bajo acoplamiento. Este enfoque respeta el principio abierto-cerrado, permitiendo añadir nuevas reglas de negocio sin alterar el código existente.

Además, se integran técnicas de polimorfismo para implementar las estrategias y simplificar la extensión del sistema. Este diseño permite escalar el software sin comprometer su funcionalidad ni estabilidad. Como conclusión, el artículo recomienda el uso de frameworks que automaticen la creación de objetos y refuercen la implementación de patrones de diseño. Así, se asegura un software que minimiza costos de mantenimiento, facilita pruebas y responde a los desafíos de entornos empresariales dinámicos.

Reflexión

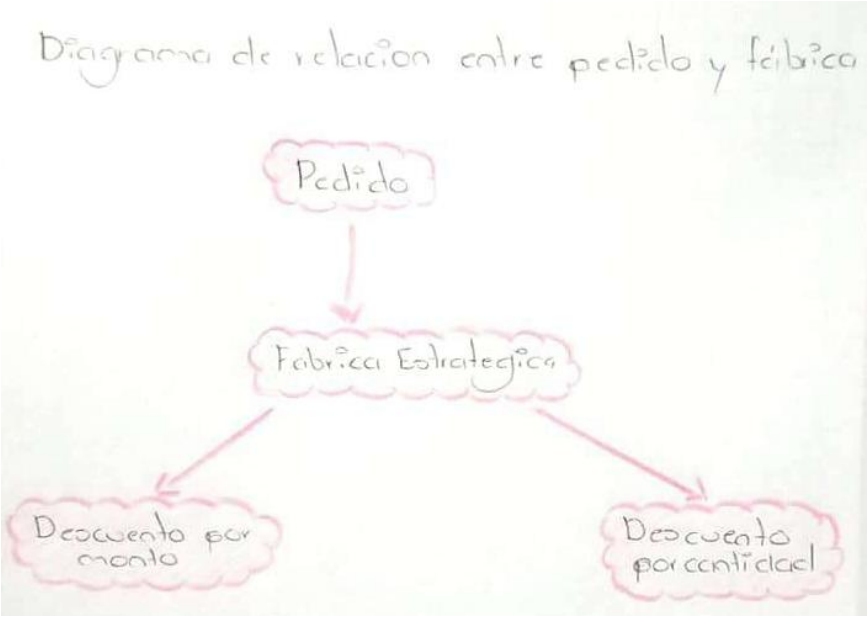
La flexibilidad del software no solo depende de la tecnología utilizada, sino también de un diseño sólido fundamentado en patrones reconocidos. Este artículo ilustra cómo los patrones de diseño, combinados con principios como el abierto-cerrado, garantizan que las aplicaciones sean sostenibles y escalables a largo plazo. En el caso del sistema de pedidos, la modularidad logrado mediante el uso de patrones estrategia y compuesto permite manejar la complejidad sin comprometer la cohesión del sistema. Por otro lado, delegar la creación de estrategias a una fábrica minimiza las dependencias, reduciendo el riesgo de errores al realizar cambios. Este enfoque resalta la importancia de formar desarrolladores no solo en herramientas, sino en conceptos de diseño que mejoran la calidad del software. Más allá de los aspectos técnicos, este método fomenta la innovación en sistemas empresariales, permitiendo a las empresas adaptarse rápidamente a un mercado competitivo, manteniendo una base tecnológica sólida y confiable.

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.

https://revistas.ucv.edu.pe/index.php/rtd/article/view/1685

Diagrama



Artículo 21: Herramienta para reusó 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 usables y eficientes mediante la reutilización de código JavaScript basado en patrones de interacción. Los patrones incluyen soluciones específicas adaptadas a problemas comunes, como botones, menús desplegables y otros componentes de interfaz. ReusMe permite personalizar estas soluciones, ofreciendo flexibilidad y control al usuario final. Esto representa una ventaja respecto a herramientas tradicionales, que suelen limitar las opciones de personalización. Además, la plataforma asegura ahorro de tiempo y recursos al evitar que los desarrolladores comiencen desde cero en cada proyecto, aumentando así 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

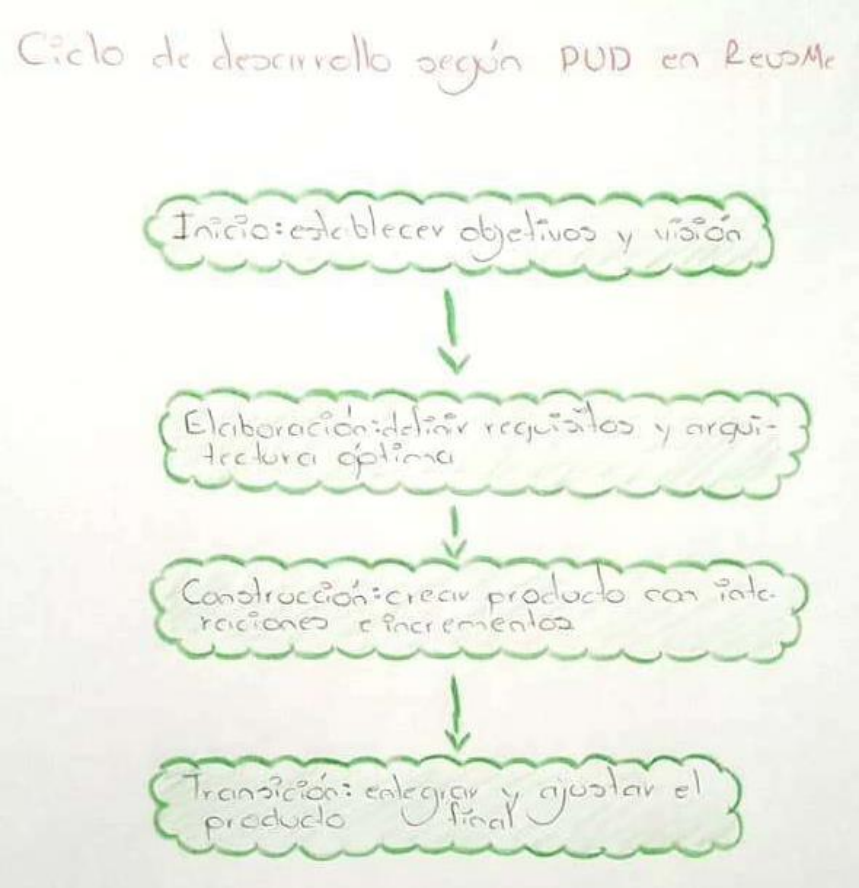
probadas y exitosas se comparten y adaptan a diferentes contextos. Sin embargo, también plantea la necesidad de mantener altos estándares de documentación y pruebas para asegurar que el código reutilizable sea confiable y funcional.

Bibliografía

Benigni, G., Antonelli, O., & Vásquez, Y. (2009). TOOL FOR REUSE OF JAVASCRIPT CODE ORIENTED TO INTERACTION PATTERNS. SABER. Multidisciplinary Journal of the Research Council of the University of the East , 21 (1), 60-69.

https://www.redalyc.org/pdf/4277/427739438009.pdf

Diagrama



Artículo 22: Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.

Resumen

El texto describe cómo las arquitecturas monolíticas limitan la capacidad de las organizaciones para mantener y actualizar aplicaciones web complejas. Este enfoque, que concentra toda la funcionalidad en un solo bloque, genera problemas como tiempos

prolongados para realizar actualizaciones, altos riesgos de fallos y dificultades para escalar sistemas. La arquitectura monolítica también incrementa 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

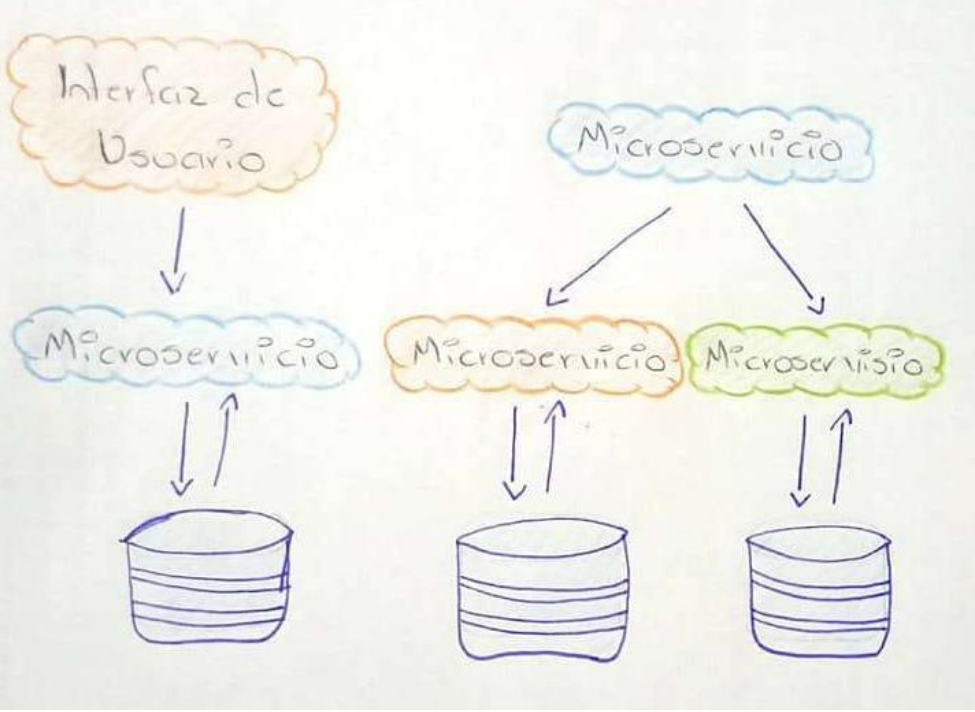
El documento subraya que la elección de una arquitectura de software es una decisión clave que afecta directamente la eficiencia, escalabilidad y mantenimiento de los sistemas. Las limitaciones de las aplicaciones monolíticas, especialmente en el sector público, dificultan la agilidad organizacional y aumentan los riesgos operativos. La arquitectura de microservicios ofrece una alternativa moderna que permite descomponer las aplicaciones en componentes pequeños 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., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.

http://138.59.13.30/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf

Diagrama



Artículo 23: 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 (ROA) 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 diseñadores crear cursos combinando objetos de aprendizaje de múltiples repositorios distribuidos.

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 estas arquitecturas pueden aplicarse en contextos menos desarrollados es crucial para democratizar la educación virtual.

Bibliografía

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

https://d1wqtxts1xzle7.cloudfront.net/101413527/ELDE120_Rojas-libre.pdf?1682298336=&response-content-disposition=inline%3B+filename%3DUna_arquitectura_de_software_para_la_int.pdf&Expires=1732232388&Signature=Zeg0TmUuWpw91abav2HZpC3QTNGbUhljab9TF0gQiGzNLKX5vSCl9yHQOuq0tM2a7jyp~EC4MD5VE4xUBZcoWYZ3a4-xM-h40Ab7qUMW3jbu09gu6VXRLzFqYm8FtFlLF~6xr29sOPI9aUzydmOwJ1pGT5GgFjFFXcqmXdX4ydvsKNdVhL73GPdKECdxEb-J6D96E7kheA9yGqjOIVLOjKVPdOdvgO2pbkH0ILRCZG9t-MLgkHQcDbJZvIEXkE8r7vEo8t-L5C56hkKzB3Rku5dwbhEmmq0FbcCMs-d223OsAVBx1-~nBPGnxmSxCLZAlRMzd5fLISGnL3DsUwDmJg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Diagrama

Artículo 24: Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones 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 estándares 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, incluye puntos calientes y congelados para permitir personalización y robustez, respectivamente.

Reflexión

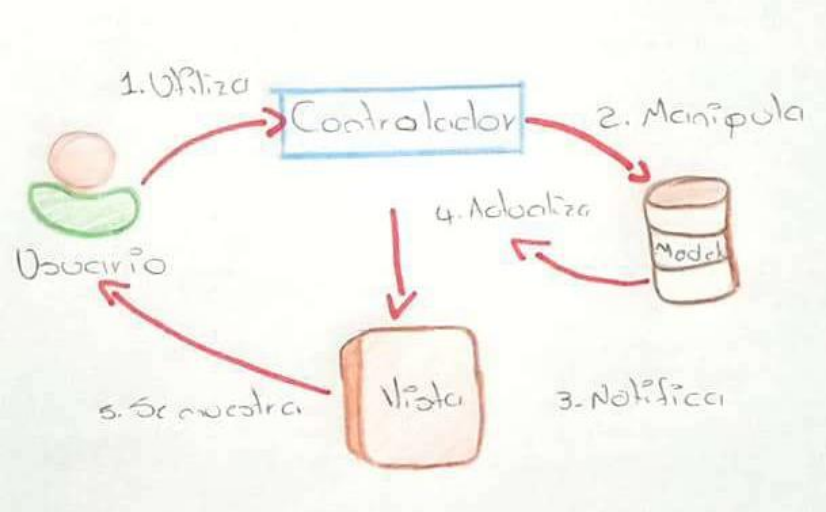
Esta propuesta es un ejemplo del impacto que tiene la reutilización de componentes y la adopción de estándares en la ingeniería del software. Permite a los desarrolladores enfocarse en los requisitos específicos de cada proyecto sin reinventar procesos básicos. Sin embargo, el éxito de este tipo de frameworks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

Bibliografía

MURILLO, M. M. (2019). Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web.

http://170.210.231.10:8080/jspui/bitstream/123456789/191/1/MURILLO%2C%20Mart%C3%ADn%20Miguel.pdf

Diagrama



Artículo 25: 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 cumplan sus objetivos, debido al desconocimiento o a la dificultad para acceder a patrones de diseño 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 (GITAE) desarrolló un entorno, EGPat, que permite la gestión y consulta de patrones de diseño, además de recomendar los

más adecuados para problemas específicos mediante minería de texto y técnicas de razonamiento basado en casos.

Reflexión

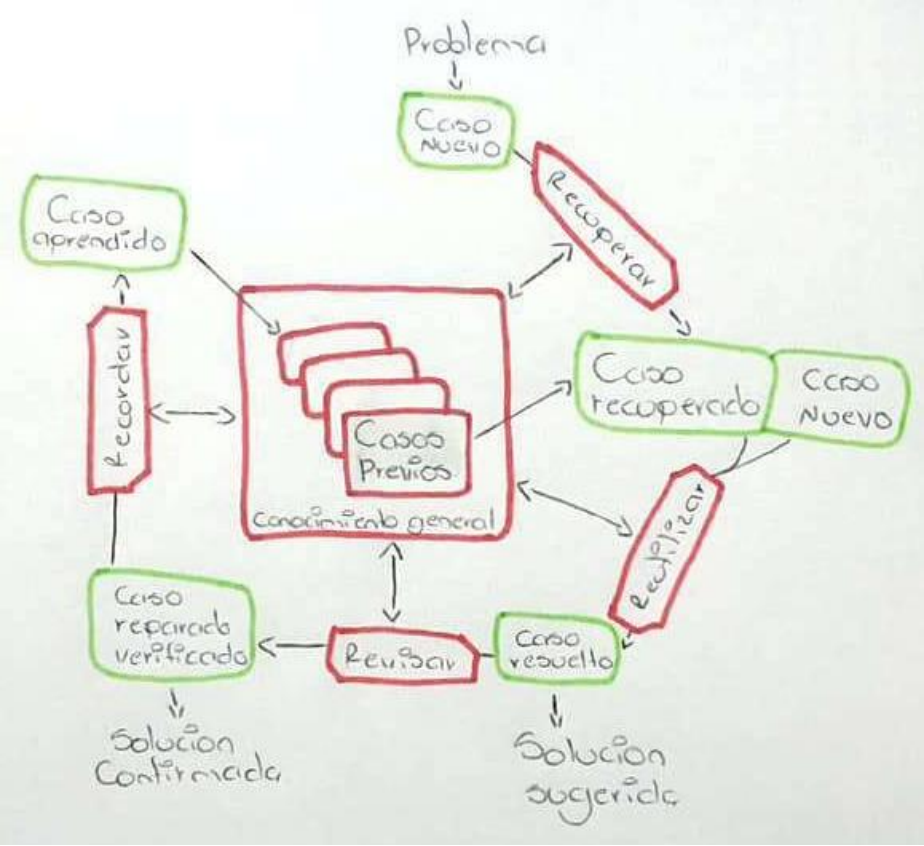
Esta investigación destaca cómo la tecnología puede democratizar el acceso a herramientas avanzadas, reduciendo la brecha técnica entre expertos y novatos. EGPat refleja una solución innovadora que no solo mejora la eficiencia en el diseño, sino que también fomenta la creación de materiales educativos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

Bibliografía

Alfonso Azcuy, R., & Llull Céspedes, L. Á. (2021). Módulo de recomendación de patrones de diseño para EGPat. Revista Cubana de Ciencias Informáticas, 15(2), 118-137.

http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992021000200118&lang=es

Diagrama



Artículo 26: 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 de código y facilidad para pruebas unitarias. La investigación concluye que no existe un patrón superior, ya que cada uno está diseñado para resolver problemas

específicos. Los patrones son herramientas esenciales para estructurar aplicaciones robustas y facilitar el mantenimiento del software.

Reflexión

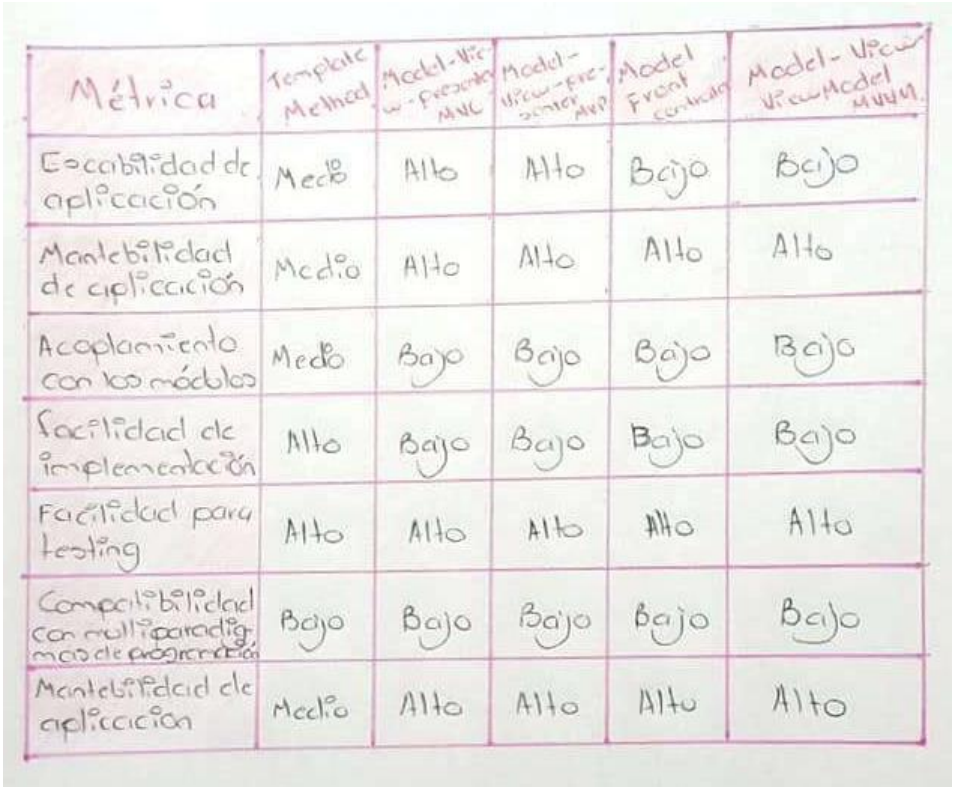
Este análisis resalta la importancia de seleccionar patrones de diseño adecuados según el contexto del proyecto. Si bien los patrones son herramientas potentes para modularidad y escalabilidad, también exigen un conocimiento técnico significativo para su implementación. Este enfoque técnico invita a los desarrolladores a evaluar cuidadosamente las necesidades del sistema antes de decidir qué patrón adoptar, optimizando así los recursos y evitando sobreingeniería.

Bibliografía

Alvarez, O. D. G., Larrea, N. P. L., & Valencia, M. V. R. (2022). Análisis comparativo de Patrones de Diseño de Software. Polo del Conocimiento: Revista científico-profesional, 7(7), 2146-2165.

https://dialnet.unirioja.es/servlet/articulo?codigo=9042927

Diagrama



Métrica	Template Method	Model-View-Presenter MVC	Model-View-Controller MVP	Model Front controller	Model-View-ViewModel MVVM
Escalabilidad de aplicación	Medio	Alto	Alto	Bajo	Bajo
Mantebilidad de aplicación	Medio	Alto	Alto	Alto	Alto
Acoplamiento con los módulos	Medio	Bajo	Bajo	Bajo	Bajo
Facilidad de implementación	Alto	Bajo	Bajo	Bajo	Bajo
Facilidad para testing	Alto	Alto	Alto	Alto	Alto
Compatibilidad con multiparadigma de programación	Bajo	Bajo	Bajo	Bajo	Bajo
Mantebilidad de aplicación	Medio	Alto	Alto	Alto	Alto

Artículo 27: Arquitectura de software académica para la compresión del desarrollo de software en capas.

Resumen

La arquitectura en capas es una metodología 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 implementa las reglas de negocio, y la capa de datos se encarga de almacenar y recuperar información en sistemas

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. Es especialmente útil en sistemas que requieren flexibilidad, mantenimiento sencillo y extensibilidad a largo plazo. 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

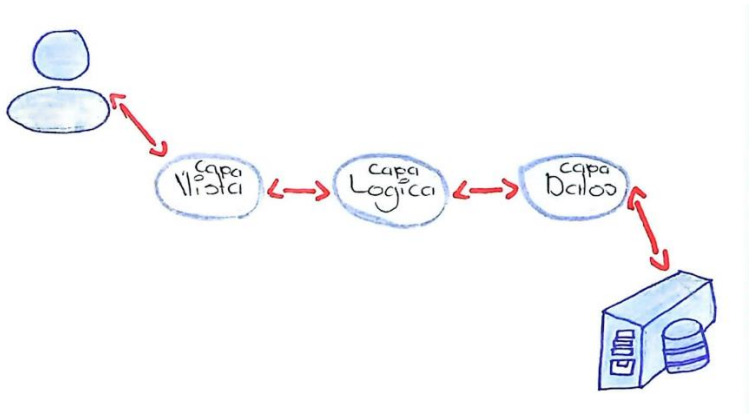
La arquitectura en capas refleja un enfoque profesional para el diseño de sistemas, priorizando la claridad, el modularidad y la separación de responsabilidades. En proyectos a largo plazo, permite incorporar nuevas funcionalidades sin comprometer la estabilidad existente. Sin embargo, su éxito depende de un diseño inicial 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 exige un esfuerzo adicional en las fases iniciales del proyecto, pero con grandes recompensas a futuro.

Bibliografía

Cardacci, D. G. (2015). Arquitectura de software académica para la comprensión del desarrollo de software en capas (No. 574). Serie Documentos de Trabajo.

https://www.econstor.eu/bitstream/10419/130825/1/837816424.pdf

Diagrama



Artículo 28: Desarrollo de una arquitectura de software para el robot móvil Lázaro.

Resumen

El robot móvil Lázaro, diseñado para terrenos irregulares, implementa una arquitectura de software basada en tres niveles principales:

1. **Nivel Base:** Responsable de gestionar sensores y actuadores a bajo nivel, operando en procesos secuenciales o multiproceso.

2. **Nivel de Desarrollo:** Permite extender las capacidades del robot mediante nuevas librerías y comportamientos 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.

La arquitectura soporta teleoperación o modos autónomos de operación, utilizando módulos XBee para comunicación remota. Se realizaron pruebas funcionales que confirmaron la escalabilidad y adaptabilidad del sistema.

Reflexión

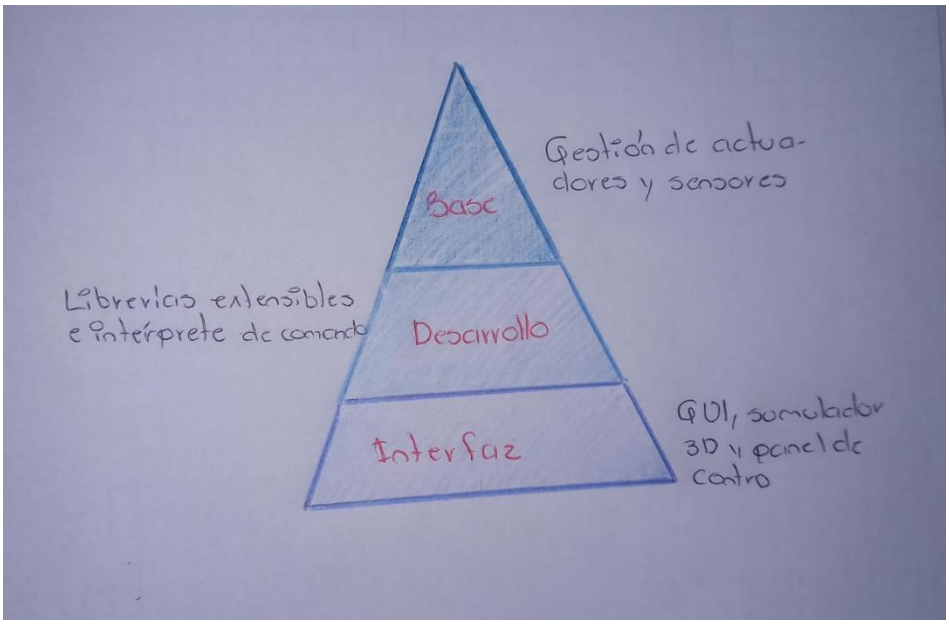
Esta arquitectura resalta cómo un diseño modular y escalable facilita la interacción entre el hardware y software, adaptándose a las necesidades cambiantes de los usuarios. Además, la inclusión de herramientas de simulación y programación accesibles mejora la eficiencia en la operación y experimentación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

Bibliografía

García, J. M., Gil, Á. E., & Sánchez, E. A. (2018). Desarrollo de una arquitectura de software para el robot móvil Lázaro. *Ingeniare. Revista chilena de ingeniería*, *26*(3), 376-390.

https://www.scielo.cl/scielo.php?pid=S0718-33052018000300376&script=sci_arttext

Diagrama



Artículo 29: Lineamientos para el diseño de aplicaciones web soportados 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 de análisis,

diseño y programación orientada a objetos (POO), permitiendo a los estudiantes de ingeniería desarrollar aplicaciones flexibles, reutilizables y con un vocabulario común.

Se describen nueve patrones principales de GRASP, como **Experto**, **Creador**, **Controlador**, y **Polimorfismo**, que guían el diseño de software más eficiente. Por ejemplo, el patrón "Controlador" asigna el flujo de eventos del sistema a clases específicas, facilitando la centralización de la validación y la seguridad. El artículo concluye con ejemplos prácticos como el diseño de un sistema de cajero automático, aplicando los patrones para garantizar un diseño modular y funcional.

Reflexión

Los patrones GRASP ofrecen un marco conceptual sólido para diseñar software orientado a objetos. Al enfocarse en asignar responsabilidades de manera clara, 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 de POO, lo que implica una curva de aprendizaje significativa para los estudiantes.

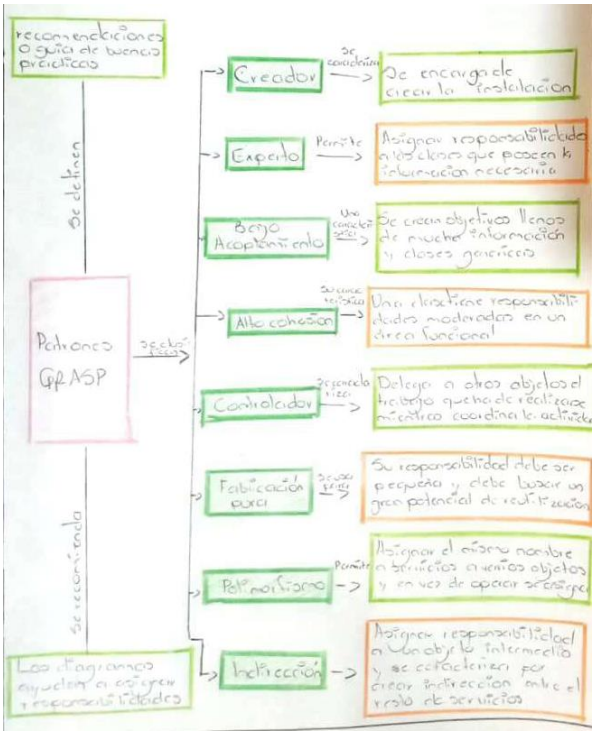
Bibliografía

Ortega, G. A. V. (2021). Lineamientos para el diseño de aplicaciones web soportados en patrones GRASP. Ciencia e Ingeniería: Revista de investigación

interdisciplinar en biodiversidad y desarrollo sostenible, ciencia, tecnología e innovación y procesos productivos industriales, 8(2), 4.

https://dialnet.unirioja.es/servlet/articulo?codigo=8742482

Diagrama



Artículo 30: 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 las distintas fases del desarrollo de software. MDA se basa en transformar modelos de alto nivel, como los CIM (Modelos Independientes del Cómputo) y PIM (Modelos Independientes de la Plataforma), en modelos específicos de implementación (PSM) y en código fuente mediante transformaciones automáticas. Este enfoque fomenta la trazabilidad, la separación de responsabilidades y la portabilidad del software.

Por otro lado, el ciclo en espiral prioriza la gestión de riesgos en un proceso iterativo, con fases como planificación, análisis de riesgos, 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 la creciente complejidad del desarrollo de software. Al enfocarse en la trazabilidad

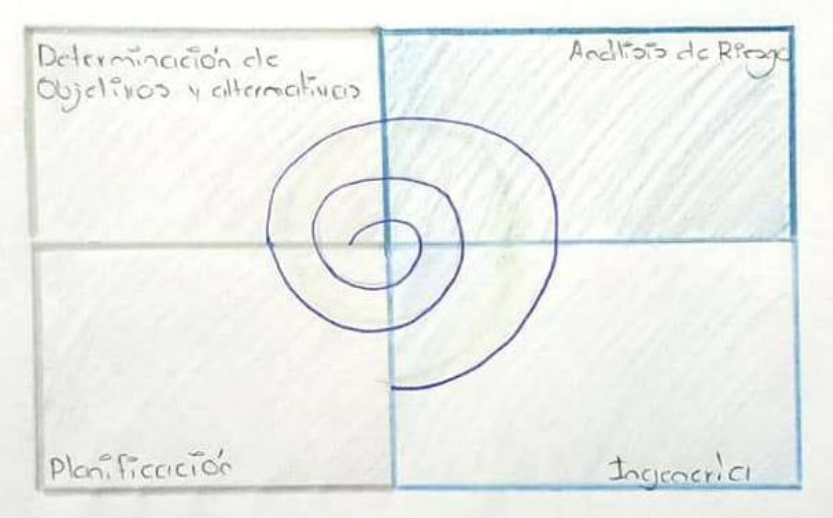
y el análisis iterativo, se mejora la calidad y la adaptabilidad del producto final. Sin embargo, implementar estas metodologías puede resultar desafiante debido a la necesidad de herramientas específicas y habilidades técnicas avanzadas.

Bibliografía

Meaurio, V. S., & Schmieder, E. (2013). La arquitectura de software en el proceso de desarrollo: integrando MDA al ciclo de vida en espiral. *Archivo de la Revista Latinoamericana de Ingeniería de Software*, *1*(4), 142-146.

https://revistas.unla.edu.ar/software/article/view/103

Diagrama



Artículo 31: 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 activas identificadas en la lógica se mapean a procesos, garantizando concurrencia adecuada.
- De procesos a física: 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 las reglas de diseño, como minimizar dependencias entre módulos.

Los escenarios actúan como vínculo para validar estas transiciones y comprobar que las interacciones cumplen con los requisitos iniciales.

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, ya que reflejan situaciones reales de uso, lo que fomenta un entendimiento común del sistema. En proyectos grandes, los escenarios mitigan riesgos al abordar elementos críticos desde las primeras fases del diseño.

Bibliografía

Kruchten, P. (1995). Planos Arquitectónicos: El Modelo de 4+ 1 Vistas de la Arquitectura del Software. IEEE software, 12(6), 42-50.

https://juliopezblog.wordpress.com/wp-content/uploads/2021/04/planos-arquitectonicos-el-modelo-de-4-1-vistas-de-la-arquitectura-del-software.pdf

Diagrama

Artículo 32: Introducción a la reingeniería de software mediante patrones de diseño.

Resumen

El artículo explora cómo la reingeniería puede transformar sistemas heredados en software moderno y funcional. Explica que el proceso comienza con la ingeniería inversa para recuperar diseño y datos relevantes. Luego, el análisis/rediseño identifica las necesidades del usuario y los ajustes necesarios en el sistema. Finalmente, la ingeniería avanzada implementa cambios mediante la modularización y pruebas para garantizar calidad y mantenibilidad. Se destacan las herramientas Colombus y Maisa para automatizar y mejorar este proceso. Colombus convierte el código fuente en diagramas UML, mientras Maisa aplica minería de patrones para identificar y optimizar el diseño existente.

Reflexión

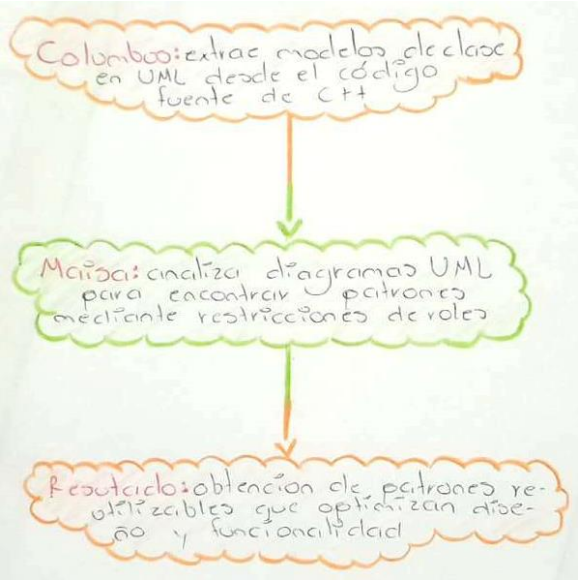
El uso de patrones de diseño en la reingeniería evidencia la importancia de construir software reutilizable y bien estructurado. Identificar y aplicar estos patrones en sistemas heredados permite mantener consistencia en el desarrollo. Además, los refactorings juegan un papel crucial al preservar el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

Bibliografía

Contreras, T. B. Introducción a la Reingeniería de Software Mediante Patrones de Diseño.

https://ccc.inaoep.mx/~balderas/documents/misc/reingenieria-sw.pdf

Diagrama



Artículo 33: Uso de patrones de diseño y meta programación 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 paradigmas como programación orientada a objetos y metaprogramación para optimizar el rendimiento y facilitar el desarrollo de aplicaciones. Se

realizaron tres iteraciones que incluyeron desde prototipos básicos hasta el uso avanzado de templates. El resultado es un API eficiente, modular y capaz de adaptarse a diversos entornos, demostrando la aplicabilidad de estas técnicas en sistemas embebidos.

Reflexión

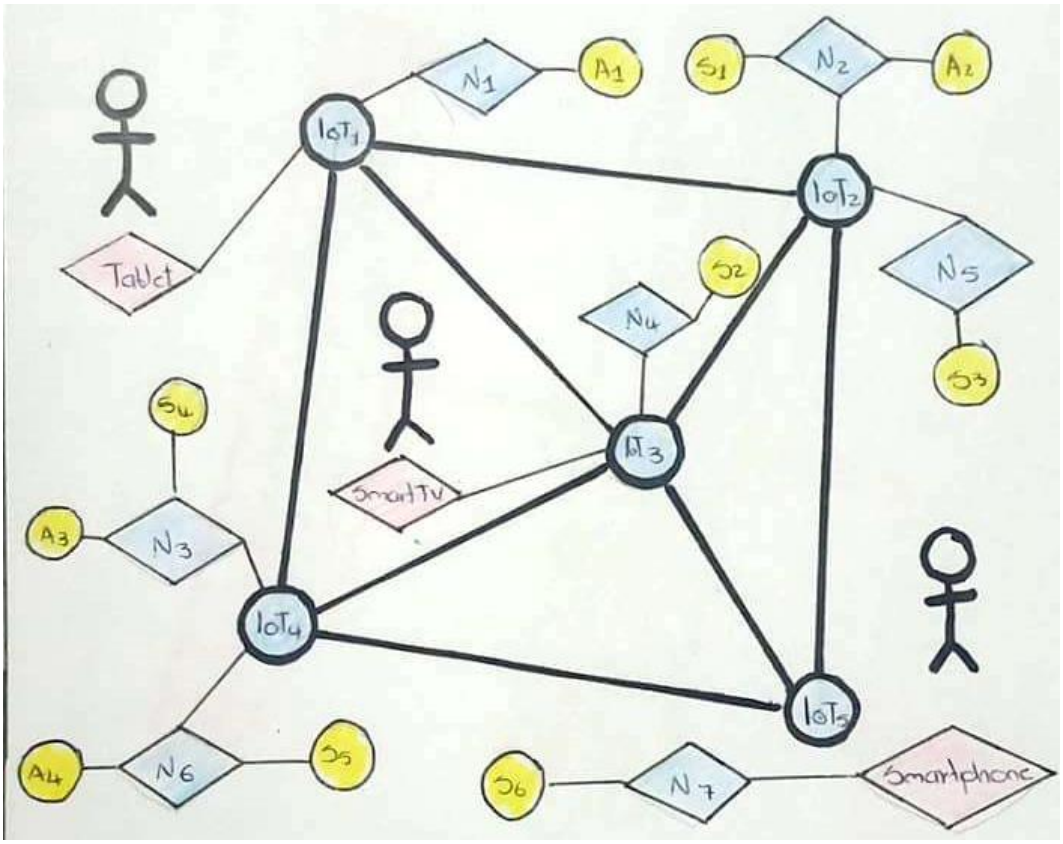
El enfoque iterativo y participativo de la investigación-acción fue clave para el éxito del proyecto. Este método permite identificar problemas prácticos y ajustar soluciones en tiempo real, evidenciando la importancia de integrar investigación y práctica en el desarrollo de tecnología IoT.

Bibliografía

Pacheco, A., Escobar, J., & Trujillo, E. USO DE PATRONES DE DISEÑO Y METAPROGRAMACIÓN PARA CONSTRUIR APIS DE IOT USANDO C+.

https://www.researchgate.net/profile/Alberto-Pacheco/publication/354694385_Uso_de_patrones_de_diseno_y_metaprogramacion_para_construir_APIs_de_IoT_usando_C/links/61480d3a519a1a381f6fce8b/Uso-de-patrones-de-diseno-y-metaprogramacion-para-construir-APIs-de-IoT-usando-C.pdf

Diagrama



Artículo 34: Diseño de framework web para el desarrollo dinámico de aplicaciones.

Resumen

El framework propuesto sigue una arquitectura cliente-servidor diseñada para adaptarse tanto a intranets como a entornos de Internet, utilizando HTTP como protocolo base. Incorpora componentes clave como un controlador central para gestionar solicitudes, un gestor de seguridad que valida roles y permisos, un gestor de formularios para operaciones CRUD, y un gestor de acceso a datos que abstrae la interacción con diversas

bases de datos. Se utilizaron tecnologías como Apache, PHP, MySQL y librerías como AdoDB para garantizar escalabilidad, rendimiento y seguridad. Además, se emplearon herramientas como Komodo Edit y Subversion para facilitar el desarrollo colaborativo y la gestión de versiones. El resultado es un entorno que permite la personalización y adaptación dinámica a diferentes proyectos empresariales.

Reflexión

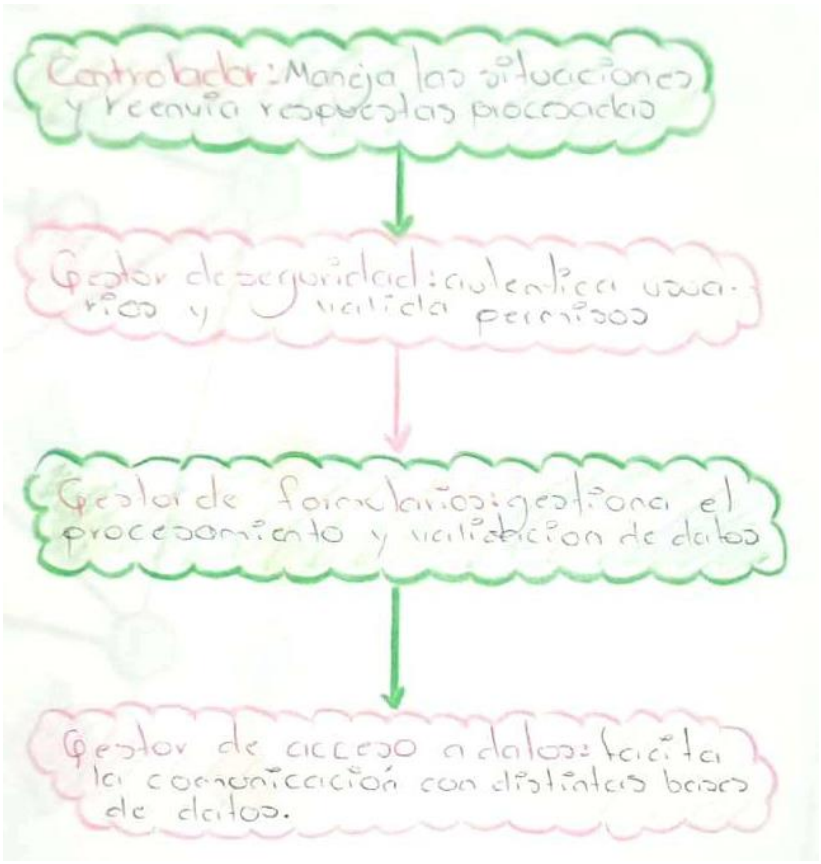
La capacidad de reutilizar componentes a través de un framework no solo optimiza los recursos, sino que también contribuye al mantenimiento y evolución de proyectos. Esta práctica permite abordar problemas comunes con soluciones probadas, lo que incrementa la calidad y la seguridad del software. Al fomentar una estructura modular, los frameworks se convierten en herramientas esenciales para garantizar la sostenibilidad y escalabilidad de los sistemas, respondiendo a las demandas de un mercado cada vez más competitivo.

Bibliografía

Villalobos, G. M., Sánchez, G. D. C., & Gutiérrez, D. A. B. (2010). Diseño de framework web para el desarrollo dinámico de aplicaciones. Scientia et technica, 16(44), 178-183.

https://www.redalyc.org/pdf/849/84917316032.pdf

Diagrama



Artículo 35: 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 en PHP basada en patrones de diseño, utilizando el Proceso Personal de Software (PSP). El objetivo principal es automatizar el mapeo 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 ORM, optimizando el manejo de datos y permitiendo un enfoque en reglas de negocio.

Reflexión

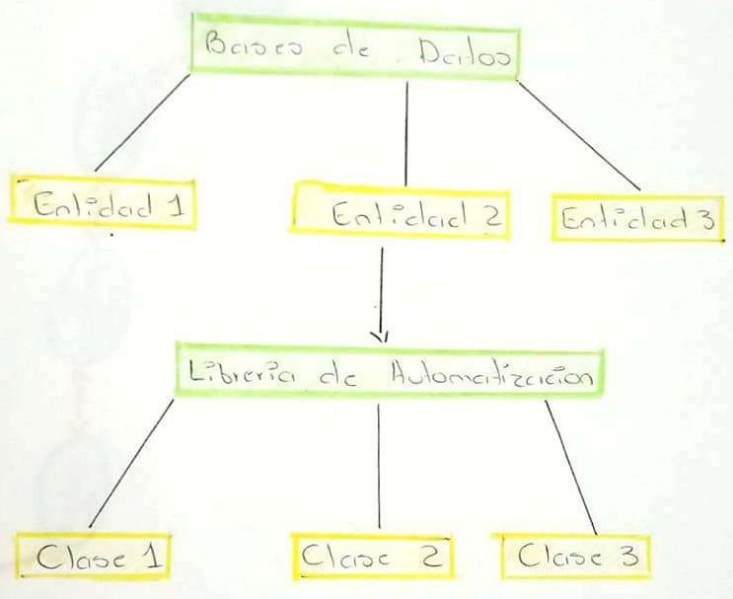
Este enfoque automatizado es crucial en el desarrollo moderno, ya que reduce la carga repetitiva y el riesgo de errores. Además, fomenta prácticas de codificación robustas y escalables. Sin embargo, el éxito de herramientas como esta depende de su capacidad para adaptarse a proyectos diversos y de mantener estándares altos, algo que el uso de patrones de diseño ayuda a lograr.

Bibliografía

Torres Morales, J. L. (2014). Construcción de una librería para generación automática de clases PHP basada en patrones de diseño utilizando PSP.

https://repositorio.puce.edu.ec/items/80bd0890-e882-419c-99ce-25b1ebc24ca2

Diagrama



Artículo 36: 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 MVC. Estos estilos permiten reutilizar patrones y optimizar procesos de desarrollo, vinculando teoría con práctica y adaptándose a distintas necesidades.

Reflexión

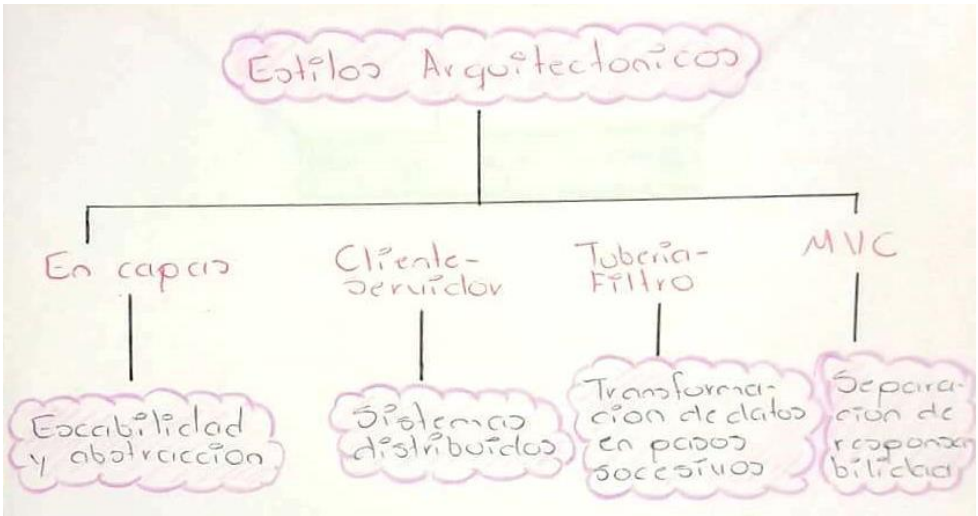
Los estilos arquitectónicos representan un marco esencial para estructurar sistemas, fomentando claridad, reutilización y mantenimiento. Este enfoque abstracto sirve como un puente entre teoría y práctica, ayudando a arquitectos y desarrolladores a diseñar sistemas escalables y robustos. Sin embargo, su elección requiere un análisis cuidadoso 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.

http://biblioteca.udgvirtual.udg.mx/jspui/handle/123456789/940

Diagrama



Artículo 37: Propuesta de arquitectura de software 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, mantenibilidad y portabilidad mediante el uso de herramientas libres y patrones arquitectónicos modernos. Se 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 producto.

Reflexión

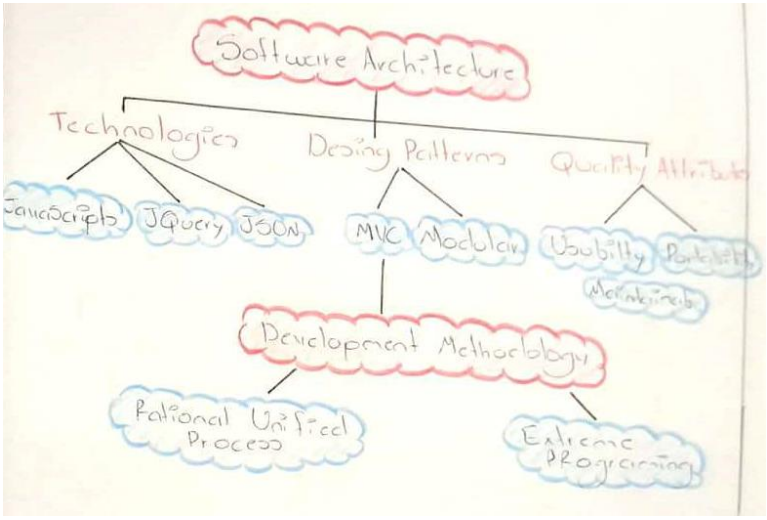
La arquitectura propuesta refleja la importancia de un diseño bien estructurado en proyectos educativos, especialmente al considerar la accesibilidad y simplicidad necesarias para niños pequeños. Su enfoque en herramientas libres y estándares claros destaca la intención de democratizar el acceso al aprendizaje. 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).

https://repositorio.uci.cu/bitstream/ident/TD_04616_11/1/TD_04616_11.pdf

Diagrama



Artículo 38: Arquitectura de programación web: backend.

Resumen

El desarrollo backend es esencial para crear aplicaciones web dinámicas que permitan interactividad y gestión eficiente de recursos en el servidor. Este ámbito se centra en las tareas que no son visibles para el usuario final, como la interacción con bases de datos y la ejecución de lógica de negocio. El texto detalla la evolución de lenguajes como PHP, Python, Ruby, y Java, así como frameworks destacados como Laravel, Django y Spring. También se exploran nuevas tendencias, 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 satisfacer las necesidades crecientes de los usuarios y las empresas. La tendencia hacia el uso de lenguajes más eficientes y tecnologías emergentes como el desarrollo serverless o blockchain indica un futuro en el que el backend será más modular, flexible y escalable. Además, el enfoque en mejorar la experiencia de usuario mediante interfaces dinámicas y servicios distribuidos resalta la importancia de un diseño centrado en las personas.

Bibliografía

Montávez Sánchez, M. (2024). Arquitectura de programación web: backend. https://crea.ujaen.es/handle/10953.1/24482

Diagrama