

# REVISTA ACADEMICA

## ARQUITECTURA DE SOFTWARE



2694667

# **REVISTA ACADEMICA DEL SENA**

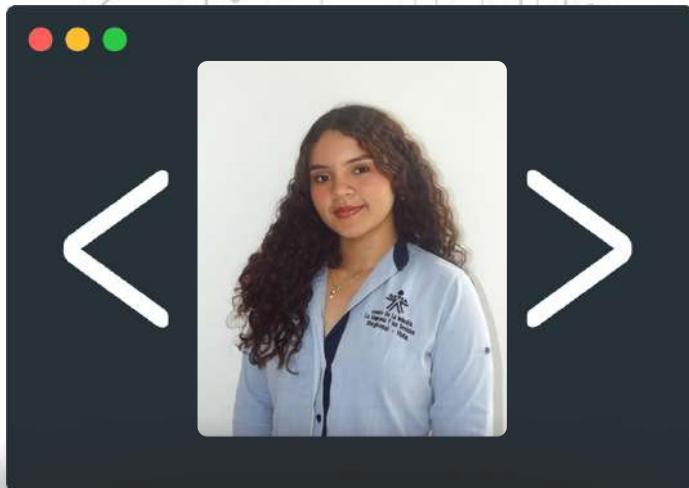
**Edición especial:  
Artículos destacados  
DESARROLLADORES**



# **DESARROLLADORES**

<b>01. Angie Lizeth Trujillo González</b>	<u><a href="#">Pag 2.</a></u>
<b>02. Anyi Zujey Gómez Casanova</b>	<u><a href="#">Pag 37.</a></u>
<b>03. Aura Maria Fierro Fierro</b>	<u><a href="#">Pag 76.</a></u>
<b>04. Camilo Andrés Bautista Cuellar</b>	<u><a href="#">Pag 107.</a></u>
<b>05. Carolina Martinez Cortes</b>	<u><a href="#">Pag 146.</a></u>
<b>06. Cristian Fernando Narváez Sánchez</b>	<u><a href="#">Pag 161.</a></u>
<b>07. Cristian Jeanpool Bahamon Granados</b>	<u><a href="#">Pag 196.</a></u>
<b>08. Dylan Santiago Narváez Pinto</b>	<u><a href="#">Pag 227.</a></u>
<b>09. Isabela Cordoba Gutiérrez</b>	<u><a href="#">Pag 228.</a></u>
<b>10. Iván Andrés Murcia Epia</b>	<u><a href="#">Pag 229.</a></u>
<b>11. John Sebastián Penna Arias</b>	<u><a href="#">Pag 230.</a></u>
<b>12. José Manuel Gasca Bonilla</b>	<u><a href="#">Pag 261.</a></u>
<b>13. Juan Pablo Betancourt Gómez</b>	<u><a href="#">Pag 269.</a></u>
<b>14. Julian David Fierro Casanova</b>	<u><a href="#">Pag 270.</a></u>
<b>15. Kevin Camilo Muñoz Campos</b>	<u><a href="#">Pag 305.</a></u>
<b>16. Laura Valentina Ariza Alejo</b>	<u><a href="#">Pag 344.</a></u>
<b>17. Manuel Ricardo Diez Corredor</b>	<u><a href="#">Pag 379.</a></u>
<b>18. Maria Del Mar Artunduaga Artunduaga</b>	<u><a href="#">Pag 380.</a></u>
<b>19. Maria José Murcia Martínez</b>	<u><a href="#">Pag 415.</a></u>
<b>20. Mariana Charry Prada</b>	<u><a href="#">Pag 439.</a></u>
<b>21. Mariana González Calderón</b>	<u><a href="#">Pag 473.</a></u>
<b>22. Maydy Viviana Conde Ladino</b>	<u><a href="#">Pag 509.</a></u>
<b>23. Mayra Alejandra Tamayo Perdomo</b>	<u><a href="#">Pag 544.</a></u>
<b>24. Patricia Sarmiento</b>	<u><a href="#">Pag 575.</a></u>
<b>25. Valentina Silva Garrido</b>	<u><a href="#">Pag 599.</a></u>
<b>26. Willian Steban González Cortes</b>	<u><a href="#">Pag 634.</a></u>
<b>27. Yordy Erik Nuñez Pineda</b>	<u><a href="#">Pag 645.</a></u>

# Angie Lizeth Trujillo González



Mi nombre es Angie Trujillo, soy desarrolladora de software. Cuento con una sólida formación en tecnología adquirida en el SENA. A la edad de 19 años, he tenido la oportunidad de adentrarme en el ámbito del desarrollo tecnológico, elaborando proyectos innovadores y funcionales que convierten ideas en realidades digitales.

Desde el inicio, he sido motivada por el anhelo de adquirir conocimiento y enfrentar desafíos, lo que me ha destacado por mi habilidad para solucionar dificultades y aportar valor en cada proyecto en el que me involucro. Mi compromiso con la excelencia y mi entusiasmo por la tecnología me motivan a continuar expandiendo mis horizontes y a explorar nuevas oportunidades en este apasionante ámbito.

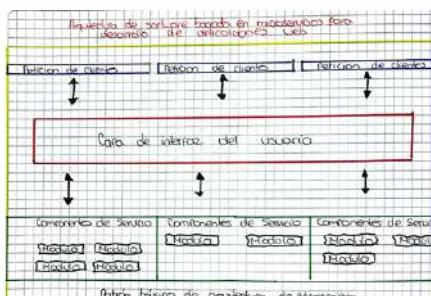
Mi objetivo es continuar desarrollando una trayectoria profesional que motive a otros jóvenes a confiar en sus capacidades y a animarse a marcar su presencia en el campo de la tecnología.

# Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.

El desarrollo de software en la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) se ha basado en una arquitectura monolítica. Este enfoque empaqueta toda la funcionalidad en una única unidad ejecutable, siguiendo las tendencias impuestas por el lenguaje de programación utilizado y la experiencia del área de desarrollo. Si bien este modelo ha sido funcional, ha generado desafíos significativos en términos de mantenimiento, escalabilidad y entrega de software, especialmente a medida que las necesidades y los sistemas han evolucionado. Actualmente, la investigación está desarrollando una propuesta para abordar estas limitaciones mediante la implementación de un prototipo basado en una nueva arquitectura, aunque aún no se han obtenido resultados definitivos.

## Reflexión

El uso de arquitecturas monolíticas presenta problemas en sistemas complejos, dificultando el mantenimiento y la escalabilidad. Esto resalta la necesidad de adoptar arquitecturas modernas, como los microservicios, que dividen aplicaciones en componentes independientes. Esta transición mejora tanto el mantenimiento como la escalabilidad, permitiendo un trabajo más ágil y eficiente para los equipos de desarrollo. La investigación actual puede transformar la forma en que la CGTIC aborda el desarrollo de software, promoviendo soluciones sostenibles.



## Bibliografía

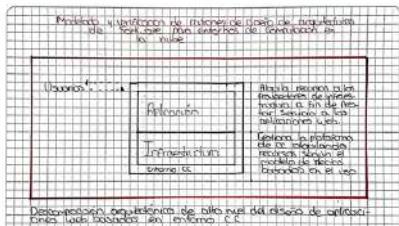
- López, D., & Maya, E. (s/f). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web. Surl.li. Recuperado el 21 de noviembre de 2024, de <http://surl.li/adpdwe>

# Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

Presenta un enfoque integral para el diseño de arquitecturas de software, especialmente para aplicaciones web. Se destaca la importancia de utilizar arquitecturas genéricas que proporcionen una estructura común para resolver problemas específicos, lo que ayuda a los arquitectos a evitar conflictos derivados de la falta de experiencia. El trabajo introduce un metamodelo de componentes arquitectónicos que identifica elementos clave en el diseño, y se complementa con una herramienta gráfica para la instalación de estos componentes. Además, se aborda la verificación de patrones de diseño para asegurar su correcta aplicación, lo que contribuye a la calidad del software. El modelo UML se utiliza para describir diferentes aspectos de la arquitectura, incluyendo la aplicación en la nube y

## Bibliografía

Gonnet, S., & INGAR - Instituto de Desarrollo y Diseño. (2019). Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, 35(35), 1-17. <https://doi.org/10.17013/risti.35.1-17>

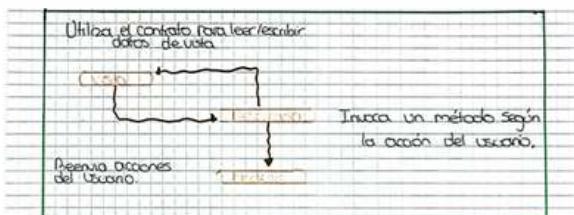


## Reflexión

El enfoque presentado en el documento es altamente relevante en el contexto actual de la computación en la nube, donde la complejidad de las aplicaciones web está en constante aumento. La propuesta de un entorno de diseño integral que combina un metamodelo con herramientas de verificación es una contribución valiosa, ya que no solo facilita el trabajo de los arquitectos de software, sino que también promueve la creación de aplicaciones más robustas y de calidad. Además, la atención a la experiencia del arquitecto resalta la necesidad de formación continua en un campo que evoluciona rápidamente. En general, este trabajo puede servir como una guía útil para profesionales y académicos interesados en mejorar sus prácticas de diseño en entornos de computación en la nube.

# Análisis comparativo de Patrones de Diseño de Software

En resumen, la evolución de las prácticas de desarrollo de software ha llevado a la implementación de pruebas durante el desarrollo para identificar problemas tempranamente y reducir costos de mantenimiento. Los patrones de diseño son herramientas esenciales que ayudan a estructurar y organizar aplicaciones, resolviendo problemas comunes con soluciones probadas. No son implementaciones directas de código, sino modelos conceptuales que guían a los desarrolladores en la resolución de problemas específicos, mejorando la calidad del software mediante el diseño orientado a objetos. El artículo analiza cinco patrones de diseño destacados: Template Method, Model-View-Controller, Model-View-Presenter, Front Controller y Model-View-ViewModel, destacando sus ventajas y desventajas. La elección del patrón adecuado depende del problema a resolver y del contexto del sistema.



## Reflexión

Los patrones de diseño son esenciales para el desarrollo de software moderno, ya que permiten la reutilización de soluciones probadas, reduciendo errores y mejorando la eficiencia. Sin embargo, seleccionar el patrón correcto puede ser un desafío para desarrolladores menos experimentados debido a la amplia variedad disponible. Patrones como MVC, MVP y MVVM ofrecen enfoques específicos para la interacción entre componentes, lo que hace que el software sea más modular y mantenable. Es crucial evaluar cada patrón en función de las necesidades del proyecto, y la documentación es fundamental para educar a los desarrolladores y garantizar un uso efectivo de estas herramientas. En resumen, los patrones de diseño establecen estándares de calidad en la industria y promueven mejores prácticas, siempre y cuando se realice un análisis adecuado y se brinde capacitación adecuada.

## Bibliografía

Análisis comparativo de Patrones de Diseño de Software. (n.d.). Unirioja.Es. Retrieved November 21, 2024, from <https://dialnet.unirioja.es/servlet/articulo?codigo=9042927>

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

El estudio analiza los patrones de diseño Gang of Four (GOF) presentados en el libro Design Patterns: Elements of Reusable Object-Oriented Software. Estos patrones, clasificados en creacionales, estructurales y de comportamiento, brindan soluciones a problemas comunes en el diseño de software orientado a objetos. Se resalta la importancia de estos patrones como buenas prácticas para mejorar la calidad del software, aunque también se mencionan desafíos como la falta de ejemplos didácticos en el catálogo original. Se sugiere la creación de un catálogo actualizado y didáctico para facilitar la comprensión y aplicación de los patrones GOF en contextos específicos como el desarrollo web. Por último, se propone como trabajo futuro crear un catálogo actualizado y didáctico que facilite la comprensión y la implementación correcta de los patrones GOF en contextos específicos como el desarrollo web.

## Bibliografía

Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web. (n.d.). Scielo.Cl. Retrieved November 21, 2024, from [https://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci\\_arttext&tlang=en](https://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci_arttext&tlang=en)

## Reflexión

Los patrones de diseño GOF han sido clave en el desarrollo de software al proporcionar soluciones reutilizables para problemas comunes, pero su alta abstracción dificulta su implementación, especialmente para desarrolladores novatos. Este estudio destaca la importancia de documentación accesible y ejemplos prácticos alineados con los modelos originales. Es esencial seleccionar los patrones con cuidado según el problema y contexto, evitando su uso innecesario para no introducir complejidad adicional. Aunque siguen siendo relevantes, la efectividad de los patrones GOF depende de la experiencia del desarrollador y de herramientas que faciliten su comprensión. Iniciativas como catálogos más didácticos pueden democratizar su uso y mejorar la calidad del software.

Patrones de diseño GOF (The Gang of Four) en el contexto de Procesos de desarrollo de aplicaciones orientada a la web

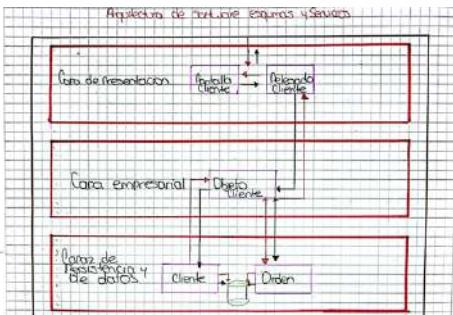
Nº	Patrón de diseño	Categoría
1	Creational	Creacionales
2	Structural	Estructurales
3	Behavioral	Comportamiento
4	Factory Method	Creacionales
5	Singleton	Creacionales
6	Adapter	Estructurales
7	Bridge	Estructurales
8	Composite	Estructurales
9	Decorator	Comportamiento
10	Facade	Creacionales
11	Visitor	Comportamiento

# Arquitectura de software, esquemas y servicios.

Las aplicaciones empresariales actuales deben cumplir con requisitos que antes eran poco comunes, como la independencia de sistemas operativos y bases de datos, el acceso desde ubicaciones distantes, la interacción con otros sistemas existentes y el manejo de grandes volúmenes de interacciones simultáneas. Esto ha generado una transición de arquitecturas monolíticas y centralizadas a entornos distribuidos y heterogéneos, lo que ha incrementado la complejidad estructural de las aplicaciones. La arquitectura de software ha emergido como una disciplina clave dentro de la ingeniería de software, ya que define las decisiones de diseño fundamentales que estructuran un sistema. Este proceso involucra determinar las partes componentes del sistema, sus interfaces, comportamientos y las relaciones entre ellas, así como su evolución.

## Bibliografía

Arquitectura de software, esquemas y servicios. (n.d.). Unirioja.Es. Retrieved November 21, 2024, from <https://dialnet.unirioja.es/servlet/Articulo?codigo=4786655>



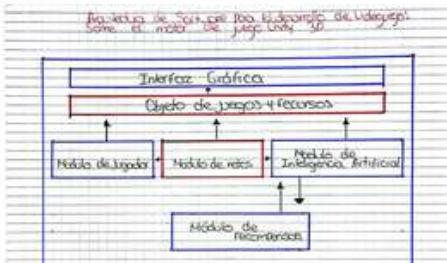
## Reflexión

La evolución hacia arquitecturas distribuidas y heterogéneas responde a las demandas de aplicaciones más flexibles y escalables. Aunque compleja, ofrece oportunidades para innovar con tecnologías emergentes. El desafío para los arquitectos de software es gestionar esta complejidad sin sacrificar eficiencia, especialmente con recursos limitados. Es crucial seguir principios de diseño sólidos y planificar cuidadosamente para crear aplicaciones sostenibles, adaptables a cambios futuros sin grandes reestructuraciones.

# Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D

El uso de las Tecnologías de Información y Comunicaciones (TIC) ha impulsado el crecimiento de la industria de videojuegos. Entre las disciplinas involucradas en su desarrollo se encuentra la programación, diseño y marketing. Los motores de videojuegos como Unity 3D simplifican y automatizan tareas complejas. Sin embargo, en la UCI se detectaron problemas como la falta de organización y reutilización limitada de componentes en videojuegos creados con Unity 3D. Para resolver esto, se diseñó una arquitectura de software que organiza las características funcionales básicas de los videojuegos. Se validó con un prototipo de videojuego de plataformas, demostrando mejoras en la reutilización y optimización de recursos. La propuesta cumple con atributos de calidad como reusabilidad y extensibilidad. Finalmente, se evaluó la solución a través del método ATAM y pruebas basadas en escenarios.

## Reflexión



## Bibliografía

Trabajo de Diploma Para Optar Por el Título, de I. en C. I. (n.d.). Arquitectura de software para videojuegos desarrollados sobre el motor de juego Unity 3D. Uci.Cu. Retrieved November 21, 2024, from [https://repositorio.uci.cu/jspui/bitstream/123456789/9382/1/TD\\_08978\\_7.pdf](https://repositorio.uci.cu/jspui/bitstream/123456789/9382/1/TD_08978_7.pdf)

El desarrollo de videojuegos refleja cómo la tecnología y la creatividad pueden converger para generar soluciones innovadoras. Sin embargo, los retos asociados a la organización y estructuración del proceso evidencian la importancia de contar con herramientas y metodologías sólidas que optimicen los recursos y promuevan la calidad del producto final. El diseño de arquitecturas de software específicas para videojuegos no solo facilita la reutilización de componentes y la eficiencia en el desarrollo, sino que también fomenta la sostenibilidad de proyectos a largo plazo. Este tipo de investigación es fundamental para formar desarrolladores capacitados y para mantener la competitividad de la industria de los videojuegos, garantizando productos que cumplan con las expectativas de los usuarios en un mercado altamente exigente.

# Arquitectura de Software orientada a la creación de micromundos para la enseñanza y el aprendizaje



## Reflexión

La educación enfrenta desafíos en un mundo digitalizado, donde la falta de integración de las TIC limita la adaptación de los estudiantes a las demandas actuales. Es necesario reestructurar los modelos educativos para fomentar habilidades tecnológicas, creatividad y autoaprendizaje, garantizando acceso equitativo a los recursos. El analfabetismo digital, especialmente en países en desarrollo, perpetúa desigualdades económicas y sociales. Para superarlo, es crucial invertir en infraestructura tecnológica, capacitar a docentes y estudiantes, y diseñar currículos flexibles que aprovechen las herramientas digitales, formando generaciones preparadas para los retos del siglo XXI.

Los procesos de enseñanza-aprendizaje enfrentan desafíos como la falta de recursos y el desconocimiento del uso de las TIC, limitando a estudiantes y docentes. Los currículos rígidos restringen la creatividad y dificultan la integración de las TIC. Muchas instituciones no aprovechan adecuadamente los recursos tecnológicos, afectando la formación. El analfabetismo digital, especialmente en países en desarrollo, crea desigualdades educativas y sociales. Es clave implementar herramientas que fomenten el autoaprendizaje, adaptar los modelos educativos y garantizar un uso eficiente de las TIC en la enseñanza.

## Bibliografía

- Guerrero, L. (2008). Arquitectura de Software orientada a la creación de micromundos para la enseñanza y el aprendizaje. *Avances Investigación en Ingeniería*, 1(8), 88-95. <https://revistas.unilibre.edu.co/index.php/avances/article/view/2637>

# USO DE PATRONES DE DISEÑO DE SOFTWARE: UN CASO PRÁCTICO

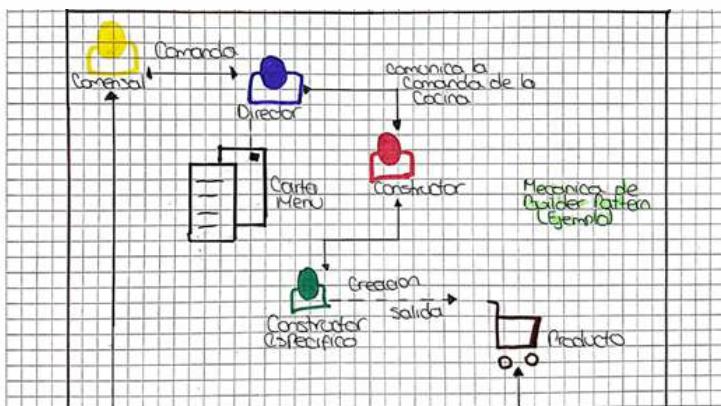
Los patrones de diseño son fundamentales en la ingeniería de software, ya que proporcionan soluciones probadas a problemas comunes y facilitan el desarrollo estructurado y eficiente de aplicaciones. Sus principales beneficios incluyen la reutilización del código, la reducción de la complejidad, el desacoplamiento de los componentes y la mejora del mantenimiento. En un experimento en la Universidad de Costa Rica, los estudiantes aplicaron patrones de diseño en un proyecto que simulaba un procesador multinúcleo basado en la arquitectura MIPS. Esto amplió la funcionalidad del simulador, mantuvo la fidelidad con las arquitecturas reales y favoreció el desarrollo de componentes desacoplados, lo que facilita su mantenimiento y extensión. Los resultados mostraron que la implementación de patrones mejora la calidad del software, aunque se debe considerar su impacto en la complejidad del código y las pruebas.

## Reflexión

Los patrones de diseño son fundamentales en la formación de ingenieros de software, ya que enseñan soluciones probadas a problemas comunes. Incorporarlos desde etapas tempranas en la universidad prepara a los estudiantes para enfrentar desafíos reales en la industria, fomentando habilidades prácticas y una mentalidad adaptable. Aunque aumentan la complejidad inicial, sus beneficios a largo plazo, como la escalabilidad y facilidad de mantenimiento, justifican su uso.

## Bibliografía

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

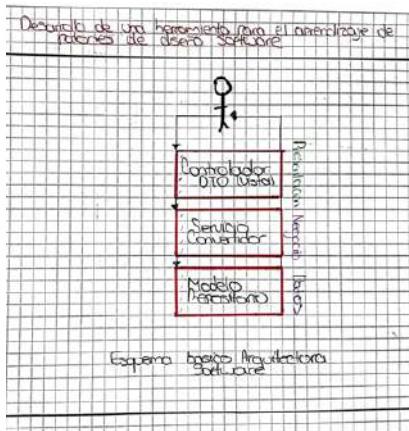


# Desarrollo de una herramienta para el aprendizaje de patrones de diseño software

Los patrones de diseño ofrecen soluciones probadas y eficaces a problemas recurrentes en programación, facilitando la creación de código limpio y fácil de mantener. Estos patrones no sólo resuelven problemas técnicos, sino que también mejoran la comunicación entre los desarrolladores, ya que el conocimiento de un patrón permite comprender rápidamente su estructura y propósito. El uso de patrones de diseño es crucial para crear aplicaciones bien estructuradas y de alta calidad. Este proyecto busca desarrollar una herramienta que facilite el aprendizaje y uso de patrones de diseño. A pesar de los retos ocasionados por el contexto remoto debido al COVID-19, el trabajo resultó en una aplicación funcional, donde se adquirieron conocimientos tanto en Backend con Spring como en Frontend. El prototipo desarrollado puede ser la base para futuras mejoras y evaluaciones por parte de otros estudiantes.

## Bibliografía

Palmero, M. A. S., Martínez, N. S., & Grass, O. Y. R. (2019). Revista Cubana de Ciencias Informáticas, 13(1), 143–157. [http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci\\_arttext](http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci_arttext)

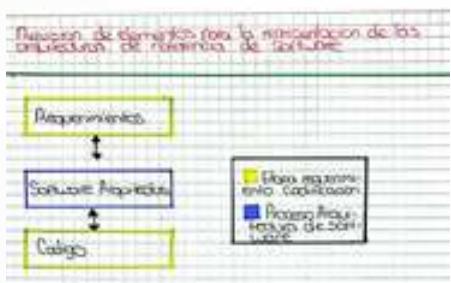


## Reflexión

El desarrollo de una herramienta para aprender patrones de diseño es clave para formar programadores, ya que enseña la importancia de un diseño estructurado. Adoptar patrones permite crear soluciones robustas y escalables, reduciendo problemas de mantenimiento y mejorando la colaboración. A través del aprendizaje de tecnologías como Spring y Frontend, el autor superó barreras y expandió habilidades. A pesar de la pandemia y el trabajo remoto, el proyecto mostró que con determinación y recursos adecuados, es posible obtener resultados satisfactorios. Este prototipo tiene un gran potencial de evolución para futuras mejoras.

# Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software

La arquitectura de software se enfoca en diseñar y organizar sistemas, incluyendo componentes con relaciones entre ellos. Los patrones guían a los desarrolladores hacia objetivos comunes. Los lenguajes de descripción arquitectónica (ADL) como UniCon, Wright, Darwin, Rapide y C2 ayudan a modelar y analizar arquitecturas antes de implementarlas, optimizando la reutilización, dinámica y evolución. Los componentes y conectores son esenciales en las arquitecturas de software, permitiendo un enfoque modular para el diseño, desarrollo y mantenimiento de sistemas complejos. Las arquitecturas de referencia (ARS) son marcos estándar que mejoran la reutilización y adaptabilidad, integrando conceptos clave para prácticas más eficientes en la industria del software. Estas integran conceptos como configuraciones, restricciones, estilos arquitectónicos y servicios, promoviendo prácticas más eficientes en la industria del software.



## Bibliografía

Bastarrica, M. C. (n.d.). Atributos de Calidad y Arquitectura del Software. Upm.Es. Retrieved November 21, 2024, from <http://www.grise.upm.es/rearview/mirror/conferencias/juisic04/Tutoriales/tu4.pdf>

## Reflexión

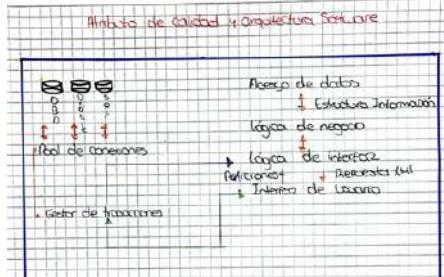
La arquitectura de software conecta las necesidades del negocio con la implementación técnica, ofreciendo una visión modular del sistema. Es clave adaptar herramientas de modelado y usar arquitecturas de referencia para reducir costos y tiempos de desarrollo. No obstante, existen desafíos como la falta de mecanismos para la evolución dinámica y la reutilización en algunos lenguajes. Comprender los componentes y conectores es esencial para crear sistemas sólidos y sostenibles, promoviendo la colaboración interdisciplinaria y enfrentando desafíos en un entorno en constante cambio. Este enfoque es crucial para alcanzar los objetivos del software y resolver problemas de forma efectiva.

# Atributos de Calidad y Arquitectura de Software

En este artículo logramos construir una arquitectura de efectiva, la cual balancear múltiples atributos de calidad (como seguridad, mantenibilidad, portabilidad y rendimiento) sin comprometer desmedidamente otros. Las decisiones tomadas durante el diseño tienen un impacto significativo en el resultado final. Aplicar una metodología que considere las diferentes perspectivas, documente de manera integral y priorice las necesidades de los stakeholders permite crear arquitecturas que sean tanto funcionales como de calidad, aunque no garantiza estos atributos automáticamente. La práctica de este enfoque se valida mediante casos reales expuestos en el curso. Sin embargo, incluso con un diseño sólido, garantizar los atributos de calidad requiere validación constante y adaptabilidad durante todo el ciclo de vida del sistema.

## Bibliografía

De la, C. de I. (n.d.). SABER. Revista Multidisciplinaria del. Redalyc.org. Retrieved November 21, 2024, from <https://www.redalyc.org/pdf/4277/427739438009.pdf>



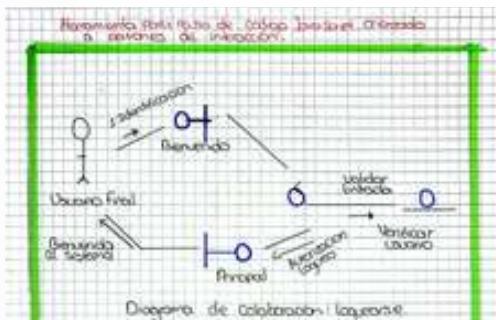
## Reflexión

La arquitectura de software no es solo un conjunto de decisiones técnicas, sino un proceso estratégico que impacta directamente el éxito y la calidad de un sistema. Enfrenta el desafío de equilibrar atributos como rendimiento, seguridad y mantenibilidad, que a menudo entran en conflicto entre sí. Este balance requiere un enfoque sistemático, apoyado en metodologías y herramientas que permitan analizar el sistema desde múltiples perspectivas y documentar de forma clara los compromisos asumidos.

# HERRAMIENTA PARA REUSO DE CÓDIGO

## JAVASCRIPT ORIENTADO A PATRONES DE INTERACCIÓN

El desarrollo de sistemas informáticos es una tarea compleja debido a los diversos problemas que pueden surgir a lo largo de todo el proceso, desde el análisis hasta la implementación. A lo largo del tiempo, han surgido diversas herramientas de diseño rápido de aplicaciones (RAD) que facilitan la creación de interfaces de usuario, como CBuilder, Visual Basic y Delphi. La usabilidad de una interfaz es crucial, ya que determina la efectividad y satisfacción del usuario al interactuar con la aplicación. La reutilización de estos patrones ayuda a acelerar el desarrollo, reducir costos y mejorar la calidad del software. Un ejemplo relevante de la aplicación de esta metodología es ReusMe, una herramienta que permite generar componentes web reutilizables en JavaScript basados en patrones de interacción. Esta aplicación facilita la creación de interfaces gráficas de usuario personalizadas, ofreciendo soluciones probadas y optimizadas para diseñadores y desarrolladores.



## Bibliografía

- Cambarieri, M. G., Difabio, F., & Martínez, N. G. (n.d.). Implementación de una Arquitectura de Software guiada por el Dominio. Edu.Ar. Retrieved November 21, 2024, from <https://sedici.unlp.edu.ar/bitstream/handle/10915/115198/Documento%20completo.pdf-PDFA.pdf?sequence=1&isAllowed=y>

## Reflexión

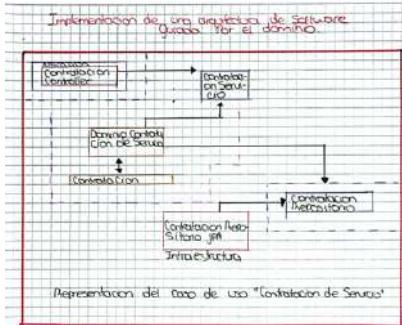
El uso de patrones de interacción en el desarrollo de interfaces de usuario y aplicaciones web mejora la eficiencia, reduce el tiempo de desarrollo y garantiza productos de calidad. Al adoptar patrones reutilizables, los diseñadores evitan reinventar soluciones y aprovechan las mejores prácticas probadas. Herramientas como ReusMe destacan la importancia de la reutilización de código, facilitando la personalización y adaptación de componentes a las necesidades del usuario, sin sacrificar usabilidad. En resumen, la reutilización de patrones y código optimiza el proceso de desarrollo y genera productos más robustos y accesibles.

# Implementación de una Arquitectura de Software guiada por el Dominio

Presenta un enfoque de implementación de arquitecturas de software guiadas por el dominio, enfatizando el diseño dirigido por el dominio (DDD). Se propone transformar una arquitectura de software típica en una arquitectura hexagonal, centrada en el dominio del negocio. Esto permite separar la complejidad del negocio de la lógica técnica, promoviendo una mejor comunicación entre expertos en dominio y desarrolladores. Se discuten los conceptos de contextos delimitados y lenguaje ubicuo, fundamentales para el DDD. Además, se valida el enfoque a través de un caso de estudio sobre una plataforma de empleo, donde se exemplifica cómo aplicar estos principios en un entorno real. La investigación concluye que esta transformación mejora la mantenibilidad y escalabilidad del software.

## Bibliografía

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053–1058. <https://doi.org/10.1145/361598.361623>



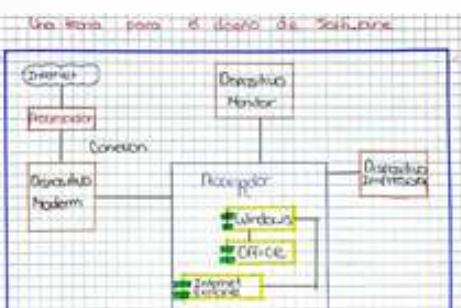
## Reflexión

El enfoque presentado en el artículo es muy relevante en el contexto actual del desarrollo de software, donde la complejidad y la rapidez de cambio son constantes. La adopción de una arquitectura guiada por el dominio no solo facilita una mejor alineación con los objetivos de negocio, sino que también promueve una colaboración más efectiva entre diferentes partes interesadas. La transformación hacia una arquitectura hexagonal, que favorece la independencia de tecnologías, es una estrategia inteligente para asegurar la adaptabilidad a futuros cambios y mejorar la calidad del software. Sin embargo, la implementación de estos conceptos puede ser desafiante y requiere un compromiso significativo por parte de todo el equipo de desarrollo.

# Una Teoría para el Diseño de Software

El diseño de software es esencial porque, al dividir un sistema en elementos que interactúan entre sí, se facilita su desarrollo y mantenimiento. Este enfoque permite asignar funciones a cada componente, establecer sus relaciones y describir su estructura, lo que reduce el costo y la complejidad a lo largo del ciclo de vida del software. El proceso de producción de software se divide en dos etapas principales: desarrollo (33% del esfuerzo total) y mantenimiento (67%). El mantenimiento, que incluye cambios, correcciones y mejoras, suele ser la parte más costosa, especialmente por la necesidad de re-testear el sistema cada vez que se introduce un cambio.

El diseño de software se vuelve crucial para reducir los costos de mantenimiento, ya que anticipa y facilita la incorporación de cambios sin comprometer la integridad del sistema. Un buen diseño permite que los cambios se realicen con menor costo, ya que organiza y descompone el código de manera eficiente.



## Bibliografía

Desarrollo de una arquitectura de software para el robot móvil Lázaro.  
(n.d.). Scielo.Cl. Retrieved November 21, 2024, from [https://www.scielo.cl/scielo.php?pid=S0718-33052018000300376&script=sci\\_arttext](https://www.scielo.cl/scielo.php?pid=S0718-33052018000300376&script=sci_arttext)

## Reflexión

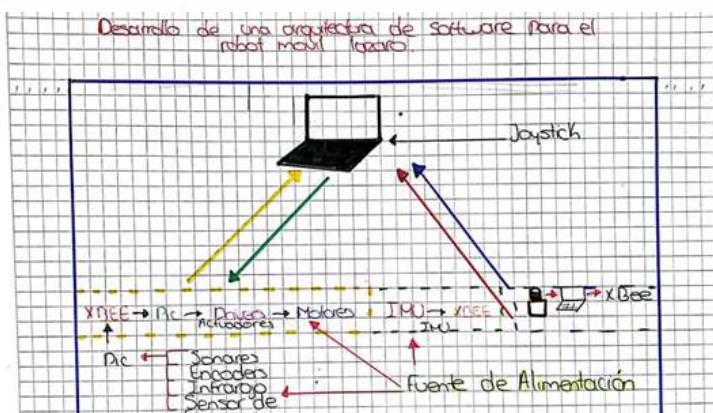
La reflexión sobre el diseño de software nos muestra la importancia de planificar y estructurar adecuadamente un sistema desde el principio. Aunque pueda parecer tentador lanzarse directamente a la programación, sin un diseño previo, los costos y problemas que surgen a lo largo del ciclo de vida del software pueden ser mucho mayores. El principio de "Diseño para el Cambio" nos recuerda que el software no es algo estático; debe ser flexible y evolucionar con el tiempo. En definitiva, el diseño adecuado no es solo una cuestión técnica, sino una estrategia económica que puede hacer la diferencia entre un sistema costoso de mantener y uno que se adapte con facilidad a las necesidades futuras.

# Desarrollo de una arquitectura de software para el robot móvil Lázaro

Se presenta una arquitectura de software para el robot Lázaro, estructurada en tres niveles que optimizan el control de actuadores y el monitoreo de sensores. Utiliza sensores como sonares y telémetros láser para detectar obstáculos y medir fuerzas, y la comunicación con un computador remoto se realiza a través de módulos XBee®. La integración de arquitecturas deliberativas y reactivas, junto con inteligencia artificial, mejora la navegación en entornos no estructurados. La evaluación muestra buena flexibilidad y facilidad de uso, con oportunidades de mejora en el aprendizaje de tareas.

## Reflexión

La propuesta de arquitectura de software para el robot Lázaro representa un avance importante en robótica móvil, combinando diversos sensores y control remoto para ofrecer versatilidad en diferentes entornos. La integración de arquitecturas deliberativas y reactivas, junto con inteligencia artificial, aborda eficazmente los retos en situaciones dinámicas. No obstante, explorar mejoras en el aprendizaje de tareas podría potenciar la autonomía del robot. Este trabajo no solo mejora el desarrollo de Lázaro, sino que también establece un marco para futuras investigaciones en arquitecturas de software para robots móviles.



## Bibliografía

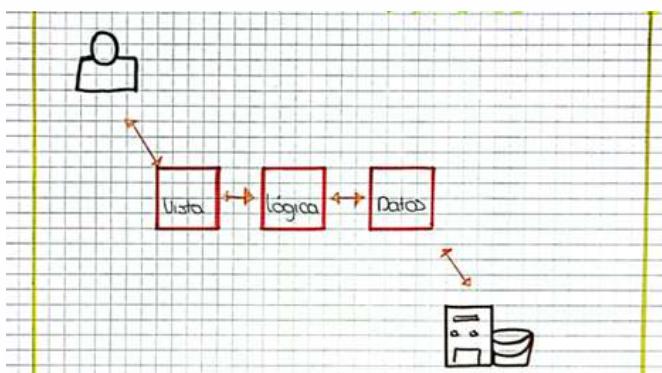
with; P. in C. (n.d.). Arquitectura de software académica para la comprensión del desarrollo de software en capas. Econstor.Eu. Retrieved November 21, 2024, from <https://www.econstor.eu/bitstream/10419/130825/1/837816424.pdf>

# Artículo de software académica para la comprensión del desarrollo de software en capas

Este artículo presenta un modelo académico de arquitectura de software con enfoque en capas, esencial para prevenir problemas a corto y largo plazo. Propone una estructura de tres capas (Vista, Lógica y Datos) y una más detallada de siete capas, optimizando el desarrollo de sistemas. Utilizando el patrón Modelo-Vista-Controlador (MVC), se facilita la separación de responsabilidades, mejorando la reutilización y mantenimiento del código. Además, describe clases que gestionan operaciones sobre la base de datos. Concluye que la adopción de arquitecturas de software es clave para sistemas flexibles, sugiriendo mejoras como la automatización de mapeadores y control de transacciones.

## Reflexión

El texto resalta la importancia de las arquitecturas de software en capas, utilizando el patrón Modelo-Vista-Controlador (MVC) para organizar y mejorar la reutilización y el mantenimiento del código. Presenta ejemplos prácticos, como las clases Cliente y Teléfono Datos, para ilustrar su aplicación. Aunque es un enfoque sólido, se sugiere profundizar en la automatización de mapeadores y el control de transacciones para mejorar la flexibilidad del sistema. En general, el texto subraya cómo una arquitectura bien estructurada favorece la creación de sistemas más sostenibles y fáciles de mantener.



## Bibliografía

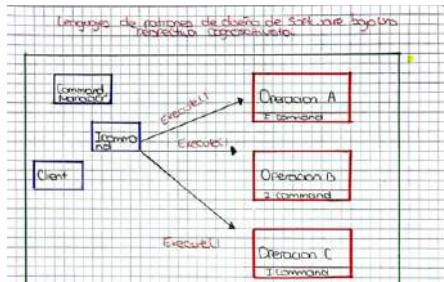
- Calderón Castro, A. (2003). Lenguajes de patrones de diseño de software bajo una perspectiva cognoscitivista. *InterSedes*, IV(7), 167-189. <https://www.redalyc.org/articulo.oa?id=66640712>

# Lenguajes de patrones de diseño de software bajo una perspectiva cognoscitivista.

Este artículo nos ayuda analizar la aplicación del concepto de "lenguaje de patrones" en el diseño de software, basándose en las ideas de Alexander (1979) y su relación con la estructura de sistemas de patrones. Aunque el término "lenguaje de patrones" se ha utilizado ampliamente, aún persisten dudas teóricas y prácticas sobre su significado y aplicación. El trabajo propone un marco teórico cognoscitivo para interpretar y mejorar el uso de estos lenguajes. Se establece una distinción clara entre los conceptos de "lenguaje de patrones", "estructura de lenguaje de patrones", "catálogos de patrones" y "sistemas de patrones". Además, se sugiere que el diseño de un lenguaje de patrones debe enfocarse en su organización y evolución para mejorar su eficacia en el campo del desarrollo de software.

## Bibliografía

Luna-García, H., Mendoza-González, R., & Álvarez-Rodríguez, F.-J. (2015). Patrones de diseño para mejorar la accesibilidad y uso de aplicaciones sociales para adultos mayores. *Comunicar*, XXII(45), 85-94.  
<https://www.redalyc.org/articulo.oa?id=15839609009>



## Reflexión

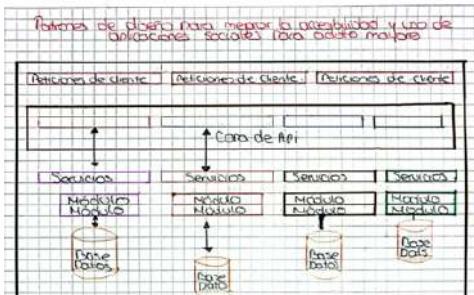
Este artículo destaca la complejidad y la importancia de los lenguajes de patrones en el diseño de software, subrayando que, aunque ampliamente utilizados, aún carecen de una base teórica sólida, lo que dificulta su implementación efectiva. La propuesta de un marco cognoscitivo para organizar y mejorar estos lenguajes responde a la necesidad de estructurar mejor el conocimiento sobre patrones. Además, el enfoque en la evolución de estos patrones sugiere que su aplicabilidad puede mejorar continuamente, avanzando en la ingeniería de software. El artículo también señala nuevas direcciones para futuras investigaciones, enfatizando la importancia de seguir innovando y clarificando estos conceptos.

# Patrones de diseño para mejorar la accesibilidad y uso de aplicaciones sociales para adultos mayores.

Este artículo propone un conjunto incompleto de 36 modelos para el diseño de interacción en aplicaciones sociales dirigidas a personas mayores. Basado en normativas y directrices previas, busca identificar y resolver problemas de usabilidad en estas interfaces. Al integrar descripciones de anomalías y soluciones alternativas, se fortalece la aplicación de estos modelos para los diseñadores. Utilizando técnicas de evaluación heurística, se recopila la percepción del usuario para mejorar el diseño. El estudio aborda el tema desde dos perspectivas: técnica y social, demostrando que el modelo facilita la creación de interfaces accesibles y mejora la calidad de vida de los usuarios mayores.

## Bibliografía

Lagunes García, G., López Martínez, I., Peláez Camarena, G. S., Abud Figueroa, M. A., & Olivares ReCIBE. Revista Electrónica de Computación, Informática, Biomédica y Electrónica, 1. <https://www.redalyc.org/articulo.oa?id=512251501004>

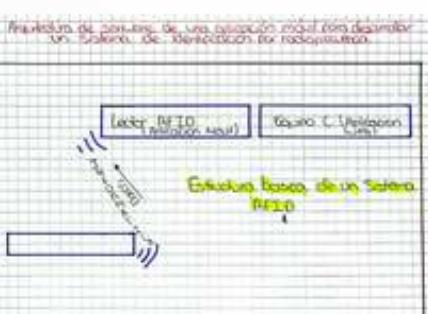


## Reflexión

Este artículo destaca la importancia de crear interfaces tecnológicas accesibles y usables para personas mayores, un grupo que enfrenta barreras cognitivas y físicas al interactuar con aplicaciones sociales. Al combinar modelos de interacción y técnicas de evaluación heurística, el enfoque aborda tanto las necesidades técnicas como sociales, promoviendo la adopción tecnológica y mejorando la calidad de vida de los usuarios mayores. Además, este enfoque no solo mejora la experiencia de usuario, sino que también favorece el bienestar emocional y social, promoviendo la conectividad y participación en la sociedad digital. El artículo subraya la necesidad de continuar desarrollando diseños inclusivos que permitan a las personas mayores aprovechar los beneficios tecnológicos, fomentando su empoderamiento y reduciendo la exclusión digital.

# Arquitectura de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia

Los sistemas de identificación por radiofrecuencia (RFID) se utilizan para identificar objetos automáticamente, y en este caso, se está desarrollando un sistema RFID para mejorar el control de inventarios en el Instituto Tecnológico de Orizaba. Actualmente, el sistema de inventarios del instituto presenta problemas como la falta de concordancia entre los activos registrados y los realmente existentes, así como un tiempo de actualización lento. El sistema RFID propuesto busca resolver estos problemas al disminuir el tiempo de registro, actualización y localización de los activos. La arquitectura de software del sistema incluye una base de datos XML para integrar la aplicación web y la aplicación RFID, y un módulo RFID que envía señales de radiofrecuencia para detectar objetos etiquetados. La implementación de este sistema incrementa la seguridad y el control de los bienes, y permite mantener el inventario actualizado en tiempo real, evitando errores humanos y robos.



## Bibliografía

Scientia Et Technica. (n.d.). Redalyc.org. Retrieved November 21, 2024, from <https://www.redalyc.org/pdf/849/84933912003.pdf>

## Reflexión

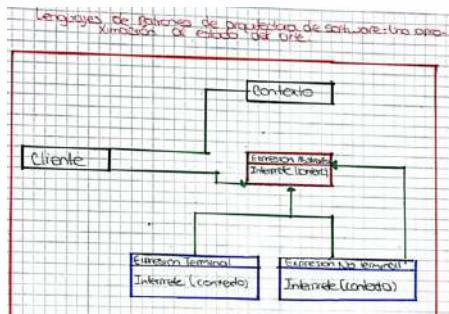
El uso de tecnología RFID para el control de inventarios en instituciones como el Instituto Tecnológico de Orizaba mejora significativamente la eficiencia y precisión en la gestión de activos. Este sistema facilita la actualización, consulta y localización de bienes de manera rápida y precisa, reduciendo los errores humanos. La integración con aplicaciones web y móviles muestra cómo la automatización transforma procesos tradicionales, mejorando la seguridad y eficiencia. Además, contribuye a una mejor organización interna, optimizando los recursos institucionales. Aunque la implementación requiere inversión y capacitación, esta innovación es clave para un modelo de gestión más moderno y eficiente en el ámbito educativo y organizacional.

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte

Examina el estado actual de los lenguajes de patrones en la arquitectura de software. Se analizan sus orígenes, avances y aplicaciones en la construcción de arquitecturas en diversos dominios, destacando su importancia para diseñadores y desarrolladores. Se revisa la evolución de la ingeniería de software desde los años 60, enfatizando la necesidad de un enfoque arquitectónico para mejorar la calidad y mantenibilidad del software. El artículo menciona a Christopher Alexander y Frank Buschmann, quienes contribuyeron al desarrollo de patrones en la arquitectura civil y de software, respectivamente. Se presentan diferentes formas de categorizar patrones y ejemplos de su aplicación en áreas como la gestión de identidades y el desarrollo de aplicaciones e-business. Se concluye que los lenguajes de patrones son herramientas valiosas para resolver problemas arquitectónicos de manera eficiente y se sugiere investigar más sobre metodologías en su creación.

## Bibliografía

Desarrollo de sistemas de software con patrones de diseño orientado a objetos. (n.d.). Edu.Pe. Retrieved November 21, 2024, from <https://cybertesis.unmsm.edu.pe/backend/api/core/bitstreams/92b01647-3651-4e75-9fd1-b00cb53826b1/content>

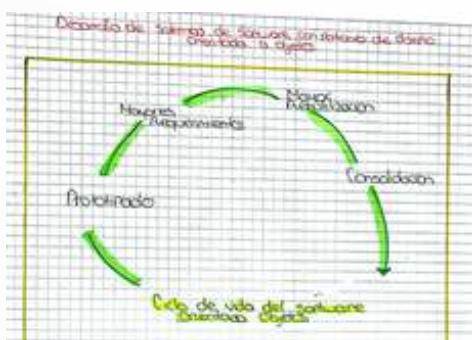


## Reflexión

El artículo destaca la evolución de los lenguajes de patrones en arquitectura de software, desde sus orígenes en la arquitectura civil hasta su adaptación en software por Frank Buschmann. Presenta ejemplos prácticos en e-business y gestión de identidades, mostrando su versatilidad. También invita a investigar metodologías para su creación, subrayando la importancia de estos patrones como soluciones eficientes para problemas complejos.

# Desarrollo de sistemas de software con patrones de diseño orientado a objetos

Aborda la implementación de patrones de diseño orientados a objetos en la creación de un sistema de software denominado "Intranet Industrial", desarrollado en la Facultad de Ingeniería Industrial de la Universidad Nacional Mayor de San Marcos. Se analiza el impacto económico de la adopción de estos patrones en comparación con un enfoque que no los considera. En la primera sección, se introducen conceptos teóricos y se clasifican los patrones más reconocidos en el ámbito industrial. A continuación, se detallan los patrones aplicados en el desarrollo del sistema y se presentan de manera formal. La sección final ofrece una descripción del sistema, sus módulos y herramientas, así como una estimación de costos utilizando el patrón "Informador" y el método COCOMO. Se concluye que la implementación de patrones contribuye a mejorar la eficiencia y a disminuir los costos en el desarrollo de software.



## 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), 85-95. <https://doi.org/10.33304/revinv.v17n1-2022007>

## Reflexión

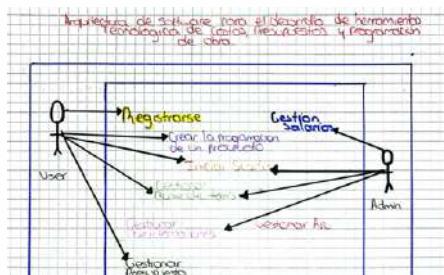
El enfoque de utilizar patrones de diseño en el desarrollo de software es una estrategia altamente efectiva que no solo optimiza el proceso de creación, sino que también promueve la reutilización y la escalabilidad. Este trabajo es un excelente ejemplo de cómo integrar teoría y práctica, mostrando que la implementación de patrones no solo es beneficiosa desde el punto de vista técnico, sino que también tiene repercusiones positivas en la gestión de costos. La investigación resalta la importancia de adoptar estándares en el desarrollo de sistemas complejos, ofreciendo un modelo que puede servir de referencia para futuros proyectos en el ámbito académico e industrial.

# Arquitectura de software para el desarrollo de herramienta Tecnológica de Costos, Presupuestos y Programación de obra

El artículo de Cárdenas-Gutiérrez, Barrientos-Monsalve y Molina-Salazar aborda el diseño de un software académico para la gestión de costos y presupuestos en Ingeniería Civil. Organiza la información sobre materiales, mano de obra, maquinaria y transporte en grupos de procesos, mejorando la productividad y facilitando la elaboración de presupuestos. Utiliza una Estructura de División del Trabajo (EDT) y la ruta crítica para la programación de obras. El software busca modernizar la enseñanza y ser una herramienta accesible y gratuita para estudiantes, preparándolos mejor para el mercado laboral. Los autores no tienen conflictos de intereses.

## Bibliografía

Ospina Torres, M. H., & Luna, C. L. (2013). Una arquitectura basada en software libre para archivos web. Enl@ce, 10(1), 53-72.  
<https://www.redalyc.org/articulo.oa?id=82326270005>



## Reflexión

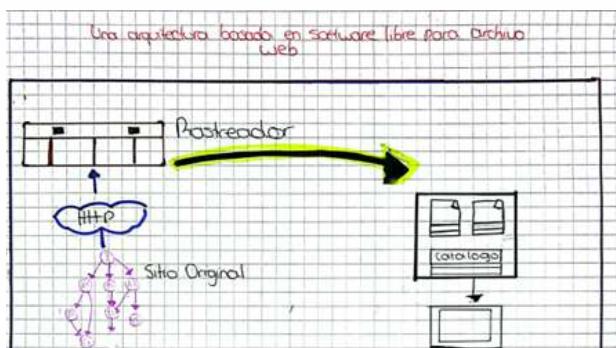
Me parece interesante porque propone un software práctico y accesible, pensado para facilitar la gestión de costos, presupuestos y programación. Me gusta que utilice metodologías como la EDT y el método de la ruta crítica, ya que ayudan a conectar la teoría con la práctica. Además, que sea gratuito lo hace aún más valioso. Solo me pregunto cómo se asegura su actualización y si ya ha sido probado con usuarios.

# Una arquitectura basada en software libre para archivos web

Este artículo aborda la necesidad de preservar la información web, destacando su importancia para proteger el patrimonio digital. Señala que, a pesar de la extensa información disponible en la web, la preservación de sitios web sigue siendo un desafío, especialmente en Latinoamérica y Venezuela, donde hay una falta de iniciativas. Se propone diseñar una arquitectura de preservación web utilizando software libre, basándose en un estudio global de métodos y herramientas. Se está desarrollando un prototipo para evaluar la integración y el rendimiento de estos componentes. El objetivo es proporcionar un marco que facilite la implementación de sistemas de preservación web sostenibles en la región.

## Reflexión

Este artículo aborda la falta de iniciativas de preservación web en Latinoamérica, especialmente en Venezuela. Se propone una arquitectura basada en software libre para garantizar la conservación de sitios web. Tras un estudio global de métodos y herramientas, se seleccionaron las más adecuadas para crear un prototipo que evalúe la integración y rendimiento. La preservación web es clave para el acceso continuo a recursos digitales y la protección del patrimonio cultural, buscando establecer un sistema efectivo y sostenible en la región.



## Bibliografía

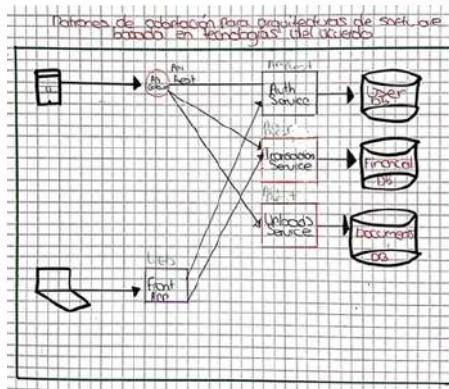
- Pérez-Sotelo, J. S., Cuesta, C. E., & Ossowski, S. (2011). Patrones de adaptación para arquitecturas de software basadas en tecnologías del acuerdo. REICIS. Revista Española de Innovación, Calidad e Ingeniería Del Software, 7(3), 6-25. <https://www.redalyc.org/articulo.oa?id=92222551003>

# Patrones de adaptación para arquitecturas de software basadas en tecnologías del acuerdo

Los sistemas actuales son cada vez más complejos y requieren un enfoque adaptativo para gestionar su evolución. La autoadaptación, que no solo abarca el comportamiento funcional, sino también las propiedades no-funcionales, es esencial en estos sistemas. Los Sistemas Multi-Agente (SMA) han surgido como una solución para resolver problemas complejos, pero aún presentan limitaciones en cuanto a su auto-adaptación. La propuesta presentada se basa en un ecosistema de servicios con Tecnologías del Acuerdo (AT), donde los agentes se organizan y coordinan dinámicamente, formando organizaciones adaptativas. Estas organizaciones evolucionan mediante reglas de adaptación y patrones predefinidos, alcanzando acuerdos estables. La plataforma THOMAS es la base para este enfoque, que busca desarrollar una arquitectura adaptativa a través de la evolución de los agentes y sus interacciones. El futuro del proyecto incluye la implementación de nuevas variantes para mejorar la adaptabilidad de los sistemas.

## Reflexión

La auto-adaptación es clave para sistemas modernos, ofreciendo flexibilidad frente a cambios. El enfoque basado en agentes y acuerdos emergentes permite la evolución sin intervención humana. Aunque la autoadaptación total aún se desarrolla, integrar patrones de adaptación en plataformas como THOMAS es esencial para crear arquitecturas adaptativas. Esto mejora la eficiencia y permite ajustar los sistemas a nuevas demandas sin perder estabilidad. La experimentación en plataformas como ATMAS será fundamental para lograr sistemas completamente auto-adaptativos.



## Bibliografía

- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (2017, September). Selección de metodologías ágiles e integración de arquitecturas de software en el desarrollo de sistemas de información. In XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017, ITBA, Buenos Aires). <https://n9.cl/53lyw>

# ¿CÓMO EMPLEAR EL SOFTWARE LIBRE EN LA ARQUITECTURA Y EL DISEÑO?

Este trabajo expone el uso del software libre en disciplinas fuera de la informática, específicamente en Arquitectura y Diseño. El enfoque destaca cómo estas herramientas pueden mejorar la calidad de los trabajos realizados con recursos limitados. Además, se presenta una guía básica para quienes se inician en el uso de software libre, orientada a aquellos sin conocimientos previos, sugiriendo programas que pueden sustituir a software propietario. El autor muestra ejemplos prácticos que demuestran las capacidades y ventajas de estas herramientas en áreas como el diseño arquitectónico e industrial. El software libre fomenta el intercambio de ideas y la creación de modelos de trabajo que promueven la libertad y creatividad en el proceso de diseño.

## Bibliografía

Crespo, V. (2007). CÓMO EMPLEAR EL SOFTWARE LIBRE EN LA ARQUITECTURA Y EL DISEÑO. AU. Arquitectura y Urbanismo, XXVIII(2), 92-95. <https://www.redalyc.org/articulo.oa?id=376839852016>

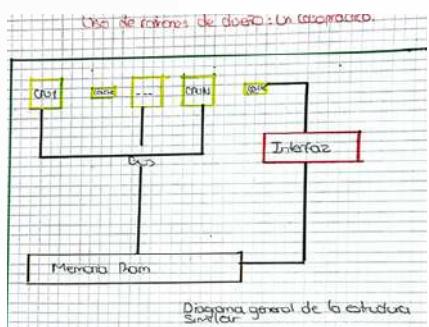
## Reflexión

El software libre transforma áreas como la Arquitectura y el Diseño al ser accesible y modificable, permitiendo que más personas participen en el proceso creativo sin barreras económicas. Fomenta la innovación, la colaboración y un desarrollo sostenible alineado con valores éticos. Al eliminar la dependencia de software propietario, promueve un diseño inclusivo y democrático, abriendo espacio para diversas soluciones creativas y enfoques nuevos.

¿Cómo emplear el software libre en la arquitectura y el diseño?	
Alternativas web	Software libre
• Navegadores web (Internet Explorer)	• Firefox, Mozilla, Firefox
• Clientes correo (Outlook)	• Thunderbird, Kmail
• Programas de gestión de fotografías (MS PhotoShop)	-
Alternativas para:	Software libre
• Adobe Photoshop, Paint	Gimp, Krita, ImageMagick
Mantenimiento	Software libre
Tecnología CAD (AutoCAD)	Blender, GIMP, Recad

# USO DE PATRONES DE DISEÑO: UN CASO PRÁCTICO

Los patrones de diseño son soluciones clave en la formación de ingenieros de software. Gómez, Jiménez y Arroyo (2009) defienden su inclusión en la educación básica de ciencias de la computación. En la Escuela de Ciencias de la Computación e Informática, los estudiantes aplican estos patrones en proyectos web. Además, en el curso de Arquitectura de Computadoras, se amplió su uso mediante la simulación de una computadora funcional. El artículo presenta el marco teórico, metodología, resultados y posibles líneas de investigación futura.



## Reflexión

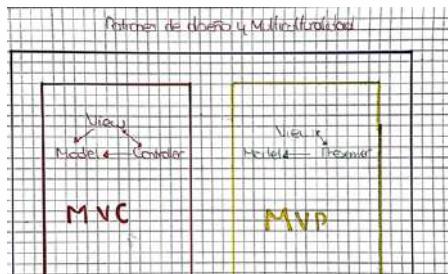
El uso de patrones de diseño en proyectos académicos no solo optimiza el desarrollo de software, sino que también proporciona a los estudiantes herramientas valiosas para enfrentar problemas complejos de manera estructurada y eficiente. Integrar estos patrones en el currículo universitario, como sugiere la investigación, fomenta una comprensión profunda de buenas prácticas en programación, algo esencial para los futuros profesionales del área. Además, al aplicar patrones en contextos más amplios, como el proyecto de simulación de una computadora, los estudiantes experimentan cómo estas soluciones pueden escalar y adaptarse a diferentes tipos de sistemas. Esto no solo mejora su capacidad técnica, sino que también refuerza su creatividad y capacidad de adaptación. Con un enfoque práctico y continuo, los patrones de diseño pueden transformar la manera en que los estudiantes abordan el desarrollo de software, permitiéndoles crear soluciones más robustas y fácilmente mantenibles.

## Bibliografía

- Salazar Bermúdez, G., Montenegro Jiménez, I., & Rodríguez Rodríguez, L. (2013). USO DE PATRONES DE DISEÑO: UN CASO PRACTICO. Revista Ingeniería, 22(2), 45-59. <https://doi.org/10.15517/ring.v22i2.8220>

# Patrones de diseño y multiculturalidad

La integración de las TICs ha transformado áreas como el comercio y la educación, promoviendo la multiculturalidad. En este contexto, la Interacción Humano-Computador (HCI) busca mejorar las interfaces respetando la diversidad cultural. Los patrones de diseño, adaptados de la arquitectura, ayudan a resolver problemas recurrentes en el desarrollo de software. Este artículo destaca la importancia de integrar principios multiculturales en el diseño web para mejorar la experiencia de usuarios de diferentes contextos.



## Reflexión

La multiculturalidad en el diseño web es tanto una necesidad técnica como ética. Con la globalización, los diseñadores deben adaptar sus sistemas para ser inclusivos y respetar la diversidad cultural. Integrar patrones de diseño multiculturales mejora la accesibilidad y efectividad de las aplicaciones, especialmente para usuarios globales. Esto no solo optimiza la interfaz, sino que también enriquece la experiencia del usuario. Promover un lenguaje común basado en estos patrones facilita soluciones más cohesivas y culturalmente sensibles, lo que incrementa la satisfacción y fidelidad del usuario.

## Bibliografía

- González, I. (2012). Patrones de diseño y multiculturalidad. *Prisma Tecnológico*, 3(1), 31-34. <https://revistas.utp.ac.pa/index.php/prisma/article/view/543>

# **La Arquitectura de Software en el Proceso de Desarrollo: Integrando MDA al Ciclo de Vida en Espiral**

La arquitectura dirigida por modelos (MDA) propuesta por la OMG se basa en el uso de modelos en diferentes niveles de abstracción (CIM, PIM, PSM) para guiar el desarrollo de software y generar automáticamente código desde esos modelos. Este enfoque promueve la transformación de modelos como mecanismo para lograr este paso entre niveles. El modelo en espiral, por su parte, es un enfoque evolutivo que incluye análisis de riesgos en sus fases de planificación, ingeniería y evaluación del cliente. El artículo aborda dos objetivos principales: analizar el grado de participación de la arquitectura del software en el ciclo de vida en espiral y evaluar el impacto de MDA en los riesgos asociados. Uno de los problemas del ciclo de vida tradicional es la pérdida de trazabilidad entre artefactos generados en distintas etapas, lo que MDA soluciona mediante transformaciones automáticas entre modelos, garantizando la trazabilidad y mejorando la gestión de riesgos. Además, MDA facilita la adaptación a cambios tecnológicos sin afectar los modelos de alto nivel.

## **Reflexión**

MDA (Model-Driven Architecture) mejora la gestión de riesgos y cambios en el desarrollo de software al permitir la transformación continua de modelos y la generación automática de código. Su principal ventaja es mantener la coherencia entre fases del ciclo de vida, adaptándose fácilmente a nuevos requisitos sin afectar la estructura del software. Aunque su implementación requiere un conocimiento profundo de herramientas y metodologías, MDA promete hacer los proyectos de software más ágiles, flexibles y resistentes a cambios.

## **Bibliografía**

La Arquitectura de Software en el Proceso de Desarrollo: Integrando MDA al Ciclo de Vida en Espiral. (n.d.). Edu.Ar. Retrieved November 21, 2024, from <https://revistas.unla.edu.ar/software/article/view/103>

# Evaluación de una Arquitectura de Software

El sistema de evaluación y calificación de las Direcciones Territoriales de Salud en Colombia, regulado por la Resolución 4505 de 2012, enfrenta desafíos en la gestión de grandes volúmenes de datos sobre la población no asegurada, generando errores en los registros. La Secretaría Municipal de Salud de Villavicencio (SMSV) valida manualmente los archivos, lo que no asegura la integridad de la información. Se propone un aplicativo que valide y reporte errores, mejorando la calidad de los informes y generando alarmas para el seguimiento de citas y procedimientos. El uso del patrón MVC garantiza flexibilidad y adaptación a futuros cambios, y el sistema podría replicarse en otras secretarías de salud con ajustes específicos.

## Bibliografía

- Agudelo, O., Sanabria, F. R., & Rodríguez, S. V. (2021). Evaluación de una Arquitectura de Software. *Prospectiva*, 19(2). <https://doi.org/10.15665/rp.v19i2.2636>

Evaluación de una arquitectura de software

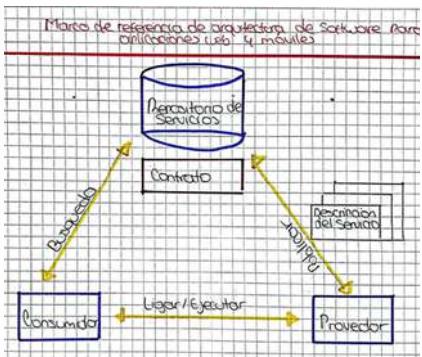
Atómico	Resumen
3. Presentación de la TA	4. Identificar enfoques arquitectónicos
5. Generar árbol de utilidad de criterios de calidad	6. Analizar enfoques arquitectónicos
7. Jiribilla de ideas y priorizar enfoques	Analizar enfoques arquitectónicos
Presentar resultados	

## Reflexión

Un sistema automatizado para la validación de datos en salud mejora la precisión y eficiencia en la gestión de información, evitando los errores del proceso manual. La implementación del patrón MVC y el Sprint cero asegura flexibilidad y adaptación a futuros cambios. Este sistema optimiza la calidad de los informes y facilita decisiones más confiables, beneficiando no solo a la SMSV, sino también a otras entidades de salud, mejorando la eficiencia en la atención pública.

# Marco de referencia de arquitectura de software para aplicaciones web y móviles

La movilidad en aplicaciones corporativas presenta desafíos de costo, pero se puede abordar mediante una plataforma que gestione la interfaz y la integración de manera segura y escalable. La arquitectura Cliente/Servidor, junto con POO y el patrón MVC, facilita el desarrollo eficiente. El uso de software libre y de código abierto reduce costos de licencias y servidores, mientras que herramientas como ExtJS y Sencha Touch permiten crear aplicaciones modulares, escalables y personalizables, optimizando recursos y tiempos de respuesta.



## Reflexión

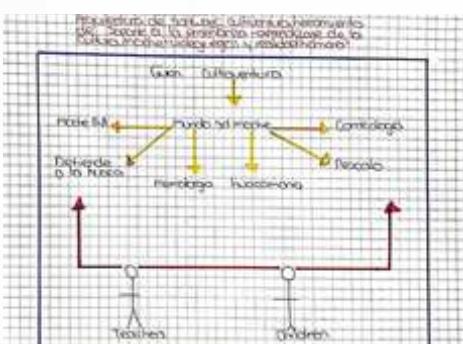
El desarrollo de aplicaciones web y móviles en entornos corporativos debe enfocarse en la funcionalidad, integración y optimización de recursos para garantizar escalabilidad. La arquitectura Cliente/Servidor y patrones como MVC facilitan un desarrollo eficiente y un mantenimiento sostenible. El uso de software libre y de código abierto no solo reduce costos, sino que ofrece flexibilidad e innovación, adaptándose a las necesidades cambiantes del entorno. La modularidad, portabilidad y mejora en tiempos de respuesta son esenciales para sistemas de alto rendimiento. En un contexto interconectado y móvil, garantizar seguridad y optimización de recursos es clave para aplicaciones que escalen con las demandas del negocio, convirtiéndose en un factor crucial para el éxito empresarial a largo plazo.

## Bibliografía

Babahoyo, I. T. S., Maliza Martínez, C. A., López Mendizábal, V. L., Babahoyo, I. T. S., Mackliff Peñafiel, V. V., & Babahoyo, I. T. S. (2016). Marco de referencia de arquitectura de software para aplicaciones web y móviles. *Journal of Science and Research*, 1(CITT2016), 72-75. <https://doi.org/10.26910/issn.2528-8083vol1isscitt2016.2016pp72-75>

# **Arquitectura de software Culti Ventura, herramienta de soporte a la enseñanza-aprendizaje de la cultura Moche “Videojuegos y realidad humana”**

El artículo describe la arquitectura de software de Cultiventura, una herramienta educativa que utiliza tecnologías de videojuego y realidad aumentada para enseñar sobre la cultura Moche. El sistema se diseñó con un enfoque ágil, permitiendo ajustes continuos a los requisitos funcionales y no funcionales. Esta arquitectura facilita la integración de conceptos culturales y la creación de recursos interactivos para mejorar el aprendizaje. Además, el proyecto se alinea con las políticas educativas de Perú, promoviendo la diversidad cultural. El modelo de desarrollo ágil optimiza la gestión de los componentes del sistema, asegurando calidad y reduciendo riesgos.



## **Bibliografía**

“Arquitectura de software Cultiventura, herramienta de soporte a la enseñanza-aprendizaje de la cultura Moche Videojuegos y realidad humana.” (n.d.). Surl.Li. Retrieved November 21, 2024, from <http://surl.li/nuluns>

## **Reflexión**

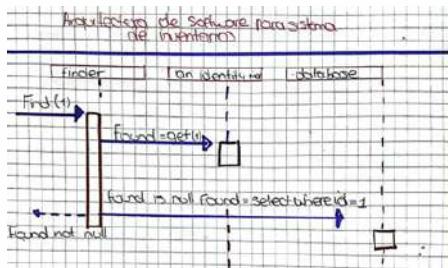
El desarrollo de Cultiventura muestra cómo la tecnología, específicamente los videojuegos y la realidad aumentada, puede ser fundamental para preservar y transmitir la identidad cultural. Al integrar elementos culturales de forma interactiva, esta herramienta no solo promueve el aprendizaje, sino también la revalorización de culturas locales, como la Moche. El enfoque ágil utilizado en su desarrollo facilita una rápida adaptación a cambios y necesidades del usuario, lo que resulta crucial en el ámbito educativo. Además, la sistematización de los recursos garantiza que el software sea accesible, funcional y de calidad, asegurando una experiencia de aprendizaje efectiva. Cultiventura es un claro ejemplo de cómo la tecnología puede enriquecer el proceso educativo y contribuir a la conservación de tradiciones culturales.

# Arquitectura de Software para Sistema Gestión de Inventarios.

Este trabajo presenta el diseño de la arquitectura de software para un Sistema de Gestión de Inventario y Almacén. El sistema es modular, flexible y enfocado en mejorar la eficiencia empresarial en la economía cubana. Utilizando la metodología RUP, se define una arquitectura que cumple con los requisitos de funcionalidad y fiabilidad, permitiendo la reutilización de componentes. El sistema es parametrizable y adaptable a diversas necesidades, buscando automatizar procesos y optimizar el control de inventarios en las empresas cubanas.

## Bibliografía

Arquitectura de Software para Sistema Gestión de Inventarios. (n.d.). Uci.Cu. Retrieved November 21, 2024, from [https://repositorio.uci.cu/jspui/handle/ident/TD 0201\\_07](https://repositorio.uci.cu/jspui/handle/ident/TD 0201_07)



## Reflexión

La arquitectura de software es crucial para el éxito de cualquier sistema, ya que define su estructura base. En el caso del sistema de gestión de inventarios en Cuba, destaca la importancia de una planificación adecuada y el uso de enfoques ágiles y modulares. Esto facilita la adaptación a cambios económicos y tecnológicos, mejorando la calidad, escalabilidad y competitividad del sistema. Un buen diseño arquitectónico es clave para enfrentar desafíos en entornos dinámicos.

# Introducción a los Patrones de Diseño

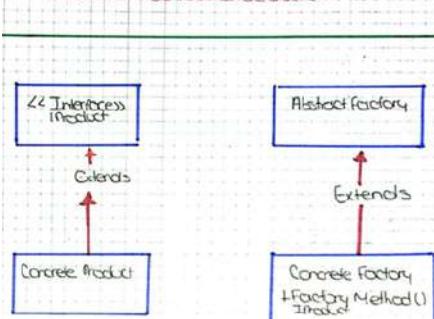
Los patrones de diseño tienen su origen en la arquitectura de edificios, iniciados por Christopher Alexander en su libro *Timeless Way of Building* en 1979, donde presentó patrones como soluciones a problemas recurrentes en el diseño arquitectónico. Más tarde, en *A Pattern Language*, formalizó estos patrones para ser aplicados en situaciones de diseño. Estos patrones no son abstractos ni específicos de una cultura o situación, sino que proporcionan soluciones aplicables en diversos contextos. En 1987, Ward Cunningham y Kent Beck adoptaron estos conceptos para la programación orientada a objetos, creando patrones de interacción hombre-máquina. Sin embargo, fue en los años 90 cuando los patrones de diseño tuvieron su mayor impacto en informática con el libro *Design Patterns* de los autores conocidos como Gang of Four (GoF), donde se presentaron 23 patrones comunes. Los patrones de diseño ofrecen soluciones reutilizables a problemas recurrentes, lo que facilita su implementación en diversos contextos y su aplicación en distintos sistemas.

## Reflexión

Los patrones de diseño en software, inspirados en la arquitectura, mejoran la organización, reutilización y flexibilidad del código. Permiten a los desarrolladores enfrentar problemas de manera eficiente, creando software más modular, escalable y fácil de mantener. La reutilización de soluciones probadas optimiza el desarrollo, aumentando la productividad y calidad. Además, su capacidad de adaptación a nuevos requerimientos, como la conexión a diferentes bases de datos, demuestra su versatilidad. En resumen, los patrones de diseño ofrecen un marco probado que beneficia tanto a desarrolladores como a organizaciones, favoreciendo la creación de software de alta calidad y arquitecturas robustas.

## Bibliografía

Introducción a los Patrones de Diseño. (n.d.) ..N9.Cl. Retrieved November 21, 2024, from <https://n9.cl/ti4ix>



# Desarrollo de sistemas de software con patrones de diseño orientado a objetos

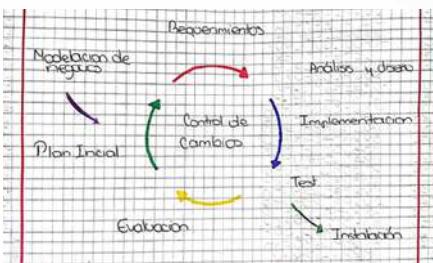
Aborda la implementación de patrones de diseño orientados a objetos en la creación de un sistema de software denominado "Intranet Industrial", desarrollado en la Facultad de Ingeniería Industrial de la Universidad Nacional Mayor de San Marcos. Se analiza el impacto económico de la adopción de estos patrones en comparación con un enfoque que no los considera. En la primera sección, se introducen conceptos teóricos y se clasifican los patrones más reconocidos en el ámbito industrial. A continuación, se detallan los patrones aplicados en el desarrollo del sistema y se presentan de manera formal. La sección final ofrece una descripción del sistema, sus módulos y herramientas, así como una estimación de costos utilizando el patrón "Informador" y el método COCOMO. Se concluye que la implementación de patrones contribuye a mejorar la eficiencia y a disminuir los costos en el desarrollo de software.

## Bibliografía

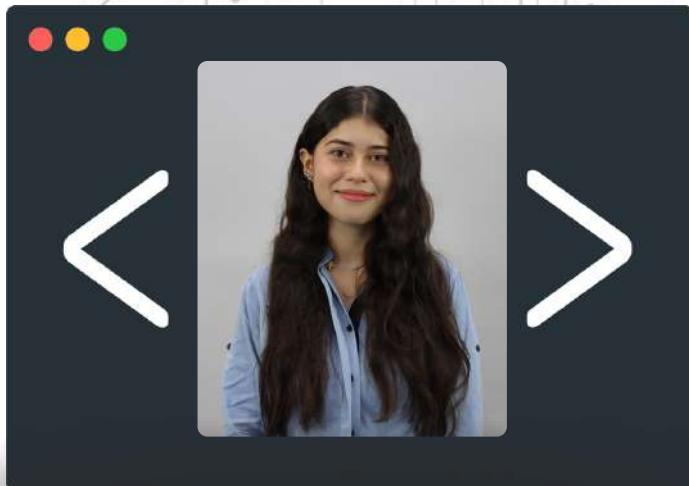
Desarrollo de sistemas de software con patrones de diseño orientado a objetos. (n.d.). Edu.Pe. Retrieved November 21, 2024, from <https://cybertesis.unmsm.edu.pe/backend/api/core/bitstreams/92b01647-3651-4e75-9fd1-b00cb53826b1/content>

## Reflexión

El enfoque de utilizar patrones de diseño en el desarrollo de software es una estrategia altamente efectiva que no solo optimiza el proceso de creación, sino que también promueve la reutilización y la escalabilidad. Este trabajo es un excelente ejemplo de cómo integrar teoría y práctica, mostrando que la implementación de patrones no solo es beneficiosa desde el punto de vista técnico, sino que también tiene repercusiones positivas en la gestión de costos. La investigación resalta la importancia de adoptar estándares en el desarrollo de sistemas complejos, ofreciendo un modelo que puede servir de referencia para futuros proyectos en el ámbito académico e industrial.



# Anyi Zujey Gómez Casanova



Me considero una persona amable, buena amiga y responsable. Siempre busco estar ahí para las personas que me rodean y brindarles apoyo cuando lo necesitan.

Aunque reconozco que a veces me cuesta un poco ser paciente, trato de mejorar cada día en ese aspecto.

Me esfuerzo por ser una persona de confianza y comprometida, cumpliendo con mis responsabilidades y aprendiendo constantemente en el proceso.

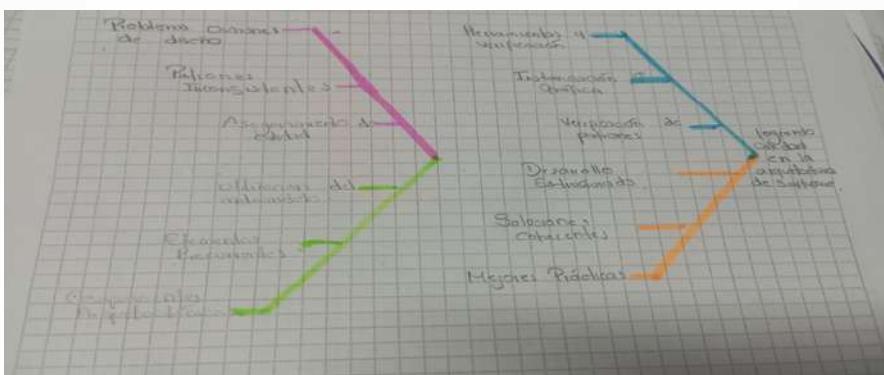
Me gusta jugar al voleibol y, además, tengo conocimientos en inglés, lo que me permite comunicarme de manera más amplia y explorar nuevas oportunidades.

# **Modelado y verificación de patrones de diseño de arquitectura de software para entornos de computación en la nube.**

Este trabajo presenta un entorno de diseño integral que permite formular diseños de arquitecturas de software destinadas a la representación de aplicaciones web. Este entorno abstrae los principales problemas identificados a nivel de diseño, planteando módulos que ayudan al arquitecto en la elaboración de diseños de calidad. Para esto, utiliza como base un metamodelo de componentes arquitectónicos que identifica un conjunto de elementos comúnmente utilizados en dichas arquitecturas. Sobre el modelo se construye una herramienta de instancia gráfica que se complementa con la verificación de patrones de diseño a fin de garantizar su correcta aplicación.

## **Reflexión**

Este trabajo presenta un entorno integral para el diseño de arquitecturas de software orientadas a aplicaciones web. Su objetivo es ayudar al arquitecto a formular diseños de calidad, abordando los problemas comunes del proceso. Basado en un metamodelo de componentes arquitectónicos, identifica y organiza elementos recurrentes que suelen emplearse en estas arquitecturas. El entorno incluye una herramienta de instancia gráfica y un sistema de verificación de patrones de diseño. Esto asegura la correcta aplicación de patrones.



## **Bibliografía**

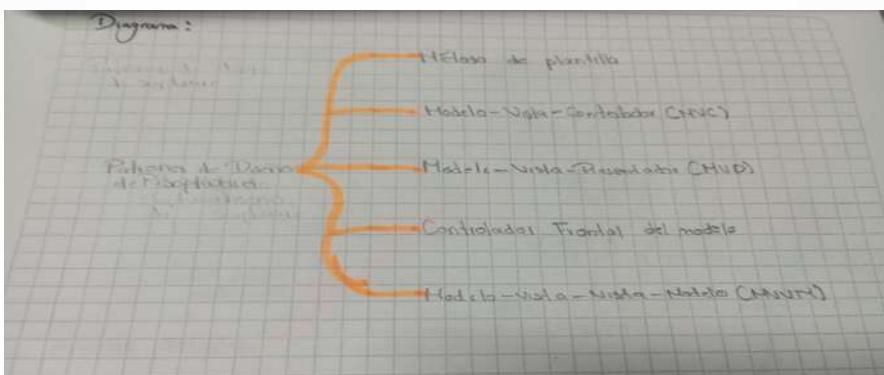
Asociación Ibérica de Sistemas y Tecnologías de Información  
<https://ri.conicet.gov.ar/handle/11336/125130>.

# Análisis comparativo de Patrones de Diseño de Software.

Los patrones de diseño brindan soluciones a problemas que se presentan durante el desarrollo de software, evitan duplicaciones de código y facilitan su reutilización. En el presente artículo se detallan la estructura, componentes, ventajas y desventajas de los patrones de diseño: Template Method, Model-View-Controller, Model-View-Presenter, Model Front Controller y Model-View-View-Model MVVM. La investigación se realizó a través de una revisión bibliográfica en bases de datos científicas y consecuentemente se determinaron las métricas que permitieron comparar los patrones en estudio.

## Reflexión

Los patrones de diseño proporcionan soluciones reutilizables a problemas comunes en desarrollo de software, evitando duplicación de código y mejorando la organización. Este análisis de patrones como Template Method, MVC, MVP, Front Controller y MVVM revela que cada uno es adecuado para contextos específicos, por lo que su elección depende de las necesidades del proyecto. En resumen, los patrones facilitan la creación de software modular, fácil de mantener y de alta calidad.



## Bibliografía

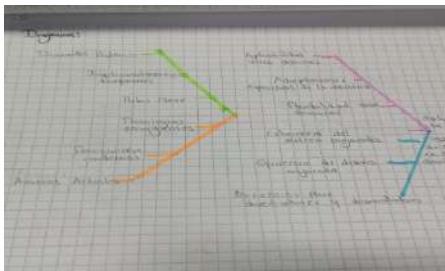
Escuela Superior Politécnica de Chimborazo (ESPOCH), Riobamba, Ecuador <https://dialnet.unirioja.es/servlet/articulo?codigo=9042927>.

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

El propósito principal de este artículo es el de mostrar el estado del arte en un área de la arquitectura de software llamada "Lenguajes de Patrones", desde sus orígenes, los avances actuales y sus aplicaciones en la construcción de arquitecturas de software en diferentes dominios de aplicación. Este último aspecto es relevante ya que como se verá en este artículo, la extensibilidad y aplicabilidad de los lenguajes de patrones a diferentes dominios se convierte en una herramienta importante para diseñadores y desarrolladores de diversos tipos de sistemas de información.

## Bibliografía

Jimenez-Torres,Victor Hugo, Tello-Borja, Wilman, Rios-Patiño Jorge Iván . Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte. Scientia Et Technica [en linea]. 2014, 19(4), 371-376[fecha de Consulta 11 de Noviembre de 2024]. ISSN: 0122-1701. Disponible en: [https://www.redalyc.org/articulo\\_oa?id=84933912003](https://www.redalyc.org/articulo_oa?id=84933912003).



## Reflexión

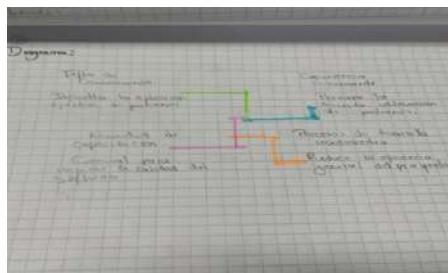
El artículo destaca el valor de los lenguajes de patrones en arquitectura de software como una herramienta poderosa que ha evolucionado desde sus inicios y sigue demostrando su utilidad en diversos dominios. Los lenguajes de patrones no solo ofrecen soluciones estructuradas y probadas para problemas comunes, sino que también fomentan la adaptabilidad y extensibilidad en el diseño de sistemas. Esta flexibilidad es crucial para los diseñadores y desarrolladores, ya que permite aplicar principios de arquitectura efectivos en contextos variados, optimizando la eficiencia y coherencia en proyectos de diferentes áreas. En definitiva, los lenguajes de patrones se convierten en un recurso valioso, ampliando las posibilidades y mejorando la calidad del software.

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

Se presenta el análisis de identificación de Patrones de Diseño definidos por The Gang of Four (GOF) en procesos de desarrollo de software orientados a la Web. Inicialmente se construye un conjunto de criterios para evaluar y seleccionar procesos de desarrollo formales de gran envergadura. Se establece el tamaño de la muestra para aplicar los criterios con estricto rigor metodológico, se realiza la inspección del código fuente para identificar el uso de patrones de diseño y se lleva a cabo un proceso que permite identificar los patrones de diseño que son utilizados por expertos del área de la ingeniería del software. Los resultados permiten concluir que en el sector productivo los patrones de diseño han sido aplicados.

## **Bibliografía**

Grupo de Investigación en Ingeniería del Software-GRIIS, [https://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci\\_arttext&tlang=en](https://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci_arttext&tlang=en)



## **Reflexión**

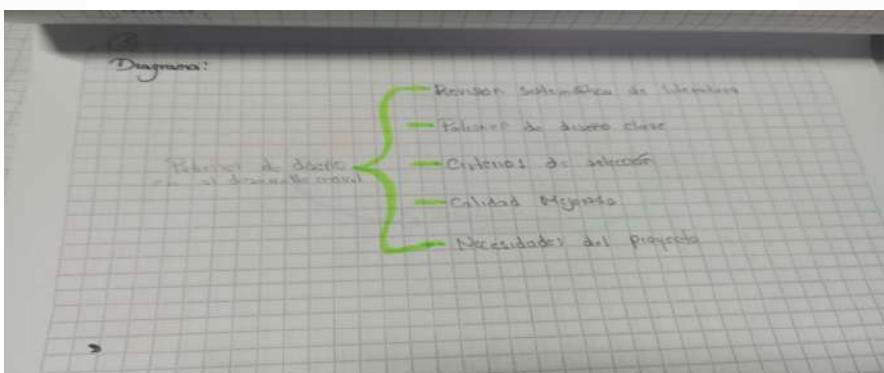
El análisis de la identificación de Patrones de Diseño, basado en los propuestos por The Gang of Four, muestra que, aunque estos patrones se aplican en el desarrollo de software web, su uso es limitado. Esto se debe a la falta de conocimiento y experiencia en su correcta aplicación, lo que impide su aprovechamiento completo. Los resultados destacan la necesidad de una mayor capacitación en la industria para que los desarrolladores puedan implementar estos patrones de manera efectiva, mejorando así la calidad del software, facilitando su mantenimiento y aumentando la eficiencia en el proceso de desarrollo.

# Análisis comparativo de patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad

Este artículo revisa los principales estudios sobre patrones de diseño para el desarrollo de aplicaciones móviles de calidad. A través de una revisión sistemática de literatura en bases de datos como IEEE, Explorer y EBSCO, se seleccionaron 16 artículos relevantes de un total de 3072 encontrados. Utilizando la estrategia PICO, se identificaron 5 patrones clave y criterios de selección para elegir el más adecuado. En conclusión, estos criterios proporcionan una herramienta útil para comparar y seleccionar patrones de diseño, mejorando la calidad del desarrollo móvil según las necesidades del proyecto.

## Reflexión

Este artículo destaca la importancia de los patrones de diseño en el desarrollo de aplicaciones móviles de calidad, basándose en una revisión de literatura de más de 3000 artículos. Se identificaron cinco patrones clave y criterios de selección que permiten a los desarrolladores elegir el patrón más adecuado para cada proyecto. Estos criterios ofrecen una herramienta útil para comparar y seleccionar patrones, lo que optimiza el proceso de desarrollo. En resumen, estos criterios facilitan la selección de soluciones de diseño eficientes, mejorando la calidad de las aplicaciones móviles desarrolladas.



## Bibliografía

Universidad-Peruana-Unión  
<https://repositorio.upeu.edu.pe/items/11034d47-e68f-4b85-b9d2-83e9ceb0ef78>

# Una breve encuesta de conceptos de arquitectura de software y arquitectura orientada a servicios.

Un problema crítico en el diseño y construcción de cualquier sistema de software complejo es su arquitectura. La arquitectura de software como una columna importante del proceso de desarrollo de software tiene varios métodos y hojas de ruta que todos ellos tienen algunos principios e inicios comunes. Se han promovido enfoques basados en la arquitectura como un medio para controlar la complejidad de la construcción y evolución de los sistemas. En este artículo tratamos de describir los conceptos básicos y la estructura principal de la arquitectura de software con una visión conceptual de este problema. Primero, se presentan las definiciones de arquitectura.

## Reflexión

Este artículo destaca la arquitectura de software como un aspecto clave en el desarrollo de sistemas complejos, ayudando a gestionar la complejidad y garantizar la evolución eficiente del software. Se presentan principios comunes y métodos arquitectónicos, con un enfoque en la arquitectura orientada a servicios (SOA) como una opción efectiva para el desarrollo de sistemas web. En resumen, una arquitectura bien estructurada es fundamental para crear sistemas sostenibles y de calidad.

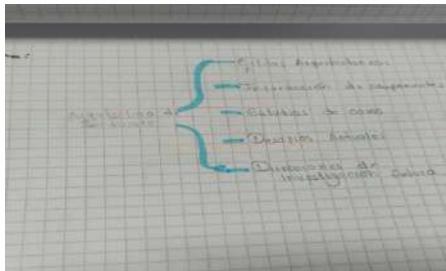


## Bibliografía

IEEE <https://ieeexplore.ieee.org/abstract/document/5235004>

# INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE.

A medida que los sistemas de software crecen, los desafíos de diseño pasan de los algoritmos y las estructuras de datos a la arquitectura del sistema (cómo se organizan e interactúan los distintos componentes). Este artículo presenta la arquitectura del software, explora los estilos arquitectónicos comunes, cómo se pueden combinar en diseños complejos y presenta estudios de casos que muestran cómo estas representaciones mejoran la comprensión de los sistemas grandes. También analiza los problemas actuales y las investigaciones futuras.



## Reflexión

La creciente complejidad de los sistemas de software subraya la importancia de la arquitectura, que permite organizar y conectar múltiples componentes de forma eficiente. A medida que el diseño se enfoca menos en algoritmos y más en cómo interactúan los elementos del sistema, la arquitectura se vuelve esencial para crear software robusto y adaptable. Este enfoque no solo mejora nuestra comprensión de sistemas complejos, sino que también impulsa un diseño más estratégico, capaz de enfrentar los retos futuros.

## Bibliografía

INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE  
DAVID GARLAN (EE.UU.) y MARY SHAW (EE.UU.)

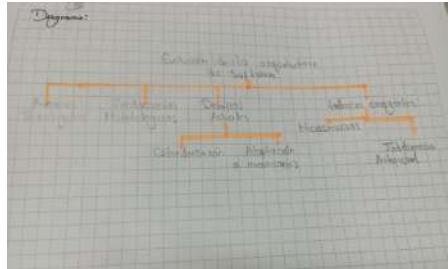
Avances en ingeniería de software e ingeniería del conocimiento. Diciembre de 1993 , 1-39

# Arquitectura de software: un diario de viaje.

En las últimas dos décadas y media, la arquitectura de software ha surgido como un subcampo importante de la ingeniería de software. Durante ese tiempo, se han producido avances considerables en el desarrollo de la base tecnológica y metodológica para tratar el diseño arquitectónico como una disciplina de ingeniería. Sin embargo, todavía queda mucho por hacer para lograrlo. Además, el rostro cambiante de la tecnología plantea una serie de desafíos para la arquitectura de software. Este libro de viajes relata la historia del campo, su estado actual de práctica e investigación, y especula sobre algunas de las tendencias emergentes, desafíos y aspiraciones importantes.

## Bibliografía

Asociación de Maquinaria Computacional, Nueva York Estados Unidos  
[https://dl.acm.org/doi/abs/10.1145/2593882.2593886.](https://dl.acm.org/doi/abs/10.1145/2593882.2593886)



## Reflexión

La arquitectura de software ha recorrido un largo camino, transformándose de un concepto básico a una disciplina compleja y fundamental en el desarrollo tecnológico. A pesar de los avances, aún enfrenta desafíos como la estandarización y la adaptación a las constantes innovaciones. Tendencias como los microservicios y la inteligencia artificial moldean el futuro de esta área, exigiendo a los arquitectos mantenerse actualizados y desarrollar nuevas habilidades.

# Arquitectura de software y diseño de software.

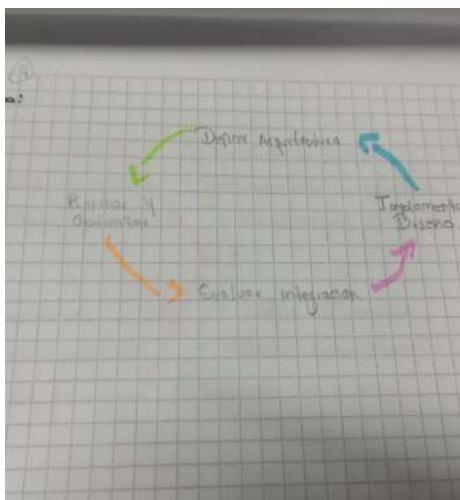
La arquitectura de software se define como el diseño estratégico de una actividad relacionada con los requisitos globales. Su solución se implementa en paradigmas de programación, estilos arquitectónicos, estándares de ingeniería de software basados en componentes, patrones arquitectónicos, seguridad, escala, integración y regularidades regidas por leyes. El diseño funcional, también descrito como diseño táctico, es una actividad relacionada con los requisitos locales que rigen una solución, como algoritmos, patrones de diseño, modismos de programación, refactorización e implementación de bajo nivel. En este artículo, me gustaría presentar algunos conceptos de arquitectura de software y diseño de software, así como la relación entre ellos.

## Bibliografía

Disponible en:  
<https://www.irjet.net/archives/V6/i11/IRJET-V6i11303.pdf>,  
Disponible en SSRN:  
<https://ssrn.com/abstract=3772387>  
<http://dx.doi.org/10.2139/ssrn.372387>

## Reflexión

La arquitectura de software es el plano general de un programa, mientras que el diseño funcional detalla cómo construir cada parte. La arquitectura define la estructura y los componentes principales, mientras que el diseño se centra en la implementación de cada funcionalidad. Ambos son esenciales para crear software eficiente y escalable.

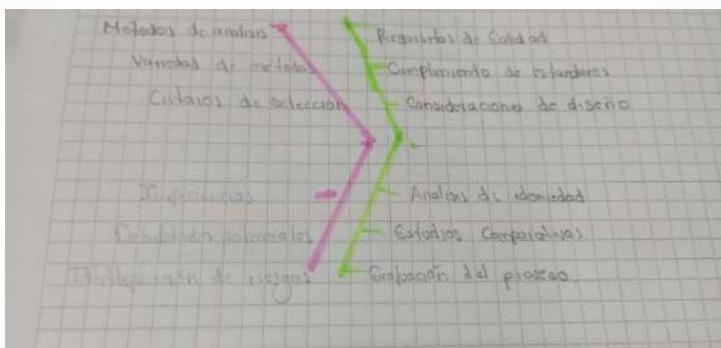


# Una encuesta sobre los métodos de análisis de la arquitectura de software.

El objetivo de la evaluación de la arquitectura de un sistema de software es analizar la arquitectura para identificar riesgos potenciales y verificar que los requisitos de calidad se han abordado en el diseño. Este estudio muestra el estado de la investigación en este momento, en este dominio, presentando y discutiendo ocho de los métodos de análisis de arquitectura más representativos. La selección de los métodos estudiados intenta cubrir tantos puntos de vista particulares como sea posible de reflexiones objetivas que se deriven del objetivo general.

## Reflexión

La evaluación de la arquitectura de software es un proceso crítico que implica analizar la estructura y el diseño de un sistema antes de su implementación. Al igual que un arquitecto revisa los planos de un edificio para detectar posibles errores de construcción, un arquitecto de software examina la estructura de un programa para identificar debilidades o insuficiencias. Esta revisión permite asegurar que el software final sea robusto, confiable y cumpla con los requisitos establecidos, evitando costosas modificaciones posteriores.



## Bibliografía

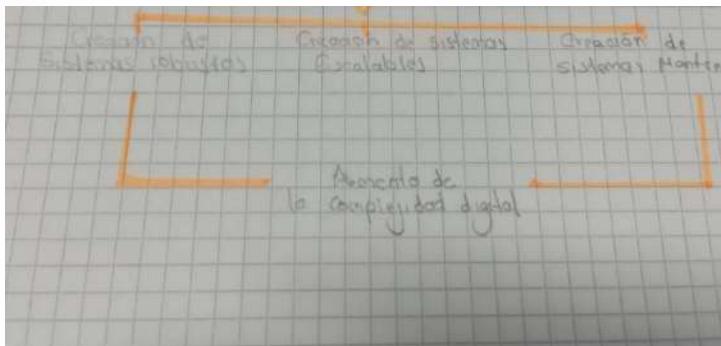
IEEE Transactions on Software Engineering.( volumen: 28 , número: 7 , julio de 2002 ).

# La edad de oro de la arquitectura de software.

La arquitectura de software ha evolucionado de manera significativa desde los años 80. Inicialmente, se usaba para describir sistemas de forma general. Hoy en día, es una disciplina fundamental para diseñar y construir software complejo. Gracias a diversos métodos y herramientas, los arquitectos de software pueden crear sistemas más robustos, escalables y mantenibles. En pocas palabras, la arquitectura de software ha pasado de ser una descripción a convertirse en una guía esencial para el desarrollo de software.

## Reflexión

La arquitectura de software ha evolucionado de una simple descripción a una disciplina fundamental en el desarrollo de software. Al igual que un arquitecto diseña edificios pensando en su funcionalidad y durabilidad, los arquitectos de software construyen las bases de sistemas complejos. Esta evolución refleja la creciente complejidad del mundo digital y la necesidad de soluciones tecnológicas sólidas y escalables. La arquitectura de software es hoy en día un campo en constante crecimiento, donde la innovación y la adaptación a las nuevas tecnologías son claves para el éxito de cualquier proyecto de software.

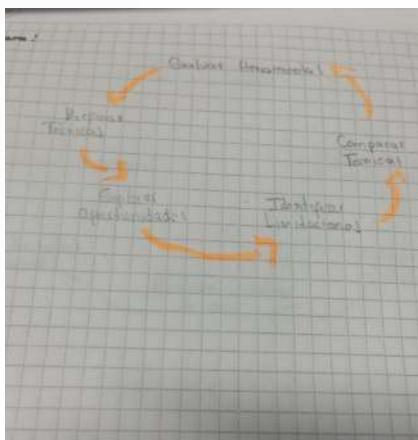


## Bibliografía

[IEEE Software \(Volumen: 23 , Número: 2 , Marzo-Abril 2006.\)](#)

# Un análisis comparativo de las técnicas de recuperación de la arquitectura de software.

El estudio evalúa la capacidad de diferentes herramientas para reconstruir automáticamente la estructura de un software a partir de su código. Se utilizaron arquitecturas de software conocidas como punto de referencia para comparar la efectividad de varias técnicas. Los resultados muestran que, aunque hay avances, la precisión de estas herramientas aún es limitada. Esto abre nuevas oportunidades para mejorar estos métodos y lograr una recuperación de arquitecturas más precisa y confiable.



## Reflexión

La ingeniería inversa de software, específicamente la recuperación de arquitecturas, se revela como un campo en constante evolución. A pesar de los avances en técnicas automatizadas, aún persisten desafíos para reconstruir de manera precisa y completa la estructura interna de un sistema a partir de su código. La falta de un estándar de oro y la complejidad inherente a los sistemas de software actuales limitan la efectividad de estas herramientas. Sin embargo, los resultados de este estudio abren nuevas vías de investigación y desarrollo. Comprender la arquitectura de un sistema es fundamental para su mantenimiento, evolución y modernización. Por tanto, perfeccionar las técnicas de recuperación es un objetivo crucial para la ingeniería de software.

## Bibliografía

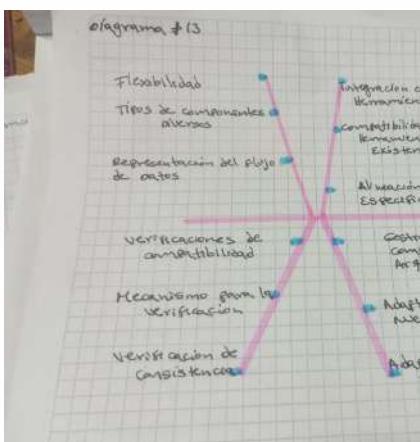
28.<sup>a</sup> Conferencia internacional IEEE/ACM sobre ingeniería de software automatizada (ASE) de 2013.

# Abstracciones para la arquitectura de software y herramientas para respaldarlas.

El modelo propuesto busca formalizar y enriquecer la descripción de arquitecturas de software, capturando las abstracciones de alto nivel que los diseñadores emplean en la práctica. Este modelo permite representar de manera precisa componentes, sus interacciones y patrones de diseño, yendo más allá de los detalles de implementación. Al admitir diversos tipos de componentes y relaciones, incluyendo flujos de datos y programación, el modelo ofrece una gran flexibilidad

## Reflexión

El modelo propuesto representa un avance significativo en la formalización de las arquitecturas de software, acercándonos a una descripción más precisa y comprensible de los sistemas complejos. Al capturar las abstracciones de alto nivel, el modelo facilita la comunicación entre los equipos de desarrollo y permite una mejor comprensión del sistema en su conjunto. Sin embargo, su éxito dependerá de su capacidad para adaptarse a la evolución constante de los sistemas y de su integración con las herramientas y prácticas existentes. Además, es fundamental abordar desafíos como la complejidad creciente de las arquitecturas y la necesidad de garantizar la calidad y la consistencia de los modelos



## Bibliografía

M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young and G. Zelesnik, "Abstractions for software architecture and tools to support them," in IEEE Transactions on Software Engineering, vol. 21, no. 4, pp. 314-335, April 1995, doi: 10.1109/32.385970.

# Patrones de reconfiguración de software para la evolución dinámica de arquitecturas de software.

Un patrón de reconfiguración de software es una solución a un problema en sistemas de software basados en componentes donde la configuración necesita ser actualizada mientras el sistema está en funcionamiento. Define cómo un conjunto de componentes que participan en un patrón de software cooperan para cambiar la configuración. Este documento describe un enfoque para diseñar patrones de reconfiguración de software. También describe cómo los patrones de reconfiguración pueden usarse en arquitecturas de líneas de productos de software reconfigurables.

## Reflexión

Los patrones de reconfiguración de software ofrecen una solución flexible para adaptar sistemas en funcionamiento. Al definir cómo los componentes colaboran para modificar la configuración, permiten crear sistemas más adaptables y resilientes. Estos patrones son especialmente útiles en arquitecturas de líneas de productos, donde se requiere una alta personalización. Sin embargo, su implementación plantea desafíos como la complejidad, la seguridad y el rendimiento. Estudios de caso y prototipos demuestran la viabilidad de estos patrones, abriendo nuevas posibilidades para la evolución dinámica de los sistemas.



## Bibliografía

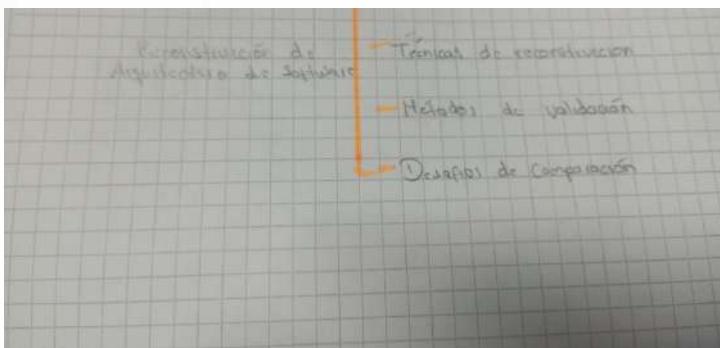
H. Gomaa and M. Hussein, "Software reconfiguration patterns for dynamic evolution of software architectures," Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004), Oslo, Norway, 2004, pp. 79-88, doi: 10.1109/WICSA.2004.1310692.

# Reconstrucción de la arquitectura de software: una taxonomía orientada a procesos.

Para mantener y comprender aplicaciones grandes, es importante conocer su arquitectura. El primer problema es que, a diferencia de las clases y los paquetes, la arquitectura no está representada explícitamente en el código. El segundo problema es que las aplicaciones exitosas evolucionan con el tiempo, por lo que su arquitectura inevitablemente cambia. Por lo tanto, reconstruir la arquitectura y verificar si sigue siendo válida es una ayuda importante. Si bien existe una gran cantidad de enfoques y técnicas que respaldan la reconstrucción de la arquitectura, no existe un estado del arte integral de la reconstrucción de la arquitectura de software y, a menudo, es difícil comparar los enfoques

## Reflexión

Comprender y mantener la arquitectura de una aplicación a gran escala es fundamental para garantizar su longevidad y adaptabilidad. Sin embargo, esta información crucial a menudo se encuentra oculta en el código y evoluciona con el tiempo. Reconstruir y validar la arquitectura es esencial para tomar decisiones informadas sobre su mantenimiento y evolución. A pesar de la existencia de numerosas técnicas para esta tarea, aún no existe un método universalmente aceptado.



## Bibliografía

S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," in IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 573-591, July-Aug. 2009, doi: 10.1109/TSE.2009.19.

# El papel de la vista de decisión en la práctica de la arquitectura de software

El desarrollo de software debe hacer frente a muchos desafíos: la creciente complejidad de los sistemas, las exigencias de una mejor calidad, la carga de las operaciones de mantenimiento, la producción distribuida y la elevada rotación de personal, por nombrar sólo algunos. Cada vez más, las empresas de software que se esfuerzan por reducir los costes de mantenimiento de sus productos exigen diseños flexibles y fáciles de mantener. La arquitectura de software constituye la piedra angular del diseño de software, clave para afrontar estos desafíos.

## Reflexión

La arquitectura de software es esencial para enfrentar los desafíos del desarrollo moderno. La creciente complejidad de los sistemas y las demandas de calidad han hecho que la arquitectura sea una disciplina fundamental. Al proporcionar una estructura sólida y flexible, la arquitectura permite crear sistemas más mantenibles y adaptables. Sin embargo, a pesar de su importancia, la arquitectura sigue siendo un campo en constante evolución, ya que las tecnologías y las necesidades de las empresas cambian rápidamente.



## Bibliografía

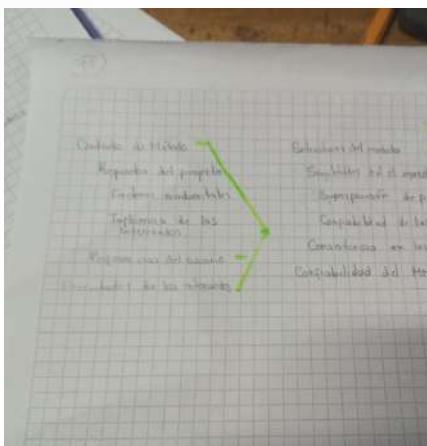
P. Kruchten, R. Capilla and J. C. Dueñas, "The Decision View's Role in Software Architecture Practice," in IEEE Software, vol. 26, no. 2, pp. 36-42, March-April 2009, doi: 10.1109/MS.2009.52.

# Comparación de métodos de evaluación de arquitectura de software basados en escenarios.

La comunidad de ingeniería de software ha propuesto varios métodos para evaluar las arquitecturas de software con respecto a los atributos de calidad deseados, como la capacidad de mantenimiento, el rendimiento, etc. Sin embargo, se han hecho pocos esfuerzos para comparar sistemáticamente dichos métodos con el fin de descubrir similitudes y diferencias entre los enfoques existentes. En este artículo, comparamos cuatro métodos de evaluación de SA basados en escenarios bien conocidos utilizando un marco de evaluación.

## Reflexión

La evaluación de arquitecturas de software es un aspecto crucial para garantizar la calidad y el éxito de los sistemas. Sin embargo, la diversidad de métodos existentes ha dificultado la comparación y selección del enfoque más adecuado para cada proyecto. Este estudio representa un avance significativo al proporcionar un marco común para comparar diferentes métodos y destacar sus similitudes y diferencias. Al identificar actividades comunes y establecer un modelo de proceso genérico, este trabajo contribuye a consolidar el campo de la evaluación de arquitecturas y a brindar a los profesionales una guía más sólida para seleccionar y aplicar las técnicas más apropiadas en cada contexto.



## Bibliografía

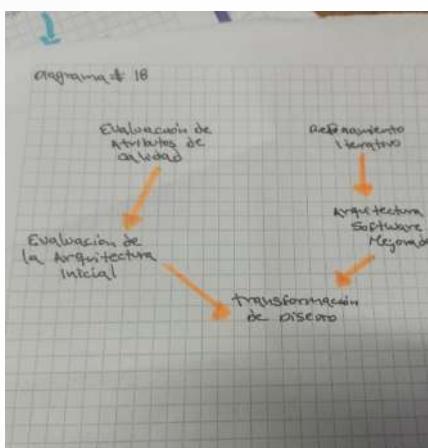
M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," 11th Asia-Pacific Software Engineering Conference, Busan, Korea (South), 2004, pp. 600-607, doi: 10.1109/APSEC.2004.38.

# Reingeniería de arquitectura de software basada en escenarios.

En este artículo se presenta un método para la reingeniería de arquitecturas de software. El método aborda explícitamente los atributos de calidad de la arquitectura de software. La evaluación de los atributos de calidad se realiza principalmente mediante escenarios. Las transformaciones de diseño se realizan para mejorar los atributos de calidad que no satisfacen los requisitos. La evaluación y la transformación del diseño se pueden realizar durante varias iteraciones hasta que se cumplan todos los requisitos. Para ilustrar el método, utilizamos como ejemplo la reingeniería de un sistema de medición prototípico en una arquitectura de software específica del dominio.

## Reflexión

El método propuesto para la reingeniería de arquitecturas de software presenta una contribución significativa al campo, al enfocarse de manera explícita en la mejora de los atributos de calidad a través de la utilización de escenarios. La iteración entre evaluación y transformación permite un refinamiento continuo de la arquitectura, asegurando que se cumplan los requisitos establecidos. La aplicación de este método a un caso de estudio real demuestra su viabilidad y potencial para mejorar la calidad de las arquitecturas de software. Sin embargo, es importante explorar la aplicabilidad de este método a una variedad más amplia de sistemas y arquitecturas, así como evaluar su eficiencia en términos de tiempo y recursos.



## Bibliografía

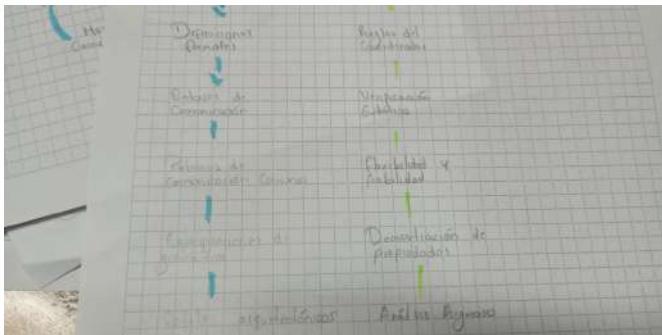
P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203), Victoria, BC, Canada, 1998, pp. 308-317, doi: 10.1109/ICSR.1998.685756.

# Descripción de estilos de arquitectura de software mediante gramáticas gráficas.

Creemos que las arquitecturas de software deberían proporcionar una base adecuada para la demostración de las propiedades de software de gran tamaño. Este objetivo se puede lograr mediante una clara separación entre computación y comunicación y una definición formal de las interacciones entre los componentes individuales. Presentamos un formalismo para la definición de arquitecturas de software en términos de grafos. Los nodos representan los agentes individuales y los bordes definen su interconexión. Los agentes individuales pueden comunicarse solo a lo largo de los enlaces especificados por la arquitectura.

## Reflexión

El enfoque propuesto ofrece una forma rigurosa de definir y analizar arquitecturas de software. Al representar arquitecturas como grafos y definir formalmente sus componentes e interacciones, se facilita la demostración de propiedades importantes del software. La introducción de estilos de arquitectura y un coordinador para gestionar la evolución dinámica añade flexibilidad y rigor al modelo. Este formalismo sienta las bases para una ingeniería de software más precisa y confiable.



## Bibliografía

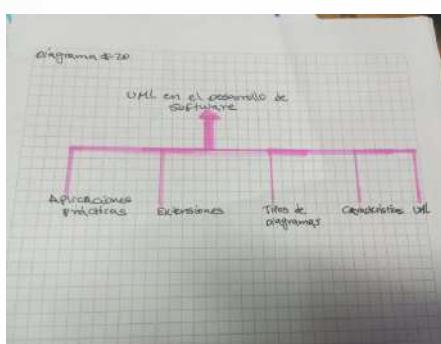
D. Le Metayer, "Describing software architecture styles using graph grammars," in IEEE Transactions on Software Engineering, vol. 24, no. 7, pp. 521-533, July 1998, doi: 10.1109/32.708567.

# En la práctica: descripción de diseño y arquitectura de software UML

El lenguaje de modelado unificado ha atraído a muchas organizaciones y profesionales. UML es ahora el lenguaje de modelado de facto para el desarrollo de software. Varias características explican su popularidad: es una notación estandarizada, rica en expresividad; UML 2.0 proporciona 13 tipos de diagramas que permiten modelar varias vistas y niveles de abstracción diferentes. Además, UML admite extensiones específicas del dominio mediante estereotipos y valores etiquetados. Finalmente, varias herramientas de casos integran el modelado UML con otras tareas, como la generación de código y la ingeniería inversa de modelos a partir del código.

## Reflexión

El UML se ha consolidado como el estándar de facto en el modelado de software, gracias a su versatilidad y capacidad para representar diversos aspectos de un sistema. Su riqueza expresiva, la amplia gama de diagramas y la posibilidad de personalizarlo mediante extensiones lo convierten en una herramienta poderosa para el desarrollo de software. Sin embargo, este estudio nos invita a profundizar más allá de las características intrínsecas del lenguaje y a analizar su aplicación práctica en proyectos reales. Al centrarse en el uso del UML y la calidad de los modelos generados, esta investigación aporta una valiosa perspectiva sobre cómo esta herramienta se utiliza en la industria y cuáles son los desafíos y oportunidades que plantea en el contexto de proyectos concretos.

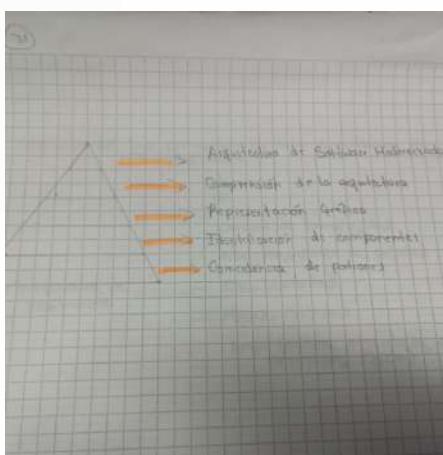


## Bibliografía

C. F. J. Lange, M. R. V. Chaudron and J. Muskens, "In practice: UML software architecture and design description," in IEEE Software, vol. 23, no. 2, pp. 40-46, March-April 2006, doi: 10.1109/MS.2006.50.

# Recuperación de la arquitectura del software basada en la coincidencia de patrones.

Este trabajo presenta una nueva técnica para comprender y visualizar la estructura interna de sistemas de software antiguos. Funciona comparando el código del sistema con plantillas predefinidas de diseños comunes (patrones arquitectónicos). Al utilizar gráficos y algoritmos de comparación, se logra identificar los componentes principales del sistema y cómo se relacionan entre sí, incluso en sistemas muy complejos. Esta técnica es útil para modernizar y mantener sistemas heredados, ya que proporciona una representación visual y comprensible de su arquitectura.



## Reflexión

La propuesta de modelar y recuperar la arquitectura de sistemas heredados mediante patrones y comparación de gráficos representa un avance importante en la ingeniería de software. Esta metodología facilita la comprensión y modernización de sistemas complejos al brindar una representación visual clara. No obstante, su efectividad depende de la calidad de los patrones y la precisión en la extracción del código fuente. Además, la aplicación a sistemas de gran escala presenta desafíos adicionales, como la necesidad de herramientas avanzadas de procesamiento de gráficos y la adaptación de patrones a arquitecturas.

## Bibliografía

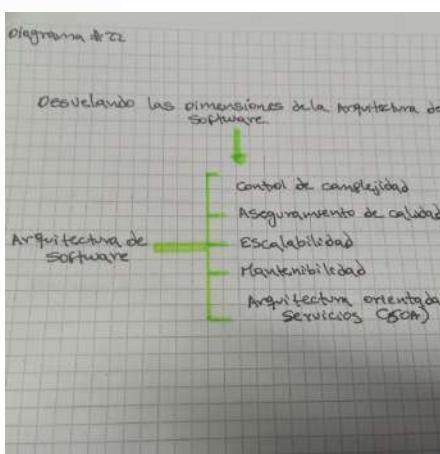
K. Sartipi, "Software architecture recovery based on pattern matching," International Conference on Software Maintenance, 2003., Netherlands, 2003, pp. 293-296, doi: 10.1109/ICSM.2003.1235434.

# Un breve estudio de los conceptos de arquitectura de software y arquitectura orientada a servicios.

Un aspecto crítico en el diseño y la construcción de cualquier sistema de software complejo es su arquitectura. La arquitectura de software, como columna importante del proceso de desarrollo de software, tiene varios métodos y hojas de ruta que tienen algunos principios y un inicio comunes. Los enfoques basados en la arquitectura se han promovido como un medio para controlar la complejidad de la construcción y evolución de los sistemas. En este artículo, tratamos de describir los conceptos básicos y la estructura principal de la arquitectura de software con una visión conceptual de este tema

## Reflexión

La arquitectura de software, ese armazón invisible que sostiene el funcionamiento de los sistemas digitales, es una disciplina que evoluciona constantemente. Al igual que un arquitecto diseña un edificio, el arquitecto de software concibe la estructura de un sistema, determinando sus componentes y cómo interactúan. Esta disciplina es fundamental para controlar la complejidad de los sistemas y garantizar su calidad, escalabilidad y mantenibilidad. La arquitectura orientada a servicios (SOA), por ejemplo, ha revolucionado la forma en que se construyen los sistemas modernos, facilitando la integración de diferentes componentes y la adaptación a las cambiantes necesidades del mercado.



## Bibliografía

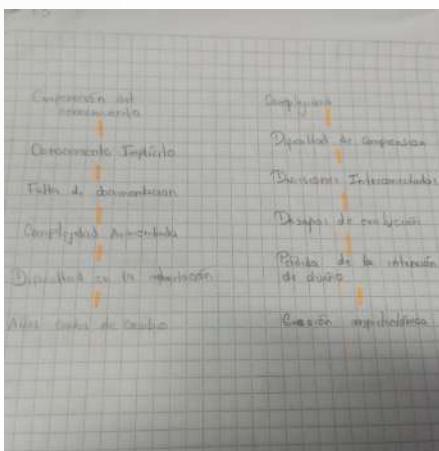
d IEEE International Conference on Computer Science and Information Technology, Beijing, China, 2009, pp. 34-38, doi: 10.1109/ICCSIT.2009.5235004.

# La arquitectura de software como un conjunto de decisiones de diseño arquitectónico.

Las arquitecturas de software tienen altos costos de cambio, son complejas y se erosionan durante la evolución. Creemos que estos problemas se deben en parte a la evaporación del conocimiento. Actualmente, casi todo el conocimiento y la información sobre las decisiones de diseño en las que se basa la arquitectura están implícitamente integrados en la arquitectura, pero carecen de una representación de primera clase. En consecuencia, el conocimiento sobre estas decisiones de diseño desaparece en la arquitectura, lo que conduce a los problemas antes mencionados.

## Reflexión

Las arquitecturas de software enfrentan desafíos significativos, como altos costos de cambio, complejidad y la tendencia a deteriorarse a medida que evolucionan. Estos problemas están, en parte, relacionados con la evaporación del conocimiento. Esto significa que el conocimiento y las razones detrás de las decisiones de diseño que constituyen la arquitectura no están claramente documentados ni representadas. En lugar de ser explícito, este conocimiento queda implícitamente integrado en la propia arquitectura, lo que provoca que se pierda con el tiempo.



## Bibliografía

A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), Pittsburgh, PA, USA, 2005, pp. 109-120, doi: 10.1109/WICSA.2005.61.

# Una arquitectura de software específica de dominio para sistemas inteligentes adaptativos

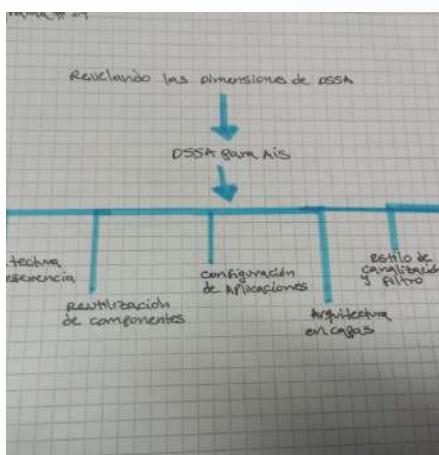
Una buena arquitectura de software facilita el desarrollo de sistemas de aplicaciones, promueve el logro de los requisitos funcionales y respalda la reconfiguración del sistema. Presentamos una arquitectura de software específica de dominio (DSSA) que hemos desarrollado para un gran dominio de aplicaciones de sistemas inteligentes adaptativos (AIS). La DSSA proporciona: (a) una arquitectura de referencia AIS diseñada para satisfacer los requisitos funcionales compartidos por las aplicaciones en este dominio, (b) principios para descomponer la experiencia en componentes altamente reutilizables y (c) un método de configuración de aplicaciones para seleccionar componentes relevantes de una biblioteca y configurar automáticamente instancias de esos componentes en una instancia de la arquitectura.

## Bibliografía

B. Hayes-Roth, K. Pfleger, P. Lalande, P. Morigot and M. Balabanovic, "A domain-specific software architecture for adaptive intelligent systems," in IEEE Transactions on Software Engineering, vol. 21, no. 4, pp. 288-301, April 1995, doi: 10.1109/32.385968.

## Reflexión

Una buena arquitectura de software facilita el desarrollo de sistemas, asegura el cumplimiento de los requisitos funcionales y permite la reconfiguración del sistema. La Arquitectura de Software Específica de Dominio (DSSA) para sistemas inteligentes adaptativos (AIS) ofrece una arquitectura de referencia que cubre necesidades comunes del dominio y promueve la reutilización de componentes. Esta arquitectura también permite configurar aplicaciones automáticamente seleccionando e instanciando los componentes adecuados.

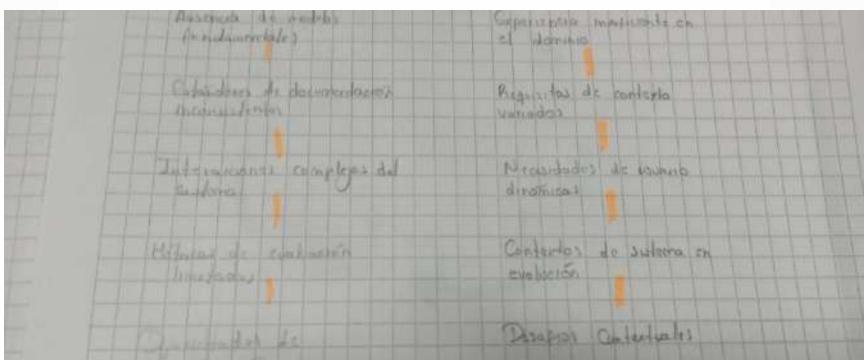


# Obtención de arquitecturas de software basadas en la verdad fundamental

La evolución no documentada de un sistema de software y su arquitectura subyacente requiere recuperar la arquitectura desde los artefactos de implementación. A pesar de las diversas técnicas propuestas, estas sufren de imprecisiones y son difíciles de evaluar debido a la falta de arquitecturas de "verdad fundamental" precisas. Para enfrentar este desafío, abogamos por establecer un conjunto de arquitecturas de verdad fundamentales utilizando un marco de recuperación que incorpora información específica del dominio y del contexto del sistema.

## Reflexión

La evolución no documentada de un sistema de software y su arquitectura subyacente presenta retos importantes, especialmente al intentar recuperar la arquitectura a partir de los artefactos de implementación. Aunque existen varias técnicas propuestas, estas suelen ser imprecisas y difíciles de evaluar debido a la falta de arquitecturas de "verdad fundamental" precisas. Para abordar este desafío, se propone establecer un conjunto de arquitecturas fundamentales utilizando un marco de recuperación que integre información del dominio y el contexto del sistema



## Bibliografía

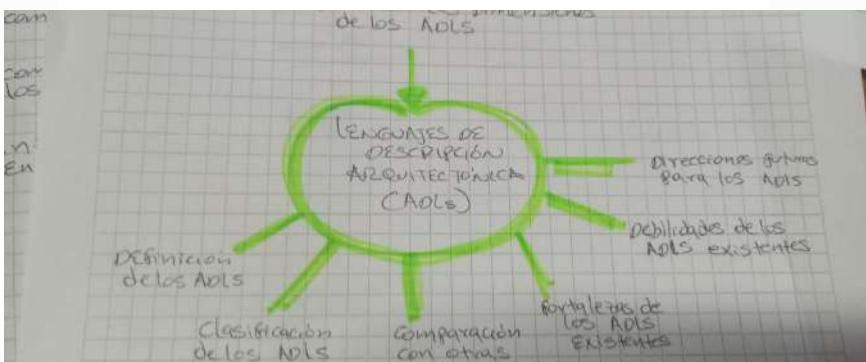
- J. Garcia, I. Krka, C. Mattmann and N. Medvidovic, "Obtaining ground-truth software architectures," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 2013, pp. 901-910, doi: 10.1109/ICSE.2013.6606639.

# Un marco de clasificación y comparación para lenguajes de descripción de arquitectura de software.

Las arquitecturas de software cambian el enfoque de los desarrolladores, pasando de las líneas de código a los elementos arquitectónicos y su interconexión. Los lenguajes de descripción de arquitectura (ADL) se han propuesto como herramientas para apoyar el desarrollo basado en arquitectura, pero aún no existe consenso sobre qué aspectos deben modelarse ni cuál es el ADL más adecuado para cada problema. Además, no se suele diferenciar entre ADL y otras notaciones como especificaciones formales, interconexión de módulos o lenguajes de programación.

## Reflexión

Las arquitecturas de software han cambiado la manera en que los desarrolladores crean sistemas, enfocándose más en los elementos arquitectónicos y sus interconexiones que en las líneas de código. Los lenguajes de descripción de arquitectura (ADL) facilitan este enfoque al proporcionar un modelo estructural claro y coherente. Sin embargo, hay falta de consenso sobre qué aspectos deben modelarse y qué ADL es el más adecuado para cada tipo de problema. Además, se confunden con otras notaciones como las especificaciones formales o lenguajes de programación.



## Bibliografía

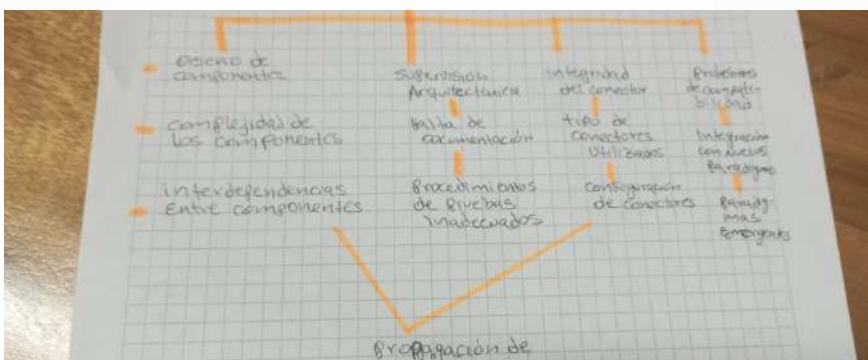
- N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," in IEEE Transactions on Software Engineering, vol. 26, no. 1, pp. 70-93, Jan. 2000, doi: 10.1109/32.825767.

# Propagación de errores en arquitecturas de software

El estudio de las arquitecturas de software está surgiendo como una disciplina importante en la ingeniería de software, debido a su énfasis en la composición a gran escala de productos de software y su apoyo a los paradigmas de ingeniería de software emergentes, como la ingeniería de líneas de productos, la ingeniería de software basada en componentes y la evolución del software. Los atributos arquitectónicos se diferencian de los atributos de software a nivel de código en que se centran en el nivel de componentes y conectores, y en que son significativos para una arquitectura.

## Reflexión

El estudio de las arquitecturas de software está emergente como una disciplina fundamental en la ingeniería de software, dado su enfoque en la composición a gran escala de productos y su apoyo a paradigmas emergentes como la ingeniería de líneas de productos, la ingeniería basada en componentes y la evolución del software. Los atributos arquitectónicos se diferencian de los atributos a nivel de código, ya que se centran en los componentes y conectores, siendo fundamentales para la arquitectura en sí.



## Bibliografía

W. Abdelmoez et al., "Error propagation in software architectures," 10th International Symposium on Software Metrics, 2004. Proceedings., Chicago, IL, USA, 2004, pp. 384-393, doi: 10.1109/METRIC.2004.1357923.

# Impartición de un curso sobre arquitectura de software

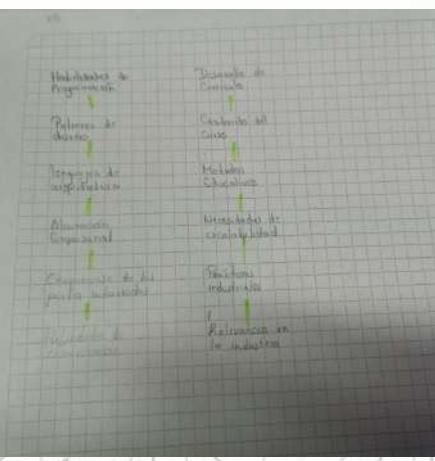
La arquitectura de software es un campo reciente pero central en la ingeniería de software, y muchas organizaciones están invirtiendo esfuerzos significativos en su desarrollo. Como consecuencia, se está incorporando en los planes de estudio de ingeniería de software. Los cursos sobre arquitectura de software suelen centrarse en dos áreas principales: una que aborda los aspectos de programación, incluyendo patrones de diseño y lenguajes de descripción de arquitectura, y otra que se enfoca en la comunicación de la arquitectura a diversas partes interesadas, ampliando su visión. En este artículo, se comparten experiencias de dos cursos de maestría que se concentran en estos aspectos comunicativos.

## Reflexión

La arquitectura de software ha ganado una relevancia significativa en la ingeniería de software, y su incorporación en los planos de estudio universitarios refleja su importancia creciente en el desarrollo de sistemas complejos. Al centrarse en dos áreas clave, la programación y la comunicación con diversas partes interesadas, los cursos de arquitectura de software permiten una comprensión integral del tema. La enseñanza de patrones de diseño y lenguajes de descripción de arquitectura, junto con el énfasis en cómo comunicar estas arquitecturas de manera efectiva, es crucial para formar profesionales capaces de gestionar proyectos de software a gran escala.

## Bibliografía

P. Lago and H. van Vliet, "Teaching a Course on Software Architecture," 18th Conference on Software Engineering Education & Training (CSEET'05), Ottawa, ON, Canada, 2005, pp. 35-42, doi: 10.1109/CSEET.2005.33.



# Métodos centrados en la arquitectura de software y desarrollo ágil

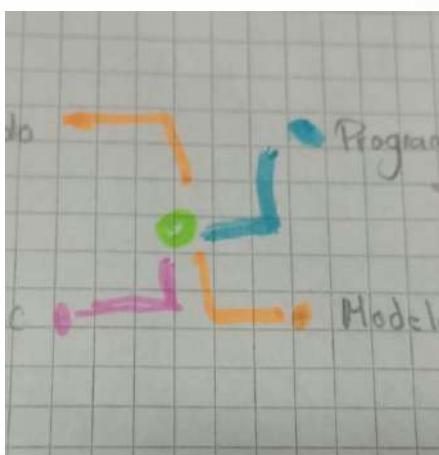
El paradigma de desarrollo de software ágil y los enfoques basados en planes tienen sus puntos fuertes y sus defectos. El primero hace hincapié en el desarrollo rápido y flexible, mientras que el segundo hace hincapié en la infraestructura de proyectos y procesos. Muchos profesionales, en particular de los métodos ágiles, tienden a considerar la arquitectura de software desde el punto de vista del lado basado en planes del espectro. Piensan que los métodos centrados en la arquitectura dan demasiado trabajo y los equiparan a procesos de alta ceremonia que enfatizan la producción de documentos. Pero muchos elementos conforman un enfoque de desarrollo exitoso, incluidos el proceso, el producto, la tecnología, las personas y las herramientas.

## Reflexión

El paradigma ágil y los enfoques basados en planos tienen fortalezas y debilidades. El enfoque ágil enfatiza la rapidez y flexibilidad en el desarrollo, mientras que el enfoque basado en planos se centra en una infraestructura robusta de proyectos y procesos. Sin embargo, muchos en el ámbito ágil tienden a ver la arquitectura de software desde una perspectiva más tradicional, asociándose con procesos rigurosos y documentación extensa. Aunque estos enfoques pueden parecer incompatibles, la arquitectura de software es esencial para la calidad del producto y no está vinculada a un proceso o tecnología específica.

## Bibliografía

R. L. Nord and J. E. Tomayko, "Software architecture-centric methods and agile development," in IEEE Software, vol. 23, no. 2, pp. 47-53, March-April 2006, doi: 10.1109/MS.2006.54.



# Plugins de enrutador: una arquitectura de software para enrutadores de próxima generación.

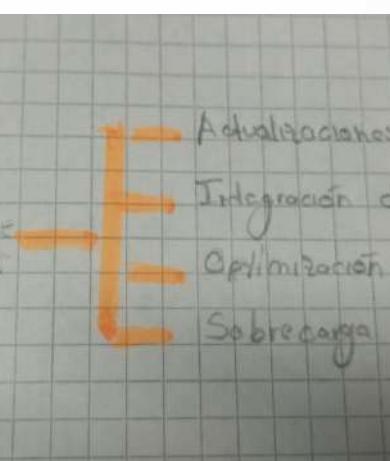
Los enrutadores actuales utilizan sistemas operativos monolíticos que dificultan su actualización. Dado el rápido desarrollo de protocolos, es crucial poder actualizar el software dinámicamente. Para resolver esto, diseñamos una arquitectura de enrutador modular y de alto rendimiento sobre el sistema operativo NetBSD, permitiendo agregar y configurar complementos en tiempo de ejecución. Una característica clave es la vinculación de complementos a flujos individuales, lo que permite que distintas implementaciones coexistan sin problemas. El rendimiento se mejora mediante una arquitectura modular, un algoritmo eficiente de clasificación de paquetes y almacenamiento en caché que aprovecha las características del tráfico de Internet.

## Reflexión

La arquitectura tradicional de los enrutadores, basada en sistemas operativos monolíticos, presenta dificultades para realizar actualizaciones dinámicas, lo cual es crucial en un entorno de rápido desarrollo de protocolos. La solución propuesta, una arquitectura modular sobre NetBSD, permite agregar y configurar complementos en tiempo real, lo que mejora significativamente la flexibilidad y la capacidad de actualización. La capacidad de vincular complementos a flujos individuales asegura que distintas implementaciones puedan coexistir sin conflictos.

## Bibliografía

D. Decasper, Z. Dittia, G. Parulkar and B. Plattner, "Router plugins: a software architecture for next-generation routers," in IEEE/ACM Transactions on Networking, vol. 8, no. 1, pp. 2-15, Feb. 2000, doi: 10.1109/90.836474.



# Desafíos de seguridad en el diseño de arquitectura de hardware y software automotriz.

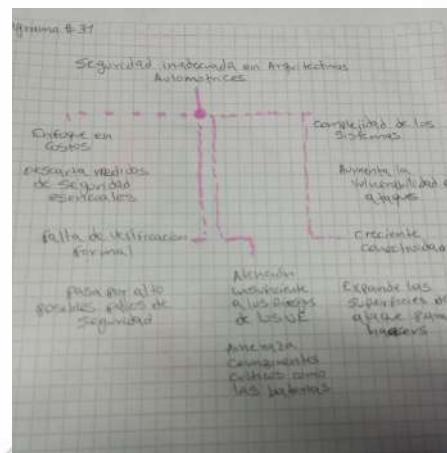
Este artículo aborda los desafíos de seguridad en el diseño de arquitecturas de hardware/software automotriz. Las arquitecturas automotrices de última generación son sistemas complejos y heterogéneos que dependen de funciones distribuidas basadas en electrónica y software. A medida que los automóviles se conectan más con su entorno, aumentan las vulnerabilidades a ataques, como los que involucran sistemas de entrada sin llave, WiFi y Bluetooth. A pesar de esta creciente amenaza, el diseño de estas arquitecturas sigue centrado en cuestiones de costos y eficiencia, dejando de lado la seguridad. El artículo describe las amenazas y vulnerabilidades potenciales, particularmente en los vehículos eléctricos, donde los ataques pueden afectar la seguridad de la batería.

## Reflexión

El artículo destaca cómo las arquitecturas de hardware/software automotrices, cada vez más complejas y conectadas, enfrentan serios desafíos de seguridad. A medida que los vehículos dependen más de la electrónica y del software distribuido, las amenazas a la seguridad, como los ataques a través de sistemas de entrada sin llave, WiFi y Bluetooth, se incrementan. Sin embargo, el diseño de estas arquitecturas sigue priorizando el costo y la eficiencia, lo que deja de lado una adecuada atención a la seguridad. En los vehículos eléctricos, estas vulnerabilidades son aún más críticas, ya que los ataques pueden comprometer elementos vitales como la batería.

## Bibliografía

F. Sagsteller et al., "Security challenges in automotive hardware/software architecture design," 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2013, pp. 458-463, doi: 10.7873/DATE.2013.102

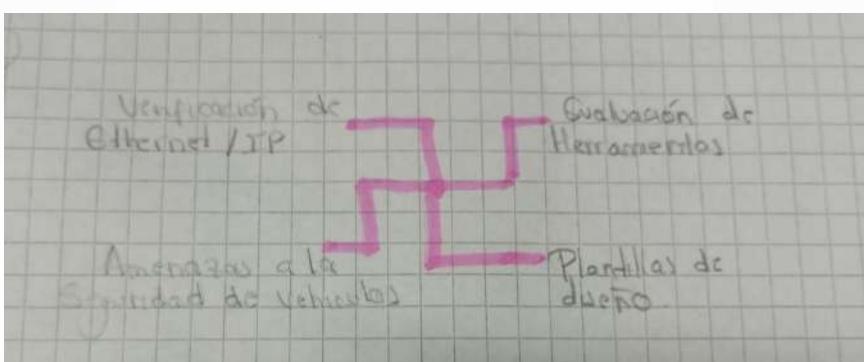


# Visualización de la arquitectura de software: un marco de evaluación y su aplicación

Con el fin de caracterizar y mejorar la práctica de visualización de la arquitectura de software, el artículo deriva y construye un marco cualitativo, con siete áreas clave y 31 características, para la evaluación de herramientas de visualización de la arquitectura de software. El marco se deriva de la aplicación del paradigma de la métrica de preguntas y objetivos a la información obtenida de una encuesta bibliográfica y aborda una serie de cuestiones de las partes interesadas. La evaluación se realiza desde múltiples perspectivas de las partes interesadas y en varios contextos arquitectónicos.

## Reflexión

La creciente conectividad de los vehículos genera nuevas vulnerabilidades. A medida que los automóviles se conectan más con su entorno, surgen amenazas como los ataques a través de WiFi, Bluetooth y sistemas de entrada sin llave. A pesar de los riesgos, el enfoque sigue centrado en la reducción de costos y eficiencia, dejando la seguridad en un segundo plano. Especialmente en los vehículos eléctricos, donde un ataque puede afectar la batería, la falta de protección representa un grave peligro.



## Bibliografía

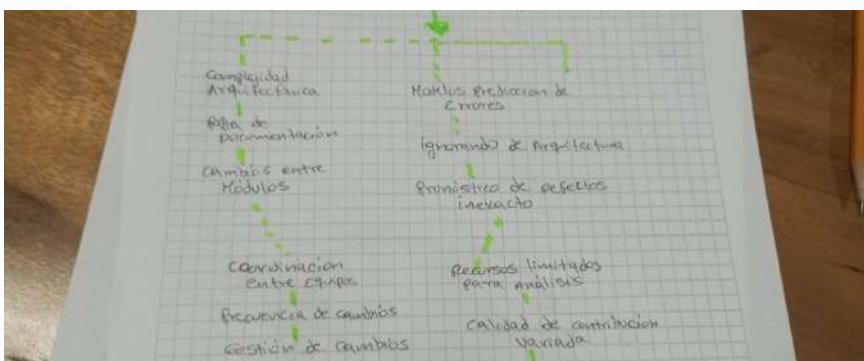
- K. Gallagher, A. Hatch and M. Munro, "Software Architecture Visualization: An Evaluation Framework and Its Application," in IEEE Transactions on Software Engineering, vol. 34, no. 2, pp. 260-270, March-April 2008, doi: 10.1109/TSE.2007.70757.

# Un estudio sobre el papel de la arquitectura del software en la evolución y la calidad del software.

La sabiduría convencional sostiene que la arquitectura de un sistema de software influye en su evolución, pero no se había investigado cómo afecta la evolución del software a partir de su historial de cambios. Esto se debe a que muchos sistemas de código abierto no documentan sus arquitecturas. Para superar este desafío, se utilizaron técnicas de recuperación de arquitectura y modelos recuperados para analizar si los cambios conjuntos que abarcan Múltiples módulos arquitectónicos están más relacionados con errores que los cambios dentro de un solo módulo.

## Reflexión

El estudio realizado pone en evidencia cómo la arquitectura de un sistema de software influye en su evolución, un aspecto que no había sido explorado a fondo en investigaciones previas. Al utilizar técnicas de recuperación de arquitectura, el análisis muestra que los cambios que afectan múltiples módulos arquitectónicos están más relacionados con la aparición de errores que aquellos que ocurren dentro de un solo módulo. Esta observación es relevante porque muchos sistemas de código abierto no documentan sus arquitecturas, lo que dificulta el análisis de su impacto en la evolución del software.



## Bibliografía

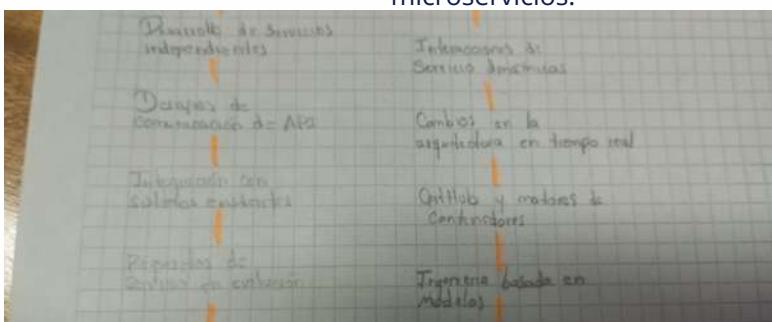
E. Kouroshfar, M. Mirakhori, H. Bagheri, L. Xiao, S. Malek and Y. Cai, "A Study on the Role of Software Architecture in the Evolution and Quality of Software,"

# Hacia la recuperación de la arquitectura de software de los sistemas basados en microservicios.

Hoy en día, empresas como Netflix, Amazon y The Guardian adoptan arquitecturas de microservicios, que consisten en pequeños servicios independientes que se comunican a través de API REST. Aunque ofrecen beneficios como mayor escalabilidad y facilidad de mantenimiento, también complican la comprensión de la arquitectura general del sistema, especialmente cuando los microservicios evolucionan en el tiempo de ejecución. Para resolver esto, se presenta MicroART, un enfoque de recuperación de arquitectura que utiliza técnicas de ingeniería basada en modelos y un lenguaje de dominio específico para representar la arquitectura de sistemas de microservicios.

## Reflexión

Las arquitecturas de microservicios, adoptadas por grandes empresas como Netflix, Amazon y The Guardian, proporcionan ventajas en términos de escalabilidad y mantenimiento, ya que permiten desarrollar servicios independientes que se comunican entre sí a través de API REST. Sin embargo, uno de los desafíos más grandes que surgen es la dificultad de comprender la arquitectura general del sistema, sobre todo cuando los microservicios evolucionan durante su ejecución. En este contexto, MicroART surge como una solución innovadora, al aplicar técnicas de ingeniería basada en modelos y un lenguaje específico para representar la arquitectura de sistemas de microservicios.



## Bibliografía

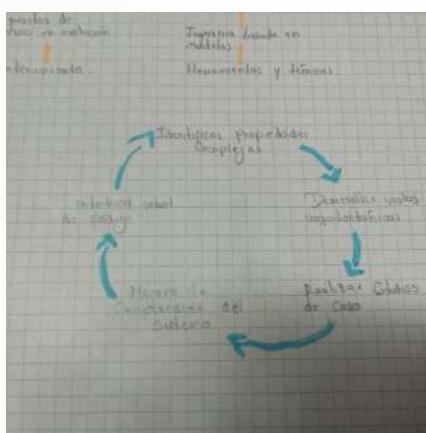
G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino and A. Di Salle, "Towards Recovering the Software Architecture of Microservice-Based Systems,"

# La vista de la arquitectura del software en tiempo de compilación

La investigación en arquitectura de software ha subrayado la necesidad de abordar los sistemas desde varios puntos de vista, como la vista lógica, de código, de proceso, de implementación y escenarios. Sin embargo, algunos sistemas muestran propiedades complejas en tiempo de compilación que no son cubiertas por los modelos tradicionales. Este artículo presenta el concepto de vistas arquitectónicas en tiempo de compilación, explicando cómo representarlas e integrarlas con los modelos arquitectónicos existentes.

## Reflexión

La investigación en arquitectura de software resalta la importancia de abordar los sistemas desde diferentes perspectivas, como la lógica, el código, los procesos y la implementación, para comprender su funcionamiento integral. Sin embargo, algunos sistemas presentan características complejas durante el tiempo de compilación que no son capturadas por los modelos tradicionales. El artículo propuesto presenta las vistas arquitectónicas en tiempo de compilación, una nueva manera de representar y entender estas propiedades. A través de estudios de caso, se demuestra cómo este enfoque mejora la comprensión del sistema, su desarrollo y la resolución de problemas.

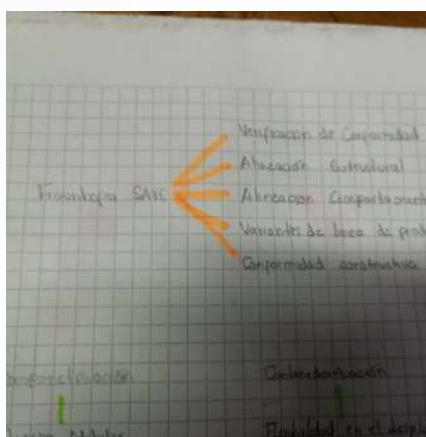


## Bibliografía

Q. Tu and M. W. Godfrey, "The build-time software architecture view," Proceedings IEEE International Conference on Software Maintenance. ICSM 2001, Florence, Italy, 2001, pp. 398-407, doi: 10.1109/ICSM.2001.972753.

# **SAVE: Visualización y evaluación de la arquitectura de software**

Fraunhofer SAVE (visualización y evaluación de la arquitectura de software) es una herramienta para analizar y optimizar la arquitectura de los sistemas de software implementados. SAVE es un desarrollo conjunto entre Fraunhofer IESE (Institute for Experimental Software Engineering IESE en Kaiserslautern, Alemania) y Fraunhofer Center Maryland (Center for Experimental Software Engineering en College Park, Maryland, EE. UU.). En este trabajo describimos las capacidades de la herramienta para asegurar la conformidad de los sistemas existentes con su arquitectura.



## **Reflexión**

Fraunhofer SAVE es una herramienta diseñada para analizar y optimizar la arquitectura de sistemas de software implementados, desarrollada en colaboración entre Fraunhofer IESE en Alemania y Fraunhofer Center Maryland en Estados Unidos. Su principal objetivo es garantizar que los sistemas existentes cumplan con su arquitectura, utilizando características avanzadas de verificación de conformidad. SAVE asegura la alineación con las vistas arquitectónicas estructurales y de comportamiento, facilita la conformidad entre variantes en líneas de productos y permite incorporar la verificación de conformidad constructiva durante el desarrollo y evolución del software.

## **Bibliografía**

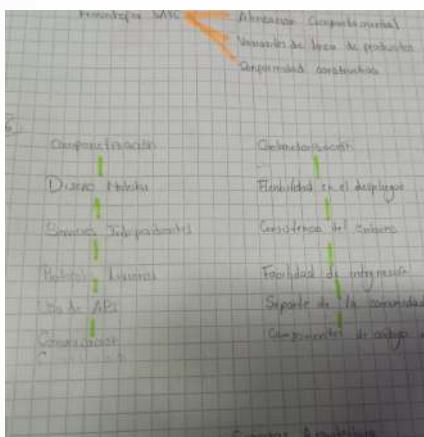
S. Duszynski, J. Knodel and M. Lindvall, "SAVE: Software Architecture Visualization and Evaluation," 2009 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany, 2009, pp. 323-324, doi: 10.1109/CSMR.2009.52. keywords

# Arquitectura y enfoques de software basados en microservicios

El estilo arquitectónico de microservicios ha surgido como una solución eficaz para aplicaciones, especialmente en la industria aeroespacial. Este enfoque divide una aplicación en pequeños servicios independientes, cada uno ejecutándose en su propio proceso y comunicándose mediante mecanismos livianos, como API. Estos servicios están diseñados en torno a capacidades comerciales y de misión específica, pueden implementarse automáticamente y facilitarán el mantenimiento al fragmentar la aplicación en componentes modulares. A diferencia de las aplicaciones monolíticas, cada componente en los microservicios se desarrolla y despliega de forma independiente, permitiendo mayor flexibilidad.

## Reflexión

La arquitectura de microservicios ofrece una solución efectiva para aplicaciones complejas, como las de la industria aeroespacial, al dividirlas en pequeños servicios independientes que facilitan su desarrollo y mantenimiento. Al diseñarse en torno a capacidades específicas y comunicarse mediante mecanismos ligeros como APIs, los microservicios permiten una implementación automatizada y flexible. A diferencia de las arquitecturas monolíticas, cada componente puede desarrollarse y desplegarse de manera autónoma, lo que mejora la adaptabilidad frente a cambios.



## Bibliografía

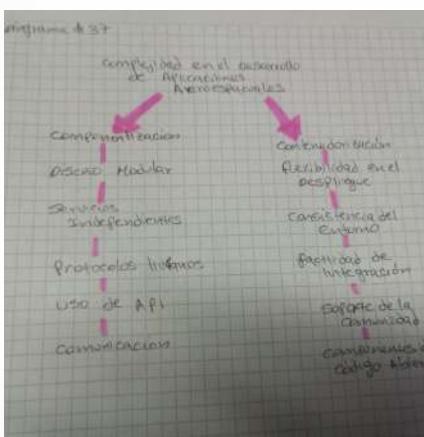
- K. Bakshi, "Microservices-based software architecture and approaches," 2017 IEEE Aerospace Conference, Big Sky, MT, USA, 2017, pp. 1-8, doi: 10.1109/AERO.2017.7943959.

# Transformaciones de la arquitectura de software

Para comprender y mejorar el software, normalmente examinamos y manipulamos su arquitectura. Por ejemplo, podemos querer examinar la arquitectura en diferentes niveles de abstracción o ampliar una parte del sistema. Podemos descubrir que la arquitectura extraída se ha desviado de nuestro modelo mental del software y, por lo tanto, podemos querer repararla. El artículo identifica el punto en común entre estas acciones de transformación arquitectónica, es decir, al manipular la arquitectura para comprender, analizar y modificar la estructura del software, de hecho estamos realizando transformaciones gráficas.

## Reflexión

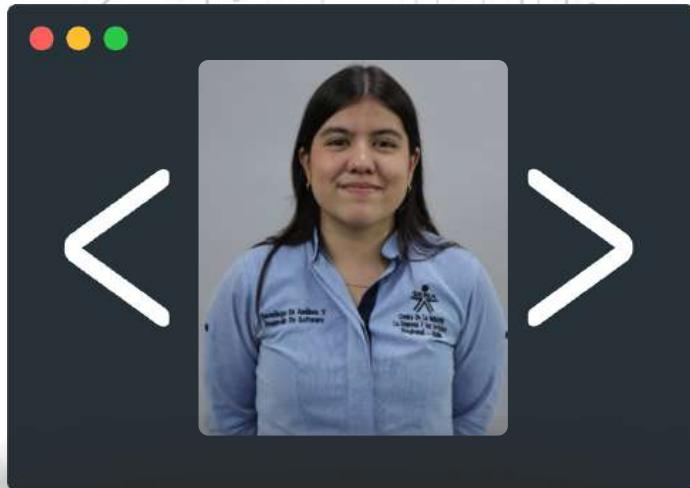
Entender y mejorar el software requiere examinar y manipular su arquitectura en distintos niveles de abstracción, ampliando o reparando partes según sea necesario. Este proceso revela desviaciones entre la arquitectura real y nuestro modelo mental, lo que demanda ajustes. El artículo conecta estas acciones con transformaciones gráficas, clasificándolas dentro de un marco unificado que permite comprenderlas mejor. Esta unificación facilita modelarlas, especificarlas y automatizarlas, optimizando la comprensión y modificación de la estructura del software. Así, trabajar con transformaciones arquitectónicas no solo mejora el análisis, sino que también fortalece la capacidad de adaptación y evolución del sistema.



## Bibliografía

Fahmy and Holt, "Software architecture transformations," Proceedings 2000 International Conference on Software Maintenance, San Jose, CA, USA, 2000, pp. 88-96, doi: 10.1109/ICSM.2000.883020. keywords: {Formal languages},

# Aura María Fierro Fierro



Soy Aura María Fierro, estudiante del SENA en la tecnología de Análisis y Desarrollo de Software.

Apasionada por el mundo del desarrollo y el arte, he encontrado en la programación una herramienta para crear soluciones innovadoras.

Además,uento con un título de Técnico en Multimedia, lo que me ha permitido combinar mi creatividad con habilidades técnicas.

Mi objetivo es seguir creciendo en este campo, aprendiendo nuevas tecnologías y mejorando mis capacidades para contribuir a la transformación digital.

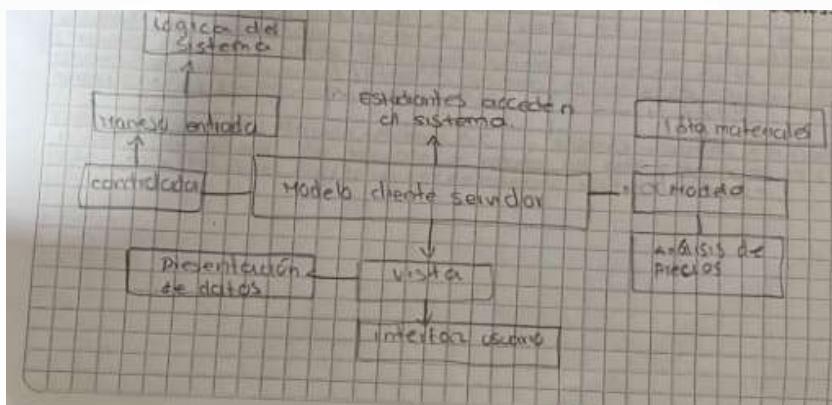
Siempre busco aprender y aplicar mis conocimientos en proyectos que generen un impacto positivo.

# Arquitectura de software para costos y programación de obra.

Este artículo presenta el desarrollo de un software educativo para la gestión de costos, presupuestos y programación de obras en el programa de Ingeniería Civil de la Universidad Francisco de Paula Santander. El objetivo es modernizar los métodos tradicionales de enseñanza mediante una herramienta accesible, interactiva y práctica que facilite el aprendizaje de conceptos complejos. Basado en el modelo Vista-Controlador, el software permite crear y gestionar presupuestos, calcular costos, planificar recursos materiales y humanos, y optimizar su administración.

## Reflexión

Este artículo refleja cómo las herramientas tecnológicas en la educación transforman la formación de futuros profesionales, permitiendo que los estudiantes adquieran conocimientos teóricos y habilidades prácticas. La iniciativa muestra cómo la tecnología supera las barreras económicas, mejora la eficiencia del aprendizaje y asegura una educación accesible y equitativa. Además, el uso del patrón MVC en el desarrollo del software destaca cómo las buenas prácticas de programación facilitan la comprensión del software, preparando a los estudiantes para los desafíos profesionales.

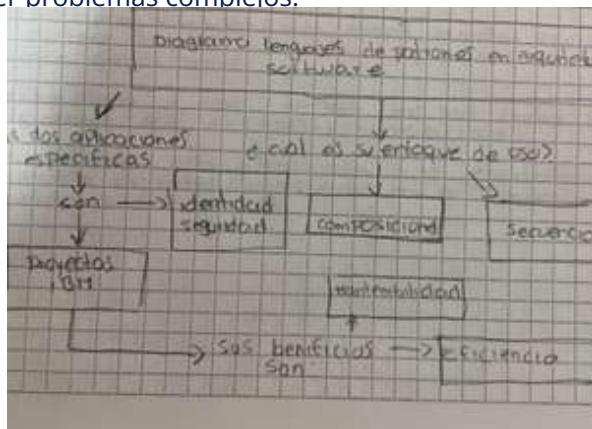


## Bibliografía

Este artículo puede compartirse bajo la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional y se referencia usando el siguiente formato: Cárdenas-Gutiérrez, JA., Barrientos-Monsalve, E. J. y 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. <http://dx.doi.org/10.33304/re>

# Patrones de Arquitectura de Software: Estado del Arte.

El artículo analiza la evolución de los lenguajes de patrones en la arquitectura de software, basado en los conceptos de Christopher Alexander en arquitectura física. Estos patrones han mejorado el diseño modular y reutilizable, incrementando la adaptabilidad y calidad de los sistemas. Son ampliamente utilizados en sectores como IBM para e-business y en áreas como la administración de identidades y seguridad. Los autores revisan enfoques como la composición de patrones y cómo estos métodos estructuran soluciones eficientes, destacando su impacto en la ingeniería de software y su capacidad para resolver problemas complejos.



## Reflexión

La reflexión del artículo "Lenguajes de Patrones de Arquitectura de Software: Una Aproximación al Estado del Arte" resalta la importancia de los lenguajes de patrones en la ingeniería de software moderna. Estos patrones facilitan la resolución de problemas recurrentes en el diseño arquitectónico y promueven un enfoque estructurado en el equipo de desarrollo, creando una base común. Al estandarizar buenas prácticas, contribuyendo a sistemas más robustos y adaptables. Su versatilidad en diversos contextos los convierte en una herramienta clave para soluciones escalables y sostenibles en un entorno tecnológico en constante cambio.

## Bibliografía

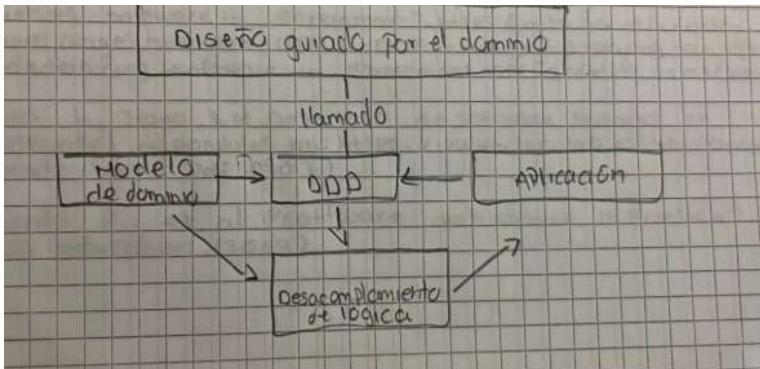
Jiménez-Torres, VH, Tello-Borja, W. & Ríos-Patiño, JI (2014). Lenguajes de Patrones de Arquitectura de Software: Aproximación al Estado del Arte. Ciencia y técnica, 19(4)

# Implementación de Arquitectura de Software por el Dominio.

Este artículo analiza la implementación de una arquitectura de software guiada por el dominio (DDD) para crear sistemas eficientes y sostenibles. DDD se enfoca en los aspectos clave del negocio, separando la complejidad técnica en capas independientes. Además, integra arquitecturas limpias y hexagonales para desacoplar el sistema, mejorando su evolución y mantenimiento. En un caso de estudio, DDD se aplica a una plataforma de empleo que conecta demandantes y oferentes, reflejando reglas del negocio. La arquitectura hexagonal facilita pruebas y cambios, adaptándose sin afectar el núcleo del sistema.

## Reflexión

La aplicación de DDD en la ingeniería de software crea sistemas centrados en el negocio y redefine cómo se percibe y desarrolla el software. Este enfoque fomenta una comprensión común entre desarrolladores y expertos, mejorando la comunicación y alineando expectativas con resultados. La arquitectura hexagonal desacopla la lógica del negocio de la infraestructura, garantizando sistemas más adaptables y menos dependientes de tecnologías específicas. Esto impulsa una evolución hacia sistemas sostenibles y robustos, donde la estrategia del negocio es el foco y la tecnología un soporte flexible.



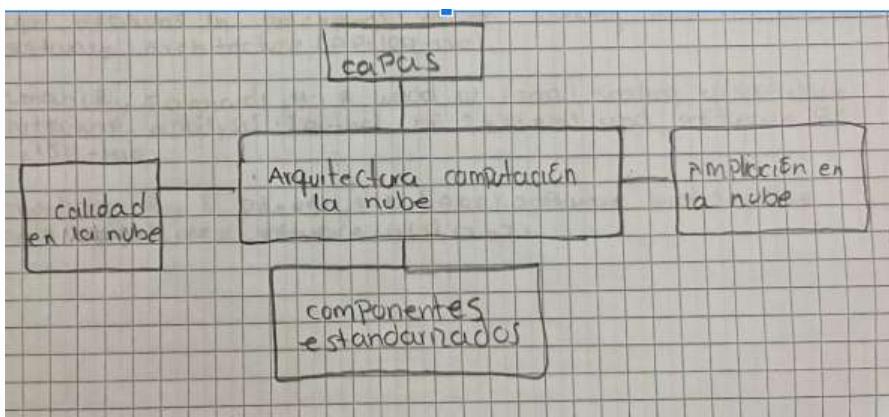
## Bibliografía

Cambarieri M, Difabio F, García Martínez N. Implementación de Una Arquitectura de Software Guiada Por El Dominio. Accessed November 19, 2024.

<https://n9.cl/dch02>

## **Modelado y Verificación de Patrones de Arquitectura en la Nube.**

Este artículo presenta un enfoque para diseñar y verificar patrones de arquitectura de software en entornos de computación en la nube. Se basa en un metamodelo de componentes arquitectónicos que organiza los elementos clave del diseño para crear aplicaciones web robustas. Incluye una herramienta de modelado gráfico que permite aplicar y evaluar patrones de diseño mediante verificación. La metodología destaca por crear arquitecturas eficientes y adaptables, organizadas en capas que separan funciones de aplicación e infraestructura. Su objetivo es optimizar la productividad, reducir la complejidad y mejorar la calidad y consistencia del software en la nube.



### **Reflexión**

Este artículo reflexiona sobre la importancia de una metodología estructurada para diseñar arquitecturas en la nube, donde la separación de funciones y la estandarización de componentes facilitan la escalabilidad. El uso de herramientas de modelado y verificación permite un control riguroso sobre los patrones, mejorando la calidad del software y alineando las decisiones arquitectónicas con los objetivos de negocio. La propuesta sugiere que los arquitectos de software en la nube deben combinar conocimientos técnicos y diseño estratégico para adaptarse a las demandas del mercado y usuarios.

### **Bibliografía**

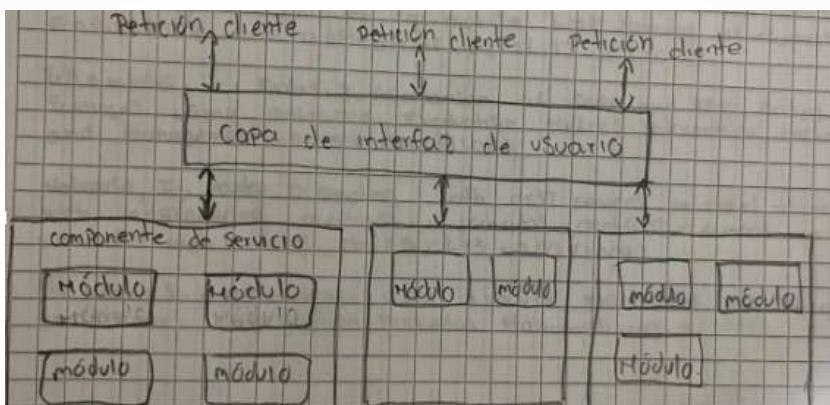
Blas MJ, Leone H, Gonnet S. Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação. 2019;(35):1-17. doi:<https://doi.org/10.17013/risti.35.1-17>

# Arquitectura de Microservicios para Aplicaciones Web.

Este artículo analiza el uso de una arquitectura de microservicios como alternativa al modelo monolítico en el desarrollo de aplicaciones web en la Asamblea Nacional del Ecuador. Los microservicios permiten separar funciones en servicios independientes, mejorando la escalabilidad, mantenibilidad y tiempos de despliegue. Cada microservicio tiene su propia lógica de negocio y base de datos, facilitando la administración descentralizada y la tolerancia a fallos. Se mencionan topologías comunes como API REST y mensajería centralizada, concluyendo que permite un desarrollo ágil.

## Reflexión

Este artículo reflexiona sobre la importancia de adoptar enfoques flexibles como la arquitectura de microservicios en el desarrollo de software. A medida que las organizaciones crecen, una estructura modular y desacoplada permite responder eficientemente a los cambios y reducir el impacto de fallos. Los microservicios mejoran la escalabilidad y aportan resiliencia, permitiendo que los componentes del sistema se desplieguen y administren de forma independiente. Esta flexibilidad es esencial para adaptarse rápidamente a las necesidades del negocio y usuario final.

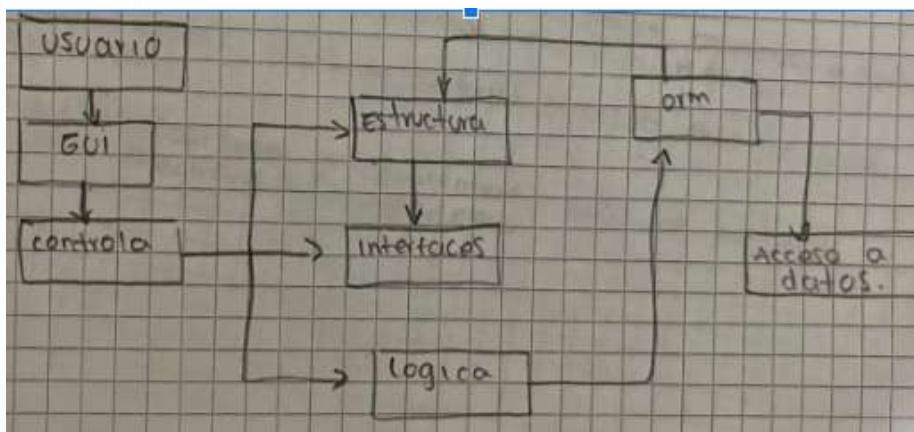


## Bibliografía

Asamblea Nacional del Ecuador, Coordinación General de Tecnologías de la Información y Comunicación, Pichincha, Ecuador (daniel.lopez@asambleanacional.gob.ec). Universidad Técnica del Norte, Instituto de Posgrados, Imbabura, Ecuador (eamaya@utn.edu.ec).

# Arquitectura Académica para Comprender el Desarrollo en Capas

El artículo presenta una arquitectura de software en capas para el ámbito educativo, diseñada para enseñar a los estudiantes sobre el desarrollo estructurado. La arquitectura se divide en cuatro capas: la capa de interfaz gráfica (GUI), que maneja la interacción con el usuario; la capa de control, que gestiona la comunicación entre la interfaz y la lógica; la capa de lógica de negocio, que contiene la funcionalidad central; y la capa de acceso a datos, que gestiona la conexión con las bases de datos. Esta estructura modular facilita el mantenimiento y la escalabilidad.



## Reflexión

Este artículo reflexiona sobre la importancia de enseñar desarrollo de software con una arquitectura en capas, que facilita el aprendizaje de conceptos clave de organización y modularidad. Permite a los estudiantes entender cómo dividir un sistema en componentes independientes, lo que es crucial para el mantenimiento y la escalabilidad en el mundo real. Además, les proporciona una base sólida para manejar la complejidad de sistemas grandes. Esta metodología prepara a los futuros profesionales para adaptarse a cambios y optimizar el rendimiento en entornos diversos.

## Bibliografía

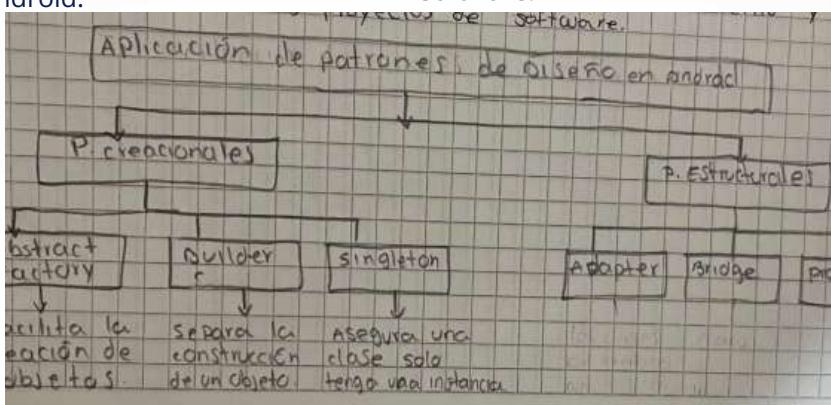
Suggested Citation: Cardacci, Darío G. (2015) : Arquitectura de software académica para la comprensión del desarrollo de software en capas, Serie Documentos de Trabajo, No. 574, Universidad del Centro de Estudios Macroeconómicos de Argentina (UCEMA), Buenos Aires

# Clasificación de Patrones de Diseño para Programación Android.

Este artículo examina la aplicación de patrones de diseño en la programación para Android, destacando su importancia para evitar prácticas deficientes como el "código espagueti." Los patrones de diseño permiten resolver problemas comunes y hacer el código más legible y eficiente. Se clasifican en dos categorías: creacionales (Factory Method, Abstract Factory, Builder, Singleton) y estructurales (Adapter, Bridge, Proxy). Su implementación mejora la robustez, adaptabilidad y mantenimiento de las aplicaciones móviles, beneficiando la calidad y el rendimiento de las aplicaciones Android.

## Reflexión

Este artículo reflexiona sobre la importancia de los patrones de diseño para mejorar la calidad del software, especialmente en el desarrollo de aplicaciones móviles. Su correcta implementación permite a los desarrolladores crear código organizado y modular, reduciendo redundancia y facilitando el mantenimiento en aplicaciones Android. Además, fomenta una cultura de buenas prácticas entre programadores, esencial para su formación continua. En un entorno competitivo, estos conocimientos marcan la diferencia en la eficiencia, rendimiento y sostenibilidad de proyectos de software.

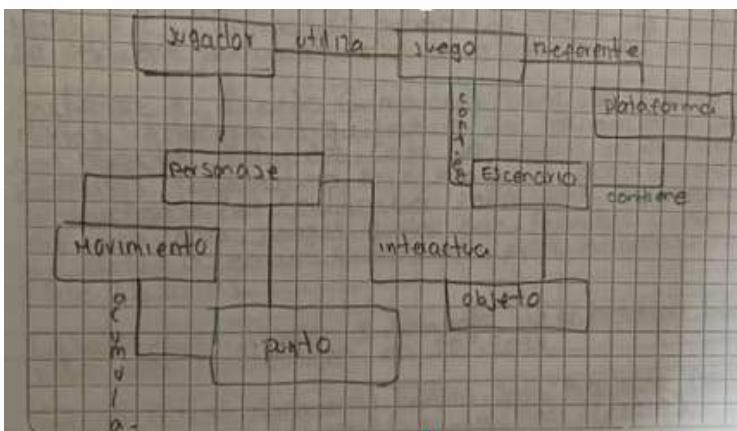


## Bibliografía

- Ibarra A, Luis J, Ulises R, Lepe C, Garzón AP. CLASIFICACIÓN DE LOS PATRONES DE DISEÑO IDÓNEOS EN PROGRAMACIÓN ANDROID. Revista Digital de Tecnologías Informáticas y Sistemas. 2024;3(1). Accessed November 19, 2024. <https://redis.org/index.php/Redis/article/view/33>

# Arquitectura de Software para Videojuegos en Unity 3D.

Este artículo describe una arquitectura de software para el desarrollo de videojuegos en Unity 3D, con el objetivo de mejorar la estructura y calidad del código. Utiliza una arquitectura en capas combinada con componentes, organizada en tres capas: la capa principal del juego, la capa de personajes y la capa de interacción con el mundo. Incluye patrones de diseño como Singleton y Event Handler para mejorar la eficiencia. La validación mediante el método ATAM evalúa atributos de calidad, logrando un modelo adecuado para videojuegos multiplataforma, facilitando el trabajo colaborativo y la continuidad.



## Reflexión

Este artículo reflexiona sobre la importancia de una arquitectura de software bien diseñada para el desarrollo eficiente de videojuegos en Unity 3D. La estructura modular y en capas permite integrar nuevas funcionalidades sin afectar la estabilidad. El uso de patrones de diseño mejora la organización y el control de eventos, clave en proyectos de entretenimiento interactivo. Esta arquitectura facilita el mantenimiento, la escalabilidad y ofrece una base reutilizable, mejorando el tiempo de desarrollo y permitiendo a los desarrolladores centrarse en la creatividad e innovación.

## Bibliografía

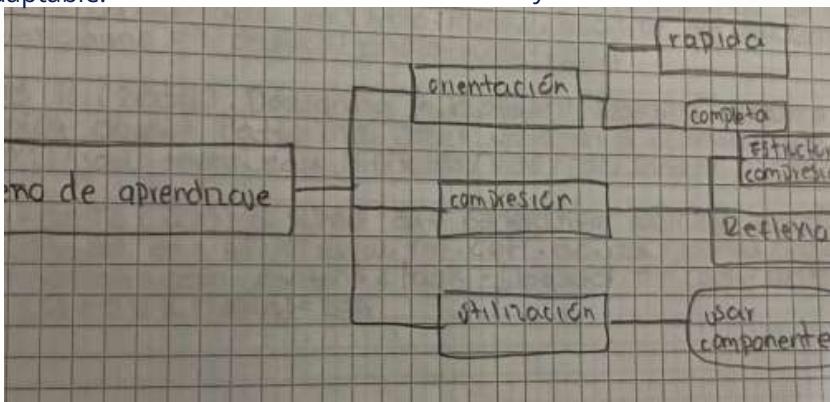
Andy Hernández Paez 1\*, Javier Alejandro Domínguez Falcón 2 , Alejandro Andrés Pi Cruz3 1,2,3 Centro de Entornos Interactivos 3D, Vertex, Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, Km 2½, Torrens, La Lisa, La Habana, Cuba

# Patrones de Diseño para Repositorios de Aprendizaje.

Este artículo explora cómo los patrones de diseño en la arquitectura de objetos de aprendizaje optimizan la creación y reutilización de recursos educativos digitales. Se propone un modelo estructurado centrado en el Centro de Excelencia (CETL) que organiza y facilita el acceso a recursos en sistemas de gestión de aprendizaje (LMS). Los repositorios centralizados permiten almacenar y categorizar objetos con metadatos, mejorando su localización y adaptabilidad. Se destaca la importancia de los Objetos de Aprendizaje Generativos (GLO) para una educación digital inclusiva y adaptable.

## Reflexión

Este artículo reflexiona sobre cómo la arquitectura de objetos de aprendizaje va más allá de ser una herramienta de almacenamiento, convirtiéndose en una estrategia para personalizar y enriquecer el proceso educativo. Un modelo estructurado y adaptativo promueve una educación que responde a las necesidades de los usuarios, fomentando mayor interacción y compromiso. El uso de metadatos y categorías facilita el acceso a contenidos pertinentes, ahorrando tiempo y recursos. Este enfoque subraya la flexibilidad y actualización continua de recursos educativos digitales en un entorno inclusivo y accesible.

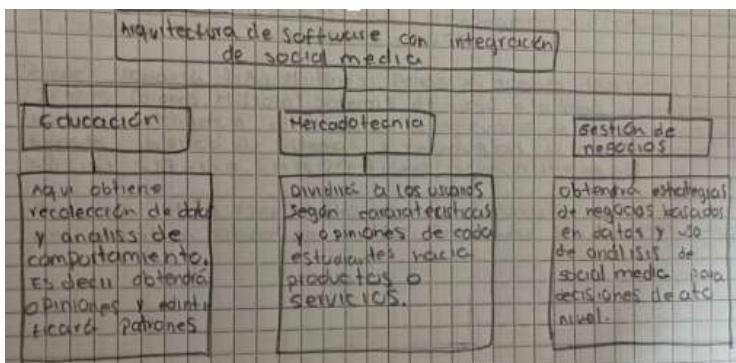


## Bibliografía

Vista de Patrones de Diseño aplicados a la organización de repositorios de objetos de aprendizaje. Revistas.um.es. Published 2024. Accessed November 19, 2024. <https://revistas.um.es/red/article/view/89331/86361>

# Aplicación de Arquitecturas de Software y Social Media.

El artículo examina cómo las arquitecturas de software pueden aprovechar las herramientas de social media para recolectar y analizar datos generados en redes sociales. Se realiza un mapeo de literatura para identificar aplicaciones en áreas como educación, mercadotecnia, gestión de negocios e investigación. El análisis resalta el valor de incorporar modelos de social media para entender comportamientos y opiniones de usuarios, ayudando a las organizaciones a ajustar sus estrategias. También se identifican retos de escalabilidad, interoperabilidad y seguridad para integrar eficazmente social media en sistemas.



## Reflexión

Este artículo reflexiona sobre cómo la integración de social media en arquitecturas de software es crucial para adaptarse a un entorno digital en constante cambio. El análisis de grandes volúmenes de datos de redes sociales permite a las organizaciones conectar mejor con su audiencia y ajustar sus estrategias. Sin embargo, esta integración plantea desafíos técnicos y éticos, como garantizar la privacidad y seguridad de los datos. Se subraya la importancia de desarrollar frameworks robustos y éticos que aprovechen el potencial de social media respetando los derechos de los usuarios.

## Bibliografía

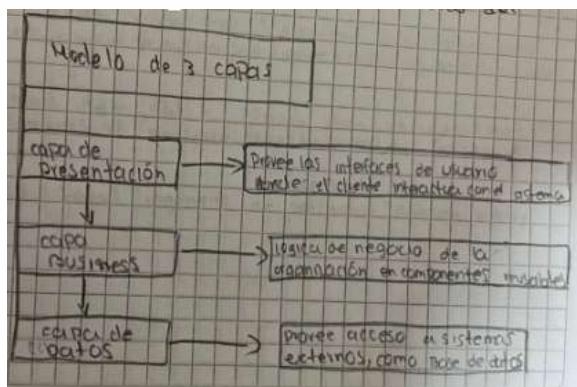
Velazquez-Solis PE, Flores-Rios BL, Ibarra-Esquer JE, et al. Identificación de áreas de aplicación de arquitecturas de software basadas en modelos, técnicas y herramientas de social media. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação. 2021;(42):12-29. doi:<https://doi.org/10.17013/risti.42.12-29>

# Desarrollo de sistemas con patrones orientados a objetos.

El artículo explora la implementación de patrones de diseño orientados a objetos para desarrollar una arquitectura de software en la Facultad de Ingeniería Industrial, centrada en la intranet "Intranet Industrial." Se destacan patrones como el de tres capas, informador y sensor, que optimizan la estructura y reducen costos de mantenimiento. Usando la metodología COCOMO, se comparan costos y efectividad con y sin patrones, destacando los beneficios de modularidad y reutilización. La investigación concluye que los patrones son esenciales para proyectos de software complejos.

## Reflexión

Este artículo reflexiona sobre la importancia de adoptar patrones de diseño en la arquitectura de software para lograr sistemas eficientes, escalables y fáciles de mantener. La "Intranet Industrial" ilustra cómo una arquitectura bien organizada reduce costos, facilita el mantenimiento y mejora el código. Patrones como el de tres capas y sensor permiten construir sistemas modulares y adaptables. La reutilización y modularidad optimizan el desarrollo y proporcionan una guía para enfrentar la complejidad de sistemas en crecimiento, creando aplicaciones robustas y flexibles.

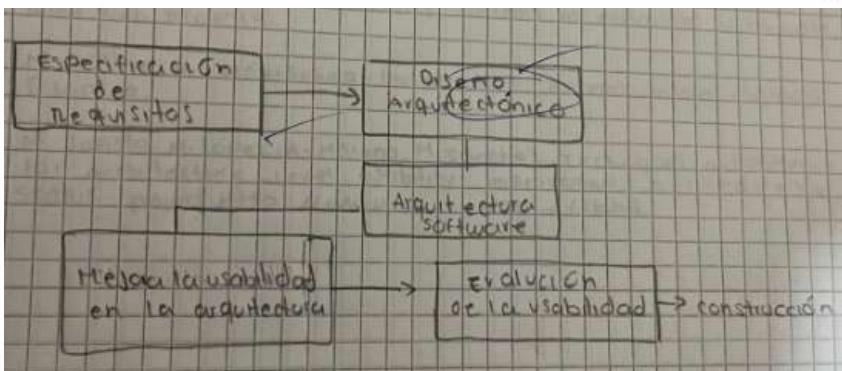


## Bibliografía

CHRISTOPHER ALEXANDER (1977) "A Pattern Language" (Oxford University Press 1977).

# Patrones de Usabilidad: Mejora desde la Arquitectura.

El artículo explora cómo mejorar la usabilidad en el desarrollo de software a través de la arquitectura, destacando el proyecto STATUS, que incorpora atributos de usabilidad desde el inicio del diseño. En lugar de ajustar la usabilidad al final, STATUS propone patrones arquitectónicos como feedback y guía de usuario. Este enfoque facilita una experiencia de usuario fluida, considerando la satisfacción, el aprendizaje y la eficiencia. Los patrones, como el feedbacker y undoer, mejoran la interacción en aplicaciones complejas y son aplicados en arquitecturas modulares reutilizables.



## Reflexión

Este artículo resalta la importancia de considerar la usabilidad desde la fase inicial del diseño de software. Incorporar atributos de usabilidad en la arquitectura asegura sistemas más eficientes y satisfactorios, evitando ajustes tardíos. STATUS propone un enfoque innovador mediante patrones específicos que crean aplicaciones funcionales y amigables. Este enfoque demuestra que la estructura y usabilidad están interconectadas, optimizando la experiencia del usuario y reduciendo costos a largo plazo. Los desarrolladores deben priorizar las necesidades del usuario para crear software más adaptable y eficaz.

## Bibliografía

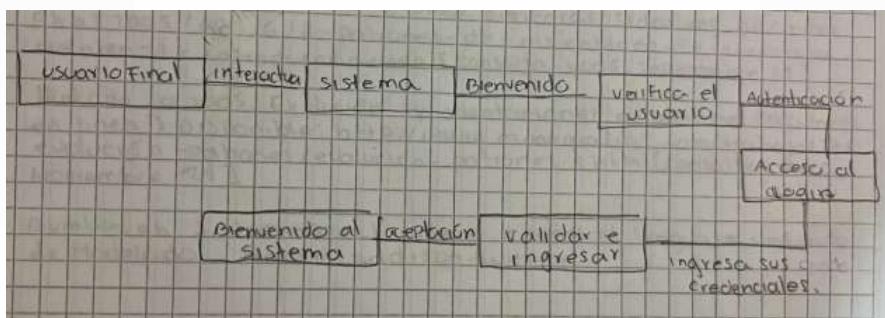
Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico Ana M. Moreno Facultad de Informática Universidad Politécnica de Madrid, España M. Isabel Sánchez Facultad de Informática Universidad Carlos III de Madrid, España

# Herramienta de Reúso JavaScript para Patrones de Interacción.

El artículo presenta ReusMe, una herramienta para mejorar la reutilización de código JavaScript en interfaces de usuario basadas en patrones de interacción. ReusMe permite personalizar código reusable, facilitando la creación de interfaces usables y optimizadas. Utilizando metodologías como UML y el Proceso Unificado de Desarrollo de Software (PUD), integra patrones de interacción para ajustar componentes web a las necesidades del usuario. La herramienta mejora la eficiencia, reduciendo tiempos y costos, y garantiza una experiencia de usuario coherente y de calidad.

## Reflexión

Este artículo destaca que la reutilización de código mediante patrones de interacción es esencial para la eficiencia y calidad en el desarrollo de interfaces de usuario web. ReusMe proporciona a los diseñadores una herramienta que ahorra tiempo y mejora la experiencia del usuario mediante patrones efectivos. El concepto de "no reinventar la rueda" permite a los desarrolladores enfocarse en otros aspectos de la aplicación. Este enfoque promueve la estandarización del diseño, mejorando la coherencia visual, la usabilidad y la satisfacción del usuario final.

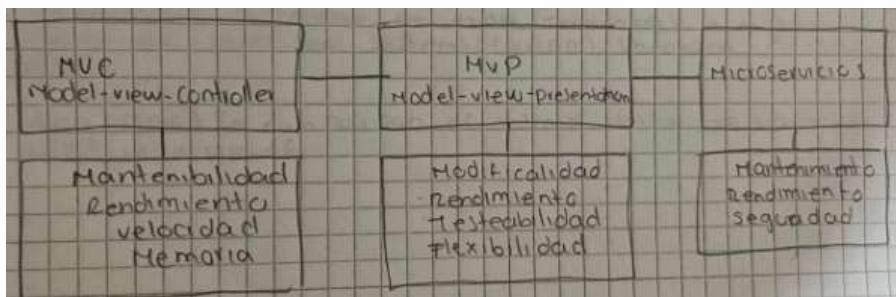


## Bibliografía

BENIGNI, GLADYS; ANTONELLI, OCTAVIO; VÁSQUEZ, YOVANNY HERRAMIENTA PARA REUSO DE CÓDIGO JAVASCRIPT ORIENTADO A PATRONES DE INTERACCIÓN SABER. Revista Multidisciplinaria del Consejo de Investigación de la Universidad de Oriente, vol. 21, núm. 1, enero-abril, 2009, pp. 60-69 Universidad de Oriente Cumaná, Venezuela

# Buenas prácticas en la construcción de software

Este artículo compara las arquitecturas de software monolíticas y de microservicios, destacando sus ventajas y limitaciones. Las arquitecturas monolíticas agrupan toda la funcionalidad en un único sistema, lo que facilita el despliegue en proyectos pequeños, pero limita la escalabilidad en aplicaciones grandes. Los microservicios, al dividir la aplicación en componentes independientes, ofrecen mayor flexibilidad y escalabilidad, aunque requieren más coordinación y una infraestructura más compleja. Empresas como Amazon y eBay optan por microservicios, mientras que Walmart utiliza monolitos para reducir costos.



## Reflexión

Este artículo reflexiona sobre la importancia de elegir una arquitectura adecuada según las necesidades del proyecto y las capacidades del equipo. La arquitectura monolítica es eficaz para aplicaciones pequeñas o con requisitos estables, ofreciendo simplicidad y menor carga operativa. Los microservicios, aunque más complejos, son ideales para proyectos grandes y en constante cambio, proporcionando mayor adaptabilidad. Los arquitectos de software deben evaluar las características del proyecto, anticipando necesidades futuras y sopesando los beneficios y desafíos operativos de cada enfoque.

## Bibliografía

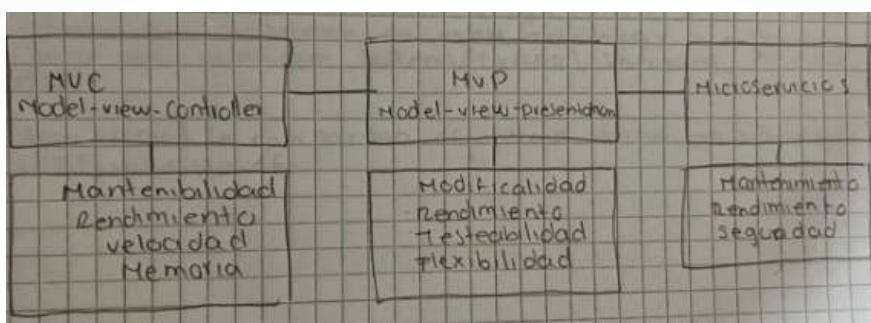
Valentín Torassa Colombero, Estelles JP, Gallegos L, Lopez P. Monolitos vs. Microservicios en Arquitectura de Software: Perspectivas para un Desarrollo Eficiente. Memorias de las JAIIO, 2024;10(5):42-54. Accessed November 19, 2024.

# Marco para Seleccionar un Patrón Arquitectónico en Software.

Este artículo presenta un marco para seleccionar el patrón arquitectónico adecuado en el desarrollo de software, evaluando factores como calidad, modularidad y escalabilidad. Se analizan patrones como MVC, MVP, microservicios y arquitecturas en la nube, adaptados a diferentes tipos de aplicaciones. Este marco ayuda a los arquitectos y desarrolladores a elegir el patrón según el tipo de aplicación y sus requisitos de seguridad, rendimiento y mantenibilidad, optimizando tiempos y costos, y reduciendo riesgos de reprocesos y problemas de escalabilidad.

## Reflexión

Este artículo resalta la importancia de un marco claro para seleccionar patrones arquitectónicos en proyectos de software. Tener una guía adecuada minimiza riesgos de fallos y promueve la escalabilidad y sostenibilidad del producto final. Una arquitectura bien planificada no solo cubre los requisitos actuales, sino que anticipa necesidades de mantenimiento y actualización. Este enfoque asegura decisiones arquitectónicas claras y prácticas de desarrollo alineadas con las mejores prácticas de la industria, optimizando cada capa del software.

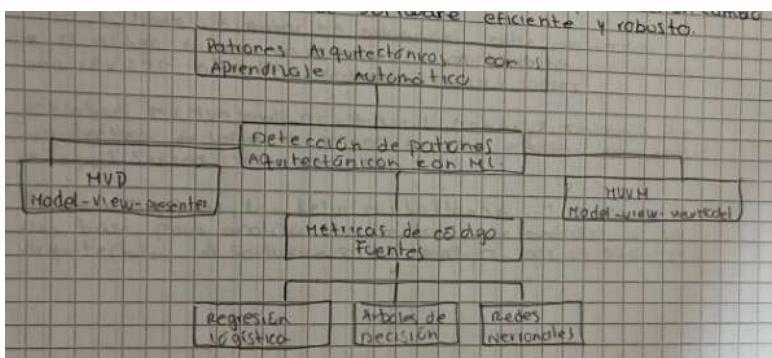


## Bibliografía

Camilo J. Alberto, Kelly G. Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software. Tdeaeduco. Published online 2022. doi:<https://dspace.tdea.edu.co/handle/tdea/2670>

# Predicción de Patrones Arquitectónicos con Aprendizaje Automático.

Este artículo presenta un enfoque de aprendizaje automático (ML) para predecir patrones arquitectónicos, específicamente MVP y MVVM, en proyectos de software. Utilizando métricas de código de más de 5,000 proyectos Java en GitHub, se entrena un modelo ML con un 83% de precisión. Los modelos se validan mediante k-fold ( $k = 5$ ), demostrando eficacia en la detección de patrones con un conjunto mínimo de métricas. Además, se resalta cómo el ML optimiza la consistencia entre el código y la arquitectura, abordando limitaciones de datos y retos de validación.



## Reflexión

Este artículo destaca que el aprendizaje automático (ML) optimiza la toma de decisiones en arquitectura de software, permitiendo identificar patrones con precisión y mejorando la coherencia entre la arquitectura planificada y la implementación. La investigación abre nuevas perspectivas para la automatización en ingeniería de software, especialmente en proyectos donde la elección de patrones impacta la escalabilidad y mantenibilidad. Integrar ML en el diseño arquitectónico podría ser clave para el desarrollo de software eficiente, ofreciendo una base sólida para futuras investigaciones en este campo.

## Bibliografía

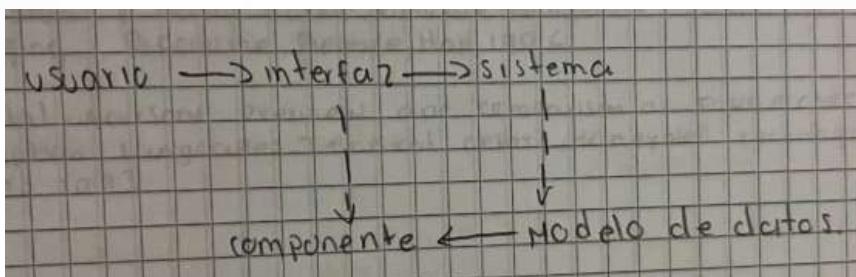
- Sirojiddin Komolov, Gcinizwe Dlamini, Swati Megha, Mazzara M. Towards Predicting Architectural Design Patterns: A Machine Learning Approach. Computers. 2022;11(10):151-151. doi:<https://doi.org/10.3390/computers11100151>

# Representación de la arquitectura de software usando UML

El artículo explora la evolución de la arquitectura de software y la necesidad de representaciones claras para una comunicación efectiva en los proyectos. Los lenguajes de descripción de arquitecturas (ADL) actuales son complejos y específicos, lo que limita su adopción. En cambio, UML se presenta como una alternativa flexible, permitiendo modelar aspectos del sistema mediante diagramas como casos de uso, clases y componentes. UML facilita el análisis y la toma de decisiones, con mecanismos de extensión que permiten personalizar la representación para adaptarse a sistemas específicos.

## Reflexión

Este artículo destaca la importancia de herramientas de modelado precisas y comprensibles para todos los involucrados en un proyecto de software. El uso de UML permite representar la arquitectura de manera accesible y reutilizable a lo largo del ciclo de vida del sistema, mejorando la comunicación entre los desarrolladores y otros interesados. Además, la capacidad de extender UML mediante estereotipos y restricciones facilita la adaptación a diferentes estilos arquitectónicos sin perder la estandarización, lo cual es clave para el éxito en la ingeniería de software.

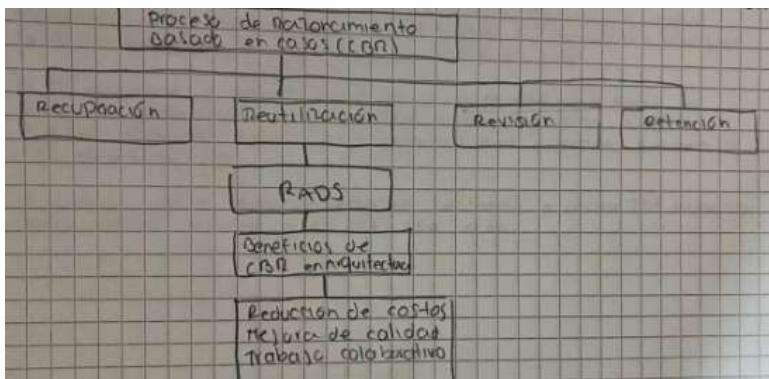


## Bibliografía

View of UML-based Scheme for Software Architecture Representations. Icesi.edu.co. Published 2024. Accessed November 19, 2024.  
<https://n9.cl/jo7i9>

# Revisión de Elementos para Arquitecturas de Software.

Este artículo explora los conceptos clave para representar arquitecturas de referencia de software (ARS). La arquitectura define la estructura de un sistema, estableciendo componentes y relaciones. Se destacan lenguajes de descripción arquitectónica (ADL) como UniCon, Wright y Darwin, que modelan, analizan y simulan arquitecturas, pero presentan limitaciones en reutilización y adaptación. También se mencionan marcos como RM-ODP y TOGAF, que estructuran la arquitectura en vistas, mejorando la comunicación. Se propone mejorar la representación del conocimiento arquitectónico para apoyar decisiones y razonamientos.



## Reflexión

Este artículo resalta la importancia de estandarizar y formalizar la arquitectura de software para mejorar la comunicación entre equipos y garantizar la integridad de los sistemas. Con la creciente complejidad de los sistemas, contar con modelos y marcos de referencia bien definidos facilita la adaptación y evolución sin comprometer la calidad. Se subraya la necesidad de herramientas y lenguajes específicos que permitan representar arquitecturas de manera precisa, fomentando la reutilización y adaptación a nuevos requerimientos, mejorando el diseño y desarrollo de sistemas.

## Bibliografía

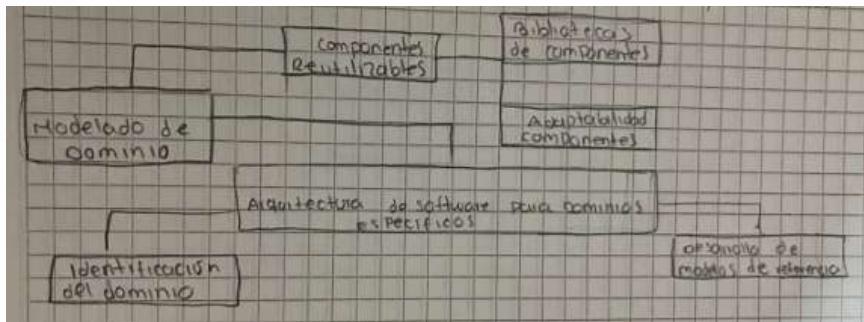
Angel M, Martínez, Nemury Silega, Yarisbel O, Angel M, Martínez, Nemury Silega, Yarisbel O. Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software. Revista Cubana de Ciencias Informáticas. 2019;13(1):143-157. Accessed November 19, 2024. [http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci\\_arttext](http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci_arttext)

# Razonamiento sobre Decisiones de Diseño de Software.

Este artículo explora cómo la arquitectura de software organiza el desarrollo de sistemas, facilitando la comunicación, el análisis y el cumplimiento de requisitos. Se presenta el Razonamiento Basado en Casos (CBR) como metodología para reutilizar conocimientos previos en nuevos desafíos de diseño. El proceso CBR incluye actividades iterativas como recuperación, reutilización, revisión y retención de casos arquitectónicos, lo que reduce costos, mejora la calidad y facilita la colaboración. Además, se detalla RADS, una herramienta que implementa CBR para apoyar decisiones arquitectónicas.

## Reflexión

Este artículo resalta la importancia de una base de conocimiento estructurada y accesible para los arquitectos de software, promoviendo la reutilización de experiencias exitosas y la mejora continua. El Razonamiento Basado en Casos (CBR) reduce la carga de trabajo y fomenta un enfoque colaborativo mediante una memoria compartida de casos, mejorando la coherencia de los diseños. La propuesta destaca la necesidad de herramientas para formalizar y capturar el conocimiento arquitectónico, optimizando la eficiencia y calidad, e integrando prácticas de inteligencia artificial para mejorar la toma de decisiones.



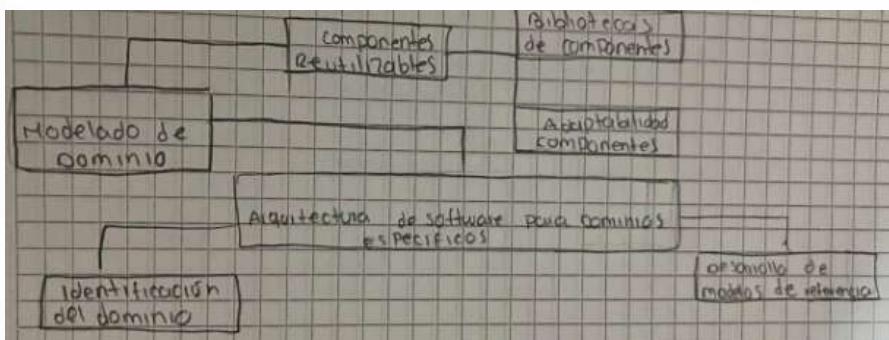
## Bibliografía

María A, Carignano C. Representación Y Razonamiento Sobre Las Decisiones de Diseño de Arquitectura de Software. Accessed November 19, 2024.

<https://n9.cl/umj6h>

# Arquitectura de Software para Dominio Específico y Reutilización.

Este artículo presenta las Arquitecturas de Software para Dominios Específicos (DSSA) como herramientas clave para gestionar la reutilización de software. Las DSSA capturan similitudes entre sistemas dentro de un dominio, permitiendo ensamblar y reutilizar componentes predefinidos, optimizando el desarrollo. Mediante el análisis de dominios, se identifican elementos comunes y relaciones, reduciendo costos y mejorando la eficiencia. Se describe un marco de trabajo en cinco etapas para construir DSSA, facilitando software adaptable y bibliotecas de componentes reutilizables.



## Reflexión

Este artículo resalta que las arquitecturas específicas para dominios (DSSA) maximizan la reutilización y optimización en el desarrollo de software. Las DSSA permiten aplicar conocimiento acumulado mediante componentes reutilizables, reduciendo la necesidad de crear desde cero y minimizando errores. Fomentan la innovación al centrarse en mejorar componentes existentes, además de fortalecer la colaboración y transferencia de conocimientos. Las DSSA son una herramienta estratégica que mejora el ciclo de vida del software, adaptándose a los cambios tecnológicos.

## Bibliografía

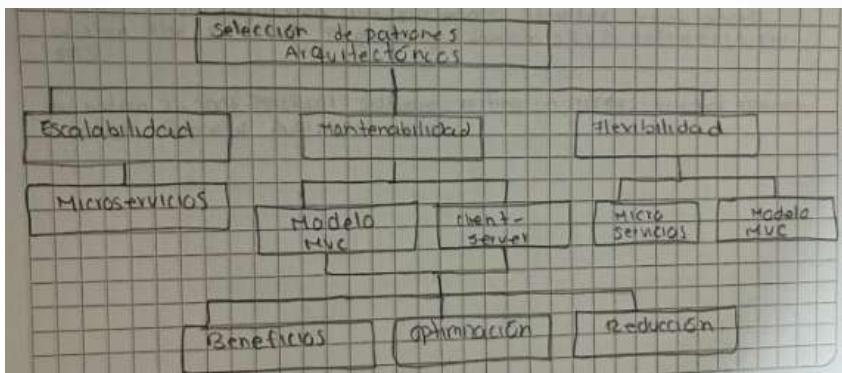
Oviedo S, Araya D, Zapata SG, Cáceres A. Arquitectura de software para dominio específico (DSSA): su aplicación en la reutilización de software. Unlpeduar. Published online May 1999. doi:<http://sedici.unlp.edu.ar/handle/10915/22260>

# Captura de Conocimientos para Diseño de Arquitectura.

El artículo destaca la importancia de los patrones de arquitectura de software en el diseño de sistemas eficaces. Debido a la dispersión del conocimiento sobre estos patrones, el estudio propone un modelo de toma de decisiones que vincula patrones con atributos de calidad específicos. Tras una revisión sistemática de la literatura, identifican 29 patrones y sus impactos en 40 atributos de calidad, ayudando a los arquitectos a tomar decisiones informadas. El estudio subraya la importancia de la adaptabilidad en la arquitectura para cumplir con necesidades tecnológicas cambiantes.

## Reflexión

El artículo resalta la importancia del conocimiento organizado en el diseño arquitectónico. Consolidar la información sobre patrones arquitectónicos permite a los arquitectos tomar decisiones informadas, cumpliendo tanto con los requisitos funcionales como de calidad. La investigación subraya que decisiones adaptables y basadas en conocimiento mejoran la calidad y la mantenibilidad de los sistemas de software. Además, fomenta una visión estratégica donde las elecciones arquitectónicas se alinean con la evolución tecnológica, beneficiando proyectos actuales y futuros.

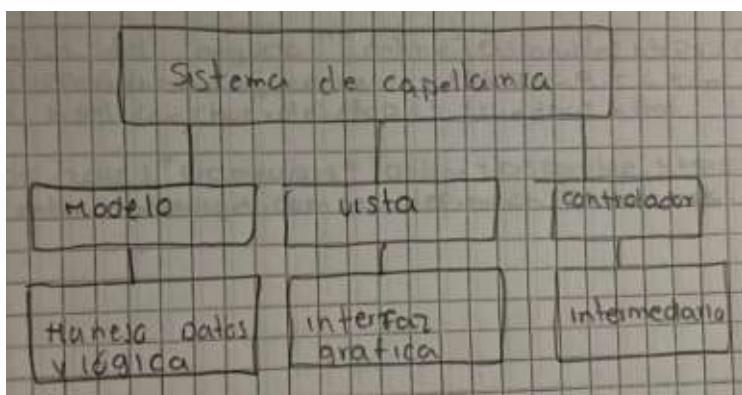


## Bibliografía

Siamak Farshidi, Jansen S, Martijn J. Capturing software architecture knowledge for pattern-driven design. Journal of Systems and Software. 2020;169:110714-110714. doi:<https://doi.org/10.1016/j.jss.2020.110714>

# Implementación del patrón MVC en el sistema de capellanía.

El artículo describe la implementación del patrón arquitectónico MVC en una aplicación web para el sistema de capellanía de la Universidad de Montemorelos, mejorando la gestión de citas y datos de estudiantes. Con un enfoque ecológico, la aplicación permite a los capellanes organizar entrevistas sin depender del papel. Utilizando AJAX, Bootstrap y PHP, la aplicación se mantiene modular, fácil de mantener y adaptable a nuevas funciones. El patrón MVC divide el sistema en tres componentes: modelo, vista y controlador, facilitando su desarrollo y mantenimiento.



## Reflexión

Este artículo reflexiona sobre la importancia de adaptar tecnologías a las necesidades de instituciones educativas y religiosas, como el sistema de capellanía. La aplicación del patrón MVC mejora la estructura y mantenimiento, permitiendo a los capellanes centrarse en la orientación espiritual sin depender de herramientas anticuadas. Destaca cómo la tecnología puede humanizar y facilitar el seguimiento de estudiantes, promoviendo relaciones más cercanas. Además, subraya la seguridad en el manejo de datos y la eficiencia de un sistema modular que facilita futuras actualizaciones.

## Bibliografía

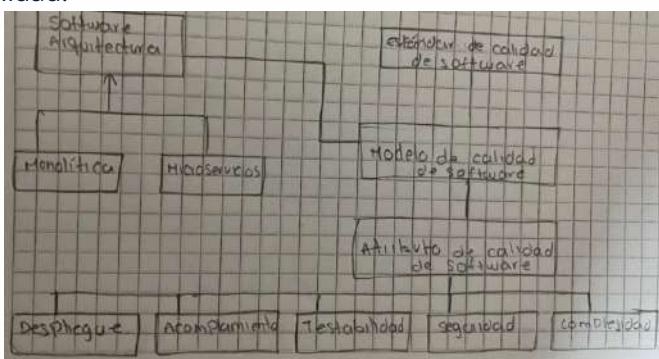
Yair P. Implementación del patrón arquitectónico MVC en aplicaciones web para la arquitectura del software del sistema de capellanía de la UM. Anuario de Investigación UM. 2020;1(1):112-120. Accessed November 19, 2024.  
<https://n9.cl/eii0f2>

# Modelo de Calidad para Optimización de Arquitecturas.

El artículo presenta un modelo centrado en la calidad para optimizar la arquitectura de software, comparando arquitecturas monolíticas y de microservicios. Evalúa atributos como acoplamiento, testabilidad, seguridad, complejidad, despliegue y disponibilidad, usando Jakarta EE y Spring. Propone un modelo matemático para tomar decisiones que optimicen la calidad, combinando elementos monolíticos y de microservicios según las necesidades del sistema. Concluye que las decisiones deben priorizar los atributos de calidad y someterse a evaluaciones continuas para asegurar la sostenibilidad y adaptabilidad.

## Reflexión

Este artículo invita a reflexionar sobre la arquitectura de software como una decisión estratégica que impacta en la calidad, sostenibilidad y adaptabilidad del sistema. Resalta la importancia de una arquitectura flexible que se ajuste a cambios y necesidades emergentes, equilibrando la simplicidad de diseños monolíticos con la escalabilidad de microservicios mediante optimizaciones basadas en la calidad. El enfoque propuesto permite a los desarrolladores tomar decisiones informadas que optimicen rendimiento, seguridad y satisfagan las expectativas de los usuarios.

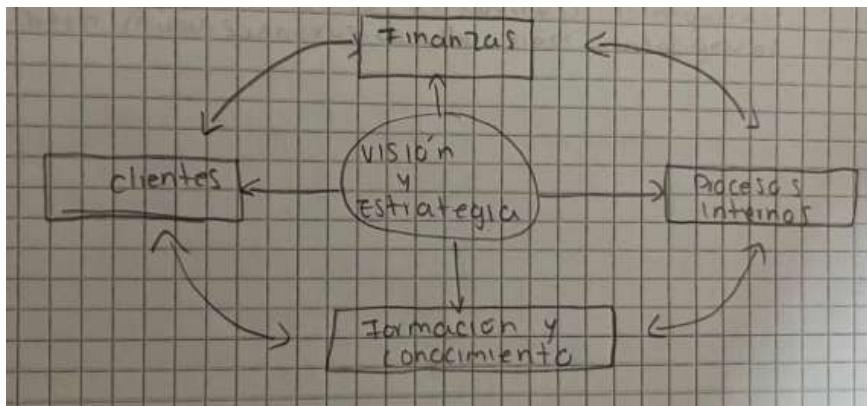


## Bibliografía

Miloš Milić, Dragana Makajić-Nikolić. Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. *Symmetry*. 2022;14(9):1824-1824. doi:<https://doi.org/10.3390/sym14091824>

# Arquitectura de Software para Cuadro de Mando.

Este artículo explora el papel crucial de la arquitectura de software en los sistemas de inteligencia de negocios, enfocándose en el cuadro de mando integral (CMI) como herramienta para optimizar la toma de decisiones empresariales. Utilizando un datawarehouse, la arquitectura organiza datos de diversas fuentes, mejorando la eficiencia en el análisis. El modelo de Kimball facilita la creación de una infraestructura de datos flexible. El CMI permite evaluar estrategias desde diferentes perspectivas, contribuyendo al rendimiento y crecimiento sostenido de las empresas.



## Reflexión

Este artículo destaca que la arquitectura de software y las herramientas de inteligencia de negocios transforman el enfoque estratégico de las organizaciones hacia una visión integrada y adaptable. La implementación del cuadro de mando integral permite gestionar recursos de manera óptima, evaluando el impacto de las decisiones en áreas clave como finanzas, procesos, clientes y aprendizaje. Estos sistemas no solo organizan datos, sino que también generan información valiosa que se convierte en conocimiento estratégico para las empresas.

## Bibliografía

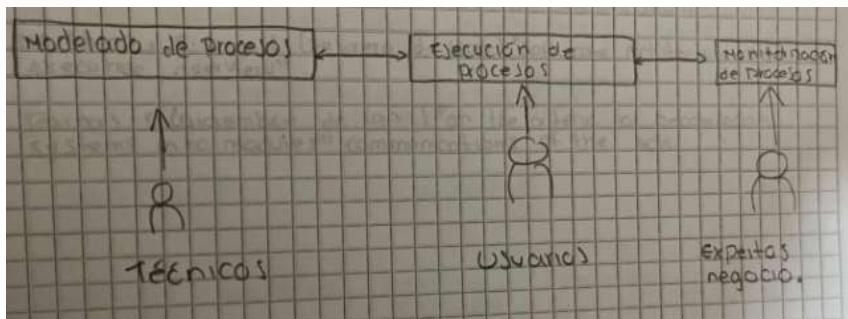
- Adolfo G. Arquitectura de software para la construcción de un sistema de cuadro de mando integral como herramienta de inteligencia de negocios. *Tecnología Investigación y Academia*. 2017;5(2):143-152. Accessed November 19, 2024. <https://revistas.udistrital.edu.co/index.php/tia/article/view/8766>

# Arquitectura de software. Arquitectura orientada a servicios

El artículo aborda la Arquitectura Orientada a Servicios (SOA) como una filosofía clave en la evolución del software, destacando su capacidad para garantizar interoperabilidad, separación de procesos y alineación con las necesidades del negocio. SOA permite la reutilización de servicios, la integración eficiente y la flexibilidad ante cambios tecnológicos. Beneficia la toma de decisiones, mejora la productividad y optimiza relaciones con clientes y proveedores. Además, se exploran conceptos como BPM, el Bus de Servicios y repositorios, esenciales para la automatización y ejecución de procesos.

## Reflexión

Este artículo destaca que la Arquitectura Orientada a Servicios (SOA) transforma el desarrollo de software, enfocándose en agilidad, integración y escalabilidad. SOA optimiza procesos y alinea objetivos tecnológicos con estrategias de negocio. La reutilización de componentes y servicios ofrece a las empresas una ventaja competitiva, permitiéndoles adaptarse rápidamente al mercado. Además, el enfoque en estándares abiertos e interoperabilidad fomenta un entorno colaborativo donde los sistemas evolucionan de forma orgánica.

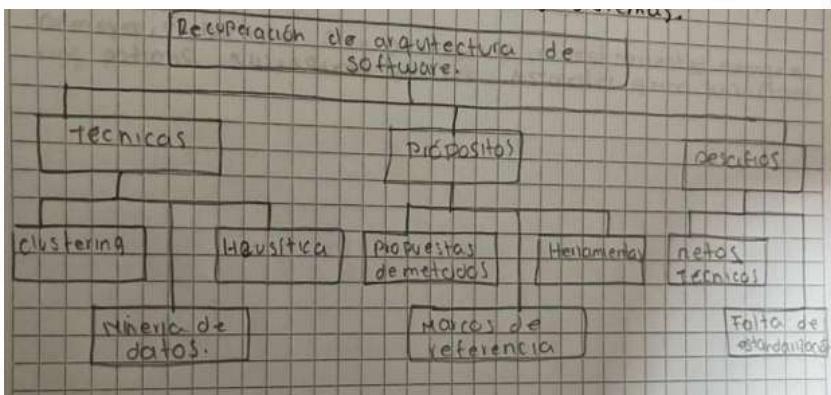


## Bibliografía

Yanet Espinal Martín. Arquitectura de software. Arquitectura orientada a servicios. Serie Científica de la Universidad de las Ciencias Informáticas. 2024;5(1):1-10.  
doi:<https://dialnet.unirioja.es/descarga/articulo/8590088.pdf>

# Recuperación de Arquitecturas de Software: Mapeo Literario.

Este artículo presenta un mapeo sistemático sobre la recuperación de arquitecturas de software, analizando 4050 documentos. Se identifican técnicas como clustering, heurística y minería de datos. La investigación aborda desafíos como la falta de estandarización en la representación de vistas arquitectónicas y los problemas de integración de resultados. Se destacan modelos basados en grafos, UML y ACME. Finalmente, se propone un catálogo de técnicas para maximizar su utilidad, facilitando la reutilización y el análisis en contextos específicos.



## Reflexión

Este artículo reflexiona sobre la importancia de abordar los desafíos en la recuperación de arquitecturas de software desde una perspectiva colaborativa entre la investigación y la industria. La falta de estandarización limita la reutilización y la interoperabilidad de herramientas. La propuesta de un catálogo unificado es crucial, pero su éxito dependerá de su adopción práctica. Además, la creciente complejidad de los sistemas requiere enfoques más robustos que combinen técnicas dinámicas y estáticas para capturar la estructura y el comportamiento del software.

## Bibliografía

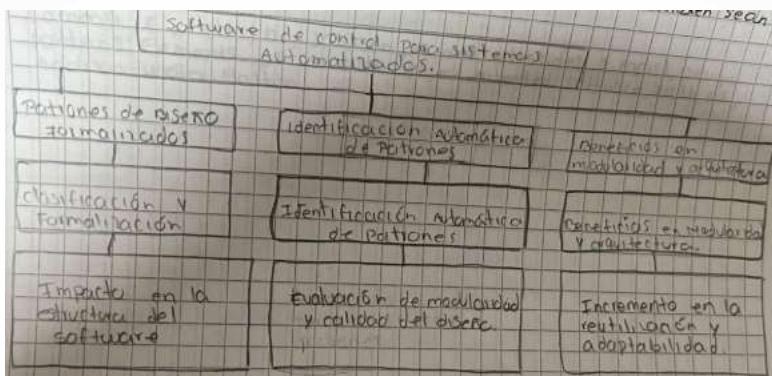
Monroy ME, Arciniegas JL, Rodríguez JC. Recuperación de Arquitecturas de Software: Un Mapeo Sistemático de la Literatura. Información tecnológica. 2016;27(5):201-220. doi:<https://doi.org/10.4067/s0718-07642016000500022>

# Formalización e Identificación de Patrones en Software PLC.

El artículo aborda la formalización e identificación automática de patrones de diseño en software de control para sistemas de producción automatizados (aPS). Dado el aumento de la complejidad por la personalización masiva, se propone un enfoque para clasificar y formalizar patrones estructurales comunes, evaluando su impacto en la modularidad y arquitectura del sistema. Un prototipo demuestra cómo identificar estos patrones automáticamente, mejorando la calidad del diseño, la reutilización y adaptabilidad del software, y optimizando el proceso de desarrollo.

## Reflexión

Este artículo reflexiona sobre cómo la formalización de patrones de diseño en software para sistemas industriales favorece la modularización y una arquitectura más sostenible y escalable. La identificación automática de patrones mejora la calidad del diseño, minimiza errores y optimiza el proceso. En entornos industriales, esto representa un avance hacia una ingeniería de software eficiente y reutilizable. Además, la modularidad no solo es una ventaja técnica, sino estratégica, al permitir a las organizaciones adaptarse ágilmente a cambios del mercado y demandas de personalización.

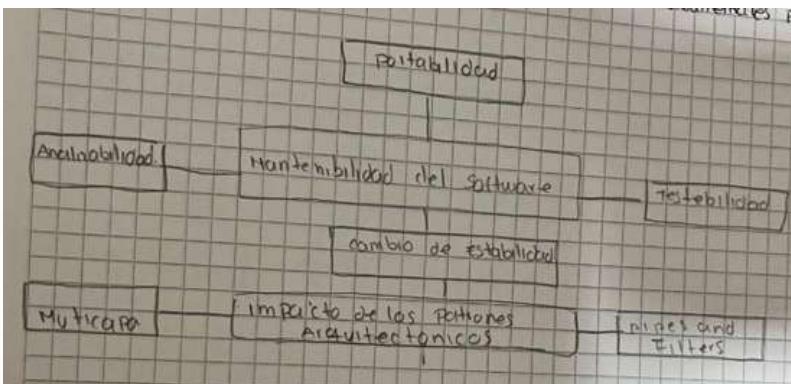


## Bibliografía

Neumann EM, Vogel-Heuser B, Fischer J, Ocker F, Diehm S, Schwarz M. Formalization of Design Patterns and Their Automatic Identification in PLC Software for Architecture Assessment. IFAC-PapersOnLine. 2020;53(2):7819-7826. doi:<https://doi.org/10.1016/j.ifacol.2020.12.1881>

# **Garantizar la Mantenibilidad con Patrones Arquitectónicos.**

El artículo destaca la importancia de la arquitectura de software para mejorar la mantenibilidad, enfocándose en cómo los patrones arquitectónicos impactan atributos clave como analizabilidad, estabilidad y testabilidad. Se propone un modelo de calidad para evaluar la mantenibilidad, desglosando atributos en subatributos medibles. Se exploran patrones como multicapa y pipes and filters, evaluando su impacto. Además, se discuten experiencias de refactorización, resaltando que una buena arquitectura reduce costos de mantenimiento, pero una implementación inadecuada puede ser contraproducente.



## **Reflexión**

El artículo resalta que la arquitectura de software es crucial para la sostenibilidad y evolución de los sistemas informáticos. Elegir patrones arquitectónicos adecuados facilita la mantenibilidad, eficiencia y adaptabilidad. No obstante, requiere un análisis detallado para evitar efectos adversos de decisiones incorrectas. Además, se subraya la necesidad de un enfoque proactivo que integre herramientas para medir y optimizar atributos de calidad desde el inicio del diseño. La refactorización muestra cómo una arquitectura más estructurada ahorra tiempo y recursos.

## **Bibliografía**

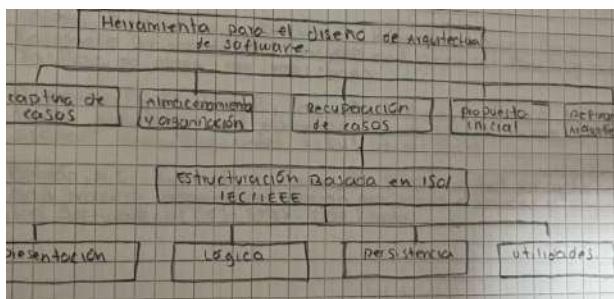
- Rahmati Z, Tanhaei M. Ensuring software maintainability at software architecture level using architectural patterns. 2021;2(1):81-102. doi:<https://doi.org/10.22060/ajmc.2021.19232.1044>

# Herramienta para Reutilizar Estrategias en Arquitecturas.

El artículo presenta RADS, una herramienta innovadora para diseñar arquitecturas de software basada en el razonamiento por casos. Permite capturar, almacenar y reutilizar estrategias de diseño según el estándar ISO/IEC/IEEE 42010:2011. RADS compara escenarios y restricciones de diseño actuales con casos previos para generar propuestas iniciales que los arquitectos pueden refinar. Opera en cuatro capas: presentación, lógica, persistencia y utilidades, y se integra con herramientas UML como Eclipse. Gestiona atributos de calidad y fomenta la sostenibilidad, ahorrando tiempo y facilitando la colaboración.

## Reflexión

El artículo resalta que la reutilización en el diseño de arquitecturas de software optimiza tiempo y recursos, fomenta la sostenibilidad y mejora la colaboración en la industria tecnológica. RADS ejemplifica cómo estructurar experiencias pasadas en un marco lógico transforma el enfoque de los problemas de diseño. Su integración con estándares internacionales y herramientas existentes enfatiza la importancia de innovaciones alineales con prácticas establecidas para potenciar su adopción y efectividad. Además, RADS destaca el valor del conocimiento compartido, brindando una base sólida para decisiones temporales.

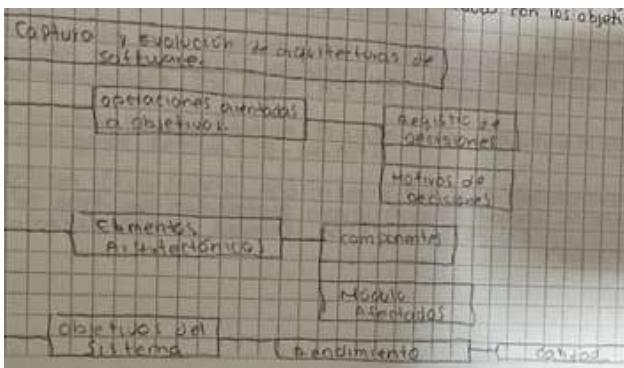


## Bibliografía

Celeste CM, Gonnet, Silvio M, Leone HP. RADS: una herramienta para reutilizar estrategias en diseños de arquitecturas de software. Unlpduar. Published online November 30, 2016. doi:<http://sedici.unlp.edu.ar/handle/10915/57164>

# Captura y Evolución de Arquitecturas con Operaciones OOP.

El artículo presenta un modelo para capturar y documentar la evolución y el razonamiento en el diseño de arquitecturas de software, crucial para sistemas complejos. Se basa en operaciones orientadas a objetivos (OpOOS) que registran decisiones y sus motivos, vinculándolas con metas como calidad, rendimiento o restricciones. Esto mejora la trazabilidad entre decisiones, productos arquitectónicos y objetivos. Además, se proponen herramientas para optimizar el proceso de diseño, ilustrado con la migración a la nube híbrida, reduciendo la pérdida de conocimiento en la evolución del sistema.



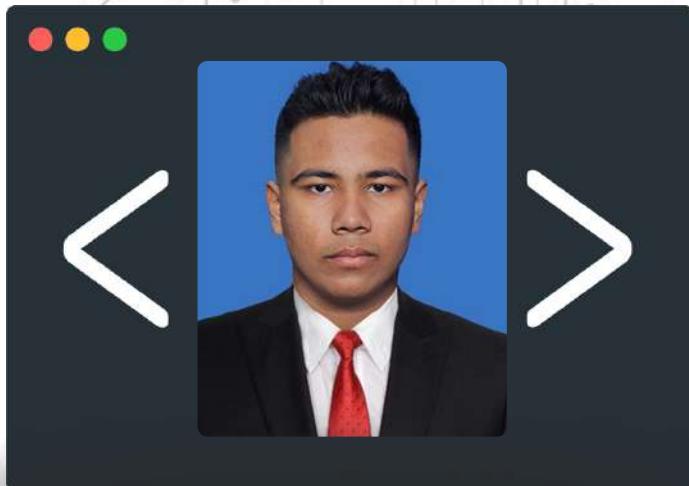
## Reflexión

El artículo reflexiona sobre la importancia de la documentación y trazabilidad en el diseño de arquitecturas de software, destacando que son posibles con enfoques sistemáticos y herramientas adecuadas. La incorporación de operaciones orientadas a objetivos permite a los arquitectos documentar decisiones y razonamientos de manera eficiente, sin aumentar la carga de trabajo. Esto mejora la calidad del diseño, facilita modificaciones futuras y fomenta prácticas sostenibles. Además, subraya la necesidad de considerar metas y restricciones desde el inicio, asegurando decisiones alineadas con los objetos.

## Bibliografía

Luciana RM, Gonnet, Silvio M, Leone HP. Captura del razonamiento y evolución de arquitecturas de software mediante la aplicación de operaciones arquitectónicas orientadas a objetivos. Unlpeduar. Published online 2015.  
doi:<http://sedici.unlp.edu.ar/handle/10915/52407>

# Camilo Andrés Bautista Cuellar



Soy Camilo Andrés Bautista Cuéllar, tengo 19 años y estoy estudiando un tecnólogo en análisis y desarrollo de software en el Sena. Desde siempre me ha apasionado explorar el mundo de la tecnología, investigando los avances que transforman nuestro día a día. Me encanta aprender, resolver problemas y desarrollar soluciones innovadoras que puedan marcar la diferencia.

Además, el deporte es una parte importante de mi vida; me ayuda a mantenerme activo, enfocado y equilibrado. Creo firmemente en la importancia de combinar el ejercicio físico con el trabajo intelectual para crecer tanto personal como profesionalmente.

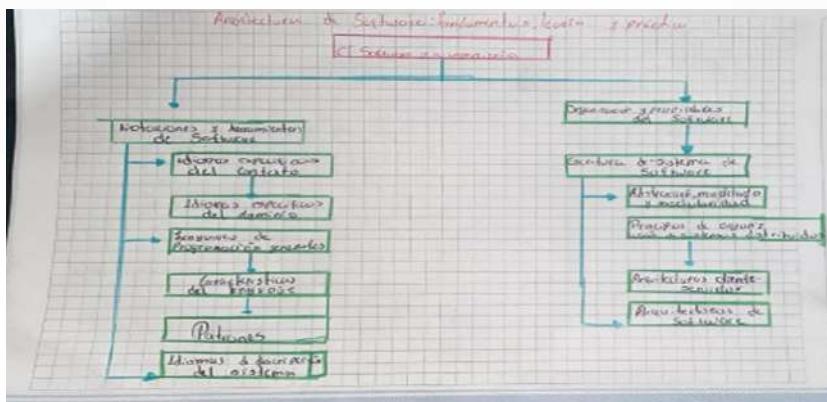
Mi meta es convertirme en un profesional destacado en el mundo de la tecnología, creando herramientas que aporten valor a las personas y contribuyan al desarrollo de la sociedad.

# Arquitectura de software: fundamentos, teoría y práctica

La arquitectura de software es esencial en el desarrollo de sistemas, pues guía su construcción y evolución mediante decisiones clave de diseño. Comprende aspectos estructurales, como componentes, conectores y configuraciones, además del despliegue, características no funcionales y patrones de cambio en tiempo de ejecución. Es especialmente valiosa para líneas de productos, al emplear estilos y patrones arquitectónicos reutilizables. Este tutorial detalla principios, elementos y prácticas, destacando tendencias emergentes y ejemplos prácticos como la arquitectura web y el estilo REST, mostrando su aplicación en la ingeniería de software.

## Reflexión

La arquitectura de software es fundamental para cualquier sistema, guiando decisiones críticas que afectan su desarrollo y mantenimiento. Abarca aspectos estructurales y propiedades no funcionales como rendimiento, seguridad y escalabilidad, asegurando que el sistema cumpla sus objetivos y se adapte a cambios tecnológicos y del negocio. En líneas de productos, la reutilización de patrones optimiza el desarrollo y garantiza consistencia. En aplicaciones como la Web, estilos como REST son clave para la escalabilidad e interoperabilidad, sirviendo como guía estratégica para maximizar el valor y la adaptabilidad del sistema.



## Bibliografía

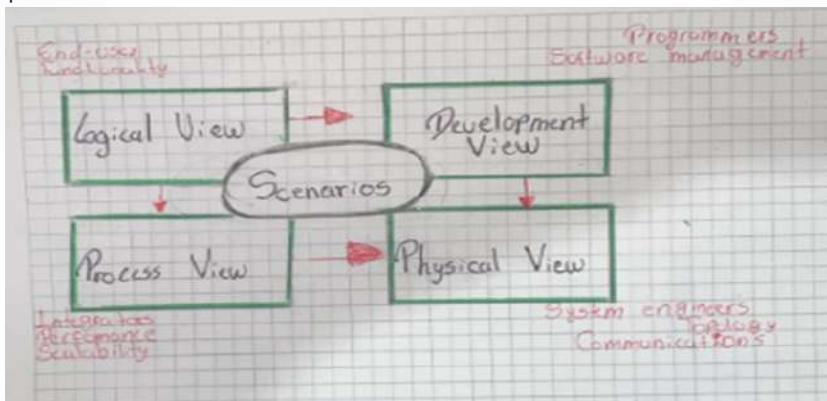
- Medvidovic, N., & Taylor, R. N. (2010, May). Software architecture: foundations, theory, and practice. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2 (pp. 471-472).

# El modelo de arquitectura 4+1 View

El modelo de vista 4+1 organiza la arquitectura de software en cinco perspectivas para capturar y validar decisiones de diseño. Las cuatro primeras vistas abordan aspectos clave del sistema: la vista lógica modela la estructura y comportamiento con diagramas de clases o entidades; la vista de proceso se centra en concurrencia y sincronización, crucial para procesos concurrentes; la vista física describe la asignación del software al hardware, resaltando arquitecturas distribuidas; y la vista de desarrollo organiza el código y componentes en el entorno de desarrollo. La quinta, la vista de escenarios, valida las anteriores simulando casos de uso para asegurar el cumplimiento de requisitos.

## Reflexión

El modelo de vista 4+1 organiza el diseño de software en perspectivas clave, asegurando una arquitectura coherente e integral. La vista lógica define la estructura y funcionalidad, la de proceso aborda la concurrencia, la física describe la asignación al hardware, y la de desarrollo organiza el código y componentes. La vista de escenarios valida las demás mediante simulaciones de casos reales, garantizando que los requisitos se cumplan. Este enfoque adaptable y robusto se ajusta a sistemas diversos, como los orientados a objetos o dependientes de datos, ofreciendo un marco sólido para arquitecturas complejas.



## Bibliografía

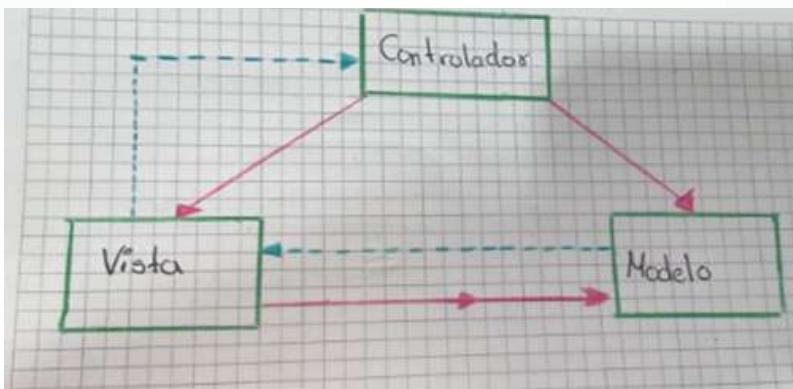
- Kruchten, P. B. (1995). The 4+1 view model of architecture. *IEEE software*, 12(6), 42-50.

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

El HTML es el lenguaje más usado para crear páginas web, definiendo su estructura mediante etiquetas para títulos, párrafos, listas, imágenes y más. Con tecnologías como XML, que permiten crear lenguajes como XHTML y RSS, surgió la necesidad de generar diversos formatos de salida, no solo HTML, para adaptarse a nuevas demandas. Hoy, los sistemas deben generar códigos para navegadores, plataformas móviles y feeds. Este proyecto implementa un framework que separa lógica de negocios, acceso a datos y presentación, facilitando la generación de formatos, simplificando el mantenimiento y evitando redundancias.

## Reflexión

La tecnología ha transformado cómo interactuamos en línea. Si bien HTML era suficiente en los inicios de la web, la diversificación en plataformas exige adaptar contenido a distintos formatos. Lenguajes como XML permiten un manejo flexible de información, facilitando el intercambio entre plataformas. Esto subraya la necesidad de sistemas escalables que se adapten sin perder eficiencia. Separar lógica de negocios, acceso a datos y presentación es clave para mejorar la mantenibilidad. Crear frameworks que simplifiquen aplicaciones multifuncionales optimiza el desarrollo y mejora la experiencia del usuario final.



## Bibliografía

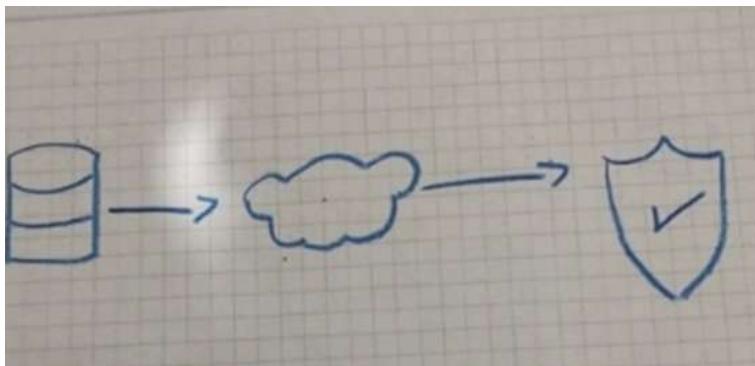
Zulian, E. R. (2011). Implementación de un framework para aplicaciones web con MVC/REST (Tesis doctoral, Univ. de Belgrano, Fac. de Tecnología Informática).

# **Arquitectura de Integración del Proceso de Descubrimiento de Conocimiento Con Sistemas de Gestión De Bases de Datos**

El crecimiento de los datos ha superado los métodos tradicionales de análisis, lo que ha creado la necesidad de nuevas herramientas para transformar datos en conocimiento útil. El Descubrimiento de Conocimiento en Bases de Datos (DCBD) es un proceso interactivo que identifica patrones útiles a partir de grandes volúmenes de datos. Incluye etapas como selección, preprocesamiento, minería de datos e interpretación de resultados, con el usuario involucrado en la toma de decisiones. La minería de datos es clave en este proceso, y las herramientas de DCBD integran técnicas de minería, consultas y visualización, resultando en arquitecturas acopladas con SGBD.

## **Reflexión**

El descubrimiento de conocimiento en bases de datos (DCBD) es fundamental para gestionar grandes volúmenes de datos. Con el crecimiento exponencial de la información, se requieren herramientas que analicen y extraigan patrones útiles eficientemente. Este proceso combina minería de datos e interpretación, con un enfoque interactivo donde el usuario guía el descubrimiento. La integración del DCBD con los sistemas de gestión de bases de datos (SGBD) optimiza el proceso, y las arquitecturas de integración, desde débilmente hasta fuertemente acopladas, proporcionan soluciones más efectivas, impulsando avances en áreas como ciencia y negocios.



## **Bibliografía**

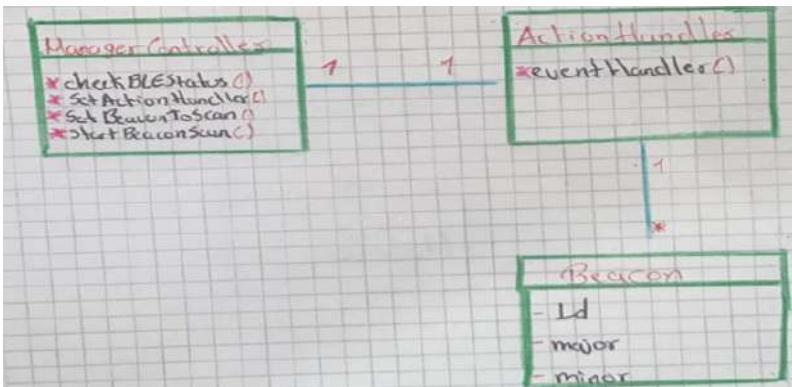
Timarán Pereira, R. (2001). Arquitecturas de integración del proceso de descubrimiento de conocimiento con sistemas de gestión de bases de datos: Un estado del arte. *Ingeniería y Competitividad*, 3(2).

# Patrón de Diseño Beacon Action Manager para comunicar Aplicaciones Móviles (IoT)

El Internet de las Cosas (IoT) ha conectado personas y objetos, permitiendo el procesamiento y recepción de información en tiempo real. Este avance ha hecho que los teléfonos inteligentes y las aplicaciones móviles sean fundamentales para entender el entorno de los usuarios. Con la aparición de IoT, surgen nuevos dispositivos como los beacons, que recopilan información contextual a través de aplicaciones móviles. Esto presenta desafíos para los desarrolladores, quienes deben adaptar sus aplicaciones para integrar estas tecnologías. El artículo presenta un patrón de diseño para aplicaciones móviles que interactúan con beacons, ofreciendo soluciones a los retos de integración y diseño.

## Reflexión

El Internet de las Cosas ha transformado nuestra interacción con el mundo, conectando personas y objetos para procesar información en tiempo real. Esta constante conexión presenta nuevas oportunidades, pero también desafíos para los desarrolladores de aplicaciones móviles, quienes deben adaptarse a tecnologías emergentes como los beacons. Estos dispositivos mejoran la experiencia del usuario al ofrecer interacciones personalizadas y contextuales. Sin embargo, esto exige un diseño eficiente que mantenga la usabilidad. La habilidad para crear aplicaciones que se adapten a este entorno será clave para agregar valor a los usuarios.



## Bibliografía

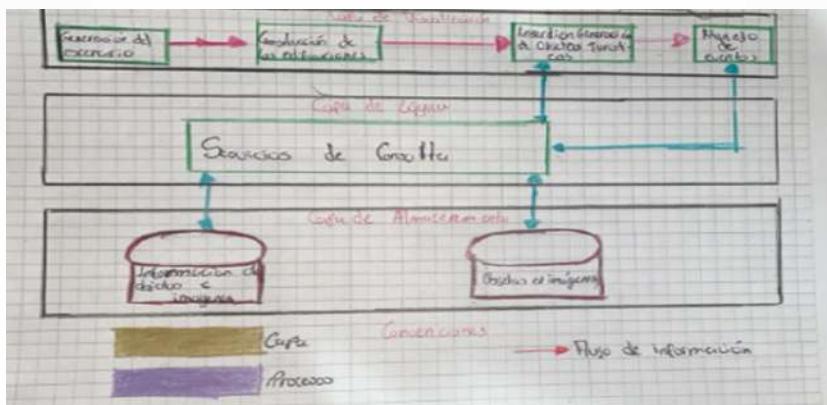
Yanina, B., Montejano, G., & Riesco, D. Patrón de Diseño Beacon Action Manager para comunicar Aplicaciones Móviles (IoT).

# Propuesta de una arquitectura software basada en realidad virtual para el desarrollo de aplicaciones de turismo cultural

El turismo cultural, clave para promover el patrimonio de las naciones, puede beneficiarse de la realidad virtual (VR) para ofrecer experiencias inmersivas. Este artículo propone una arquitectura de software para desarrollar aplicaciones de turismo cultural, usando VR para crear entornos virtuales. Se valida mediante un prototipo de museo virtual de Cartagena de Indias, que presenta una galería de fotos representativas. Los resultados muestran que esta arquitectura es útil para crear servicios VR en turismo, como recorridos y museos virtuales, adaptando el sector a las nuevas tecnologías y mejorando la accesibilidad y la interacción.

## Reflexión

La propuesta de una arquitectura de software basada en realidad virtual para el turismo cultural transforma la experiencia del patrimonio, ofreciendo acceso más amplio y flexible a la historia y el arte, sin barreras físicas. Los museos y recorridos virtuales no solo enriquecen la experiencia, sino que también ayudan a conservar los bienes culturales. Además, permite que personas de todo el mundo aprecien otras culturas sin necesidad de viajar. No obstante, es esencial que este avance complemente la experiencia presencial, manteniendo la autenticidad y el valor del patrimonio.



## Bibliografía

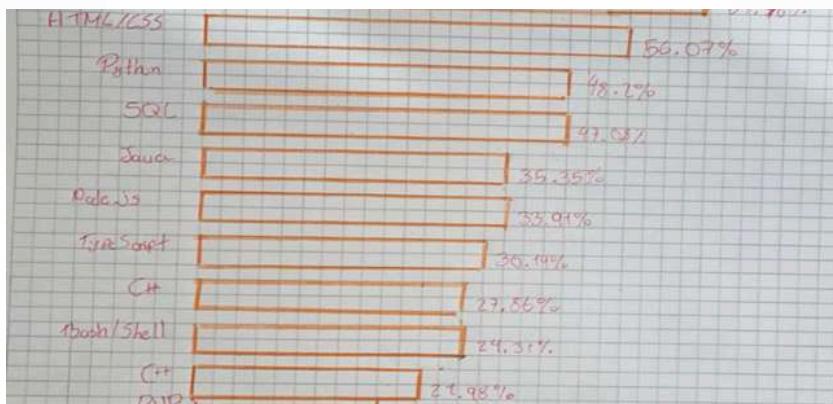
- Chanchí, G. E., Saba, M., & Monroy, M. (2020). Arquitectura de software basada en realidad virtual para aplicaciones de turismo cultural. Revista Ibérica de Sistemas y Tecnologías de la Información E, 36, 157-170.

# La Programación Reactiva: Un nuevo enfoque para trabajar con código asincrónico en la programación web

JavaScript es ampliamente utilizado en aplicaciones web y móviles, pero el manejo imperativo de eventos puede generar problemas de acoplamiento y dificultar la reutilización del código. La programación reactiva ofrece una alternativa mediante abstracciones como el Observable, basadas en patrones de diseño como Observer e Iterador. Este enfoque permite manejar eventos de manera más eficiente, mejorando la calidad, la organización y la escalabilidad del código. A través de un análisis, se destaca cómo la programación reactiva supera las limitaciones del enfoque imperativo y procedural.

## Reflexión

La adopción de JavaScript en diversas plataformas ha transformado el desarrollo web y móvil, pero su enfoque tradicional de manejo de eventos presenta limitaciones. La programación reactiva surge como una solución más eficiente, al reducir la complejidad en la gestión de eventos y promover un código más modular, reutilizable y fácil de mantener. Este cambio mejora la calidad técnica de los proyectos, optimiza el rendimiento, reduce errores y favorece una experiencia de desarrollo más organizada y predecible, beneficiando tanto a los desarrolladores como a las soluciones tecnológicas.



## Bibliografía

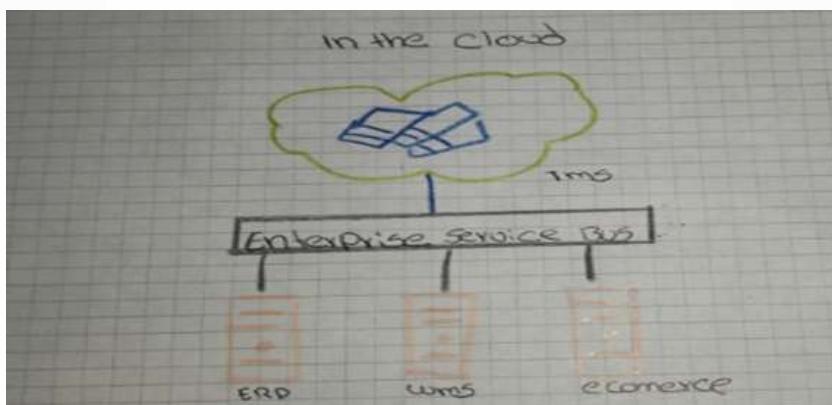
Fernández Vidal, M. (2022). La Programación Reactiva: Un nuevo enfoque para trabajar con código asincrónico en la programación web (Doctoral dissertation, Universidad Nacional de La Plata).

# Influencia de la Arquitectura de software en plataformas móviles de comercio electrónico

Los avances en tecnologías de información y comunicación han impulsado nuevos modelos de negocios, con aplicaciones móviles y comercio electrónico como claves para acceder a Internet y adquirir productos o servicios. Estas aplicaciones evolucionan rápidamente y las arquitecturas de software robustas son esenciales para fortalecer los sistemas en dispositivos móviles. Este documento analiza los tipos de aplicaciones comerciales, los requisitos técnicos de dispositivos y sitios web de comercio online, y aspectos de diseño y seguridad. Se destaca cómo una buena arquitectura contribuye a la eficacia de las aplicaciones móviles como alternativa al comercio tradicional.

## Reflexión

La evolución de la tecnología en aplicaciones móviles y comercio electrónico ha transformado la manera en que consumimos productos y servicios. Este cambio presenta tanto oportunidades como desafíos, ya que las empresas deben adaptarse a usuarios que exigen plataformas rápidas, seguras y confiables. Una arquitectura de software sólida y buenas prácticas de diseño no solo fortalecen las aplicaciones, sino que también construyen confianza en el usuario. Reflexionar sobre estos avances resalta cómo el comercio en línea y las aplicaciones móviles redefinen los negocios, acercando a consumidores y empresas en un entorno global e interconectado.



## Bibliografía

- Buenaño, D. P. C., Taco, C., Morejón, M. A. E., & Ramos, E. N. A. (2023). Influencia de la Arquitectura de software en plataformas móviles de comercio electrónico en Ecuador. Polo del Conocimiento: Revista científico-profesional, 8(6), 889-901.

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

Este análisis aborda el uso de los patrones de diseño de The Gang of Four (GOF) en el desarrollo de software orientado a la web. Se establecieron criterios rigurosos para evaluar procesos de desarrollo, asegurando su validez metodológica. A partir de una muestra representativa, se inspeccionó el código fuente para identificar la aplicación de estos patrones. Los resultados indican que, aunque se usan en la industria, su implementación es limitada debido al desconocimiento y la falta de experiencia en su correcta aplicación. Esto subraya la necesidad de mejorar la capacitación sobre estos patrones en el sector del software.

## Reflexión

Este análisis sobre el uso de patrones de diseño en el desarrollo web resalta la importancia de la capacitación continua en ingeniería de software. Los patrones de diseño mejoran la eficiencia, mantenibilidad y calidad de los sistemas, pero su uso limitado, debido al desconocimiento o falta de experiencia, muestra que no se aprovechan completamente. Esto subraya la necesidad de educación en metodologías avanzadas y fomenta una cultura de aprendizaje que pueda mejorar la calidad del software y facilitar la adaptación a las demandas del mercado.

Nro.	Patrón de Diseño	Categoría
1	Builder	Creacionales
2	Factory Method	Creacionales
3	Singleton	Creacionales
4	Decorator	Estructurales
5	Facade	Estructurales
6	Iterator	Estructurales
7	Strategy	De Comportamiento
8	Template Method	De Comportamiento

## Bibliografía

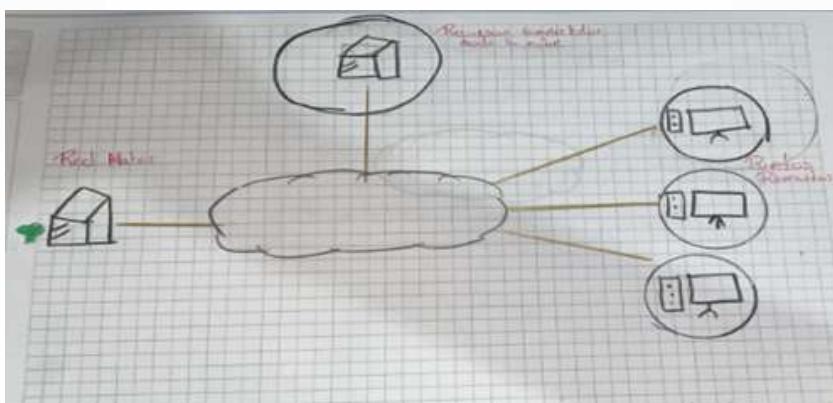
- Guerrero, C. A., Suárez, J. M., & Gutiérrez, L. E. (2013). Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web. *Información tecnológica*, 24(3), 103-114.

# Diseño de una arquitectura de seguridad informática para incrementar la seguridad de información en la empresa Bafing S.A.C.

Este proyecto muestra cómo una arquitectura de seguridad informática mejora la protección de la información en BAFING S.A.C., especializada en seguridad de la información. Se evaluó el impacto de la "Arquitectura de Seguridad Informática" en la "Seguridad de Información", enfocándose en la integridad, confidencialidad y disponibilidad de los datos. Se compararon software de seguridad, facilitando la planificación de su adquisición o renovación. Las entrevistas con expertos proporcionaron herramientas de evaluación, disponibles para futuras revisiones. Los resultados mostraron mejoras en la seguridad de la información, fortaleciendo la estrategia de protección de la empresa.

## Reflexión

Este proyecto destaca la importancia de una arquitectura de seguridad informática robusta para proteger la información, un activo clave para cualquier empresa. Al evaluar diversos componentes de seguridad, se demuestra que se puede mejorar la protección de datos y optimizar recursos al seleccionar las herramientas adecuadas. La experiencia de BAFING S.A.C. muestra cómo el análisis detallado y el uso de tecnologías avanzadas hacen de la seguridad un factor estratégico para el éxito y la sostenibilidad de la empresa, generando confianza en clientes y colaboradores.



## Bibliografía

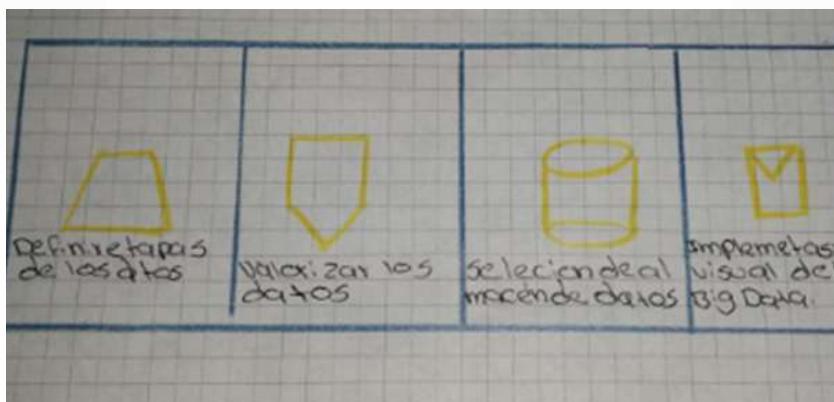
Asurza Caceares, J. D. (2022). Diseño de una arquitectura de seguridad informática para incrementar la seguridad de información en la empresa Bafing SAC en 2021.

# Arquitectura de consolidación de la información para seguros de la salud mediante Big Data

El trabajo tiene como objetivo desarrollar una arquitectura para consolidar la información en el sector de seguros de salud mediante Big Data, utilizando un enfoque empírico-analítico cuasi experimental con análisis cuantitativo. Se clasificarán diferentes arquitecturas computacionales aplicables al sector y se propondrá un modelo para una empresa ecuatoriana. Además, se evaluarán los factores viables para su implementación, concluyendo que el 73% de los factores son viables en empresas de seguros de salud en Ecuador. Esta arquitectura facilita el análisis de datos, mejora la toma de decisiones y proporciona indicadores clave para optimizar la gestión de seguros de salud.

## Reflexión

La implementación de Big Data en el sector de los seguros de salud transforma la gestión y análisis de datos, ofreciendo una visión más completa de las necesidades de los asegurados. Esto optimiza la toma de decisiones, mejora la eficiencia operativa y permite la personalización de servicios. No obstante, el éxito depende de la correcta integración y gestión de datos, así como de la capacidad de adaptación a nuevas tecnologías. A pesar de los retos, como la seguridad y la privacidad, Big Data ofrece amplias oportunidades para mejorar los servicios de salud y la calidad del servicio al asegurado.



## Bibliografía

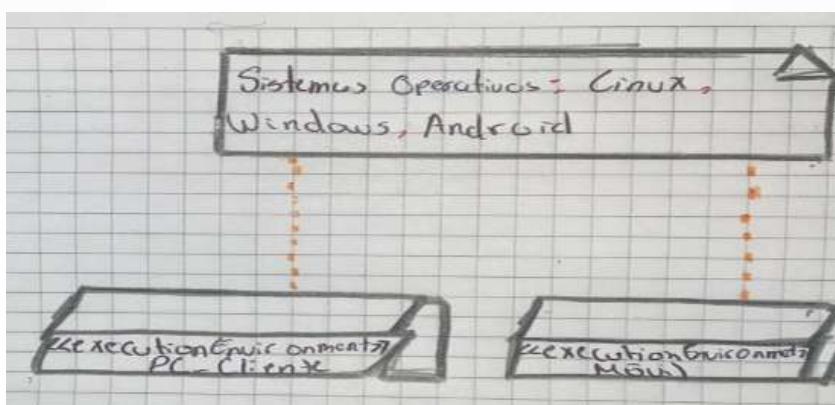
Data, B. Arquitectura de consolidación de la información para seguros de la salud mediante Big Data.

# Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D

Este estudio tiene como objetivo desarrollar una arquitectura de software para videojuegos creados con Unity 3D, organizando y estructurando sus funcionalidades básicas. A través del análisis de diversas arquitecturas de videojuegos, se agruparon clases relevantes, identificando paquetes principales, dependencias, patrones de diseño y buenas prácticas. La propuesta integra arquitecturas en capas y basadas en componentes. Un prototipo funcional de un videojuego de plataformas fue desarrollado, evaluado mediante el método de Análisis de Acuerdos de Arquitectura de Software, identificando riesgos y evaluando atributos de calidad según el modelo ISO/IEC 25010.

## Reflexión

La arquitectura de software es crucial tanto para el rendimiento técnico como para alinear el desarrollo con los objetivos estratégicos. En los videojuegos creados con Unity 3D, una arquitectura estructurada facilita la modularidad y flexibilidad, optimizando la gestión de cambios. Métodos como el análisis de acuerdos de arquitectura permiten anticipar riesgos y mejorar la calidad del software. Este enfoque planificado y reflexivo es esencial para el éxito de proyectos dinámicos como los videojuegos.



## Bibliografía

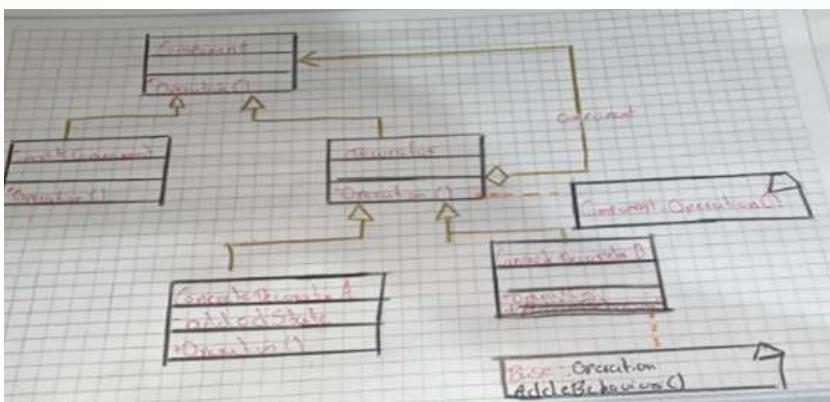
Hernández Páez, A., Domínguez Falcón, J., & Pi Cruz, A. (2018). Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D. I+D Tecnológico, 14 (1), 54-65.

# Comparación de Uso del Patrón de Diseño Decorator y la Programación Orientada a Aspectos en .NET para Modularizar Incumbencias Cruzadas

Este trabajo explora cómo desarrollar soluciones modulares en .NET utilizando el patrón de diseño Decorator y PostSharp, que implementa Programación Orientada a Aspectos (POA), para abordar incumbencias cruzadas como el logging. Se presentan ejemplos prácticos para aplicar ambas herramientas, detallando sus ventajas y desventajas. El estudio destaca que mientras el patrón Decorator y PostSharp logran mejorar la modularidad, PostSharp, mediante POA, ofrece una integración más eficiente de funcionalidades cruzadas. La comparación proporciona una guía para elegir la herramienta más adecuada según las necesidades del proyecto.

## Reflexión

La exploración de Decorator y PostSharp en .NET resalta la importancia de la modularización de incumbencias cruzadas, como el logging, mejorando la organización y mantenimiento del código. Decorator es adecuado para soluciones claras pero puede complicarse con capas adicionales. PostSharp, mediante POA, reduce el código repetitivo, aunque introduce mayor complejidad y dependencia de herramientas externas. La elección de la técnica depende del equilibrio entre simplicidad, eficiencia y las necesidades del proyecto, y combinarlas puede maximizar los beneficios.



## Bibliografía

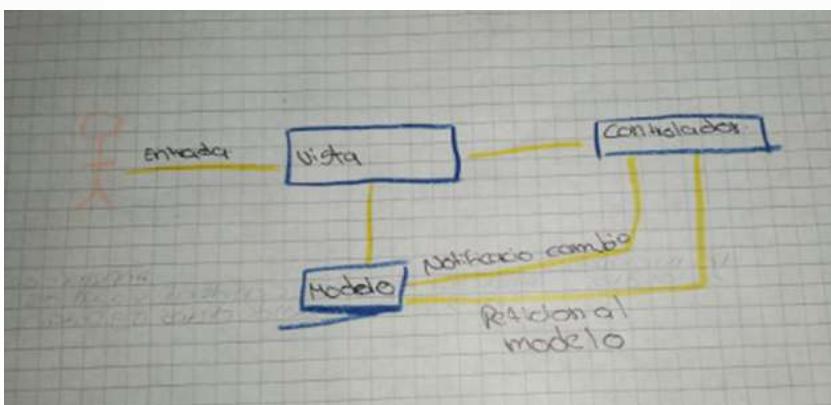
- Pereira-Vásquez, C. A., Vidal-Silva, C. L., & Morris, M. A. (2017). Comparación de Uso del Patrón de Diseño Decorator y la Programación Orientada a Aspectos en .NET para Modularizar Incumbencias Cruzadas. *Información tecnológica*, 28(5), 37-44.

# Arquitectura de software para sistemas de tiempo real particionados

Los sistemas de tiempo real críticos para misiones aeroespaciales deben cumplir estrictos requisitos de seguridad y fiabilidad debido a las condiciones extremas. Tradicionalmente, se utilizaban procesadores dedicados para separar los componentes de misión y control, lo que evitaba la propagación de fallos. Sin embargo, los procesadores más potentes actuales permiten ejecutar múltiples aplicaciones en un solo nodo, aunque las técnicas de planificación estáticas usadas hasta ahora limitan su eficiencia. Se propone una arquitectura con planificación dinámica que mejora la eficiencia y usa técnicas de monitorización para garantizar la seguridad, integrándose en modelos de componentes y facilitando el desarrollo.

## Reflexión

La propuesta de una arquitectura eficiente y segura para sistemas de tiempo real en misiones aeroespaciales responde a la necesidad de equilibrar innovación y fiabilidad. La evolución de procesadores más potentes ofrece la oportunidad de optimizar recursos, pero desafía las técnicas de planificación estática tradicionales. La integración de planificación dinámica y monitorización permite mejorar la eficiencia sin comprometer la seguridad, ofreciendo una solución moderna que responde a los altos estándares de la industria aeroespacial.



## Bibliografía

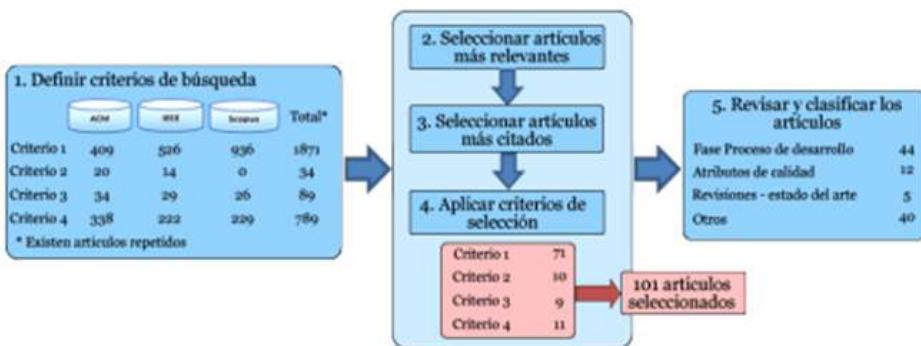
Pavón, J. A. P. (2007). Arquitectura de software para sistemas de tiempo real particionados (Doctoral dissertation, Universidad Politécnica de Madrid).

# Desarrollo de aplicaciones basadas en micro servicios: tendencias y desafíos de investigación

Los microservicios son una arquitectura de software compuesta por pequeños servicios independientes que se comunican mediante APIs bien definidas y protocolos ligeros, lo que permite crear sistemas más flexibles y escalables. Un análisis de la literatura revela que las principales tendencias en el desarrollo de microservicios incluyen la escalabilidad y la calidad del servicio. Los desafíos destacados son la definición de la granularidad de los microservicios, la modularización, la integración con interfaces de usuario, la seguridad, la orquestación, el monitoreo y la gestión, así como la tolerancia a fallas y la auto reparación para garantizar la robustez del sistema.

## Reflexión

Los microservicios ofrecen una arquitectura flexible y escalable, pero presentan desafíos como la granularidad, la seguridad y la modularización, que afectan la estabilidad y el rendimiento. La integración de orquestación, monitoreo y tolerancia a fallas es crucial, lo que implica un cambio cultural y organizativo. La recuperación y auto reparación son esenciales para la resiliencia del sistema. Aunque no es una solución universal, cuando se implementan correctamente, los microservicios transforman el desarrollo de software, respondiendo a las demandas de rapidez, calidad y flexibilidad.



## Bibliografía

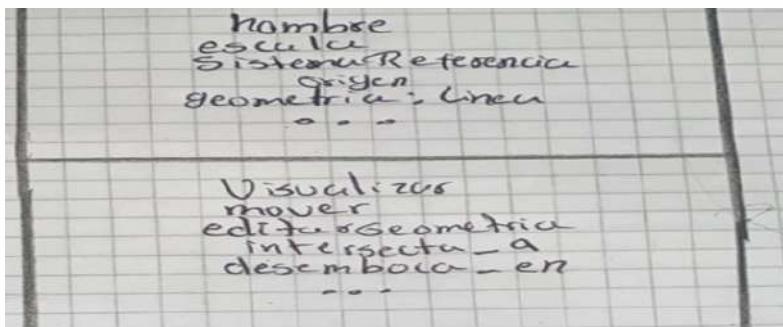
Vera-Rivera, F. H., Astudillo, H., & Gaona, C. (2019). Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación. RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação, (E23 (2019)), 107-120.

# Patrones de diseño para el modelo de redes en sistemas de información geográfica

Este artículo explora dos patrones de diseño que mejoran el desarrollo de sistemas de información geográfica (SIG) orientados a objetos. El patrón Objeto Espacial captura las características estructurales y dinámicas comunes de las entidades espaciales en distintos SIG, permitiendo un diseño unificado y coherente. El patrón Grafo Espacial se centra en la representación, manipulación y visualización de redes de servicios, proporcionando una solución reutilizable para modelar redes de forma precisa, donde cada nodo o enlace se define como un objeto espacial. Ambos patrones promueven la reutilización, simplifican el diseño de aplicaciones SIG y facilitan la gestión de datos espaciales y redes de servicios, ofreciendo herramientas útiles para desarrolladores y diseñadores en este campo.

## Reflexión

Los patrones de diseño presentados destacan la importancia de la reutilización y la estandarización en sistemas de información geográfica (SIG). El patrón Objeto Espacial facilita la abstracción de características comunes de las entidades espaciales, mejorando la flexibilidad. El patrón Grafo Espacial optimiza la representación y manipulación de redes de servicios, simplificando la visualización de datos espaciales. Ambos enfoques permiten diseñar aplicaciones SIG más escalables y eficientes, siendo fundamentales para el desarrollo de sistemas robustos y sostenibles.



## Bibliografía

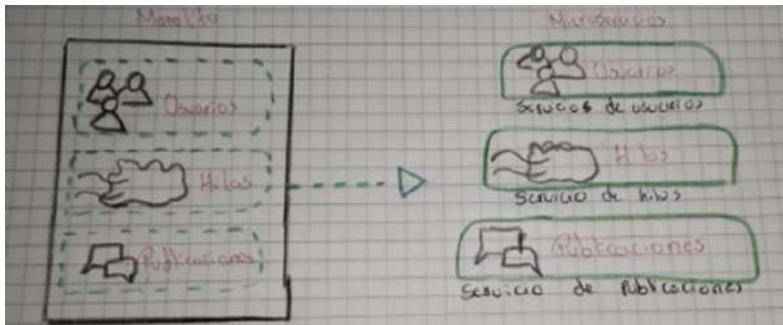
- Montilva, J. A., & Ramos, Y. (2000). Patrones de diseño para el modelo de redes en sistemas de información geográfica. Revista Colombiana de Computación, 1(1), 91-104.

# Análisis de patrones de resiliencia en una arquitectura basada en micro servicios

En los últimos años, las arquitecturas de micro servicios han ganado popularidad como respuesta a las limitaciones de los sistemas monolíticos, destacando la importancia de la resiliencia en grandes empresas. Este trabajo analiza los patrones de diseño para resiliencia, enfocados en mitigar los efectos negativos de fallos en sistemas como el ecosistema de micro servicios de Pedidos Ya. Particularmente, se estudió el micro servicio Niles, responsable de retornar menús de restaurantes, detallando su funcionamiento y las dependencias críticas para su operación. Se evaluaron patrones de resiliencia, como circuit breakers y timeouts, analizando su impacto en escenarios típicos de fallas.

## Reflexión

La implementación de arquitecturas de micro servicios ha transformado la manera en que las empresas gestionan y operan sus sistemas. Sin embargo, esta estructura distribuida también trae consigo nuevos desafíos, especialmente en lo que respecta a la resiliencia ante fallas. La capacidad de un sistema para seguir funcionando ante errores es crucial, ya que cualquier interrupción puede afectar directamente el negocio. A través de la experiencia de Pedidos Ya, se ha demostrado cómo los patrones de resiliencia pueden ser la clave para mitigar los efectos negativos de los fallos en los micro servicios. Este análisis no solo resalta la importancia de anticipar y gestionar posibles errores, sino también de diseñar soluciones que permitan a los sistemas adaptarse y mantenerse operativos en todo momento.



## Bibliografía

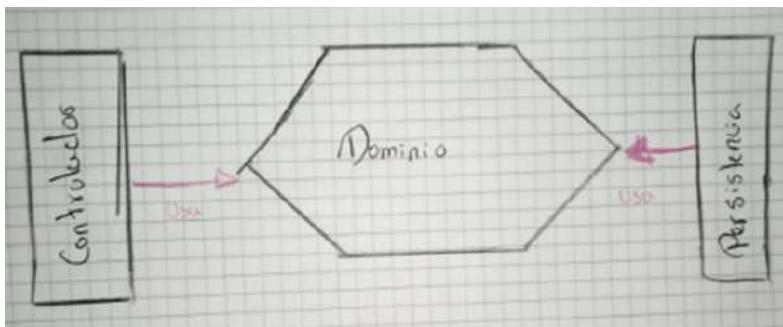
Suarez, S. L. (2023). Análisis de patrones de resiliencia en una arquitectura basada en microservicios (Doctoral dissertation, Universidad Nacional de La Plata).

# Implementación de una arquitectura de software guiada por el dominio

El diseño de software bajo un enfoque sólido y sistemático utiliza herramientas y técnicas que simplifican la complejidad del negocio, centrando su atención en el dominio. El diseño dirigido por el dominio (DDD) se basa en principios, patrones y actividades que permiten construir un modelo de dominio como pieza principal. Este enfoque asegura que la arquitectura de software esté alineada con las funcionalidades del negocio. El trabajo propuesto explora la implementación de una arquitectura orientada al dominio, mostrando su valor estratégico al mapear conceptos del negocio hacia los artefactos del software. Se presenta la adaptación de una arquitectura típica de tres capas hacia una centrada en el dominio específico del negocio.

## Reflexión

El diseño de software dirigido por el dominio (DDD) nos invita a reflexionar sobre la importancia de alinear la tecnología con las necesidades reales del negocio. Este enfoque no solo permite desarrollar soluciones más efectivas, sino que también fomenta una colaboración profunda entre expertos del dominio y desarrolladores, creando un lenguaje común que facilita la comunicación y el entendimiento mutuo. Además, priorizar el dominio en la arquitectura de software asegura que las funcionalidades se mantengan relevantes y adaptables frente a los cambios del entorno empresarial. La transformación de arquitecturas tradicionales hacia modelos centrados en el dominio no es solo un desafío técnico, sino también un ejercicio estratégico que redefine cómo entendemos y resolvemos problemas.



## Bibliografía

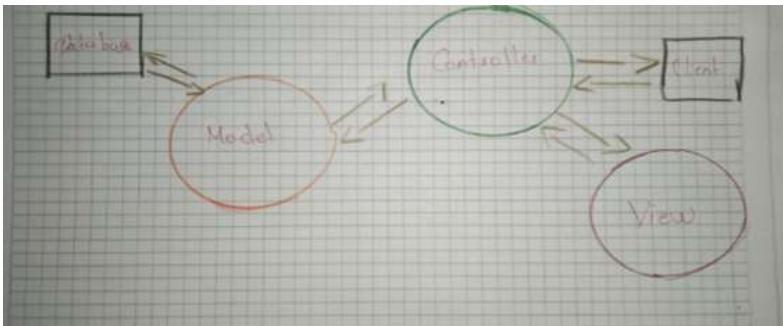
- Cambarieri, M., Difabio, F., & García Martínez, N. (2020). Implementación de una arquitectura de software guiada por el dominio. In XXI Simposio Argentino de Ingeniería de Software (ASSE 2020)-JAIIS 49 (Modalidad virtual).

# **Impacto del patrón modelo vista controlador (MVC) en la seguridad, interoperabilidad y usabilidad de un sistema informático durante su ciclo de vida**

El patrón de diseño Modelo-Vista-Controlador (MVC) es uno de los más utilizados en el desarrollo de software debido a su capacidad para mejorar la seguridad, interoperabilidad y usabilidad de los sistemas. Este patrón, clasificado como arquitectónico, ha sido ampliamente adoptado por programadores, lo que ha llevado a investigar sus impactos en el ciclo de vida del software. A través de una revisión bibliográfica y consultas en sitios especializados, se ha demostrado que MVC influye positivamente en estos aspectos, facilitando el desarrollo y mantenimiento de aplicaciones. La comunidad de desarrolladores lo considera indispensable.

## **Reflexión**

La adopción del patrón Modelo-Vista-Controlador (MVC) en el desarrollo de software refleja la importancia de estructurar correctamente los sistemas para garantizar su eficacia y sostenibilidad. Este enfoque no solo permite separar responsabilidades, sino que también mejora la claridad del código, facilitando la colaboración entre equipos y el mantenimiento del sistema a largo plazo. Además, su impacto positivo en la seguridad y usabilidad resalta cómo un diseño bien planificado puede prevenir problemas críticos desde las primeras etapas del desarrollo. La popularidad de MVC entre los desarrolladores experimentados subraya la necesidad de utilizar patrones que no solo sean funcionales.



## **Bibliografía**

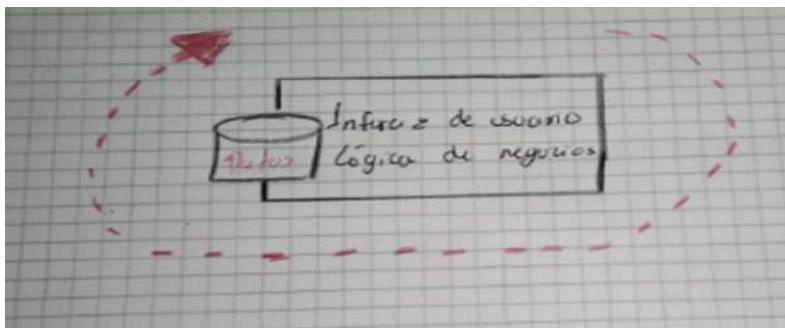
- Enríquez, F., Fierro, S., Flores, B., Esparza, D. I., & Michelena, J. (2023). Impacto del patrón modelo vista controlador (MVC) en la seguridad, interoperabilidad y usabilidad de un sistema informático durante su ciclo de vida. EASI: Ingeniería y Ciencias Aplicadas en la Industria, 2(1), 11-16.

# Migración semiautomática de sistemas Legacy hacia arquitecturas orientadas a servicios

Los sistemas legacy son software anticuado que sigue siendo crucial para muchas empresas, pero su actualización o integración con nuevas tecnologías suele ser compleja. Este trabajo propone un enfoque teórico-práctico para migrar sistemas legacy hacia arquitecturas orientadas a servicios (SOA) de forma semiautomática. La metodología incluye migrar desde tecnologías como Visual Basic o Delphi hacia plataformas modernas como Web o Mobile, apoyándose en un proceso adaptable a diversas tecnologías. El enfoque fue probado en un sistema legacy real y se desarrolló un prototipo de aplicación móvil que ejemplifica la integración eficiente con la arquitectura SOA.

## Reflexión

La migración de sistemas legacy hacia arquitecturas modernas, como SOA, refleja la necesidad de mantener la funcionalidad y relevancia de sistemas que, aunque antiguos, son fundamentales para los negocios. Este proceso no solo evita la obsolescencia tecnológica, sino que también ofrece oportunidades para mejorar la eficiencia, escalabilidad y adaptabilidad de las aplicaciones a nuevos entornos, como Web o Mobile. La propuesta de una migración semiautomática es especialmente valiosa, ya que reduce costos y riesgos asociados con un cambio total. Además, la implementación de un prototipo móvil demuestra que estas transformaciones no solo son teóricas, sino también prácticas y funcionales.



## Bibliografía

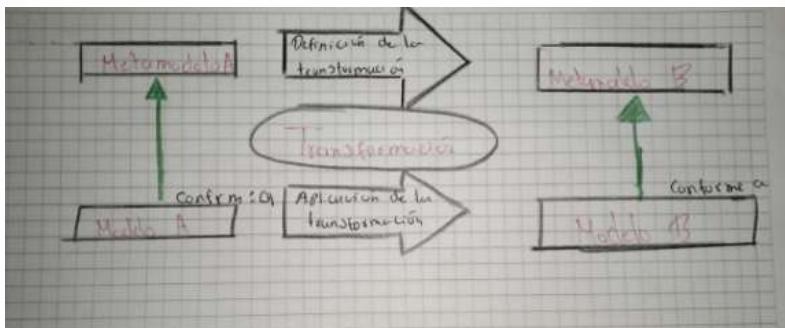
Barzola, M. E. (2017). Migración semiautomática de sistemas Legacy hacia arquitecturas orientadas a servicios (Doctoral dissertation, Universidad Nacional de La Plata).

# MDA y el papel de los modelos en el proceso de desarrollo de software

Los modelos desempeñan un papel crucial en el desarrollo de software al fomentar el reuso de elementos y facilitar el trabajo de los distintos roles involucrados. La Arquitectura Dirigida por Modelos (MDA) introduce un enfoque basado en la creación y transformación de modelos. Este enfoque se sustenta en tres principios clave: abstracción, automatización y estandarización. El proceso principal de MDA consiste en transformar modelos desde el espacio del problema (CIM) hacia modelos específicos de la plataforma (PSM), pasando por modelos independientes de la tecnología (PIM). Estos pasos permiten abordar el problema desde diferentes niveles de detalle y abstraer la solución antes de implementarla.

## Reflexión

La MDA refleja cómo la abstracción y la estandarización pueden transformar el desarrollo de software en un proceso más eficiente y escalable. Al permitir que los modelos evolucionen desde la conceptualización del problema hasta la implementación específica, se promueve un enfoque ordenado y reutilizable, donde la tecnología deja de ser una limitación inicial. Esto no solo optimiza el trabajo en equipo al asignar roles más claros, sino que también fomenta la innovación al liberar a los desarrolladores de las restricciones de una plataforma particular. En un mundo donde la complejidad del software crece, reflexionar sobre el papel de los modelos nos recuerda que el verdadero desafío está en estructurar soluciones antes de codificarlas.



## Bibliografía

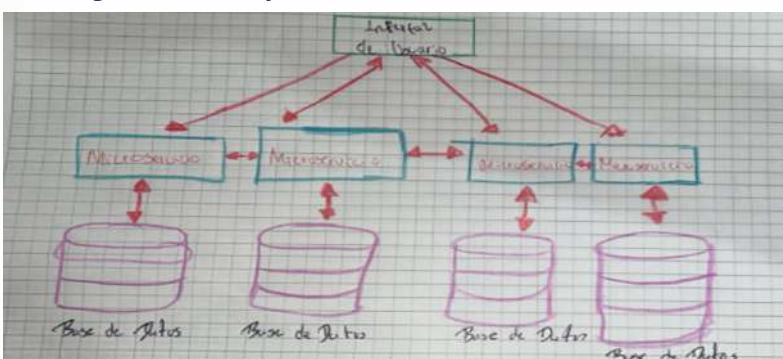
Quintero, J. B., & Anaya, R. (2007). MDA y el papel de los modelos en el proceso de desarrollo de software. revista EIA, (8), 131-146.

# Adaptación de un patrón de software en seguridad a la arquitectura de un Micro servicio

Este trabajo analiza los riesgos de seguridad asociados con la adopción de la arquitectura de micro servicios, destacando su creciente popularidad frente a las arquitecturas monolíticas tradicionales. Aunque los micro servicios ofrecen beneficios como modularidad, escalabilidad independiente y flexibilidad tecnológica, su naturaleza descentralizada incrementa la superficie de ataque y complica la gestión de seguridad. La investigación enfatiza la necesidad de un enfoque holístico para prevenir accesos no autorizados y proteger los sistemas. Además, se subraya la importancia de diseñar patrones de seguridad específicos para estas arquitecturas dinámicas, como el "micro servicios Security Pattern API Gateway" (MSPAG), desarrollado con tecnologías como JWT y H256.

## Reflexión

La adopción de la arquitectura de micro servicios plantea una dualidad entre las ventajas tecnológicas y los desafíos de seguridad. Aunque esta arquitectura potencia la escalabilidad, modularidad y flexibilidad en el desarrollo de software, también abre una superficie de ataque más amplia debido a su descentralización. Esto refleja la necesidad de priorizar la seguridad desde el diseño, ya que un enfoque reactivo puede ser insuficiente ante amenazas crecientes. La implementación de patrones de seguridad como MSPAG evidencia cómo la innovación tecnológica debe ir acompañada de soluciones robustas y especializadas para mitigar riesgos. En un mundo donde la transformación digital avanza rápidamente, garantizar sistemas seguros no solo protege los datos, sino que también fomenta la confianza de usuarios y empresas en arquitecturas modernas.



## Bibliografía

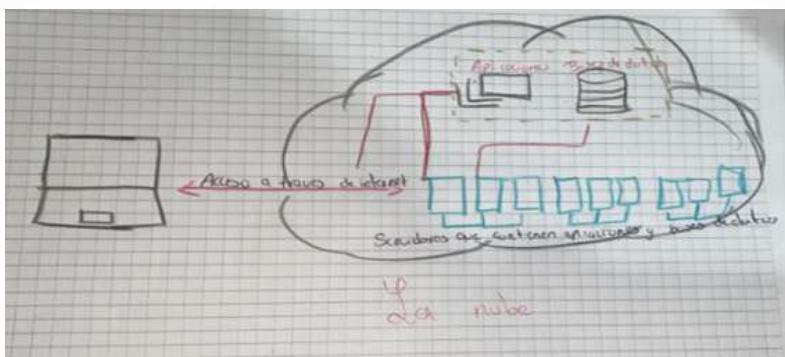
Hernandez Guzman, K. M. (2023). Adaptación de un patrón de software en seguridad a la arquitectura de un Microservicio.

# Arquitecturas de Referencia Edgar Computing para la Industria

Las arquitecturas de Edge Computing han surgido como soluciones clave para reducir la latencia, mejorar la privacidad y optimizar el uso del ancho de banda en aplicaciones del Internet de las Cosas (IoT). Estas tecnologías son fundamentales en áreas como ciudades inteligentes, agricultura, industria 4.0 y consumo eficiente de energía. Este estudio revisa las principales arquitecturas propuestas por organizaciones como el Edge Computing Consortium, el Proyecto FAR-Edge, el Industrial Internet Consortium y la colaboración entre INTEL y SAP. Además, se realiza una comparación detallada de sus características y enfoques.

## Reflexión

El Edge Computing representa un avance significativo en la tecnología, al ofrecer soluciones eficientes para abordar los desafíos del Internet de las Cosas (IoT). Su capacidad para reducir la latencia, mejorar la privacidad y optimizar el uso del ancho de banda tiene un impacto directo en sectores clave como la industria 4.0, las ciudades inteligentes y la agricultura. Reflexionar sobre las arquitecturas existentes revela la importancia de la colaboración entre grandes consorcios y proyectos, ya que su trabajo conjunto impulsa la innovación. Sin embargo, este panorama también invita a pensar en la necesidad de ampliar el enfoque hacia escenarios más diversos. El diseño de nuevas arquitecturas adaptables permitirá no solo aprovechar al máximo el Edge Computing.



## Bibliografía

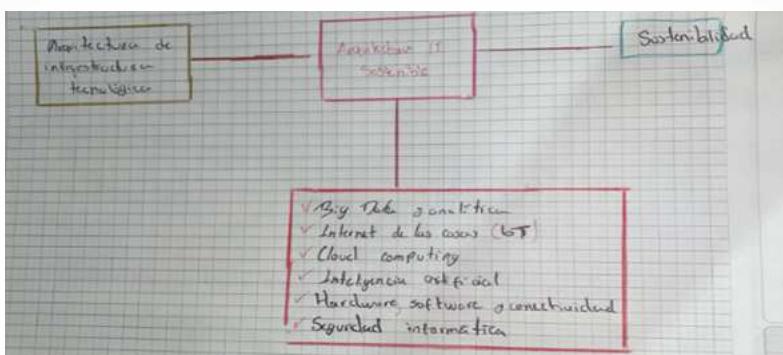
- Sittón-Candanedo, I., Alonso, R. S., Muñoz, L., & Rodríguez-González, S. (2019, August). Arquitecturas de Referencia Edge Computing para la Industria 4.0: una revisión. In Memorias de Congresos UTP (pp. 16-23).

# Un modelo sostenible para la gestión de decisiones de diseño en arquitectura software

Desde hace más de 30 años, las arquitecturas software son fundamentales en el diseño de sistemas. Tradicionalmente, se han entendido como un conjunto de componentes y conectores que resuelven problemas específicos. Sin embargo, desde 2004, se han considerado también como el resultado de decisiones de diseño, reconociendo estas decisiones como elementos clave que capturan el conocimiento de los expertos. Esta perspectiva ayuda a evitar la pérdida de información valiosa y facilita la evolución y el mantenimiento del sistema. Un desafío importante es definir la cantidad de conocimiento arquitectónico necesario para capturar. Además, se busca medir la sostenibilidad de las arquitecturas software.

## Reflexión

La evolución en la forma de entender las arquitecturas software refleja la creciente complejidad y necesidad de sostenibilidad en los sistemas actuales. Reconocer las decisiones de diseño como elementos clave para capturar conocimiento arquitectónico subraya la importancia de preservar el saber experto, especialmente en contextos donde la adaptabilidad y el mantenimiento son críticos. Esta visión no solo cambia el enfoque hacia el diseño consciente, sino que también plantea el desafío de crear métricas específicas para evaluar la sostenibilidad de estas decisiones. Reflexionar sobre la sostenibilidad arquitectónica nos lleva a valorar no solo la funcionalidad de un sistema, sino también su capacidad para perdurar y adaptarse en un entorno cambiante.



## Bibliografía

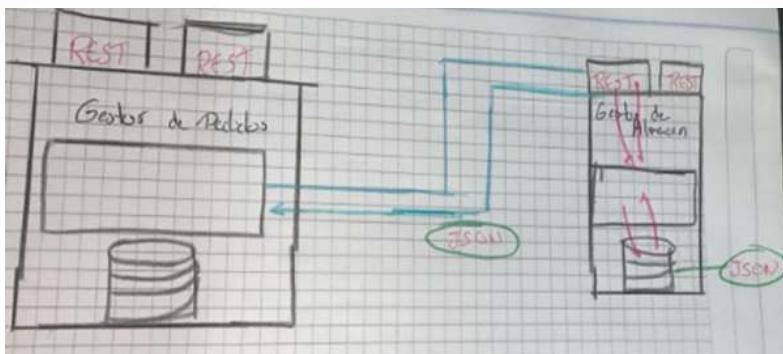
Carrillo Sánchez, C. (2017). Un modelo sostenible para la gestión de decisiones de diseño en arquitectura software (Doctoral dissertation, ETSIS\_Telecomunicacion).

# Salesforce API framework para patrones de diseño de micro servicios

Este trabajo explora el uso de Salesforce Sales y Service Cloud en arquitecturas basadas en micro servicios. Se describe la arquitectura de micro servicios y cómo Salesforce, a través de Apex REST, puede actuar como un cliente e implementador de micro servicios. Además, se analizan seis patrones de diseño de micro servicios (PDMS): Aggregator, API Gateway, Chained, Asynchronous, Database or Shared Data, Event Sourcing y Branch, proponiendo su aplicación en el entorno de Salesforce. El proyecto implementa un sistema productor-consumidor compuesto por múltiples REST APIs que se comunican como micro servicios, demostrando los PDMS en acción.

## Reflexión

La adopción de arquitecturas basadas en micro servicios representa un cambio significativo en la forma en que las empresas desarrollan y gestionan sus sistemas, buscando flexibilidad, escalabilidad y eficiencia. Salesforce, con herramientas como Apex REST, demuestra ser una plataforma robusta para implementar este enfoque, integrando tecnologías modernas en entornos empresariales. Reflexionar sobre este trabajo nos permite valorar cómo los patrones de diseño de micro servicios (PDMS) no solo organizan mejor las aplicaciones, sino que también fomentan la interoperabilidad entre servicios. La capacidad de Salesforce para facilitar pruebas y comunicación mediante REST APIs refuerza su papel en arquitecturas complejas.



## Bibliografía

Milbradt Rodríguez, D. (2023). Salesforce API framework para patrones de diseño de microservicios.

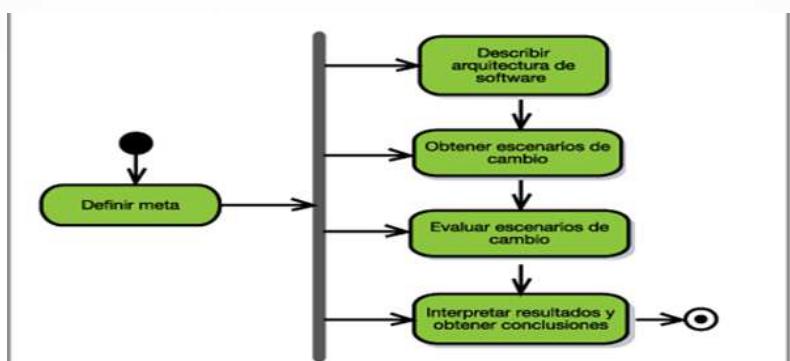
# Especificando una arquitectura de software

Esta investigación explora los componentes clave de una arquitectura de software y su interrelación mediante patrones arquitecturales. En un mundo donde las plataformas móviles y web crecen exponencialmente, la forma de diseñar y gestionar infraestructuras tecnológicas ha cambiado drásticamente. Se destacan los desafíos actuales del software, como la creciente complejidad de tareas, la necesidad de entregas rápidas, el manejo eficiente de datos y la seguridad en transacciones. Los patrones arquitecturales ayudan a estructurar y esquematizar estos aspectos, proporcionando una base sólida para soportar proyectos en constante expansión.

## Reflexión

Esta reflexión sobre la investigación en arquitectura de software resalta la importancia de los patrones arquitecturales como herramientas fundamentales para enfrentar los desafíos actuales en el desarrollo tecnológico. En un entorno donde la evolución de plataformas móviles y web ocurre a una velocidad vertiginosa, los desarrolladores se ven en la necesidad de adoptar estructuras sólidas y escalables que permitan la adaptabilidad y eficiencia.

La creciente complejidad de las tareas, junto con la exigencia de entregas rápidas y seguras, requiere un enfoque estratégico en el diseño de software. Aquí es donde los patrones arquitecturales actúan como guías, aportando claridad y cohesión al estructurar infraestructuras tecnológicas.



## Bibliografía

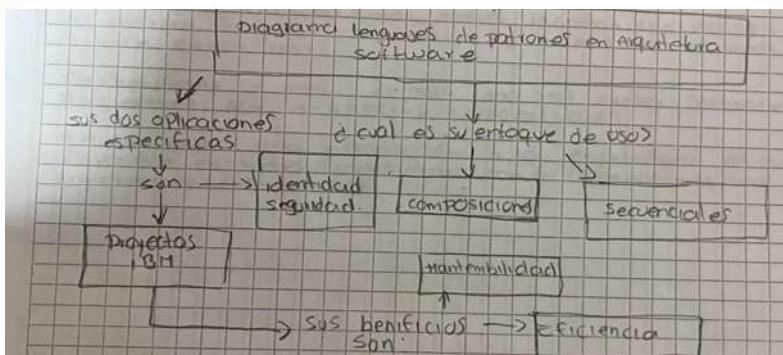
- Quilindo, L. A., & Vega, J. S. (2022). Especificando una arquitectura de software. TIA Tecnología, investigación y academia, 10(2), 136-149.

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado

El artículo analiza la evolución de los lenguajes de patrones en la arquitectura de software, comenzando con los conceptos de Christopher Alexander en la arquitectura física, los cuales fueron adaptados posteriormente al diseño de sistemas informáticos. Se destaca cómo estos patrones han facilitado la creación de diseños modulares y reutilizables, lo que mejora la adaptabilidad y la calidad de los sistemas. Los lenguajes de patrones ayudan a resolver problemas de diseño comunes y se utilizan ampliamente en diversas industrias, como en IBM para arquitecturas de e-business, así como en áreas específicas como la gestión de identidades y la seguridad de la información.

## Reflexión

La reflexión derivada del artículo "Lenguajes de Patrones de Arquitectura de Software: Una Aproximación al Estado del Arte" subraya la relevancia de los lenguajes de patrones en la ingeniería de software actual. Estos patrones no solo facilitan la resolución de problemas comunes en el diseño arquitectónico, sino que también fomentan un enfoque organizado y compartido entre los miembros del equipo de desarrollo, creando una comprensión común. Al estandarizar buenas prácticas y soluciones, los lenguajes de patrones contribuyen a la construcción de sistemas más sólidos y flexibles, brindando a los arquitectos de software herramientas probadas para abordar desafíos complejos.



## Bibliografía

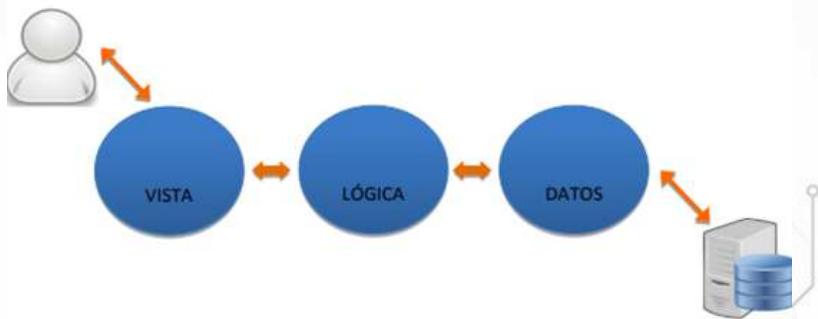
- Jimenez-Torres, V. H., Tello-Borja, W., & Rios-Patiño, J. I. (2014). Lenguajes de patrones de arquitectura de software: una aproximación al estado del arte. *Scientia et technica*, 19(4), 371-376.

# Arquitectura de software académica para la comprensión del desarrollo de software en capas.

El artículo expone una arquitectura de software en capas diseñada específicamente para el ámbito educativo, con el propósito de ayudar a los estudiantes a comprender los principios básicos de un desarrollo de software estructurado y bien organizado. Esta arquitectura se divide en capas claramente definidas: la capa de interfaz gráfica (GUI), que gestiona la interacción visual con el usuario; la capa de control, responsable de la comunicación entre la interfaz y la lógica del sistema; la capa de lógica de negocio, donde se encuentra la funcionalidad principal; y la capa de acceso a datos, que administra la conexión con las bases de datos y el manejo de la información.

## Reflexión

Este artículo destaca la relevancia de enseñar el desarrollo de software mediante una arquitectura en capas, un enfoque que facilita la comprensión de conceptos clave como la organización y la modularidad. Este modelo permite a los estudiantes aprender a dividir un sistema en componentes independientes, algo fundamental para garantizar el mantenimiento y la escalabilidad del software en escenarios reales. Asimismo, ofrece una base sólida para enfrentar la complejidad de sistemas de gran tamaño, donde cada capa tiene un rol específico y minimiza la interdependencia entre los componentes. Este método de enseñanza prepara a los futuros profesionales para adaptarse a cambios y optimizar el rendimiento de los sistemas en contextos diversos.



## Bibliografía

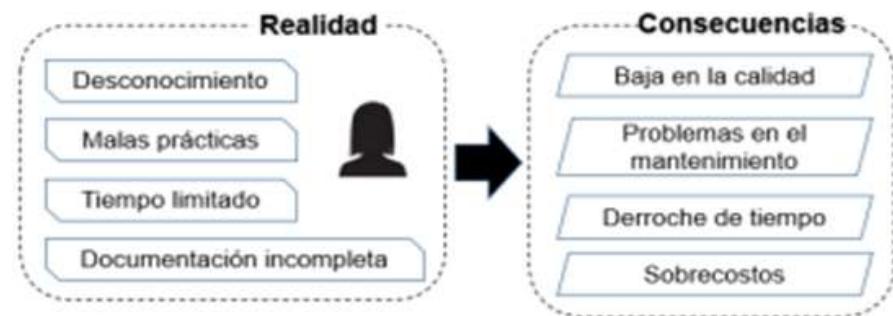
- Cardacci, D. G. (2015). Arquitectura de software académica para la comprensión del desarrollo de software en capas (No. 574). Serie Documentos de Trabajo.

# Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software.

El artículo propone un marco de trabajo orientado a facilitar la selección del patrón arquitectónico más adecuado para el desarrollo de software, tomando en cuenta aspectos esenciales como la calidad, la modularidad y la escalabilidad. Este enfoque identifica patrones como MVC, MVP, micro servicios y arquitecturas basadas en la nube, destacando las características específicas que los hacen apropiados para diferentes tipos de aplicaciones. El marco guía a arquitectos y desarrolladores en la elección del patrón más adecuado según factores como el tipo de aplicación (web, móvil o de escritorio) y requisitos de seguridad, rendimiento y mantenibilidad.

## Reflexión

El artículo resalta la relevancia de contar con un marco de trabajo definido para seleccionar patrones arquitectónicos, un aspecto clave en proyectos de software de alta calidad. Tener una guía que ayude a los desarrolladores a elegir el patrón más adecuado para sus proyectos reduce significativamente los riesgos de fallos estructurales y favorece la escalabilidad y sostenibilidad del producto final. Este enfoque demuestra que una arquitectura de software bien diseñada no solo satisface los requisitos actuales de la aplicación, sino que también se adapta a futuras necesidades de mantenimiento y actualización. La implementación de un marco como este brinda claridad en las decisiones arquitectónicas y promueve prácticas alineadas con los estándares de la industria.



## Bibliografía

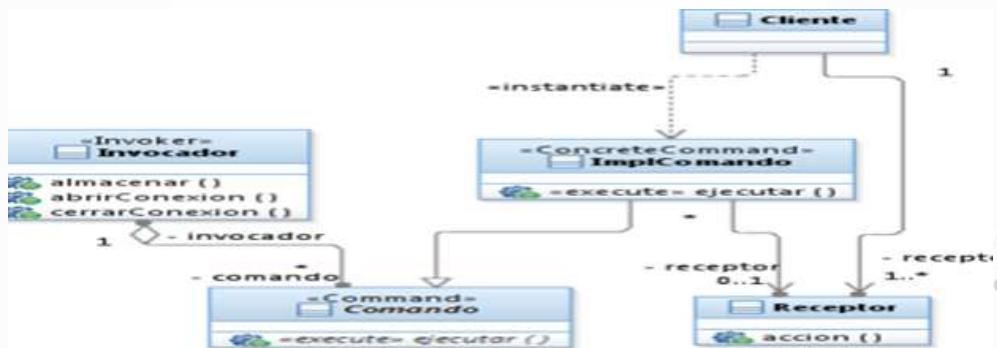
- Mejía, J. C. G., Agudelo, F. A. V., & Gil, K. G. (2021). Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software. Revista Ibérica de Sistemas e Tecnologias de Informação, (E43), 568-581.

# Perfiles UML para la Definición de Patrones de Diseño de Comportamiento

El artículo sugiere la utilización de perfiles UML junto con restricciones OCL para definir patrones de comportamiento en el contexto de la ingeniería de software. Estos patrones son esenciales para comprender las interacciones entre objetos y sus respectivas responsabilidades dentro de un sistema. Este método permite tanto especificar como validar patrones en modelos estáticos y dinámicos, basándose en diagramas de clases y de secuencia de UML. La definición de patrones de comportamiento se realiza a través de un perfil UML diseñado para cada patrón, el cual incluye restricciones expresadas en OCL. Este enfoque posibilita la validación tanto de la estructura como del comportamiento de los modelos.

## Reflexión

El artículo propone el empleo de perfiles UML y restricciones OCL como herramientas para definir patrones de comportamiento en la ingeniería de software. Estos patrones desempeñan un papel crucial en la comprensión de las interacciones entre objetos y sus responsabilidades dentro de un sistema. Este enfoque permite tanto la especificación como la validación de patrones en modelos estáticos y dinámicos, utilizando diagramas de clases y de secuencia de UML como soporte. La definición de los patrones se realiza mediante perfiles UML específicos para cada patrón, los cuales incorporan restricciones definidas en OCL.



## Bibliografía

Cortez, A., Riesco, D. E., & Garis, A. G. (2012). Perfiles UML para la definición de patrones de diseño de comportamiento. In XIV Workshop de Investigadores en Ciencias de la Computación.

# Integración de Arquitectura de Software en el Ciclo de Vida de las Metodologías Ágiles

El texto analiza la intersección entre las Metodologías Ágiles (MA) y la Arquitectura de Software (AS), dos enfoques tradicionalmente vistos como opuestos debido a sus diferencias en la gestión de requisitos. Mientras que las MA se centran en la flexibilidad y reducen al mínimo el trabajo inicial, la AS se fundamenta en decisiones tempranas que determinan el diseño del sistema. Para cerrar esta brecha, surge el concepto de Arquitectura Ágil (AA), que introduce los Requisitos Significativos para la Arquitectura (RSA) como elementos clave para lograr una arquitectura eficiente. Los RSA, que abarcan tanto requisitos funcionales como no funcionales, son complejos de identificar y validar debido a su naturaleza subjetiva, relativa e interactiva.

## Reflexión

El artículo propone el empleo de perfiles UML y restricciones OCL como herramientas para definir patrones de comportamiento en la ingeniería de software. Estos patrones desempeñan un papel crucial en la comprensión de las interacciones entre objetos y sus responsabilidades dentro de un sistema. Este enfoque permite tanto la especificación como la validación de patrones en modelos estáticos y dinámicos, utilizando diagramas de clases y de secuencia de UML como soporte. La definición de los patrones se realiza mediante perfiles UML específicos para cada patrón, los cuales incorporan restricciones definidas en OCL.



## Bibliografía

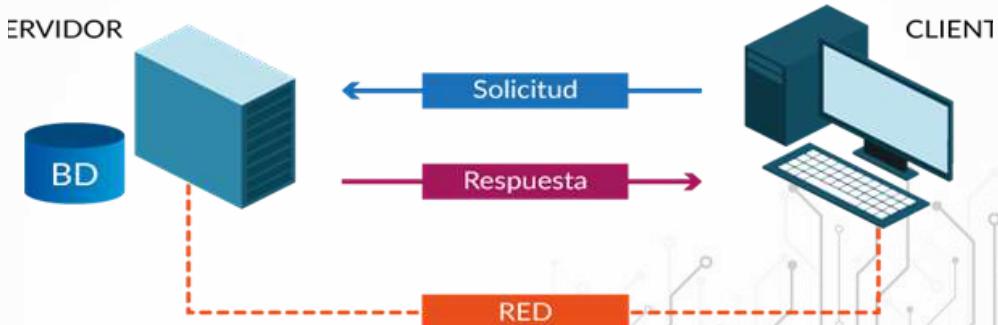
- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (2017, September). Integración de arquitectura de software en el ciclo de vida de las metodologías ágiles. In XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017, ITBA, Buenos Aires).

# Desarrollo de una Arquitectura de Software para Gestión de Información no Estructurada

El documento detalla el diseño e implementación de una arquitectura de software orientada a la gestión de grandes volúmenes de datos no estructurados a través de un sistema cliente-servidor. Este sistema permite realizar operaciones como insertar, buscar, actualizar y eliminar datos, incluyendo contenido multimedia y datos geolocalizados, sin depender de una estructura rígida. La propuesta profundiza en conceptos esenciales como HTTP, REST, API, Node.js, Express.js, y en bases de datos SQL y NoSQL, destacando a MongoDB, Memcached y Redis como opciones ideales por su alto rendimiento y escalabilidad para manejar datos no estructurados.

## Reflexión

El desarrollo de sistemas capaces de procesar grandes volúmenes de datos no estructurados se ha vuelto esencial en un mundo digital donde la información se genera y circula de manera masiva y acelerada. La transición de bases de datos tradicionales a modelos NoSQL responde a la necesidad de adoptar paradigmas que prioricen la flexibilidad y la velocidad, aspectos clave para el éxito de las aplicaciones modernas. La posibilidad de gestionar datos eficientemente sin depender de estructuras rígidas fomenta una mayor innovación en el ámbito del desarrollo de software. Este enfoque no solo beneficia a las grandes corporaciones que manejan enormes cantidades de datos



## Bibliografía

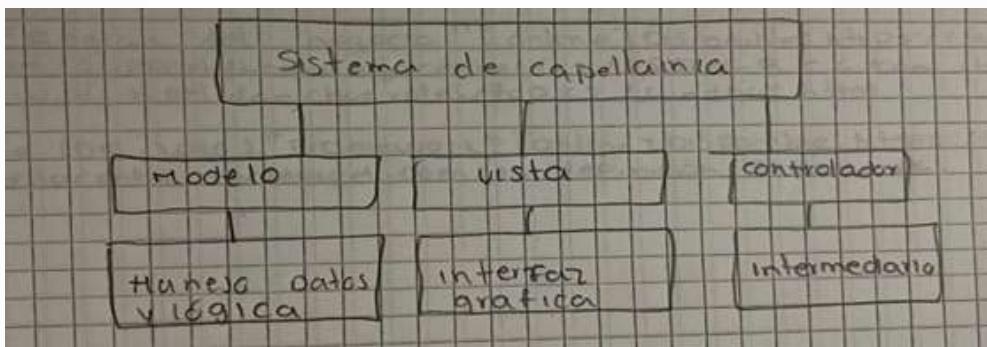
Florez Mendoza, A. M., & Daza Díaz, D. C. Desarrollo de una arquitectura de software para gestión de información no estructurada.

# Implementación del patrón arquitectónico MVC en aplicaciones web para la arquitectura del software del sistema de capellanía de la UM.

El artículo analiza la aplicación del patrón arquitectónico Modelo-Vista-Controlador (MVC) en el desarrollo de una aplicación web para el sistema de capellanía de la Universidad de Montemorelos, con el objetivo de optimizar la gestión y accesibilidad de la información estudiantil. Esta herramienta permite a los estudiantes programar citas con los capellanes, quienes pueden organizar y consultar fácilmente los datos recopilados durante entrevistas y encuestas, promoviendo un enfoque más sostenible al reducir el uso de papel.

## Reflexión

Este artículo resalta la relevancia de adaptar la tecnología a las necesidades específicas de instituciones educativas y religiosas, como los sistemas de capellanía. La implementación del patrón arquitectónico MVC no solo aporta una estructura organizada y de fácil mantenimiento, sino que también permite a los capellanes concentrarse en brindar apoyo espiritual y académico a los estudiantes, evitando depender de herramientas obsoletas. Este enfoque demuestra cómo la tecnología puede mejorar la orientación personal al facilitar un seguimiento detallado y eficiente de cada alumno, haciéndola más accesible y humanizada.



## Bibliografía

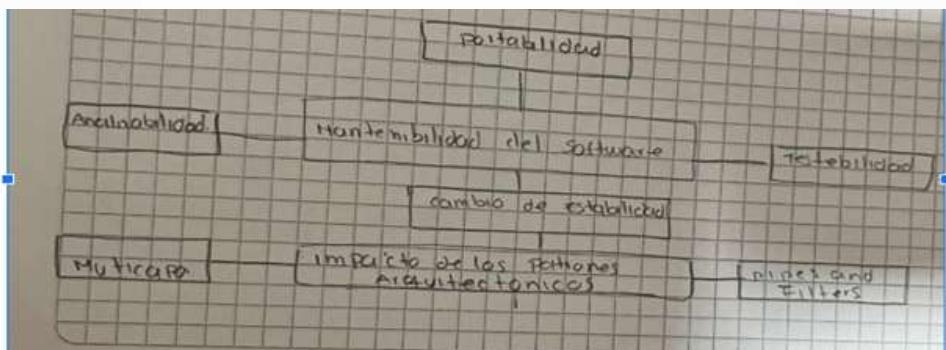
- Sánchez, P. Y. C. (2020). Implementación del patrón arquitectónico MVC en aplicaciones web para la arquitectura del software del sistema de capellanía de la UM. Anuario de Investigación UM, 1(1), 112-120.

# **Garantizar la mantenibilidad del software a nivel de arquitectura de software utilizando patrones arquitectónicos.**

El artículo examina la relevancia de la arquitectura de software para mejorar la mantenibilidad del mismo, destacando cómo los patrones arquitectónicos influyen en aspectos clave como la capacidad de análisis, adaptación, estabilidad, facilidad de pruebas, comprensión y portabilidad. Se propone un modelo de calidad para evaluar la mantenibilidad arquitectónica, dividiendo estos atributos en subatributos medibles mediante métricas concretas. Además, se exploran patrones arquitectónicos comunes como el patrón multicapa y el patrón de pipes and filters, analizando su impacto en la mantenibilidad.

## **Reflexión**

Este artículo resalta la importancia crucial de la arquitectura de software en la sostenibilidad y evolución de los sistemas informáticos. Elegir los patrones arquitectónicos adecuados no solo facilita el mantenimiento, sino que también mejora la eficiencia y la adaptabilidad. Sin embargo, este proceso exige un análisis cuidadoso para evitar los efectos negativos que puedan surgir de decisiones arquitectónicas incorrectas. Además, se subraya la necesidad de adoptar un enfoque proactivo en la planificación arquitectónica incorporando herramientas que permitan medir y optimizar los atributos de calidad desde las primeras fases del diseño.



## **Bibliografía**

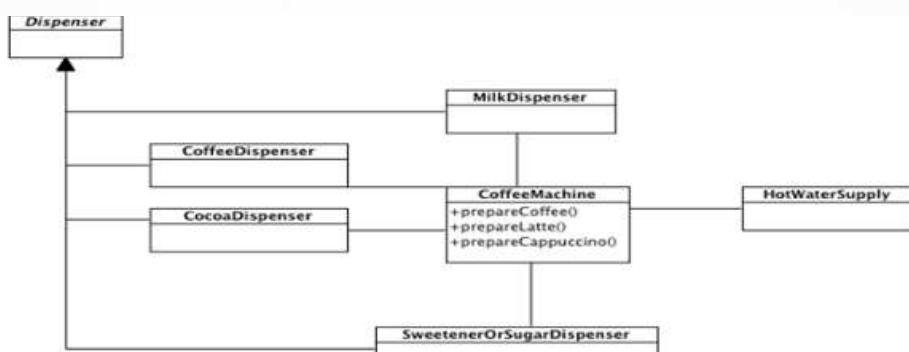
Rahmati Z, Tanhaei M. Ensuring software maintainability at software architecture level using architectural patterns. 2021;2(1):81-102.

# Patrones de Diseño, Refactorización y Anti patrones. Ventajas y Desventajas de su Utilización en el Software Orientado a Objetos

El autor destaca que, a pesar de los avances en la ingeniería de software y la adopción de tecnologías como la programación orientada a objetos, muchos sistemas siguen siendo rígidos y difíciles de adaptar a cambios, lo que afecta su flexibilidad y calidad. Se analiza cómo la experiencia en desarrollo de software se puede dividir en dos categorías principales: Patrones de Diseño y Anti patrones. Los patrones de diseño son soluciones probadas para problemas recurrentes en el diseño de software, mientras que los anti patrones describen soluciones incorrectas que generan efectos negativos, y la refactorización es el proceso mediante el cual un anti patrón se convierte en una solución adecuada.

## Reflexión

El uso de patrones de diseño y anti patrones en el desarrollo de software ha representado un gran avance para la industria, ya que ofrece a los desarrolladores un conjunto de soluciones comprobadas para problemas comunes. Este enfoque no solo optimiza la eficiencia del proceso de diseño, sino que también promueve la colaboración y el aprendizaje compartido, permitiendo a los desarrolladores aprovechar la experiencia acumulada por otros a lo largo del tiempo. NO obstante, la implementación de estos patrones debe realizarse con cuidado. Aunque los patrones de diseño pueden ser beneficiosos, aplicarlos de manera incorrecta o en el contexto inapropiado puede aumentar la complejidad.



## Bibliografía

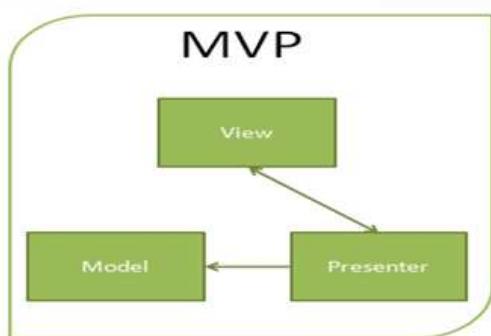
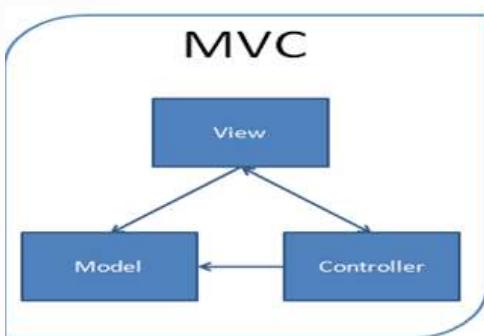
Campo, G. D. (2009). Patrones de Diseño, Refactorización y Antipatrones. Ventajas y Desventajas de su Utilización en el Software Orientado a Objetos. Cuadernos de Ingeniería, (4), 101-136.

# Patrones de diseño

El texto presenta el concepto de patrones de diseño en la programación orientada a objetos (POO), destacando su papel en la resolución de problemas frecuentes en el diseño de sistemas. A pesar de ser una metodología poderosa, la programación orientada a objetos enfrenta desafíos como la identificación y estructuración adecuada de clases, así como su reutilización eficiente. Los patrones de diseño emergen como una herramienta fundamental para superar estas dificultades. Un patrón de diseño es una solución generalizada para un problema recurrente dentro de un contexto determinado.

## Reflexión

El empleo de patrones de diseño en la programación orientada a objetos facilita la creación de soluciones complejas de manera organizada y eficiente. La programación orientada a objetos, aunque potente, puede presentar dificultades al tratar de identificar las clases adecuadas, organizarlas correctamente y asegurar la reutilización del código. En este contexto, los patrones de diseño sirven como guías que ayudan a los desarrolladores a construir sistemas más sólidos, modulares y fáciles de mantener. La principal ventaja de los patrones de diseño es que ofrecen soluciones probadas y reutilizables para problemas comunes, lo que no solo incrementa la eficiencia en el desarrollo, sino que también mejora la comunicación entre los equipos de programación.



## 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.

# Aplicación de Patrones de Diseño para Garantizar Alta Flexibilidad en el Software

Las empresas a menudo deben modificar sus reglas de negocio para seguir siendo competitivas y satisfacer mejor las necesidades de sus clientes. Por esta razón, las aplicaciones de software empresarial deben ser lo suficientemente adaptables para manejar estos cambios de manera eficiente y ágil, sin requerir grandes inversiones de recursos y tiempo. Para alcanzar esta flexibilidad, el software debe ser diseñado con la previsión de que se realizarán modificaciones, especialmente en las reglas de negocio. En este contexto, los patrones de diseño desempeñan un papel clave al abordar los desafíos de adaptabilidad.

## Reflexión

La capacidad de las empresas para adaptarse rápidamente a los cambios en las reglas de negocio es esencial para mantener su ventaja competitiva en el mercado actual. Por lo tanto, diseñar software flexible y adaptable es fundamental, ya que permite a las organizaciones implementar soluciones tecnológicas que evolucionen según las necesidades del negocio. Los principios de diseño de software deben reflejar esta necesidad de cambio continuo. En lugar de crear aplicaciones rígidas y monolíticas, los desarrolladores deben adoptar enfoques que fomenten la modularidad, la extensibilidad y la adaptabilidad del código.



## Bibliografía

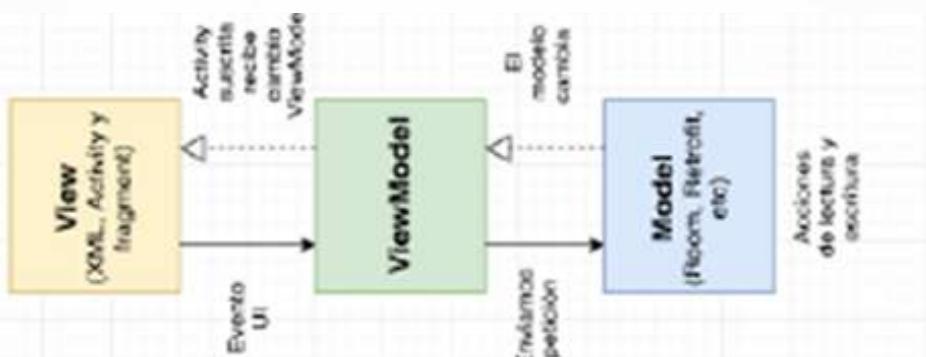
Escalante, L. C. (2014). Aplicación de patrones de diseño para garantizar alta flexibilidad en el software. *Tecnología y Desarrollo* (Trujillo), 12(1), 77-82.

# Análisis Comparativo De Patrones De Diseño De Software

Los patrones de diseño ofrecen soluciones eficaces a los problemas comunes en el desarrollo de software, ayudando a evitar la duplicación de código y facilitando su reutilización. Este artículo presenta la estructura, componentes, ventajas y desventajas de diversos patrones de diseño, como Template Method, Model-View-Controller (MVC), Model-View-Presenter (MVP), Model Front Controller y Model-View-ViewModel (MVVM). La investigación se llevó a cabo mediante una revisión bibliográfica en bases de datos científicas, utilizando métricas que permitieron comparar los patrones estudiados. A través de un análisis comparativo de métricas y parámetros, se concluyó que no existe un patrón que sea superior en términos generales.

## Reflexión

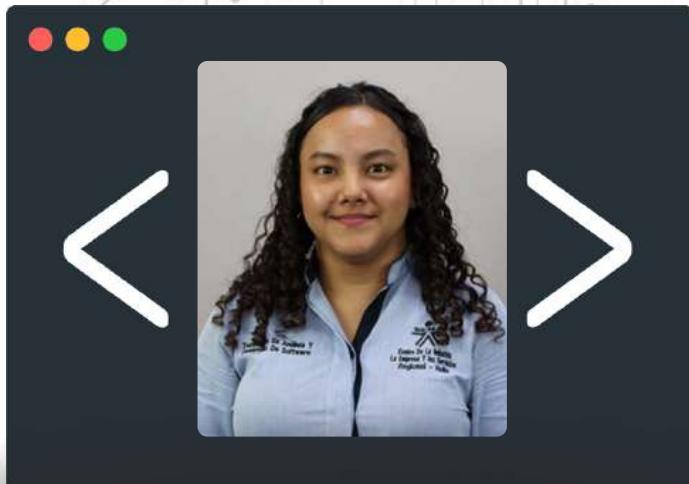
Esta investigación resalta la importancia del rol del desarrollador al identificar qué patrón de diseño se adapta mejor a las necesidades específicas del proyecto. Esto implica que tener un buen conocimiento y comprensión de los patrones de diseño es esencial para tomar decisiones acertadas y garantizar la calidad del software. Además de comparar los patrones, el artículo subraya un punto clave: los patrones de diseño ayudan a mantener una organización coherente en el código, promoviendo la modularidad, el mantenimiento y la comprensión del sistema. Como resultado, esto ofrece beneficios prácticos, como una mayor facilidad para adaptar el software a cambios y una mejora en la calidad general del producto final.



## 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.

# Carolina Martínez Cortes



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.

# Generadores de código y patrones arquitectónicos

Este documento es una revisión sistemática sobre Generadores de Código Fuente (GCF) y patrones de arquitectura en el desarrollo de software, enfocado en aplicaciones web. Analiza cómo los GCF automatizan tareas repetitivas, como la creación de interfaces y conexiones a bases de datos, reduciendo tiempos, costos y errores, permitiendo a los desarrolladores centrarse en aspectos más complejos. Además, destaca la importancia de patrones arquitectónicos como MVC o MVVM, que estructuran el software para mejorar su mantenimiento, escalabilidad y calidad. A través de un análisis exhaustivo, identifica herramientas, frameworks y lenguajes más usados, siguiendo una metodología de revisión sistemática.



## Reflexión

En un mundo de constante evolución tecnológica, las aplicaciones rápidas y sofisticadas son esenciales. Los Generadores de Código Fuente (GCF) y los patrones de arquitectura se han convertido en herramientas clave para acelerar el desarrollo y mejorar la calidad del software. Los GCF automatizan tareas repetitivas, mientras que los patrones arquitectónicos, como MVC o MVVM, garantizan organización, escalabilidad y mantenimiento. Este estudio resalta la importancia de elegir herramientas y patrones adecuados para cumplir expectativas actuales y facilitar futuras actualizaciones, destacando cómo decisiones informadas pueden transformar la eficiencia y durabilidad de los proyectos.

## Bibliografía

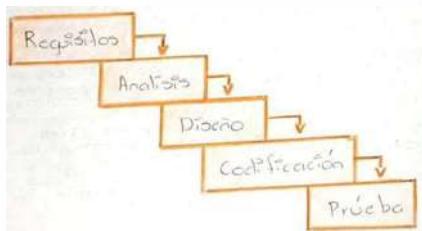
- Casas, M. R. H. (2020). Revisión sistemática sobre generadores de código fuente y patrones de arquitectura (Master's thesis, Pontificia Universidad Católica del Perú (Peru)). <https://n9.cl/jedyi>

# Una teoría para el diseño de software

Este documento presenta una teoría sobre el diseño de software, destacando su papel en todo el ciclo de desarrollo. Propone dividir el software en componentes interconectados con funciones claras, lo que facilita el mantenimiento y reduce costos, que pueden alcanzar el 67% del proyecto. Introduce el concepto de "Diseño para el Cambio," que anticipa modificaciones futuras, y el "diseño calculado," que combina intuición y reglas estructuradas para crear componentes sólidos. El diseño se organiza en tres niveles: estilo arquitectónico (como Cliente/Servidor), patrones de diseño y definición detallada de componentes. Su evolución muestra su importancia para la calidad y adaptación del sistema.

## Reflexión

Esta teoría resalta el diseño como un pilar esencial en el desarrollo de software, promoviendo estructuras flexibles que faciliten la adaptación a nuevas demandas sin reestructuraciones costosas. Considera el diseño una inversión a largo plazo, ya que reduce los costos de cambios y mantenimiento. El principio de "Diseño para el Cambio" cobra relevancia en un entorno tecnológico en constante evolución, mientras que un enfoque calculado garantiza decisiones estructurales consistentes. Al combinar estilos arquitectónicos y patrones de diseño, fomenta la colaboración y minimiza problemas de integración, asegurando sistemas sostenibles y preparados para el futuro.



## Bibliografía

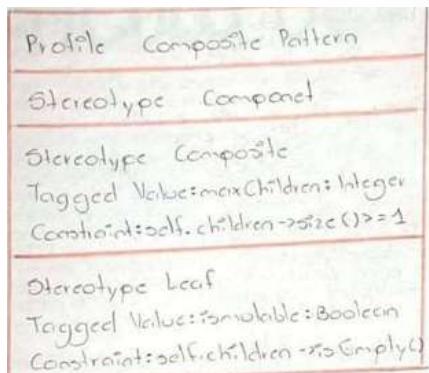
Cristiá, M. (2021). Una Teoría para el Diseño de Software.  
<https://www.fceia.unr.edu.ar/ingsoft/intro-diseno.pdf>

# Perfiles UML para definición de Patrones de Diseño

trones de diseño de forma más precisa. Aunque UML estándar es útil, a menudo carece de expresividad para modelar patrones como el "Composite". Los perfiles UML resuelven estas limitaciones mediante estereotipos, etiquetas y restricciones en OCL, personalizando UML para dominios específicos. Este enfoque permite modelar patrones comunes, facilita la reutilización y elimina la necesidad de herramientas específicas. Los autores destacan su efectividad y sugieren explorar herramientas UML que optimicen esta técnica.

## Reflexión

definir patrones de diseño mejora la expresividad y flexibilidad de UML, adaptándolo a las necesidades específicas de cada proyecto. Este enfoque permite modelar patrones complejos sin depender de herramientas externas, haciendo el proceso más fluido y coherente con el diseño final. Además, crea un vocabulario visual común que facilita la comunicación y colaboración en equipos de desarrollo. Al personalizar UML para ajustarse a los requerimientos de patrones específicos, se ahorra tiempo, se mejora la adaptabilidad y se impulsa un trabajo más dinámico y eficiente.



## Bibliografía

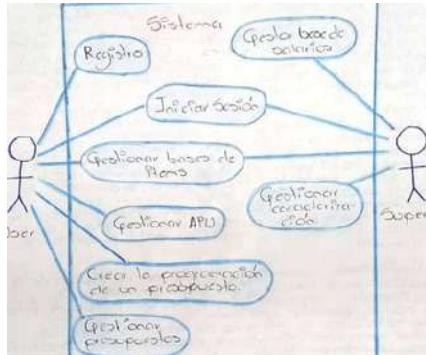
- Garis, A. G., Riesco, D. E., & Montejano, G. A. (2006). Perfiles UML para definición de Patrones de Diseño. In VIII Workshop de Investigadores en Ciencias de la Computación.  
[https://sedici.unlp.edu.ar/bitstream/handle/10915/20784/Documento\\_completo.pdf?sequence=1&isAllowed=y](https://sedici.unlp.edu.ar/bitstream/handle/10915/20784/Documento_completo.pdf?sequence=1&isAllowed=y)

# Arquitectura para Herramienta de Costos y Programación

El artículo describe el desarrollo de una arquitectura de software educativo para la materia de "Costos, Presupuestos y Programación de Obra" en Ingeniería Civil, en la Universidad Francisco de Paula Santander. Este software apoya el aprendizaje autónomo de los estudiantes, gestionando temas como el análisis de precios unitarios (APU), planificación de actividades y optimización de recursos. Utilizando diagramas UML, como el de secuencia, simula el flujo desde el registro hasta la creación de presupuestos, ayudando a los estudiantes a comprender el impacto de decisiones financieras y logísticas en proyectos de construcción.

## 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://n9.cl/c7q9l>



## Reflexión

Esta herramienta transforma la educación en Ingeniería Civil, combinando teoría y práctica de manera interactiva. Permite gestionar costos directos (materiales, mano de obra) e indirectos (gastos generales), simulando decisiones profesionales. Este enfoque de aprendizaje autodirigido no solo facilita habilidades técnicas, sino que también fomenta competencias críticas como la toma de decisiones, el pensamiento lógico y la autonomía. Su arquitectura responde a la necesidad de que los futuros ingenieros se adapten a un entorno tecnológico en constante cambio, donde el dominio de herramientas digitales es esencial.

# Arquitectura de Software guiada por el Dominio

La arquitectura hexagonal, o de puertos y adaptadores, aísla la lógica de negocio de las interfaces externas (como bases de datos, UI y APIs). El núcleo del negocio interactúa con el exterior mediante "puertos" (interfaces) y "adaptadores" (implementaciones específicas). Esto asegura que el software mantenga su funcionalidad principal intacta, incluso si se modifican herramientas, bases de datos o interfaces. Es ideal para sistemas que necesitan adaptabilidad y resistencia al cambio, favoreciendo un desarrollo modular y sostenible.

Arquitectura Tradicional	Arquitectura guiada por el dominio
<ul style="list-style-type: none"><li>Componentes monolíticos</li><li>Alto acoplamiento</li><li>Exhibibilidad pobre</li><li>Integración difícil</li><li>Flexibilidad baja</li></ul>	<ul style="list-style-type: none"><li>Servicios autónomos</li><li>Comunicación independiente</li><li>Bajo acoplamiento</li><li>Exhibibilidad alta</li><li>Flexibilidad alta</li></ul>

## 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?sequence=1&isAllowed=y](https://sedici.unlp.edu.ar/bitstream/handle/10915/115198/Documento_completo.pdf?sequence=1&isAllowed=y)

## 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.

# Verificación de Patrones Arquitectónicos en la Nube

Este artículo presenta un entorno de diseño especializado para crear arquitecturas de software en aplicaciones web en la nube. Basado en un metamodelo de componentes arquitectónicos, el entorno utiliza elementos comunes y patrones de sistemas cloud. Además de facilitar el modelado de arquitecturas complejas, incluye una herramienta de verificación gráfica que asegura la correcta aplicación de los patrones de diseño. La estructura modular del proceso permite generar diseños alineados con los requisitos del sistema, especialmente los no funcionales, como escalabilidad y seguridad.



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

## 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.

# Patrones de Usabilidad en la Arquitectura del Software

El artículo presenta una metodología innovadora para mejorar la usabilidad en el desarrollo de software, incorporando patrones de usabilidad desde la fase inicial de diseño. En el proyecto europeo STATUS, se propone integrar estos patrones en la arquitectura del sistema, optimizando la interacción del usuario desde el inicio. A diferencia de enfoques tradicionales, STATUS aborda la estructura interna del software, incluyendo funciones como "deshacer" y soporte para "múltiples idiomas". Este enfoque inductivo asegura que la usabilidad sea parte del diseño, reduciendo cambios posteriores y mejorando la experiencia del usuario.

## Reflexión

El enfoque de STATUS para mejorar la usabilidad desde la arquitectura del software es un avance respecto a métodos convencionales. Anticipa problemas de usabilidad que antes se identificaban al final del desarrollo, haciendo el proceso más eficiente y menos costoso. Integra la usabilidad como un componente estructural, no solo en la interfaz gráfica, creando un sistema intuitivo desde el núcleo. Al usar patrones arquitectónicos, STATUS vincula la estructura del sistema con la experiencia del usuario, facilitando diseños robustos que mejoran la eficiencia, satisfacción y reducen la necesidad de ajustes.

## Bibliografía

- 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/](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

# Arquitectura del software, esquemas y servicios

La arquitectura orientada a servicios (SOA) es un enfoque de diseño para crear aplicaciones distribuidas, donde servicios independientes se comunican a través de interfaces definidas. A diferencia de los componentes monolíticos, los servicios en SOA son reutilizables, autónomos y fáciles de integrar, cada uno con su propia lógica y base de datos, lo que asegura escalabilidad y fiabilidad. Utilizando protocolos abiertos, SOA facilita la interoperabilidad entre plataformas y lenguajes. Sin embargo, presenta desafíos en la gestión de la comunicación, el tiempo de respuesta y el tamaño de los mensajes.

Arquitecturas tradicionales	Arquitecturas orientadas a servicios (SOA)
<ul style="list-style-type: none"><li>Componentes monolíticos</li><li>No acoplamiento</li><li>Scalabilidad limitada</li><li>Integración difícil</li><li>Flexibilidad baja</li></ul>	<ul style="list-style-type: none"><li>Servicios autónomos</li><li>Comunicación independiente</li><li>Bajo acoplamiento</li><li>Exclusividad alta</li><li>Flexibilidad alta</li></ul>

## Reflexión

La evolución hacia arquitecturas orientadas a servicios (SOA) marca un cambio significativo en el diseño y desarrollo de aplicaciones. Mientras que las arquitecturas monolíticas eran más fáciles de desarrollar inicialmente, se vuelven difíciles de mantener y adaptar con el crecimiento de las aplicaciones. SOA promueve flexibilidad al permitir que los servicios sean independientes y se comuniquen mediante interfaces abiertas. Aunque la integración de múltiples servicios puede ser compleja, SOA facilita la adaptabilidad, la rapidez de respuesta y es ideal para el desarrollo de aplicaciones modernas, a pesar de los retos de comunicación y latencias.

## Bibliografía

- Romero, P. Á. (2006). Arquitectura de software, esquemas y servicios. Ingeniería Industrial, 27(1), 1. <https://dialnet.unirioja.es/servlet/articulo?codigo=4786655>

# Lenguajes de Patrones de Arquitectura: Estado del Arte

Este artículo aborda el concepto de "Lenguajes de Patrones" en la Arquitectura de Software, resaltando su impacto en la calidad y mantenibilidad del software. Desde sus inicios en la ingeniería de software, los patrones de diseño han facilitado la creación de sistemas más robustos y escalables. Estos lenguajes, formados por patrones interconectados, permiten abordar problemas comunes de manera estructurada. Ejemplos de su aplicación incluyen la resolución de problemas de autenticación en sistemas de información y la estructuración de la arquitectura en aplicaciones de e-business.



## Reflexión

Los lenguajes de patrones de arquitectura de software son un avance significativo en la ingeniería de software, simplificando el diseño y mejorando la calidad de los sistemas. Su capacidad para aplicarse a diversos dominios, como seguridad o e-business, demuestra su flexibilidad. Sin embargo, depender demasiado de patrones puede limitar la innovación si se usa de manera rígida. Es crucial que los desarrolladores mantengan un equilibrio entre la reutilización de patrones existentes y la creatividad en el diseño de soluciones únicas para abordar problemas específicos.

## Bibliografía

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>

# Introducción a la Arquitectura de Software

Los estilos arquitectónicos son fundamentales para definir la estructura de un sistema, ya que describen sistemas completos, no solo partes. Se caracterizan por los tipos de componentes, patrones de interacción y estructuras permitidas, influyendo en cómo se construye e implementa el sistema. Según Shaw y Garlan, la arquitectura guía la construcción de software, y su éxito puede establecer un estándar en una organización o industria. Además, los requisitos no funcionales, como desempeño y seguridad, deben considerarse desde el inicio para evitar problemas costosos en etapas finales.

## Bibliografía

Research-Gate.[Online]. Recuperado de:  
<https://www.researchgate.net/publication/251932352> Introducción a la Arquitectura de Software.  
[https://www.fceia.unr.edu.ar/~mcristia/Introduccion\\_a\\_la\\_Arquitectura\\_de\\_Software.pdf](https://www.fceia.unr.edu.ar/~mcristia/Introduccion_a_la_Arquitectura_de_Software.pdf)

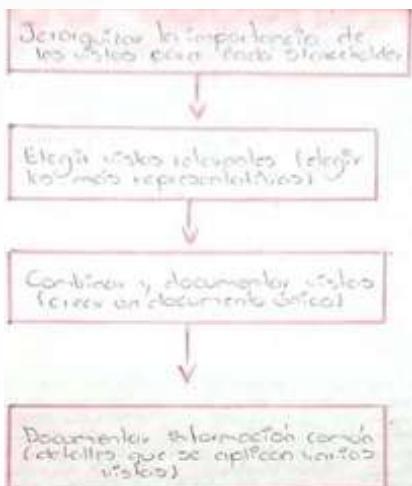
Arquitectura	Diseño
• Nivel de abstracción alto	• Nivel de detalle
• Define componentes y relaciones principales de los componentes definidos en la arquitectura	• Relaciona la implementación y las relaciones principales de los componentes definidos en la arquitectura
• Enfoque en conectividad. Se enfoca en la disposición interna de los componentes	• Se enfoca en la disposición interna de los componentes
• Considera todos los requerimientos funcionales y no funcionales	• No se ocupan tanto de los requerimientos no funcionales como de los funcionales

## Reflexión

El ciclo de la arquitectura debe ser un proceso continuo que considere no solo los requerimientos funcionales, sino también los no funcionales desde el inicio. Aspectos como seguridad, rendimiento y escalabilidad a menudo son ignorados en fases tempranas, lo que puede generar complicaciones costosas más adelante. Es clave tener una visión integral desde el principio, ya que estos aspectos impactan todo el sistema, no solo componentes aislados. La arquitectura debe ser adaptable y capaz de enfrentar estos desafíos a medida que el sistema crece y evoluciona.

# Atributos de calidad y arquitectura de software

La arquitectura de software es clave para lograr sistemas que no solo sean funcionales, sino que también cumplan con atributos esenciales como seguridad, disponibilidad, reusabilidad e interoperabilidad. Estos atributos suelen ser difíciles de equilibrar, ya que mejorar uno puede afectar negativamente a otro. Para enfrentarlo, la arquitectura debe ser definida desde diversas perspectivas, analizando características estáticas, dinámicas y la interacción con el entorno. Los estilos arquitectónicos influyen en la calidad del sistema, priorizando ciertas cualidades a expensas de 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/rearview/mirror/conferencias/jiisic04/Tutoriales/tu4.pdf>

# Arquitectura para Comunidades Académicas Interactivas

El artículo presenta una arquitectura de software para comunidades académicas virtuales en televisión digital interactiva (TDi), integrando servicios de e-learning con tecnologías Web 2.0. Utilizando un esquema REST-JSON, la arquitectura mejora la flexibilidad e interoperabilidad. Los componentes clave incluyen un directorio de servicios, un mediador de canales y un set-top box para facilitar la interacción del usuario. Esta propuesta se valida mediante el curso AgroEDiTV, promoviendo la televisión interactiva como herramienta educativa en áreas con acceso limitado a Internet.



## Reflexión

La implementación de comunidades académicas virtuales en televisión digital interactiva (TDi) tiene el potencial de transformar el acceso a la educación. La arquitectura optimiza la interactividad y permite una educación más flexible y accesible mediante servicios como foros y chats en tiempo real. Al emplear tecnologías como REST-JSON, se facilita la expansión hacia otras aplicaciones, como t-comercio y t-salud. Esta flexibilidad es clave para abordar los desafíos educativos actuales y aprovechar las ventajas de la televisión interactiva en un entorno digital.

## Bibliografía

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

# Integración de Arquitectura en Metodologías Ágiles

La investigación explora la integración de los Requisitos No Funcionales (RSA) en las Metodologías Ágiles (MA) y su impacto en la Arquitectura de Software (AS). Se destaca que los RSA, aunque difíciles de identificar, son fundamentales para una arquitectura adecuada en proyectos ágiles. La identificación se basa en evaluar necesidades específicas y la interacción de los stakeholders, utilizando métodos como entrevistas y análisis de objetivos comerciales. El estudio valida la viabilidad de integrar los RSA en las MA, analizando los beneficios con ejemplos empíricos y contribuyendo con nuevas estrategias para el desarrollo ágil.

Aplicación de los RSA	Descripción
Tipo de aplicaciones	orientadas a funcionales y no funcionales. Los RSA son los más funcionales, pero pueden incluir funcionalidades claves.
Impacto en la arquitectura	Los RSA influyen profundamente la estructura del sistema y sus componentes puede conflictos nublados alinear las demandas.
Identificación de los RSA	Con subtletades, retrasos e interacciones entre ellos, lo cual resulta en un desarrollo más eficiente y flexible.
Resolución en MA	Con Atividas MA suelen poder ser más eficientes y efectivas, lo cual puede resultar en una mejor calidad.
Identificación conceptual	Mejorar la calidad de las necesidades y satisfacer las expectativas del cliente.
Adaptabilidad y flexibilidad	Es crucial tener expectativas claras y cumplir con los RSA identificados y las expectativas de los stakeholders.

## Reflexión

En los proyectos de software, la colaboración entre metodologías ágiles y arquitectura de software ofrece un desarrollo equilibrado. La identificación temprana de Requisitos No Funcionales (RSA) asegura una arquitectura sólida y proporciona una guía estratégica frente a los cambios de las metodologías ágiles. Esto resalta la importancia de interpretar correctamente los RSA para el éxito del sistema. En un entorno donde la adaptabilidad y la estructura sólida parecen contradecirse, la "Arquitectura Ágil" emerge como una solución viable para mitigar estos conflictos y facilitar un desarrollo alineado con las expectativas del cliente.

## Bibliografía

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

# Patrones GOF en el Desarrollo de Aplicaciones Web

En el desarrollo de aplicaciones web, los patrones de diseño GOF son esenciales para mejorar la calidad del software, pero su implementación aún es limitada debido a la falta de conocimiento y experiencia. Los patrones creacionales como Singleton y Factory Method son los más utilizados, seguidos por los patrones de comportamiento como Iterator y Template Method. Sin embargo, su adopción es parcial y depende del marco de trabajo utilizado. La investigación confirma que los patrones son clave para el desarrollo de software de calidad, pero subraya la necesidad de mayor capacitación y conocimiento práctico para su correcta aplicación.

## Bibliografía

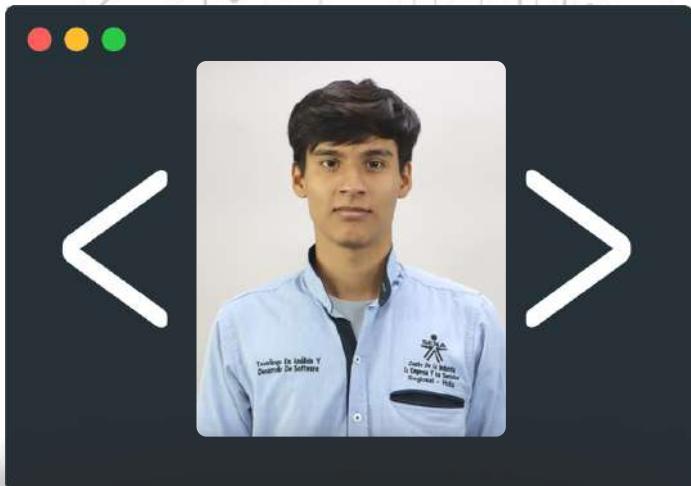
Guerrero, C. A., Suárez, J. M., & Gutiérrez, L. E. (2013).<https://www.scielo.cl/pdf/infotec/v24n3/art12.pdf>

Categoría	Descripción	Ejemplos	Aplicación en Proyectos
Patrones Creacionales	Controlan la creación de objetos.	Singleton, Factory Method	67% de los proyectos
Patrones Estructurales	Organizan las relaciones entre clases y objetos.	Decorator, Facade	50% de los proyectos
Patrones de Comportamiento	Gestión de la iteración y distribución de responsabilidades.	Iterator, Template Method	67% de los proyectos
Resumen	Aplicación efectiva de estos patrones GOF en proyectos web.		

## 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.

# Cristian Fernando Narváez Sánchez



Mi nombre es Cristian Fernando Narváez Sánchez, nací el 11 de enero de 2006 y tengo 18 años. Actualmente, soy aprendiz en el programa de Análisis y Desarrollo de Software en el SENA, una etapa clave en mi crecimiento profesional. Desde joven, me interesé por la tecnología y su potencial para transformar el mundo, lo que me motivó a enfocarme en el desarrollo de software.

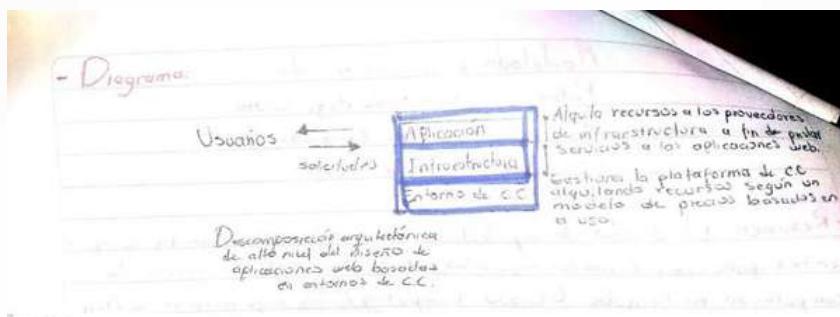
A lo largo de mi formación, he aprendido diversas herramientas y lenguajes de programación, los cuales utilizo para crear soluciones innovadoras. Mi pasión por aprender me impulsa a enfrentar desafíos con dedicación y a mejorar constantemente en cada proyecto. Mi objetivo es contribuir al avance tecnológico a través de la creación de sistemas intuitivos y útiles.

# Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

El diseño de arquitecturas de software organiza componentes para cumplir con los requisitos del sistema. En la computación en la nube, se divide en dos capas: una de aplicación con funcionalidades y otra de infraestructura que las soporta. Los arquitectos deben equilibrar atributos como rendimiento y seguridad. Un desafío común es la falta de patrones de diseño específicos para la nube, lo que complica el diseño para los menos experimentados. Se utilizan arquitecturas genéricas que permiten un diseño flexible y adaptable, aplicando patrones tradicionales adaptados a la nube para organizar los componentes eficientemente.

## Reflexión

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



## Bibliografía

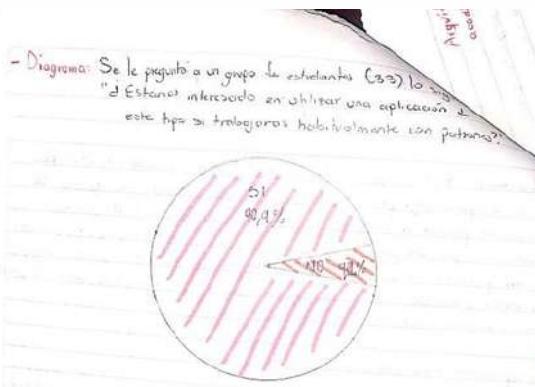
- Albin, S. T. (2003). *The art of software architecture: design methods and techniques*. Hoboken: John Wiley & Sons. [ [Links](#) ]

# Desarrollo de una herramienta para el aprendizaje de patrones de diseño software

Este proyecto desarrolla una aplicación web educativa para facilitar el aprendizaje de patrones de diseño orientados a objetos. Los profesores pueden configurar actividades para que los estudiantes identifiquen, apliquen e implementen estos patrones. Inicialmente, se incluirán patrones del libro de Gamma, pero la estructura permitirá añadir más patrones. La aplicación sigue una arquitectura de tres capas: presentación (interfaz de usuario), negocio (lógica y operaciones) y datos (gestión de la base de datos). Se implementarán pruebas unitarias para asegurar la calidad y flexibilidad del código.

## Reflexión

Este proyecto propone una plataforma innovadora para enseñar patrones de diseño, transformando cómo estudiantes y profesionales aprenden y aplican estos conceptos clave en desarrollo de software. Su arquitectura de tres capas mejora la escalabilidad y facilita la expansión futura de contenidos. La herramienta enfrenta desafíos típicos de proyectos pioneros, como la falta de referencias, lo que requiere un análisis detallado para garantizar su robustez. Al ser extensible y de código abierto, permite que desarrolladores contribuyan sin comprometer la integridad del sistema.

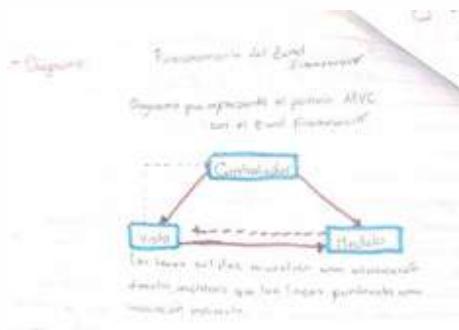


## Bibliografía

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

# Framework utilizando patrones de diseño y arquitectura MVC/REST

Este proyecto tiene como objetivo reemplazar un framework obsoleto para mejorar la gestión de contenidos audiovisuales en una empresa. Inicialmente, se introduce HTML, pero con el avance de XML, se requiere un sistema flexible que genere múltiples formatos. El problema del framework actual radica en su alto acoplamiento y la necesidad de especialistas para su mantenimiento. Se evalúa modificar el sistema existente o adoptar uno nuevo, como el Zend Framework, que permite una arquitectura de capas, ofreciendo flexibilidad, rapidez en el aprendizaje y capacidad de expansión, mejorando la oferta digital de la empresa.



## Reflexión

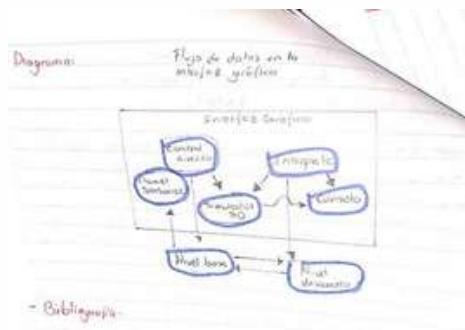
La renovación de un framework para adaptarse a los cambios tecnológicos refleja una visión estratégica y proactiva, esencial en el ámbito actual. La empresa enfrenta el desafío de mejorar su estructura interna para responder a las demandas del mercado y ofrecer una experiencia óptima a sus clientes. La implementación de un nuevo framework que separa la lógica de negocio, acceso a datos y presentación resuelve problemas de acoplamiento y mantenimiento, sentando las bases para un crecimiento sostenible. Este enfoque destaca la importancia de los patrones de software como soluciones probadas, permitiendo integrar nuevas tecnologías sin comprometer la estabilidad.

## Bibliografía

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

# Desarrollo de una arquitectura de software para el robot móvil Lázaro

Este artículo explora arquitecturas de software aplicadas en robótica, enfocándose en los enfoques deliberativo, reactivo e híbrido. La arquitectura deliberativa permite una actuación rápida en entornos conocidos, mientras que la reactiva responde eficazmente a cambios imprevistos. Las arquitecturas híbridas combinan ambos enfoques y son útiles en robots que requieren adaptabilidad. Se presenta un caso práctico: el desarrollo de una arquitectura de tres niveles para el robot móvil Lázaro, diseñado para exploración en terrenos irregulares. Implementada en C#, incluye control de actuadores, monitoreo de sensores y una interfaz para programación y simuladores 3D.



## Reflexión

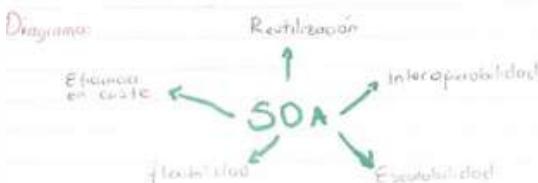
Este artículo resalta la importancia de desarrollar arquitecturas de software flexibles y adaptables para robots, permitiéndoles actuar en entornos cambiantes y enfrentar situaciones imprevistas. Los distintos enfoques arquitectónicos—deliberativo, reactivo e híbrido—buscan equilibrar la planificación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

## Bibliografía

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

# Arquitectura de software, esquemas y servicios

El artículo analiza la evolución de la arquitectura de software hacia arquitecturas orientadas a servicios (SOA) para aplicaciones empresariales. Se destaca la transición de sistemas monolíticos a arquitecturas modulares y escalables con bajo acoplamiento. Los "esquemas" se definen como clases unificadas con funcionalidades reutilizables, optimizando el desarrollo. En SOA, cada funcionalidad es un servicio autónomo que se comunica mediante mensajes, lo que asegura flexibilidad y escalabilidad. Sin embargo, SOA presenta retos técnicos, que se solucionan centralizando procesos en servicios de negocio dedicados.



## Reflexión

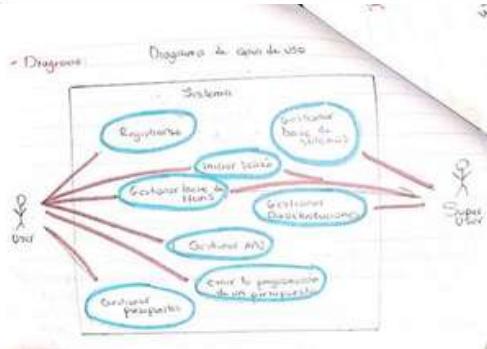
La arquitectura de software orientada a servicios representa un avance esencial para responder a las demandas modernas de flexibilidad y escalabilidad en aplicaciones empresariales. Al separar funciones en servicios independientes y reutilizables, se promueve una mayor adaptabilidad, facilitando la integración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

## Bibliografía

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

# Arquitectura de software para el desarrollo de herramienta Tecnológica de Costos, Presupuestos y Programación de obra

La introducción destaca la importancia de las Tecnologías de la Información y las Comunicaciones (TIC) en el desarrollo humano, la economía y la educación. Las TIC son clave en la formación profesional, lo que exige que las instituciones de educación superior adapten sus métodos de enseñanza. En Ingeniería Civil, el artículo resalta la necesidad de herramientas tecnológicas, especialmente en gestión de costos y presupuestos. El objetivo es diseñar un software educativo accesible que complemente el aprendizaje teórico y práctico en la UFPS, apoyando el proceso de formación en "Costos, Presupuestos y Programación de obra".



## Reflexión

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

## Bibliografía

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

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte

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

## - Diagrama -



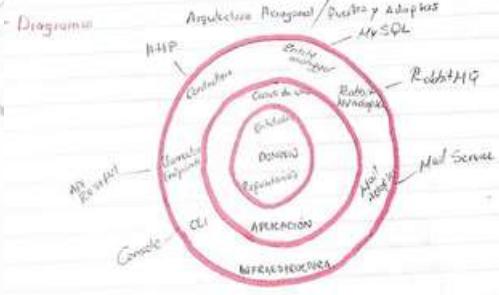
## Reflexión

La evolución de la Arquitectura de Software, particularmente a través de los lenguajes de patrones, refleja el crecimiento constante y la adaptación de la ingeniería del software a los desafíos del mundo tecnológico. En un entorno donde los sistemas de información se vuelven cada vez más complejos, la capacidad para diseñar soluciones eficientes y mantenibles es crucial. Los lenguajes de patrones no solo permiten resolver problemas arquitectónicos con mayor eficacia, sino que también brindan una base sólida para futuros avances en el campo.

## Bibliografía

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

# Implementación de una Arquitectura de Software guiada por el Dominio



## Reflexión

El Diseño Orientado al Dominio (DDD) ve el desarrollo de software como un proceso ligado al entendimiento del negocio. Involucrando a expertos del dominio, DDD crea soluciones que responden a las necesidades del negocio, generando un "lenguaje ubicuo" que une a todos los miembros del equipo. Al aplicar la Arquitectura Limpia, se garantiza flexibilidad y adaptabilidad, permitiendo a las organizaciones responder ágilmente a cambios.

## Bibliografía

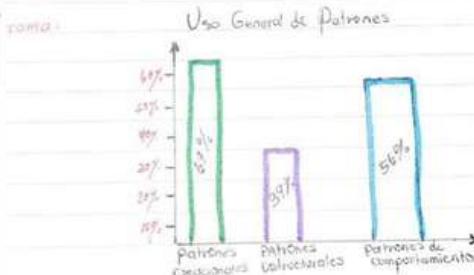
Vivas, L; Cambarieri, M: Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo. CACIC. (2013)

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

# Patrones de diseño GOF (The Gang of Four) en el contexto de procesos de Desarrollo de Aplicaciones Orientadas a la web

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

Toma:



## Reflexión

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

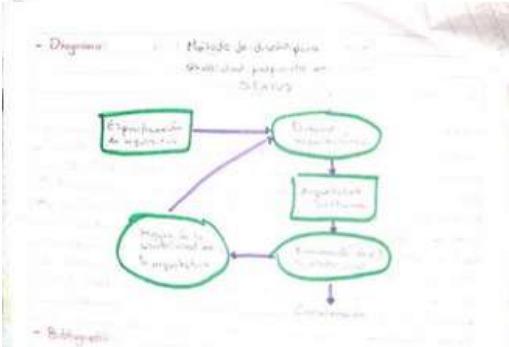
## Bibliografía

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

# Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el Momento Arquitectónico.

Este artículo presenta una aproximación para mejorar la usabilidad de los sistemas software desde las etapas iniciales del desarrollo, específicamente en el diseño arquitectónico.

La propuesta, parte del proyecto Europec IST STATUS, busca anticipar la evaluación y mejora de la usabilidad antes de la construcción del sistema, a diferencia del enfoque tradicional que se centra en mejorar la usabilidad una vez que el sistema está completo. La investigación identifica patrones de usabilidad, que son mecanismos específicos (como "deshacer" o "alertas") que se incorporan a la arquitectura para mejorar la usabilidad del producto final. Estos patrones son descritos con un enfoque arquitectónico, lo que permite aplicar mejoras en las primeras fases del diseño.



## Reflexión

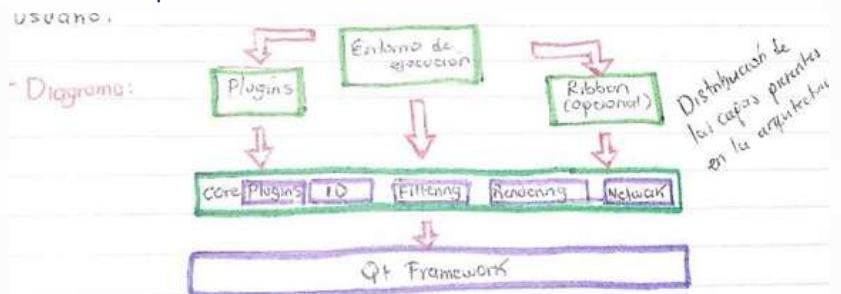
El artículo refleja la importancia de la accesibilidad y la experiencia del usuario en el desarrollo de software, enfatizando cómo estas prácticas deben ser integradas desde el inicio de un proyecto. La accesibilidad no solo mejora la inclusión de personas con discapacidades, sino que también beneficia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

## Bibliografía

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

# Arquitectura de software para el sistema de visualización médica Vismedic

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



## Reflexión

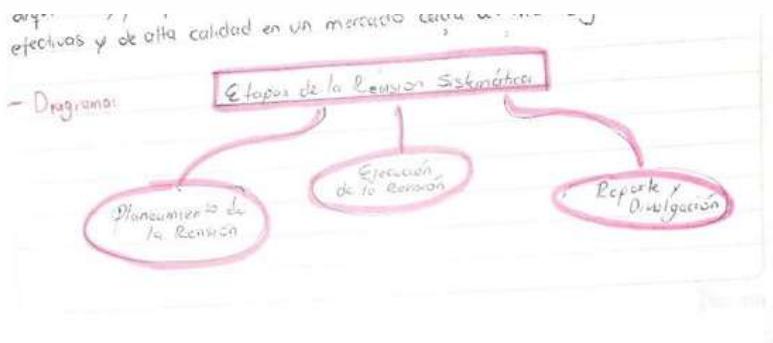
La propuesta arquitectónica para Vismedic subraya la importancia de un diseño modular y flexible en sistemas médicos, donde la precisión y la eficiencia son esenciales. Al integrar principios de componentes, capas y filtros, el sistema no solo logra una adaptabilidad y escalabilidad que facilitan su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

## Bibliografía

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

# Revisión sistemática sobre generadores de código fuente y patrones de arquitectura

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



## Reflexión

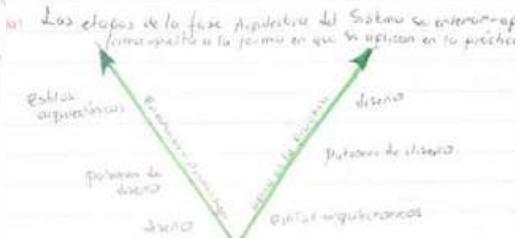
La automatización mediante generadores de código y la correcta elección de patrones de arquitectura son fundamentales para el desarrollo eficiente de software. Integrar estas herramientas y enfoques permite a los desarrolladores crear aplicaciones más organizadas, mantenibles y escalables, optimizando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

## Bibliografía

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

# Una Teoría para el Diseño de Software

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



## Reflexión

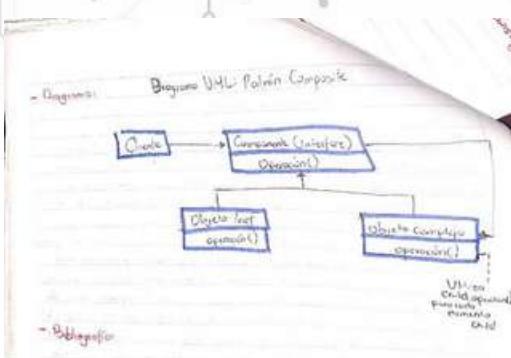
Este artículo resalta la importancia del diseño de software como una fase fundamental en el desarrollo de sistemas tecnológicos. Reflexionando sobre ello, podemos concluir que un diseño sólido no solo define la estructura de un sistema, sino que también influye en la eficiencia, la escalabilidad y la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

## Bibliografía

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

# Perfiles UML para definición de Patrones de Diseño

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



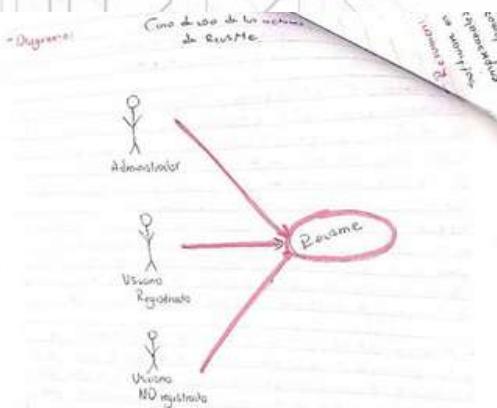
## Reflexión

El artículo muestra el valor de los perfiles UML para representar patrones de diseño en el desarrollo de software, destacando como estos perfiles amplían la flexibilidad y precisión en la documentación de soluciones reutilizables. Refleja la importancia de los patrones como guías que los desarrolladores pueden seguir para resolver problemas recurrentes, y como los perfiles UML contribuyen a establecer un lenguaje común para ellos, lo que facilita la colaboración y comprensión en los equipos de trabajo. Sin embargo, el artículo también sugiere la necesidad de seguir investigando para integrar estos perfiles en herramientas UML, de manera que se haga más accesible su aplicación en la práctica.

## Bibliografía

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

# Herramienta para reúso de código JavaScript Orientado a Patrones de interacción



## Reflexión

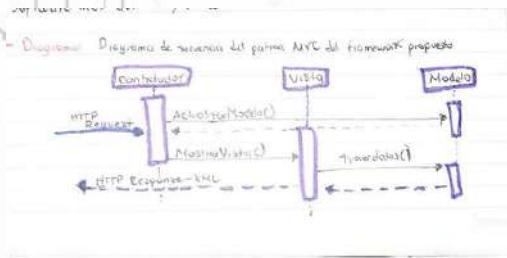
La reflexión sobre la herramienta "ReusMe" destaca su papel en la optimización del desarrollo de aplicaciones web mediante la reutilización de componentes. Al contar con patrones predefinidos y modelos UML, esta herramienta permite a los desarrolladores construir aplicaciones de manera más rápida y eficiente, aprovechando componentes ya diseñados y probados. Esto no solo ahorra tiempo y recursos, sino que también facilita la creación de interfaces de usuario consistentes, estandarizadas y de alta calidad.

## Bibliografía

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

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

# Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web



## Reflexión

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

Este trabajo busca crear un framework para el desarrollo rápido de aplicaciones web, enfocado en minimizar costos y tiempos de desarrollo a través de un diseño reutilizable, basado en patrones de diseño y de interfaz. Este framework implementa el patrón MVC y utiliza PHP5 orientado a objetos y tecnologías como AJAX para facilitar la interacción en entornos web. La herramienta permite a los desarrolladores concentrarse en los requisitos funcionales específicos de cada aplicación, mientras reutilizan una estructura probada que favorece la escalabilidad, la usabilidad y el mantenimiento de aplicaciones web.

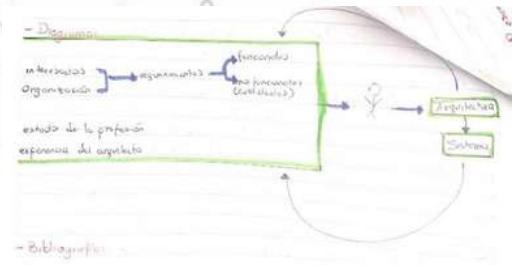
## Bibliografía

Brooke John, "SUS - A quick and dirty usability scale", <http://www.usabilitynet.org/trump/documents/suschart.doc>, Fecha de acceso: 17/12/2008

# Introducción a la Arquitectura de Software

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

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



## Reflexión

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

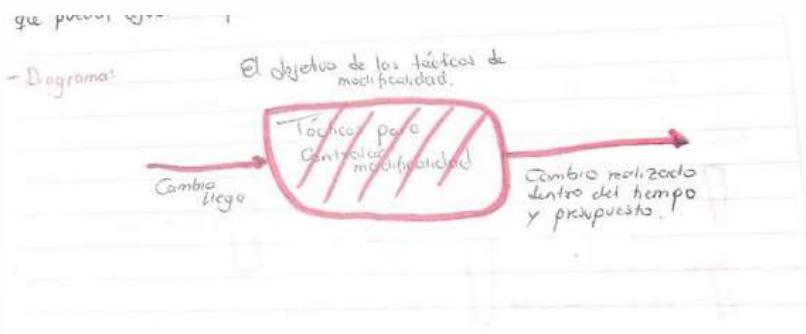
Este enfoque no solo resuelve problemas técnicos, sino que también crea puentes entre los objetivos de negocio y las capacidades técnicas, convirtiendo la arquitectura en una disciplina estratégica.

## Bibliografía

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

# Aplicaciones de patrones de Diseño para garantizar Alta flexibilidad en el Software

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



## Reflexión

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

## Bibliografía

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sornmerla, Michael Stal. "PATTERN- ORIENTED SOFTWARE ARCHITECTURE, Volumen 1: A System of Patterns". John Wiley & Sons, 1996.

# Arquitectura de software académica para la comprensión del desarrollo de software en capas

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



## Reflexión

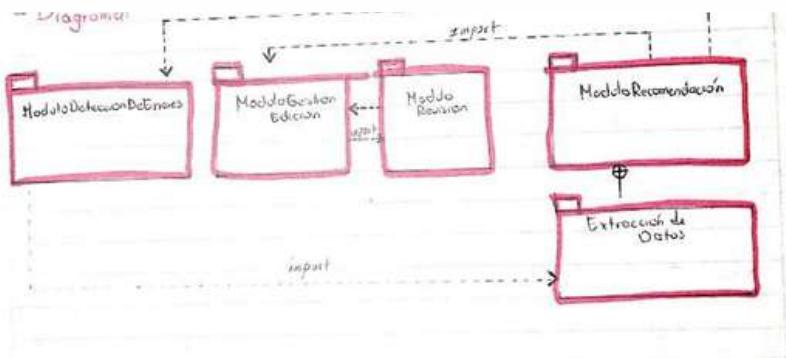
La arquitectura en capas es fundamental en el desarrollo de software, ya que permite organizar el sistema en unidades funcionales separadas, lo que facilita la modularización y la separación de responsabilidades. Este enfoque mejora la escalabilidad, el mantenimiento y la evolución del sistema, permitiendo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

## Bibliografía

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

# Módulo de recomendación de patrones de diseño para EGPat

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



## Reflexión

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

## Bibliografía

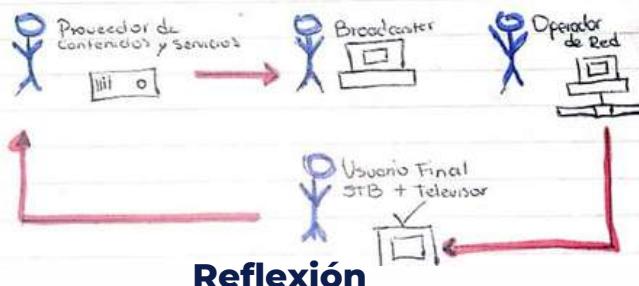
ACM. (2013). ACM Recommender Systems community. PACM RecSys 2013. <http://recsys.acm.org/recsys13/> [ Links ]

# Arquitectura de Software para el Soporte de Comunidades Académicas Virtuales en Ambientes de Televisión Digital Interactiva

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

brechas sociales y construir comunidades más inclusivas.

- Diagrama: Cadena extremo a extremo para TDI



## Reflexión

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

## Bibliografía

- 1) Arreniemi, P., Modeling and Content Production of Distance Learning concept for Interactive Digital Television. Tesis Doctoral. Universidad de Tecnología de Helsinki, Finlandia. <http://lib.tkk.fi/Diss/2006/isbn9512285428/> (2006).

# Identificación y clasificación de patrones de diseño de servicios web para mejorar QoS

Esta tesis aborda el uso de patrones de diseño en servicios web para optimizar la calidad del servicio (QoS) en arquitecturas basadas en SOA y microservicios. Se propone un inventario de patrones que incluye su descripción, contexto de uso y beneficios asociados. A través de un análisis detallado, se examina cómo estos patrones influyen en aspectos clave como interoperabilidad,

funcionalidad y eficiencia del sistema. Además, se evalúan los desafíos que surgen al implementar estos patrones y se presentan recomendaciones para su correcta integración en entornos empresariales. El estudio demuestra cómo la aplicación estratégica de patrones mejora la capacidad de respuesta, escalabilidad y mantenibilidad de los sistemas distribuidos, garantizando así soluciones más robustas y adaptadas a las necesidades actuales.



## Reflexión

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

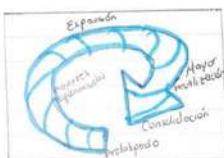
## Bibliografía

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

# Desarrollo de sistemas de software con patrones de diseño orientado a objetos

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

Diagrama: Ciclo de vida del Software orientado a Objetos



## Reflexión

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

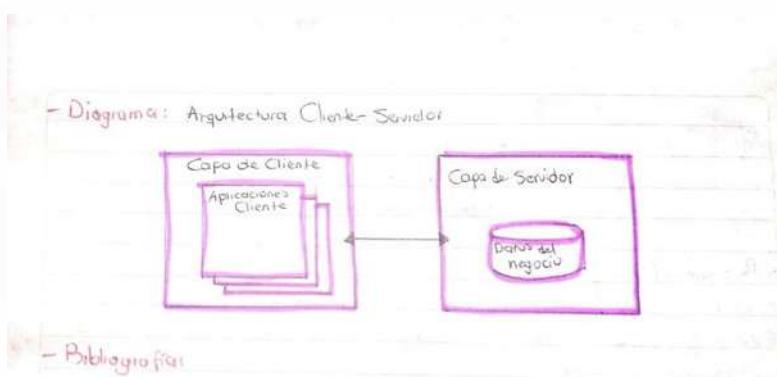
## Bibliografía

[ALE77] CHRISTOPHER ALEXANDER (1977) "A Pattern Language" (Oxford University Press 1977).

# Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales

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

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



## Reflexión

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

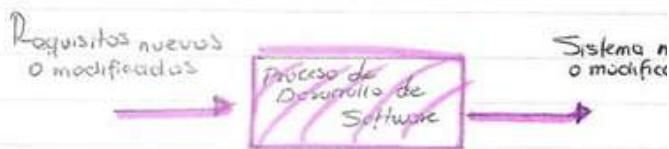
## Bibliografía

(informática), W. -A. (21 de 06 de 2014). Obtenido de [http://es.wikipedia.org/wiki/Arquitectura\\_en\\_pipeline\\_\(inform%C3%A1tica\)](http://es.wikipedia.org/wiki/Arquitectura_en_pipeline_(inform%C3%A1tica))

# Atributos de Calidad y Arquitectura del Software

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

grama: Desarrollo de Software



iografía:

## Reflexión

El artículo resalta la complejidad y la importancia de diseñar una arquitectura de software que no solo cumplía con los requisitos funcionales, sino que también garantice la calidad del sistema en aspectos como la mantenibilidad, seguridad y rendimiento. A medida que se equilibran estos atributos, es fundamental comprender que cualquier cambio en la arquitectura para mejorar un aspecto puede tener repercusiones en otros, lo que hace necesario un enfoque estratégico y bien informado.

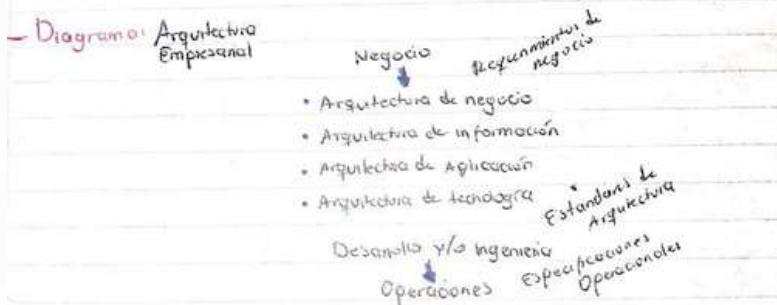
## Bibliografía

Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice. SEI Series in Software Engineering. Addison-Wesley, 2 edition, 2003.

# Arquitectura de Software en el Proceso de Desarrollo Ágil. Una Perspectiva Basada en Requisitos Significantes para la Arquitectura.

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

un software más robusto y alineado con las necesidades del negocio



## Reflexión

La reflexión sobre la integración de la Arquitectura de Software (AS) con las Metodologías Ágiles (MA) nos muestra que, aunque en principio parecen incompatibles, pueden complementarse si se enfocan en los Requisitos Significantes para la Arquitectura (RSA). Estos requisitos esenciales permiten una arquitectura flexible que se adapta a los cambios sin perder estabilidad. La clave es encontrar un equilibrio entre la flexibilidad de las MA y la estabilidad necesaria en la AS, lo que fomenta una mayor colaboración entre los equipos y un desarrollo más eficiente.

## Bibliografía

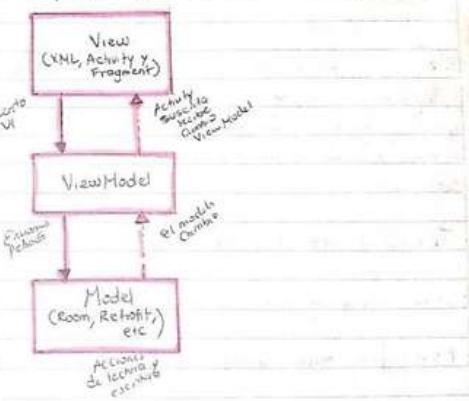
Schwaber, K., Sutherland, J., "The scrum guide. The Definitive Guide to Scrum". En: <https://www.scrumguides.org/docs/scrum-guide/v2017/2017-Scrum-Guide-US.pdf>. Accedido el 20/02/2017.

# Análisis comparativos de patrones de Diseño de Software

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

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

-Diagrama: Componentes patrón de diseño Model View ViewModel



## Reflexión

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

## Bibliografía

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

# Evaluando la arquitectura de software

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

1. ALMA (Architecture Level Modifiability Analysis), que analiza la facilidad de modificación en sistemas para estimar costos de mantenimiento, identificar riesgos o comparar arquitecturas según su adaptabilidad.

2. PASA (Performance Assessment of Software Architecture), enfocado en medir el desempeño de la arquitectura mediante análisis de casos de uso críticos, anti-patrones y modelos de rendimiento.

3. SNA (Survivable Network Analysis), desarrollado por el CERT, que evalúa la capacidad de supervivencia de un sistema ante ataques, fallas o accidentes, garantizando la continuidad de servicios esenciales.

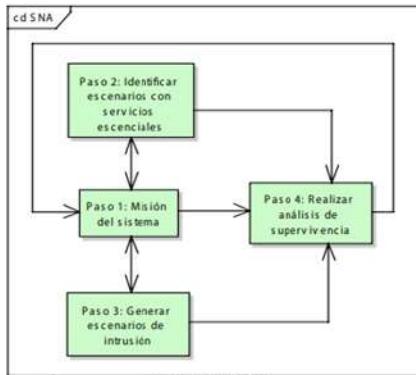


Figura 4. Método SNA

## Reflexión

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

## Bibliografía

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

# Especificación de la arquitectura de software

El artículo aborda los componentes clave de la arquitectura de software y la importancia de los patrones arquitecturales para organizar sistemas complejos en un contexto de creciente demanda tecnológica. Destaca que una arquitectura efectiva combina métodos documentados, intuición y reutilización de soluciones previas. Se exploran diferentes tipos de patrones: arquitectónicos, que afectan la estructura global del sistema; de diseño, que definen relaciones entre subsistemas; y de lenguaje, específicos para implementar problemas usando características de programación. También se detallan categorías como estructuras organizadas, sistemas distribuidos, sistemas interactivos, y sistemas adaptables, enfatizando su contribución a la eficiencia, usabilidad y adaptabilidad del software.

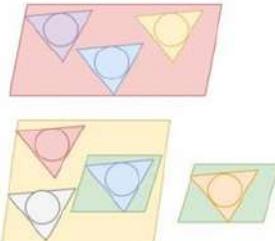


Ilustración 2: Ejemplos de sistemas de software con diferentes patrones

## Reflexión

La arquitectura de software no es simplemente un conjunto de reglas estáticas o recetas para seguir, sino un proceso dinámico que requiere creatividad, experiencia y un enfoque estratégico. Este artículo nos invita a reflexionar sobre la importancia de los patrones arquitecturales como herramientas que, lejos de simplificar de forma mecánica el diseño de sistemas complejos, potencian nuestra capacidad de resolver problemas.

## Bibliografía

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

# Evaluación de una Arquitectura de Software

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

Se evaluaron arquitecturas mediante el método SAAM, priorizando la modificabilidad debido a los constantes cambios normativos. La arquitectura MVC fue seleccionada por su modularidad, bajo costo de actualización y escalabilidad.

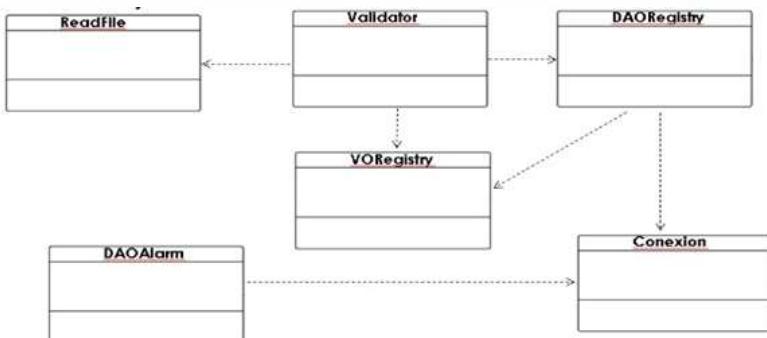


Figura 3. Estructura básica del Modelo

## Reflexión

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

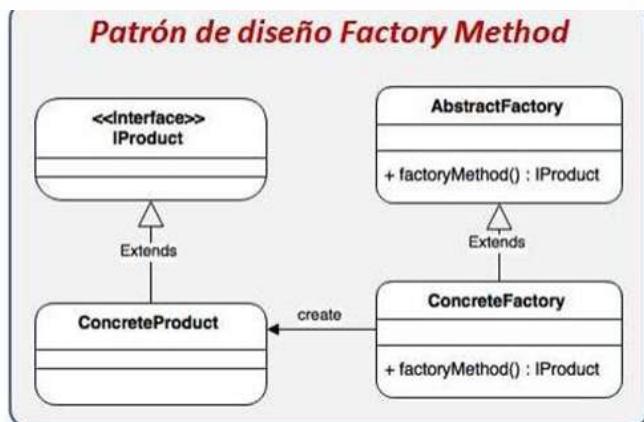
## Bibliografía

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

# Introducción a los patrones de diseño

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

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



## Reflexión

Los patrones de diseño son soluciones reutilizables para problemas comunes en el desarrollo de software, basadas en experiencias previas que han demostrado su efectividad. Su implementación permite evitar la necesidad de reinventar soluciones para cada problema, mejorando la calidad, la escalabilidad y el mantenimiento del código. Sin embargo, su uso requiere experiencia, ya que es crucial identificar el contexto adecuado para cada patrón. Aplicarlos de manera incorrecta puede generar más complejidad.

## Bibliografía

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

# Utilización de antipatrones y patrones en el análisis de software

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

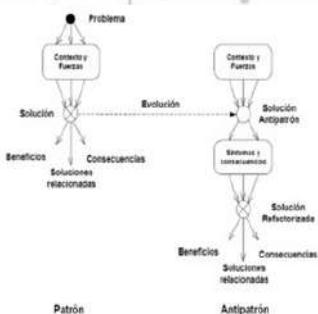


Figura 1. Comparación entre Patrón y Antipatrón

## Reflexión

El artículo subraya la importancia de los patrones y antipatrones en el análisis y desarrollo de software, brindando herramientas esenciales para mejorar la calidad del producto final. Mientras que los patrones de análisis proporcionan soluciones probadas que optimizan el proceso de modelado y diseño, los antipatrones alertan sobre errores comunes y malas prácticas que pueden tener consecuencias negativas. Este enfoque no solo ayuda a evitar errores, sino que también promueve una mejor comunicación y comprensión entre los miembros del equipo de desarrollo.

## Bibliografía

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

# Arquitectura de software para los Laboratorios Virtuales

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

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

## Bibliografía

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

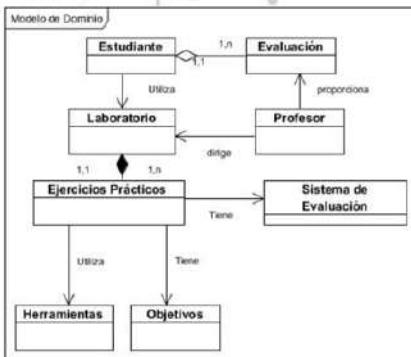


Figura 6. Modelo de dominio.

## Reflexión

La propuesta de una arquitectura de software para laboratorios virtuales destaca la importancia de una estructura bien definida para agilizar el desarrollo y facilitar la reutilización de componentes. En un mundo donde la eficiencia y la calidad son clave, contar con una AS sólida permite no solo cumplir con los requisitos funcionales, sino también garantizar la usabilidad, fiabilidad y sostenibilidad a largo plazo del sistema. La modularidad y la integración de componentes son esenciales para crear soluciones escalables y flexibles, adaptables a diversas necesidades educativas, lo que refuerza el valor de la arquitectura en el éxito de proyectos tecnológicos.

# Arquitectura de software para sistema gestión de inventarios

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

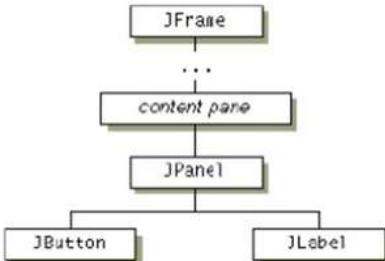


Fig. 12. Ejemplo de Jerarquía de clases de Swing

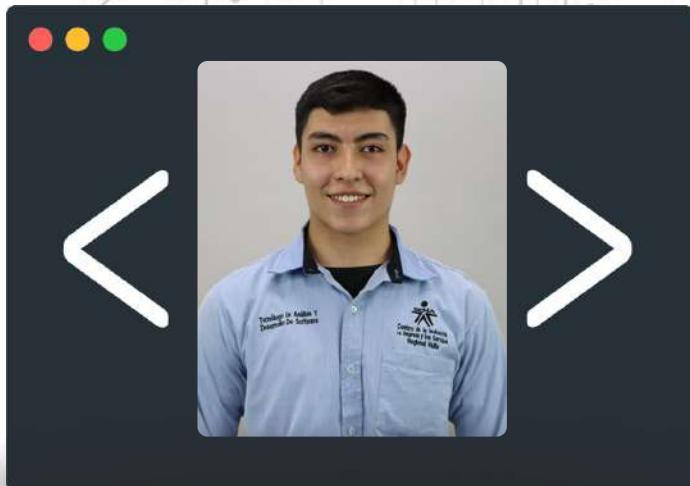
## Reflexión

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

## Bibliografía

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

# Cristian Jeanpool Bahamon Granado



Mi nombre es Cristian Jeanpool Bahamon Granados, tengo 18 años y actualmente soy aprendiz del programa de Tecnólogo en Análisis y Desarrollo de Software en el Sena.

El mundo del desarrollo de software me entusiasma mucho, especialmente cuando se trata de encontrar soluciones innovadoras y efectivas a los problemas.

En mis ratos libres, me dedico a expandir mis conocimientos sobre programación, nuevas tecnologías y metodologías ágiles.

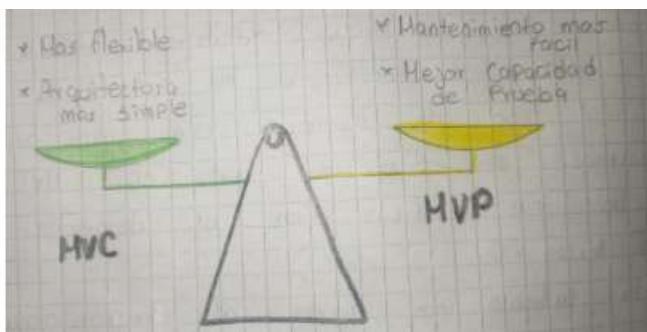
Mi objetivo es completar con éxito el programa de tecnólogo y luego continuar mis estudios en Ingeniería de Software, lo que me permitirá consolidar mi formación y poder trabajar como ingeniero.

# Análisis comparativo de patrones de diseño MVC y MVP para el rendimiento de aplicaciones web

En este artículo se evaluó la eficiencia de los patrones de diseño en la realización de un proyecto, enfocándose en elegir el patrón más adecuado. Lo primero fue identificar y validar los patrones existentes, seleccionando el MVC y el MVP. Cada uno tuvo su propio proyecto para realizar la prueba , donde se evaluaron diferentes aspectos, como el tiempo de desarrollo, las líneas de código y el uso de memoria ram, entre otros. Como resultado, se demostró que el MVC es mucho mas eficaz en el desarrollo web.

## Reflexión

Este artículo nos ayuda al proporcionarnos la evaluación de dos patrones de diseño y cual de los dos es más eficiente en el desarrollo web. Gracias a esto, podemos saber cuál utilizar en algún proyecto futuro, ya que nos ofrece una amplia evaluación con toda la información sobre las pruebas. Nos indica, desde cual ahorra mas linea de codigo, hasta cual consume menos ram. Esto es muy importante, ya que con esta información, podemos asegurar que nuestro proyecto tenga un mejor rendimiento.



## Bibliografía

Gonzales Gonzales, C. E. (2023). Análisis comparativo de patrones de diseño MVC y MVP para el rendimiento de aplicaciones web.

# Análisis comparativo de patrones de diseño de software

El artículo nos dirá cómo los patrones de diseño nos ayudan con algunos problemas comunes en el desarrollo de software, también nos explica sobre los patrones como el MVC, MVP , MVVM, dando su explicación de su estructura, sus componentes, ventajas y desventajas. El artículo también nos dice que ningún patrón es superior en todos los aspectos ya que cada uno cumple con un propósito en particular y ya dependerá del programador cual usa



## Reflexión

El artículo ofrece una comprensión clara de los patrones de diseño y como su correcta aplicación puede optimizar el desarrollo de software. La idea de que no hay un patrón universalmente superior es crucial, ya que refuerza la importancia de seleccionar la herramienta adecuada según el contexto y las necesidades específicas . Esto nos invita a pensar de manera más clara y estratégicamente sobre qué patrón usar en proyectos futuros.

## 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.

# **Modelo para la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software**

El artículo propone un modelo basado en inteligencia artificial para ayudar a personas a seleccionar el patrón que se acomode a su proyecto. Clasifica los patrones en estructurales, creacionales y de comportamiento dando sus ventajas y características. El modelo facilita la toma de decisiones ya que identificará el patrón para cada proyecto y esto promueve la reutilización y eficiencia.



## **Bibliografía**

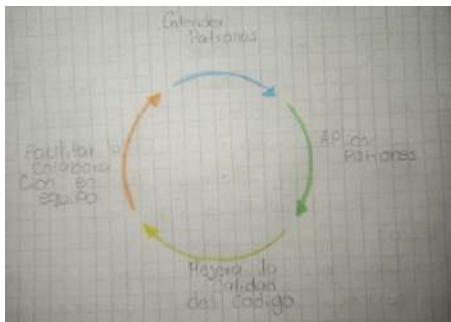
Villa, M. C., & Pérez, P. Y. P. MODELO PARA LA AYUDA A LA TOMA DE DECISIONES EN LA SELECCIÓN DE PATRONES DE DESARROLLO DE SOFTWARE.

## **Reflexión**

El artículo tiene como objetivo resaltar el uso del modelo para la identificación de patrones de diseño ya que facilita la decisión de elegir qué patrón usar. Con la ayuda del modelo se fomenta la realización, la modularidad y el diseño eficiente, esto nos demuestra cómo la inteligencia artificial nos ayuda en procesos técnicos y en la mejora de calidad del software.

# Patrones de diseño

El artículo nos explica cómo los patrones de diseño ayudan a resolver problemas comunes en el desarrollo de software. Se mencionan patrones creacionales como Observer y Strategy. También detalla sus ventajas y desventajas y añade su propósito al igual nos dice cómo los patrones mejoran la modularidad, escalabilidad y flexibilidad del código. Además, aborda los patrones para interfaces gráficas.



## Reflexión

El artículo nos muestra cómo los patrones son importantes para crear sistemas más organizados y eficientes. La aplicación de patrones de diseño no solo facilita la solución de problemas comunes, sino que también mejora la calidad del código y la colaboración entre desarrolladores. Sin embargo, es importante tener en cuenta sus desventajas como la necesidad de aprendizaje inicial y su cuidadosa implementación para evitar errores.

## Bibliografía

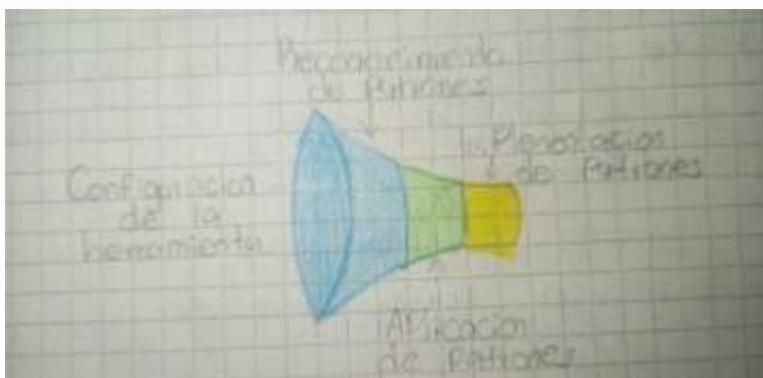
- Alpízar, J. C. M., Rodriguez, C. O., & Bolaños, L. E. (2017). Patrones de diseño.

# Desarrollo de una herramienta para el aprendizaje de patrones de diseño

Este artículo documenta el proceso de creación de una herramienta diseñada para facilitar el aprendizaje de los patrones de diseño orientada a objetos. La herramienta permitirá que las personas puedan identificar, aplicar y entender los patrones de diseño a través de una herramienta web más interactiva. Se creó con interfaz intuitiva y herramientas como cuestionarios para poder ayudar a los usuarios a poder seleccionar el patrón específico.

## Reflexión

Este artículo nos muestra como el desarrollo de una herramienta web para el aprendizaje de patrones de diseño orientados a objetos facilita la comprensión y aplicación de estos patrones. Con una interfaz intuitiva y el uso de herramientas como cuestionarios, la herramienta no solo ayuda a los usuarios a identificar y aplicar patrones, sino que también hace más accesible el aprendizaje de conceptos clave en el desarrollo de software



## Bibliografía

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

## Relaciones con Patrones de Diseño

El artículo nos presenta una herramienta llamada pLinker, diseñada para facilitar el uso e integración de patrones de diseño en diagramas UML. Su objetivo principal es gestionar las relaciones entre patrones y automatizar su integración en diseños existentes, permitiendo que los usuarios mejoren sus proyectos de software. Además, la herramienta ofrece sugerencias sobre qué patrones son más adecuados para arquitecturas de software específicas, lo que contribuye a crear proyectos más organizados y flexibles.



## Bibliografía

Liebener, L., Rossi, M. A., & Marcos, C. (2003). pLinker: Relaciones con Patrones de Diseño. Facultad de Ciencias Exactas.

## Reflexión

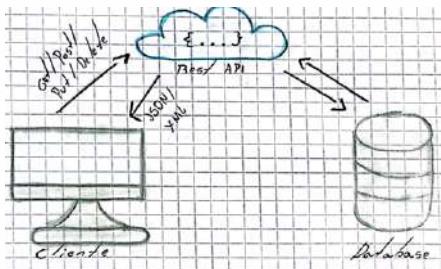
La herramienta es muy útil porque ayuda a los usuarios a trabajar con patrones de diseño de forma más sencilla, incluso si no tienen mucha experiencia. Al automatizar la integración de los patrones y sugerir cuál es el más adecuado para su proyecto, logra que los diseños sean más organizados y flexibles, mejorando así la calidad del trabajo realizado.

# Implementación de MVCC para la Generación de Aplicaciones Web en Tiempos de Entrega Reducidos

El artículo nos muestra una arquitectura innovadora para aplicaciones web llamada MVCC (Modelo - Vista- Controlador - Controlador), que optimiza tiempos de entrega al dividir la lógica de control en 2 controladores: uno en el cliente usando JavaScript con AJAX y otro en el servidor e implementando REST. Esto permite mejor compatibilidad con dispositivos móviles. La propuesta también incluye el uso de JSON para la comunicación entre controladores y el framework RedBeanPHP para facilitar el manejo de base de datos.

## Bibliografía

Rodríguez, I. Y. A., & Cortes, E. E. P. R. Implementación de MVCC para la generación de Aplicaciones Web en tiempos de entrega reducidos.  
<https://n9.cl/rctbb>



## Reflexión

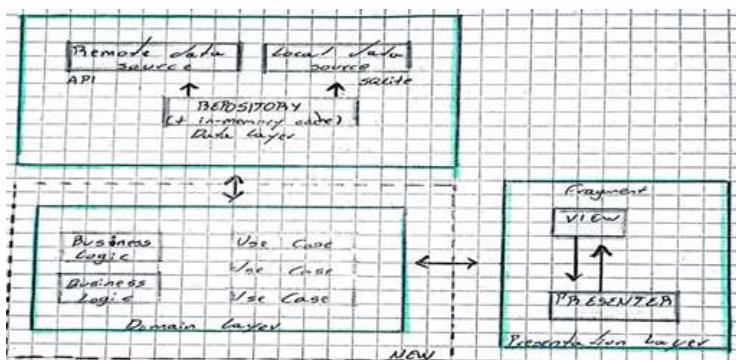
La arquitectura MVCC nos demuestra como la incorporación de tecnologías modernas puede transformar el desarrollo web, equilibrando la carga entre cliente y servidor, lo que permite aplicaciones más rápidas y versátiles. Actualizar framework y técnicas con una curva de aprendizaje baja, la propuesta resulta accesible para desarrolladores con diferentes niveles de experiencia, fomentando buenas prácticas de programación sin sacrificar la eficiencia. Esta metodología nos muestra la importancia de actualizar herramientas y paradigmas para satisfacer las necesidades actuales.

# Repercusión de Arquitectura Limpia y la Norma ISO/IEC 25010 en la mantenibilidad de Aplicativos Android

El artículo evalúa el impacto de la arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad del aplicativo móvil Educar Teacher, comparándolo con una aplicación convencional (CRM Distribución). La arquitectura limpia, propuesta por Robert Martin, organiza las aplicaciones en capas separadas por responsabilidades, promoviendo cohesión y bajo acoplamiento. La investigación emplea cuasi-experimental con métrica de analizabilidad, cambiabilidad, estabilidad y testeabilidad. Los resultados muestran mejoras significativas, lo cual evidencia que adoptar una arquitectura limpia mejora la capacidad de mantenimiento, etc.

## Reflexión

El artículo muestra la relevancia de la arquitectura limpia y las normas de calidad en el desarrollo de software, especialmente en aplicaciones móviles que requieren actualizaciones constantes. La mejora en las habilidades demuestra la eficiencia de dividir responsabilidades en capas, mientras que los incrementos modestos en otras métricas reflejan desafíos en la implementación completa de buenas prácticas. Este enfoque fomenta la creación de aplicaciones más duraderas y escalables, aunque existe un compromiso inicial de aprendizaje. Esto garantiza su sostenibilidad, reduce costos y aumenta la satisfacción del cliente.



## Bibliografía

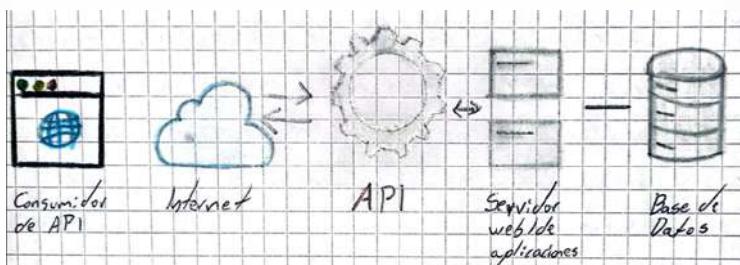
- Arias-Orezano, J. F., Reyna-Barreto, B. D., & Mamani-Apaza, G. (2021). Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android. *TecnoLógicas*, 24(52), 226-241. <https://n9.cl/s4de8>

# Metodología para la Enseñanza del Desarrollo de Software Modular

El artículo presenta una metodología educativa para la enseñanza del desarrollo de software modular, orientada a estudiantes de ingeniería en sistemas. La metodología integra el contexto como elemento clave para conectar teoría y práctica, fomentando habilidades de diseño y codificación escalables. A través de un enfoque iterativo, los estudiantes desarrollan una API web, siguiendo etapas como levantamiento de requisitos, diseño conceptual, implementación y pruebas. Se destacan conceptos claves como la programación orientada a objetos, patrones de diseño y el uso de herramientas como Swagger y DocAsCode.

## Reflexión

El artículo refleja como una metodología bien estructurada puede cerrar problemas en la formación de desarrolladores, mostrando que integrar el contexto y herramientas adecuadas facilita el aprendizaje. El énfasis en prácticas modernas, como código limpio y documentación integrada, prepara a los estudiantes para enfrentar desafíos reales. Ese enfoque resalta la importancia de enseñar no sólo técnicas de codificación, sino también habilidades críticas de diseño, validación y documentación, asegurando un buen futuro profesional.

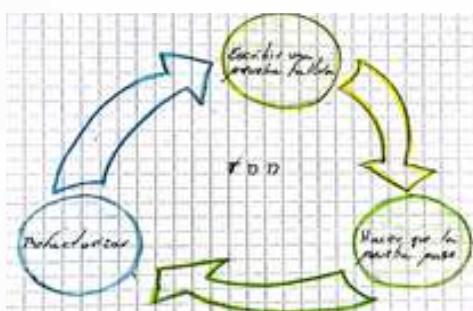


## Bibliografía

Guevara, W. J. T. Metodología para la enseñanza del Desarrollo de Software Modular. <https://n9.cl/7hu2ba>

# Calidad Ágil: Patrones de Diseño en un Contexto de Desarrollo Dirigido por Pruebas

El artículo explora la aplicación de patrones de diseño en el desarrollo dirigido por pruebas (TDD) dentro de las metodologías ágiles. Este resalta la importancia de asegurar la testeabilidad de los patrones, haciendo pruebas unitarias, usando JUnit para evaluar patrones como observador, factoría abstracta y MVC. Estos patrones deben cumplir ciertas condiciones, como encapsulamiento y ausencia de dependencias ocultas, para ser efectivos en TDD. Se hizo una investigación del sistema automático de control de velocidad de un vehículo (SCACV). Los resultados indican que los patrones mejoran la calidad, pero algunos presentan limitaciones en términos de complejidad y mantenimiento.



## Bibliografía

Capel, M. I., Grimán, A. C., & Garví, E. Calidad Ágil: Patrones de Diseño en un contexto de Desarrollo Dirigido por Pruebas. : <https://n9.cl/6wk3a>

## Reflexión

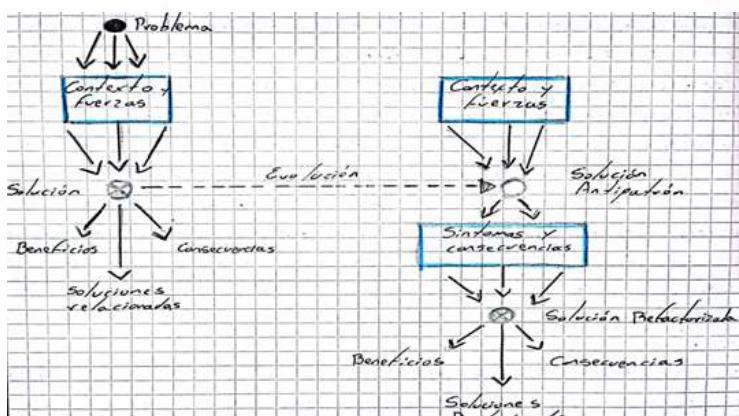
El artículo nos muestra los beneficios y desafíos de combinar patrones de diseño con TDD en el contexto ágil. La integración de patrones de diseño probado nos ayuda a estructurar el código de forma modular, facilitando su mantenimiento y reutilización. Igualmente, nos muestra que no todos los patrones se adaptan fácilmente al TDD, cheque algunos aumentan la complejidad del sistema y dificulta la creación de las pruebas unitarias, especialmente en las interfaces del usuario. Mostrándonos la importancia de seleccionar patrones de diseño con criterios de testabilidad en mente.

# Utilización de Antipatrones y Patrones en el Análisis de Software

El artículo explora el papel de los antipatrones y patrones en el desarrollo de software. Los antipatrones se definen como soluciones recurrentes, que no resuelven problemas, sino, que generan consecuencias negativas, mientras que los patrones representan buenas prácticas que se pueden aplicar en diferentes etapas del desarrollo. Los autores no muestran la importancia de identificar y evitar antipatrones para mejorar la calidad de software, al tiempo que nos sugieren que los patrones pueden acelerar la definición de modelos conceptuales y facilitar la validación de requerimientos.

## Reflexión

Reconocer los antipatrones no sólo permite a los desarrolladores evitar errores comunes, sino que también proporciona una valiosa lección sobre la importancia del conocimiento. Al mismo tiempo, la adopción de patrones como guías para la solución de problemas fomenta a la innovación y a la mejora continua en el proceso de desarrollo de software. Lo cual nos invita a reflexionar sobre nuestras decisiones y enfoques, lo cual promueve el aprendizaje y adaptación. Así que los antipatrones y los patrones son herramientas que conducen a la creación de software de alta calidad.



## Bibliografía

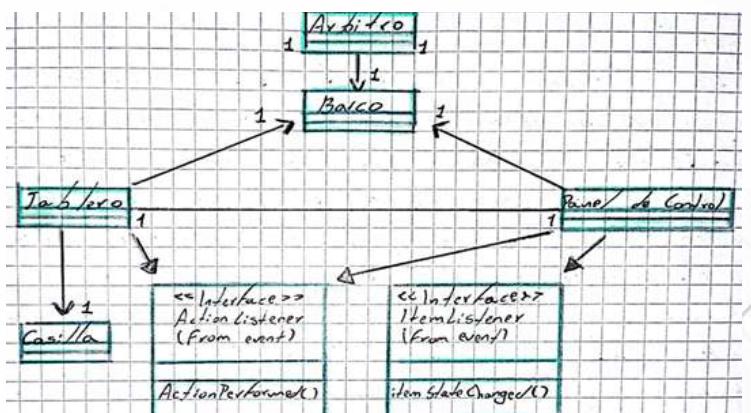
Garnero, A. B., & Horenstein, N. Utilización de antipatrones y patrones en el análisis de software. <https://n9.cl/ae8jq6>

# Aprendizaje Práctico de Patrones de Diseño en Asignaturas de Programación de Nivel III

El artículo nos presenta los resultados de una evaluación en la cual se aplicaron patrones de diseño a un curso de programación. A través de las encuestas que se realizaron, se pudo notar que estos no sólo comprenden la teoría, sino que también aprecian su utilidad práctica. Los resultados indican un alto porcentaje de estudiantes considera que aplicarán patrones a sus proyectos. Además, nos muestra los beneficios de utilizar patrones, la reducción de tiempo de desarrollo, mejora en la documentación de software y en su calidad.

## Reflexión

El artículo nos muestra la importancia de usar conceptos teóricos con práctica en la educación, más en áreas complejas como lo son la programación. El aplicar patrones de diseño demuestra que no sólo mejora la comprensión, sino que también nos prepara para enfrentar desafíos reales en el desarrollo de software. Nos demuestra también que la comunicación es crucial para un mayor aprendizaje. Más ahora donde la tecnología avanza rápidamente es importante adoptar métodos y herramientas de enseñanza donde sea notable la mejora continua del desarrollo de software.



## Bibliografía

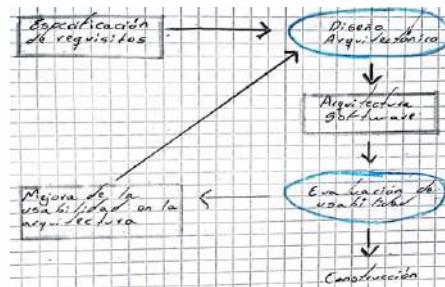
Marticorena, R., López, C., García Osorio, C. I., & Pardo, C. (2002). Aprendizaje práctico de patrones de diseño en asignaturas de programación de nivel III. <https://n9.cl/4cusk>

# Patrones de Usabilidad Temprana en el Modelo Conceptual

El artículo nos muestra la importancia de la usabilidad en el desarrollo de aplicaciones web como el artículo nos resalta como muchos sistemas fallan debido a deficiencias. Se nos menciona que la usabilidad es la que determina la satisfacción del usuario al interactuar con un sistema, por eso debe ser considerada desde las etapas iniciales del ciclo de vida del software. Nos proponen patrones de usabilidad que pueden ser incorporados en el modelo desde la fase de requisitos. Igualmente nos presentan patrones específicos que abordan distintos criterios de usabilidad, como la prevención de errores de entrada, etc.

## Bibliografía

Moreno, J. C., Marciszack, M. M., & Groppe, M. A. (2020). Patrones de Usabilidad Temprana en el Modelo Conceptual. AJEA, (5). <https://n9.cl/ewi5om>



## Reflexión

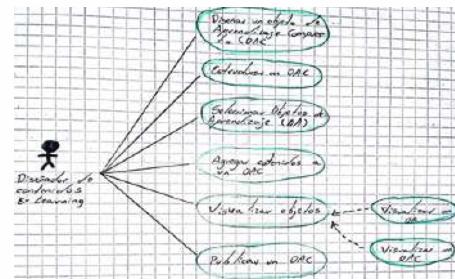
Incorporar patrones de usabilidad desde las etapas iniciales de nuestro desarrollo del software es algo valioso ya que ayuda a cómo se diseña y construye en aplicaciones. El establecer un patrón que guíe a los desarrolladores a la creación de interfaces más intuitivas y accesibles, asegura un desarrollo más centrado al usuario. Además, es importante que nosotros los desarrolladores estemos equipados e informados de las herramientas para que podamos implementar soluciones efectivas.

# Una Arquitectura de Software para la Integración de Objetos de Aprendizaje Basados en Servicios Web

El artículo nos presenta un modelo arquitectónico diseñado en el cual se facilita la integración de Learning Management Systems (LMS) y Repositories of Learning Objects (ROA). La arquitectura se compone de 5 vistas: funcional, estructural, de comportamiento, de implementación y de despliegue, cada una de estas aborda diferentes aspectos del sistema. La arquitectura presentada busca resolver problemas de interoperabilidad y la reutilización de objetos, para ofrecer una solución más práctica para el desarrollo.

## 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://n9.cl/4lhz>

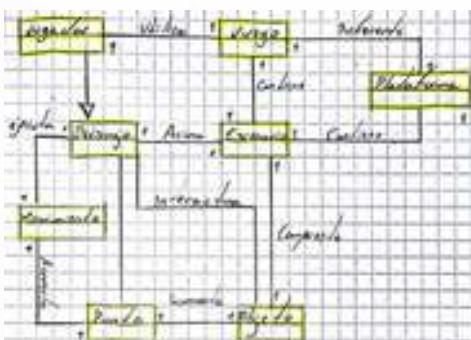


## Reflexión

Este nos resalta que una buena arquitectura bien definida en el ámbito de la educación, donde la integración de recursos de aprendizaje es crucial. Hoy la propuesta de que un modelo basado en servicios web no sólo aborda la necesidad de la interoperabilidad entre los diferentes sistemas, sino que también nos promueve a la reutilización de objetos de aprendizaje como también al ofrecernos un marco estructurado que facilita la búsqueda y el diseño de contenidos, nos permite crear materiales más efectivos y accesibles.

# Arquitectura de Software para el Desarrollo de Videojuegos sobre el Motor de Juegos Unity 3D

El artículo nos presenta una arquitectura de software diseñada específicamente para la creación de videojuegos utilizando Unity 3D. Esta arquitectura se organiza en capas como incluye un Game Manager, cada uno con funciones específicas, también incluye un State Machine, Sound Manager, Data Manager y Scene Manager. Se resalta la importancia de patrones de diseño como el singleton para asegurar la existencia de una única instancia de ciertos componentes. También discuten algunas restricciones arquitectónicas para garantizar la multiplataforma y la facilidad de actualización de los componentes.



## Bibliografía

- Paez, A. H., Falcón, J. D., & Cruz, A. A. P. (2018). Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D. Revista de I+D tecnológico, 14(1), 54-64 <https://n9.cl/dbt4a>

## Reflexión

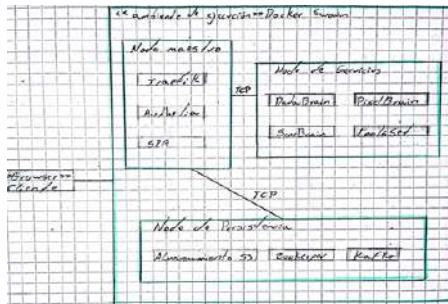
El artículo nos muestra la importancia de una arquitectura de software bien estructurada en el desarrollo de videojuegos, un campo el cual es bastante creativo. La arquitectura propuesta permite a los desarrolladores gestionar de manera más efectiva los diversos aspectos del juego desde la lógica hasta la interacción con el usuario. Al implementar patrones de diseño, hay creación de un código más limpio y mantenable, lo cual es crucial en este entorno donde los cambios son constantes, permitiendo a los desarrolladores innovar.

# Propuesta de Diseño de Arquitecturas Software para la Gestión, Análisis y Procesamiento de datos de Neurociencia

El artículo nos presenta un modelo arquitectónico destinado a abordar los desafíos de interoperabilidad, mantenimiento, escalabilidad y funcionalidad en el manejo de neurodatos. El artículo se centra en el diseño de una arquitectura que permite la adquisición automática y manual de datos en diversos formatos, así como la automatización del flujo de trabajo y los procesos de investigación. Ejemplar un método como el análisis documental y el modelado para desarrollar algo que facilite el procesamiento de datos y el aprendizaje automático, además que tiene un enfoque a microservicios, lo que permite estabilidad y eficiencia.

## Bibliografía

González, J. E. F., & García, C. A. O. Propuesta de diseño de arquitectura de software para la gestión, análisis y procesamiento de datos de neurociencia. <https://n9.cl/xz9s7>

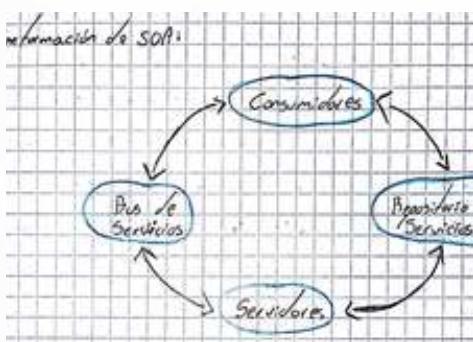


## Reflexión

El artículo nos destaca la importancia de una arquitectura de software bien diseñada en el campo de la ciencia, donde una buena gestión de datos es importante para la investigación. Al estar basado en microservicios no sólo nos permite la escalabilidad y mantenimiento, sino que también promueve la interoperabilidad entre distintos sistemas y formatos. También nos facilita la automatización de procesos en la integración de algoritmos de aprendizaje automático, lo cual nos da nuevas oportunidades para el análisis de datos complejos.

# Arquitectura de Software. Arquitectura Orientada a Servicios

El artículo nos muestra el concepto de Arquitectura Orientada a Servicios (SOA). Como un enfoque arquitectónico que nos permite la creación de sistemas de software flexibles y escalables. SOA se basa en que los servicios son componentes independientes, los cuales pueden comunicarse entre sí con interfaces bien definidas. El artículo también nos muestra los beneficios de implementar SOA, lo cual lleva a la mejora, la reducción de costos en mantenimiento y la capacidad de adaptarse rápidamente a los diferentes cambios.



## Bibliografía

- Martín, Y. E. (2012). Arquitectura de software. Arquitectura orientada a servicios. Serie Científica de la Universidad de las Ciencias Informáticas, 5(1), 1-10.  
<https://n9.cl/p7pho>

## Reflexión

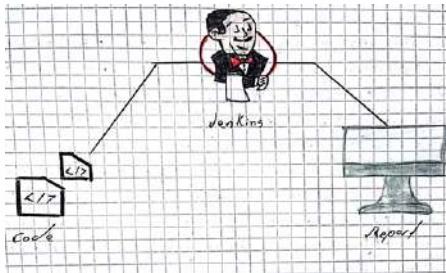
La arquitectura SOA representa un cambio significativo en la forma en la que diseñamos y gestionamos los sistemas de software actualmente. Ya que los servicios se comunican de manera independiente, SOA no sólo mejora la interoperabilidad, sino que también nos permite una mayor flexibilidad en el desarrollo de aplicaciones. La implementación de SOA en nuestros desarrollos puede reducir los costos y aumentar la eficiencia. Mostrándonos que la arquitectura orientada a servicios nos ayuda a mejorar en nuestros proyectos de software.

# Aprendiendo arquitectura de software a partir de proyectos de código abierto en GitHub

El artículo nos muestra cómo en la universidad de Sevilla, enseña una arquitectura de software a estudiantes mediante el análisis de proyectos de código abierto en GitHub. Se inspira en un método usado en la Delft University, esto les permite a los estudiantes aprender los conceptos arquitectónicos aplicándolos a proyectos reales. Los estudiantes trabajaron en grupos y utilizaron herramientas como SonarQube. Esto se evaluó con 258 estudiantes teniendo como resultado mejores calificaciones, lo que permitió desarrollar mejores habilidades y ayudó a un mayor conocimiento.

## Bibliografía

Sánchez, A. B., Parejo, J. A., Estrada-Torres, B., Márquez-Chamorro, A. E., del-Río-Ortega, A., & Segura, S. (2023). Aprendiendo arquitectura software a partir de proyectos de código abierto en GitHub. <https://n9.cl/x90ie>



## Reflexión

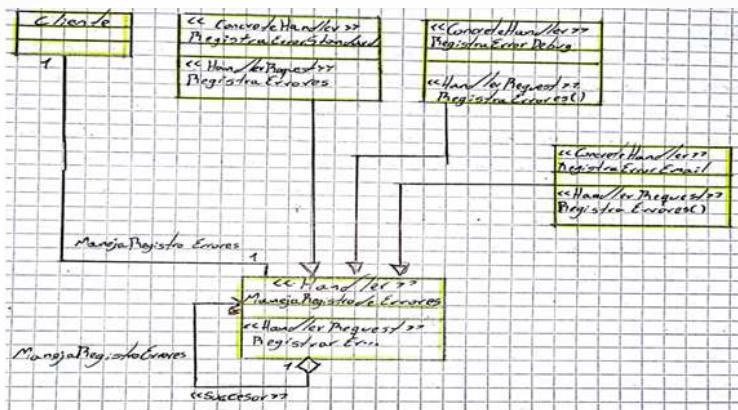
Utilizar ese tipo de prácticas en la arquitectura de software muestra una mejora en la educación. El aplicar este enfoque brinda una oportunidad de aplicar conceptos a proyectos reales, lo cual ayudó a la comprensión y retención de ese conocimiento. También el uso de herramientas mejora el desarrollo, lo cual nos prepara mejor para enfrentar entornos de desarrollo reales. Cabe mencionar que esto mejoró las habilidades colaborativas al trabajar en grupo siendo este un recurso valioso para fortalecer las competencias para el futuro.

# Una Propuesta de Implementación para Especificaciones de Patrones de Comportamiento

El artículo nos presenta una propuesta para implementar especificaciones de patrones de comportamiento (PDC) utilizando perfiles UML y restricciones OCL. En el proceso se definen 3 niveles de perfiles UML para representar patrones de diseño. Eso permite modular tantos aspectos, tanto estructurales como dinámicos, eso lo aplica diagrama de clase y secuencia en una herramienta de software. Eso permite a los desarrolladores reutilizar soluciones de diseño, lo cual hace todo más fácil en proyectos complejos, permitiendo un mejor desarrollo.

## Reflexión

Nos muestra la importancia de estructurar patrones de comportamiento en el desarrollo de software, más cuando se manejan sistemas complejos. La incorporación de UML y restricciones OCL, facilita a los desarrolladores el mantenimiento y las actualizaciones del sistema. Lo cual permite la reutilización, sino también la mejora en la colaboración y claridad del diseño. Por último, el artículo nos invita a reflexionar sobre las especificaciones y las herramientas adecuadas en la creación de arquitectura adaptable y escalables.



## Bibliografía

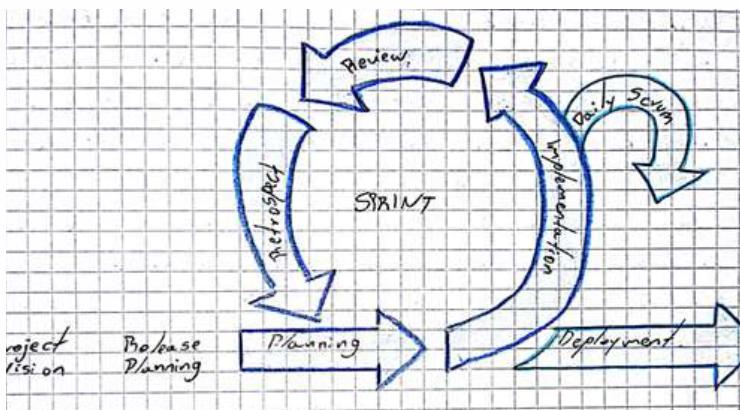
Cortez, A. A., & Naveda, C. A. Una propuesta de implementación para especificaciones de patrones de comportamiento. <https://n9.cl/x6wz7>

# Selección de Metodologías Ágiles e Integración de Arquitecturas de Software en el Desarrollo de Sistemas de Información

El artículo muestra las metodologías ágiles como ICONIX Y SCRUM. Estas metodologías ágiles priorizan la adaptabilidad y la respuesta rápida a los cambios. El artículo nos propone un modelo que permite integrar arquitecturas flexibles en proyectos, lo que permite mantener los principios de calidad y minimiza los riesgos y costos. Además, menciona que ICONIX Y SCRUM soportan beneficios para proyectos variables. Lo cual busca un modelo ágil, adaptable en cualquier momento y útil en la gestión de sistemas.

## Reflexión

El implementar una arquitectura de software representa un avance significativo para el desarrollo del proyecto. El Sprint 0 en SCRUM es innovador, ya que establece una arquitectura sólida sin afectar la flexibilidad. Esto es importante ya que garantiza que nuestro sistema pueda adaptarse a diferentes cambios sin que su calidad se vea afectada. Al combinarse con ICONIX, nos ofrece un camino viable para implementar soluciones ágiles sin que se pierda el control de toda la estructura del software, lo que permite maximizar la inversión.



## Bibliografía

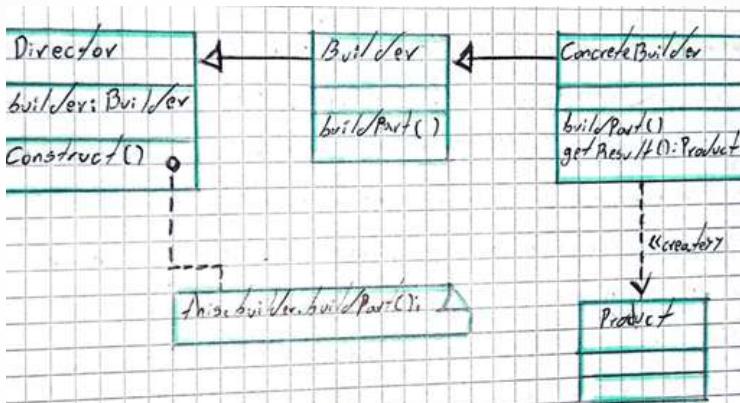
- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (2017, September). Selección de metodologías ágiles e integración de arquitecturas de software en el desarrollo de sistemas de información. In XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017, ITBA, Buenos Aires). <https://n9.cl/53lyw>

# CLASIFICACIÓN DE LOS PATRONES DE DISEÑO IDÓNEOS EN PROGRAMACIÓN ANDROID

Este artículo muestra la importancia de usar patrones de diseño en el desarrollo de software, especialmente en las aplicaciones móviles de Android. Ya que los patrones de diseño son soluciones reutilizables para problemas comunes en programación que ayudan a no crear código espagueti. Se mencionó la evolución de los patrones de diseño desde su introducción. Nos muestran las 3 categorías de patrones: creacionales, estructurales y de comportamiento, aunque en el artículo se centra especialmente en los 2 primeros, mencionando también que esto lleva a un desarrollo más eficiente.

## Reflexión

El implementar patrones de diseño en la programación no sólo es recomendado, sino una necesidad en el desarrollo de software actual. Usar patrones de diseño es una herramienta esencial para nosotros los desarrolladores. Esto nos proporciona un marco para poder resolver problemas, promover la reutilización de código y mejorar la calidad. Evitar el código espagueti es importante, ya que un código bien estructurado es mucho más fácil de entender y mantener, también disminuye los riesgos a errores y fallos que éste pueda presentar.



## Bibliografía

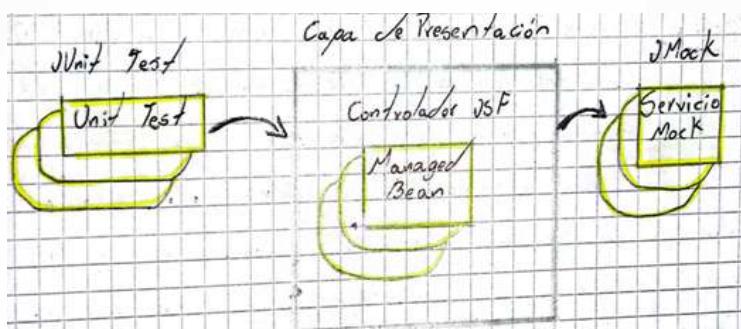
- Astorga, J. A. I., Tirado, J. L. O., Ramírez, R. U. R., Mendoza, J. C. L., & Garzón, A. P. (2019). Clasificación de los patrones de diseño idóneos en programación Android. Revista Digital de Tecnologías Informáticas y Sistemas, 3(1). <https://n9.cl/g02z3p>

# Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo

En este artículo se nos presenta un marco de trabajo para integrar arquitecturas de software en metodologías ágiles, utilizando Extreme Programming (XP) y guiado por pruebas TDD. Nos propone una arquitectura en capas, donde está la capa de presentación, negocio y persistencia, donde esta se testean mediante TDD para asegurar la calidad del software. También utilizan la herramienta JUnit, frameworks como JSF, Spring e Hibernate para poder implementar cada capa de la arquitectura. Esto ayuda a mantener la calidad del código.

## Reflexión

El combinar metodologías ágiles con una arquitectura sólida, proponiendo un marco de trabajo que emplea TDD dentro de la arquitectura, permite a todos los desarrolladores tener mejor control del sistema y una mejor respuesta al cambio de este. Lo cuál enseña como una estructura incapaz con pruebas unitarias, nos permite detectar errores mucho más fácil, lo cual nos facilita el mantenimiento. Usar las herramientas mencionadas igualmente facilita el desarrollo y nos permite crear un software más adaptable y escalable.

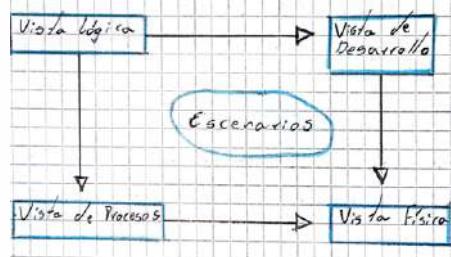


## Bibliografía

Vivas, H. L., Cambarieri, M. G., García Martínez, N., Muñoz Abbate, H., & Petroff, M. (2013). Un marco de trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo. <https://n9.cl/ebfpu>

# ARQUITECTURAS DE SOFTWARE PARA ENTORNOS MÓVILES

Este artículo nos presenta una arquitectura de software móvil (ASM) para aplicaciones Android colaborando con la universidad de Quindío y EtherealGF S.A.S. El objetivo que nos presentan es querer adaptar los principios de la arquitectura tradicional para entornos móviles, para sí mismo asegurar la calidad y la eficiencia de la aplicación. Esta se organizó en 3 tipos de capas: dependiente, independiente y conceptual, cada uno orientada a distintos módulos, como Google Maps y REST. También se usó el modelo de vistas cuatro uno para asegurar la calidad, rendimiento y la usabilidad.



## Reflexión

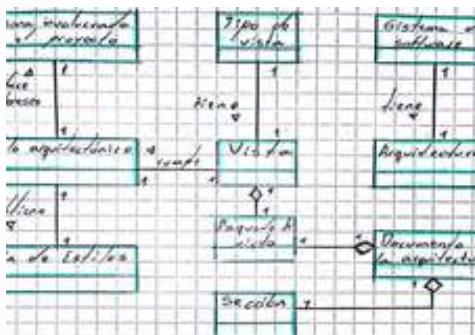
La arquitectura de software para las aplicaciones móviles son fundamentales y en este artículo nos muestran cómo adaptar arquitecturas tradicionales para la necesidad de las plataformas móviles, donde la calidad y eficiencia son importantes ya que hay dispositivos limitados en sus especificaciones técnicas. El artículo nos muestra la importancia de aplicar prácticas arquitectónicas sólidas para la creación de aplicaciones móviles, lo cual beneficia a la eficiencia en el desarrollo de estas aplicaciones móviles.

## Bibliografía

Granada, E. Z., Rodríguez, L. E. S., Montoya, C. E. G., & Uribe, C. A. C. (2014). Arquitecturas de software para entornos móviles. Revista de Investigaciones Universidad del Quindío, 25(1), 20-27. <https://n9.cl/1gp12>

# Documentando la Arquitectura de Software

El artículo nos habla de los principios básicos para documentar la arquitectura de software, nos presenta algunos conocidos, como el modelo 4 + 1 de Philippe Kruchten y el método de Robert L. Nord. Amo menciona en el uso de múltiples vistas como lo son: la lógica, proceso, desarrollo, física, estos representan varios aspectos del sistema. Nos muestran la propuesta del SEI (Vistas y más allá) y lo que es el estándar IEEE. Igual nos dan recomendaciones para optimizar la documentación, estilos arquitectónicos reutilizables, entre otros.



## Bibliografía

Gómez, O. Documentando la arquitectura de software.  
<https://n9.cl/wv1mw>

## Reflexión

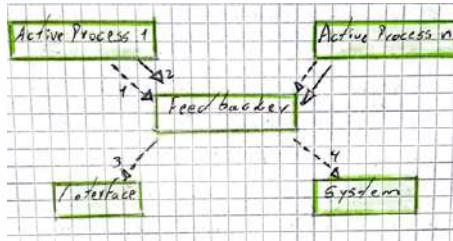
El artículo nos muestra cómo la buena documentación de la arquitectura de software es de gran ayuda para la comunicación, mantenimiento y actualizaciones de los sistemas. El tener diferentes enfoques y adaptar la documentación del software a las necesidades, facilita la toma de decisiones y asegura que todo cumpla con los requisitos de calidad. Lo cual nos permite a nosotros los desarrolladores no sólo cumplir con los requerimientos sino también facilitar la colaboración en los equipos de desarrollo de software.

# Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el Momento Arquitectónico

El artículo nos presenta como la usabilidad puede mejorar el sistema de software mediante la implementación de patrones arquitectónicos. Se presentan varios patrones específicos diseñados para resolver problemas de usabilidad, cada uno tiene una descripción, ventaja y beneficios en términos de la usabilidad. Un ejemplo es el patrón **Feedbacker**, este proporciona información al usuario sobre el estado del sistema, eso aumenta la satisfacción y reduce la carga de trabajo. El artículo nos muestra la importancia de la usabilidad desde el inicio del desarrollo.

## 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://n9.cl/6s49x>



## Reflexión

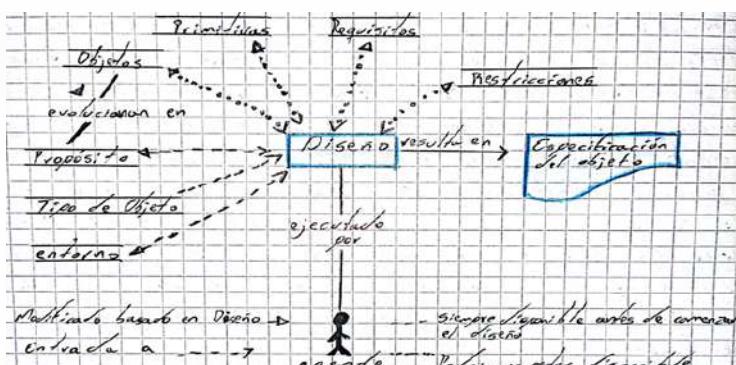
El integrar patrones de usabilidad en nuestro diseño arquitectónico es importante para crear un software mucho más intuitivo y mucho más eficiente, ya que, si abordamos las necesidades desde un inicio del desarrollo, se puede mejorar la experiencia al usuario y mejorar el rendimiento del software. Esto ayudaría en gran medida a la interacción del usuario con el sistema. Al usar la usabilidad en nuestros desarrollos de software, nos garantiza un software mucho más accesible y satisfactorio.

# Soporte a la Actividad de Diseño Basado en Patrones de Diseño

El artículo presenta la importancia de los patrones de diseño en la arquitectura de software, mostrándonos las especificaciones y el desarrollo de sistemas. Nos menciona que la descripción de un objeto y el proceso del diseño son importantes para generar soluciones efectivas y proponen la investigación para mejorar la especificación de patrones y sus aplicaciones en el desarrollo de software. En todo el documento nos plantean actividades de investigación y desarrollo que buscan avanzar en la interpretación y aplicación de patrones de diseño.

## Reflexión

El artículo muestra la importancia de patrones de diseño y herramientas para estructurar y optimizar bien nuestro desarrollo de software. Poniendo en práctica lo planteado en el artículo mejora la calidad del software al final y promueve una mejor colaboración de trabajo en equipo al momento de desarrollar, nos incentiva a la innovación en el diseño de software. Al avanzar rápidamente la tecnología, cuando la adaptación y la evolución de los patrones se vuelve importante para los desafíos a futuro.

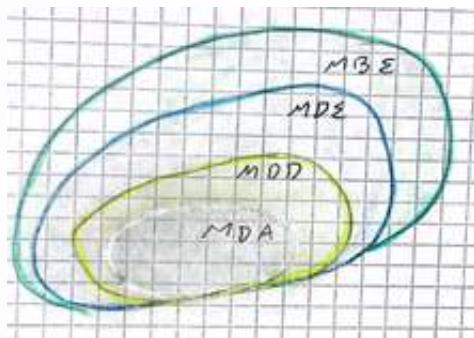


## Bibliografía

- Roqué Fourcade, L. E., & Arakaki, L. (2015, May). Soporte a la actividad de diseño basado en patrones de diseño. In XVII Workshop de Investigadores en Ciencias de la Computación (Salta, 2015). <https://n9.cl/3375h>

# **MODELO DE CASOS DE USO. SU APLICACIÓN EN UN ENTORNO DE DESARROLLO GLOBAL DE SOFTWARE**

El artículo analiza el modelo de casos de uso con una herramienta clave para definir requisitos funcionales de software. Aunque ampliamente utilizado en entornos de desarrollo co - localizados, su aplicación en contextos globales distribuidos presenta desafíos únicos. La investigación propone la plantilla CUPIDo, diseñada para documentar casos de uso de manera flexible, adaptándose a las necesidades de desarrollo colaborativo, ya sea sincrónico o asincrónico. También explora cómo estos pueden integrarse en procesos de Desarrollo Dirigido por Modelos (MDD), generando diagramas útiles para las etapas posteriores al desarrollo.



## **Bibliografía**

Lund, M. I., Chavez, S. B., Ormeño, E. G., Martin, A., & Matturro, G. MODELO DE CASOS DE USO. SU APLICACIÓN EN UN ENTORNO DE DESARROLLO GLOBAL DE SOFTWARE. <https://n9.cl/hr2oz>

## **Reflexión**

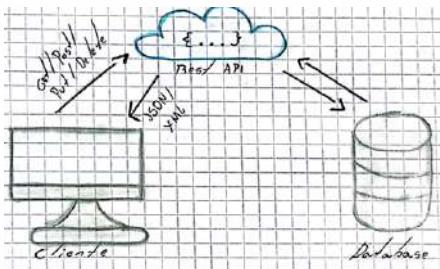
El artículo nos muestra cómo el desarrollo global transforma las prácticas tradicionales de ingeniería de software, exigiendo herramientas adaptadas a contextos multiculturales y distribuidos. La plantilla CUPIDo es un ejemplo de innovación que facilita la especificación de requisitos y la integración de modelos en procesos más amplios, como él MDD. La investigación refleja la importancia de la comunicación en equipos remotos y como el uso de tecnologías adecuadas pueden mitigar problemas. Esto refuerza la idea de que un software de calidad no sólo depende de la técnica, sino también de la capacidad de adaptación.

# Implementación de MVCC para la Generación de Aplicaciones Web en Tiempos de Entrega Reducidos

El artículo nos muestra una arquitectura innovadora para aplicaciones web llamada MVCC (Modelo - Vista- Controlador - Controlador), que optimiza tiempos de entrega al dividir la lógica de control en 2 controladores: uno en el cliente usando JavaScript con AJAX y otro en el servidor e implementando REST. Esto permite mejor compatibilidad con dispositivos móviles. La propuesta también incluye el uso de JSON para la comunicación entre controladores y el framework RedBeanPHP para facilitar el manejo de base de datos.

## Bibliografía

Rodríguez, I. Y. A., & Cortes, E. E. P. R. Implementación de MVCC para la generación de Aplicaciones Web en tiempos de entrega reducidos.  
<https://n9.cl/rctbb>



## Reflexión

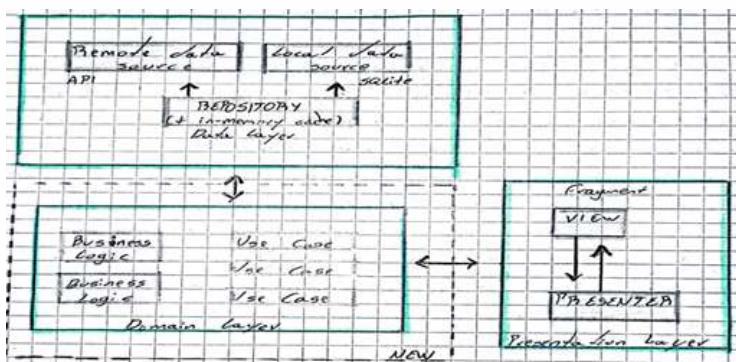
La arquitectura MVCC nos demuestra como la incorporación de tecnologías modernas puede transformar el desarrollo web, equilibrando la carga entre cliente y servidor, lo que permite aplicaciones más rápidas y versátiles. Actualizar framework y técnicas con una curva de aprendizaje baja, la propuesta resulta accesible para desarrolladores con diferentes niveles de experiencia, fomentando buenas prácticas de programación sin sacrificar la eficiencia. Esta metodología nos muestra la importancia de actualizar herramientas y paradigmas para satisfacer las necesidades actuales.

# Repercusión de Arquitectura Limpia y la Norma ISO/IEC 25010 en la mantenibilidad de Aplicativos Android

El artículo evalúa el impacto de la arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad del aplicativo móvil Educar Teacher, comparándolo con una aplicación convencional (CRM Distribución). La arquitectura limpia, propuesta por Robert Martin, organiza las aplicaciones en capas separadas por responsabilidades, promoviendo cohesión y bajo acoplamiento. La investigación emplea cuasi-experimental con métrica de analizabilidad, cambiabilidad, estabilidad y testeabilidad. Los resultados muestran mejoras significativas, lo cual evidencia que adoptar una arquitectura limpia mejora la capacidad de mantenimiento, etc.

## Reflexión

El artículo muestra la relevancia de la arquitectura limpia y las normas de calidad en el desarrollo de software, especialmente en aplicaciones móviles que requieren actualizaciones constantes. La mejora en las habilidades demuestra la eficiencia de dividir responsabilidades en capas, mientras que los incrementos modestos en otras métricas reflejan desafíos en la implementación completa de buenas prácticas. Este enfoque fomenta la creación de aplicaciones más duraderas y escalables, aunque existe un compromiso inicial de aprendizaje. Esto garantiza su sostenibilidad, reduce costos y aumenta la satisfacción del cliente.



## Bibliografía

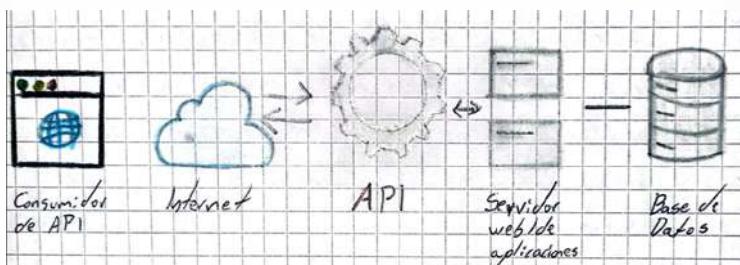
- Arias-Orezano, J. F., Reyna-Barreto, B. D., & Mamani-Apaza, G. (2021). Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android. *TecnoLógicas*, 24(52), 226-241. <https://n9.cl/s4de8>

# Metodología para la Enseñanza del Desarrollo de Software Modular

El artículo presenta una metodología educativa para la enseñanza del desarrollo de software modular, orientada a estudiantes de ingeniería en sistemas. La metodología integra el contexto como elemento clave para conectar teoría y práctica, fomentando habilidades de diseño y codificación escalables. A través de un enfoque iterativo, los estudiantes desarrollan una API web, siguiendo etapas como levantamiento de requisitos, diseño conceptual, implementación y pruebas. Se destacan conceptos claves como la programación orientada a objetos, patrones de diseño y el uso de herramientas como Swagger y DocAsCode.

## Reflexión

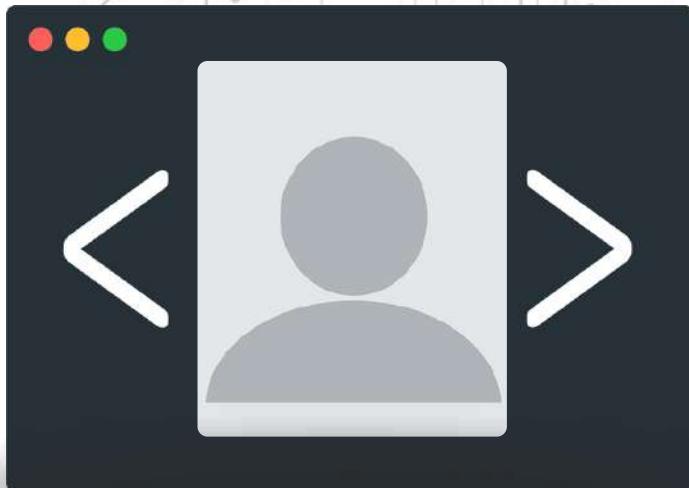
El artículo refleja como una metodología bien estructurada puede cerrar problemas en la formación de desarrolladores, mostrando que integrar el contexto y herramientas adecuadas facilita el aprendizaje. El énfasis en prácticas modernas, como código limpio y documentación integrada, prepara a los estudiantes para enfrentar desafíos reales. Ese enfoque resalta la importancia de enseñar no sólo técnicas de codificación, sino también habilidades críticas de diseño, validación y documentación, asegurando un buen futuro profesional.



## Bibliografía

Guevara, W. J. T. Metodología para la enseñanza del Desarrollo de Software Modular. <https://n9.cl/7hu2ba>

# Dylan Santiago Narváez Pinto

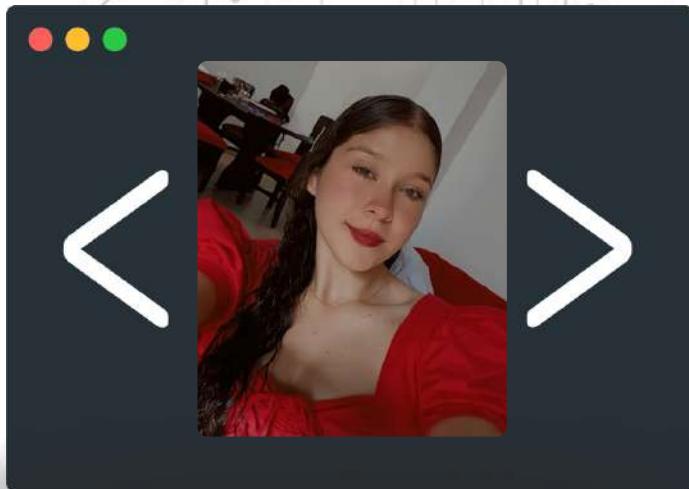


Tengo una gran pasión por el derecho y me encuentro en el proceso de formación profesional. En mis tiempos libres, disfruto mucho de la lectura y explorar la naturaleza, lo que me permite mantener una conexión con mi entorno mientras amplío mi comprensión del mundo.

Me gusta jugar al voleibol, una actividad que combina la competencia y el trabajo en equipo, lo cual refleja mi carácter en el ámbito profesional. A lo largo de mi formación en derecho, he aprendido a desarrollar habilidades analíticas y de resolución de problemas, y aplico estas competencias en mis estudios y en la vida diaria.

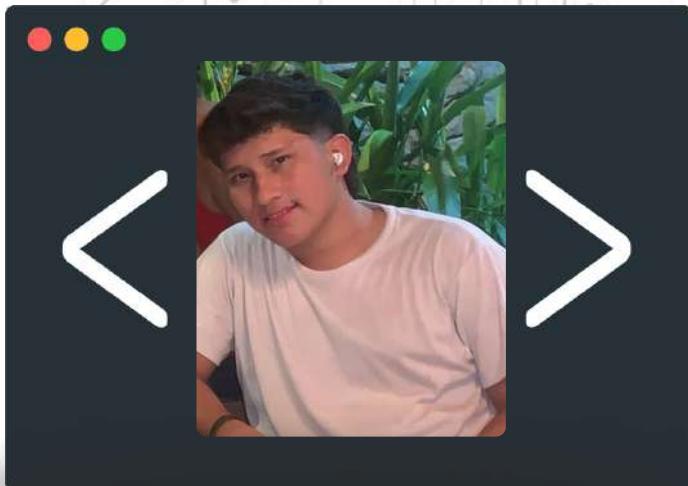
Mi objetivo es seguir creciendo tanto profesional como personalmente, manteniendo un equilibrio entre mi carrera y mis pasiones.

# Isabella Córdoba Gutiérrez



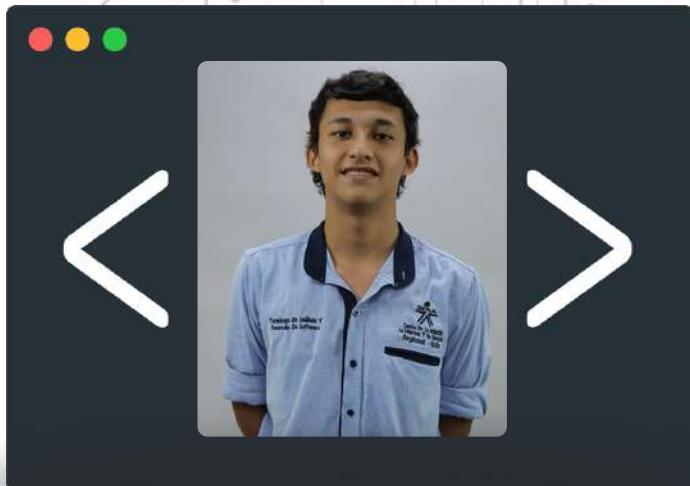
Mi nombre es Isabela Córdoba Gutiérrez, tengo 18 años y vivo en Campoalegre, Huila. Estudié en el colegio Eugenio Ferro Falla (Municipal), donde además de terminar mi bachillerato, hice un técnico en Diseño e Integración de Multimedia. Esa experiencia me ayudó a descubrir lo mucho que me gusta combinar creatividad y tecnología. En mi tiempo libre me encanta escuchar música, cocinar y pintar. Pasar tiempo con mi familia es algo que valoro muchísimo, porque ellos son mi mayor apoyo y alegría. Siempre estoy buscando crecer como persona y ser mejor cada día, no solo para mí, sino también para los demás. Ayudar a quienes me rodean me motiva un montón, porque creo que todos podemos aportar algo positivo al mundo. Con el corazón lleno de sueños y ganas de aprender, mi meta es construir un futuro que me haga sentir orgullosa, rodeada de cosas que amo y personas que me inspiran.

# Iván Andrés Murcia Epia



Mi nombre es Iván Andrés Murcia, nací el 17 de octubre de 2005 en la ciudad de Bogotá, y actualmente tengo 19 años. Resido en Neiva, Huila, junto a mis padres y mis dos hermanos. Finalicé mis estudios de bachillerato en 2022, donde obtuve un bachillerato técnico y un certificado en diseño e integración de multimedia por el Sena. Estos logros han sido fundamentales para mi desarrollo personal y académico. En la actualidad, estoy cursando dos programas simultáneos: un tecnólogo en Análisis y Desarrollo de Software y un técnico en Investigación Judicial. La tecnología me fascina porque transforma ideas en realidades palpables; me permite desarrollar soluciones innovadoras que pueden mejorar la vida de las personas. Por otro lado, la criminalística despierta mi interés por entender el comportamiento humano y las técnicas que ayudan a resolver crímenes. Ambas áreas son esenciales para mí, ya que combinan el pensamiento analítico y la investigación, habilidades que considero cruciales para enfrentar los retos del futuro.

# John Sebastián Penna Arias



Tengo 19 años y soy una persona apasionada por el fútbol, el voleibol y las actividades al aire libre. Disfruto mantenerme activo, tanto física como mentalmente, y encuentro en el deporte una forma de equilibrio y motivación.

Actualmente, estoy terminando el programa de Tecnólogo en Análisis y Desarrollo de Software, después de haber culminado mi Técnico en Programación de Software en 2022. La programación me ha permitido descubrir mi interés por resolver problemas y crear soluciones innovadoras.

Me considero una persona dinámica, responsable y con un fuerte deseo de superación. Me gusta trabajar en equipo, pero también soy autónomo y organizado cuando se trata de cumplir mis metas individuales. Estoy comprometido con aprender constantemente y aportar lo mejor de mí en cada proyecto.

# Patrones de Diseño Creacionales - Singleton

El patrón Singleton es uno de los patrones de diseño creacionales que asegura la existencia de una única instancia de una clase en todo el ciclo de vida de una aplicación. Este patrón es particularmente útil en aplicaciones donde es necesario que solo exista una instancia de una clase específica, por ejemplo, en clases de configuración o de manejo de recursos compartidos como conexiones a bases de datos.

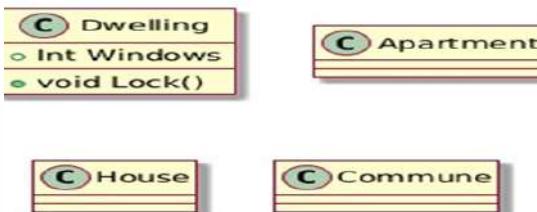
En el patrón Singleton, la creación de una instancia está controlada y el acceso a esta instancia se maneja mediante un método de acceso estático, comúnmente llamado `getInstance`. Esto permite que otras partes de la aplicación accedan a la instancia única sin necesidad de crear múltiples instancias.

## Reflexión

El patrón Singleton refleja una necesidad común en diseño de software: el control sobre la creación de instancias de una clase. Esta necesidad surge especialmente en sistemas donde múltiples instancias podrían resultar en incoherencias o en un uso ineficiente de recursos.

Sin embargo, el Singleton debe utilizarse con moderación, ya que su naturaleza de estado compartido puede introducir dependencias no deseadas en el código. Esto dificulta la separación de responsabilidades y hace que el código sea menos modular y menos flexible ante cambios.

Classes - Class Diagram



## Bibliografía

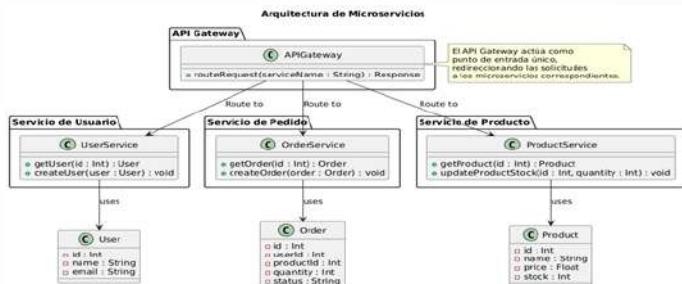
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.  
Freeman, E., & Freeman, E. (2004). Head First Design Patterns. O'Reilly Media.  
Bloch, J. (2008). Effective Java. Addison-Wesley.

# Arquitectura de Microservicios

La arquitectura de microservicios se refiere a una estructura modular que divide aplicaciones complejas en componentes independientes llamados microservicios. Cada microservicio tiene una funcionalidad específica y puede ser desarrollado, desplegado y escalado de manera autónoma. Esta arquitectura facilita el uso de diferentes tecnologías para cada servicio y permite el mantenimiento y la actualización sin afectar a los demás componentes. En comparación con las arquitecturas monolíticas, los microservicios mejoran la escalabilidad, resiliencia y permiten una mejor asignación de recursos. Al permitir el despliegue independiente de cada servicio, las organizaciones pueden reducir costos operativos y aumentar la eficiencia operativa.

## Reflexión

La adopción de la arquitectura de microservicios representa una evolución significativa en la forma en que las aplicaciones son construidas y gestionadas. Esta metodología no solo mejora la eficiencia técnica, sino que también promueve una organización más ágil, donde cada equipo puede ser responsable de una parte del sistema. Sin embargo, también impone desafíos como la gestión de la comunicación entre servicios y la complejidad operativa. En mi experiencia, la clave para aprovechar al máximo los microservicios es entender bien el dominio del negocio y aplicar los principios adecuados para estructurar los servicios.



## Bibliografía

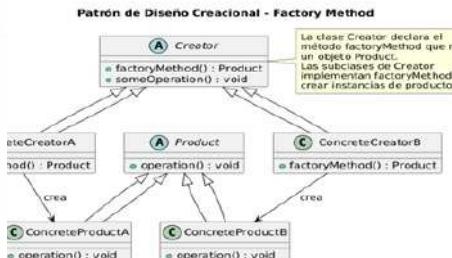
"Arquitectura de microservicios: ¿Qué es y cuáles son sus ventajas?"  
Valtx.pe

"Arquitectura de microservicios: ejemplos y conceptos claves"  
Pragma.co.

# Patrón de Diseño - Factory Method

El patrón de diseño Factory Method es un patrón creacional que permite a una clase delegar la creación de objetos a sus subclases. En lugar de crear objetos de manera directa dentro de una clase, se define un método de fábrica que es implementado por las subclases para generar los objetos adecuados. Este patrón es útil cuando se necesita crear instancias de diferentes tipos de objetos, pero se quiere mantener el código desacoplado y flexible ante cambios futuros.

El patrón está compuesto por una **\*interfaz o clase abstracta\*** (el creador), que define un método para crear objetos, y las **\*subclases concretas\*** que implementan este método. Esto permite que el proceso de creación de objetos se delegue y se puedan añadir nuevas subclases sin modificar el código existente.



## Reflexión

El patrón Factory Method resalta la importancia de desacoplar la creación de objetos del resto del sistema. Esto permite que el código sea más mantenible y flexible, ya que, si bien el cliente solicita un objeto, no necesita conocer los detalles de cómo se crea. En lugar de modificar todo el sistema cada vez que se agregan nuevos tipos de objetos, solo es necesario extender las clases creadoras, lo que mejora la escalabilidad del código.

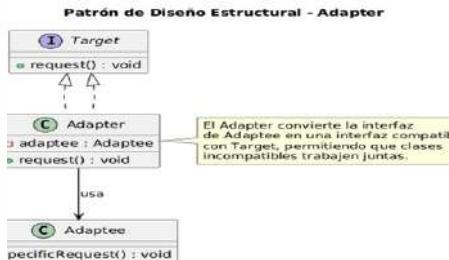
## Bibliografía

"Patrón de diseño Factory Method" DevExpert  
"Factory Method: Creación de objetos" Dev.to.

# Patrón de Diseño Adapter

El patrón de diseño Adapter es un patrón estructural que permite que dos interfaces incompatibles se conecten mediante un adaptador, facilitando la interacción entre ellas sin modificar el código original. Este patrón se utiliza cuando se necesita integrar una nueva clase con una interfaz diferente, o cuando se desea reutilizar una clase sin alterar su implementación. En lugar de modificar las clases existentes, el patrón introduce una capa adaptadora que convierte las interfaces y permite que trabajen juntas.

El patrón Adapter tiene diversas aplicaciones en situaciones donde se requieren conversiones entre distintos formatos, como al transformar datos de XML a JSON o integrar bibliotecas con diferentes interfaces. Esto permite la flexibilidad de incorporar nuevas tecnologías o componentes sin alterar el sistema ya existente, lo que a su vez fomenta la reutilización de código.



## Reflexión

Nos enseña la importancia de mantener nuestros sistemas flexibles y escalables, evitando cambios innecesarios en el código existente. En lugar de realizar ajustes directos a las clases, que podrían causar problemas de compatibilidad en el futuro, el adaptador proporciona una solución elegante y mantenible. La capacidad de integrar distintos sistemas y tecnologías sin alterar el núcleo de nuestro código es fundamental para crear aplicaciones robustas y fácilmente extensibles.

## Bibliografía

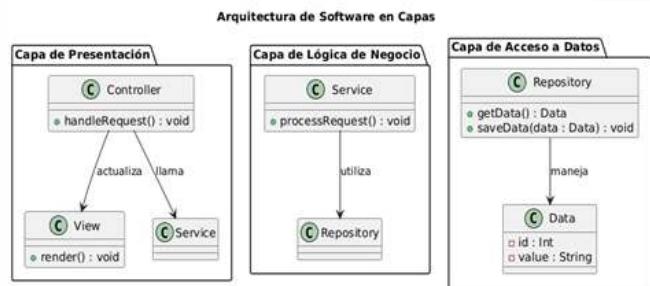
- Genbeta. "Patrones de diseño: Adapter". Recuperado de: [Genbeta] (<https://www.genbeta.com>)  
Refactorizando. "Patrón de diseño Adapter en Java". Recuperado de: [Refactorizando] (<https://www.refactorizando.com>).

# Arquitectura de software en capas

es un patrón estructural que divide una aplicación en diferentes capas, cada una encargada de una responsabilidad específica. Generalmente, se compone de tres capas: presentación, lógica de negocios y acceso a datos. La capa de presentación gestiona la interacción con el usuario, la capa de lógica de negocios aplica las reglas del sistema, y la capa de acceso a datos maneja la comunicación con la base de datos. Este patrón facilita la modularidad, mantenimiento y escalabilidad del sistema, ya que las capas funcionan de manera independiente, lo que permite modificar una capa sin afectar las demás.

## Reflexión

La arquitectura en capas es fundamental en el desarrollo de sistemas grandes y complejos. Su principal ventaja es la modularidad, lo que facilita la integración de nuevas funcionalidades y el mantenimiento del software. Sin embargo, es importante considerar que la implementación de este patrón puede generar una mayor complejidad en el diseño, además de un aumento en el consumo de recursos debido a la necesidad de comunicación entre capas. A pesar de estos desafíos, la arquitectura en capas sigue siendo una elección popular debido a su capacidad para organizar y separar las responsabilidades, lo que hace que el sistema sea más escalable y fácil de mantener.



## Bibliografía

LatamTech, "Qué es la arquitectura en capas: descubre sus ventajas y ejemplos" [Latam Tech](<https://latamtech-sac.com>)  
FourWeekMBA, "Arquitectura en capas"

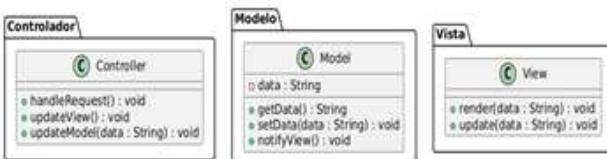
# Modelo-Vista-Controlador (MVC)

El patrón arquitectónico Modelo-Vista-Controlador (MVC) es ampliamente utilizado en el desarrollo de software, especialmente en aplicaciones web. Este patrón divide la aplicación en tres componentes principales: el Modelo, la Vista y el Controlador. El Modelo es responsable de la lógica de negocio y la gestión de datos. La Vista se encarga de la presentación, mostrando la interfaz de usuario, mientras que el Controlador actúa como intermediario, recibiendo las entradas del usuario y actualizando el Modelo y la Vista. Esta separación de responsabilidades permite que las aplicaciones sean más organizadas, fáciles de mantener y escalar, además de mejorar la flexibilidad al permitir modificaciones en uno de los componentes sin afectar a los demás.

## Reflexión

El patrón MVC no solo promueve la organización del código, sino que también resalta la importancia de la separación de responsabilidades en el desarrollo de software. Al dividir una aplicación en tres componentes, los desarrolladores pueden centrarse en aspectos específicos de la funcionalidad sin tener que preocuparse por los efectos secundarios en otras partes del sistema. Esta arquitectura también facilita la implementación de cambios sin afectar la estabilidad general del sistema, un aspecto crucial en el desarrollo de aplicaciones a gran escala. Además, la capacidad de trabajar de manera modular permite que los equipos colaboren de forma más eficiente, con un mejor enfoque en el diseño y la implementación de cada componente.

Patrón de Diseño Arquitectural - Modelo-Vista-Controlador (MVC)



## Bibliografía

"MVC (Modelo-Vista-Controlador): CodingOrNot. Recuperado (<https://codingornot.com>)

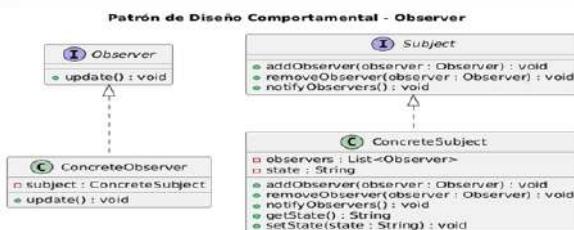
¿qué es y para qué sirve?"  
de [codingornot.com]

# Patrón Observer

El patrón Observer se utiliza cuando un cambio en el estado de un objeto (el sujeto) debe provocar una reacción en otros objetos (los observadores), sin necesidad de que el sujeto conozca cuántos observadores existen o quiénes son. Este patrón ayuda a disminuir el acoplamiento entre el sujeto y los observadores, haciendo que los sistemas sean más modulares y fáciles de mantener. Se aplica típicamente en aplicaciones de interfaces gráficas, sistemas de notificaciones, y sistemas de eventos como los usados en desarrollo web con el patrón publish-subscribe.

## Reflexión

El patrón Observer es fundamental en situaciones donde la reactividad y la sincronización entre componentes son cruciales, como en aplicaciones que manejan grandes volúmenes de eventos o actualizaciones, y donde la independencia de los módulos es clave. Sin embargo, su implementación debe hacerse cuidadosamente para evitar problemas como el rendimiento o las referencias inválidas cuando los observadores ya no son necesarios.

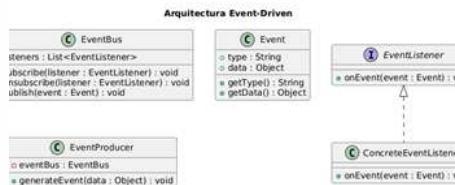


## Bibliografía

Wikipedia, "Observer (patrón de diseño)" enlace [WIKIPEDIA, LA ENCICLOPEDIA LIBRE](#)  
Java Design Patterns, "Observer Pattern" enlace [JAVA DESIGN PATTERNS](#)

# Arquitectura Event-Driven

La Arquitectura Event-Driven se basa en un modelo en el que los eventos son generados, transmitidos y procesados de forma asíncrona. En este enfoque, los componentes del sistema no están directamente acoplados entre sí. En lugar de eso, los eventos (cambios de estado significativos) se envían a través de un canal de eventos a los componentes que necesitan reaccionar ante ellos. Este tipo de arquitectura es ideal para sistemas distribuidos, donde la comunicación entre servicios y componentes se realiza mediante la publicación y suscripción a eventos. Es particularmente útil para aplicaciones de alta escalabilidad y rendimiento, como en sistemas de procesamiento de transacciones o eventos en tiempo real.



## Reflexión

La Arquitectura Event-Driven ofrece flexibilidad y escalabilidad, ya que permite que diferentes partes de una aplicación interactúen de manera desacoplada. Esto mejora la capacidad de respuesta del sistema a los eventos y facilita la integración de nuevos servicios sin necesidad de modificar los existentes. Sin embargo, se debe tener en cuenta que la gestión de eventos y la correcta sincronización entre servicios puede aumentar la complejidad del sistema.

## Bibliografía

- "Event-driven architecture." Wikipedia. Enlace  
Lascu, A. (2018). "Event-Driven Architecture: Concepts and Patterns." Journal of Software Engineering.

# Patrón Decorator

El Patrón Decorator es un patrón estructural que permite agregar comportamientos adicionales a un objeto de manera flexible y dinámica, sin modificar su código original. Se logra mediante la creación de "decoradores", que son clases que envuelven al objeto original y le agregan nuevas funcionalidades. Este patrón es útil cuando se necesitan nuevas capacidades sin alterar la estructura del objeto ni su comportamiento básico, y es más eficiente que la herencia en escenarios donde las funcionalidades deben ser combinadas de manera flexible.

## Bibliografía

"Decorator Pattern." Wikipedia. Enlace Freeman, E. (2004). "Head First Design Patterns." O'Reilly Media.



## Reflexión

El Patrón Decorator promueve la extensión de funcionalidades de manera modular y flexible. Es una excelente alternativa al uso de subclases cuando se requiere una combinación de comportamientos. Sin embargo, puede llevar a una complejidad adicional si no se maneja adecuadamente, especialmente cuando se añaden demasiados decoradores a un objeto.

# Arquitectura Serverless

La Arquitectura Serverless es un modelo en el que las aplicaciones se ejecutan sin la necesidad de administrar los servidores de forma explícita. Los desarrolladores escriben el código de la función, y el proveedor de la nube gestiona la infraestructura, escalabilidad y disponibilidad. Este enfoque permite a los equipos centrarse únicamente en el desarrollo de la lógica de negocio, lo que resulta en una mayor eficiencia y reducción de costos operativos. Sin embargo, las aplicaciones serverless pueden enfrentar limitaciones en términos de control sobre la infraestructura y posibles problemas de latencia.

## Reflexión

La arquitectura Serverless simplifica el proceso de desarrollo y despliegue, permitiendo que las aplicaciones escalen automáticamente según sea necesario. A pesar de sus ventajas, los desarrolladores deben ser conscientes de las posibles desventajas, como las limitaciones de ejecución, la latencia y la dependencia del proveedor de servicios en la nube.

### Arquitectura Serverless



## Bibliografía

"Serverless computing." Wikipedia. Enlace Roberts, M. (2018). "The Serverless Architecture." O'Reilly Media.

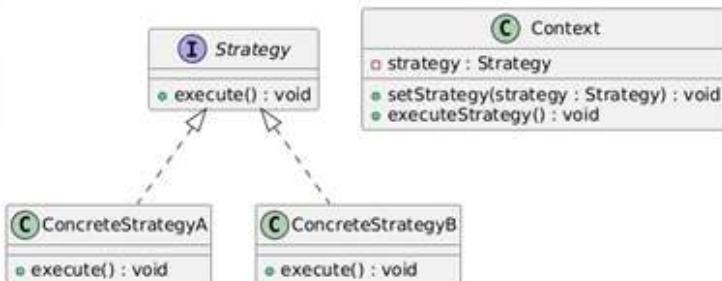
# Patrón Strategy

El Patrón Strategy es un patrón de diseño de comportamiento que permite cambiar el comportamiento de un objeto de manera dinámica. Este patrón define una familia de algoritmos y los encapsula, permitiendo que se seleccione el algoritmo a utilizar en tiempo de ejecución. Es útil cuando se requieren diferentes comportamientos que pueden variar, pero se desea evitar el uso de múltiples condicionales o de subclases para implementar cada variante del comportamiento.

## Reflexión

El Patrón Strategy proporciona flexibilidad al permitir cambiar el comportamiento de un objeto sin modificar su estructura. Este patrón facilita la extensibilidad y mantenimiento del código. Sin embargo, en sistemas complejos, la creación de numerosas estrategias podría incrementar la complejidad del sistema, por lo que debe aplicarse de manera prudente.

Patrón de Diseño Comportamental - Strategy



## Bibliografía

- "Strategy Pattern." Wikipedia. Enlace  
Freeman, E. (2004). "Head First Design Patterns." O'Reilly Media.

# Arquitectura Hexagonal

La Arquitectura Hexagonal, también conocida como "Ports and Adapters", es un estilo arquitectónico que busca aislar la lógica de negocio del resto del sistema, permitiendo que la aplicación interactúe con diferentes tipos de interfaces externas como bases de datos, interfaces de usuario o servicios externos. Esta arquitectura divide el sistema en un núcleo (lógica de negocio) y diversos adaptadores que permiten la comunicación con el mundo exterior. Esto mejora la mantenibilidad y prueba del sistema, ya que cada parte está desacoplada.

Arquitectura Hexagonal



## Reflexión

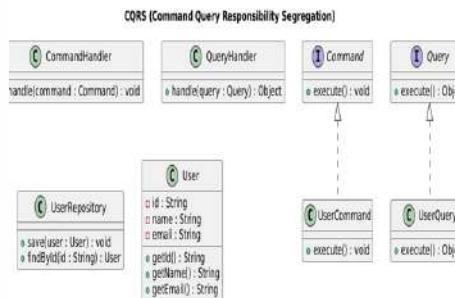
La Arquitectura Hexagonal promueve la independencia de la lógica de negocio, lo que facilita la integración con diferentes tecnologías y hace que las pruebas sean más sencillas. Sin embargo, puede ser más difícil de implementar en aplicaciones simples, donde el enfoque tradicional de monolitos podría ser más efectivo.

## Bibliografía

"Hexagonal Architecture." Wikipedia. Enlace Alistair Cockburn (2005). "Hexagonal Architecture." Object Mentor.

# CQRS (Command Query Responsibility Segregation)

CQRS es un patrón arquitectónico que separa las operaciones de lectura (queries) de las de escritura (commands) en un sistema. Esto permite optimizar cada tipo de operación y tratar de forma diferente las actualizaciones y las consultas. Es especialmente útil en sistemas con requisitos complejos de escalabilidad y rendimiento, como los que manejan grandes volúmenes de datos. CQRS también puede trabajar junto con Event Sourcing para mantener el estado del sistema.



## Reflexión

CQRS mejora la escalabilidad y la flexibilidad al separar las responsabilidades de lectura y escritura. Sin embargo, puede incrementar la complejidad del sistema, especialmente cuando se requiere mantener coherencia entre las dos partes y gestionar las actualizaciones de estado de forma eficiente.

## Bibliografía

"Command Query Responsibility Segregation." Wikipedia. Enlace Microsoft (2010). "CQRS: Command Query Responsibility Segregation." Microsoft Patterns & Practices.

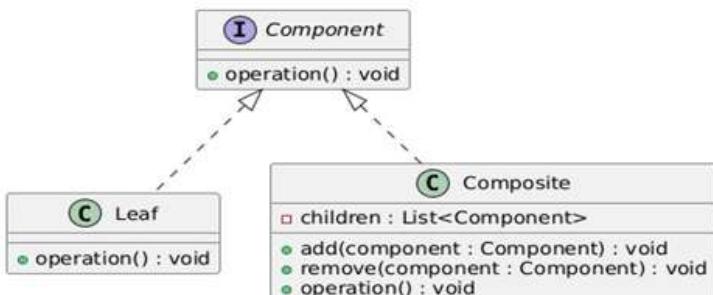
# Patrón Composite

El Patrón Composite es un patrón estructural que permite tratar objetos individuales y colecciones de objetos de manera uniforme. Este patrón crea una estructura de árbol donde los objetos pueden ser simples o compuestos por otros objetos, lo que facilita la manipulación de estructuras jerárquicas como menús, documentos o gráficos.

## Reflexión

El Patrón Composite facilita la gestión de estructuras jerárquicas complejas, permitiendo tratar tanto a los elementos simples como a los compuestos de la misma manera. No obstante, su implementación puede resultar más complicada cuando los objetos tienen una gran diversidad de comportamientos.

### Patrón de Diseño Estructural - Composite



## Bibliografía

"Composite Pattern." Wikipedia. Enlace Freeman, E. (2004). "Head First Design Patterns." O'Reilly Media.

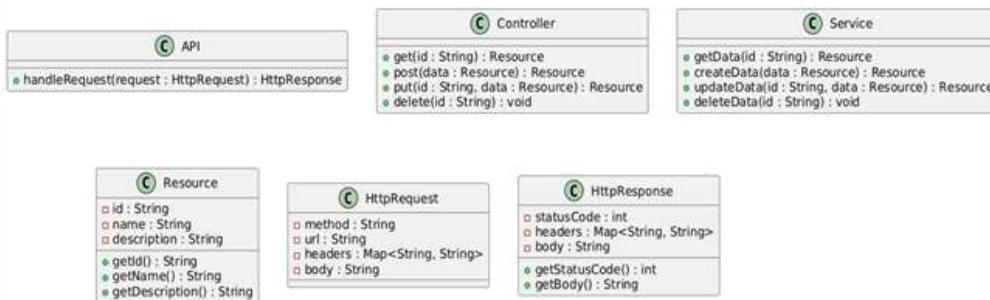
# Arquitectura RESTful

La Arquitectura RESTful es un estilo arquitectónico para el diseño de servicios web basado en el protocolo HTTP. Utiliza métodos estándar como GET, POST, PUT y DELETE para interactuar con los recursos, que son identificados por URLs. REST promueve una comunicación simple, escalable y eficiente, lo que lo convierte en la base de muchas aplicaciones web modernas.

## Reflexión

La Arquitectura RESTful es fácil de implementar y altamente eficiente para servicios web que requieren un alto rendimiento y escalabilidad. Sin embargo, es menos flexible que otros estilos arquitectónicos, como GraphQL, cuando se requieren consultas más complejas o cuando la relación entre los recursos es intrincada.

Arquitectura RESTful

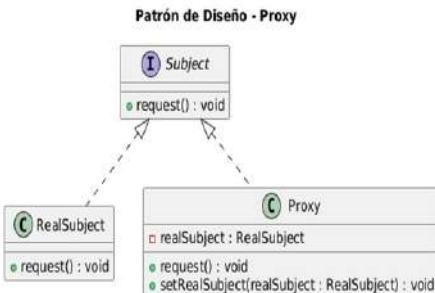


## Bibliografía

"Representational state transfer." Wikipedia. Enlace Fielding, R. (2000). "Architectural Styles and the Design of Network-based Software Architectures." University of California, Irvine.

# Patrón Proxy

El Patrón Proxy es un patrón estructural que proporciona un sustituto u objeto intermedio para otro objeto. El objetivo principal de este patrón es controlar el acceso a un objeto original, protegiéndolo de la interacción directa o añadiendo funcionalidades adicionales como la carga perezosa, el control de acceso, la autorización, entre otros. Se utiliza en situaciones donde es necesario intervenir en la comunicación entre un cliente y un objeto objetivo, como en los sistemas distribuidos o en el manejo de recursos costosos en términos de tiempo o memoria.



## Reflexión

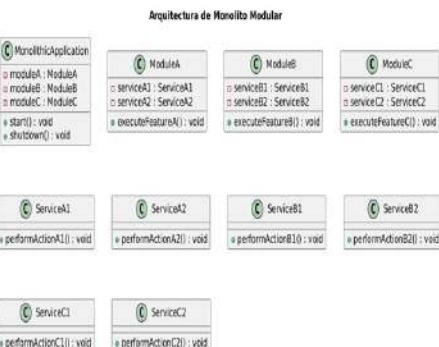
El Patrón Proxy facilita la gestión de recursos y mejora el control sobre los objetos. Al ofrecer una interfaz uniforme, permite encapsular la lógica de acceso y control sin modificar el comportamiento de los objetos originales. Sin embargo, puede introducir complejidad en la arquitectura al agregar una capa intermedia, lo que podría impactar en el rendimiento si no se maneja adecuadamente.

## Bibliografía

"Proxy Pattern." Wikipedia. Enlace Freeman, E. (2004). "Head First Design Patterns." O'Reilly Media.

# Arquitectura de Monolito Modular

La Arquitectura de Monolito Modular es un enfoque intermedio entre la arquitectura monolítica tradicional y la microarquitectura. En lugar de tener un único bloque de código monolítico, se organiza la aplicación en módulos independientes pero dentro de un mismo proyecto. Cada módulo tiene su propia lógica, pero todos comparten el mismo repositorio de código y ejecutan en una única base de datos. Esta arquitectura proporciona los beneficios de la modularidad y la reutilización sin las complejidades asociadas con la adopción de una arquitectura distribuida.



## Reflexión

El Monolito Modular permite un mejor mantenimiento y escalabilidad en comparación con un monolito tradicional, mientras conserva la simplicidad en términos de implementación. Sin embargo, la evolución hacia microservicios puede volverse inevitable a medida que el sistema crece en complejidad, por lo que este enfoque es adecuado para organizaciones que desean mantener una base de código compartida sin las complicaciones del escalado distribuido.

## Bibliografía

- Newman, S. (2015). Building Microservices. O'Reilly Media.  
"Monolithic Architecture." Wikipedia. Enlace

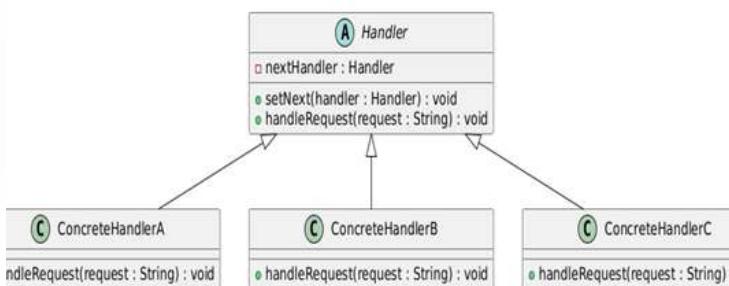
# Patrón Chain of Responsibility

El Patrón Chain of Responsibility es un patrón de comportamiento que permite pasar una solicitud a lo largo de una cadena de manejadores, donde cada uno decide si procesar la solicitud o pasárla al siguiente en la cadena. Este patrón elimina el acoplamiento entre el remitente de una solicitud y su receptor, y se utiliza comúnmente para manejar una serie de operaciones o eventos que pueden ser procesados por diferentes objetos sin necesidad de que cada uno conozca el otro.

## Reflexión

El Patrón Chain of Responsibility ofrece gran flexibilidad y escalabilidad, permitiendo que nuevas acciones sean incorporadas fácilmente a la cadena sin modificar los existentes. Sin embargo, su uso puede llevar a una mayor complejidad y a la falta de control sobre el orden de ejecución de las solicitudes, por lo que debe aplicarse con cuidado.

Patrón de Diseño - Chain of Responsibility



## Bibliografía

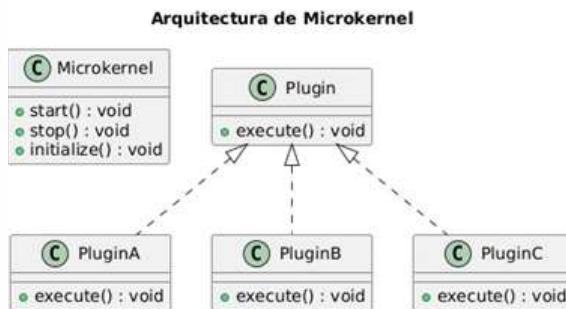
"Chain of Responsibility Pattern." Wikipedia. Enlace Freeman, E. (2004). Head First Design Patterns. O'Reilly Media.

# Arquitectura de Microkernel

La Arquitectura de Microkernel se basa en la creación de un núcleo mínimo del sistema que proporciona la funcionalidad básica, dejando la mayor parte de la lógica del sistema en módulos complementarios o plugins. Este enfoque es útil para aplicaciones que requieren alta personalización, ya que los módulos externos pueden ser fácilmente reemplazados o actualizados sin afectar el núcleo del sistema.

## Reflexión

El patrón Microkernel promueve la extensibilidad y la flexibilidad, permitiendo que se agreguen o eliminen funcionalidades sin alterar la arquitectura principal. No obstante, el desafío radica en la gestión de estos módulos y la seguridad del sistema, ya que el núcleo debe mantenerse lo más ligero y eficiente posible.

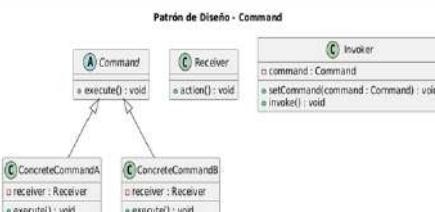


## Bibliografía

- "Microkernel architecture." Wikipedia. Enlace  
Shaw, M., & Clements, P. (2006). The Architecture of Open Source Applications. Lulu Press.

# Patrón Command

El Patrón Command es un patrón de comportamiento que convierte las solicitudes en objetos independientes, lo que permite parametrizar objetos con diferentes solicitudes, colas de peticiones, y registro de solicitudes. Este patrón permite separar el emisor de una acción del receptor de la misma, facilitando la implementación de deshacer y rehacer acciones en sistemas complejos.



## Reflexión

El Patrón Command aumenta la flexibilidad y la modularidad de un sistema, permitiendo que las solicitudes sean tratadas de manera independiente. Sin embargo, si no se gestionan adecuadamente, los objetos de comando pueden acumularse, volviendo el sistema más complejo de lo necesario.

## Bibliografía

"Command Pattern." Wikipedia.  
Enlace  
Freeman, E. (2004). Head First Design Patterns. O'Reilly Media.

# Arquitectura de Eventos y Sagas

La Arquitectura de Eventos y Sagas se enfoca en la gestión de transacciones distribuidas y procesos de larga duración. Utiliza eventos para transmitir el estado de una operación y Sagas para coordinar las distintas transacciones que componen un proceso complejo. Las sagas permiten que, en caso de error, se pueda retroceder o compensar las transacciones de manera que el sistema mantenga su consistencia.

## Reflexión

Esta arquitectura permite diseñar sistemas resilientes y escalables al gestionar transacciones distribuidas. Las sagas son fundamentales para garantizar la consistencia eventual en sistemas de microservicios, aunque su implementación puede ser compleja y requerir una adecuada gestión de errores.

Arquitectura de Eventos y Sagas



## Bibliografía

"Saga Pattern." Wikipedia. Enlace Kamil, A. (2016). Building Event-Driven Microservices. O'Reilly Media.

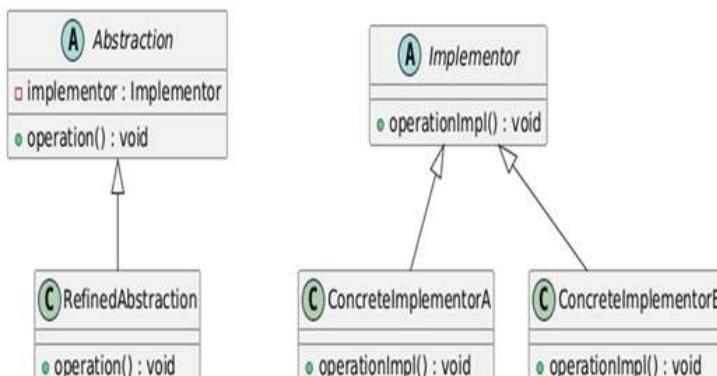
## Patrón Bridge

El Patrón Bridge es un patrón estructural que permite separar la abstracción de la implementación de un objeto. Mediante este patrón, se puede variar la implementación sin afectar la abstracción. Esto permite modificar la implementación de un sistema o añadir nuevas sin afectar el resto de la aplicación, lo cual es útil para aplicaciones que necesitan extensibilidad o actualización frecuente de sus componentes.

## Reflexión

El Patrón Bridge permite un diseño flexible y fácil de mantener, evitando la creación de subclases innecesarias. No obstante, este patrón puede aumentar la complejidad si no se aplica en el contexto adecuado.

### Patrón de Diseño - Bridge



## Bibliografía

"Bridge Pattern." Wikipedia. Enlace Freeman, E. (2004). Head First Design Patterns. O'Reilly Media.

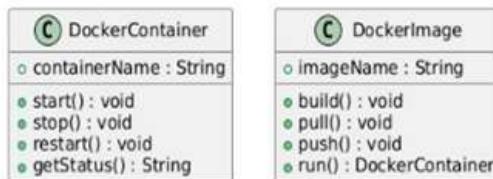
# Arquitectura de Contenedores y Docker

La Arquitectura de Contenedores se basa en la utilización de contenedores, como Docker, para empaquetar aplicaciones y sus dependencias en entornos aislados. Este enfoque permite que las aplicaciones se ejecuten de manera consistente en cualquier entorno sin importar las diferencias en la infraestructura subyacente. Docker, como una de las herramientas más populares, permite la creación, gestión y despliegue de contenedores.

## Reflexión

La arquitectura de contenedores, especialmente con Docker, ha transformado la manera en que se gestionan y despliegan aplicaciones. Ofrece portabilidad, escalabilidad y eficiencia, aunque implica desafíos en la gestión de múltiples contenedores y la orquestación de los mismos en entornos de gran escala.

Arquitectura de Contenedores y Docker

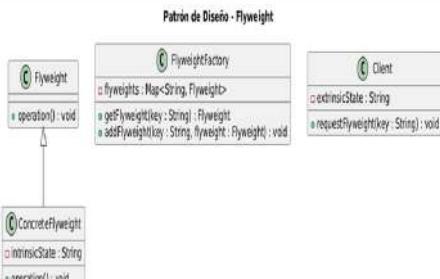


## Bibliografía

"Docker (software)." Wikipedia. Enlace Turnbull, J. (2014). The Docker Book. James Turnbull.

# Patrón Flyweight

El Patrón Flyweight es un patrón estructural que permite reducir el consumo de memoria al compartir objetos en lugar de crear instancias duplicadas. Este patrón es útil cuando se necesitan crear grandes cantidades de objetos similares, como en aplicaciones de gráficos o juegos, y ayuda a optimizar el uso de recursos al compartir partes comunes de los objetos.



## Reflexión

El Patrón Flyweight mejora la eficiencia en términos de memoria y procesamiento, permitiendo compartir recursos comunes. Sin embargo, su implementación puede ser compleja, ya que requiere un control adecuado sobre los objetos compartidos para evitar errores y garantizar el rendimiento.

## Bibliografía

"Flyweight Pattern." Wikipedia.  
Enlace  
Freeman, E. (2004). Head First Design Patterns. O'Reilly Media.

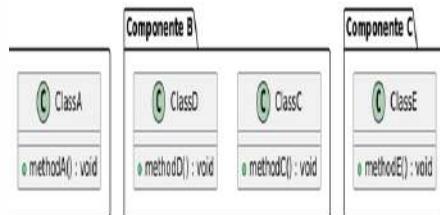
# Arquitectura de Espacios de Nombres

La Arquitectura de Espacios de Nombres organiza y estructura los elementos del sistema en jerarquías lógicas que permiten aislar y agrupar los componentes de manera eficiente. Al dividir el sistema en espacios de nombres, se facilita la gestión de dependencias, la escalabilidad y el mantenimiento del código. Este patrón asegura que los diferentes módulos del sistema no entren en conflicto entre sí, ya que permite que cada espacio tenga su propio contexto y no interfiera con otros, mejorando la modularidad del sistema. A medida que el sistema crece, la organización en espacios de nombres facilita tanto su extensión como la reutilización de componentes.

## Bibliografía

Aprender BIG DATA. Introducción a los Documentos de Arquitectura  
APRENDER BIG DATA

Arquitectura de Espacios de Nombres



## Reflexión

El uso de espacios de nombres es una estrategia clave para proyectos a gran escala, pues no solo organiza el código, sino que también permite gestionar la complejidad de manera más eficiente. Es crucial en sistemas distribuidos, donde múltiples módulos pueden necesitar interactuar, pero deben mantenerse independientes y funcionales. Esta organización permite escalar proyectos sin perder el control, especialmente cuando se integran nuevos componentes.

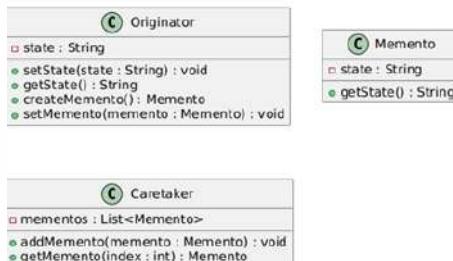
# Patrón Memento

El patrón Memento es utilizado para capturar y almacenar el estado interno de un objeto de manera que pueda ser restaurado más tarde sin exponer sus detalles internos. Este patrón es útil en situaciones donde los usuarios necesitan poder deshacer o revertir cambios realizados en el sistema, como en editores de texto o aplicaciones de juegos. El memento actúa como un "restaurador de estado" sin que el objeto original deba conocer los detalles de cómo se lleva a cabo este proceso. El patrón generalmente involucra tres componentes: el originador (el objeto cuyo estado se guarda), el memento (que almacena el estado), y el cuidador (que administra la restauración del estado cuando es necesario).

## Bibliografía

Diseño de Patrones de Software  
NATIVOS DIGITALES

Patrón de Diseño - Memento



## Reflexión

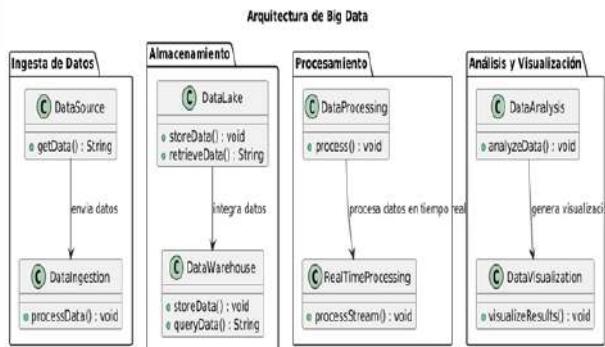
Implementar el patrón Memento otorga a las aplicaciones una gran flexibilidad al permitir que los usuarios reviertan cambios o se deshagan de acciones previas. Es particularmente valioso en aplicaciones donde las interacciones del usuario son frecuentes y donde un error podría ser costoso. Sin embargo, es necesario gestionar adecuadamente el almacenamiento de estos "estados", ya que guardar demasiados momentos podría aumentar el consumo de memoria y complicar la administración del sistema.

# Arquitectura de Big Data

La Arquitectura de Big Data se refiere a la infraestructura y a las tecnologías necesarias para almacenar, procesar y analizar grandes volúmenes de datos, tanto estructurados como no estructurados. Dado que los datos provienen de diversas fuentes (como sensores, redes sociales y transacciones), las soluciones de Big Data deben ser escalables, distribuidas y capaces de procesar información en tiempo real. Entre las tecnologías que forman parte de esta arquitectura se incluyen Hadoop, Spark y NoSQL, las cuales permiten manejar grandes cantidades de información a través de múltiples nodos, mejorando la eficiencia del procesamiento. Además, se debe tener en cuenta la necesidad de realizar análisis de datos a gran escala, utilizando herramientas como MapReduce para el procesamiento paralelo.

## Reflexión

La arquitectura de Big Data ha revolucionado la manera en que las empresas abordan el análisis de datos. Con la capacidad de procesar cantidades masivas de información, las organizaciones pueden extraer insights que anteriormente no eran posibles. Sin embargo, el manejo de Big Data también implica desafíos, como la administración de los costos de infraestructura, la seguridad de los datos y la necesidad de personal capacitado para operar estas complejas tecnologías. El desarrollo de una arquitectura adecuada es fundamental para aprovechar todo el potencial de Big Data sin caer en sobrecargas o desventajas operativas.



## Bibliografía

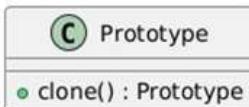
Introducción a los Documentos de Arquitectura  
APRENDER BIG DATA

# Patrón Prototype

El patrón Prototype es utilizado para crear nuevos objetos duplicando un prototipo de objeto ya existente, en lugar de crear uno nuevo desde cero. Este patrón es particularmente útil cuando los objetos que se van a crear son complejos o cuando la creación de nuevos objetos consume muchos recursos. En lugar de crear un nuevo objeto, el patrón permite "copiar" el estado de un objeto prototipo y realizar modificaciones según sea necesario. Este proceso es más rápido y eficiente, especialmente en sistemas que requieren generar muchos objetos similares con diferentes configuraciones.

## Reflexión

El patrón Prototype es una excelente solución para sistemas que necesitan generar objetos de forma eficiente, pero con ligeras variaciones. Ayuda a reducir el costo de creación de nuevos objetos, ya que evita la repetición de procesos costosos. Sin embargo, su uso debe ser medido, ya que depende de la disponibilidad de un prototipo adecuado, y puede generar una dependencia fuerte entre los objetos y el prototipo si no se implementa correctamente. A pesar de estos desafíos, el patrón Prototype sigue siendo muy valioso cuando se manejan sistemas que requieren una rápida creación y modificación de objetos.



## Bibliografía

Diseño de Patrones de Software  
NATIVOS DIGITALES

# Arquitectura de Capas Lógicas

La arquitectura de capas lógicas es una estructura que organiza un sistema de software en capas que interactúan entre sí, pero tienen responsabilidades claramente definidas. Las capas más comunes incluyen la capa de presentación, la capa de lógica de negocio y la capa de acceso a datos. Esta estructura promueve la separación de responsabilidades y facilita la mantenibilidad, ya que las modificaciones en una capa no afectan directamente a las otras. Además, cada capa puede ser sustituida o modificada sin afectar el funcionamiento general del sistema. Este enfoque permite que las aplicaciones sean más modulares, escalables y fáciles de probar.

## Reflexión

La arquitectura de capas lógicas es una de las más utilizadas en el diseño de software debido a su simplicidad y efectividad. Ofrece una clara separación de responsabilidades, lo que facilita el mantenimiento, la evolución y el escalado de sistemas. Sin embargo, también puede presentar desafíos de rendimiento, ya que la interacción entre capas puede introducir latencias si no se gestiona adecuadamente. A pesar de ello, su uso sigue siendo muy recomendable en sistemas grandes, donde la organización y la modularización son esenciales.



## Bibliografía

Introducción a la Arquitectura de Software  
APRENDER BIG DATA

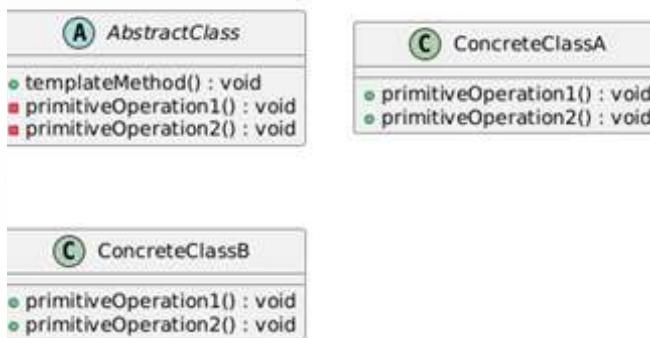
# Patrón Template Method

El patrón Template Method define el esqueleto de un algoritmo en un método base, dejando algunos pasos del proceso definidos por subclases. Este patrón permite que las subclases implementen ciertos pasos del algoritmo sin cambiar su estructura general. Es útil cuando se tiene un algoritmo común para varios casos, pero se desea que ciertos pasos varíen según el contexto. Por ejemplo, en un proceso de generación de reportes, el Template Method podría definir la estructura básica del reporte, mientras que cada subclase podría definir cómo se recopila la información o cómo se presenta.

## Reflexión

El patrón Template Method promueve la reutilización de código y mantiene la consistencia en la implementación de procesos similares, lo que mejora la eficiencia y la claridad. A pesar de su utilidad, es importante no sobrecargar el método base con demasiadas implementaciones específicas, ya que esto podría hacer que el código se vuelva rígido y difícil de mantener. Su uso es especialmente adecuado en sistemas donde el flujo de trabajo básico es constante, pero hay variaciones en los detalles específicos de cada caso.

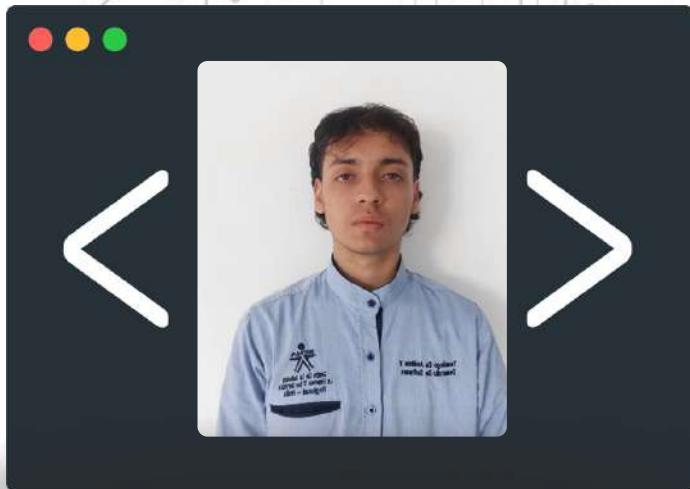
## Patrón de Diseño - Template Method



## Bibliografía

Diseño de Patrones de Software  
NATIVOS DIGITALES

# José Manuel Gasca Bonilla

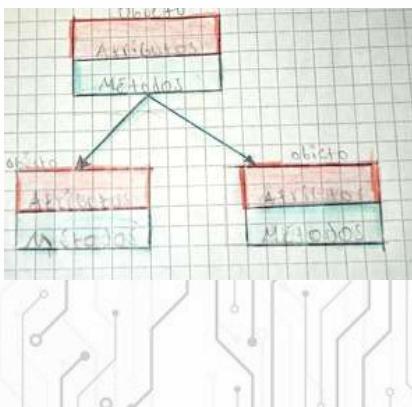


Soy Jose Manuel Gasca Bonilla, tengo 19 años y actualmente curso el programa de Tecnólogo en Análisis y Desarrollo de Software en el Sena. Me apasiona profundamente el mundo del desarrollo de software, especialmente el proceso de aprender a resolver problemas de manera creativa y eficiente.

En mi tiempo libre, me dedico a ampliar mis conocimientos en programación, nuevas tecnologías y metodologías ágiles, siempre buscando mantenerme actualizado en este campo tan dinámico. Mi objetivo principal es finalizar con éxito el tecnólogo para luego continuar mis estudios en una carrera de Ingeniería de Software, lo que me permitirá consolidar mi formación y ejercer profesionalmente como ingeniero. Aspiro a contribuir al desarrollo de soluciones innovadoras que impacten positivamente en la sociedad y a seguir creciendo tanto personal como profesionalmente en este fascinante ámbito.

# Lenguajes de Patrones de Arquitectura de Software Una Aproximación Al Estado del Arte

Este artículo aborda los lenguajes de patrones en la arquitectura de software, explorando su evolución, aplicaciones y contribuciones. Comienza con sus raíces en los patrones de diseño introducidos por Christopher Alexander, quien desarrolló un enfoque sistemático para resolver problemas comunes mediante patrones reutilizables. Posteriormente, se adaptó este concepto al desarrollo de software, permitiendo diseñar arquitecturas más eficientes y escalables. Los lenguajes de patrones no solo organizan patrones individuales, sino que también ofrecen soluciones integrales a problemas interrelacionados. Ejemplos como la gestión de identidades y las aplicaciones de negocio destacan su versatilidad en dominios específicos. Con base a esto, los lenguajes de patrones han transformado el diseño de software, facilitando procesos más estructurados y soluciones de alta calidad para problemas complejos.



## Reflexión

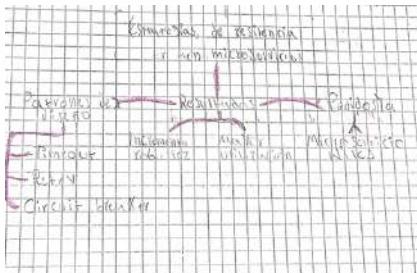
El artículo nos proporciona una visión detallada sobre los lenguajes de patrones en la arquitectura de software, explorando su origen, evolución y aplicaciones. Los lenguajes de patrones son herramientas poderosas para resolver problemas comunes en el diseño de sistemas, permitiendo soluciones reutilizables y eficientes. La idea de descomponer problemas complejos en subproblemas más manejables y luego aplicar patrones específicos para resolverlos es un enfoque práctico y adaptable a distintos contextos. Me pareció interesante la influencia de Christopher Alexander, quien inició este concepto desde la arquitectura física, y cómo su trabajo inspiró aplicaciones en software. También valoro cómo este artículo resalta que los lenguajes de patrones no solo son herramientas técnicas, sino también un puente entre la teoría y la práctica, ayudando a mejorar la calidad del software y reduciendo tiempos de desarrollo.

## 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.

# Implementando estrategias de resiliencia en una arquitectura basada en microservicios

El artículo analiza la implementación de estrategias de resiliencia en una arquitectura de microservicios, utilizando como caso de estudio el sistema de PedidosYa. Se enfoca en el microservicio Niles, encargado de proporcionar el menú de los restaurantes. El texto explica cómo los fallos en sistemas distribuidos pueden afectar el funcionamiento de toda la arquitectura y destaca la importancia de la resiliencia para mitigar estos problemas. Se describen tres patrones de diseño principales: Timeout, Retry y Circuit Breaker, explicando su funcionamiento y beneficios. Los autores realizaron experimentos aplicando estos patrones a Niles y midieron su impacto. Los resultados mostraron mejoras significativas en el rendimiento y la estabilidad del sistema, como la reducción de tiempos de respuesta y la disminución de errores. El artículo concluye que la implementación de estos patrones de resiliencia convirtió a Niles en uno de los componentes más robustos y utilizados en la arquitectura de PedidosYa.



## Reflexión

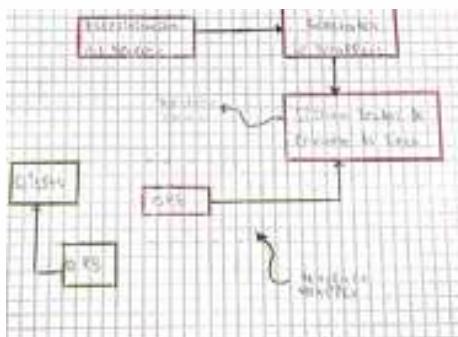
Después de leer este artículo sobre la implementación de estrategias de resiliencia en una arquitectura de microservicios, me quedo con varias reflexiones interesantes. Lo que más me llamó la atención es cómo estos patrones de diseño pueden marcar una gran diferencia en la estabilidad y rendimiento de sistemas distribuidos complejos. El caso de estudio de PedidosYa muestra claramente los beneficios tangibles de aplicar patrones como Timeout, Retry y Circuit Breaker. Me parece fascinante cómo algo aparentemente simple como establecer tiempos de espera adecuados puede mejorar drásticamente los tiempos de respuesta. También me impresiona cómo el patrón Retry logró reducir significativamente los errores, y cómo Circuit Breaker evitó saturar servicios degradados.

## Bibliografía

- Suárez, S. L., Rucci, E., Betran, V., & Montezanti, D. M. (2024). Implementando estrategias de resiliencia en una arquitectura basada en microservicios. In XXIX Congreso Argentino de Ciencias de la Computación (CACIC)(Luján, 9 al 12 de octubre de 2023).

# Evolución e Integración de Aplicaciones Legadas: Comenzar de Nuevo o Actualizar?

Este artículo analiza el desafío de actualizar e integrar sistemas de software antiguos (llamados "legados") con tecnologías modernas. El texto explica que muchas organizaciones aún dependen de estos sistemas viejos, que son difíciles de modificar y mantener. Se presentan varias opciones para abordar este problema, como reemplazar completamente el sistema, mantenerlo como está, o aplicar técnicas de "reingeniería" para modernizarlo gradualmente. El artículo se enfoca principalmente en la reingeniería, describiendo diferentes enfoques como el uso de "wrappers" para encapsular el sistema viejo, la conversión de características en componentes más pequeños, y la integración con servicios web.



## Reflexión

Después de leer este artículo sobre la evolución e integración de aplicaciones legadas, me quedo con varias reflexiones interesantes. El texto explora la problemática de los sistemas de software antiguos que siguen siendo vitales para muchas organizaciones, pero que se han vuelto difíciles de mantener y actualizar. Me llamó la atención cómo se presentan diferentes alternativas para abordar estos sistemas legados, desde reemplazarlos completamente hasta aplicar técnicas de reingeniería para modernizarlos gradualmente.

## Bibliografía

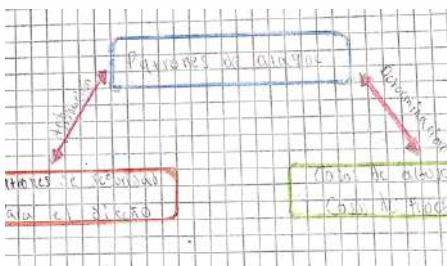
Pedraza-García, G. (2008). Evolución e integración de aplicaciones legadas: Comenzar de nuevo o actualizar?. In III Congreso Colombiano de Com-putación.

# Patrones de Ataque y de Seguridad como guía en el desarrollo de software

Este artículo explora el uso de patrones de seguridad y patrones de ataque como guías para desarrollar software más seguro. Se centra en la arquitectura de software al analizar cómo estos patrones pueden integrarse en las diferentes etapas del ciclo de desarrollo. Los patrones de seguridad ofrecen soluciones probadas a problemas de seguridad comunes, mientras que los patrones de ataque describen métodos utilizados para explotar vulnerabilidades. El texto destaca la importancia de catalogar y relacionar ambos tipos de patrones para facilitar su aplicación práctica. Propone un enfoque donde, al identificar posibles ataques, los desarrolladores puedan seleccionar patrones de seguridad apropiados para mitigarlos. El artículo también discute la necesidad de validar la efectividad de estas correspondencias entre patrones de ataque y seguridad. Finalmente, sugiere que este enfoque basado en patrones puede ayudar a integrar consideraciones de seguridad de manera más sistemática en la arquitectura y diseño de software, especialmente para desarrolladores no expertos en seguridad.

## Reflexión

Después de leer este artículo sobre patrones de ataque y patrones de seguridad en el desarrollo de software, me quedo con varias reflexiones interesantes. El texto explora cómo estos patrones pueden servir como guías para crear software más seguro desde las etapas iniciales del desarrollo. Me llamó la atención cómo los patrones de ataque describen las técnicas que usan los atacantes, mientras que los patrones de seguridad proponen soluciones probadas para mitigar esas amenazas.



## Bibliografía

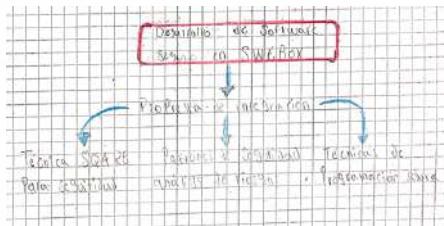
- Ramos, I. (2015). Patrones de Ataque y de Seguridad como guía en el desarrollo de software. In XVIII Concurso de Trabajos Estudiantiles (EST 2015)-JAIIO 44 (Rosario, 2015).

# Desarrollo de Software Seguro y su relación con el Cuerpo de Conocimiento para la Ingeniería de Software

Este artículo analiza el Cuerpo de Conocimiento de la Ingeniería de Software (SWEBOK) desde la perspectiva de la seguridad del software. Los autores argumentan que, aunque SWEBOK menciona aspectos de seguridad, no les da suficiente énfasis dado el creciente impacto de los problemas de seguridad en el desarrollo de software. El texto sugiere cómo integrar consideraciones de seguridad en cada área de conocimiento del SWEBOK, proponiendo herramientas, métodos y mejores prácticas específicas. Por ejemplo, en el área de Requisitos, sugieren usar técnicas como SQUARE para identificar requisitos de seguridad. En Diseño, recomiendan el uso de patrones de seguridad y análisis de riesgos. Para la Construcción, proponen técnicas de programación segura. El artículo enfatiza que la seguridad debe ser una preocupación transversal en todo el ciclo de vida del desarrollo, afectando la arquitectura del software desde las etapas iniciales hasta el mantenimiento. Los autores concluyen que es necesario actualizar el SWEBOK para reflejar la importancia crítica de la seguridad en la arquitectura y desarrollo de software moderno.

## Bibliografía

Vega, V. R., Gasca, G. P., Carrillo, J. D., & Tovar, E. Desarrollo de Software Seguro y su relación con el Cuerpo de Conocimiento para la Ingeniería de Software.

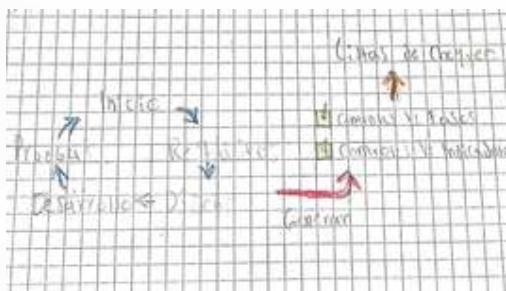


## Reflexión

Después de leer este artículo sobre el desarrollo de software seguro y su relación con el cuerpo de conocimiento de la ingeniería de software (SWEBOK), me quedo con varias reflexiones interesantes. El texto analiza cómo el SWEBOK aborda los aspectos de seguridad en el desarrollo de software y propone formas de fortalecerlos. Me llamó la atención cómo los autores sugieren incorporar prácticas, herramientas y métodos específicos de seguridad en las distintas áreas de conocimiento del SWEBOK. En cuanto a la arquitectura de software, el artículo destaca la importancia de considerar la seguridad desde las etapas tempranas del diseño arquitectónico. Se propone utilizar técnicas como el análisis de riesgos, patrones de seguridad y evaluación de atributos de calidad para crear arquitecturas inherentemente más seguras.

# **Modelo y herramienta software para la gestión de riesgos en el desarrollo de aplicaciones web soportado en el estándar ISO/IEC 27005**

Este artículo presenta un modelo para la gestión de riesgos en el desarrollo de aplicaciones web basado en el estándar ISO/IEC 27005. El modelo propone una arquitectura de software con tres perspectivas: conceptual, lógica y física. La perspectiva conceptual define 5 fases: creación del proyecto, parametrización, evaluación, identificación de riesgos y gestión de riesgos. La perspectiva lógica utiliza diagramas UML para modelar las clases y relaciones del sistema. La perspectiva física define la estructura de la base de datos. El modelo solo soporta ciclos de vida incrementales e iterativos, y está orientado a proyectos medianos y grandes.



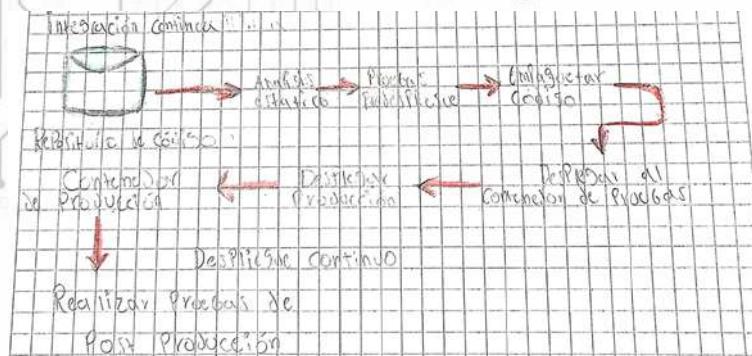
## **Bibliografía**

Angarita Cuéllar, A. J. (2012). Modelo y herramienta software para la gestión de riesgos en el desarrollo de aplicaciones web soportado en el estándar ISO/IEC 27005.

## **Reflexión**

Este artículo me hizo reflexionar sobre la importancia de considerar los riesgos desde el inicio al desarrollar aplicaciones web. Me pareció interesante cómo propone un modelo integral que combina estándares de seguridad con las fases típicas de desarrollo de software. La idea de crear una arquitectura en capas (conceptual, lógica y física) para organizar todo el proceso de gestión de riesgos me pareció muy inteligente. Me gustó que no solo se enfoca en identificar problemas, sino también en proponer planes de acción concretos. El uso de controles e indicadores le da mucha solidez a la propuesta. Aunque los detalles técnicos son complejos, entendí que busca crear un sistema flexible que pueda adaptarse a diferentes tipos de proyectos web.

# Método de automatización del despliegue continuo en la nube para la implementación de microservicios



## Reflexión

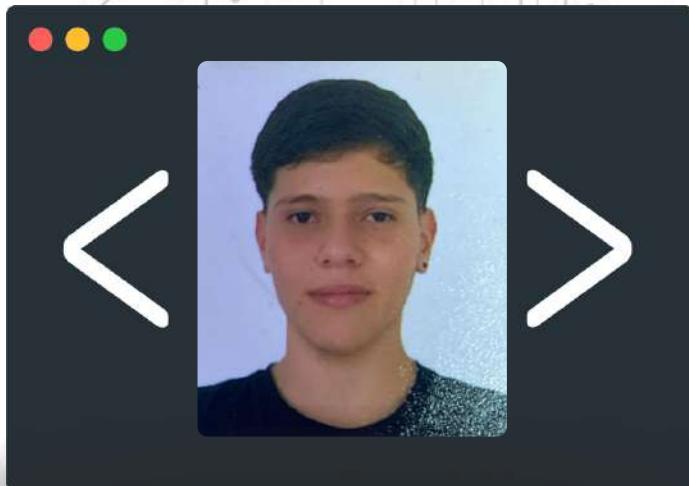
Este artículo analiza la importancia de la arquitectura de software basada en microservicios, un enfoque que permite dividir aplicaciones complejas en pequeños servicios independientes. La arquitectura de microservicios aborda desafíos como la escalabilidad y el despliegue continuo, facilitados por prácticas como DevOps, que integra desarrollo y operaciones para automatizar procesos. Destaca el uso de contenedores para garantizar que los servicios funcionen de manera consistente en entornos de prueba y producción. Además, se propone un método para automatizar el despliegue continuo, asegurando calidad mediante pruebas estáticas, unitarias y funcionales, todo integrado en una pipeline. Este enfoque demuestra cómo la arquitectura de software evoluciona para responder a las necesidades modernas de agilidad y eficiencia en el desarrollo y mantenimiento de aplicaciones.

Este artículo me hizo reflexionar sobre cómo la arquitectura de software se adapta a las necesidades actuales de las empresas, especialmente con los microservicios. Es interesante ver cómo esta metodología divide aplicaciones grandes en pequeñas partes independientes, lo que permite una flexibilidad increíble. Lo que más me llamó la atención es cómo DevOps y la automatización del despliegue continuo transforman los procesos de desarrollo y despliegue en algo más ágil y seguro.

## Bibliografía

Vera-Rivera, F. H. (2018). Método de automatización del despliegue continuo en la nube para la implementación de microservicios. In Proc. 21st Conf. Iberoamericana Ingeniería Softw.(CIBSE)(pp. 597-604).

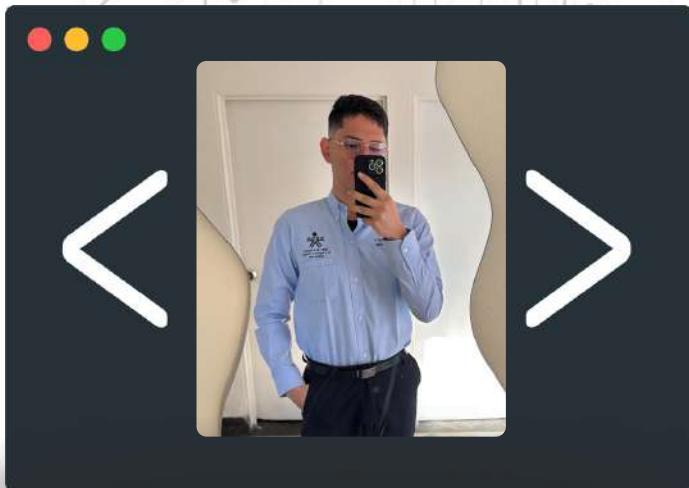
# Juan Pablo Betancourt Gómez



Tengo 19 años y soy una persona apasionada por el fútbol y el deporte en general, lo que me ha ayudado a desarrollar una mentalidad disciplinada y competitiva. Además de mi amor por el deporte, siento un gran interés por la programación, una pasión que me ha motivado a adquirir habilidades técnicas y a construir mi camino en el ámbito tecnológico.

Actualmente, estoy terminando mis estudios como Tecnólogo en Análisis y Desarrollo de Software, donde he aprendido las bases para desarrollar aplicaciones y sistemas. Mi objetivo es combinar mi creatividad y mi enfoque lógico para contribuir al mundo de la tecnología y explorar nuevas oportunidades en el desarrollo de software.

# Julian David Fierro Casanova



¡Hola! Me llamo Julián David Fierro Casanova, tengo 19 años. Desde siempre, me ha gustado mantenerme activo y disfrutar de diferentes actividades deportivas, especialmente el voleibol y la natación, que son mis pasatiempos favoritos. Creo que el deporte no solo es una forma de ejercitarse el cuerpo, sino también de desarrollar habilidades como el trabajo en equipo, la disciplina.

Me considero una persona responsable y juiciosa, siempre buscando mejorar y afrontar los retos con una actitud positiva. Me gusta aprender cosas nuevas y, sobre todo, aplicar lo que aprendo para crecer como persona.

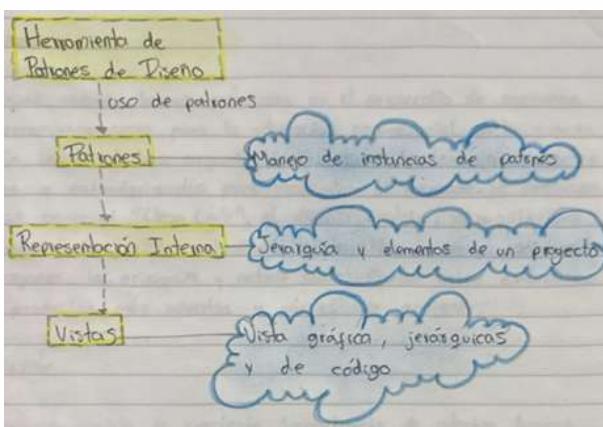
Además valoro mucho los momentos que paso con mi familia y amigos, ya que son las personas que siempre me apoyan y me motivan a seguir adelante. En el futuro, espero seguir desarrollándome y ser capaz de contribuir en proyectos que me apasionan. Creo firmemente que con esfuerzo y dedicación, se puede lograr cualquier cosa que uno se proponga.

# Una Arquitectura para una Herramienta de Patrones de Diseño

El artículo Una Arquitectura para una Herramienta de Patrones de Diseño describe una arquitectura para integrar patrones de diseño en herramientas de desarrollo de software orientado a objetos. Esta arquitectura permite la creación, manipulación y gestión de patrones como elementos de modelado básicos, lo cual optimiza la eficiencia y la reusabilidad del software. Utiliza patrones como Composite, Command y Observer para facilitar la interacción entre la interfaz de usuario y la lógica interna, ofreciendo al usuario vistas gráficas, jerárquicas y de código, y manteniendo la consistencia de los patrones en el sistema.

## Reflexión

El artículo Una Arquitectura para una Herramienta de Patrones de Diseño describe una arquitectura para integrar patrones de diseño en herramientas de desarrollo de software orientado a objetos. Esta arquitectura permite la creación, manipulación y gestión de patrones como elementos de modelado básicos, lo cual optimiza la eficiencia y la reusabilidad del software. Utiliza patrones como Composite, Command y Observer para facilitar la interacción entre la interfaz de usuario y la lógica interna, ofreciendo al usuario vistas gráficas, jerárquicas y de código, y manteniendo la consistencia de los patrones en el sistema.



## Bibliografía

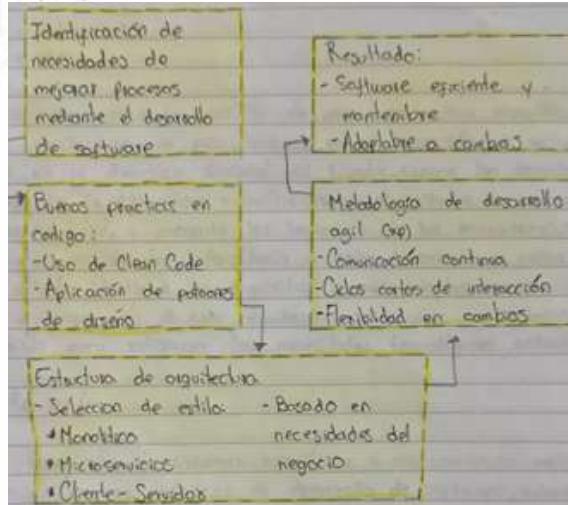
Molina, J. (1999). Una arquitectura para una herramienta de patrones de diseño. <https://n9.cl/drm696>

# Buenas prácticas en la construcción de software

El artículo aborda las buenas prácticas en el desarrollo de software como herramienta clave para la adaptación empresarial en la cuarta revolución industrial. Las organizaciones deben implementar sistemas complejos y estandarizados para responder al mercado. Se destacan prácticas como el "Clean Code", el diseño modular, la arquitectura basada en eventos, y la metodología XP, enfatizando en su contribución para mejorar la eficiencia y reducir costos. Aplicar estos estándares asegura productos más robustos y fáciles de mantener.

## Bibliografía

Prieto, C. A., & Madrid, D. A. (2022). Buenas prácticas en la construcción de software. Revista TIA: <https://n9.cl/k81iu>

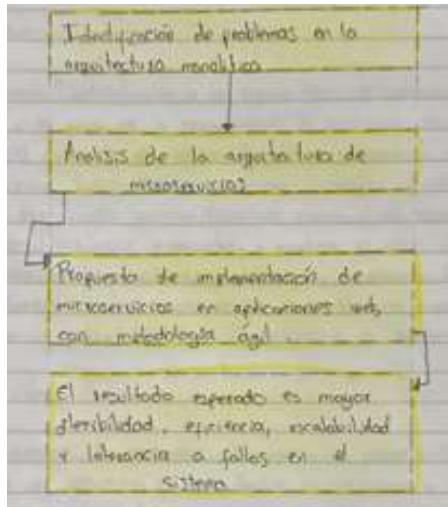


## Reflexión

Este artículo refleja la creciente importancia de adoptar buenas prácticas en el desarrollo de software, considerando tanto la calidad del producto final como la eficiencia en los equipos de trabajo. Implementar prácticas como el "Clean Code" y el uso de patrones arquitectónicos no solo beneficia la mantenibilidad del software, sino que también fomenta una cultura organizacional orientada a la excelencia. Este enfoque permite responder rápidamente a los cambios del mercado y cumplir con estándares internacionales, convirtiendo a las organizaciones en actores competitivos en la era digital.

# Arquitectura de Software basada en Micro-servicios para desarrollo de aplicaciones web

El artículo analiza la transición de una arquitectura monolítica a una arquitectura de micro-servicios en el desarrollo de aplicaciones web en la Asamblea Nacional del Ecuador. Expone los desafíos que presenta la estructura monolítica, como dificultades en mantenimiento y escalabilidad, y presenta los beneficios de los micro-servicios, que incluyen modularidad, independencia y mejor tolerancia a fallos. Además, el estudio identifica los requisitos y metodologías necesarios para la implementación de esta arquitectura y propone una estructura flexible para satisfacer las necesidades tecnológicas actuales.



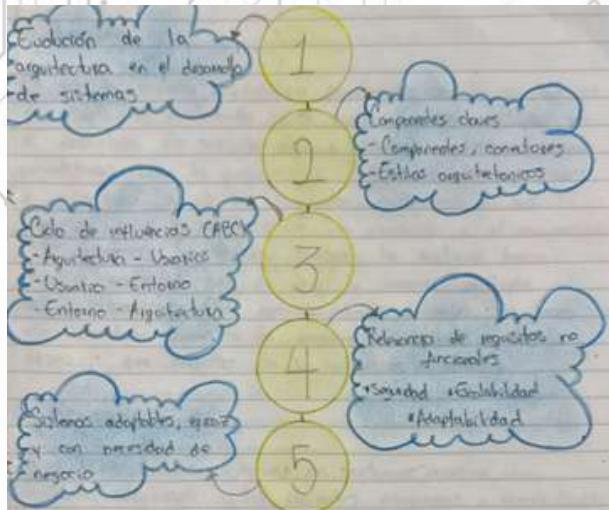
## Bibliografía

López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web. Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017, San José, Costa Rica. <https://n9.cl/0tbddk>

## Reflexión

La transición de sistemas monolíticos a micro-servicios refleja una evolución en las prácticas de desarrollo de software, adaptándose a un entorno tecnológico dinámico. Implementar micro-servicios no solo promueve la independencia de los módulos, sino también la eficiencia en el despliegue y mantenimiento de aplicaciones críticas. Sin embargo, este enfoque plantea desafíos adicionales en seguridad y coordinación. Este artículo muestra cómo la adopción de micro-servicios puede mejorar la flexibilidad organizacional, siempre que se implementen metodologías y herramientas adecuadas para garantizar la integridad y eficiencia del sistema.

# Introducción a la Arquitectura de Software



## Reflexión

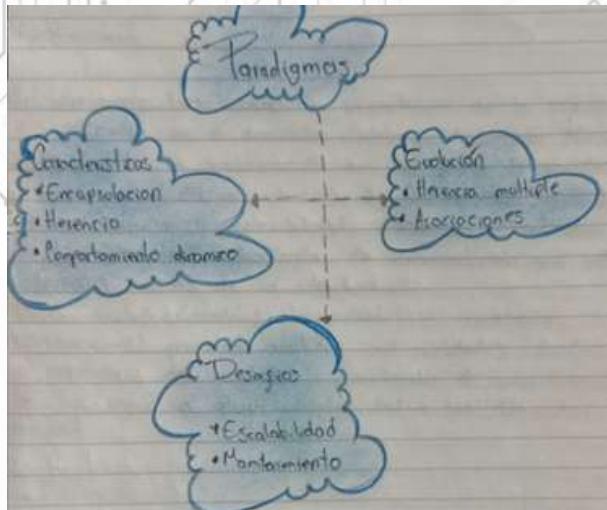
La arquitectura de software desempeña un papel crítico en la ingeniería de sistemas, ya que define la estructura y relaciones de sus componentes. Este enfoque estructural no solo optimiza el rendimiento técnico, sino que también responde a las necesidades y limitaciones del entorno de negocio. La obra de Cristián subraya la importancia de anticipar requisitos no funcionales desde el inicio, como la seguridad o la escalabilidad, los cuales influyen profundamente en la adaptabilidad futura. Así, la arquitectura de software se posiciona como una disciplina esencial para el desarrollo sostenible y adaptable de sistemas.

El documento "Introducción a la Arquitectura de Software" explica la evolución y el papel de la arquitectura de software en el ciclo de vida del desarrollo de sistemas. Desde su surgimiento en los años 90, esta disciplina busca resolver problemas complejos al establecer un nivel de abstracción superior. Describe componentes, conectores y estilos arquitectónicos esenciales, y explora el ciclo de influencias (ABC) entre la arquitectura, sus usuarios y su entorno. La obra analiza también la relevancia de los requisitos no funcionales para garantizar la adaptabilidad y la eficacia del sistema.

## Bibliografía

Cristiá, M. (2007). Introducción a la Arquitectura de Software. Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Obtenido de <https://n9.cl/vwg2l>

# Paradigmas en la construcción de software



El artículo explora diferentes paradigmas en la construcción de software, destacando cómo cada uno ha influido en el desarrollo de metodologías y enfoques modernos. Examina paradigmas como la programación basada en objetos, clases y orientada a objetos avanzados, cada uno con sus principios, como la encapsulación, la herencia y el comportamiento dinámico. Se destacan también la evolución hacia técnicas avanzadas que incluyen la herencia múltiple y las asociaciones. Este análisis permite entender las bases que han dado forma a la programación actual y los desafíos que persisten en su aplicación efectiva.

## Reflexión

La evolución de los paradigmas en software muestra una constante adaptación a las crecientes demandas y complejidades del desarrollo. La transición de la programación estructurada a la orientada a objetos resalta la importancia de conceptos como el modularidad y la reutilización. Reflexionar sobre estos paradigmas permite valorar cómo han facilitado la creación de sistemas complejos, y cómo los desafíos actuales impulsan una innovación continua. Esta adaptación constante es crucial para enfrentar los problemas de escalabilidad y mantenimiento que surgen con la tecnología moderna.

## Bibliografía

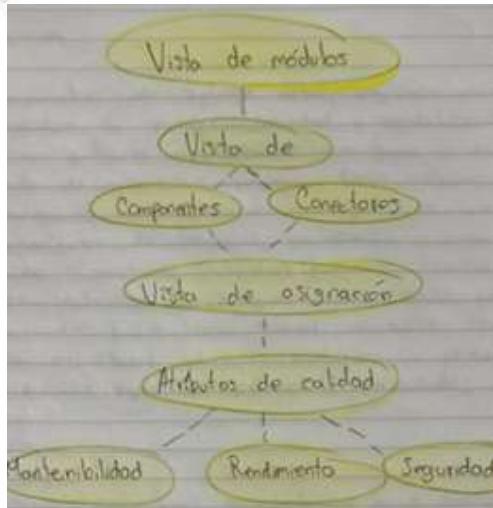
Sethi, R. (1992). Programming Languages, Concepts and Constructs. Wilmington, Delaware, USA: Addison Wesley. Recuperado de <https://n9.cl/b64r9t>

# Atributos de Calidad y Arquitectura del Software

El documento trata sobre la importancia de la arquitectura del software en el desarrollo moderno, destacando cómo esta facilita la comunicación con las partes interesadas y guía el diseño e implementación. Se examinan atributos de calidad como mantenibilidad, rendimiento y seguridad, y se presenta una metodología de documentación que integra diferentes vistas de la arquitectura. Estas vistas (módulos, componentes y conectores, y asignación) permiten analizar las cualidades dinámicas y estáticas del software, facilitando el cumplimiento de los requisitos y la adaptabilidad a las demandas de calidad.

## Bibliografía

GRISE, UPM. (s.f.). Documentación sobre Arquitectura de Software. Universidad Politécnica de Madrid, Grupo de Investigación en Ingeniería del Software Empírica (GRISE). Recuperado de <https://n9.cl/9gjrb>



## Reflexión

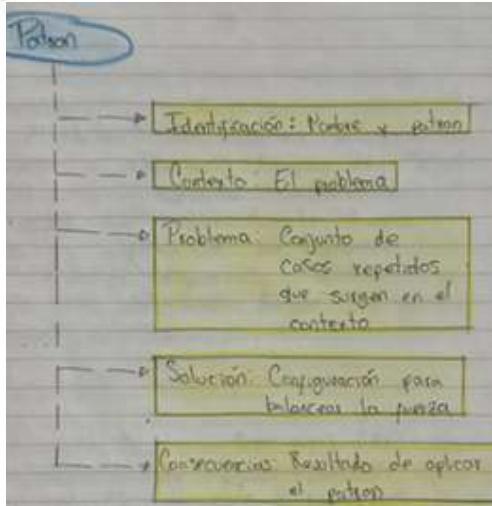
Este documento resalta la importancia de una arquitectura de software bien diseñada y documentada para lograr un sistema que cumpla con los atributos de calidad deseados, como seguridad y portabilidad. La integración de múltiples vistas en la documentación subraya la complejidad del diseño arquitectónico y la necesidad de equilibrar las demandas de distintas cualidades sin comprometer la funcionalidad general. Reflexionar sobre este proceso muestra cómo la arquitectura es fundamental para crear software que no solo cumpla los objetivos inmediatos, sino que sea adaptable a futuros desafíos.

# Especificación de la arquitectura de software

La arquitectura de software actual, impulsada por las necesidades de rapidez, seguridad y escalabilidad, se basa en patrones arquitectónicos que guían el diseño y desarrollo de aplicaciones complejas. Estos patrones, divididos en categorías como patrones de arquitectura, de diseño y específicos de lenguajes, estandarizan y organizan el software, permitiendo solucionar problemas recurrentes de manera coherente. Con enfoques variados, desde patrones de sistemas distribuidos hasta sistemas adaptables, los patrones permiten estructurar soluciones específicas y personalizables para distintos tipos de sistemas y necesidades.

## Bibliografía

RJ Code Advance. (2019, junio 25). Patrones de software: Qué es un patrón de arquitectura, parte 3. Recuperado de <https://n9.cl/aob3a>



## Reflexión

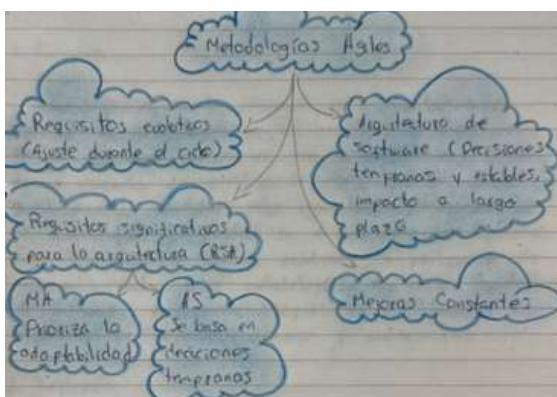
La aplicación de patrones arquitectónicos en el desarrollo de software no solo mejora la eficiencia y la seguridad, sino que también permite una comunicación clara entre los miembros del equipo, reduciendo la ambigüedad en el proceso de diseño. Los patrones representan un punto de partida estructurado que permite a los desarrolladores crear soluciones innovadoras basadas en experiencias previas. Incorporar estos patrones en proyectos futuros, entendiendo sus características y limitaciones, contribuye a un enfoque más sostenible y flexible en el desarrollo de software, garantizando una arquitectura sólida y adaptable.

# Integración de arquitectura de software en el ciclo de vida de las metodologías ágiles

Las Metodologías Ágiles (MA) promueven la adaptabilidad, colaboración y trabajo en equipo en el desarrollo de software, respondiendo a cambios y reduciendo tiempos de entrega. Al evitar definir requisitos exhaustivos al inicio, suponen que estos evolucionarán durante el proyecto. Por su parte, la Arquitectura de Software (AS) se centra en decisiones tempranas y estables, con repercusiones a largo plazo. Aunque se consideraban enfoques opuestos, la Arquitectura Ágil (AA) integra ambas perspectivas, enfocándose en Requisitos Significativos para la Arquitectura (RSA), facilitando su coexistencia.

## Reflexión

El concepto de Arquitectura Ágil (AA) sugiere que la integración de enfoques opuestos, como las MA y la AS, es viable y beneficiosa en desarrollo de software. Esta combinación aprovecha la adaptabilidad de las MA sin comprometer la solidez de la AS, al identificar y priorizar Requisitos Significativos para la Arquitectura (RSA). Así, las MA pueden adaptarse a los cambios, mientras que la AS garantiza estabilidad. La AA, por lo tanto, representa un avance que equilibra la flexibilidad y previsión estructural, mejorando la respuesta a cambios sin afectar la arquitectura.



## Bibliografía

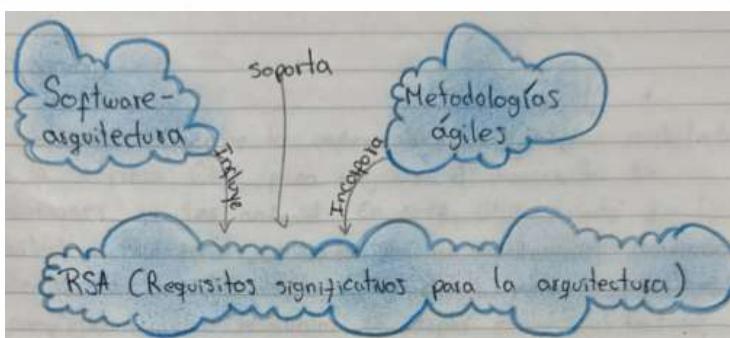
- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (s.f.). Integración de Arquitectura de Software en el Ciclo de Vida de las Metodologías Ágiles. Una Perspectiva Basada en Requisitos. Departamento de Informática, F.C.E.F. y N., U.N.S.J. Recuperado de: <https://n9.cl/qjynz7>

# Arquitectura de Software en el Proceso de Desarrollo Ágil. Una Perspectiva Basada en Requisitos Significantes para la Arquitectura

El documento explora la integración de la arquitectura de software en metodologías ágiles, resaltando la importancia de los Requisitos Significantes para la Arquitectura (RSA). A través de un proyecto de investigación, se plantea un modelo que permite capturar estos requisitos, mejorando así la flexibilidad y calidad en el desarrollo de software. Esta investigación busca resolver la percepción de incompatibilidad entre metodologías ágiles y arquitectura de software, proponiendo enfoques de RSA basados en objetivos de negocio, dominio y frameworks.

## Reflexión

La investigación abre una perspectiva innovadora al combinar arquitectura de software con metodologías ágiles, ámbitos que tradicionalmente parecían incompatibles. La identificación de los RSA en un contexto ágil responde a la necesidad de adaptabilidad en el desarrollo de software, manteniendo al mismo tiempo estándares de calidad. Esta sinergia podría ofrecer beneficios en proyectos de software modernos, en los cuales los requisitos cambian constantemente, y permitiría desarrollar sistemas más robustos y adaptables a los cambios del mercado.



## Bibliografía

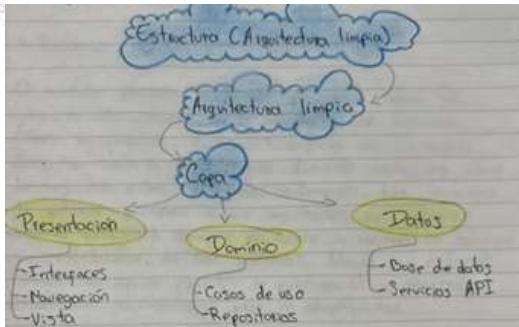
- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., & Rueda, J. R. (2018). Arquitectura de Software en el Proceso de Desarrollo Ágil. Una Perspectiva Basada en Requisitos Significantes para la Arquitectura. En XX Workshop de Investigadores en Ciencias de la Computación. RedUNCI - UNNE. Recuperado de <https://n9.cl/rk41rs>

# Clean architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD

El documento describe la implementación de Clean Architecture en la empresa GMD para mejorar el desarrollo de aplicaciones móviles en Android. En 2017, GMD detectó que la arquitectura tradicional utilizada en sus aplicaciones generaba problemas de mantenimiento y pruebas, incrementando costos y complejidad. Para resolverlo, se adoptó Clean Architecture, enfocada en la separación de capas y principios SOLID, mejorando la escalabilidad, mantenimiento y capacidad de pruebas. Los resultados fueron positivos, con una mayor eficiencia en la gestión del ciclo de vida del software y reducción de costos a largo plazo.

## Bibliografía

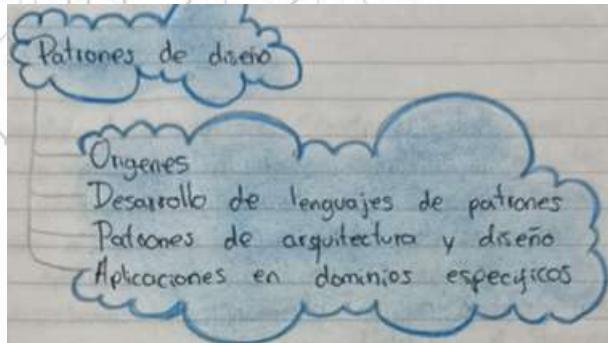
Montes Anccasi, A. J. (2018). Clean architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD. Informe de trabajo de suficiencia profesional, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software. <https://n9.cl/3j7r8>



## Reflexión

La implementación de Clean Architecture en GMD demuestra cómo una arquitectura bien definida permite a las empresas de tecnología mejorar sus procesos de desarrollo y mantenimiento. Este cambio no solo redujo los costos operativos, sino que también facilitó la adaptabilidad y robustez del software frente a cambios futuros. La experiencia destaca la importancia de las arquitecturas limpias en el desarrollo de software, especialmente en proyectos de larga duración o complejidad creciente. Esto sugiere que invertir en arquitecturas sólidas aporta valor a largo plazo y promueve una cultura de mejora continua en el desarrollo de software.

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte



El artículo presenta el estado del arte sobre los lenguajes de patrones en la arquitectura de software. Expone su evolución, desde sus orígenes con Christopher Alexander hasta su aplicación en dominios específicos como la seguridad y las aplicaciones e-Business. Los lenguajes de patrones permiten a los arquitectos de software resolver problemas comunes en diseño y mejorar la calidad y mantenibilidad de los sistemas. Además, se destacan ejemplos de uso en proyectos de investigación y desarrollo, resaltando su importancia para la adaptabilidad y eficiencia en la creación de software.

## Bibliografía

Jiménez-Torres, VH, Tello-Borja, W. y Ríos-Patiño, JI (2014). Lenguajes de Patrones de Arquitectura de Software: Una Aproximación al Estado del Arte . \*Ciencia y Scientia et Technica, 19 (4), 371-  
<https://n9.cl/ahapn>

## Reflexión

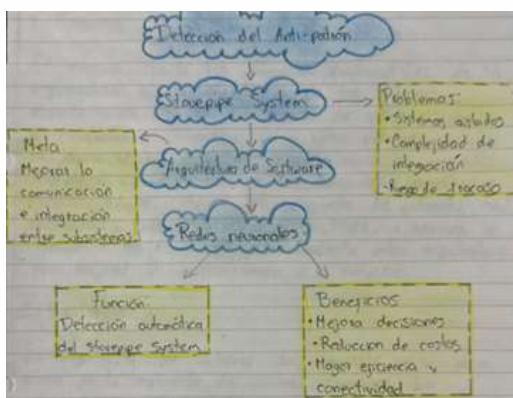
La adopción de lenguajes de patrones en la arquitectura de software representa un avance significativo en la disciplina, aportando soluciones probadas que simplifican y optimizan el proceso de diseño. Al seguir patrones establecidos, los desarrolladores aseguran consistencia y escalabilidad, lo que resulta en sistemas más robustos y adaptables. Sin embargo, el reto radica en adaptar estos patrones a las necesidades específicas de cada proyecto, evitando una aplicación rígida. Así, los lenguajes de patrones no solo mejoran el diseño técnico, sino que también promueven una perspectiva integral que beneficia a usuarios y desarrolladores.

# Implementación de una Arquitectura de Software

El documento aborda el anti-patrón Stovepipe System, que surge cuando los sistemas son altamente complejos y aislados, dificultando la integración y comunicación. Este anti-patrón puede llevar al fracaso en proyectos de software, especialmente en instituciones con sistemas heredados y múltiples subsistemas. La propuesta es un modelo teórico apoyado en redes neuronales para identificar el Stovepipe System durante la implementación de arquitecturas de software, lo cual mejoraría la toma de decisiones y la eficiencia del sistema mediante soluciones adaptativas.

## Reflexión

La identificación y análisis de anti-patrones, como el Stovepipe System, es fundamental en la arquitectura de software, pues permite anticiparse a problemas recurrentes que podrían limitar el éxito de un proyecto. El enfoque del artículo de utilizar redes neuronales para detectar este anti-patrón es innovador, ya que facilita una toma de decisiones basada en experiencias previas y aprendizaje automático. Esta metodología podría ayudar a las organizaciones a optimizar sus recursos y a reducir costos, mejorando la interconectividad entre subsistemas y la resiliencia de sus sistemas.

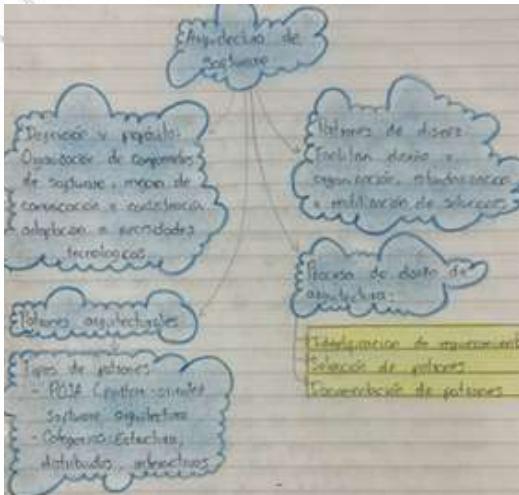


## Bibliografía

Candia Peñaloza, J. C. (s.f.). Modelo Teórico para la Identificación del Antipatrón "Stovepipe System" en la Etapa de la Implementación de una Arquitectura de Software. Revista PGI - Investigación, Ciencia y Tecnología, 1(1), 121-127. Recuperado de <https://n9.cl/n963p>

# Arquitectura de software

El artículo aborda los elementos fundamentales de la arquitectura de software y la importancia de los patrones arquitecturales en la construcción de sistemas robustos y escalables. Describe cómo estos patrones facilitan la comunicación, estructura y consistencia en proyectos complejos, adaptándose a las demandas tecnológicas actuales. Además, expone las categorías de patrones más conocidas, como los POSA y los PEAA, y subraya la relevancia de documentar adecuadamente estas arquitecturas para mejorar el mantenimiento y la evolución de los sistemas a lo largo del tiempo.



## Reflexión

La arquitectura de software se convierte en un pilar clave en la ingeniería de sistemas complejos. La estructuración mediante patrones facilita que los desarrolladores manejen desafíos de escalabilidad, flexibilidad y seguridad. Al aplicar estos patrones, se fomenta una cultura de reutilización y claridad en la comunicación, permitiendo construir soluciones sostenibles en el tiempo. Esta reflexión invita a comprender que la calidad en el diseño de software no solo reside en la innovación, sino en la disciplina y en el respeto por metodologías probadas que mejoran los resultados y disminuyen la ambigüedad en el desarrollo.

## Bibliografía

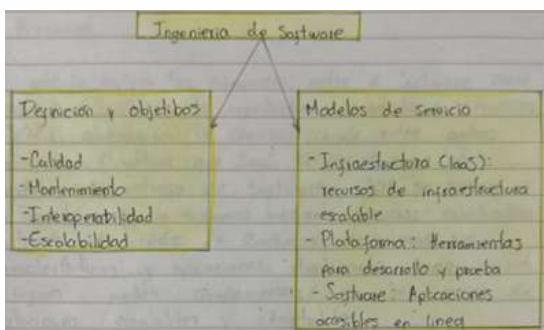
- Quilindo, L. A., & Vega, J. S. (2023). Especificación de la arquitectura de software. Revista TIA: Tecnología, Investigación y Academia, 11(2), 170-181. Recuperado de <https://n9.cl/aob3a>

# INGENIERÍA DE SOFTWARE Y COMPUTACIÓN EN LA NUBE. CONCEPTOS BÁSICOS QUE DEBEN SER ENSEÑADOS EN EL CURSO DE INGENIERÍA DESOFTWARE I, SOFTWARE ENGINEERING AND CLOUD COMPUTING BASIC

El artículo explora los conceptos fundamentales de la ingeniería de software y la computación en la nube, destacando su relevancia en la formación de estudiantes de software. Se enfoca en los modelos de servicios de la nube, como infraestructura, plataforma y software como servicio, y sus aplicaciones en la industria. También se revisan estudios que abordan desafíos y beneficios de la migración a la nube, incluyendo compatibilidad y control de calidad. Finalmente, se enfatiza la importancia de adaptar los programas de estudio para capacitar a futuros ingenieros en este campo emergente.

## Reflexión

El avance hacia la computación en la nube representa un cambio significativo en cómo se desarrolla y mantiene el software, obligando a los profesionales a repensar los enfoques tradicionales. Este paradigma abre nuevas posibilidades en términos de escalabilidad y eficiencia, pero también plantea desafíos de seguridad y compatibilidad. Es crucial que las universidades integren estos temas en sus programas de ingeniería de software, para que los futuros ingenieros puedan adaptarse a las demandas tecnológicas actuales y aplicarlas de manera efectiva en entornos empresariales.



## Bibliografía

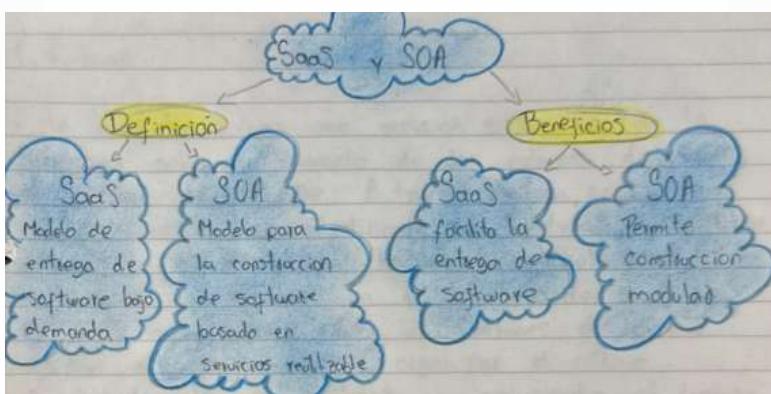
- Saldaña-Barrios, J. J., & Moreno, Y. (2016). Ingeniería de software y computación en la nube: Conceptos básicos que deben ser enseñados en el curso de ingeniería de software I. Plus (+) Economía, 4(2), 33-40. Recuperado de <https://n9.cl/d6jy6h>

# Distinguiendo entre SaaS y SOA

El artículo explora las diferencias entre el Software como Servicio (SaaS) y la Arquitectura Orientada a Servicios (SOA), abordando la confusión común entre ambos conceptos. Mientras que SaaS se define como un modelo de entrega de software, SOA se centra en la construcción de sistemas mediante servicios reutilizables. A través del modelo de Zachman, se aclaran sus características y aplicaciones, destacando cómo ambos enfoques pueden complementarse en la creación de soluciones escalables y adaptables. Las tecnologías de servicios web facilitan la integración y adopción tanto de SaaS como de SOA en la industria.

## Reflexión

Comprender la distinción entre SaaS y SOA es crucial en la arquitectura de software, ya que el uso incorrecto de estos términos puede llevar a decisiones de diseño ineficaces. SaaS ofrece eficiencia al facilitar la entrega de software bajo demanda, mientras que SOA permite flexibilidad y reutilización en la construcción de sistemas. Integrar estos modelos puede ofrecer soluciones más robustas y escalables, pero es esencial que los profesionales de TI comprendan sus características específicas para aplicarlos adecuadamente. La adopción de tecnologías de servicios web fortalece la implementación de estos modelos en la práctica.



## Bibliografía

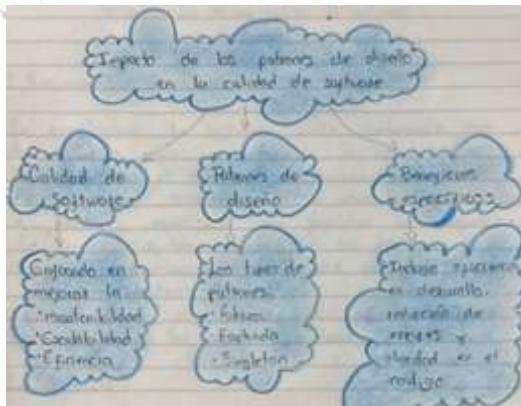
González-Fuente, J. J. (2013). Distinguendo entre SaaS y SOA. JJEGonzálezF. Recuperado de <https://n9.cl/meir0>

# El impacto de los patrones de diseño en la calidad del software: una revisión sistemática de la literatura

Este artículo realiza una revisión sistemática de la literatura sobre el impacto de los patrones de diseño en la calidad del software. A través de un análisis de estudios previos, se identifican las ventajas de utilizar patrones de diseño en términos de mantenibilidad, reutilización, y eficiencia en el desarrollo de software. El artículo destaca cómo los patrones de diseño ayudan a estructurar y organizar el código, facilitando la escalabilidad y reduciendo el tiempo de desarrollo. Además, se exponen estudios que vinculan patrones específicos con beneficios particulares en la calidad del software, como los patrones de fábrica y de fachada, que mejoran la cohesión y reducen la dependencia entre módulos.

## Bibliografía

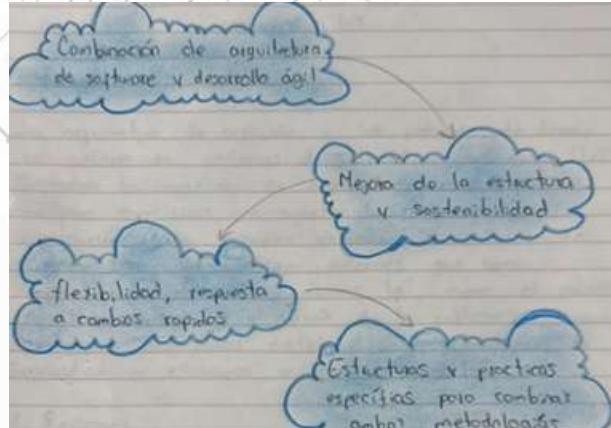
Wedyan, F. y Abufakher, S. (2020), Impacto de los patrones de diseño en la calidad del software: una revisión sistemática de la literatura. IET Softw., 14: 1-17. <https://n9.cl/w2vf4>.



## Reflexión

El uso de patrones de diseño en la arquitectura de software no solo contribuye a la organización y claridad del código, sino que también permite abordar problemas complejos de manera estructurada. Estos patrones, al ser soluciones probadas, reducen la probabilidad de errores y mejoran la eficiencia, además de facilitar el trabajo colaborativo en equipos de desarrollo. Sin embargo, un punto de reflexión es que, aunque los patrones de diseño aportan muchos beneficios, su implementación también puede agregar complejidad al proyecto si no se usan de manera adecuada o si no se entienden completamente sus propósitos. Por lo tanto, los desarrolladores deben equilibrar la aplicación de patrones con las necesidades específicas del proyecto y la experiencia del equipo.

# Un estudio de mapeo sistemático sobre la combinación de arquitectura de software y desarrollo ágil



Este artículo explora cómo la arquitectura de software puede combinarse con metodologías ágiles para mejorar la eficiencia y flexibilidad en el desarrollo de software. A través de un mapeo sistemático, se identifican prácticas y patrones específicos que facilitan la integración de una arquitectura sólida con los principios ágiles. Se discuten casos de estudio que muestran cómo las prácticas arquitectónicas pueden apoyar el desarrollo ágil, particularmente en proyectos donde los requisitos cambian rápidamente.

## Bibliografía

- Chen, Y., Liang, P., & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, 111, 157-184.  
<https://n9.cl/ty5z9>

## Reflexión

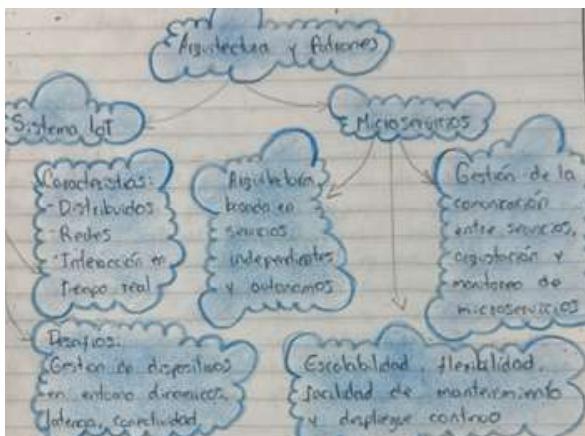
La combinación de una arquitectura de software robusta con metodologías ágiles es crucial para mantener la flexibilidad en el desarrollo, especialmente en entornos donde los cambios son constantes. Este artículo revela que los patrones de diseño ayudan a que el desarrollo ágil sea más estructurado, lo cual permite adaptarse rápidamente a nuevas necesidades sin comprometer la calidad del software. No obstante, integrar ambas prácticas requiere un equilibrio cuidadoso; demasiada rigidez en la arquitectura puede obstaculizar la agilidad, mientras que una arquitectura demasiado flexible puede afectar la sostenibilidad a largo plazo.

# Panorama de la arquitectura y patrones de diseño para sistemas IoT

Este artículo proporciona una visión general de cómo la arquitectura de software y los patrones de diseño se aplican en sistemas de Internet de las Cosas (IoT). Destaca la importancia de arquitecturas escalables y patrones específicos que soporten la comunicación y procesamiento de grandes volúmenes de datos en tiempo real. Se analizan patrones que son especialmente útiles en entornos IoT, como el patrón de microservicios y el patrón de publicación-suscripción, para gestionar dispositivos distribuidos y garantizar la seguridad y eficiencia en los sistemas IoT.

## Reflexión

La arquitectura de sistemas IoT requiere adaptaciones específicas para gestionar la complejidad y la distribución de dispositivos. Este artículo resalta que, en un entorno IoT, la elección de patrones arquitectónicos es fundamental para soportar escalabilidad, seguridad y procesamiento en tiempo real. Los patrones de diseño ofrecen soluciones para abordar los desafíos de comunicación y sincronización entre múltiples dispositivos. No obstante, integrar estos patrones puede incrementar la complejidad del sistema, por lo que es crucial evaluar su impacto en la escalabilidad y el rendimiento.



## Bibliografía

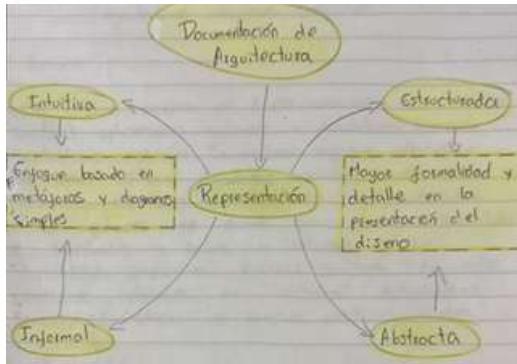
H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, EB Fernandez y N. Yoshioka, "Paisaje de la arquitectura y patrones de diseño para sistemas IoT", en IEEE Internet of Things Journal <https://n9.cl/pupxw3>.

# Documentación de la arquitectura de software desde una perspectiva de gestión del conocimiento: representación del diseño

Este artículo explora cómo documentar la arquitectura de software desde una perspectiva de gestión del conocimiento. Se analiza cómo la documentación bien estructurada facilita la comprensión y el mantenimiento del software, y cómo actúa como una herramienta de comunicación entre equipos. La representación del diseño es fundamental para capturar el conocimiento técnico y no técnico del sistema, promoviendo una colaboración más efectiva y la capacidad de actualización de la arquitectura a medida que evoluciona el proyecto.

## Bibliografía

Kruchten, P. (2009). Documentation of software architecture from a knowledge management perspective - Design representation. En Software Architecture Knowledge Management <https://n9.cl/fgzbe>.



## Reflexión

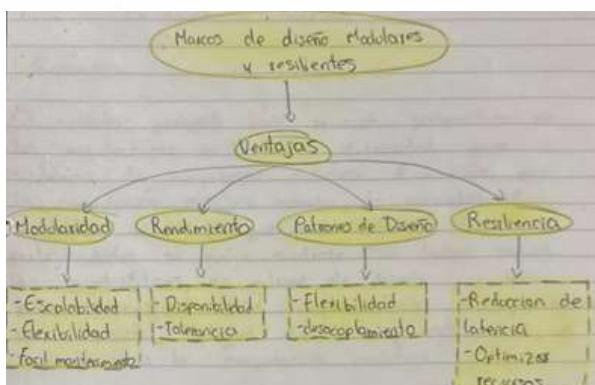
Documentar la arquitectura de software no solo ayuda en la implementación actual, sino que también garantiza la sostenibilidad a largo plazo de un proyecto. Este artículo destaca que la documentación adecuada es clave para la transferencia de conocimiento y permite que nuevos miembros del equipo puedan integrarse rápidamente. Sin embargo, mantener la documentación actualizada puede ser un desafío, especialmente en proyectos ágiles. Encontrar un equilibrio entre la documentación detallada y la flexibilidad es esencial para aprovechar su potencial.

# **Marcos de diseño avanzados para aplicaciones modernas y escalables: enfoques estratégicos para crear arquitecturas modulares, resilientes y de alto rendimiento en sistemas distribuidos**

Este artículo examina marcos de diseño avanzados que permiten construir aplicaciones modernas y escalables mediante arquitecturas modulares. Se analizan enfoques estratégicos para crear sistemas distribuidos resilientes y de alto rendimiento, abordando la importancia del modularidad y los patrones de diseño que soportan escalabilidad y resistencia a fallos en entornos complejos. El artículo proporciona ejemplos prácticos de estos principios aplicados en sistemas distribuidos.

## **Reflexión**

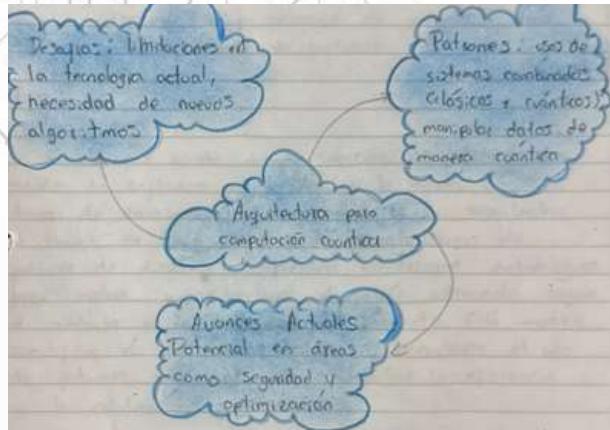
La modularidad y la resiliencia son fundamentales para el éxito de sistemas modernos y distribuidos. Este artículo subraya cómo los patrones de diseño avanzados no solo mejoran el rendimiento del sistema, sino que también facilitan la capacidad de actualización y adaptación ante cambios o fallos. Sin embargo, lograr un equilibrio entre modularidad y complejidad es crucial para no añadir costos excesivos al desarrollo. Adoptar un enfoque estratégico permite crear arquitecturas sostenibles y de alto rendimiento que pueden crecer con las necesidades de la organización.



## **Bibliografía**

Sara Moreno. (2021). Advanced Design Frameworks for Modern, Scalable Applications: Strategic Approaches to Building High-Performance, Resilient, and Modular Architectures in Distributed Systems. *Sage Science Review of Applied Machine Learning*, 4(1), 66–95. <https://n9.cl/onxI9>.

# Arquitectura de software para sistemas de computación cuántica: una revisión sistemática



El artículo presenta una revisión sistemática de las arquitecturas de software diseñadas para sistemas de computación cuántica. A medida que la computación cuántica avanza, la necesidad de arquitecturas específicas que puedan manejar sus particularidades se vuelve evidente. Este trabajo revisa las características de estas arquitecturas, sus patrones de diseño, y los retos específicos de implementación, proporcionando una visión integral de la arquitectura en el contexto de la computación cuántica.

## Bibliografía

- Khan, A. A., Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Mikkonen, T., & Abrahamsson, P. (2023). Software architecture for quantum computing systems — A systematic review. *Journal of Systems and Software*, 201, 111682. <https://n9.cl/4r1uu>.

## Reflexión

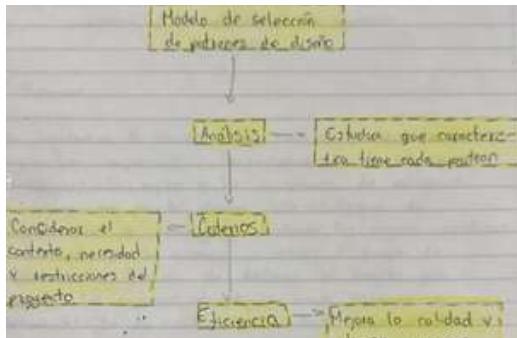
El desarrollo de arquitecturas para la computación cuántica está en sus primeras etapas, y este artículo proporciona un panorama claro de los desafíos que enfrenta. Las limitaciones de hardware y los requisitos específicos de los algoritmos cuánticos hacen que el diseño de estas arquitecturas sea particularmente desafiante. A pesar de ello, los avances en el software cuántico están sentando bases sólidas para futuras aplicaciones, mostrando que la computación cuántica podría transformar diversas industrias una vez que estas arquitecturas se implementen a gran escala.

# Modelo para la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software

Este artículo propone un modelo que asiste en la selección de patrones de diseño de software, enfocado en la ayuda a la toma de decisiones para desarrolladores y arquitectos. A través de un análisis de características de patrones de diseño, se presenta un enfoque metodológico para evaluar y seleccionar el patrón adecuado según el contexto y las necesidades del proyecto. Este modelo simplifica el proceso de decisión, promoviendo el uso de patrones adecuados para maximizar la eficiencia y la calidad del software.

## Bibliografía

Cuesta Villa, M. (2007). Modelo para la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software  
<https://n9.cl/5j5vg>



## Reflexión

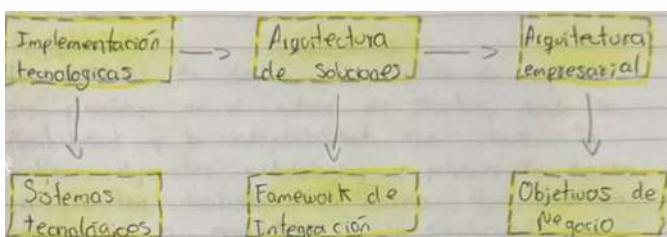
La selección de patrones de diseño es una decisión crucial en el desarrollo de software. Este modelo ayuda a la toma de decisiones representando un recurso valioso, especialmente para proyectos con limitaciones específicas. Un enfoque estructurado para la elección de patrones permite que los equipos de desarrollo optimicen el rendimiento y eviten la complejidad innecesaria. Sin embargo, es importante recordar que este modelo debe usarse como una guía flexible, adaptándose a las particularidades de cada proyecto.

# **Enfoque de arquitectura de soluciones, mecanismo para reducir la brecha entre la arquitectura empresarial y la implementación de soluciones tecnológicas**

Este artículo explora cómo la arquitectura de soluciones sirve como un puente entre la arquitectura empresarial y la implementación de soluciones tecnológicas. Se enfoca en la importancia de establecer un marco que permita alinear las estrategias de negocio con la tecnología, abordando las mejores prácticas para optimizar la eficiencia y reducir el tiempo de implementación. Además, se destacan los desafíos que enfrentan las empresas al integrar nuevos sistemas y herramientas en una infraestructura existente, especialmente en sectores con alta demanda de adaptabilidad y escalabilidad. La arquitectura de soluciones permite establecer una guía clara para los procesos de integración y asegurar el cumplimiento de los objetivos organizacionales.

## **Reflexión**

La importancia de la arquitectura de soluciones en la actualidad no puede subestimarse, especialmente en un entorno donde la transformación digital es una prioridad para las empresas. Este enfoque no solo ayuda a reducir la brecha entre los objetivos empresariales y la tecnología, sino que también facilita una implementación coherente y ordenada de soluciones. Personalmente, considero que aplicar estos principios en proyectos reales permitiría mejorar la eficiencia y minimizar los costos de desarrollo a largo plazo. La arquitectura de soluciones, además, promueve un enfoque colaborativo entre equipos técnicos y de negocio, permitiendo un entendimiento común de los objetivos y las necesidades.



## **Bibliografía**

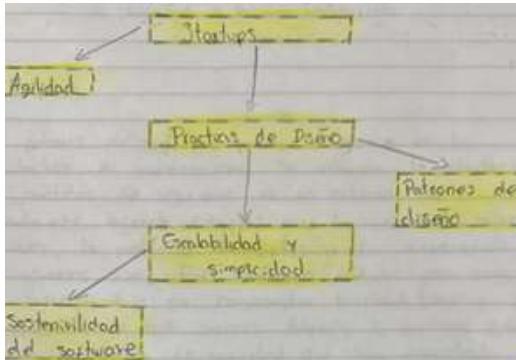
Arango-Serna, MD, Londoño-Salazar, JE, & Branch-Bedoya, JW (2015). Enfoque de arquitectura de soluciones, mecanismo para reducir la brecha entre la arquitectura empresarial y la implementación de soluciones tecnológicas. DYNA, 82 <https://n9.cl/1q4o6>

# **Estudio descriptivo de las prácticas de diseño y arquitectura de desarrollo de Software en las compañías (Startups).**

Este artículo examina las prácticas de diseño y arquitectura de software que prevalecen en el entorno de startups, donde la flexibilidad y la adaptabilidad son clave. Se analizan diferentes metodologías y patrones de diseño que permiten a las startups desarrollar software de manera ágil, con un enfoque en la escalabilidad y la simplicidad para enfrentar el rápido crecimiento y los cambios del mercado. Los autores presentan estudios de casos en los que demuestran cómo estas prácticas ayudan a mejorar la eficiencia y reducir los costos, manteniendo la calidad y la sostenibilidad del software en entornos de alta presión y recursos limitados.

## **Bibliografía**

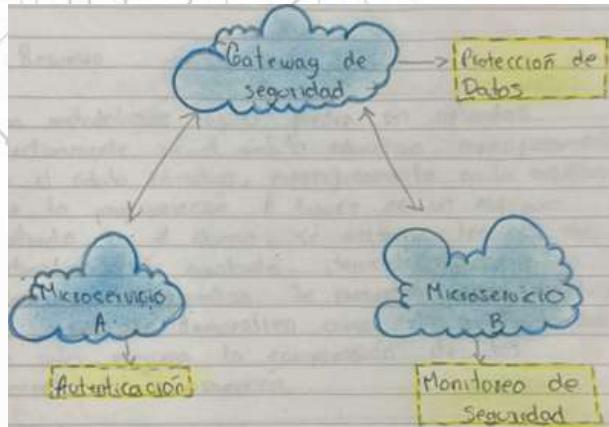
Estudio descriptivo de las prácticas de diseño y arquitectura de desarrollo de Software en las compañías Startups. // Descriptive Study about design and architecture practices in Startups. (2017). CIENCIA UNEMI, 10(23), 125-132. <https://n9.cl/f237r>



## **Reflexión**

La arquitectura en las startups es fundamentalmente diferente de la que se implementa en empresas más grandes. Este estudio muestra cómo las startups pueden beneficiarse de una arquitectura flexible y escalable que se adapta a las necesidades cambiantes del mercado. Reflexionando sobre esta lectura, creo que muchas empresas emergentes pueden optimizar sus recursos adoptando estas prácticas y principios de diseño. La capacidad de ajustar rápidamente la infraestructura de software es clave en un mundo de negocios donde la adaptabilidad es esencial para la supervivencia y el éxito a largo plazo.

# Adaptación de un patrón de software en seguridad a la arquitectura de un Microservicio



El artículo presenta un análisis de cómo los patrones de seguridad pueden integrarse en la arquitectura de microservicios. Se examina la adaptación de prácticas de seguridad en un entorno que generalmente enfrenta desafíos como la comunicación entre servicios, la protección de datos y la autenticación de usuarios. Los autores describen métodos para asegurar la infraestructura sin comprometer la flexibilidad y la independencia de cada servicio. Además, se analizan ejemplos de implementación de seguridad que han demostrado ser efectivos para mitigar riesgos en un sistema distribuido, brindando una visión integral de las consideraciones de seguridad en microservicios.

## Bibliografía

Hernández Guzmán, KM (2023). Adaptación de un patrón de software en seguridad a la arquitectura de un microservicio <https://n9.cl/6nckty>.

## Reflexión

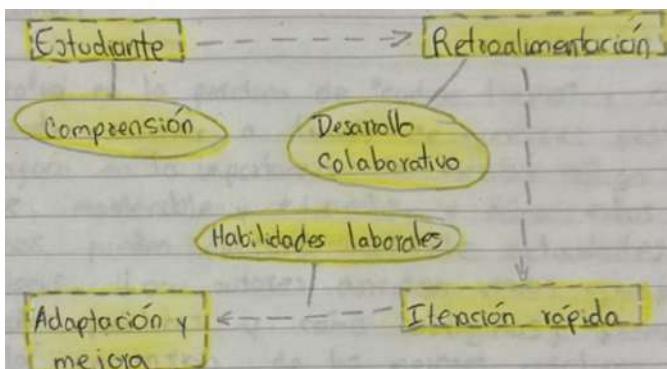
La seguridad es un aspecto crítico en la arquitectura de microservicios, y este artículo proporciona una visión valiosa sobre cómo adaptar patrones de seguridad a este contexto. Personalmente, considero que integrar estos patrones es esencial para proteger los sistemas de ataques, especialmente en una arquitectura distribuida que expone múltiples puntos de acceso. Las empresas que adopten estos enfoques de seguridad tendrán una ventaja significativa al poder ofrecer servicios confiables sin comprometer la independencia y la flexibilidad de cada componente del sistema.

# Metodologías ágiles de desarrollo aplicadas a la enseñanza de la programación

Este artículo discute cómo las metodologías ágiles pueden ser aplicadas efectivamente en el ámbito educativo, específicamente en la enseñanza de la programación. A través de un enfoque centrado en el alumno, se destacan técnicas de retroalimentación constante, iteración rápida y desarrollo colaborativo. Se presentan varios estudios de caso que demuestran cómo estas metodologías no solo mejoran la comprensión de los conceptos de programación, sino que también preparan a los estudiantes para enfrentar los desafíos del mundo laboral, promoviendo habilidades como la adaptabilidad y el trabajo en equipo.

## Reflexión

La aplicación de metodologías ágiles en educación representa una innovación valiosa para el aprendizaje de programación. Reflejando en esta lectura, pienso que adoptar estas prácticas ayudaría a los estudiantes a desarrollar una mentalidad más flexible y colaborativa. Además, este enfoque podría reducir la frustración asociada con el aprendizaje de conceptos complejos, al permitir a los estudiantes progresar a su propio ritmo y recibir retroalimentación constructiva en cada etapa de su aprendizaje.



## Bibliografía

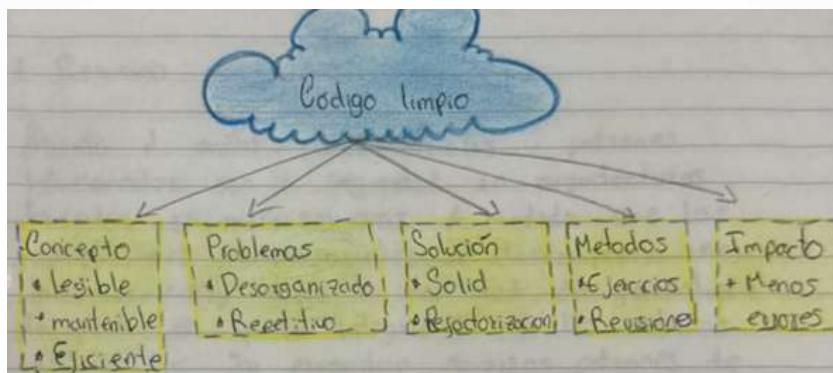
Metodologías ágiles de desarrollo aplicadas a la enseñanza de la programación. (2021, julio 29). Google Académico. <https://n9.cl/mom8g>

# Código Limpio: Cómo aprenderlo a través de un ejercicio práctico

Este artículo se centra en la práctica de "código limpio" y cómo se puede enseñar a través de ejercicios prácticos. Se enfoca en la importancia de escribir código legible, mantenible y eficiente, y cómo estos principios pueden aplicarse mediante actividades prácticas. Los autores discuten varios ejemplos de malas prácticas y cómo corregirlas, promoviendo así la comprensión de las mejores prácticas de programación. La práctica de código limpio, aseguran, no solo mejora la calidad del software, sino que también facilita la colaboración en equipo y la escalabilidad del proyecto.

## Reflexión

El concepto de "código limpio" es fundamental para cualquier desarrollador, y enseñar estos principios de manera práctica puede ser muy efectivo. Este artículo me hizo reflexionar sobre la importancia de aplicar buenas prácticas en el código desde el principio. Implementar código limpio no solo reduce errores y simplifica el mantenimiento, sino que también hace que el trabajo sea más comprensible para otros. Es un enfoque valioso que todos los desarrolladores deberían integrar en su formación.



## Bibliografía

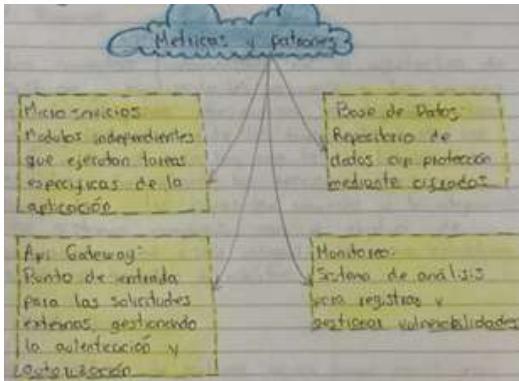
Pérez Lagunas, JA (2023). Aprendiendo a aplicar código limpio a través de un ejercicio práctico <https://n9.cl/1t78u>

# Estudio de Métricas y Patrones de Seguridad en Microservicios

Este trabajo de investigación aborda el análisis de métricas y patrones relacionados con la seguridad en arquitecturas basadas en microservicios. A medida que las aplicaciones empresariales se descomponen en microservicios, surge la necesidad de garantizar la protección y la integridad de los sistemas distribuidos. Se presentan diversos patrones de diseño y métricas de seguridad que pueden ayudar a mitigar vulnerabilidades en este tipo de arquitecturas, además de ofrecer recomendaciones para una implementación más segura y eficiente.

## Bibliografía

Estudio de Métricas y Patrones de Seguridad en Microservicios. Tesis de Maestría.. También se recomienda revisar libros de microservicios y seguridad, como Building Microservices de Sam Newman (2015), y Microservices: A Software Architecture Pattern de Martin Fowler (2014).  
<https://n9.cl/5ug42>



## Reflexión

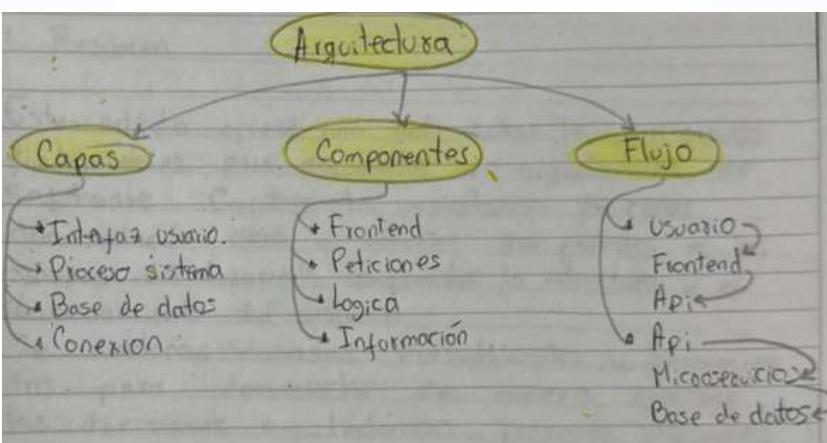
La seguridad en microservicios es fundamental en un entorno distribuido y dinámico. La naturaleza descentralizada de los microservicios aumenta las complejidades en la gestión de la seguridad. Este estudio resalta la importancia de adoptar un enfoque preventivo y multidimensional, utilizando métricas precisas para evaluar y mejorar la seguridad. Además, la implementación de patrones de seguridad adecuados, como autenticación descentralizada y control de acceso, es esencial para proteger los datos y servicios, así como para evitar brechas de seguridad.

# Arquitectura de Software, Esquemas y Servicios

Este artículo aborda los conceptos fundamentales de la arquitectura de software, enfatizando los esquemas y los servicios que la componen. Se analizan los diferentes tipos de arquitecturas, desde las tradicionales hasta las más modernas como las arquitecturas basadas en servicios. Se exploran las relaciones entre los componentes, las capas de servicio y la integración de sistemas complejos, con el objetivo de proporcionar una visión integral sobre cómo estructurar soluciones escalables y robustas.

## Reflexión

La arquitectura de software es un campo crucial para el desarrollo de aplicaciones complejas, ya que permite garantizar la escalabilidad, fiabilidad y mantenimiento a largo plazo. Este artículo subraya la importancia de diseñar arquitecturas bien estructuradas y de comprender las interacciones entre los diferentes componentes. Los esquemas de arquitectura y los servicios permiten una modularización que facilita la expansión y evolución del sistema, al mismo tiempo que mejora la gestión de los recursos.



## Bibliografía

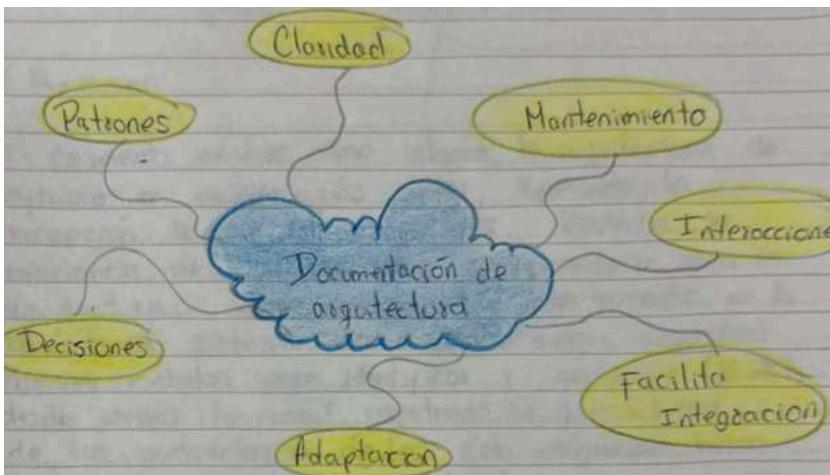
Romero, P. Á. (2006). Código Limpio: Cómo aprenderlo a través de un ejercicio práctico. Ingeniería Industrial, 27(1). Ejemplar dedicado a: Ingeniería Industrial. ISSN-e 1815-5936. <https://n9.cl/vzvqm>.

# Documentando la Arquitectura de Software: Principios Básicos

Este artículo ofrece una guía sobre los principios fundamentales para documentar una arquitectura de software. Enfatiza la importancia de crear documentación clara y concisa que permita a los equipos de desarrollo comprender la estructura y el funcionamiento del sistema. Se destacan diversas metodologías y herramientas para documentar de manera efectiva las decisiones arquitectónicas, patrones utilizados, y la interacción entre los distintos componentes del sistema.

## Reflexión

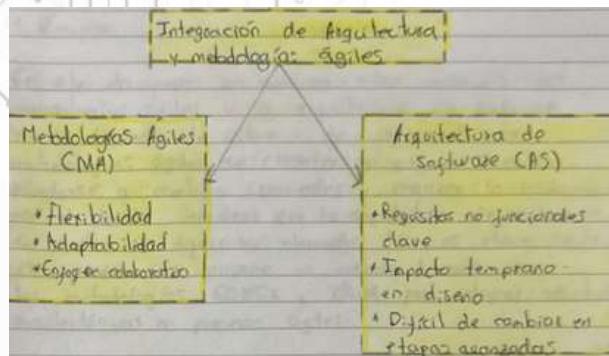
La documentación de la arquitectura de software es esencial para garantizar la comprensión y mantenimiento de los sistemas a largo plazo. Este artículo resalta cómo una buena documentación puede facilitar la integración de nuevos miembros en un equipo, la resolución de problemas y la adaptación a nuevas tecnologías. Documentar no solo las decisiones técnicas, sino también las razones detrás de ellas, proporciona un contexto crucial para futuras modificaciones y mejoras.



## Bibliografía

- Gómez, O. (2006). Documentando la arquitectura de software: Principios básicos. Revista Ingeniería Industrial, 27(1) <https://n9.cl/wvImw>.

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



El documento analiza cómo integrar la arquitectura de software en metodologías ágiles, superando la percepción de que son incompatibles. Destaca la importancia de los Requisitos Significantes para la Arquitectura (RSA), aquellos que impactan directamente en la calidad del sistema, como rendimiento y seguridad. Proponer métodos para identificar y gestionar RSA desde etapas tempranas, respetando la flexibilidad de las metodologías ágiles. Esta integración busca mejorar la calidad y adaptabilidad de los sistemas desarrollados en contextos 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. En XX Workshop de Investigadores en Ciencias de la Computación (pp. 635-639). <https://n9.cl/rk41rs>

## Reflexión

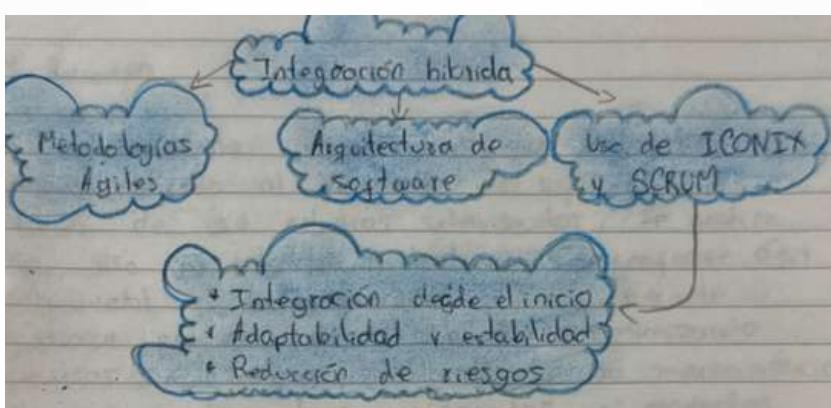
La integración entre arquitectura de software y metodologías ágiles demuestra que enfoques aparentemente opuestos pueden complementarse si se centran en objetivos claros, como los RSA. Es un recordatorio de que la adaptabilidad no debe sacrificar calidad ni planificación estratégica. Este modelo no solo mejora proyectos tecnológicos, sino que también refleja cómo equilibrar flexibilidad y estructura en cualquier campo, subrayando el valor de la colaboración interdisciplinaria para resolver problemas complejos.

# Selección de Metodologías Ágiles e Integración de Arquitecturas de Software en el Desarrollo de Sistemas de Información

Este trabajo analiza cómo combinar las metodologías ágiles y la arquitectura de software en el desarrollo de sistemas de información. Las metodologías ágiles se caracterizan por ser flexibles, adaptarse a cambios frecuentes y priorizar la colaboración con el cliente, mientras que la arquitectura de software se centra en definir elementos técnicos clave desde el principio. Se propone un modelo basado en las metodologías ICONIX y SCRUM para integrar actividades arquitectónicas en procesos ágiles, mejorando la calidad, la adaptabilidad y la eficiencia de los sistemas desarrollados.

## Reflexión

La combinación de metodologías ágiles con prácticas de arquitectura de software resalta la importancia de unir flexibilidad y planificación en el desarrollo de sistemas. Las metodologías ágiles ofrecen dinamismo y enfoque en el cliente, mientras que la arquitectura asegura estabilidad y calidad técnica. Lograr este balance no solo mejora los resultados, sino que también fomenta un desarrollo más eficiente y sostenible. Este enfoque híbrido es un recordatorio de que la adaptabilidad y el rigor técnico pueden coexistir en beneficio del usuario final.

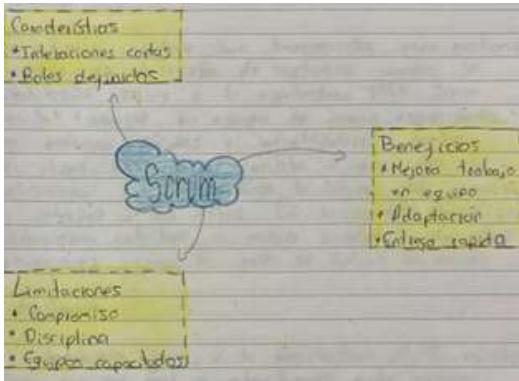


## Bibliografía

- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., & Rueda, J. R. (2017). Selección de Metodologías Ágiles e Integración de Arquitecturas de Software en el Desarrollo de Sistemas de Información. En XIX Workshop de Investigadores en Ciencias de la Computación (pp. 632-635). Universidad Nacional de San Juan. <https://n9.cl/53lyw>

# Revisión Sistemática de la Metodología Scrum para el Desarrollo de Software

El estudio sistemáticamente revisa la metodología Scrum aplicada al desarrollo de software. A través de 24 estudios seleccionados, se analiza su uso en proyectos, destacando su enfoque ágil, incremental y autoorganizado. Scrum permite gestionar proyectos con flexibilidad, optimizando recursos y tiempos. Sin embargo, su implementación requiere equipos bien conformados y proyectos correctamente dimensionados. Es ideal para proyectos pequeños y medianos, pero puede no ser adecuado para sistemas complejos que exigen alta precisión y estabilidad.



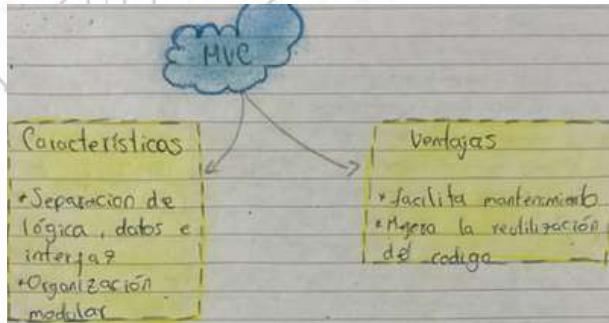
## Reflexión

Scrum es una metodología ágil que transforma la manera en que trabajamos en equipo, promoviendo la organización, flexibilidad y entrega de valor constante. Este enfoque fomenta la colaboración y la capacidad de adaptarse a cambios. Sin embargo, no es una solución universal; requiere planeación adecuada y equipos capacitados para maximizar sus beneficios. Su éxito depende en gran medida del compromiso de los participantes y de la implementación correcta de sus principios.

## Bibliografía

López Morales, José Raúl.  
46 Congreso Academia de  
Journals Morelia 2019.  
Tecnológico Nacional de  
México, 2019.  
<https://n9.cl/4nnzyu>

# Propuesta de una herramienta basada en la metodología Scrum para la gestión del desarrollo de software



El artículo propone una herramienta para gestionar proyectos de desarrollo de software usando la metodología Scrum y la arquitectura MVC. Scrum permite trabajar en equipo de forma organizada, con entregas rápidas y adaptándose a cambios. Se eligió MVC porque facilita el desarrollo de aplicaciones web al separar la lógica del negocio, la interfaz y los datos. La herramienta busca ser colaborativa, intuitiva y resolver problemas de otras aplicaciones, como la falta de trabajo en equipo o interfaces difíciles de usar.

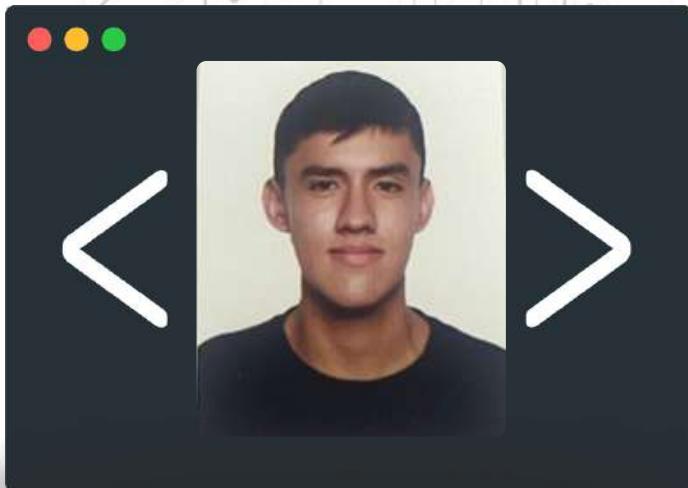
## Bibliografía

Romero, Juan Carlos; Rivera, Gabriel. "Reflexiones sobre los cambios en la pedagogía y sus implicaciones en la educación". Dialnet, Universidad de La Rioja, 2017. <https://n9.cl/prlgZ>

## Reflexión

La metodología Scrum y la arquitectura MVC muestran cómo un enfoque ágil y estructurado puede mejorar la gestión de proyectos de software. Scrum fomenta el trabajo en equipo y la adaptación al cambio, mientras que MVC organiza las aplicaciones para facilitar su mantenimiento. Este enfoque sugiere que diseñar herramientas más accesibles y enfocadas en la colaboración es clave para atender las necesidades reales de los equipos de desarrollo y superar las limitaciones de las herramientas.

# **Kevin Camilo Muñoz Campos**



Nací el 12 de noviembre de 2005 en Bogotá, Colombia. Realicé mis estudios de bachillerato y actualmente estoy cursando el tecnólogo en Análisis y Desarrollo de Software en el SENA.

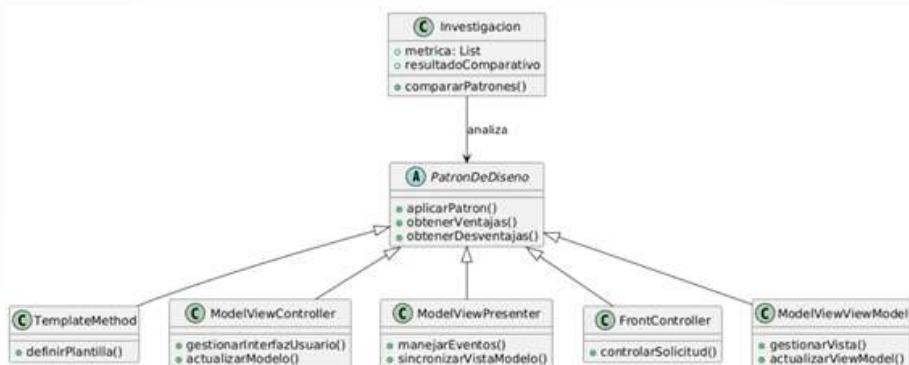
Además, trabajó en la empresa Tabasco de Claro, lo que me permite combinar mis estudios con una experiencia laboral valiosa. Soy una persona apasionada por la tecnología, pero también encuentro tiempo para mis pasatiempos, como escuchar música, jugar fútbol y voleibol, disfrutar de videojuegos y entrenar en el gimnasio. Estos pasatiempos me ayudan a mantener un equilibrio entre mi vida personal y profesional. Mi meta a futuro es graduarme de la universidad como profesional en Análisis y Desarrollo de Software o Mecatrónica. Aspiro a conseguir un trabajo destacado en mi área y, posteriormente, emprender y establecer mis propias empresas en diversos sectores. Me considero una persona amigable, respetuosa, y tolerante. Disfruto ayudando a las personas y construyendo relaciones positivas.

# Análisis comparativo de Patrones de Diseño de Software

Los patrones de diseño son herramientas que facilitan la resolución de problemas comunes en el desarrollo de software, promoviendo la organización, reutilización, y modularidad del código. Este artículo analiza cinco patrones específicos: Template Method, MVC, MVP, Front Controller y MVVM, comparándolos según métricas específicas. No existe un patrón "mejor", ya que cada uno está diseñado para contextos específicos. Su correcta aplicación depende del criterio del desarrollador, buscando mejorar la estructura, comprensión y mantenimiento del código, contribuyendo así a software de calidad.

## Reflexión

Los patrones de diseño son guías flexibles, no soluciones universales, que exigen un análisis crítico para elegir la más adecuada según las necesidades del proyecto. Su uso refleja una ética profesional que valora el equilibrio entre teoría y práctica, con el objetivo de crear sistemas claros, sostenibles y comprensibles. Estos patrones enriquecen el trabajo creativo al permitir la adaptación y aprendizaje continuo en cada proyecto.



## Bibliografía

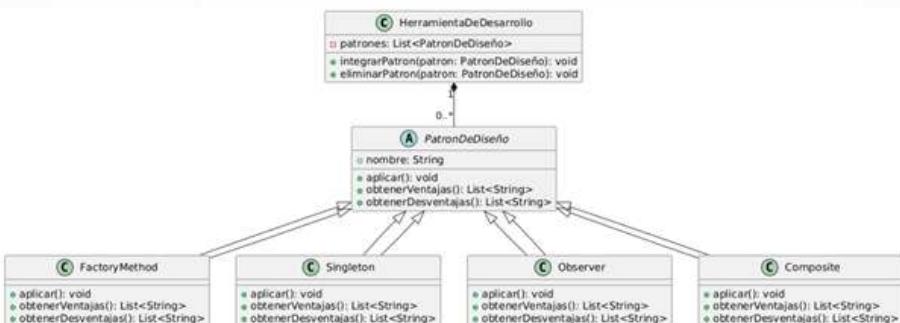
- Alvarez, O. D. G., Larrea, N. P. L., & Valencia, M. V. R. (2022). Análisis comparativo de Patrones de Diseño de Software. Polo del Conocimiento: Revista científico-profesional, 7(7), 2146-2165.  
<https://acortar.link/hVLV7Q>

# Una arquitectura para una Herramienta de Patrones de Diseño

El artículo propone una arquitectura para herramientas de desarrollo que integren patrones de diseño, lo cual facilitaría su uso en la creación de software. Aunque los patrones son soluciones probadas y estructuradas para problemas comunes, actualmente pocas herramientas los incluyen de forma integrada. Existen puntos de vista distintos sobre esta integración: algunos creen que los patrones deben guiar a los programadores sin automatizar el código, mientras que otros apoyan su implementación directa en herramientas. La arquitectura presentada permite representar y manipular patrones de manera flexible, con el objetivo de mejorar la eficiencia en el desarrollo orientado a objetos y facilitar el diseño de código modular y reutilizable.

## Reflexión

La propuesta resalta la importancia de contar con herramientas que faciliten el uso de patrones de diseño, lo cual podría revolucionar la forma en que los desarrolladores abordan problemas comunes en el software. Si bien es cierto que los patrones de diseño no deben reemplazar la creatividad y el análisis humano, su inclusión en las herramientas puede potenciar el trabajo de los programadores al reducir esfuerzos y mejorar la calidad del código. En última instancia, el uso adecuado de los patrones dentro de un entorno de desarrollo podría ayudar a crear software más limpio, estructurado y mantenable.



## Bibliografía

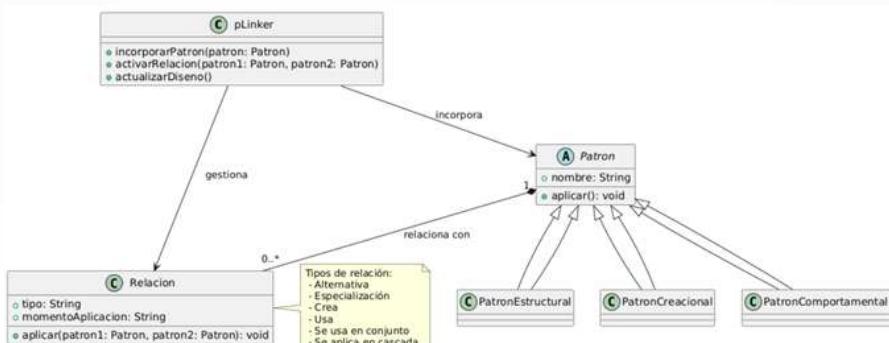
Martínez, J. S., Molina, J. G., & García, P. J. J. (1999). Una Arquitectura para una Herramienta de Patrones de Diseño. Dpto. de Informática y Sistemas, Universidad de Murcia, Murcia. <https://acortar.link/Eo9KOT>

# pLinker: Relaciones con Patrones de Diseño

Los patrones de diseño, útiles de forma individual, pueden complementarse para resolver problemas complejos. Sin embargo, la documentación rara vez detalla cómo combinar o relacionar estos patrones. Este trabajo clasifica formalmente seis tipos de relaciones entre patrones, como su uso conjunto, especialización o creación. Además, se presenta pLinker, un editor gráfico que permite modelar clases en UML, integrar patrones en diseños existentes y gestionar relaciones automáticamente. La herramienta facilita construir diseños orientados a objetos, integrando patrones de forma coherente y manteniendo la consistencia, especialmente en arquitecturas grandes como MVC.

## Reflexión

La propuesta de este artículo destaca que los patrones de diseño no solo resuelven problemas puntuales, sino que pueden combinarse para enriquecer las soluciones. Al clasificar las relaciones entre ellos, se facilita la creación de software más flexible y adaptado. La herramienta pLinker automatiza la integración de patrones, permitiendo que se complementen o modifiquen sin afectar la estructura original, optimizando el diseño y promoviendo una programación más eficiente. Este enfoque favorece software modular, coherente y fácil de mantener, mejorando la escalabilidad y reutilización.



## Bibliografía

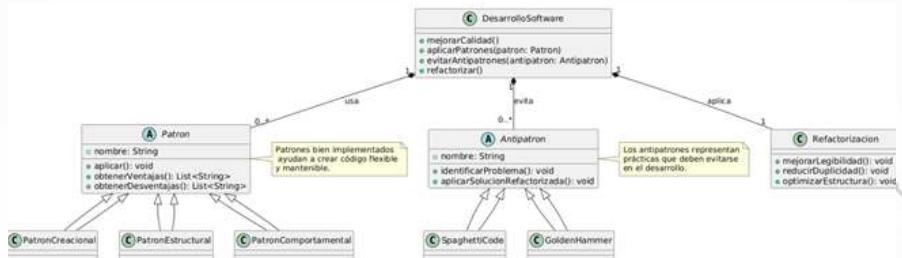
- Liebener, L., Rossi, M. A., & Marcos, C. (2003). pLinker: Relaciones con Patrones de Diseño. Facultad de Ciencias Exactas. <https://acortar.link/CCd7JR>

# Patrones de diseño, refactorización y antipatrones

El artículo aborda tres elementos clave del desarrollo orientado a objetos: patrones de diseño, refactorización y antipatrones. Los patrones, como Singleton o Factory Method, brindan soluciones a problemas comunes, mejorando la flexibilidad y el mantenimiento. Los antipatrones, como Spaghetti Code o Golden Hammer, identifican prácticas dañinas que deben evitarse. La refactorización, por su parte, reorganiza el código para hacerlo más claro sin cambiar su funcionalidad. La combinación de estos elementos ayuda a crear software sólido, reutilizable y menos propenso a errores, facilitando su evolución y mejorando el producto final.

## Reflexión

Este análisis resalta que desarrollar software no es solo escribir código funcional, sino aplicar estrategias basadas en experiencia. Los patrones de diseño ofrecen guías para estructurar software escalable y comprensible, mientras que los antipatrones identifican prácticas ineficaces que deben evitarse para mejorar la calidad del código. La refactorización, clave en este proceso, permite evolucionar el código sin comprometer su legibilidad, fomentando un mantenimiento continuo. Estas prácticas, en conjunto, agilizan el desarrollo, mejoran la adaptabilidad y optimizan recursos, resultando en software de alta calidad.



## Bibliografía

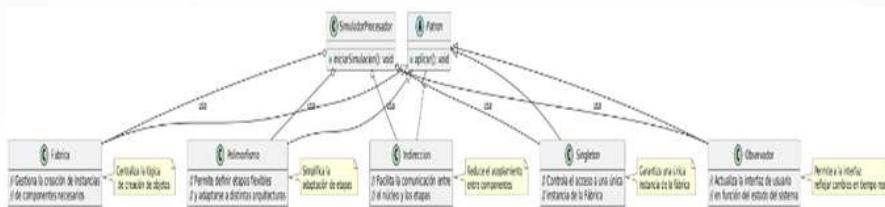
- Campo, G. D. (2009). Patrones de diseño, refactorización y antipatrones. <https://acortar.link/5I0YB2>

# Uso de Patrones de Diseño: Un caso Práctico

El artículo destaca cómo los patrones de diseño ofrecen soluciones prácticas para problemas comunes en el desarrollo de software, mejorando la estructura y funcionalidad del código. Usando un simulador multinúcleo como caso de estudio, se aplican patrones como Polimorfismo, Fábrica, Singleton, Indirección y Observador para aumentar la extensibilidad y el mantenimiento. El Polimorfismo aporta flexibilidad, Fábrica y Singleton simplifican la creación de objetos, Indirección reduce el acoplamiento, y Observador actualiza la interfaz sin alterar la lógica, optimizando el simulador para cambios sin afectar su núcleo.

## Reflexión

La aplicación de patrones de diseño demuestra cómo mejorar la organización y flexibilidad del software. Estos patrones permiten trabajar con código más limpio, facilitando cambios y nuevas funcionalidades sin comprometer la estabilidad. Sin embargo, como señala el artículo, es crucial aplicarlos en el contexto adecuado para evitar complejidades innecesarias. Este equilibrio entre diseño y eficiencia es esencial para crear software escalable y mantenable. Además, la experiencia práctica con patrones enriquece la comprensión del diseño orientado a objetos, promoviendo soluciones duraderas y de calidad.



## Bibliografía

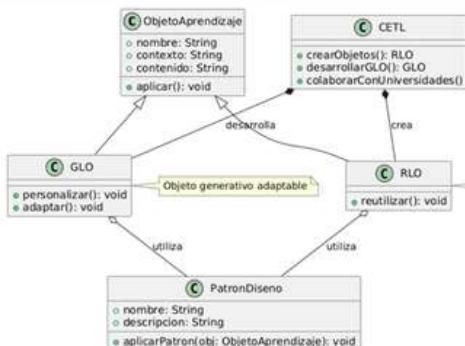
- Bermúdez, G. S., Jiménez, I. M., & Rodríguez, L. R. (2012). Uso de Patrones de Diseño: Un caso Práctico. Ingeniería, 22(2), 45-59. <https://acortar.link/o8qNw>

# Patrones de diseño: ejemplo de aplicación en los Generative Learning Object

El artículo describe cómo los patrones de diseño pueden mejorar la creación y reutilización de objetos de aprendizaje en el ámbito del e-learning. Al definir soluciones reutilizables para problemas comunes en la enseñanza virtual, estos patrones facilitan la creación de recursos educativos flexibles y adaptables. Uno de los proyectos analizados, el Centro de Excelencia CETL, se centra en la creación de objetos de aprendizaje generativos (GLO) que permiten a los docentes personalizar contenido en función de sus necesidades educativas. Los GLO se construyen a partir de objetos de aprendizaje reutilizables (RLO), logrando materiales educativos adaptables y eficientes para el aprendizaje en línea.

## Reflexión

Este trabajo destaca el valor de los patrones de diseño en la educación, donde la adaptabilidad y la reutilización de recursos son esenciales para satisfacer diversas necesidades educativas. La posibilidad de crear objetos de aprendizaje generativos demuestra cómo los patrones no solo optimizan el proceso de desarrollo de contenido, sino que también permiten una enseñanza más personalizada y efectiva. Al incorporar soluciones probadas y adaptables, los educadores pueden enfocarse en la calidad pedagógica, confiando en herramientas estructuradas que facilitan el aprendizaje interactivo y accesible en entornos digitales.



## Bibliografía

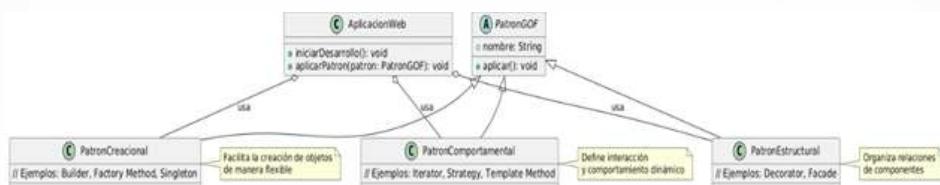
- Tello, J. C. (2009). Patrones de diseño: ejemplo de aplicación en los Generative Learning Object. Revista de Educación a Distancia (RED). <https://acortar.link/tlvXim>

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

El artículo analiza la implementación de patrones de diseño "Gang of Four" (GOF) en el desarrollo de aplicaciones web, observando su uso en procesos formales de desarrollo de software en Colombia. Los patrones de diseño, clasificados en creacionales, estructurales y de comportamiento, ayudan a resolver problemas comunes y mejoran la calidad del software. Sin embargo, la investigación revela que el uso de patrones en la industria sigue siendo limitado, en parte debido a la falta de conocimiento o experiencia. En general, el patrón `Singleton` es uno de los más utilizados en los proyectos analizados, mientras que otros patrones, como algunos de tipo estructural, son menos empleados.

## Reflexión

Este trabajo destaca la importancia de los patrones de diseño como herramientas que estandarizan y optimizan el desarrollo de software, especialmente en aplicaciones complejas como las aplicaciones web. La investigación muestra que, aunque algunos patrones se usan con frecuencia, otros patrones útiles podrían aprovecharse mejor. Esto sugiere que el entrenamiento y la familiarización de los desarrolladores con todos los patrones del catálogo GOF pueden potenciar la calidad y sostenibilidad de los proyectos de software en el país. A través del uso adecuado de estos patrones, los equipos de desarrollo pueden crear aplicaciones más eficientes y adaptables, optimizando tiempo y recursos a largo plazo.



## Bibliografía

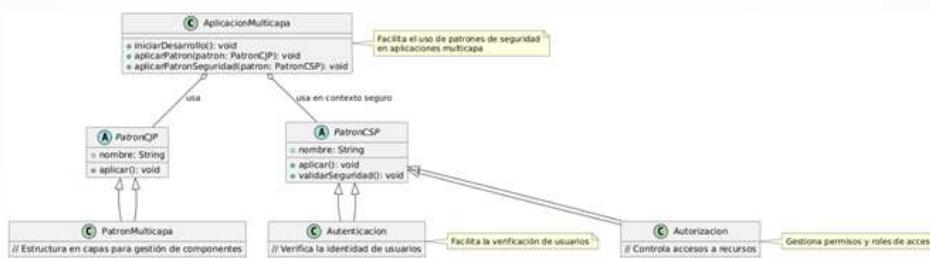
- Guerrero, C. A., Suárez, J. M., & Gutiérrez, L. E. (2013). Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web. *Información tecnológica*, 24(3), 103-114. <https://acortar.link/o3XI7c>

# Integración de patrones de seguridad y patrones de diseño J2EE

Este trabajo analiza dos de los catálogos de patrones de diseño más conocidos: el \*\*Core Security Patterns (CSP)\*\* y el \*\*Core J2EE Pattern Catalogue (CJP)\*\*, ambos originados de la experiencia de Sun Microsystems. El objetivo principal es integrar de manera coherente los patrones de seguridad (CSP) con los patrones multicapa (CJP), para así facilitar su uso conjunto. Para esto, el estudio examina los patrones de seguridad en el contexto de una arquitectura multicapa, definiendo una secuencia de patrones necesarios para comprender completamente los patrones de seguridad en aplicaciones multicapa. Se desarrollan dos casos prácticos para mostrar la implementación conjunta de patrones de ambos catálogos.

## Reflexión

Este trabajo destaca la importancia de integrar patrones de diseño para fortalecer arquitecturas de aplicaciones complejas. En particular, se analiza cómo los patrones de seguridad y multicapa pueden complementarse, mejorando tanto la estructura como la protección de datos. Esto resulta clave en la industria, donde la seguridad es fundamental. Desde lo educativo, se propone un enfoque secuencial que guía a estudiantes y desarrolladores desde conceptos básicos hasta patrones avanzados en contextos multicapa. Esta integración fomenta arquitecturas más robustas, seguras y fáciles de mantener.



## Bibliografía

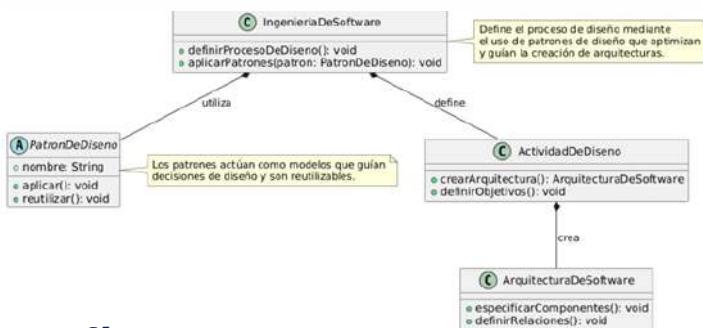
Parra Valdés, W. E. (2014). Integración de patrones de seguridad y patrones de diseño J2EE. <https://acortar.link/BhPDNA>

# Soporte a la actividad de diseño basado en patrones de diseño

Este trabajo explora cómo los patrones de diseño pueden servir como soporte para el proceso de diseño en ingeniería de software, permitiendo una reusabilidad similar a la que se observa en otras disciplinas de ingeniería. La investigación destaca que los patrones de diseño, al ser tanto un objeto (la solución) como una regla (cuándo y cómo aplicarla), ofrecen un marco para anticipar decisiones de diseño y optimizar el desarrollo. El estudio revisa las definiciones de diseño de autores como Ralph, Wand y Buschmann, que consideran el diseño tanto una actividad como un objeto que deriva en la arquitectura del software. Se plantea un modelo conceptual y contextual para estructurar la actividad de diseño y se proponen líneas de investigación para desarrollar herramientas y técnicas basadas en patrones.

## Reflexión

Este artículo resalta la importancia de los patrones de diseño como herramientas que no solo proporcionan soluciones reutilizables, sino que también guían el proceso de diseño en la ingeniería de software. Al tratar los patrones como modelos anticipados que definen cuándo y cómo aplicarse, los desarrolladores pueden optimizar el diseño, especialmente en arquitecturas complejas. La investigación también aborda la necesidad de un enfoque más sistemático para la enseñanza y la práctica del diseño basado en patrones. Esto impulsa una ingeniería de software más madura, donde el uso de patrones permite organizar y estructurar mejor el trabajo, haciendo posible construir software de forma más consistente y eficaz.



## Bibliografía

- Roqué Fourcade, L. E., & Arakaki, L. (2015, May). Soporte a la actividad de diseño basado en patrones de diseño. In XVII Workshop de Investigadores en Ciencias de la Computación (Salta, 2015). <https://acortar.link/s60guA>

# El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing

El artículo se enfoca en el patrón de diseño Modelo-Vista-Controlador (MVC) y su aplicación en Java Swing. Este patrón busca dividir una aplicación en tres componentes: el Modelo, la Vista y el Controlador.

1. **\*\*Modelo\*\*:** Representa la lógica y los datos de la aplicación, desconociendo cómo serán visualizados o controlados.
2. **\*\*Vista\*\*:** Se encarga de presentar la información del Modelo al usuario, y se actualiza automáticamente cuando el Modelo cambia.
3. **\*\*Controlador\*\*:** Gestiona la entrada del usuario y actualiza el Modelo y la Vista según sea necesario.

El artículo también explora la implementación de MVC en Java Swing, donde el patrón presenta una variación: el controlador y la vista se unifican en el "delegado de interfaz de usuario" (UI delegate). Esto permite que cada componente visual esté vinculado directamente con su Modelo, facilitando la actualización y adaptación de los datos.

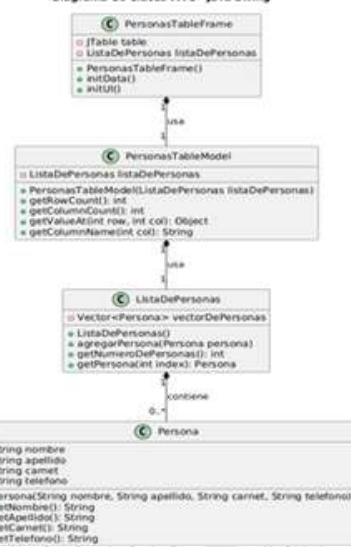
## Bibliografía

Pantoja, E. B. (2004). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. *Acta Nova*, 2(4), 493. <https://acortar.link/bklBI9>

## Reflexión

El patrón MVC aporta organización y claridad a los sistemas complejos, promoviendo el desarrollo modular. Aunque aumenta el tiempo inicial de desarrollo, ofrece a largo plazo una mayor mantenibilidad y facilidad para actualizar o extender una aplicación. En Java Swing, el MVC mejora la experiencia de desarrollo al proporcionar un marco estructurado para componentes gráficos, aunque requiere una curva de aprendizaje.

Diagrama de Clases MVC - Java Swing

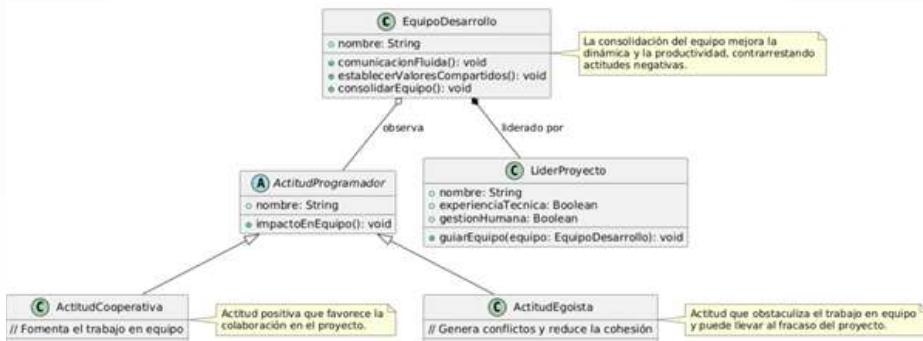


# Un nuevo Patrón de Diseño de Software: Programmer's Attitude.

El artículo presenta el patrón de diseño "Programmer's Attitude" para resolver problemas de comportamiento en equipos de desarrollo causados por diferencias en actitudes y personalidades. Estas incluyen falta de cooperación en programadores experimentados y sobreestimación de habilidades en novatos, además de problemas de comunicación y roles poco claros. La solución propone consolidar equipos con comunicación fluida, valores compartidos y líderes experimentados que combinen habilidades técnicas y de gestión humana, mejorando la dinámica y productividad del equipo.

## Reflexión

Este artículo destaca la importancia del factor humano en el éxito de proyectos de software. El patrón Programmer's Attitude aplica principios de diseño para abordar problemas de comportamiento y cohesión en equipos. Resalta que, además de un buen código, un equipo cohesionado y motivado es clave para construir software de calidad. La relación entre desarrolladores y líderes del proyecto es esencial para lograr resultados consistentes. Este enfoque fomenta gerencias técnicas efectivas y equipos donde la cooperación y la comunicación mitiguen conflictos, impulsando objetivos comunes.



## Bibliografía

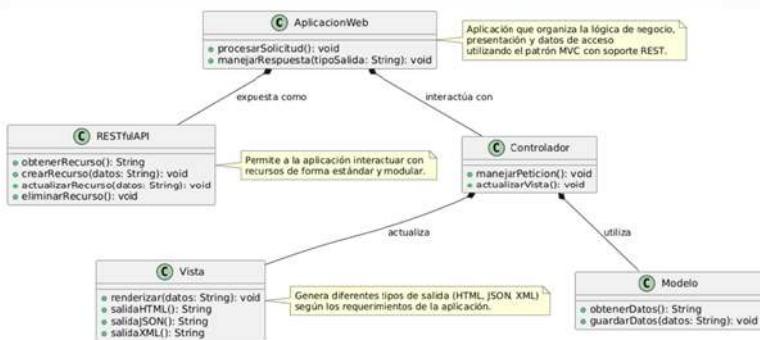
Pantaleo, G. G., & Roitbarg, D. A. " Un nuevo Patrón de Diseño de Software: Programmer's Attitude. <https://acortar.link/Jm3H2c>

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

Este trabajo aborda la implementación de un framework para aplicaciones web usando el patrón MVC y la arquitectura REST, con Zend Framework como base. Facilita la separación de la lógica de negocios, la presentación y el acceso a datos, promoviendo una estructura modular y escalable. Se destacan tecnologías clave como Memcached para mejorar el rendimiento y ejemplos de cómo este framework se aplica en aplicaciones web para generar contenido multimedia, soportando múltiples salidas como HTML, XML y JSON.

## Reflexión

Este framework basado en los patrones MVC y REST destaca la importancia de una arquitectura modular y flexible en aplicaciones web modernas. Al separar la lógica del negocio, el acceso a datos y la presentación, se mejora la adaptabilidad y la gestión. El uso de Memcached optimiza el rendimiento en entornos de alta demanda, mientras que este enfoque promueve buenas prácticas de diseño y reduce la duplicación de código, esencial para proyectos a largo plazo. En resumen, demuestra cómo los patrones de diseño crean aplicaciones robustas y eficientes, adaptables a nuevas necesidades.



## Bibliografía

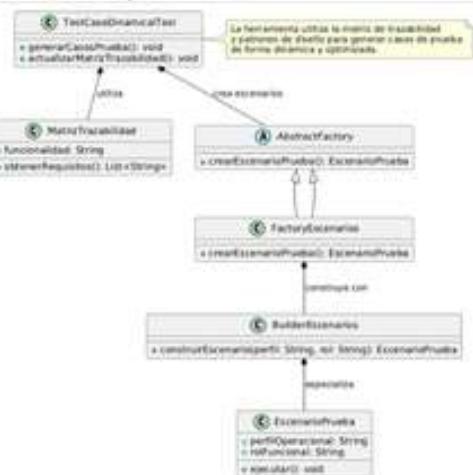
- : Zulian, E. R. (2011). Implementación de un framework para el desarrollo de aplicaciones web utilizando patrones de diseño y arquitectura MVC/REST (Doctoral dissertation, Universidad de Belgrano. Facultad de Tecnología Informática.). <https://acortar.link/xtxLEw>

# Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño.

Este artículo propone un método innovador para automatizar la creación de casos de prueba funcionales mediante regresión, utilizando patrones de diseño. La propuesta se basa en una herramienta de generación dinámica de casos de prueba (Test-Case-Dynamical Tool, TCDT), la cual emplea patrones como \*\*Abstract Factory\*\* y \*\*Builder\*\* para crear casos de prueba que se adaptan a la funcionalidad y trazabilidad de cada sistema de software en tiempo de ejecución. A través de una matriz de trazabilidad, la herramienta puede actualizarse con nuevos requisitos y generar solo los casos de prueba necesarios, optimizando tiempo y recursos de pruebas en proyectos complejos.

## Bibliografía

Leticia, D. N., Guerrero, O. M., Juárez, I. A., & de la Vega, J. A. Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño.  
<https://acortar.link/7o3A2q>



## Reflexión

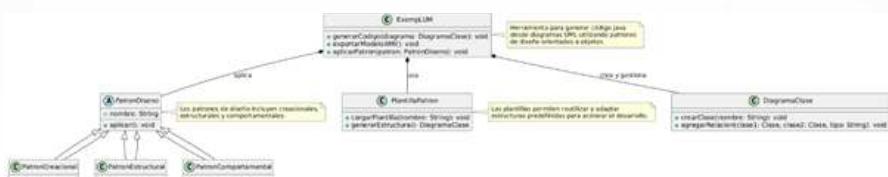
Este framework basado en los patrones MVC y REST destaca la importancia de una arquitectura modular y flexible en aplicaciones web modernas. Al separar la lógica del negocio, el acceso a datos y la presentación, se mejora la adaptabilidad y la gestión. El uso de Memcached optimiza el rendimiento en entornos de alta demanda, mientras que este enfoque promueve buenas prácticas de diseño y reduce la duplicación de código, esencial para proyectos a largo plazo. En resumen, demuestra cómo los patrones de diseño crean aplicaciones robustas y eficientes, adaptables a nuevas necesidades.

# Herramienta de generación de código a partir de patrones de diseño

ExempLUM es una herramienta que genera código Java a partir de diagramas UML utilizando patrones de diseño orientados a objetos. Facilita el diseño y la implementación de software mediante una interfaz gráfica para crear diagramas UML y la generación automática de código basada en los patrones de la "Pandilla de los Cuatro". Es útil para diseñadores novatos, ya que ofrece plantillas reutilizables, acelera el desarrollo y reduce errores. Además, permite exportar modelos en formato XMI, mejorando la interoperabilidad y flexibilidad en proyectos de software.

## Reflexión

ExempLUM destaca la importancia de los patrones de diseño en la programación orientada a objetos, proporcionando soluciones estandarizadas que mejoran la claridad y modularidad del código. La herramienta facilita el aprendizaje para desarrolladores novatos al ofrecer plantillas y generar código automáticamente, lo que optimiza el tiempo de desarrollo y reduce errores. Fomenta la creación de software robusto, mantenable y escalable, sin necesidad de empezar desde cero. Además, permite exportar modelos en formato XMI, lo que aumenta la interoperabilidad entre plataformas y herramientas, haciendo el desarrollo más ágil y accesible para equipos de diversos niveles de experiencia.



## Bibliografía

Garcés Rodríguez, B. J., Martínez García, D. A., Morales Durán, C. D., & Sánchez Santana, A. Herramienta de generación de código a partir de patrones de diseño. <https://acortar.link/vXD1Fm>

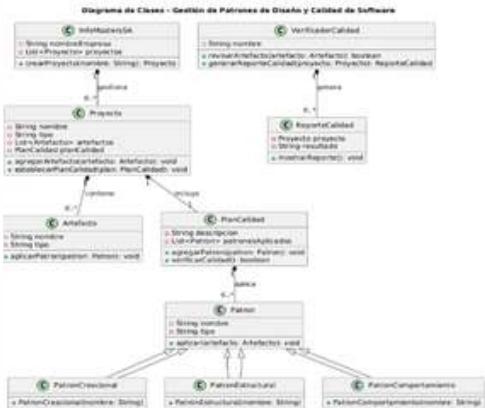
# Aseguramiento de la calidad del software aplicando patrones de diseño durante el proceso de desarrollo.

El documento analiza el uso de patrones de diseño en InfoMasters S.A., una empresa dedicada al desarrollo de software financiero. La implementación de patrones de diseño busca mejorar la calidad del software y optimizar el proceso de desarrollo. Se establece un plan de aseguramiento de calidad con base en estos patrones, los cuales son clasificados en creacionales, estructurales y de comportamiento. La investigación sugiere que los patrones de diseño mejoran la claridad del código, facilitan la colaboración y reducen errores, aumentando así la satisfacción del cliente y disminuyendo los costos de mantenimiento y corrección.

## Bibliografía

López Umaña, J. P. (2005). Aseguramiento de la calidad del software aplicando patrones de diseño durante el proceso de desarrollo.

<https://acortar.link/MPMyK7>



## Reflexión

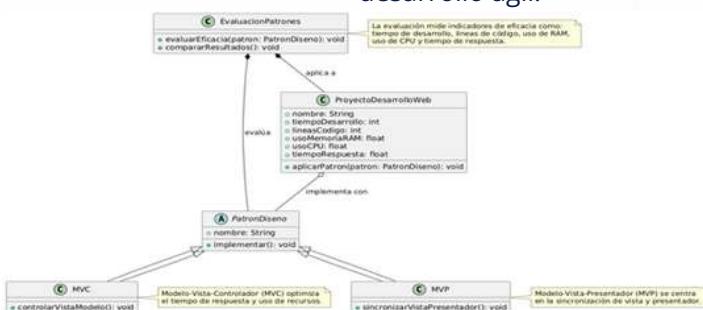
El uso de patrones de diseño en InfoMasters S.A. no solo estructura el proceso de desarrollo, sino que también eleva los estándares de calidad, beneficiando tanto al equipo como al cliente. Al promover la reutilización de componentes y reducir la complejidad del código, los patrones favorecen la eficiencia y la claridad en el desarrollo. Implementar estos patrones fomenta un ciclo de mejora continua, en el que cada proyecto se convierte en una fuente de aprendizaje y estandarización. En esencia, este enfoque no solo permite crear productos más robustos y confiables, sino que también establece una base sólida para la evolución tecnológica y la satisfacción en el trabajo en equipo.

# Análisis comparativo de patrones de diseño MVC y MVP para el rendimiento de aplicaciones web

Este trabajo de investigación evaluó empíricamente los patrones de diseño en el desarrollo de software web, enfocándose en la incertidumbre al seleccionar patrones adecuados. Se identificaron y validaron varios patrones mediante puntuaciones de eficacia, seleccionando los patrones MVC y MVP para una evaluación más detallada en dos proyectos. Se midieron indicadores como el tiempo de desarrollo, líneas de código, uso de memoria RAM, uso de CPU y tiempo de respuesta. Los resultados mostraron que el patrón MVC fue más eficaz que MVP en términos de rendimiento y eficiencia en el desarrollo de software web.

## Reflexión

La evaluación comparativa de los patrones MVC y MVP subraya la importancia de seleccionar el diseño adecuado, ya que cada patrón afecta la eficiencia, el uso de recursos y la productividad del equipo. El estudio demuestra que los patrones de diseño no solo ofrecen soluciones técnicas, sino que también influyen en la agilidad del desarrollo y el rendimiento final. MVC, al ser más eficaz, es preferible para proyectos web que requieren alta modularidad y rapidez en el tiempo de respuesta. Sin embargo, la elección de patrones debe considerar el contexto y los requisitos específicos de cada proyecto, permitiendo decisiones basadas en datos objetivos y fomentando un software optimizado y adaptado al desarrollo ágil.



## Bibliografía

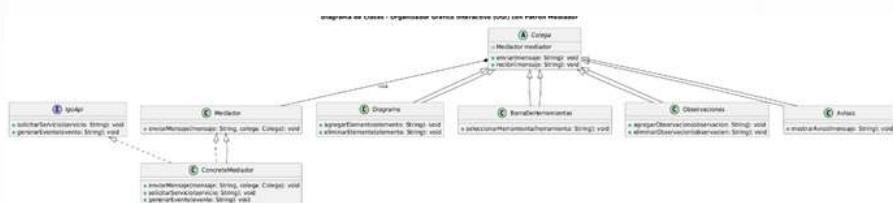
Gonzales Gonzales, C. E. (2023). Análisis comparativo de patrones de diseño MVC y MVP para el rendimiento de aplicaciones web. <https://acortar.link/NQkXGw>

# Desarrollo de componentes de software en base al patrón de diseño mediador: el caso del organizador gráfico interactivo

El documento trata sobre la aplicación del patrón de diseño Mediador en el rediseño de una herramienta educativa llamada Organizador Gráfico Interactivo (OGI). Este organizador, inicialmente desarrollado en Adobe Flash, presentaba problemas de alto acoplamiento entre sus elementos gráficos, lo que dificultaba la actualización y reutilización del software. Para mejorar la modularidad y reducir el acoplamiento, se utilizó el patrón Mediador, permitiendo que los distintos componentes del OGI (como las capas de diagramas, herramientas y observaciones) se comunicaran a través de un mediador central. Este rediseño facilitó la interacción con otros componentes, permitiendo al OGI funcionar de manera independiente o en conjunto con otras aplicaciones.

## Reflexión

El uso del patrón Mediador en el OGI muestra cómo un patrón de diseño puede resolver problemas estructurales y mejorar la calidad y adaptabilidad de un software. Al reducir el acoplamiento entre componentes, el patrón Mediador permite que cada parte de la aplicación se modifique y se mejore sin afectar al sistema completo. Además, el enfoque en una arquitectura de componentes abre la puerta a la reutilización y extensión del OGI en otros contextos, mejorando su longevidad y valor educativo. Esta experiencia resalta la importancia de elegir patrones de diseño que se adapten a las necesidades de modularidad y flexibilidad en proyectos educativos o de desarrollo complejo.



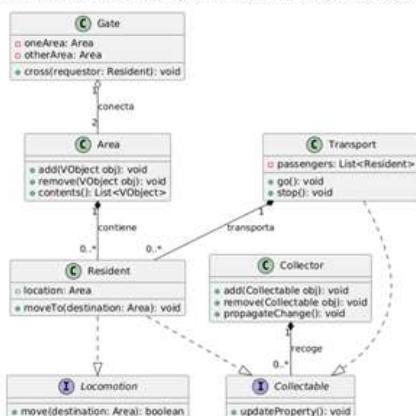
## Bibliografía

- Almarza, F. A., Ponce, H. R., & López, M. J. (2010). Desarrollo de componentes de software en base al patrón de diseño mediador: el caso del organizador gráfico interactivo. In J. Sánchez, Congreso Iberoamericano de Informática Educativa (pp. 571-578). <https://acortar.link/UzjjnW>

# Patrones de Diseño para Mundos Virtuales Orientados a Objetos (MOO's)

El documento explora el uso de patrones de diseño para construir \*\*Mundos Virtuales Orientados a Objetos (MOOs)\*\*, proporcionando un lenguaje de patrones específico para abordar problemas comunes en este contexto. Los patrones propuestos, como \*\*Área\*\*, \*\*Locomoción\*\*, \*\*Puerta\*\* y \*\*Transporte\*\*, ayudan a organizar espacios virtuales, manejar la movilidad de los objetos y definir interacciones y relaciones entre estos componentes. Estos patrones permiten estructurar la navegación, la interactividad y la disposición espacial en mundos virtuales, y pueden aplicarse en ámbitos como la educación, la simulación y el entretenimiento.

Diagrama de Clases - Patrones de Diseño para Mundos Virtuales Orientados a Objetos (I)



## Bibliografía

Martínez, F. C., Miguez, A. A., Fernández, A., & Díaz, A. (2000). Patrones de diseño para mundos virtuales orientados a objetos (MOO's). In VI Congreso Argentino de Ciencias de la Computación. <https://acortar.link/6gNFax>

## Reflexión

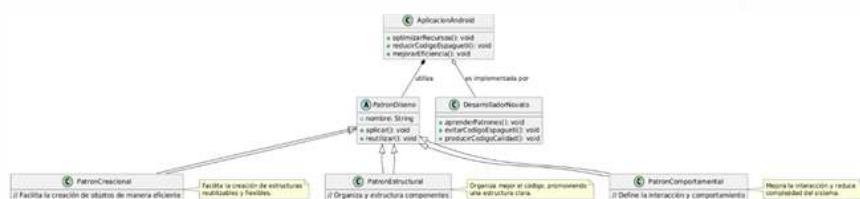
La propuesta de un lenguaje de patrones específico para mundos virtuales orientados a objetos evidencia cómo los patrones de diseño ayudan a simplificar y organizar complejidades inherentes en estos entornos. La reutilización de soluciones estructuradas permite a los diseñadores mejorar la eficiencia, la claridad y la cohesión en la construcción de estos mundos. Además, al proporcionar un lenguaje común, los patrones facilitan la comunicación entre equipos y permiten que los desarrolladores se beneficien del conocimiento colectivo. Esta metodología fomenta un diseño intuitivo y escalable, en el cual la experiencia del usuario y la lógica interna del sistema se optimizan, haciendo los mundos virtuales más accesibles y funcionales.

# Clasificación de los patrones de diseño idóneos en programación Android

Este artículo enfatiza el valor de los patrones de diseño en el desarrollo de aplicaciones móviles en Android, una plataforma que requiere soluciones eficientes debido a sus limitaciones de hardware. Inspirados en la arquitectura y popularizados por el grupo Gang of Four, los patrones de diseño brindan soluciones probadas para problemas comunes en programación, ayudando a los desarrolladores a evitar errores como el "código espagueti". Los programadores novatos suelen carecer de experiencia en estas prácticas, lo cual puede resultar en código redundante y difícil de mantener. La investigación propone un catálogo de patrones específicos para Android, que oriente a los nuevos desarrolladores hacia un código más estructurado y de alta calidad.

## Reflexión

Este trabajo muestra cómo los patrones de diseño son esenciales para guiar a los desarrolladores, especialmente a los nuevos, en la creación de aplicaciones robustas y eficientes. Los patrones no solo resuelven problemas técnicos, sino que también fomentan buenas prácticas de programación, evitando errores comunes y facilitando el mantenimiento. En una plataforma como Android, la implementación de patrones de diseño permite optimizar el uso de recursos y crear aplicaciones que cumplan con los estándares de calidad necesarios en un entorno de rápida evolución.



## Bibliografía

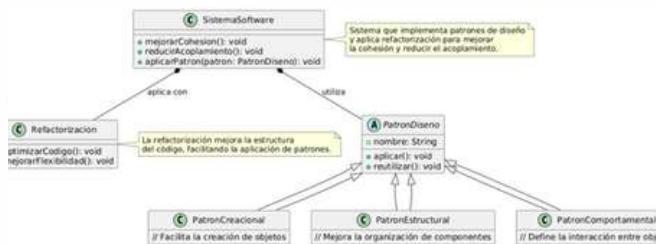
- Astorga, J. A. I., Tirado, J. L. O., Ramírez, R. U. R., Mendoza, J. C. L., & Garzón, A. P. (2019). Clasificación de los patrones de diseño idóneos en programación Android. Revista Digital de Tecnologías Informáticas y Sistemas, 3(1). <https://acortar.link/rQMZST>

# Patrones de diseño: una herramienta de desarrollo de software y su aplicación

Los patrones de diseño ofrecen soluciones reutilizables y eficientes a problemas de diseño de software que suelen surgir repetidamente en el desarrollo de aplicaciones. Estos patrones se enfocan en la creación e interacción de objetos, y se basan en principios fundamentales de la programación orientada a objetos, como la alta cohesión, el bajo acoplamiento y el uso de interfaces. Más allá de la herencia y el polimorfismo, aplicar correctamente los patrones de diseño requiere comprender conceptos como la composición sobre la herencia. Además, la refactorización juega un papel clave en la preparación del código para recibir patrones de diseño, ya que permite organizar el código y mejorar su estructura para lograr mayor flexibilidad y claridad.

## Reflexión

Este texto destaca que los patrones de diseño no son solo soluciones técnicas, sino prácticas maduras que reflejan los principios más sólidos de la programación orientada a objetos. La alta cohesión, el bajo acoplamiento y el uso de interfaces permiten construir software modular y adaptativo, y los patrones de diseño son una herramienta que facilita este objetivo. Además, la refactorización emerge como un primer paso crucial, permitiendo a los desarrolladores adaptar y mejorar su código para la introducción de patrones de diseño. La comprensión de estos principios y su integración a través de patrones permiten a los desarrolladores crear sistemas de software más sólidos, mantenibles y preparados para evolucionar.



## Bibliografía

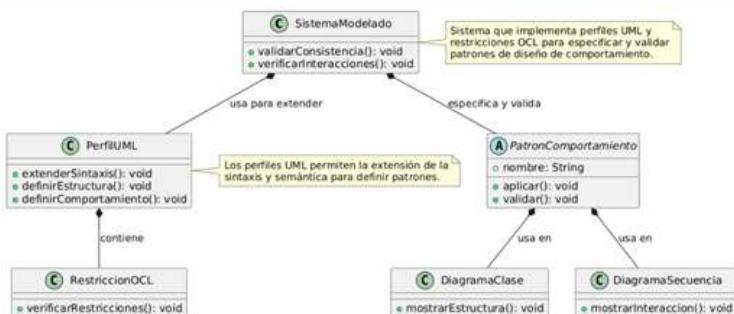
Gómez Placencia, M. (2005). Patrones de diseño: una herramienta de desarrollo de software y su aplicación. <https://acortar.link/cMKUiI>

# PERFILES UML PARA LA DEFINICIÓN DE PATRONES DE DISEÑO DE COMPORTAMIENTO

Este trabajo propone un enfoque para especificar y validar patrones de diseño de comportamiento utilizando perfiles UML y restricciones en OCL. Los perfiles UML extienden la sintaxis y semántica del lenguaje UML, permitiendo definir patrones aplicables tanto en modelos estructurales como de comportamiento. El estudio implementa una arquitectura de tres niveles para organizar estos patrones y los valida en la herramienta Rational Software Developer (RSA) de IBM. Como resultado, se logró una especificación clara de cada patrón, facilitando la verificación automática de la consistencia y la interacción en los modelos.

## Reflexión

Este trabajo muestra cómo los patrones de diseño pueden formalizarse y estandarizarse mediante perfiles UML y restricciones OCL. En lugar de aplicar patrones de forma manual, el uso de UML y OCL permite formalizar y verificar su implementación con precisión. Esta automatización asegura que los modelos de software cumplan con estándares de consistencia y funcionalidad, validando tanto la estructura como el comportamiento del sistema. El enfoque demuestra cómo integrar los patrones de diseño en herramientas de modelado, facilitando la verificación de calidad y reduciendo errores en sistemas complejos.



## Bibliografía

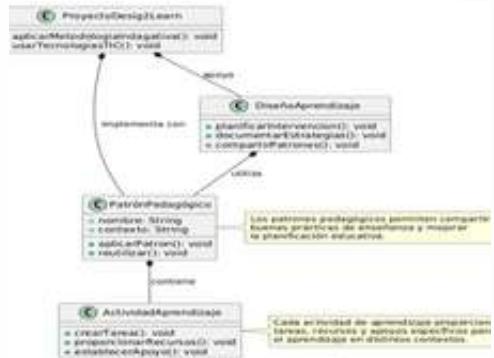
Cortez, A., Riesco, D. E., & Garis, A. G. (2012). Perfiles UML para la definición de patrones de diseño de comportamiento. In XIV Workshop de Investigadores en Ciencias de la Computación. <https://acortar.link/9KIDNr>

# Los patrones de diseño como herramientas para guiar la práctica del profesorado

Este artículo explora el uso de patrones pedagógicos en el diseño del aprendizaje, con el objetivo de capturar y compartir buenas prácticas educativas aplicables en diferentes contextos. Estos patrones buscan documentar las estrategias de los docentes para mejorar la planificación y crear intervenciones educativas más efectivas. Inspirados en los patrones de diseño en software, ofrecen una estructura reutilizable que ayuda a los educadores a desarrollar actividades de aprendizaje adaptadas a contextos específicos. El proyecto "Desig2Learn" aplica estos patrones en el ámbito universitario, utilizando tecnologías y metodologías basadas en el aprendizaje indagativo.

## Bibliografía

- : Gros Salvat, B., Escofet Roig, A., & Marimon Martí, M. (2016). Los patrones de diseño como herramientas para guiar la práctica del profesorado. Revista Latinoamericana de Tecnología Educativa-RELATEC, 2016, vol. 15, num. 3, p. 11-25. <https://acortar.link/NyFioi>



## Reflexión

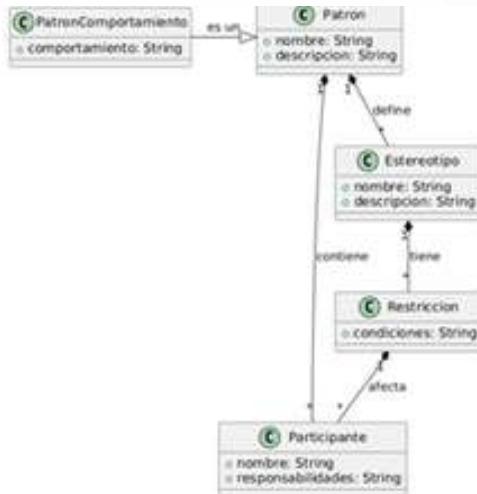
Este trabajo muestra cómo los patrones de diseño pueden formalizarse y estandarizarse mediante perfiles UML y restricciones OCL. En lugar de aplicar patrones de forma manual, el uso de UML y OCL permite formalizar y verificar su implementación con precisión. Esta automatización asegura que los modelos de software cumplan con estándares de consistencia y funcionalidad, validando tanto la estructura como el comportamiento del sistema. El enfoque demuestra cómo integrar los patrones de diseño en herramientas de modelado, facilitando la verificación de calidad y reduciendo errores en sistemas complejos.

# Formalización de patrones de diseño de comportamiento

Este trabajo explora el uso de perfiles UML y restricciones OCL para especificar y estandarizar patrones de comportamiento en el desarrollo de software, basados en la clasificación de GoF. Los patrones de diseño de comportamiento gestionan interacciones complejas entre objetos. Mediante perfiles UML, se propone un marco para definir estos patrones de manera precisa y reutilizable, aprovechando las herramientas UML existentes. La investigación establece una arquitectura de tres niveles: general para todos los patrones, intermedio con categorías (estructurales, creacionales, de comportamiento), y específico para cada patrón, incluyendo la identificación de clases y objetos clave, la definición de estereotipos y la aplicación de restricciones para asegurar el comportamiento deseado.

## Bibliografía

Cortez, A., Garis, A. G., & Riesco, D. E. (2011). Formalización de patrones de diseño de comportamiento. In XIII Workshop de Investigadores en Ciencias de la Computación. <https://acortar.link/h7UojC>

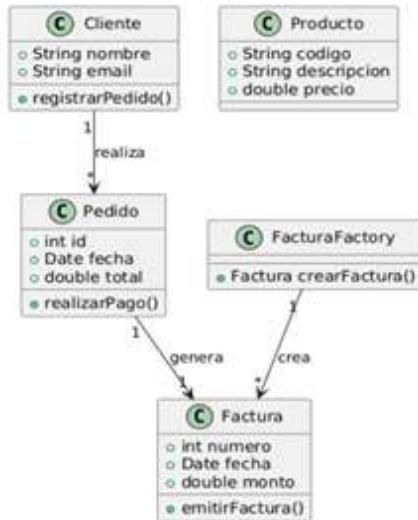


## Reflexión

Este trabajo muestra cómo los perfiles UML y las restricciones OCL pueden formalizar y estandarizar los patrones de diseño. Esta metodología hace que los patrones de comportamiento sean más accesibles y precisos, permitiendo a los desarrolladores aplicarlos de manera consistente. Además, aprovecha UML para representar y validar patrones complejos sin necesidad de crear nuevas herramientas. La arquitectura en niveles propuesta no solo facilita la especificación de patrones, sino que también promueve su reutilización y claridad, ayudando a crear software con una estructura sólida y organizada.

# Análisis de la adecuación de lenguajes de programación Web a un desarrollo basado en patrones de diseño J2EE de alto nivel

Este trabajo compara tecnologías como \*\*ASP.NET/C#\*\*, \*\*PHP\*\* y \*\*Zend Framework\*\* frente a \*\*J2EE\*\* para el desarrollo de aplicaciones web empresariales. El análisis se enfoca en cómo cada una de estas tecnologías implementa los patrones de diseño J2EE, los cuales representan buenas prácticas ampliamente utilizadas para construir sistemas web estandarizados, comprensibles y mantenibles. Además, se crea un mapeo entre las diferentes fuentes de patrones J2EE, lo que facilita su comprensión y fomenta una discusión abierta sobre sus aplicaciones en distintas tecnologías.



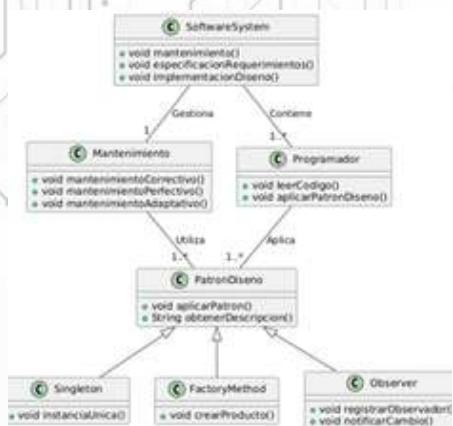
## Reflexión

El trabajo destaca la importancia de los patrones de diseño en el desarrollo de software, especialmente para aplicaciones web, ya que proporcionan soluciones estandarizadas que mejoran la escalabilidad, mantenibilidad y comprensión del código. Al comparar tecnologías como ASP.NET, PHP y Zend Framework, se observa que todas pueden implementar estos patrones, aunque con sus particularidades. La elección de la tecnología adecuada depende no solo de factores como la curva de aprendizaje o el rendimiento, sino también de cómo facilita la implementación de buenas prácticas. En resumen, los patrones de diseño son esenciales para crear aplicaciones robustas y de alta calidad, independientemente de la tecnología empleada.

## Bibliografía

Morales Franco, Ó. M. (2009). Análisis de la adecuación de lenguajes de programación Web a un desarrollo basado en patrones de diseño J2EE de alto nivel. <https://acortar.link/3JF4uE>

# Análisis del impacto del uso de patrones de diseño en la fase de mantenimiento



## Reflexión

Los patrones de diseño juegan un papel fundamental en la mejora de la calidad del software, especialmente en lo que respecta a su mantenimiento a largo plazo. Al ser soluciones ya probadas y estandarizadas para problemas recurrentes en el desarrollo de software, estos patrones permiten que el código sea más comprensible, modular y fácil de modificar. Esto no solo facilita la tarea de los programadores cuando se requiere hacer ajustes o mejoras en el sistema, sino que también reduce el tiempo necesario para entender y corregir posibles errores. La investigación que se lleva a cabo en este estudio resalta la importancia de los patrones de diseño en la escalabilidad y evolución de los sistemas, ya que, en entornos de software cambiantes, la capacidad de adaptarse a nuevas necesidades o tecnologías es crucial.

El mantenimiento de software es esencial para asegurar que los sistemas sigan cumpliendo con las necesidades de los usuarios a lo largo del tiempo. Se identifican tres tipos de mantenimiento: correctivo, perfectivo y adaptativo. El uso de patrones de diseño, que son soluciones probadas para problemas comunes, facilita este mantenimiento al mejorar la comprensión del código y reducir el tiempo dedicado a las modificaciones.

La investigación de estudiantes de la Universidad Distrital analiza el uso de los 23 patrones de diseño de GoF en empresas de software en Bogotá. Mediante una encuesta aplicada a 100 proyectos, se busca evaluar su efectividad en términos de escalabilidad y evolución del software, además de proponer un modelo arquitectónico para un servicio web de evaluación de buenas prácticas.

## Bibliografía

De León Correa, M. J., Hoyos Beltrán, I. R., & Quintero Díaz Granados, F. E. Análisis del impacto del uso de patrones de diseño en la fase de mantenimiento.

<https://acortar.link/ij8hCz>

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

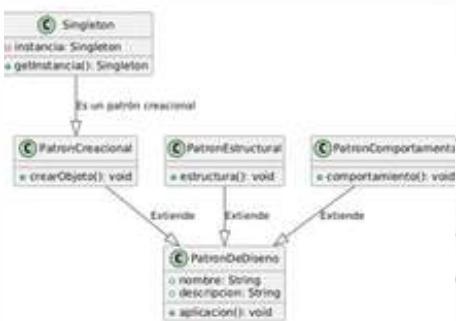
En la búsqueda por mejorar la calidad del software, los patrones de diseño se presentan como una estrategia clave, ofreciendo soluciones reutilizables a problemas comunes en el desarrollo. Estos patrones no solo promueven la reutilización de código, sino que también crean soluciones estructurales eficientes. Plataformas como Microsoft con su "Enterprise Library" y Java con sus complementos para la JVM han desarrollado sus propios patrones. Sin embargo, estos patrones pueden ser difíciles de comprender. Este proyecto busca resolver este problema proporcionando una guía clara sobre los patrones de diseño, basada en Design Patterns de Erich Gamma, utilizando UML, Java y MySQL como herramientas para exemplificar su aplicación.

## Bibliografía

Aldas Mena, D. E., & Andrade Cadena, M. A. (2011). Guía práctica para el uso de patrones de diseño en el desarrollo de software (Bachelor's thesis, QUITO/EPN/2011).  
<https://acortar.link/jMoRvZ>

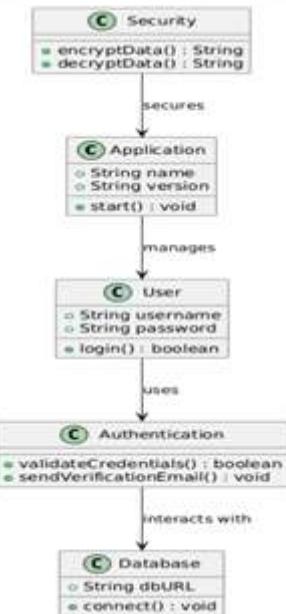
## Reflexión

Los patrones de diseño son herramientas clave para garantizar que el software sea flexible, fácil de mantener y escalable. Al ofrecer soluciones probadas para problemas recurrentes, estos patrones permiten a los desarrolladores adaptarlas a sus necesidades, mejorando la calidad y eficiencia del trabajo. Además, facilitan un lenguaje común entre los desarrolladores, promoviendo la colaboración y la comprensión del código. Sin embargo, es crucial evitar que los patrones se conviertan en elementos opacos o cajas negras, como puede suceder con las implementaciones de plataformas populares, para asegurar su efectividad y comprensión.



# Incidencia de los patrones de diseño de software en la seguridad de aplicaciones web

El archivo resalta cómo los \*\*patrones de diseño de software\*\* son fundamentales no solo para mejorar la estructura y la organización del código, sino también para incrementar la \*\*seguridad\*\* de las aplicaciones web. Estos patrones, al ser soluciones probadas para problemas comunes, son herramientas valiosas que permiten a los desarrolladores diseñar aplicaciones más robustas frente a amenazas. A través de prácticas como la \*\*validación de datos\*\*, \*\*autenticación segura\*\* y la \*\*segregación de responsabilidades\*\*, los patrones de diseño ayudan a identificar y mitigar vulnerabilidades en las aplicaciones web, reduciendo la posibilidad de ataques. Además, al usar estos patrones, las aplicaciones se vuelven más fáciles de mantener y actualizar, disminuyendo los costos asociados a la gestión de la seguridad a largo plazo.



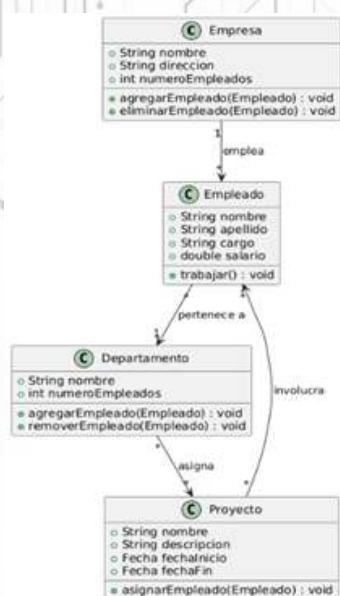
## Reflexión

Los patrones de diseño no solo mejoran la eficiencia y organización del código, sino que también son cruciales para la seguridad de las aplicaciones. Al utilizar soluciones estructuradas y probadas, los desarrolladores no solo resuelven problemas de diseño, sino que pueden anticipar y mitigar vulnerabilidades antes de que se conviertan en amenazas. Estos patrones ayudan a crear aplicaciones más resistentes a los ataques y facilitan su mantenimiento frente a nuevas amenazas o actualizaciones necesarias. Así, los patrones de diseño no solo mejoran la calidad del software, sino que también incrementan su seguridad, reduciendo el riesgo de fallos de seguridad.

## Bibliografía

Mesías-Valencia, J. J., & Cevallos-Muñoz, F. D. (2024). Incidencia de los patrones de diseño de software en la seguridad de aplicaciones web. *MQRIInvestigar*, 8(1), 236-259. <https://acortar.link/4qFwJY>

# Aplicación de patrones de diseño estructural para el modelamiento de clases de los sistemas empresariales



Este trabajo de investigación se centra en los patrones de diseño aplicados al modelado de clases en sistemas empresariales, reconociendo la creciente complejidad de los sistemas modernos. Dado que los patrones generales no cubren adecuadamente todos los problemas de diseño, especialmente en sistemas de información administrativa, el estudio propone patrones más específicos. Realizado con 76 alumnos de la Universidad Católica de Santa María, el enfoque cuantitativo y preexperimental incluyó encuestas y análisis estadísticos para validar la relación positiva entre la aplicación de estos nuevos patrones y la mejora en el modelado de datos. Los resultados indican que los patrones desarrollados son eficaces en el diseño y modelado de sistemas empresariales.

## Reflexión

Este estudio resalta la necesidad de adaptar los patrones de diseño a contextos específicos como los sistemas empresariales. Aunque los patrones generales son útiles, la especificidad mejora la eficiencia en el desarrollo. Los resultados muestran que los patrones creados son efectivos en el modelado de datos y tienen un impacto positivo en aplicaciones reales, especialmente en sistemas de información administrativa, subrayando la importancia de seguir evolucionando los patrones de diseño para enfrentar desafíos emergentes.

## Bibliografía

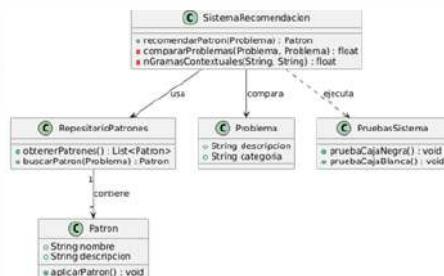
Zúñiga Carnero, M. M. (2020). Aplicación de patrones de diseño estructurales para el modelamiento de clases de los sistemas empresariales. <https://acortar.link/bejZeM>

# PATRÓN DE DISEÑO; RECURSOS EDUCATIVOS ABIERTOS; SISTEMAS DE RECOMENDACIÓN

Con el crecimiento de las Tecnologías de la Información y las Comunicaciones en la educación, los Recursos Educativos Abiertos (REA) han ganado popularidad, pero muchos carecen de calidad pedagógica. Para abordar este problema, los diseñadores de REA han adoptado el uso de patrones de diseño. Este estudio propone un sistema de recomendación que ayuda a seleccionar los patrones adecuados al comparar problemas de diseño con una base de datos de patrones utilizando n-gramas contextuales. Las pruebas realizadas mostraron que el sistema funciona correctamente, y la evaluación de satisfacción indicó resultados positivos en cuanto a su efectividad.

## Bibliografía

Arteaga Gómez, Y., & Menéndez Martínez, Y. (2015). Sistema de recomendación de patrones de diseño para Recursos Educativos Abiertos (Bachelor's thesis, Universidad de las Ciencias Informáticas. Facultad 4). <https://acortar.link/H4y5qf>



## Reflexión

El desarrollo de un sistema de recomendación de patrones de diseño para REA subraya la importancia de orientar a los diseñadores en la elección de las herramientas adecuadas, mejorando tanto la calidad como la reutilización de los recursos educativos. La investigación demuestra cómo los patrones de diseño no solo resuelven problemas comunes, sino que también garantizan la consistencia pedagógica de los REA, facilitando su reutilización de manera más efectiva. El uso de técnicas avanzadas de comparación textual, como los n-gramas contextuales, mejora la precisión de las recomendaciones, adaptándose a las necesidades específicas de cada proyecto. Así, se destaca el potencial de los patrones de diseño como un recurso valioso para la mejora continua de los REA, promoviendo una educación accesible y de calidad.

# Integración de proyectos de software mediante el uso de patrones y arquitecturas de desarrollo

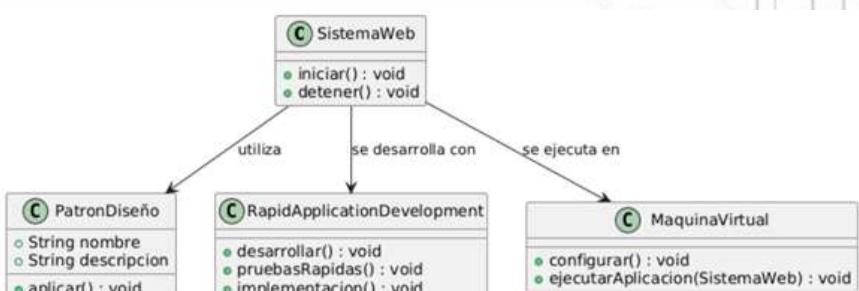
Este trabajo busca servir como una guía para quienes desean dedicarse al desarrollo y diseño de aplicaciones web, utilizando casos de estudio y un sistema web previamente desarrollado para exemplificar buenas prácticas. Además de enseñar el uso de patrones de diseño y técnicas avanzadas, el enfoque incluye el empleo de máquinas virtuales para ejecutar aplicaciones, eliminando la necesidad de configurar un servidor HTTP local. La metodología también abarca el proceso RAD (Rapid Application Development), una estrategia que permite acelerar el desarrollo de aplicaciones utilizando el lenguaje PHP, priorizando la rapidez y la eficiencia.

## Bibliografía

Moreno López, E. A. (2013). Integración de proyectos de software mediante el uso de patrones y arquitecturas de desarrollo (Doctoral dissertation). <https://acortar.link/gDibUU>

## Reflexión

La investigación resalta cómo los patrones de diseño son esenciales para estructurar y organizar el código, lo que resulta clave en el desarrollo de aplicaciones web. Al combinarse con técnicas como RAD, se logra una integración eficiente entre rapidez y buena arquitectura en el software. Los patrones no solo resuelven problemas comunes, sino que proporcionan un marco que favorece la escalabilidad y el mantenimiento a largo plazo. Además, el uso de máquinas virtuales permite a los desarrolladores enfocarse en el código y el diseño sin preocuparse por la configuración de servidores. Esta combinación de patrones de diseño, RAD y virtualización ofrece un enfoque moderno y eficaz, facilitando la creación de aplicaciones robustas, escalables y de rápido desarrollo.

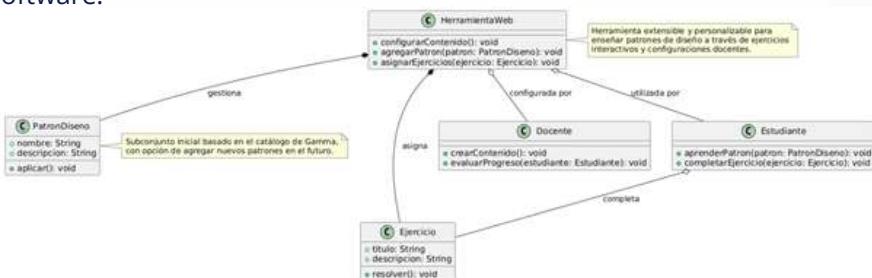


# Desarrollo de una herramienta para el aprendizaje de patrones de diseño software

Este trabajo desarrolla una herramienta web diseñada para enseñar y poner en práctica patrones de diseño orientados a objetos, especialmente útil en contextos educativos. La aplicación permite a los docentes o formadores configurar ejercicios y contenido interactivo para que los estudiantes aprendan a reconocer, aplicar e implementar patrones de diseño. Inicialmente, la herramienta incluirá un subconjunto de patrones del catálogo de Gamma, pero se ha diseñado para ser extensible, permitiendo agregar nuevos patrones y contenido en el futuro. Con esta herramienta, se busca facilitar el aprendizaje práctico de los patrones, promoviendo su correcta aplicación en el desarrollo de software.

## Reflexión

Esta herramienta resalta el papel fundamental de los patrones de diseño en la educación de la programación orientada a objetos. Su capacidad para enseñar de manera interactiva y personalizable refleja la importancia de acercar a los estudiantes a soluciones probadas y reutilizables que mejoren la calidad del software. Además, la extensibilidad de la herramienta asegura su relevancia a largo plazo, adaptándose a nuevas necesidades pedagógicas. Proyectos como este demuestran cómo la tecnología puede facilitar el aprendizaje de conceptos complejos, haciendo que los patrones de diseño sean más accesibles y comprensibles para futuras generaciones de desarrolladores.

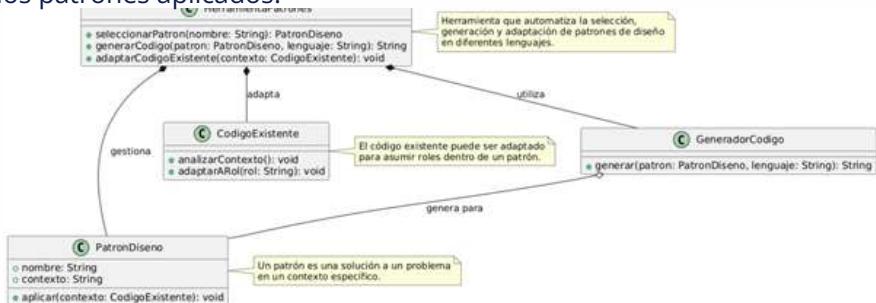


## Bibliografía

Ferrandis Homsi, A. (2021). Desarrollo de una herramienta para el aprendizaje de patrones de diseño software (Doctoral dissertation, Universitat Politècnica de València). <https://acortar.link/GEszec>

# Una integración de patrones de diseño en procesos de ingeniería forward de modelos estáticos UML

Los patrones de diseño proporcionan soluciones probadas a problemas recurrentes en el desarrollo de software, mejorando la comunicación y reduciendo los tiempos de desarrollo. Sin embargo, su implementación manual es a menudo tediosa, propensa a errores y puede dificultar la trazabilidad. Muchas herramientas actuales ofrecen soporte limitado, con procesos básicos como el "corte y pega" de código, que luego debe ajustarse manualmente. Además, estas herramientas no son independientes del lenguaje ni permiten generar código en múltiples lenguajes. Dado que los patrones rara vez existen de forma aislada, su implementación implica no solo crear nuevas clases y métodos, sino también adaptar el código existente para cumplir nuevos roles dentro de los patrones aplicados.



## Bibliografía

- Martinez, L., & Favre, L. M. (2004). Una integración de patrones de diseño en procesos de ingeniería forward de modelos estáticos UML. In VI Workshop de Investigadores en Ciencias de la Computación. <https://acortar.link/i9ZMvb>

## Reflexión

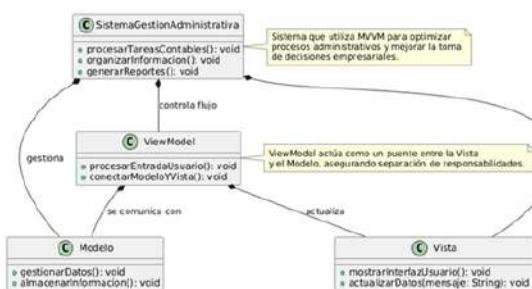
Este análisis subraya la necesidad de avanzar hacia herramientas más sofisticadas que automaticen o asistan significativamente en la implementación de patrones de diseño. Si bien los patrones estructuran el código y optimizan la colaboración entre desarrolladores, permitiendo soluciones modulares y escalables, la dependencia de procesos manuales limita su potencial. Esto resalta la importancia de desarrollar entornos independientes del lenguaje, que integren patrones con flexibilidad y adaptabilidad al contexto. Un enfoque de este tipo podría revolucionar la forma en que los patrones se aplican en proyectos, promoviendo un diseño más eficiente y alineado con las necesidades reales del software.

# Sistema web bajo el patrón de diseño de software MVVM en los procesos de gestión administrativa de la radio interoceánica

Este trabajo se centró en desarrollar un sistema de gestión administrativa para mejorar la eficiencia de los procesos administrativos en la Radio Interoceánica. Utilizando el patrón de diseño MVVM (Model-View-ViewModel), el sistema optimizó tareas contables, organizó información y mejoró la toma de decisiones empresariales. Esto resultó en mayor rendimiento, satisfacción del cliente y claridad en los resultados empresariales. La investigación se apoyó en una metodología cualitativa basada en entrevistas al personal gerencial, y para el desarrollo técnico se empleó la metodología UWE junto con la tecnología MEAN Stack (basada en JavaScript), asegurando un diseño moderno y eficiente.

## Reflexión

Este trabajo evidencia cómo los patrones de diseño, como MVVM, pueden transformar la gestión administrativa al crear sistemas estructurados y claros que mejoran la productividad y la toma de decisiones. MVVM separa la lógica del negocio de la interfaz gráfica, lo que facilita el mantenimiento y la escalabilidad del sistema. Al integrar herramientas modernas como MEAN Stack, el proyecto demuestra cómo combinar patrones de diseño con tecnologías actuales puede generar soluciones innovadoras y efectivas para las necesidades empresariales. Esto refuerza la importancia de los patrones en el desarrollo de software como guías para resolver problemas complejos con estructuras reutilizables y probadas.



## Bibliografía

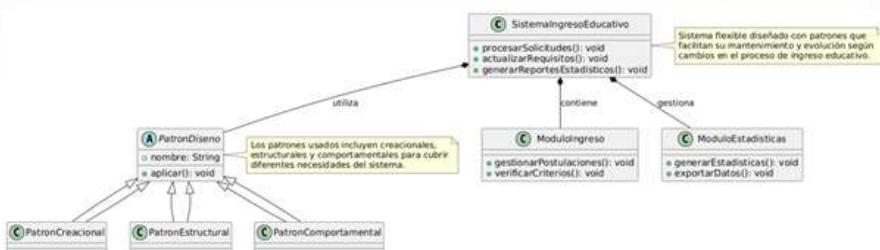
Rodríguez Cabrera, R. F. (2019). Sistema web bajo el patrón de diseño de software MVVM en los procesos de gestión administrativa de la radio interoceánica. <https://acortar.link/LerMk4>

# Sistema web bajo el patrón de diseño de software MVVM en los procesos de gestión administrativa de la radio interoceánica

Este trabajo se centró en desarrollar un sistema de gestión administrativa para mejorar la eficiencia de los procesos administrativos en la Radio Interoceánica. Utilizando el patrón de diseño MVVM (Model-View-ViewModel), el sistema optimizó tareas contables, organizó información y mejoró la toma de decisiones empresariales. Esto resultó en mayor rendimiento, satisfacción del cliente y claridad en los resultados empresariales. La investigación se apoyó en una metodología cualitativa basada en entrevistas al personal gerencial, y para el desarrollo técnico se empleó la metodología UWE junto con la tecnología MEAN Stack (basada en JavaScript), asegurando un diseño moderno y eficiente.

## Reflexión

Este caso evidencia cómo los patrones de diseño no solo optimizan el desarrollo de software, sino que también aseguran su capacidad de adaptación frente a escenarios cambiantes. Los patrones creacionales, estructurales y comportamentales proporcionan una base sólida para diseñar sistemas modulares, claros y fáciles de mantener. En un ámbito tan dinámico como la gestión educativa, contar con un sistema flexible resulta clave para responder a las necesidades de estudiantes y autoridades. Este trabajo refuerza la idea de que los patrones de diseño no son solo herramientas técnicas, sino que también son estrategias que ayudan a construir sistemas resilientes y preparados para el futuro.



## Bibliografía

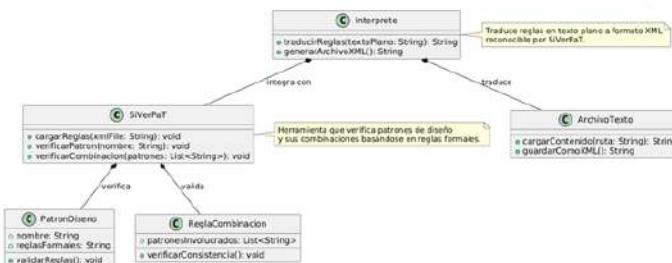
- García Ojalvo, I., Sepúlveda Lima, R., & Ugalde, A. (2022). Patrones de diseño en el modelo computacional del proceso de ingreso a la educación superior cubana. Revista San Gregorio, 1(51), 1-14. <https://acortar.link/FL4rQx>

# Herramienta de Soporte al Modelado de Software por Combinación de Patrones de Diseño

Esta tesis amplía la funcionalidad del sistema SiVerPaT, una herramienta diseñada para verificar la correcta aplicación y combinación de patrones de diseño en software. Originalmente, SiVerPaT soportaba los patrones Iterator, Composite y Observer. En esta investigación, se integran nuevos patrones como Strategy, Composite, Factory Method y Template Method, junto con sus reglas de combinación, utilizando un intérprete desarrollado para este propósito. Este intérprete traduce reglas formales desde un archivo de texto plano a un formato XML reconocido por SiVerPaT. Las pruebas realizadas demuestran que el intérprete funciona correctamente, lo que permite verificar las combinaciones de patrones en proyectos de software, asegurando su correcta implementación.

## Reflexión

Este trabajo destaca la importancia de herramientas como SiVerPaT para automatizar y garantizar la correcta aplicación de patrones de diseño en software. La capacidad de verificar combinaciones de patrones es un avance significativo, ya que estos no suelen implementarse de manera aislada. Integrar patrones como Strategy y Factory Method resalta la necesidad de manejar interacciones complejas entre ellos, evitando errores y garantizando soluciones más robustas y estructuradas. Este enfoque no solo ayuda a los desarrolladores a aplicar patrones correctamente, sino que también promueve la creación de software más modular, escalable y mantenable.



## Bibliografía

Ibanez Ocampo, M. B. (2016). Herramienta de Soporte al Modelado de Software por Combinación de Patrones de Diseño. <https://acortar.link/BHNCU6>

# Patrones de diseño para la construcción de cursos on-line en un entorno virtual de aprendizaje

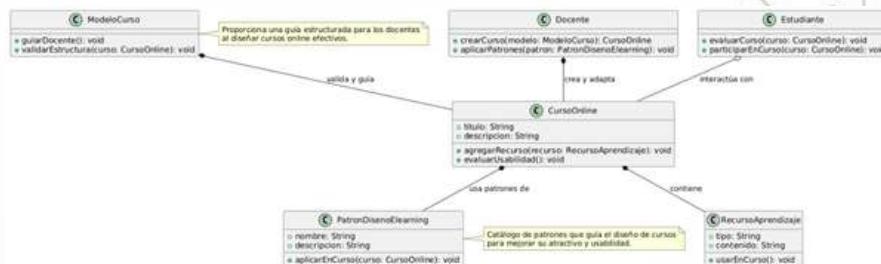
Este artículo aborda la problemática de los cursos online poco atractivos en entornos virtuales de aprendizaje (EVAs), debido a la falta de prácticas adecuadas de diseño. Se proponen patrones de diseño específicos para e-learning junto con un enfoque pedagógico adaptado, y se presenta un catálogo de diez patrones y un modelo de curso online como guía para los docentes. Las pruebas con docentes y estudiantes demostraron una mejora en la usabilidad y atractividad de los cursos creados con estos patrones.

## Bibliografía

Pástor, D., Jiménez, J., Arcos, G., Romero, M., & Urquiza, L. (2018). Patrones de diseño para la construcción de cursos on-line en un entorno virtual de aprendizaje. *Ingeniare. Revista chilena de ingeniería*, 26(1), 157-171. <https://acortar.link/x5M2SR>

## Reflexión

Este trabajo evidencia cómo los patrones de diseño pueden trascender el ámbito técnico y aplicarse en áreas como la educación virtual, ofreciendo soluciones estructuradas y reutilizables a problemas recurrentes. Los patrones de diseño e-learning ayudan a los docentes a crear cursos que no solo son pedagógicamente sólidos, sino también más atractivos y utilizables para los estudiantes. Este enfoque fomenta un aprendizaje más efectivo y demuestra cómo los principios de diseño pueden adaptarse a diferentes contextos para mejorar experiencias tanto para creadores como para usuarios. Además, el uso de patrones reduce la improvisación y garantiza una calidad consistente en los cursos.



# Módulo de Gestión de Patrones de Diseño para EGPat con soporte para interoperabilidad

Este trabajo describe el desarrollo de un módulo para la gestión de patrones de diseño aplicados a recursos educativos. Su objetivo es prevenir errores comunes en el diseño de estos recursos y mejorar la organización y uso de patrones. El módulo permite gestionar patrones, sus colecciones, y facilita la interoperabilidad mediante exportación e importación de datos. Para su desarrollo, se usaron Python, Django, PyCharm, y PostgreSQL, asegurando compatibilidad técnica. Las pruebas de caja negra y caja blanca confirmaron que el módulo cumple con sus objetivos, proporcionando una herramienta robusta para la gestión de recursos educativos.

## Reflexión

Este trabajo muestra cómo los patrones de diseño no solo son clave en el desarrollo de software, sino también en la creación de recursos educativos. Al prevenir errores y estructurar soluciones, garantizan la calidad y consistencia del diseño. El módulo desarrollado resalta la importancia de la interoperabilidad, permitiendo que diferentes plataformas compartan y reutilicen patrones eficientemente. Este enfoque mejora el diseño de los recursos y promueve la colaboración y el intercambio de conocimiento entre instituciones educativas, beneficiando tanto a docentes como a estudiantes.



## Bibliografía

Menéndez Figueredo, M. D. L. C. (2022). Módulo de Gestión de Patrones de Diseño para EGPat con soporte para interoperabilidad (Bachelor's thesis, Universidad de las Ciencias Informáticas. Facultad 4.). <https://acortar.link/wSjdKo>

# Aprendizaje de patrones de diseño de microservicios mediante un juego serio

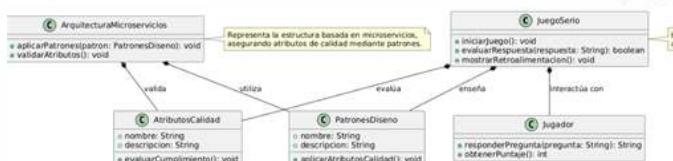
La adopción de arquitectura basada en microservicios presenta retos significativos, especialmente durante la fase de diseño, donde la selección de patrones de diseño y atributos de calidad juega un papel crucial. Este trabajo propone el diseño de un juego serio como herramienta de aprendizaje para enseñar patrones de diseño que cumplen con los atributos de calidad requeridos en arquitecturas de microservicios. Los juegos serios, además de hacer más accesible el conocimiento, permiten a los desarrolladores y estudiantes enfrentar los desafíos del diseño de microservicios de manera práctica e interactiva, promoviendo la aplicación correcta de patrones de diseño en proyectos de software.

## Reflexión

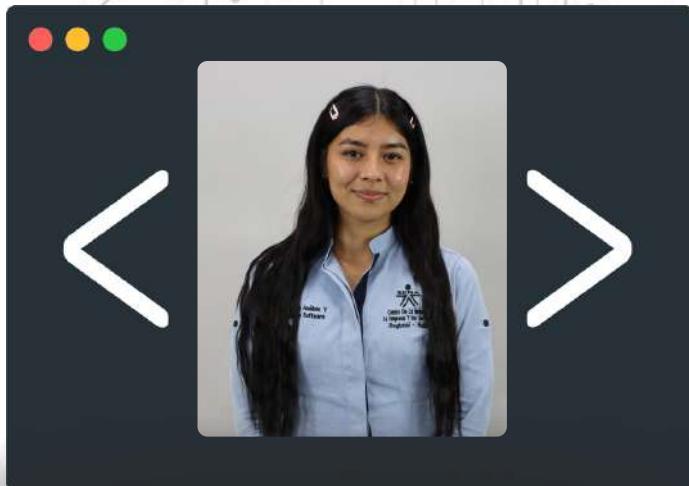
Este proyecto demuestra cómo los patrones de diseño pueden ser enseñados de forma innovadora mediante herramientas interactivas como los juegos serios. La enseñanza de patrones en el contexto de microservicios es particularmente valiosa, ya que estas arquitecturas requieren soluciones bien estructuradas para garantizar su calidad y escalabilidad. Este enfoque no solo fomenta una comprensión más profunda de los patrones, sino que también prepara a los desarrolladores para abordar desafíos reales en la industria. Al combinar aprendizaje y práctica, los juegos serios impulsan la correcta aplicación de patrones, mejorando tanto la calidad del software como las habilidades de los desarrolladores.

## Bibliografía

:Aguirre Valera, J. A. Aprendizaje de patrones de diseño de microservicios mediante un juego serio. <https://acortar.link/h0dTx>



# Laura Valentina Ariza Alejo



Me llamo Laura Valentina Ariza Alejo , tengo 19 años, soy una persona entusiasta y apasionada por aprender cosas nuevas, siempre en busca de oportunidades para crecer y mejorar en todos los aspectos de su vida.

Destaca por su capacidad para adaptarse a los cambios y enfrentar retos con una mentalidad positiva y creativa. Entre sus habilidades se encuentran la resolución de problemas, el compromiso con el trabajo bien hecho y la perseverancia para superar cualquier obstáculo.

Aunque aún tiene áreas que quiere perfeccionar, Valentina está constantemente esforzándose por mejorar, lo que la convierte en alguien confiable y capaz de aportar valor a cualquier equipo.

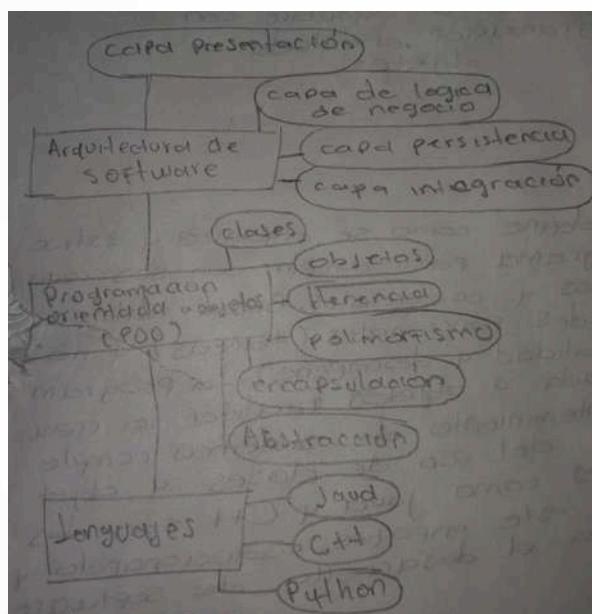
Su curiosidad natural y su enfoque en el aprendizaje continuo la impulsan a explorar nuevas ideas, adquirir nuevas habilidades y llevar sus proyectos al siguiente nivel.

# Arquitectura de software con programación orientada a objeto

La arquitectura de software define cómo se organiza y estructura un programa para cumplir con sus requisitos técnicos y comerciales. Está basada en componentes y sus relaciones, y busca satisfacer necesidades funcionales (tareas que el sistema debe realizar) y no funcionales (calidad y desempeño). La programación orientada a objetos, como paradigma popular, facilita la creación y mantenimiento de sistemas complejos a través del uso de clases y objetos. Lenguajes como Java y C++ son destacados en este enfoque, contribuyendo a la evolución y mejora continua en el desarrollo de software.

## Reflexión

La arquitectura de software es como un mapa detallado que guía la construcción de programas, asegurando que todas las partes encajen y funcionen bien juntas. La programación orientada a objetos (POO) agrega herramientas poderosas a este proceso, como las clases y los objetos, que permiten dividir tareas complejas en piezas más manejables. Los desarrolladores pueden crear soluciones rápidas y eficientes, abordando tanto necesidades técnicas como expectativas del usuario. En esencia, combinar una buena arquitectura con POO permite entregar software de mayor calidad en menos tiempo.



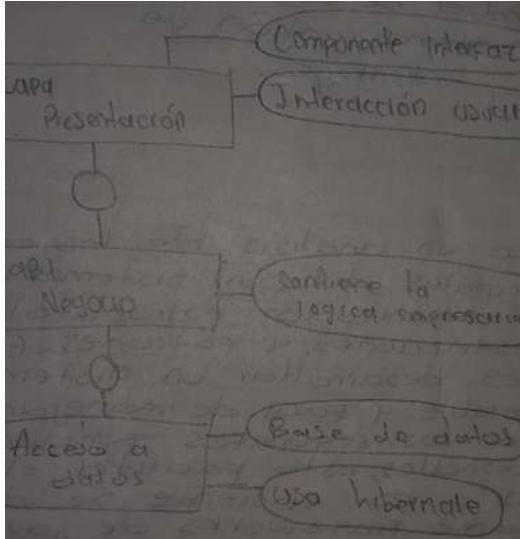
## Bibliografía

Bolaños, M., & Farinango, G. (2017). Análisis de Asimilación de la Programación Orientada a Aspectos (POA), en los Estudiantes de la Materia Aplicaciones en Ambientes https://bibdigital.epn.edu.ec/bitstream/15000/17322/1/CD-7817.pdf, pp.93. Arquitectura de software con programación orientada a objeto - Dialnet

# Arquitectura de software para sistema gestión de inventarios

Se realiza un análisis de los componentes principales del sistema, incluyendo su organización estructural, conectores, restricciones, y patrones arquitectónicos empleados. El objetivo es desarrollar un sistema modular, flexible, y testeable que cumpla con los requisitos funcionales y no funcionales, optimizando aspectos como la mantenibilidad y la eficiencia.

Se utilizan estilos arquitectónicos como la arquitectura en capas, y tecnologías como frameworks de Java (Swing, Spring, Hibernate).



## Reflexión

garantiza que los sistemas informáticos sean eficaces y adaptables. Este trabajo subraya la importancia de las decisiones arquitectónicas para cumplir con requisitos técnicos y comerciales. La elección de una arquitectura en capas muestra cómo una estructura bien definida puede facilitar la escalabilidad y el mantenimiento del sistema, mientras que el uso de frameworks como Spring e Hibernate demuestra la importancia de utilizar herramientas de desarrollo probadas para garantizar la calidad del software.

## Bibliografía

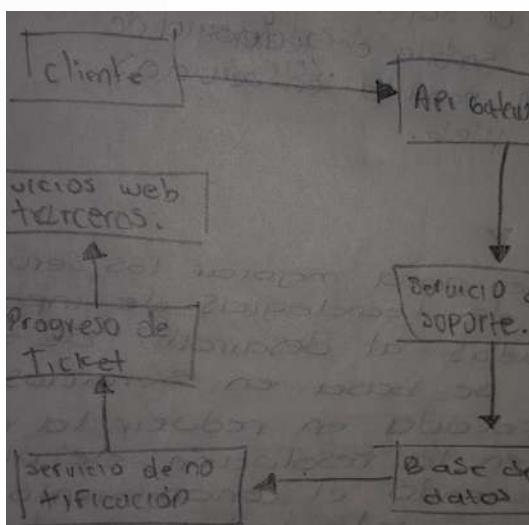
Gamma, E. et al. (2002). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.  
[Bolañoshttps://repositorio.uci.cu/bitstream/ident/TD\\_0201\\_07/1/TD\\_0201\\_07.pdf](https://repositorio.uci.cu/bitstream/ident/TD_0201_07/1/TD_0201_07.pdf)

# Arquitectura de software para el servicio de tecnología basada en servicios web

diseñada para mejorar los servicios de soporte de tecnologías de información (TI) en organizaciones dedicadas al desarrollo de software. La solución propuesta se basa en servicios web y está enfocada en reducir la redundancia en la resolución de problemas, promoviendo la reutilización del conocimiento acumulado.

## Reflexión

es una idea muy útil para mejorar cómo las empresas de tecnología resuelven problemas en sus servicios de soporte. Muchas veces, los equipos repiten soluciones que ya se habían encontrado antes, perdiendo tiempo y recursos. Con esta arquitectura, se pueden almacenar esas soluciones en un lugar común (como una base de conocimiento), para que otros equipos las puedan usar cuando las necesiten.



## Bibliografía

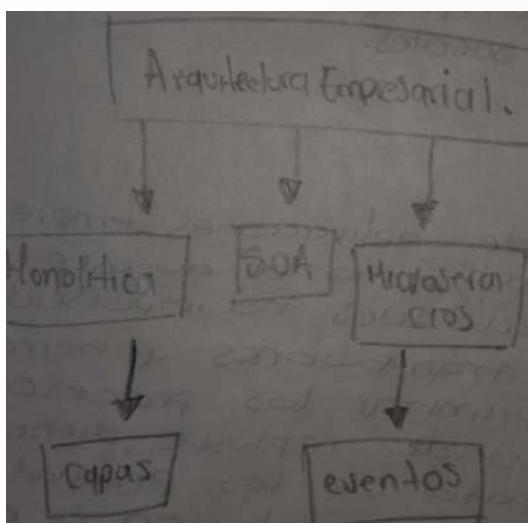
- Josuttis, M. (2007). SOA in Practice: The Art of Distributed System Design. O'Reilly Media.
- Gamma, E. et al. (2002). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Bolaños, https://repository.uci.edu/bitstream/ident/TD\_0201\_07/1/TD\_0201\_07.pdf

# Documentación y análisis frameworks en aplicaciones empresariales

se enfoca en la arquitectura de software y su relevancia en el desarrollo de aplicaciones empresariales, destacando cómo el avance tecnológico y las necesidades del sector empresarial han impulsado la creación de frameworks y herramientas para optimizar procesos. La arquitectura de software es esencial para desarrollar sistemas estructurados, eficientes y escalables, y su diseño adecuado influye directamente en el éxito de los proyectos empresariales. Se aborda su evolución histórica desde los años 60, cuando se comenzaron a sentar las bases del modularidad y la estructuración del software, hasta la llegada de la programación orientada a objetos (OOP) en los 80, que transformó los métodos de desarrollo.

## Reflexión

muestra lo importante que es tener una buena arquitectura de software, especialmente en las empresas que hoy dependen mucho de la tecnología para hacer más eficientes sus procesos. A lo largo del tiempo, el diseño del software ha evolucionado, y ahora es una parte fundamental del desarrollo de cualquier sistema. Las herramientas y metodologías actuales ayudan a los desarrolladores a crear soluciones que sean más rápidas, escalables y fáciles de mantener.

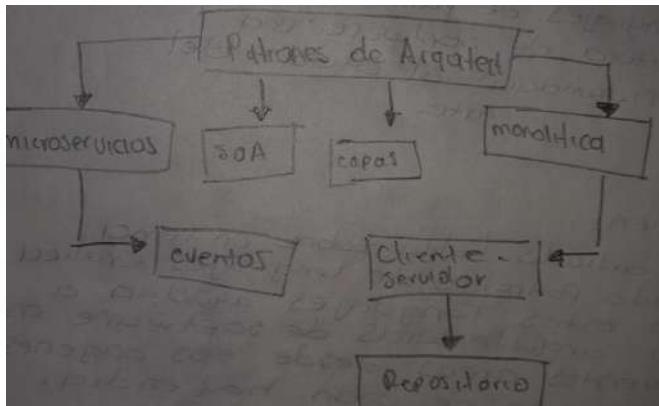


## Bibliografía

- Bas, J. P. (29 de 05 de 2009). Blog de Juan Peláez en Geeks.ms. Recuperado el Marzo de 2015. Josuttis, M. (2007). SOA in Practice: The Art of Distributed System Design. O'Reilly Media. Gamma, E. et al. (2002). Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.Bolañoshttps://repositorio.uci.cu/bitstream/ident/TD\_0201\_07/1/TD\_0201\_07.pdf

# Lenguajes de Patrones de Arquitectura de Software

Este artículo habla sobre un tema de la arquitectura de software llamado "Lenguajes de Patrones". Explica cómo estos lenguajes ayudan a crear arquitecturas de software en diferentes áreas, desde sus orígenes hasta cómo se usan hoy en día. Se destaca que la capacidad de estos lenguajes para adaptarse a diferentes tipos de proyectos los convierte en una herramienta muy útil para los diseñadores y programadores.



## Reflexión

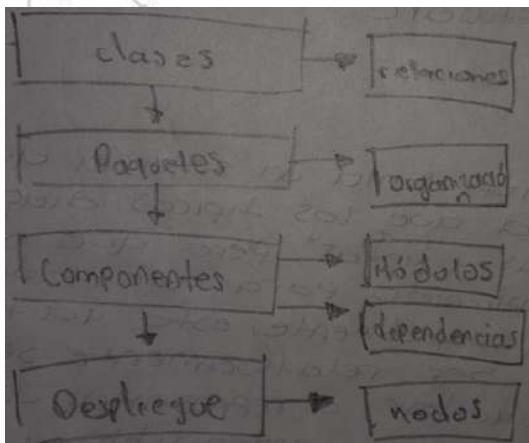
Ha evolucionado mucho desde sus comienzos, y los Lenguajes de Patrones son una herramienta clave para mejorar la forma en que diseñamos y construimos software. Estos lenguajes ayudan a los programadores a lidiar con los problemas y desafíos que surgen cuando se crean sistemas complejos. Lo mejor es que pueden aplicarse a diferentes tipos de proyectos, lo que permite crear soluciones más eficientes y duraderas.

## Bibliografía

FAIRBANKS, George: Just Enough Software Architecture: A Risk Driven Approach. Bolder SOA in Practice: The Art of Distributed System Design. O'Reilly Media.Gamma, E. et al. (2002). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.Bolañoshttps://repositorio.uci.cu/bitstream/ident/TD\_0201\_07/1/TD\_0201\_07.pdf

## Esquema basado en UML para representaciones de arquitectura

no solo es útil para documentar el sistema, sino también para organizar nuestras ideas, trabajar de manera más eficiente y asegurarnos de que todo encaje bien. Si se usa de manera adecuada, puede hacer que el desarrollo de software sea más ordenado, claro y efectivo.



## Bibliografía

RationalSoftware Corporation. Addison-Wesley, 1999.  
[https://www.icesi.edu.co/revistas/index.php/sistemas\\_tematica/article/view/918/943](https://www.icesi.edu.co/revistas/index.php/sistemas_tematica/article/view/918/943)

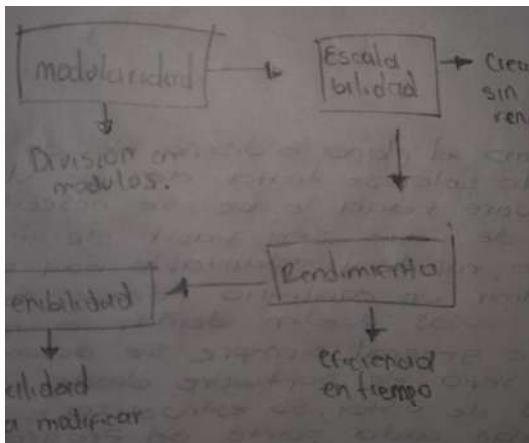
## Reflexión

permite documentar el sistema de forma clara, sino que también ofrece herramientas para gestionar su complejidad. Gracias a su capacidad de extenderse y adaptarse a las necesidades específicas de un proyecto, UML se ha consolidado como el estándar de facto en el desarrollo de software. Al representar visualmente los componentes y sus interacciones, se facilita la colaboración entre equipos interdisciplinarios y se mejora la toma de decisiones durante el ciclo de vida del sistema.

# Atributos de calidad y arquitectura software

La arquitectura de software define cómo implementar estos atributos, equilibrando objetivos técnicos y de negocio. Por ejemplo, diseños modulares mejoran la mantenibilidad, mientras que mecanismos de seguridad robustos protegen el sistema, aunque podrían afectar el rendimiento. Un buen diseño arquitectónico asegura que el sistema cumpla con los estándares de calidad esperados.

## Reflexión



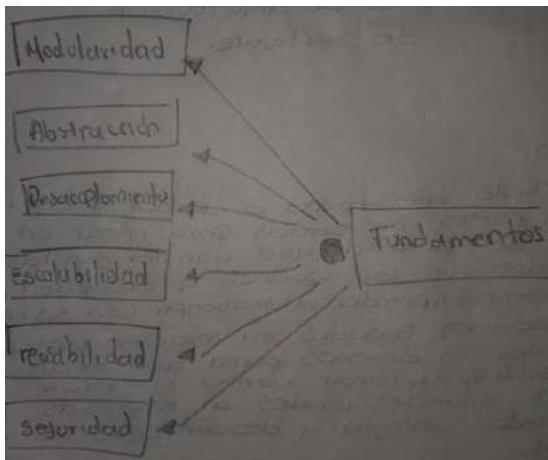
Estos atributos nos llevan a valorar la importancia de pensar a largo plazo, creando soluciones que no solo resuelvan problemas actuales, sino que también sean sostenibles y escalables. Al final, un software bien diseñado no solo funciona; también genera confianza, facilita el trabajo en equipo y proporciona una base sólida para el crecimiento continuo..

## Bibliografía

RationalSoftware Corporation. Addison-Wesley, 1999.  
https://www.Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice. SEI Series in Software Engineering. Addison-Wesley, 2 edition, 2003. Lecture Notes in Computer Science: .icesi.edu.co/revistas/index.php/sistemas\_tematica/article/view/918 /943

# Fundamentos de arquitectura de software

es la estructura organizativa de un sistema, que define sus componentes principales, sus relaciones y cómo interactúan para cumplir con los requisitos funcionales y no funcionales. Es una disciplina esencial en el desarrollo de software, ya que establece la base para el diseño, la implementación y el mantenimiento del sistema. No solo determina la forma en que se construye un sistema.



## Reflexión

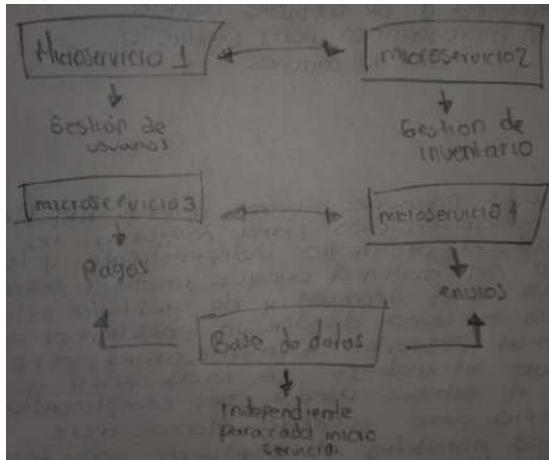
Invita a entender que cada elección tiene un impacto directo en la calidad, la eficiencia y el mantenimiento del sistema. Al construir una arquitectura sólida, no solo estamos creando un producto que funcione, sino una infraestructura que pueda crecer, adaptarse y mantenerse en el tiempo, maximizando su valor a largo plazo. En última instancia, la buena arquitectura no solo resuelve problemas técnicos, sino que también facilita la colaboración, la innovación y el éxito continuo en un entorno de desarrollo dinámico.

## Bibliografía

Richardson, C. (2019). Microservices patterns: with examples in Java (1st edition.). Manning Publications. <https://n9.cl/ps8d25>

## Arquitectura de software basada en microservicios

La arquitectura de microservicios divide una aplicación en pequeños servicios autónomos, cada uno con su propia lógica de negocio y base de datos. Estos servicios se comunican entre sí a través de APIs, lo que permite un desarrollo ágil, escalabilidad independiente y resiliencia. Los microservicios son ideales para sistemas grandes y complejos, ya que permiten que diferentes equipos trabajen en paralelo y que cada servicio se despliegue, actualice y escale por separado. Sin embargo, su implementación presenta desafíos como la gestión de la comunicación entre servicios.



### Reflexión

Representa una evolución significativa en el diseño de software, ofreciendo gran flexibilidad y agilidad. Al permitir que los equipos trabajen de manera más independiente y que los servicios sean escalados o actualizados sin afectar al sistema completo, los microservicios facilitan la adaptabilidad a los cambios rápidos del mercado y las necesidades del usuario. Sin embargo, este enfoque también exige una gestión cuidadosa de la comunicación y la sincronización de los datos.

### Bibliografía

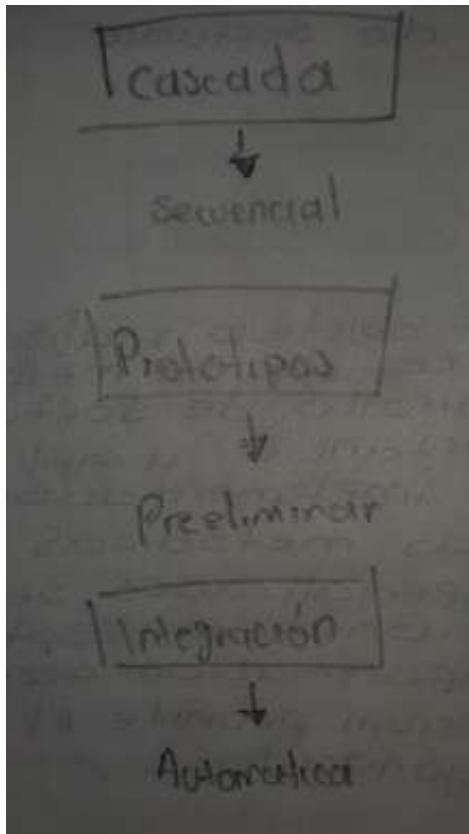
Carneiro, C.; Schmelmer, T.: *Microservices From Day One*. Apress. Berkeley, CA. (2016). <https://n9.cl/0tbddk>

## Revisión sistemática de la metodología

Es un enfoque riguroso y estructurado para recopilar, evaluar y sintetizar la evidencia disponible sobre un tema específico, con el fin de proporcionar una visión integral y objetiva. En el contexto de una metodología, una revisión sistemática tiene como objetivo analizar las diversas metodologías utilizadas en un campo determinado, como el desarrollo de software, para identificar cuáles son las más efectivas y adecuadas según el contexto y los resultados obtenidos.

### Reflexión

Muy fundamental para mejorar la toma de decisiones en el ámbito del desarrollo de software. Al realizar un análisis profundo y estructurado de diversas metodologías, los equipos pueden tomar decisiones más informadas sobre qué enfoques adoptar, dependiendo de las características y necesidades específicas de sus proyectos. Además, proporciona una base sólida para la mejora continua de las prácticas en el desarrollo de software, al identificar las metodologías que han demostrado ser más eficaces en la práctica.



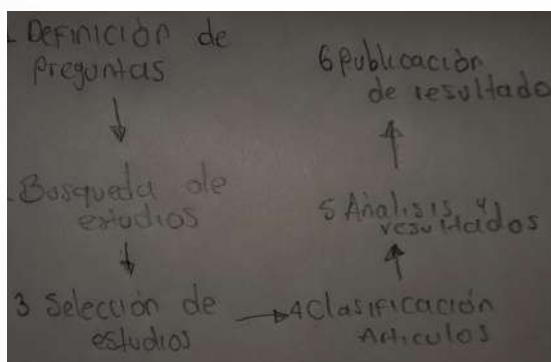
### Bibliografía

Abrahamsson, P.; Salo, O.; Ronkainen, J. & Warsta, J. (2002), Agile software development methods: Review and analysis, Espoo 2002, VTT Publications 478, Oulu.

<https://dialnet.unirioja.es/servlet/articulo?codigo=8384028>

# Mapeo de Arquitecturas de Software

En este artículo se presentan los resultados de un mapeo sistemático de la literatura sobre el tema de recuperación de vistas arquitectónicas de sistemas software, el cual se llevó a cabo aplicando una propuesta metodológica establecida en la literatura. Se identificaron los estudios existentes sobre el tema, especificando el tipo de estudio realizado, su propósito, las técnicas de recuperación que utilizan, los aspectos y elementos de la arquitectura que recuperan y los mecanismos que utilizan para representar las vistas arquitectónicas recuperadas, entre otros aspectos.



## Bibliografía

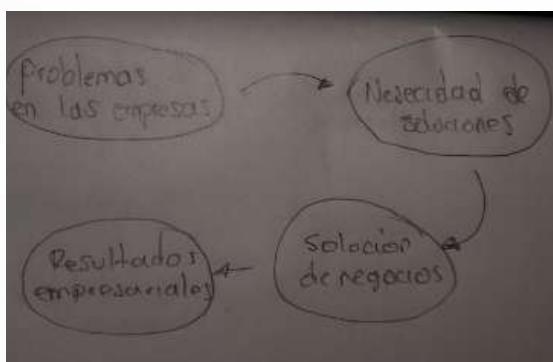
Acher, M.; Cleve, A.; Collet, P.; Merle, P.; Duchien, L. y P. Lahire, Extraction and evolution of architectural variability models in plugin-based systems, *Software and Systems Modeling*: 13(4), 1367-1394 (2014) [https://www.scielo.cl/scielo.php?pid=S0718-07642016000500022&script=sci\\_arttext](https://www.scielo.cl/scielo.php?pid=S0718-07642016000500022&script=sci_arttext)

## Reflexión

Es como la ingeniería inversa, y en particular la recuperación de arquitectura, juega un papel crucial en el mantenimiento y la evolución del software. El alto costo del mantenimiento de sistemas, especialmente aquellos que carecen de documentación o cuya información está desactualizada, subraya la necesidad de contar con mecanismos eficaces para recuperar y representar las vistas arquitectónicas de manera clara y útil. El artículo enfatiza que, sin un marco común para representar estas vistas, los resultados pueden ser difíciles de interpretar y reutilizar.

# Arquitectura de software para la construcción de un sistema

Dentro de las dificultades que se presentan en unas empresas se pueden presentar situaciones como perder cuota de mercado con respecto a la competencia o perder oportunidades de negocio por no tener la información precisa y a tiempo. Esto se evidencia cuando en las empresas se llevan procesos de toma de información manuales para luego almacenarlos en distintos repositorios, físicos o digitales, o valerse de información engañosa o no confiable para la toma de decisiones. Estos problemas se presentan cuando se desconocen las herramientas y metodologías existentes disponibles



## Reflexión

En un entorno empresarial cada vez más competitivo y dinámico, no basta con recolectar datos; es esencial garantizar su calidad y convertirlos en conocimiento accionable. Esto implica no solo implementar herramientas, sino construir modelos robustos que incorporen las mejores prácticas de desarrollo de software. Una empresa que invierte en estas tecnologías y metodologías no solo mejora su eficiencia operativa, sino que también fortalece su capacidad para adaptarse y prosperar en un mercado cambiante.

## Bibliografía

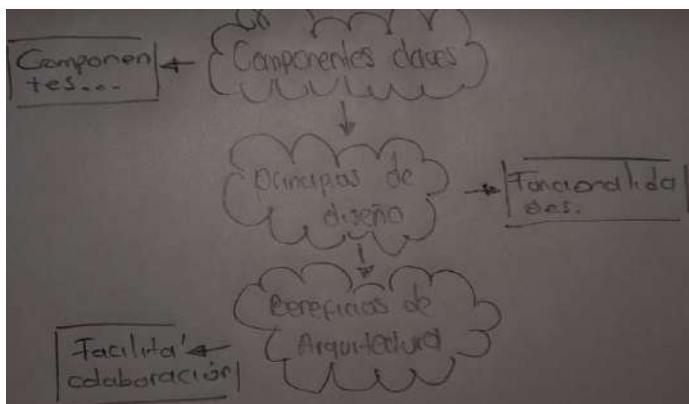
- Hernández Cabrera, G. A. (2017). Arquitectura de software para la construcción de un sistema de cuadro de mando integral como herramienta de inteligencia de negocios. *Tecnología Investigación y Academia*, 5(2), 143-152. <https://revistas.udistrital.edu.co/index.php/tia/article/view/8766>

# Arquitectura Software y Principios de Diseño

Campo fundamental en la ingeniería de software que se enfoca en definir la estructura, los componentes y las relaciones de un sistema para garantizar su funcionalidad, mantenibilidad y escalabilidad. La arquitectura proporciona un marco que guía el desarrollo, mientras que los principios de diseño, como los del acrónimo SOLID, buscan asegurar que el software sea claro, modular y adaptable. Al aplicar correctamente estos principios, se facilita la colaboración entre equipos, se optimizan los procesos y se mejora la calidad del producto final.

## Reflexión

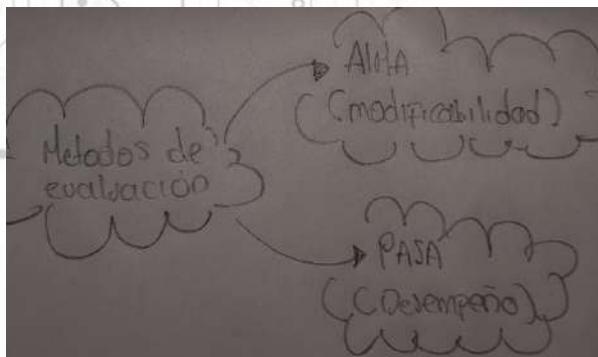
No son solo actividades técnicas, sino estrategias para resolver problemas complejos y responder a las necesidades cambiantes de los usuarios y negocios. Una buena arquitectura es la base de sistemas sostenibles y eficientes, mientras que los principios de diseño garantizan que las soluciones sean elegantes y fáciles de modificar. Ignorar estas prácticas puede llevar a software difícil de mantener, costoso de escalar y propenso a errores.



## Bibliografía

Blas, María Julia; Leone, Horacio Pascual; Gonnet, Silvio Miguel; Modelado y verificación de patrones de diseño de arquitectura de software <https://ri.conicet.gov.ar/handle/11336/125130>

# Evaluando la arquitectura de software



## Reflexión

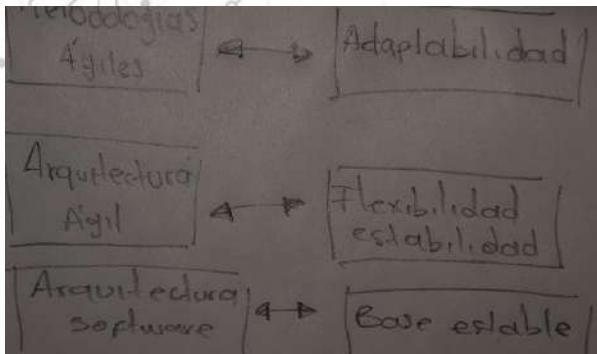
La evaluación de arquitecturas es clave para garantizar que los sistemas de software sean efectivos, flexibles y capaces de cumplir con los requerimientos actuales y futuros. Métodos como ALMA y PASA ofrecen enfoques estructurados para abordar problemas específicos, como la modificabilidad y el desempeño, brindando un marco que reduce riesgos y mejora la toma de decisiones. Estas evaluaciones no solo optimizan costos y recursos, sino que también generan confianza en la capacidad del sistema para adaptarse a un entorno dinámico.

## Bibliografía

Bengtsson, PerOlof, Nico Lassing and Jan Bosch, Vliet, Hans van. "Architecture-Level Modifiability Analysis (Alma)." *The Journal of Systems and Software* 69 (2004): 129-147. [https://osgg.net/omarsite/resources/papers/e\\_v\\_arch\\_02.pdf](https://osgg.net/omarsite/resources/papers/e_v_arch_02.pdf)

El artículo detalla dos métodos de evaluación de arquitecturas de software: ALMA y PASA, diseñados para analizar atributos de calidad específicos. Utiliza escenarios de cambio para predecir costos de mantenimiento, evaluar riesgos de modificaciones o comparar arquitecturas. Sus cinco pasos incluyen definir metas, describir la arquitectura, obtener y evaluar escenarios, e interpretar resultados. Es un método maduro, validado en múltiples dominios como sistemas médicos y telecomunicaciones. Su proceso abarca nueve pasos, desde presentar la evaluación y analizar casos de uso críticos hasta identificar objetivos de desempeño y aplicar técnicas como el uso de anti-pa

# Integración de arquitectura de software en ciclos de vida metodológicos



El artículo aborda la interacción entre Metodologías Ágiles (MA) y la Arquitectura de Software (AS), explorando cómo estas dos áreas, tradicionalmente consideradas incompatibles, pueden converger. Mientras que las MA se enfocan en adaptabilidad, trabajo en equipo y ciclos iterativos que asumen cambios constantes en los requisitos, la AS implica decisiones tempranas y fundamentales que condicionan el desarrollo del sistema y dificultan cambios significativos en etapas posteriores.

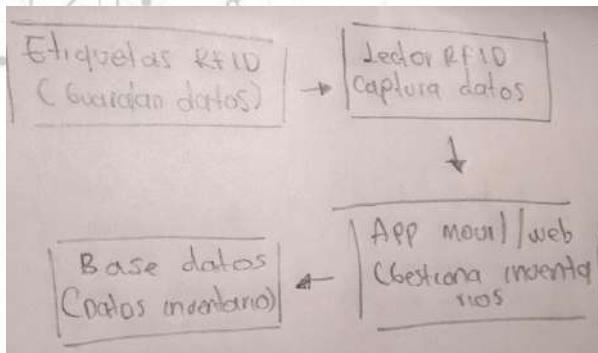
## Reflexión

Representa un desafío y una oportunidad en el desarrollo de software. Por un lado, es evidente que las MA ofrecen agilidad y capacidad de adaptación, ideales para responder a las demandas dinámicas de los usuarios y del mercado. Por otro lado, la AS proporciona una base sólida que garantiza estabilidad y coherencia en proyectos complejos.

## Bibliografía

<http://sedici.unlp.edu.ar/handle/10915/62077>

# Arquitectura móvil para identificación RFID



Permite la identificación y seguimiento de objetos mediante etiquetas que almacenan datos únicos. La tecnología opera con tres componentes principales: etiquetas o tags RFID, lectores que captan y procesan los datos de las etiquetas, y un sistema de software que gestiona la información. Este software puede integrarse con sistemas empresariales como ERP y WMS para optimizar la logística y la gestión de inventarios en tiempo real. Además, su capacidad de procesar y filtrar grandes volúmenes de datos lo convierte en una herramienta clave para tomar decisiones comerciales informadas.

## Reflexión

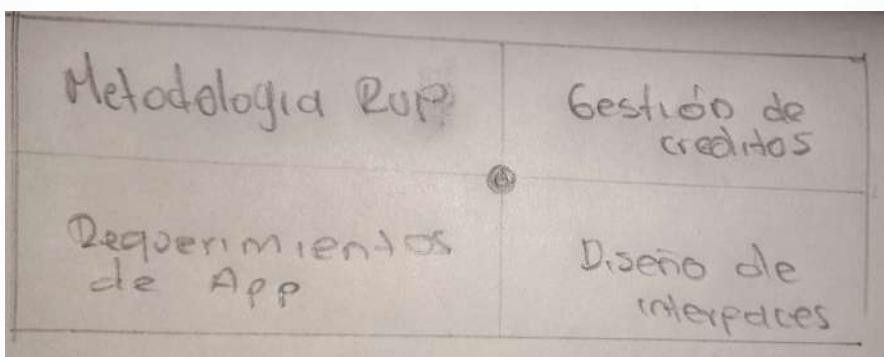
No solo mejora la eficiencia en la gestión de inventarios, sino que también habilita capacidades avanzadas como el seguimiento en tiempo real y la optimización logística. Sin embargo, su integración con otras plataformas empresariales requiere un diseño arquitectónico robusto, especialmente en aplicaciones móviles. Esta combinación es crítica para garantizar un acceso continuo a la información y una interacción eficiente entre los componentes del sistema. La arquitectura de software debe ser flexible y escalable, asegurando que pueda adaptarse a las necesidades cambiantes de las empresas y a los avances en la tecnología RFID.

## Bibliografía

- avón Mestras, J. (2009). Estructura de las Aplicaciones Orientadas a Objetos. El patrón Modelo-Vista-Controlador (MVC). Madrid, Madrid, España.  
<https://recibe.cucei.udg.mx/index.php/ReCIBE/article/view/41>

## Arquitectura de Software Empresarial en RPG

Permite la identificación y seguimiento de objetos mediante etiquetas que almacenan datos únicos. La tecnología opera con tres componentes principales: etiquetas o tags RFID, lectores que captan y procesan los datos de las etiquetas, y un sistema de software que gestiona la información. Este software puede integrarse con sistemas empresariales como ERP y WMS para optimizar la logística y la gestión de inventarios en tiempo real. Además, su capacidad de procesar y filtrar grandes volúmenes de datos lo convierte en una herramienta clave para tomar decisiones comerciales informadas.



## Reflexión

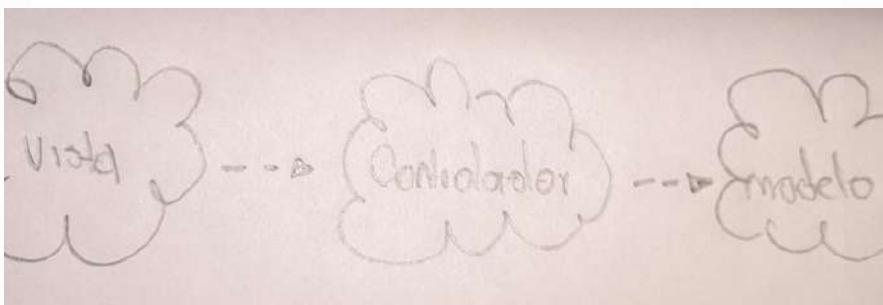
El proyecto propuesto tiene un enfoque muy valioso al aplicar una arquitectura de software moderna para mejorar el proceso de desarrollo y mantenimiento de aplicaciones en RPG IV, un lenguaje tradicionalmente utilizado en entornos empresariales. Este tipo de enfoque es crucial, especialmente en industrias donde el software ha estado en uso durante muchos años, pero necesita evolucionar para seguir siendo competitivo y eficiente.

## Bibliografía

Barranzuela Imán, N. A. (2019). [https://alicia.concytec.gob.pe/vufind/Record/RUMP\\_8ed36e1531e7862df37c24b79b9aaaa7/Description#tabnav](https://alicia.concytec.gob.pe/vufind/Record/RUMP_8ed36e1531e7862df37c24b79b9aaaa7/Description#tabnav)

## **MODELO-VISTA-CONTROLADOR. LENGUAJE UML**

Se realiza un análisis sobre el origen y evolución del patrón MVC y el uso de UML para modelar aplicaciones de software, identificando cómo estos enfoques han mejorado la organización y estructuración del código, y cómo permiten gestionar la complejidad de las aplicaciones. MVC, por ejemplo, separa las preocupaciones del modelo, la vista y el controlador, permitiendo un desarrollo más limpio y escalable. La incorporación de UML facilita la visualización de estos componentes y sus interacciones, lo que refuerza la comprensión y la gestión de proyectos complejos.



### **Reflexión**

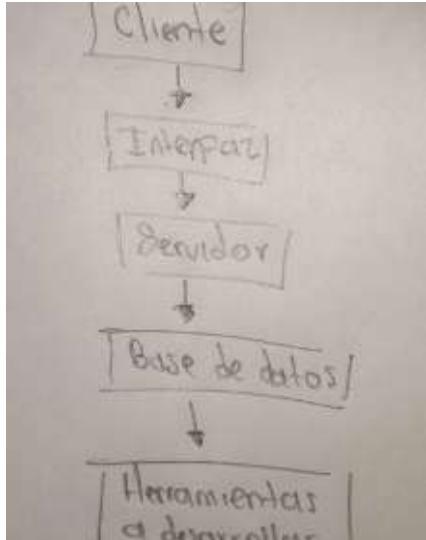
Es que la integración de patrones como MVC y el uso de UML no solo mejora la calidad del software, sino también la enseñanza de desarrollo de aplicaciones. Estos enfoques proporcionan un marco claro que puede adaptarse tanto al aprendizaje como a la práctica profesional, ayudando a reducir errores y mejorar la mantenibilidad del software

### **Bibliografía**

<https://crea.ujaen.es/handle/10953.1/11437>

## Marco de arquitectura para apps web y móviles

Describe la creación de un marco de referencia de arquitectura de software para aplicaciones web y móviles basado en tecnologías de Software Libre y Código Abierto resalta la importancia de utilizar herramientas robustas y escalables. Al integrar Java con tecnologías como JPA, EJB, y un servidor de aplicaciones como WildFly, el sistema asegura una arquitectura eficiente, con soporte para transacciones y seguridad. Además, el uso de tecnologías modernas como ExtJS, Sencha Touch y JasperReports en el lado del cliente permite crear interfaces gráficas de alto rendimiento y con un rico contenido visual.



## Reflexión

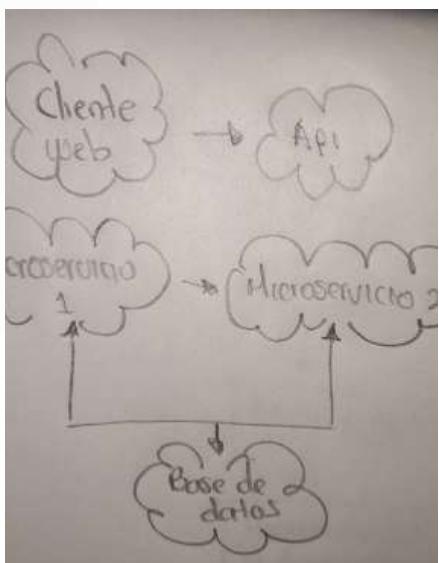
Es una estrategia acertada en términos de flexibilidad, costo y comunidad de soporte. Al ser accesible y modificable, permite a las organizaciones adaptar las soluciones a sus necesidades específicas, lo cual es esencial para proyectos de largo plazo. La Programación Orientada a Objetos (POO) con Java asegura una estructura robusta y mantenible, facilitando el escalado y la incorporación de nuevas funcionalidades.

## Bibliografía

- Maliza Martinez, C. A., López Mendizábal, V. L., & Mackliff Peñafiel, V. V. (2016). Marco de referencia de arquitectura de software para aplicaciones web y móviles. *Journal of Science and Research*, 1(CITT2016), 72–75. <https://doi.org/10.26910/issn.2528-8083vol1issCITT2016.2016pp72-75>

# Microservicios para desarrollo de aplicaciones web de la Asamblea Nacional

En cuanto a las tecnologías que facilitan la implementación de microservicios, se destacan Docker para la contenedorización de servicios, Kubernetes para la orquestación, y lenguajes de programación como Java o Node.js. Además, el uso de metodologías ágiles y herramientas de integración continua contribuyen a un desarrollo más dinámico y eficaz.



## Reflexión

Revela la importancia de modernizar y evolucionar las arquitecturas de software utilizadas en las instituciones públicas. El uso de arquitecturas monolíticas, aunque efectivo en su momento, presenta una serie de desafíos en términos de escalabilidad, mantenimiento y flexibilidad. Este estudio evidencia la necesidad de adoptar enfoques más modernos como los microservicios, que permiten la creación de aplicaciones modulares y más fáciles de mantener y actualizar.

## Bibliografía

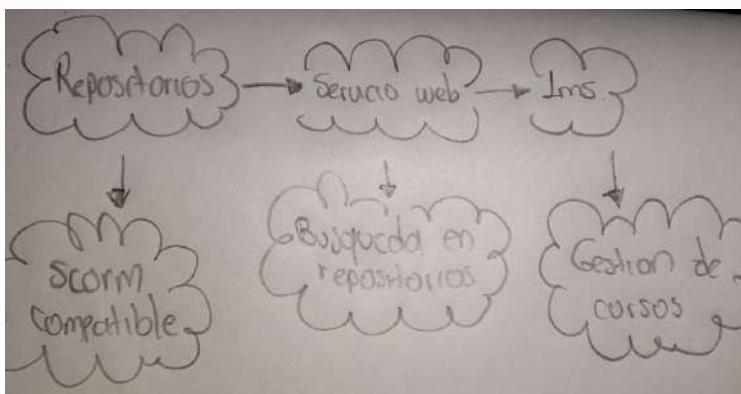
Richards, M. Software Architecture Patterns. O'Reilly Media, Inc. (2015). <https://repositorio.utn.edu.ec/bitstream/123456789/7603/1/PG%20569%20TESIS.pdf>

# Arquitectura de software para la integración de objetos de aprendizaje basada en servicios web

El artículo aborda los avances en la educación virtual, destacando cómo la reutilización de objetos de aprendizaje, almacenados en repositorios, ha optimizado la creación y distribución de contenidos educativos. El uso de estándares como SCORM asegura la interoperabilidad y reusabilidad de estos objetos. Además, se menciona que los Sistemas de Gestión de Aprendizaje (LMS) permiten el diseño de contenidos educativos, pero generalmente funcionan de manera independiente a los repositorios de objetos de aprendizaje.

## Reflexión

a integración de repositorios de objetos de aprendizaje con sistemas LMS mediante una arquitectura de software basada en servicios web representa un avance significativo en el campo de la educación digital. A medida que los entornos de E-Learning evolucionan, se hace cada vez más necesario superar las barreras de interoperabilidad entre plataformas y sistemas. La propuesta presentada en el artículo refleja cómo los estándares como SCORM, junto con el diseño basado en servicios web



## Bibliografía

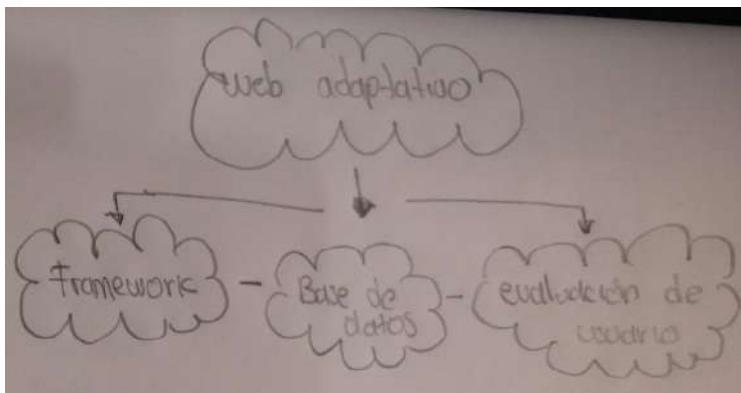
- Anced, (2010). Libro de buenas prácticas de E-Learning. ANCED, Asociación Nacional de Centros de E-Learning y Distancia. <http://www.buenaspracticas-E-Learning.com/capitulo-16-estandares-E-Learning.html> 10/03/2011.

# Arquitectura de software para web adaptativa manejadora del secuencia miento de objetos de aprendizaje

En la actualidad la Universidad Señor de Sipán, no cuenta con una web adaptativa para gestionar objetos de aprendizaje, que le sirva como herramienta para la enseñanza y aprendizaje del Lenguaje de Programación JAVA. La finalidad es optimizar la manera de cómo se enseña y aprende el Lenguaje de Programación JAVA, para ello se ha desarrollado una aplicación web adaptativa, que le permita a los estudiantes aprender las técnicas básicas de programación del lenguaje. La d software desarrollada en la aplicación web adaptativa, está compuesta por un modelo de dominio, usuario y de adaptación.

## Reflexión

Esta iniciativa demuestra el potencial de las tecnologías educativas modernas para transformar la enseñanza y el aprendizaje. Sin embargo, siempre hay espacio para la mejora continua, tanto en la calidad del contenido como en la manera en que se motiva a los estudiantes. La integración de tecnologías como los OA y el aprendizaje adaptativo es crucial para el futuro de la educación, ya que permite a los estudiantes avanzar a su propio ritmo, aprender de manera más personalizada y desarrollar habilidades prácticas de programación de forma efectiva.



## Bibliografía

Escurra

<https://repositorio.uss.edu.pe/handle/20.500.12802/9819>

Cisneros,

Jose

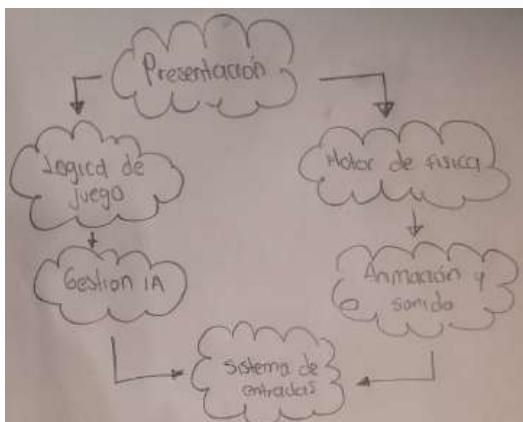
Lino.

# Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D

A implementación de una arquitectura sólida desde las fases iniciales del desarrollo de un videojuego no solo contribuye al éxito del proyecto, sino que también facilita la integración de nuevas tecnologías, la resolución de problemas a largo plazo y el mantenimiento constante a medida que el juego crece y evoluciona. La investigación subraya cómo una buena arquitectura puede mejorar la calidad y la eficiencia del desarrollo de videojuegos, ofreciendo un camino claro para desarrolladores que busquen optimizar sus procesos y entregar productos finales de alta calidad.

## Reflexión

La investigación presentada aborda la importancia de estructurar correctamente las características funcionales de un videojuego mediante una arquitectura bien definida. Esto no solo permite un desarrollo más organizado, sino que también ayuda a mitigar los riesgos que pueden surgir durante el ciclo de vida del proyecto, como la dificultad para escalar o mantener el código a medida que el juego evoluciona.

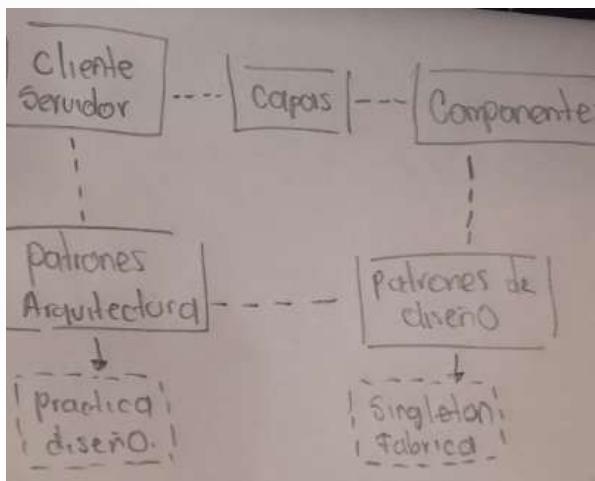


## Bibliografía

Stack, P. History of video game consoles. Time Magazine website 2005; Available from: [http://www.time.com/time/covers/1101050523/consol\\_e\\_timeline](http://www.time.com/time/covers/1101050523/consol_e_timeline)

# Estilos y patrones en la estrategia de arquitectura de Microsoft

los patrones de diseño y arquitectura, junto con los estilos arquitectónicos, se han consolidado como temas clave en la arquitectura de software, desempeñando roles complementarios pero también opuestos en muchos casos. Los patrones de diseño se enfocan más en la implementación práctica y los detalles técnicos del desarrollo de software, mientras que los estilos arquitectónicos se centran en las configuraciones de alto nivel, la teoría y la estructura de la arquitectura de software.



## Reflexión

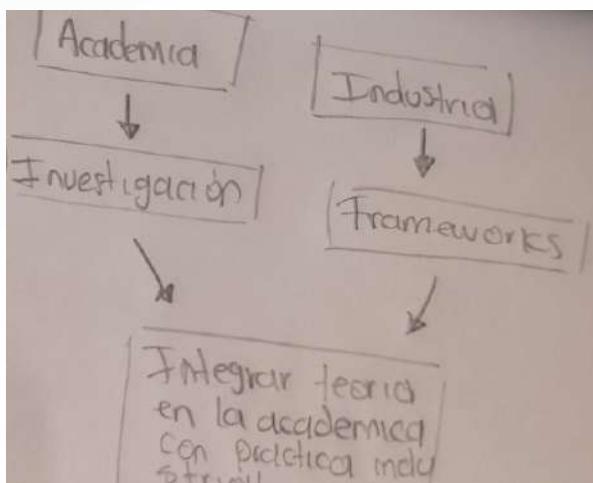
La distinción entre estilos y patrones refleja una diferencia importante en los enfoques de la arquitectura de software. Los estilos se ocupan de la visión global y estructural de los sistemas, mientras que los patrones abordan problemas concretos de implementación y diseño. Esta diferenciación permite que ambos enfoques se complementen, ofreciendo a los arquitectos y desarrolladores un conjunto de herramientas y conceptos que pueden aplicar dependiendo del nivel de abstracción y las necesidades del proyecto.

## Bibliografía

Reynoso, C. y Kicillof, N. (2004). Estilos y patrones en la estrategia de arquitectura de Microsoft. Recuperado de: <http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF>

# Introducción a la Arquitectura de Software

Sus herramientas y sus patrones de diseño. Hay múltiples razones para desarrollar esta presentación. Por empezar, no hay todavía textos en lengua castellana que brinden aproximaciones actualizadas a la Arquitectura de Software (en adelante, AS). El proceso editorial es hoy mucho más lento que el flujo de los acontecimientos y el cambio tecnológico; casi toda la producción en papel manifiesta atraso respecto de los elementos de juicio que urge considerar, tanto en el plano conceptual como en el tecnológico. Pero aún operando en binario y en banda ancha sobre la red de redes, el flujo de información de la industria rara vez se cruza con los intercambios del circuito académico



## Bibliografía

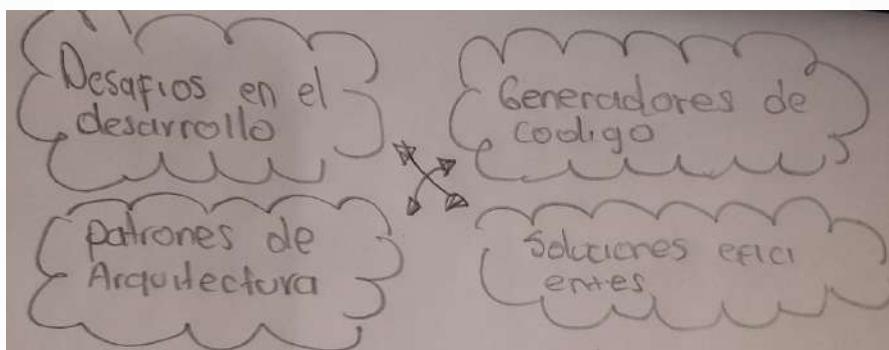
Pekka Abrahamsson, Outi Salo, Jussi Ronkainen y Juhani Warsta. "Agile Software Development Methods". VTT Publications 478, Universidad de Oulu, Suecia, 2002.

## Reflexión

Destaca la necesidad de un puente entre la teoría académica y la práctica industrial en la Arquitectura de Software (AS). Por un lado, la academia se centra en definir marcos conceptuales y metodologías, mientras que la industria prioriza soluciones prácticas y rápidas. Esta desconexión genera diferencias en prioridades y diagnósticos, haciendo crucial articular estos enfoques para aprovechar mejor las capacidades de la Arquitectura de Software.

# **Revisión sistemática sobre generadores de código fuente y patrones de arquitectura**

El desarrollo de software enfrenta desafíos como retrasos en la entrega y productos de baja calidad, a menudo derivados de una organización deficiente del código y una falta de integración de componentes. Los Generadores de Código Fuente (GCF) son herramientas que automatizan la creación de código, especialmente en procesos repetitivos como interfaces de datos o conexiones a bases de datos. Estas herramientas aumentan la productividad y aseguran consistencia en el desarrollo.



## **Reflexión**

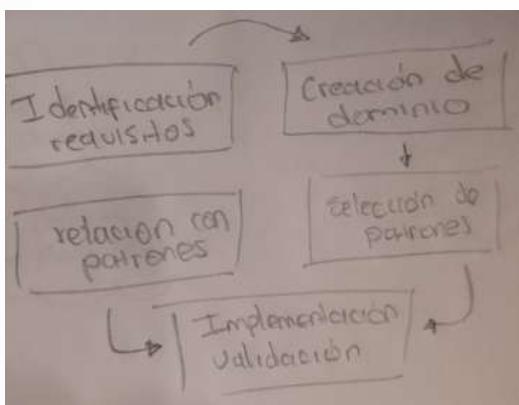
La integración de Generadores de Código Fuente con patrones de arquitectura representa un avance significativo en la industria del software. Mientras que los GCF automatizan tareas repetitivas, los patrones garantizan la estandarización y la organización lógica del sistema, reduciendo riesgos y mejorando la calidad. Esta sinergia permite a los desarrolladores enfocarse en áreas críticas del software, promoviendo la eficiencia y la reutilización de componentes. Sin embargo, el éxito de estas herramientas depende de una adecuada planificación y diseño inicial.

## **Bibliografía**

Hawa Çetiner Altıparmak ; Büşra Tokgöz ; Ökkeş Emin Balçık ; Aslıhan Özkaya ; Ahmet Arslan, Source code generation for large scale application, Turkey, 2013.<https://tesis.pucp.edu.pe/repositorio/handle/20.500.12404/16457>

# Tipificación de Dominios de Requerimientos para la Aplicación de Patrones Arquitectónicos

El artículo aborda una investigación orientada a mejorar la fase de diseño de arquitectura de software mediante la creación de un dominio de requerimientos. Este dominio agrupa elementos comunes en proyectos de desarrollo web enfocados en la integración de plataformas y ecosistemas digitales. A partir de este dominio, se relacionaron patrones arquitectónicos actuales para facilitar la selección del más adecuado en proyectos específicos.



## Reflexión

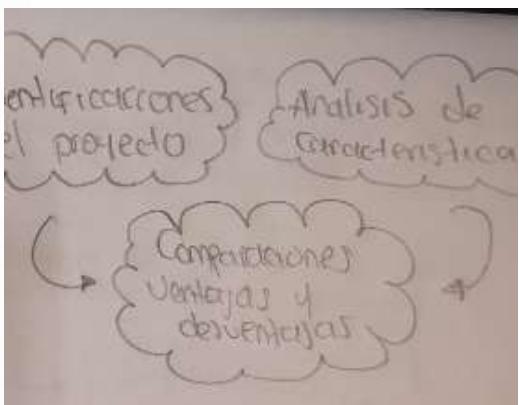
Empresas de software podrían beneficiarse significativamente al invertir en la capacitación de sus equipos para manejar las etapas arquitectónicas de manera integral, reduciendo la dependencia exclusiva de expertos externos. Este enfoque fomenta la colaboración y permite a las organizaciones construir capacidades internas sostenibles.

## Bibliografía

Hawa Çetiner Altıparmak ; Büşra Tokgöz ; Ökkeş Emin Balçık ; Aslıhan Özka ; Ahmet Arslan, Source code generation for large scale application, Turkey, 2013.<https://tesis.pucp.edu.pe/repositorio/handle/20.500.12404/16457>

# Análisis comparativo de Patrones de Diseño de Software

El artículo presenta un análisis comparativo de cinco patrones de diseño de software: Template Method, Model-View-Controller (MVC), Model-View-Presenter (MVP), Front Controller, y Model-View-ViewModel (MVVM). Los patrones de diseño son soluciones reutilizables y probadas para problemas comunes durante el desarrollo de software, lo que facilita la organización, el mantenimiento y la calidad del código.



## Reflexión

La elección de un patrón de diseño es una decisión estratégica en el desarrollo de software, ya que define la estructura organizativa del código y su mantenibilidad futura. Este artículo enfatiza que no existe un enfoque universalmente mejor, sino que la selección debe basarse en las características del proyecto y los objetivos deseados.

## Bibliografía

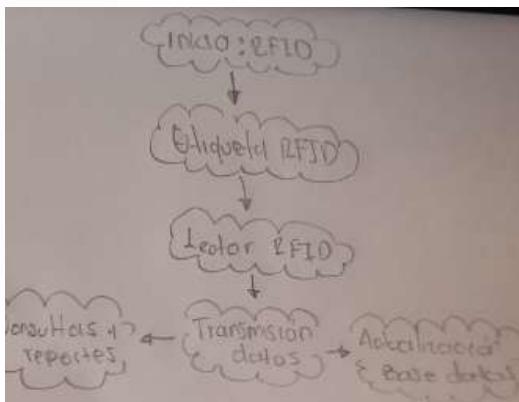
G. Carrera y J. Germania, «Análisis comparativo de la productividad entre los patrones de diseño Modelo Vista Controlador (MVC) y Modelo Vista Presentador (MVP) aplicado al desarrollo del Sistema Nómina de Empleados y Rol de Pagos de la Distribuidora Soria C.A.», 2014. <https://dialnet.unirioja.es/servlet/articulo?codigo=9042927>

# Arquitectura de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia

El artículo aborda el diseño de una arquitectura de software para integrar un sistema RFID en el Instituto Tecnológico de Orizaba, con el objetivo de mejorar la gestión de inventarios. Este sistema emplea etiquetas RFID para identificar y localizar objetos de manera rápida y precisa, reduciendo el tiempo de actualización y minimizando errores humanos. La propuesta incluye una aplicación móvil que interactúa con lectores RFID y utiliza una base de datos en formato XML como intermediario entre el middleware y los dispositivos. Se destacan las ventajas de las etiquetas RFID, tanto activas como pasivas, y se presentan aplicaciones específicas que demuestran la efectividad de la arquitectura desarrollada.

## Reflexión

La integración de tecnología RFID en la gestión de activos es una solución eficaz para mejorar la precisión y eficiencia en los inventarios. Este artículo evidencia cómo la implementación de una arquitectura bien diseñada no solo optimiza procesos, sino que también minimiza riesgos, como la pérdida de bienes o errores manuales.

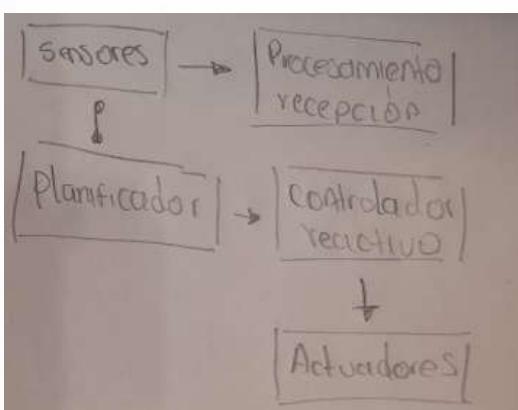


## Bibliografía

- Pavón Mestras, J. (2009). Estructura de las Aplicaciones Orientadas a Objetos. El patrón Modelo-Vista-Controlador (MVC). Madrid, Madrid, España. <https://www.redalyc.org/pdf/5122/512251501004.pdf>

# Arquitecturas de Software para el diseño de Robots Móviles Autónomos

La arquitectura de software para robots móviles autónomos es un área clave en la robótica, ya que define la estructura y las interacciones del sistema para garantizar la navegación, planificación y ejecución de tareas en entornos dinámicos. Este tipo de sistemas combina tecnologías avanzadas de sensores, controladores y algoritmos para adaptarse a diferentes escenarios.



## Bibliografía

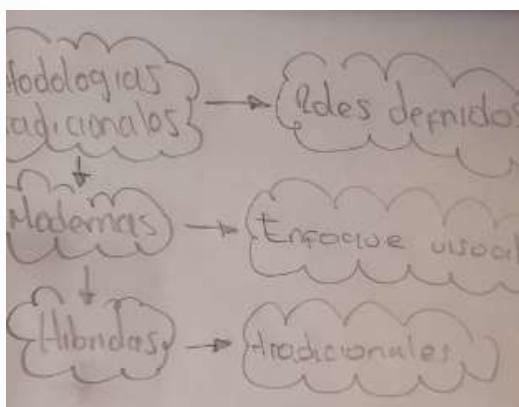
M. Shaw and D. Garlan. Software Architecture. Perspectives on an Emerging Discipline, Prentice Hall, 1996, Upper Saddle River, New Jersey.  
[https://sedici.unlp.edu.ar/bitstream/handle/10915/21720/Documento\\_completo.pdf?sequence=1&isAllowed=y](https://sedici.unlp.edu.ar/bitstream/handle/10915/21720/Documento_completo.pdf?sequence=1&isAllowed=y)

## Reflexión

Estos sistemas no solo mejoran la eficiencia de los robots, sino que también abren puertas a nuevas aplicaciones, desde la logística hasta la exploración espacial, donde la autonomía y adaptabilidad son esenciales. A medida que la tecnología avanza, se espera que estas arquitecturas evolucionen, integrando más inteligencia artificial y capacidades adaptativas para enfrentar entornos cada vez más complejos.

# **Una revisión comparativa de la literatura acerca de metodologías tradicionales y modernas de desarrollo de software**

El artículo analiza las metodologías de desarrollo de software, clasificándolas en tradicionales y modernas. Entre las metodologías tradicionales más estudiadas se encuentran el modelo en cascada y el modelo en espiral, caracterizados por procesos lineales o iterativos, respectivamente. Por otro lado, las metodologías modernas, como las ágiles, destacan por su adaptabilidad y eficiencia, incluyendo enfoques como Scrum, Programación Extrema (XP), y Desarrollo Orientado a Funcionalidades (FDD).



## **Reflexión**

La elección de una metodología depende de las características del proyecto, el equipo y los objetivos organizacionales. Las metodologías tradicionales son adecuadas para proyectos bien definidos y estables, mientras que las ágiles se adaptan mejor a entornos dinámicos. En la actualidad, las ágiles predominan debido a la necesidad de adaptarse rápidamente a cambios en los requisitos del cliente y la tecnología.

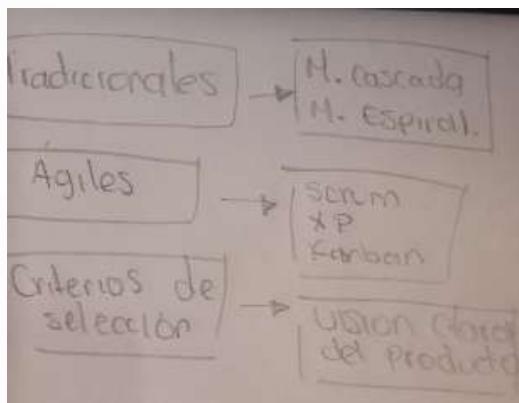
## **Bibliografía**

H. D. Ortiz Alzate, L. G. Muñoz Marín, J. Cardeño Espinosa, y N. C. Alzate Osorno.

<https://revistas.pascualbravo.edu.co/index.php/cintex/article/view/334>

# Criterios de selección de metodologías de desarrollo de software

El artículo describe la evolución de las metodologías de desarrollo de software, abordando el debate entre metodologías tradicionales y ágiles. Las tradicionales, como el modelo en cascada o en espiral, priorizan el control y seguimiento del proceso, ofreciendo estabilidad y una estructura bien documentada. Las ágiles, como Scrum y Programación Extrema (XP), ponen énfasis en la flexibilidad, la colaboración humana y la entrega continua mediante iteraciones cortas.



## Bibliografía

H. D. Ortiz Alzate, L. G. Muñoz Marín, J. Cardeño Espinosa, y N. C. Alzate Osorno.

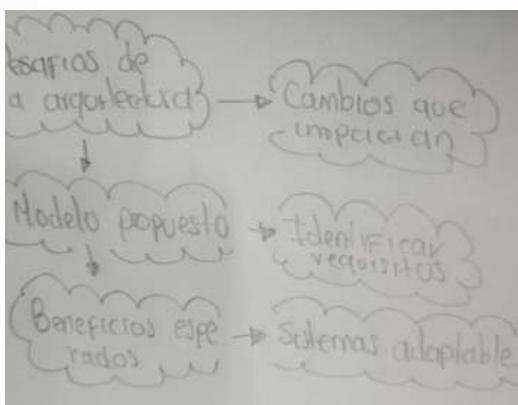
<https://revistas.pascualbravo.edu.co/index.php/cintex/article/view/334>

## Reflexión

La selección de una metodología adecuada es crítica para el éxito del desarrollo de software. Las metodologías tradicionales son ideales para proyectos con requisitos claros y bien definidos, mientras que las ágiles son más efectivas en entornos dinámicos donde los requisitos pueden cambiar con frecuencia. Además, el marco teórico aportado por Dijkstra sigue siendo relevante, ya que subraya principios fundamentales como la estructuración lógica del código y la resolución sistemática de problemas.

# **Arquitectura de software en el proceso de desarrollo ágil: una perspectiva basada en requisitos significantes para la arquitectura**

El trabajo analiza la relación entre las metodologías ágiles y la arquitectura de software, destacando las diferencias en el manejo de requisitos. Las metodologías ágiles gestionan los requisitos de manera evolutiva a lo largo del ciclo de desarrollo, adaptándose a cambios constantes. Por otro lado, la arquitectura de software requiere una definición inicial clara y precisa de los requisitos clave, ya que cambios posteriores pueden afectar significativamente el diseño y calidad del producto final.



## **Reflexión**

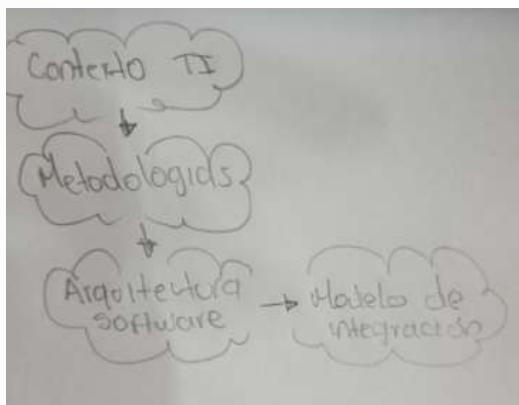
El enfoque presentado en este trabajo es prometedor porque combina lo mejor de ambos mundos: la adaptabilidad de las metodologías ágiles y la estabilidad de un diseño arquitectónico planificado. Esto no solo mejora la calidad del software, sino que también reduce riesgos relacionados con cambios de requisitos en etapas avanzadas.

## **Bibliografía**

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

# **Selección de metodologías ágiles e integración de arquitecturas de software en el desarrollo de sistemas de información**

La Ingeniería de Software (IS) se ha vuelto esencial en el contexto organizacional, donde la gestión de proyectos depende fuertemente de los Sistemas de Información (SI) y las Tecnologías de la Información (TI). En este entorno, las Metodologías Ágiles (MA) han ganado popularidad debido a su capacidad para adaptarse a entornos cambiantes, a diferencia de las metodologías tradicionales que han demostrado ser ineficientes en proyectos donde los requisitos cambian rápidamente.



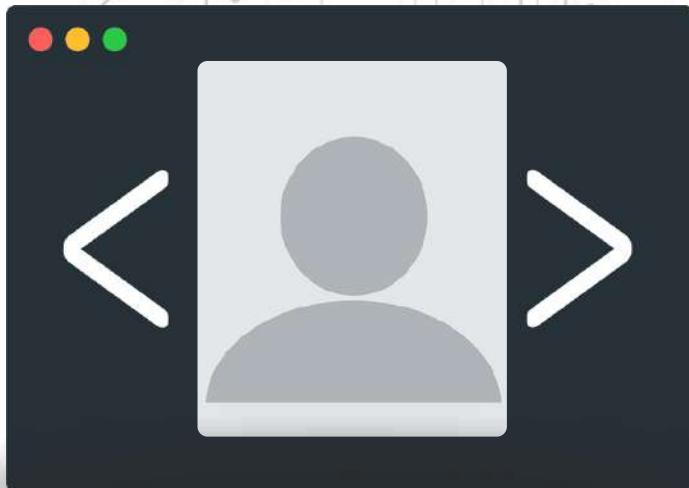
## **Reflexión**

La evolución de la Ingeniería de Software hacia la integración de metodologías ágiles representa un cambio significativo en la forma en que los proyectos de software son gestionados. Las metodologías ágiles, al centrarse en la flexibilidad y adaptación rápida a los cambios, han proporcionado una solución eficaz para proyectos en entornos de alta incertidumbre. Sin embargo, la arquitectura de software, que es esencial para el desarrollo de sistemas robustos y escalables, no debe ser descuidada en este enfoque ágil.

## **Bibliografía**

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

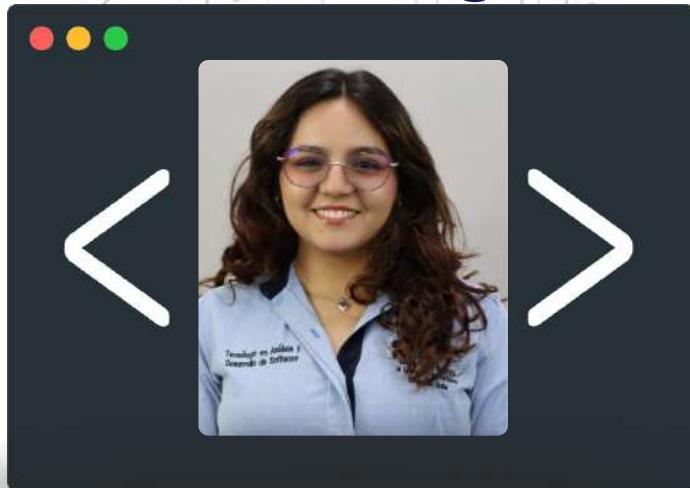
# Manuel Ricardo Diez Corredor



Mi nombre es Manuel Ricardo Diez Corrdor, nací el 26 de abril de 2003 en San Vicente del Caguán. Vivo en Neiva con mis padres y dos hermanas. Terminé el bachillerato en 2021, obteniendo un título como técnico en sistemas del Sena, cursado durante mis dos últimos años de colegio. Actualmente, estudio el tecnólogo en Análisis y Desarrollo de Software, una carrera que me motiva a superarme cada día.

Soy una persona emprendedora y curiosa. Me apasiona reparar cosas, lo que me llevó a aprender, tanto de forma empírica como con estudios, a arreglar aparatos electrónicos. También disfruto viajar y explorar nuevas aventuras, actividades que me llenan de energía y motivación. Mi familia es una prioridad para mí, siempre estoy atento a su bienestar. Me esfuerzo por construir un futuro sólido y alcanzar mis metas, aplicando disciplina y dedicación en cada aspecto de mi vida.

# Maria Del Mar Artunduaga Artunduaga



Soy desarrolladora Fullstack con una profunda pasión por la tecnología y el desarrollo de soluciones innovadoras. Mi experiencia incluye el consumo de APIs utilizando Django Rest Framework, así como el manejo de diversas tecnologías y lenguajes de programación como JavaScript, Python, Java, MySQL, HTML, React y Tailwind CSS. También tengo competencias en herramientas como Postman, Git, GitHub y en entornos como Linux e inteligencia artificial.

Me destaco por mis habilidades de liderazgo y trabajo en equipo, lo que me permite colaborar eficazmente en proyectos orientados al desarrollo y la implementación de software de calidad. Estoy comprometida con el aprendizaje continuo y la aplicación de mis conocimientos para resolver desafíos tecnológicos y aportar valor en cada proyecto en el que participo.

# Documentación Y Análisis De Los Principales Frameworks De Arquitectura De Software En Aplicaciones Empresariales

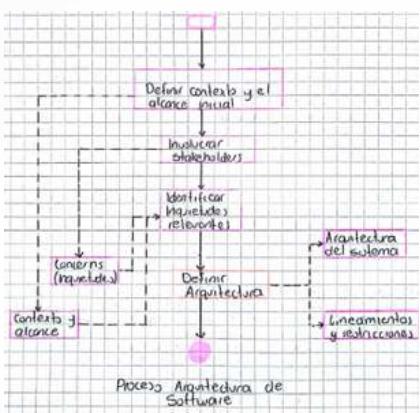
El trabajo destaca la importancia de la arquitectura de software en el ámbito empresarial para alinear estrategias de negocio con soluciones tecnológicas, mejorando la gestión y eficiencia interna. Subraya la necesidad de definir una arquitectura sólida que integre procesos organizacionales y TI, seleccionando frameworks adecuados según las necesidades específicas de la empresa. También resalta la gestión y comunicación como claves para la implementación exitosa, evitando resistencias. La investigación busca ser una guía inicial para explorar el uso de frameworks en la arquitectura empresarial.

## Reflexión

La arquitectura de software en aplicaciones empresariales combina tecnología y procesos de negocio para lograr éxito organizacional. Su implementación requiere un enfoque integral que considere necesidades, recursos y capacidad de adaptación a cambios tecnológicos. El éxito no radica solo en la calidad del sistema, sino en una gestión efectiva que involucre a toda la organización y alinee decisiones tecnológicas con objetivos estratégicos. Una arquitectura bien diseñada optimiza recursos, mejora la interacción con clientes y fomenta el crecimiento empresarial. Vincular tecnología con estrategia, liderazgo y cultura organizacional es clave para generar valor sostenible.

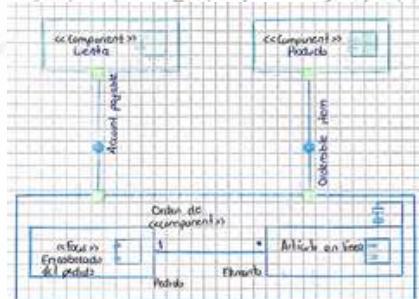
## Bibliografía

Palmero, M. A. S., Martínez, N. S., Grass, O. y. R., Palmero, M. A. S., Martínez, N. S., & Grass, O. y. R. (s. f.). Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software. [http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci\\_arttext](http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci_arttext)



# Revisión de Conceptos para Arquitecturas de Referencia de Software

La representación del conocimiento en arquitecturas de referencia de software incluye la definición estándar y descomposición de sus componentes. Esta investigación identificó formas comunes de representación y destacó elementos clave y sus limitaciones. Se encontró que las arquitecturas suelen describirse en lenguaje natural, lo que limita el uso de herramientas para análisis, reutilización de información y verificación de especificaciones. Los hallazgos permitieron desarrollar una propuesta para representar el conocimiento arquitectónico, enfocándose en las decisiones arquitectónicas y su razonamiento.



## Reflexión

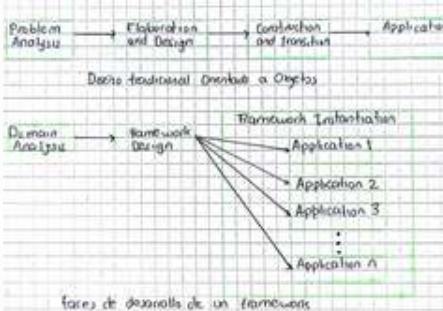
La investigación profundizó en los elementos clave para definir las arquitecturas de referencia de software. Los conceptos más relevantes, como Componente y Conector, están basados en el estándar IEEE 1471 (Rozanski & Woods, 2012), que propone un modelo conceptual para la definición de Arquitectura de Software. Se encontró que la mayoría de los modelos revisados usan la vista arquitectónica como elemento principal, lo que limita el uso de herramientas automatizadas al no emplear un lenguaje formal. Estos hallazgos guiarán una propuesta para representar el conocimiento arquitectónico, enfocándose en decisiones y razonamientos arquitectónicos.

## Bibliografía

- Rossi, G. H. (2016, 12 abril). Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales. <https://sedici.unlp.edu.ar/handle/10915/52183>

# Arquitectura Reutilizable para Desarrollo de Aplicaciones Web

Este trabajo busca desarrollar una herramienta que reduzca tiempo y costos en el desarrollo de aplicaciones web mediante un framework reutilizable basado en patrones de diseño e interfaces. La solución aborda problemas comunes de desarrollo y diseño de interfaces, permitiendo generar o extender aplicaciones web funcionales de forma rápida y sencilla. El framework fue aplicado en proyectos de gestión de contenidos periodísticos, comparando su eficiencia con métodos tradicionales usando COCOMO II y COCOMO clásico. Los resultados mostraron que el framework requiere menos tiempo y esfuerzo, destacando COCOMO II como el método más preciso.



## Reflexión

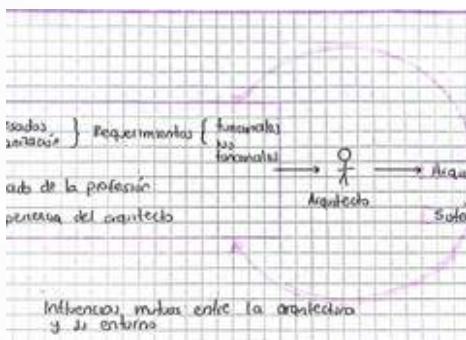
El desarrollo de un framework reutilizable para aplicaciones web ofrece varias ventajas. Basado en patrones de diseño e interfaces y tecnologías modernas, permite generar aplicaciones web de forma más rápida y eficiente, reduciendo tiempos y costos frente a métodos tradicionales. La reutilización de componentes mejora la calidad, usabilidad y mantenimiento de las aplicaciones, como se demostró en las evaluaciones realizadas. Además, fomenta arquitecturas de software robustas y escalables, proporcionando una base reutilizable. Este enfoque también aumenta la productividad y satisfacción de los programadores al ofrecer herramientas estandarizadas que optimizan su trabajo.

## Bibliografía

Murillo, M. [Martín Miguel Murillo]. (2009). ARQUITECTURA SOFTWARE REUTILIZABLE BASADA EN PATRONES DE DISEÑO Y PATRONES DE INTERACCIÓN, PARA EL DESARROLLO RÁPIDO DE APLICACIONES WEB [Tesis, Universidad Nacional de Santiago del Estero Facultad de Ciencias Exactas y Tecnologías].<https://n9.cl/36fcsk>

# Introducción A La Arquitectura De Software

La arquitectura de software, surgida en los años 90 como un nivel superior al diseño tradicional, fue impulsada por la Carnegie-Mellon University y el Software Engineering Institute. El proceso arquitectónico incluye la elección del estilo, selección de patrones y diseño de componentes, aunque se enseña en orden inverso. Influenciada por factores técnicos y no técnicos, se enmarca en el Architecture Business Cycle (ABC). Los requerimientos no funcionales son clave, pues su modificación posterior es compleja y costosa. La arquitectura destaca por sus conectores complejos, restricciones importantes y componentes variados, usando estilos que describen sistemas completos y no solo partes.



## Bibliografía

Cristiá, M. (2007). Introducción a la arquitectura de software. Centro Internacional Franco-Argentino de Ciencias de la Información y de Sistemas. <https://n9.cl/vwg2l>

## Reflexión

La arquitectura de software, en constante evolución, impacta significativamente el diseño de sistemas al abstraer y simplificar la complejidad. Permite a desarrolladores y analistas enfocarse en los aspectos clave del sistema. Su proceso se divide en etapas como elección del estilo arquitectónico, selección de patrones y diseño de componentes, abordando la complejidad de forma sistemática. Además, el ciclo de negocios de la arquitectura (ABC) destaca la influencia de factores técnicos, sociales y de negocio, promoviendo una perspectiva integral para crear soluciones que satisfagan tanto a usuarios como a organizaciones.

# Modelado y Verificación de Patrones de Arquitectura para la Nube

Los arquitectos de software enfrentan desafíos al diseñar sistemas para entornos de computación en la nube, especialmente al intentar garantizar la calidad del software, ya que las funcionalidades son independientes de los atributos de calidad. La dificultad radica en la falta de un lenguaje adecuado para expresar estos atributos, la falta de documentación sobre patrones arquitectónicos y las limitaciones para generar diseños basados en ellos. La adopción de la nube obliga a cambios en normas, estándares e infraestructuras, y los patrones de computación en la nube aún no están completamente establecidos, lo que ralentiza el proceso de diseño.

Patrones de diseño	Disecciones Arquitectónicas	Dependencias de Infraestructura
Instanciación de Modelos Arquitectónicos		
Metamodelo Arquitectónico		

## Reflexión

La arquitectura de software, en constante evolución, impacta significativamente el diseño de sistemas al abstraer y simplificar la complejidad. Permite a desarrolladores y analistas enfocarse en los aspectos clave del sistema. Su proceso se divide en etapas como elección del estilo arquitectónico, selección de patrones y diseño de componentes, abordando la complejidad de forma sistemática. Además, el ciclo de negocios de la arquitectura (ABC) destaca la influencia de factores técnicos, sociales y de negocio, promoviendo una perspectiva integral para crear soluciones que satisfagan tanto a usuarios como a organizaciones.

## Bibliografía

- Blas, M. J., Leone, H., & Gonnet, S. (2019). Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube. RISTI - Revista Ibérica de Sistemas E Tecnologias de Informação, 35, 1-17. <https://doi.org/10.17013/risti.35.1-17>

# Aplicación de Patrones de Diseño para Garantizar Alta Flexibilidad en el Software

Las empresas cambian constantemente sus reglas de negocio para mantenerse competitivas, por lo que las aplicaciones de software empresarial deben ser flexibles para adaptarse a estos cambios de manera ágil y eficiente, sin requerir grandes recursos. Para lograr esta flexibilidad, el software debe ser diseñado con la posibilidad de modificaciones, especialmente en las reglas de negocio. Los patrones de diseño, como Estrategia, Compuesto y Fábrica, juegan un papel clave al solucionar problemas de adaptabilidad, permitiendo desarrollar software flexible frente a cambios.

## Reflexión



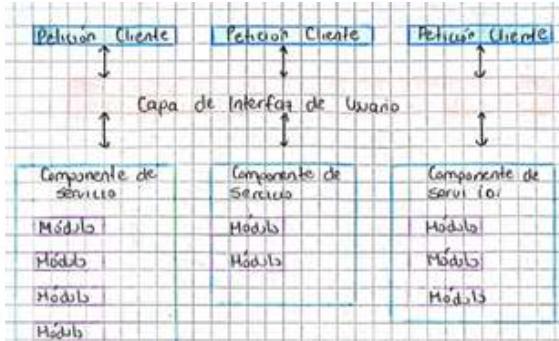
## Bibliografía

- Escalante, L. C. (2016). Aplicación de Patrones de Diseño para Garantizar Alta Flexibilidad en el Software. REVISTA TECNOLOGÍA & DESARROLLO, 12(1), 77-82. <https://doi.org/10.18050/td.v12i1.696>

La capacidad de adaptación rápida de las empresas a cambios en las reglas de negocio es clave para mantener la competitividad. Para ello, es crucial el diseño de software flexible, que permita evolucionar con las necesidades del negocio. Adoptar principios de diseño como modularidad y adaptabilidad, mediante patrones como Estrategia, Compuesto y Fábrica, facilita separar las reglas de negocio de la lógica de programación, mejorando la flexibilidad. Esta adaptabilidad reduce costos y mejora la eficiencia al evitar costosas reescrituras, acelerando el retorno de la inversión y aumentando la productividad.

# Arquitectura de Microservicios para Desarrollo Web

El proceso de desarrollo de software de la CGTIC de la Asamblea Nacional del Ecuador emplea una arquitectura monolítica, lo que ha generado dificultades en mantenimiento, escalabilidad y entregas. Este estudio tuvo como objetivo identificar las tecnologías, metodologías y arquitectura utilizadas, y explorar alternativas para la implementación de microservicios. Con un enfoque cualitativo, se aplicó un grupo focal y revisión bibliográfica sobre microservicios. El análisis identificó el estado del arte y los requisitos para desarrollar aplicaciones web, proponiendo una arquitectura de software adecuada.



## Reflexión

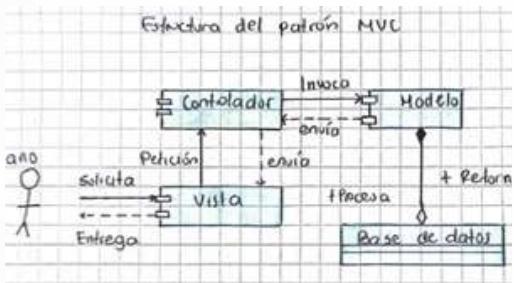
La adopción de una arquitectura monolítica en la CGTIC de la Asamblea Nacional del Ecuador ha generado desafíos en mantenimiento, escalabilidad y entrega de aplicaciones web. La investigación busca identificar tecnologías, metodologías y arquitecturas para mejorar el desarrollo de aplicaciones web, enfocándose en microservicios. Este enfoque modular, escalable y mantenable promete mejorar la agilidad y adaptabilidad ante cambios en los requisitos del negocio. La transición hacia microservicios ofrecerá una solución más eficiente y de mayor calidad para las necesidades actuales y futuras de la Asamblea Nacional.

## Bibliografía

- Lopez, D., & Maya, E. (2017). Arquitectura de software basada en microservicios para desarrollo de aplicaciones web [Tesis, Universidad Técnica del Norte, Instituto de Posgrados]. En Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017 Gestión de las TICs para la Investigación y la Colaboración, San José, del XX al XX de julio de 2017. <https://n9.cl/0tbddk>

# Análisis Comparativo De Patrones De Diseño De Software

Los patrones de diseño ofrecen soluciones a problemas comunes en el desarrollo de software, evitando duplicación de código y facilitando su reutilización. Este artículo analiza los patrones Template Method, Model-View-Controller, Model-View-Presenter, Front Controller y MVVM, destacando sus ventajas y desventajas. A través de una revisión bibliográfica y comparación de métricas, se concluye que no hay un patrón superior, ya que cada uno tiene un propósito específico. Los patrones de diseño permiten una mejor organización del código, mejorando la calidad, mantenimiento y comprensión del software.



## Bibliografía

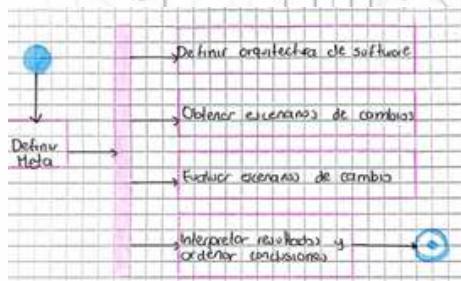
- Álvarez, O. D. G., Larrea, N. P. L., & Valencia, M. V. R. (2022). Análisis comparativo de Patrones de Diseño de Software. Dialnet. <https://n9.cl/qcniy>

## Reflexión

La investigación destaca que el rol del desarrollador es identificar el patrón de diseño más adecuado para la solución que se está construyendo. El conocimiento y comprensión de estos patrones es esencial para tomar decisiones informadas y crear software de calidad. Además de la comparación entre patrones, el artículo subraya que los patrones de diseño ayudan a mantener una organización lógica en el código, promoviendo la modularidad, el mantenimiento y la comprensión del sistema. Esto facilita la evolución del software, su adaptación a cambios y mejora la calidad del producto final.

# Marco para Selección de Arquitectura de Software

Este documento identifica y caracteriza patrones de software que ofrecen soluciones comprobadas en varios niveles de abstracción, desde arquitecturas de información hasta patrones de navegación e interacción. Se propone un marco de trabajo para seleccionar el patrón adecuado según el contexto de la aplicación a desarrollar, basado en la clasificación de proyectos de software. A través de una encuesta, se determinó que el patrón en la nube es el más recomendado (40%), seguido por MVC (26,7%) y microservicios (13,3%). También se destacó la arquitectura MVP y MVC para dispositivos móviles y sitios web.



## Reflexión

La investigación destaca que el rol del desarrollador es identificar el patrón de diseño más adecuado para la solución que se está construyendo. El conocimiento y comprensión de estos patrones es esencial para tomar decisiones informadas y crear software de calidad. Además de la comparación entre patrones, el artículo subraya que los patrones de diseño ayudan a mantener una organización lógica en el código, promoviendo la modularidad, el mantenimiento y la comprensión del sistema. Esto facilita la evolución del software, su adaptación a cambios y mejora la calidad del producto final.

## Bibliografía

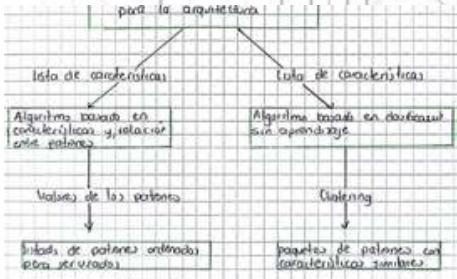
- Camilo, G. M. J., & Alberto, V. A. F. (2020, 28 agosto). Marco de trabajo para la selección de la arquitectura de un proyecto de software mediante la aplicación de patrones arquitectónicos. <https://n9.cl/63xm0m>

# Modelo para Selección de Patrones de Desarrollo

Los patrones de diseño son fundamentales para obtener software de calidad, pero muchos desarrolladores, diseñadores y arquitectos carecen de los conocimientos necesarios para aplicarlos correctamente. Esta tesis propone un modelo basado en inteligencia artificial para ayudar en la toma de decisiones sobre qué patrón aplicar en situaciones específicas. El modelo investiga la aplicación de patrones de diseño en la arquitectura de software y sistemas, presentando algoritmos que facilitan la decisión, ayudando a profesionales inexpertos a seleccionar el patrón adecuado en el desarrollo de software.

## Bibliografía

Cuesta Villa, M., & Piñero Pérez, P. (2007). MODELO PARA LA AYUDA A LA TOMA DE DECISIONES EN LA SELECCIÓN DE PATRONES DE DESARROLLO DE SOFTWARE. [Tesis, Universidad de las Ciencias Informáticas]. <https://n9.cl/qa465>

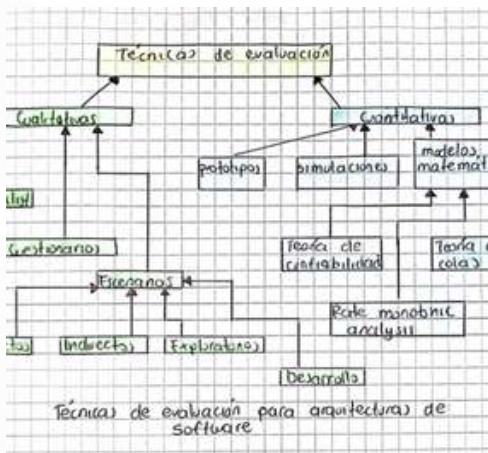


## Reflexión

La complejidad de los patrones de diseño y la falta de madurez en su aplicación requieren herramientas que complementen el conocimiento de los equipos de desarrollo. Este modelo, al caracterizar los patrones y utilizar algoritmos para guiar su selección, busca facilitar la toma de decisiones informadas, incluso para profesionales menos experimentados. En resumen, el enfoque basado en inteligencia artificial representa un avance hacia la consolidación de los patrones de diseño como práctica estándar en la industria del software, beneficiando a organizaciones y profesionales.

# Evaluación de Arquitecturas de Software en el Sector Empresarial

El éxito de un sistema de software depende de su arquitectura, por lo que es crucial evaluarla. Evaluar arquitecturas permite identificar riesgos y verificar que se cumplan los requisitos de calidad. Este documento presenta el proceso PEASSE, diseñado para evaluar arquitecturas de software en el sector empresarial. PEASSE identifica patrones de diseño y utiliza métodos de evaluación existentes para analizar la arquitectura y asegurar que cumpla con los estándares de calidad y funcionalidad necesarios.



## Bibliografía

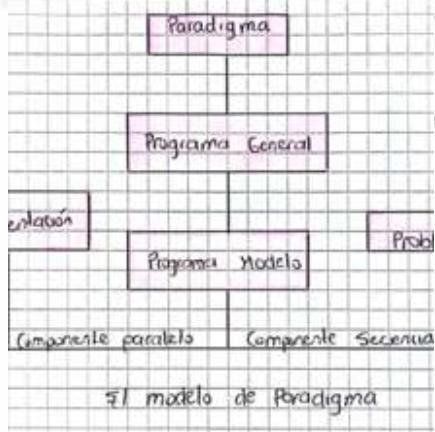
Item 1008/105 | Repositorio CIMAT. (2016, 15 noviembre). <https://cimat.repositorioinstitucional.mx/jspui/handle/1008/105>

## Reflexión

La evaluación de la arquitectura de software es clave para el éxito de los sistemas empresariales, como lo muestra el proceso PEASSE. No solo se trata de diseñar e implementar una arquitectura, sino de evaluarla para identificar riesgos y asegurar que cumpla con los requisitos de calidad. PEASSE, al integrar patrones de diseño y métodos de evaluación existentes, ofrece un enfoque estructurado para validar decisiones arquitectónicas. Este proceso ayuda a optimizar la arquitectura antes de la implementación, asegurando el éxito del proyecto y alineando el sistema con las necesidades del negocio.

# Implementación de Patrones de Diseño Paralelos en Java

El trabajo propone diseñar e implementar algoritmos paralelos en JAVA para resolver problemas como ordenación o simulación mediante una biblioteca de clases de Objetos Paralelos. Los algoritmos se estructuran como Patrones de Diseño Paralelos, donde cada diseño representa la estructura paralela común a una técnica algorítmica. Se utiliza la técnica de "divide y vencerás" como Patrón de Diseño Paralelo, implementando aplicaciones como Quicksort y Búsqueda Binaria de forma concurrente mediante hilos. El documento explica el desarrollo de estos patrones paralelos.



## Bibliografía

López, R., Toledo, V., & Torres, P. (s. f.). Implementación de patrones de diseño paralelos en JAVA como una biblioteca de clases de objetos paralelos [Tesis, Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Computación]. <https://n9.cl/qhvzw2>

## Reflexión

La propuesta de implementar algoritmos paralelos mediante patrones de diseño en Java simplifica la programación paralela al usar una biblioteca de clases de Objetos Paralelos, haciendo tecnologías avanzadas accesibles. El patrón "divide y vencerás" aplicado en algoritmos como Quicksort y Búsqueda Binaria demuestra cómo los conceptos teóricos de la programación paralela se traducen en soluciones prácticas y reutilizables. Esta metodología mejora la eficiencia, proporciona estructura para el desarrollo de software paralelo y facilita la creación de soluciones escalables y mantenibles.

# Documentando La Arquitectura De Software

El documento discute las mejores prácticas para documentar la arquitectura de software, destacando enfoques como el modelo de 4+1 vistas de Kruchten y las cuatro vistas de Nord et al. Se enfoca en la evolución hacia una documentación centrada en las vistas más útiles para los interesados y en las cualidades clave del sistema. También aborda el marco "Vistas y más allá" del SEI, que prioriza las vistas basadas en intereses específicos y el uso de estilos arquitectónicos para cumplir atributos de calidad, junto con el estándar IEEE 1471 para describir arquitecturas de manera flexible.

## Reflexión

La evolución en la documentación de arquitectura de software refleja un enfoque centrado en el usuario y los requisitos no funcionales. Los marcos modernos recomiendan personalizar la documentación según las necesidades e intereses de los interesados, permitiendo a los arquitectos enfocarse en los aspectos más relevantes y comunicarse de manera accesible. Además, el énfasis en el análisis y la justificación arquitectónica resalta su rol integral en el desarrollo de software, indicando una maduración de la práctica con un enfoque en alinear el diseño con los objetivos de calidad y las necesidades de los interesados.

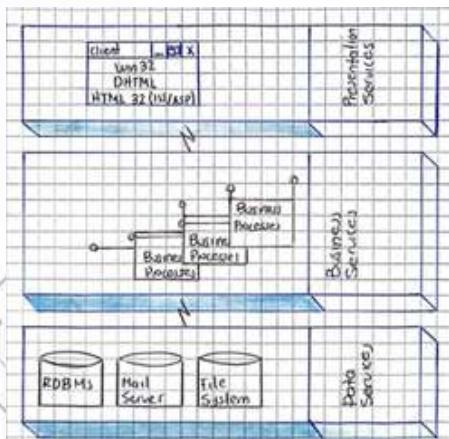
## Bibliografía

Gómez, O. (s. f.). Documentando la arquitectura de software [Tesis]. Osgg. Recuperado 10 de noviembre de 2024, de <https://n9.cl/wvImw>

Extracto del marco conceptual del Estándar IEEE 1471.

# Introducción A La Arquitectura De Software

El documento ofrece una introducción a la Arquitectura de Software (AS) y su relación con la estrategia arquitectónica de Microsoft, destacando la falta de textos actualizados en español y la brecha entre la academia y la industria. A través de un análisis del Microsoft Solutions Framework (MSF), el texto busca conectar este marco con las tendencias actuales en teoría y práctica arquitectónica. El objetivo es establecer una base sólida para la discusión teórica y práctica de modelos de desarrollo basados en arquitectura, invitando a la comunidad de arquitectos a enriquecer y expandir los contenidos presentados.



## Reflexión

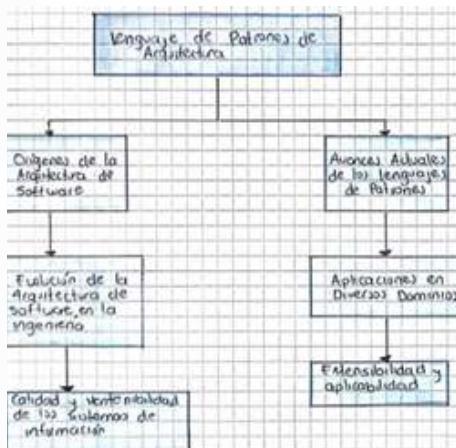
La reflexión en el texto es la idea de que la Arquitectura de Software no debe ser vista sólo como una disciplina académica o teórica, sino como un área estrechamente conectada con las necesidades reales y cambiantes de la industria tecnológica; el vacío entre la academia y la industria, especialmente en cuanto a la resolución de problemas prácticos, subraya la importancia de integrar la teoría y la práctica de manera efectiva. Si se logra este vínculo, no solo mejorarán las soluciones tecnológicas sino que también se formarán profesionales más completos, capaces de abordar desafíos tanto desde un enfoque conceptual como práctico.

## Bibliografía

Reynoso, C. B. (2004). Introducción a la arquitectura software [Proyecto, Universidad de Buenos Aires]. <https://n9.cl/x380cs>

# Lenguajes de Patrones de Arquitectura de Software: Estado del Arte

El artículo analiza la evolución de los lenguajes de patrones en la arquitectura de software, desde sus inicios hasta los avances actuales. Su objetivo es resaltar cómo estos lenguajes han transformado el diseño y la estructura de los sistemas de software en diversos dominios. Los lenguajes de patrones son herramientas conceptuales que facilitan la creación de arquitecturas al ofrecer soluciones a problemas comunes en el diseño. Permiten a los desarrolladores reutilizar patrones probados, adaptándolos a sus necesidades, reduciendo la complejidad y mejorando la calidad del diseño.



## Reflexión

Al leer el artículo se crea una conciencia sobre la importancia de encontrar soluciones reutilizables y adaptables en un mundo cada vez más complejo y cambiante. Los patrones de diseño no solo representan herramientas técnicas, sino una filosofía de trabajo colaborativo, donde los ingenieros pueden apoyarse en conocimientos previos para evitar reinventar la rueda. Esto no sólo acelera el desarrollo, sino que también mejora la calidad de los sistemas y facilita su mantenimiento a largo plazo; vivimos en una era donde la extensibilidad y la flexibilidad son esenciales para afrontar la rapidez con la que evolucionan las tecnologías.

## Bibliografía

- Jimenez Torres, V. H., Tello Borja, W., & Ríos Patiño, J. I. (2013). Lenguajes de patrones de arquitectura de software: una aproximación al estado del arte [Tesis, Universidad Tecnológica de Pereira]. <https://n9.cl/evq16>

# Patrones GOF en el Desarrollo de Aplicaciones Web

Este artículo examina el uso de los patrones de diseño GOF (Gang of Four) en el desarrollo de software en las Unidades Tecnológicas de Santander, Colombia. Basado en un catálogo de 23 patrones de diseño, la investigación evalúa su aplicación en la práctica de programación orientada a objetos (POO) y su calidad en la industria colombiana. La muestra se seleccionó considerando factores como el número de desarrolladores, uso de UML, criticidad del sistema y costos del proyecto, para reflejar las mejores prácticas en el desarrollo de software en el país.

no	Patrón de Diseño	Categoría
1	Builder	
2	Factory Method	Creacionales
3	Singleton	
4	Decorator	Estructurales
5	Facade	
6	Iterator	
7	Strategy	De comportamiento
8	Template Method	

## Bibliografía

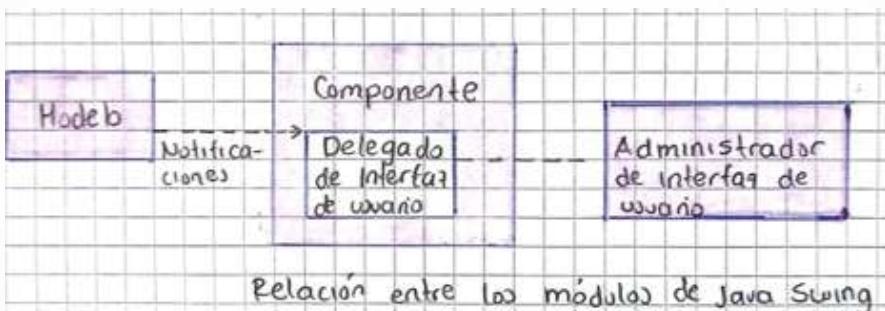
Patrones de diseño GOF (The Gang of Four) en el contexto de procesos de desarrollo de aplicaciones orientadas a la web. (2012). [Tesis, Información Tecnológica]. <https://n9.cl/44wvd1>

## Reflexión

La investigación sobre los patrones y su aplicación en el contexto del software subraya la importancia de estandarizar las prácticas de diseño en la ingeniería de software. El uso de patrones de diseño bien establecidos no solo mejora la calidad y la eficiencia del proceso de desarrollo, sino que también facilita la mantenibilidad y escalabilidad de los sistemas a largo plazo. Al aplicar estos patrones, los equipos de desarrollo pueden construir soluciones más modulares y reutilizables.

# Patrón MVC y su Implementación en Java Swing

El patrón de diseño Modelo-Vista-Controlador (MVC) divide una aplicación en tres componentes: Modelo, Vista y Controlador, lo que mejora la organización del código y facilita la separación de responsabilidades. Originado en SmallTalk en los años 70, sigue siendo clave en la arquitectura de aplicaciones modernas, especialmente en interfaces gráficas. Java, con su portabilidad y simplicidad, usa Java Swing para implementar componentes visuales como botones y tablas, manteniendo la independencia de la plataforma y permitiendo cambiar el estilo de la interfaz en tiempo de ejecución gracias a MVC.



## Bibliografía

Bascón Pantoja, E. (2004). El Patrón de Diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing [Tesis, Universidad Católica Boliviana]. <https://n9.cl/6zf9t>

## Reflexión

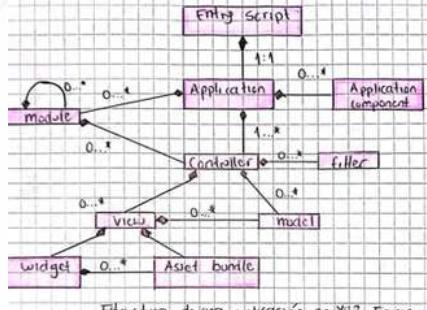
El patrón MVC ha sido uno de los pilares fundamentales en el desarrollo de software, especialmente en el diseño de interfaces gráficas de usuario, y sigue siendo relevante en la actualidad. El hecho de que el modelo no dependa de las vistas ni del controlador refuerza el principio de modularidad, permitiendo que el mismo modelo de datos pueda ser reutilizado en diferentes interfaces, sin necesidad de modificar la lógica de negocio.

# MVC en Yii2 para Módulo de Consulta SIHR

El enfoque arquitectónico MVC utilizado en el desarrollo de aplicaciones para organizar los componentes de presentación de manera eficiente divide la aplicación en tres partes, dentro de sus ventajas destaca la posibilidad de modificar interfaces sin afectar la lógica y mejora la mantenibilidad, pero presenta desventajas como la necesidad de una mayor dedicación inicial en el desarrollo como una mayor complejidad al gestionar múltiples capas. Un ejemplo de la aplicación es el Yii2 Framework, que facilita la implementación de MVC en el desarrollo de aplicaciones web, como en el caso del Sistema Integral Hospital Rovirosa, donde se desarrolló un módulo de Consulta Externa utilizando este Framework.

## Bibliografía

Figueroa Escudero, W. O., & Pérez Vasconcelos, M. (2020). Aplicación del Patrón de Diseño MVC utilizando YII2 Framework en el desarrollo del Módulo de Consulta Externa del Sistema Integral Hospital Rovirosa (SIHR). Wordpress.  
<https://acortar.link/3PhESz>

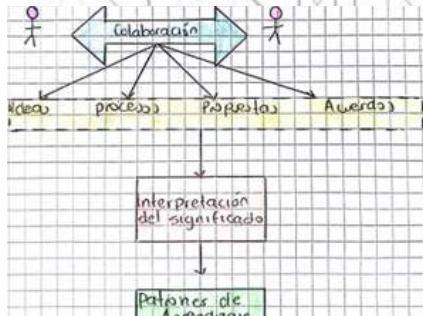


## Reflexión

El patrón de diseño MVC nuevamente demuestra ser fundamental en el desarrollo de software moderno, ya que reduce el riesgo de introducir errores cuando se hacen cambios en una parte del sistema lo que lo convierte en una opción atractiva para proyectos grandes y dinámicos. Este tipo de patrones junto con Frameworks como Yii2 no solo optimizan el desarrollo sino que también permiten que las aplicaciones puedan evolucionar con el tiempo, respondiendo a las necesidades cambiantes de los usuarios y el mercado.

# Patrones de diseño en Generative Learning Object

Este artículo explora los patrones de diseño aplicados a los objetos de aprendizaje generativos (GLO), explicando su propósito, clasificación y ejemplos. Se destaca cómo estos patrones, inicialmente de la arquitectura, se han adaptado a la informática y la educación, ofreciendo soluciones reutilizables en el diseño de sistemas educativos, especialmente en e-learning. Se mencionan proyectos como el CETL, que agrupa universidades como London Metropolitan, Cambridge y Nottingham, para crear repositorios de objetos de aprendizaje reutilizables (RLO) y desarrollar GLO para contenido educativo flexible.



## Reflexión

Los patrones de diseño proporcionan soluciones probadas para problemas recurrentes, y su aplicación en la educación digital es clave en un mundo orientado al aprendizaje en línea. La reutilización de objetos de aprendizaje, como los GLO, mejora la eficiencia en el desarrollo de materiales educativos y fomenta la colaboración entre instituciones. Sin embargo, los patrones no son soluciones universales; deben adaptarse a las necesidades específicas de los usuarios y al contexto del entorno educativo.

## Bibliografía

- Tello, J. C. (2009). Patrones de diseño: ejemplo de aplicación en los Generative Learning Object. <https://n9.cl/ndjvr2>

# Patrones, Refactorización y Antipatrones en OOP

El autor destaca que, aunque la ingeniería de software ha avanzado con la adopción de la orientación a objetos, muchos sistemas siguen siendo rígidos y difíciles de adaptar. Se abordan los conceptos de Patrones de Diseño, Antipatrones y Refactorización. Los patrones son soluciones probadas a problemas comunes en el diseño, mientras que los antipatrones representan soluciones erróneas que afectan negativamente al sistema. La refactorización transforma un antipatrón en una solución adecuada. Además, se explica cómo los patrones de diseño, originados en la arquitectura, ofrecen flexibilidad, reutilización y mantenimiento dentro del código.



## Reflexión

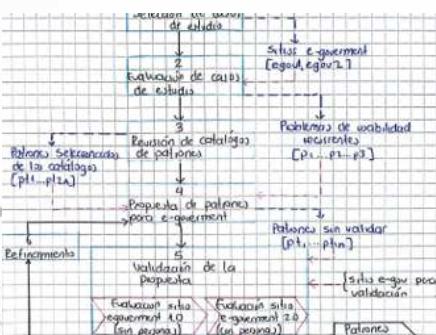
El uso de patrones de diseño y antipatrones en el desarrollo de software ha avanzado significativamente, ofreciendo soluciones probadas para problemas comunes, mejorando la eficiencia y promoviendo la colaboración entre desarrolladores. No obstante, su implementación debe ser cautelosa, ya que aplicarlos incorrectamente puede aumentar la complejidad y generar nuevos problemas. La refactorización es clave en este proceso, ya que permite adaptar y evolucionar el software de manera ordenada, eliminando malas prácticas y manteniendo el código limpio y fácil de modificar.

## Bibliografía

Campo, G. D. (2009). Patrones de diseño, refactorización y antipatrones.  
<https://n9.cl/pwj0n2>

# Patrones de Usabilidad para Sitios de Gobierno Electrónico

Este artículo analiza la evolución de los sitios de e-government en Ecuador, enfocándose en la usabilidad y la experiencia del usuario. Aunque han pasado de ser informativos a interactivos, muchos sitios carecen de un diseño intuitivo, lo que genera desconfianza. A través de una evaluación empírica, se identificaron problemas comunes como vínculos mal diferenciados y formularios confusos. Se propusieron patrones de usabilidad para abordar estos problemas, como mejorar la visibilidad de los vínculos, proporcionar retroalimentación clara y asegurar la transparencia, lo que mejoró significativamente la experiencia del usuario.



## Reflexión

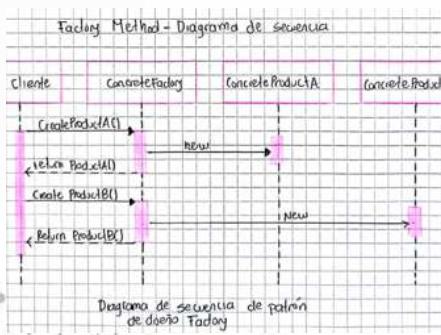
Este estudio resalta un aspecto crucial de la administración pública digital: La importancia de diseñar sitios web de gobierno que no solo ofrecen servicio en línea, sino que también sean fáciles de usar y generen confianza en los ciudadanos. La accesibilidad, la claridad en la información, y la transparencia no son solo deseos estéticos o técnicos, sino principios fundamentales para asegurar una interacción efectiva entre los ciudadanos y el gobierno. El diseño adecuado de estos sitios puede tener un impacto directo en la participación cívica, la eficiencia administrativa y la legitimidad de las instituciones públicas.

## Bibliografía

- Cordovez, P., Jiménez, C., & Lata, V. (2018). Patrones de usabilidad para sitios de gobierno electrónico. NOVASINERGIA REVISTA DIGITAL DE CIENCIA INGENIERÍA y TECNOLOGÍA, 1(1), 41-50.  
<https://doi.org/10.37135/unach.n.s.001.01.05>

# Introducción a los Patrones de Diseño

El libro aborda el uso de patrones de diseño en la programación de software, destacando su importancia para la madurez de los desarrolladores al utilizar soluciones probadas para problemas comunes. Los patrones de diseño son soluciones documentadas que ahorran tiempo, evitan errores y crean un lenguaje común entre los programadores. Se dividen en tres categorías: creacionales (cómo se crean los objetos), estructurales (organización y relación entre clases) y de comportamiento (asignación de responsabilidades e interacción entre objetos).



## Reflexión

Los patrones de diseño son un testimonio del valor de la experiencia en la programación. Al utilizar soluciones ya comprobadas, evitamos "reinventar la rueda" y aprovechamos las mejores prácticas desarrolladas por otros. Sin embargo, es importante recordar que los patrones no deben ser utilizados de manera rígida; su efectividad depende de la correcta identificación del problema y de su adecuado contexto de uso. No son una solución mágica para todos los problemas de diseño, pero son herramientas poderosas que, bien aplicadas, nos permiten construir sistemas más robustos, mantenibles y escalables.

## Bibliografía

Introducción a los patrones de diseño: Un enfoque práctico (Primera Edición, Vol. 1). (2016). [Google Scholar]. Oscar Javier Blancarte Iturrealde. <https://acortar.link/JOws56>

# Patrones de diseño

El texto introduce los patrones de diseño en programación orientada a objetos (POO), destacando cómo resuelven problemas comunes en el diseño de sistemas. La POO plantea retos como la identificación y estructura de clases, así como su reutilización eficiente. Los patrones de diseño son soluciones generales a problemas recurrentes en contextos específicos, proporcionando una forma estandarizada de resolver problemas comunes. Estos patrones no deben confundirse con bibliotecas de clases, frameworks o herramientas de refactorización.



## Bibliografía

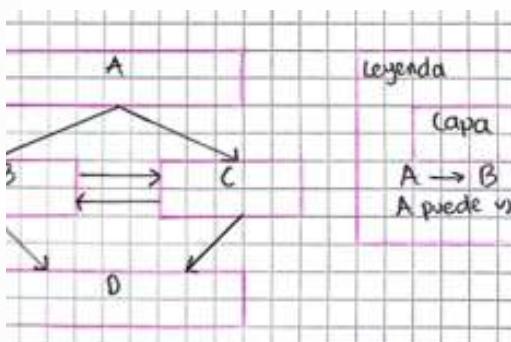
Patrones de diseño. (2004). Infor. Recuperado 12 de noviembre de 2024, de <https://www.infor.uva.es/~felix/datos/pri/iitema7.pdf>

## Reflexión

El uso de patrones de diseño en programación orientada a objetos facilita la creación de soluciones estructuradas y eficaces. Ayudan a superar desafíos como identificar clases, organizarlas correctamente y asegurar la reutilización del código. Los patrones proporcionan soluciones probadas para problemas recurrentes, mejorando la eficiencia y fomentando la comunicación entre equipos de desarrollo. Al ser soluciones estándar, permiten a los desarrolladores compartir un "vocabulario común", facilitando la colaboración y reduciendo errores.

# Arquitectura del Software

El texto describe diferentes tipos de arquitecturas de software: no distribuidas (monolíticas), distribuidas (componentes en varias máquinas) y arquitecturas híbridas o especializadas. El diseño arquitectónico es crucial para estructurar el sistema, conectando los requisitos con decisiones técnicas. Se distingue entre arquitectura interna (estructura de programas) y externa (organización global). La arquitectura afecta a los requisitos no funcionales como rendimiento y seguridad. La arquitectura en capas organiza el sistema en niveles diferenciados, con ventajas como la facilidad de cambios y desventajas como la complejidad y problemas de rendimiento.



## Bibliografía

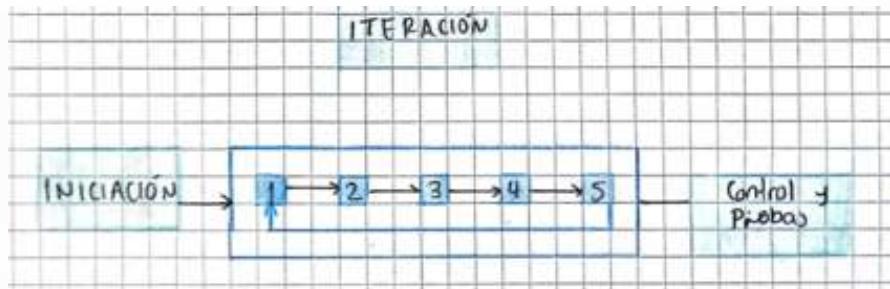
- M. Castro, L. (2015). Arquitectura Software (Primera, Vol. 1) [Google Scholar]. Universidade da coruña. <https://n9.cl/xhdpf>

## Reflexión

El diseño arquitectónico de sistemas de software es fundamental para garantizar el cumplimiento de los requisitos funcionales y no funcionales, abordando aspectos como rendimiento, seguridad y escalabilidad. Los patrones y modelos arquitectónicos facilitan este proceso, ofreciendo soluciones adaptables. La arquitectura en capas, que organiza el sistema en componentes manejables, mejora la flexibilidad y la adaptación del sistema. No obstante, el desafío radica en lograr una división clara de responsabilidades y optimizar la comunicación entre capas, evitando la sobrecarga del sistema.

# Arquitectura de Software para Gestión de Información

El documento describe la creación de una arquitectura de software para gestionar grandes volúmenes de datos no estructurados mediante un sistema cliente-servidor. Utiliza tecnologías como HTTP, REST, API, Node.js, Express.js y bases de datos NoSQL como MongoDB, Memcached y Redis, ideales por su escalabilidad y rendimiento. Se enfoca en una arquitectura modular y flexible que optimiza la comunicación asíncrona entre el cliente y el servidor, mejorando la velocidad y eficiencia en entornos con alta demanda, sin necesidad de hardware sofisticado.



## Reflexión

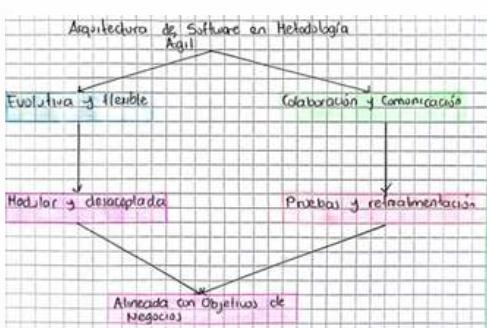
El desarrollo de sistemas para manejar grandes volúmenes de datos no estructurados es esencial en un entorno digital de alta velocidad. La transición a bases de datos NoSQL refleja la necesidad de flexibilidad y velocidad para aplicaciones modernas. Gestionar información sin estructuras rígidas fomenta la innovación en software, beneficiando tanto a grandes empresas como a aplicaciones cotidianas como redes sociales y servicios en la nube. Las arquitecturas asíncronas y las bases NoSQL son claves para garantizar escalabilidad y eficiencia, permitiendo que las aplicaciones se adapten sin comprometer el rendimiento.

## Bibliografía

- Daza Díaz, D. C., & Florez Mendoza, A. M. (2013). DESARROLLO DE UNA ARQUITECTURA DE SOFTWARE PARA GESTIÓN DE INFORMACIÓN NO ESTRUCTURADA [Tesis, Universidad Militar Nueva Granada Facultad de Ingeniería]. <https://n9.cl/jhidc3>

# Arquitectura de Software

La revista destaca la arquitectura de software en un enfoque ágil, que se desarrolla de manera iterativa y flexible, priorizando la entrega continua de valor al cliente y adaptándose a cambios. En lugar de una arquitectura rígida, se permite su evolución durante el desarrollo, basándose en simplicidad, comunicación y colaboración. El diseño busca ser modular y desacoplado, facilitando cambios sin afectar todo el sistema. Se promueven tecnologías que apoyen el desarrollo rápido, como frameworks para integración continua, asegurando que las decisiones arquitectónicas estén alineadas con los objetivos del negocio.



## Bibliografía

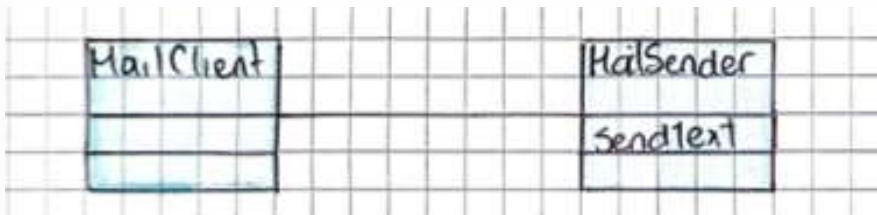
- Fernández, L. F. (2006). Arquitectura de software: Metodologías ágiles. Software Guru, 3, 2-4. <https://n9.cl/hamps>

## Reflexión

La integración de la metodología ágil en la arquitectura de software permite una visión dinámica y flexible, adaptándose a cambios rápidos en el entorno tecnológico. A diferencia de enfoques tradicionales, el enfoque ágil prioriza el valor incremental y la alineación constante con las necesidades del cliente. Aunque menos predecible al principio, ofrece mayor flexibilidad y adaptabilidad, mitigando riesgos al identificar y corregir errores rápidamente. Este enfoque mejora la calidad del producto, optimiza recursos y fomenta una colaboración estrecha entre desarrolladores y usuarios, viendo la evolución como una oportunidad.

# Arquitectura de Software

El texto aborda la arquitectura de software desde un enfoque pedagógico, destacando la colaboración y los roles complementarios en su desarrollo. Utiliza el ejemplo de un sistema bancario ("i-Banco Casero") para ilustrar problemas comunes, como concurrencia y escalabilidad, y soluciones arquitectónicas, como control de concurrencia y replicación. Además, se enfoca en la evaluación de arquitecturas según requisitos como rendimiento y tolerancia a fallos, destacando la importancia de los stakeholders en la toma de decisiones para seleccionar la arquitectura adecuada.



## Reflexión

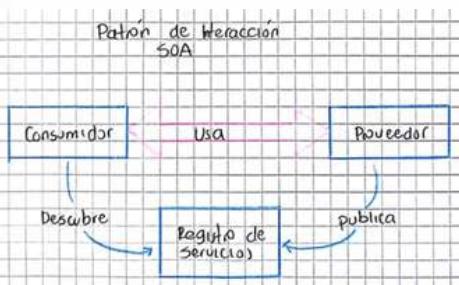
la importancia de una perspectiva sistémica en la arquitectura de software, integrando objetivos de negocio con decisiones técnicas y priorizando propiedades como escalabilidad, confiabilidad y flexibilidad. Se destaca que la arquitectura es un proceso iterativo, no estático, que se adapta a problemas reales y soluciones continuas. La implicación de los stakeholders en la evaluación de arquitecturas subraya la naturaleza interdisciplinaria de la disciplina. Este enfoque sirve como base para reflexionar sobre la importancia de una buena arquitectura en el éxito de los sistemas.

## Bibliografía

Astudillo, H. (2014). Arquitectura de software (Departamento de Informática, Ed.) [Diapositivas; PDF]. <https://acortar.link/BGCNOJ>

# Arquitectura de Software, esquemas y servicios

la evolución de las aplicaciones empresariales hacia arquitecturas distribuidas, destacando el rol de la arquitectura de software y del arquitecto. Se enfatiza la importancia de los esquemas para evitar duplicación de código y reducir la complejidad, aunque integrar estos esquemas en sistemas existentes presenta desafíos. Introduce la arquitectura orientada a servicios (SOA) como solución, permitiendo que los procesos empresariales se conceban como servicios independientes, escalables y adaptativos, aunque enfrenta desafíos técnicos, como tiempos de respuesta y confiabilidad en la red.



## Bibliografía

- Romero, P. Á. (2005). Arquitectura de software, esquemas y servicios (21.a ed.) [Tesis; Google Scholar]. Imprenta Cuaje. <https://dialnet.unirioja.es/servlet/articulo?codigo=4786655>

## Reflexión

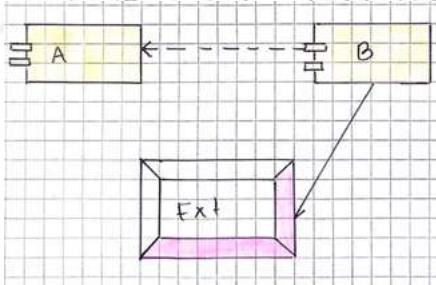
destaca cómo la complejidad de las aplicaciones empresariales actuales exige un cambio de paradigma en su diseño. La arquitectura monolítica ya no es viable para sistemas que requieren escalabilidad y flexibilidad. La arquitectura orientada a servicios (SOA) promueve un diseño modular y adaptable, mejorando la eficiencia y capacidad de respuesta ante cambios. Sin embargo, su implementación presenta desafíos, como la necesidad de estándares adecuados para garantizar la interoperabilidad. Resalta la importancia de invertir en talento especializado y herramientas avanzadas para diseñar sistemas robustos y sostenibles.

# Representación de la arquitectura de software usando UML

La arquitectura de software es esencial para representar sistemas de manera integral mediante diagramas o vistas, facilitando la comunicación y toma de decisiones. Los lenguajes de Definición de Arquitecturas (ADL) presentan limitaciones como falta de herramientas y un enfoque restringido. En cambio, UML (Lenguaje de Modelado Unificado) surge como una alternativa estándar, versátil y aplicable en todas las etapas del desarrollo. UML incluye diagramas como clases, casos de uso, componentes y despliegue, y mecanismos de extensión que permiten personalizarlo, adaptándose a las necesidades específicas del sistema.

## Bibliografía

- Gil, S. V. H. (2006). UML-based Scheme for Software Architecture Representations. *Sistemas y Telemática*, 1(1), 63. <https://doi.org/10.18046/syt.v1i1.918>



## Reflexión

ha evolucionado de ser un concepto abstracto a una disciplina clave en el desarrollo moderno. Representar un sistema con múltiples vistas mejora la comunicación, fomenta el análisis integral y facilita la toma de decisiones. Aunque los ADL son especializados, su limitación destaca la necesidad de herramientas más accesibles como UML. UML promueve la colaboración multidisciplinaria gracias a su flexibilidad y capacidad de personalización. En un entorno de creciente complejidad, invertir en herramientas que mejoren claridad y sostenibilidad es esencial para desarrollar sistemas robustos.

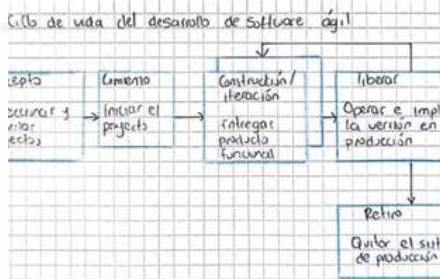
# Integración de Arquitectura en Metodologías Ágiles

explora la convergencia entre Metodologías Ágiles (MA) y Arquitectura de Software (AS), dos enfoques tradicionalmente opuestos debido a su manejo de requisitos. Las MA priorizan la flexibilidad, mientras que la AS toma decisiones tempranas que afectan el diseño. Para integrar ambos, surge la Arquitectura Ágil (AA), que utiliza los Requisitos Significativos para la Arquitectura (RSA), fundamentales para una adecuada arquitectura. Estos requisitos, que incluyen aspectos funcionales y no funcionales, son complejos de identificar y validar. La investigación evalúa metodologías y casos reales para optimizar el diseño de sistemas.

## Bibliografía

Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (2017, 4 septiembre). Integración de arquitectura de software en el ciclo de vida de las metodologías ágiles.

<https://sedici.unlp.edu.ar/handle/10915/62077>



## Reflexión

La integración de Metodologías Ágiles y Arquitectura de Software no solo es deseable, sino necesaria, dado el creciente dinamismo en los entornos de desarrollo.

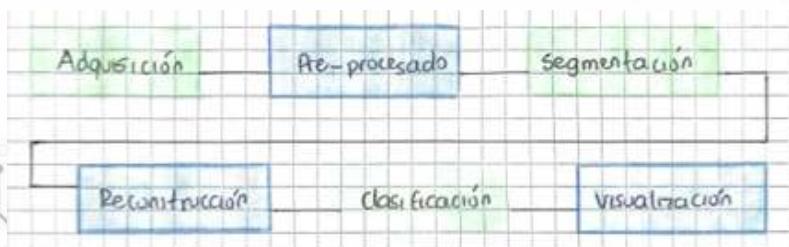
Aunque tradicionalmente percibidas como incompatibles, la adopción del enfoque de Arquitectura Ágil permite mitigar estas tensiones al centrarse en los RSA. Esto no solo mejora la calidad del diseño arquitectónico, sino que también refuerza la colaboración y la adaptabilidad que son esenciales en proyectos modernos. Sin embargo, la identificación de RSA sigue siendo un desafío crítico que requiere herramientas y estrategias avanzadas. Este enfoque marca un paso hacia un desarrollo más holístico, integrando flexibilidad y robustez en los sistemas.

# Arquitectura de Software para Vismedic

El artículo aborda la necesidad de redefinir la arquitectura del sistema Vismedic, dedicado a la visualización médica digital, que enfrenta problemas de fuerte acoplamiento, baja reutilización y dificultades para incorporar nuevas funcionalidades. Se propone mejorar la arquitectura adoptando estilos como la arquitectura basada en componentes, en capas y de tuberías y filtros. Además, se recomienda el uso de la especificación OSGi para implementar un sistema modular de plugins. La evaluación arquitectónica es esencial para identificar riesgos y fortalezas, garantizando que los requisitos del sistema se cumplan eficazmente.

## Reflexión

El caso de Vismedic destaca la importancia de una arquitectura robusta y flexible en proyectos de software, especialmente en sectores como la medicina, donde la precisión y eficiencia son claves. Un diseño débil limita la adaptabilidad y escalabilidad, obstaculizando la innovación. La transición a arquitecturas basadas en componentes y estilos probados, como la arquitectura en capas, no solo resuelve problemas actuales, sino que establece un camino hacia un desarrollo sostenible. Adoptar estándares como OSGi facilita la modularidad y el crecimiento incremental, ofreciendo una lección sobre la inversión en un diseño arquitectónico sólido.



## Bibliografía

Dolores, R. P. A., & Guillermo, S. R. L. (s. f.). Arquitectura de software para el sistema de visualización médica Vismedic. <https://n9.cl/mkviyc>

# Diversidad de Estilos de Aprendizaje en Arquitectura del Software

El artículo aborda los desafíos de enseñar arquitectura de software (AS), una disciplina que requiere habilidades de abstracción avanzada y aborda complejidades como rendimiento y escalabilidad. Además de las competencias técnicas, la docencia de AS exige habilidades no técnicas como toma de decisiones y comunicación. Se destaca la diversidad de estilos de aprendizaje, según el modelo de Kolb, que clasifica a los estudiantes en estilos como asimilador, convergente, acomodador y divergente. Sin embargo, las metodologías tradicionales no consideran todos estos estilos de manera efectiva.



## Reflexión

El enfoque tradicional de enseñanza de la arquitectura de software enfrenta desafíos debido a la complejidad y los altos niveles de abstracción de los conceptos. Aunque las metodologías actuales intentan replicar situaciones reales de trabajo, no siempre son efectivas para todos los estilos de aprendizaje, limitando la comprensión de ciertos estudiantes. La propuesta de utilizar juegos de rol en la docencia de AS es innovadora, ya que involucra activamente a los estudiantes y les permite experimentar dinámicas de AS de manera práctica, mejorando su capacidad de tomar decisiones informadas y comprender sus implicaciones.

## Bibliografía

Castro, L. M. (2023). Atención a la diversidad de estilos de aprendizaje: experiencia en la docencia de arquitectura del software. <https://n9.cl/2al2dz>

# Una Especificación Precisa para Patrones GoF

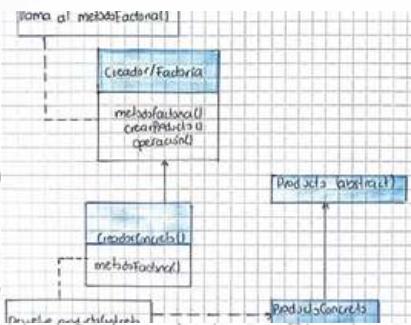
El artículo presenta el desarrollo de una especificación formal para definir la semántica de los patrones de diseño, enfocándose en los patrones de Gama (GoF). El objetivo es mejorar la seguridad y calidad del software, permitiendo verificar el uso adecuado de los patrones y evitando contradicciones con su semántica definida. El modelo incluye tres tipos de acciones: invocación, instancia e invocación a clases superiores en la jerarquía de herencia, representadas en RSL (RAISE Specification Language). Además, se incorporan extensiones para abordar variaciones y aplicar la formalización a patrones de análisis de dominio, como los de Fowler.

## Reflexión

La formalización de patrones de diseño propuesta en este artículo es clave para mejorar la calidad y seguridad del software. Especificar con precisión la semántica de los patrones facilita su correcta aplicación y permite desarrollar herramientas para automatizar la verificación de diseños, asegurando las mejores prácticas. La extensión del modelo para incluir variaciones y su aplicación en análisis de dominios específicos amplía su utilidad. Una herramienta que verifique automáticamente los diseños con base en la semántica formalizada representaría un avance significativo, garantizando que los patrones se apliquen correctamente y mejorando la calidad del producto final.

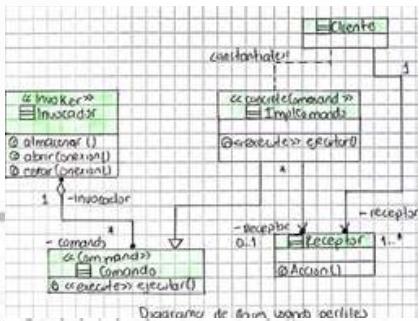
## Bibliografía

- Cechich, A., & Moore, R. (2001). Una especificación precisa para patrones GoF. <https://sedici.unlp.edu.ar/handle/10915/21723>



# Perfiles UML para Patrones de Diseño de Comportamiento

El artículo propone el uso de perfiles UML y restricciones OCL para definir patrones de comportamiento en ingeniería de software, permitiendo especificar y validar patrones en modelos estáticos y dinámicos. Se emplean diagramas de clase y secuencia de UML como base para esta especificación, con un perfil UML para cada patrón y restricciones en OCL para validar la estructura y el comportamiento. Además, se introduce una arquitectura de tres niveles para facilitar la reutilización y la consistencia en el modelado. La herramienta utilizada es Rational Software Developer (RSA) de IBM.



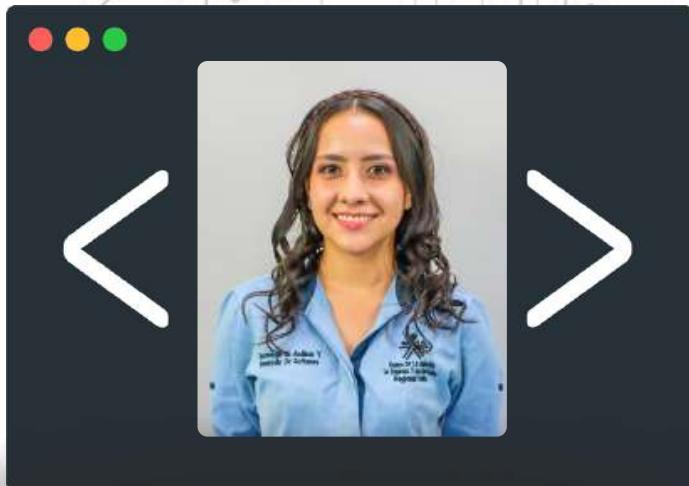
## Reflexión

aborda un reto crucial en ingeniería de software: validar patrones de comportamiento en modelos de sistemas. Utilizando perfiles UML y restricciones OCL, se asegura que los modelos sean correctos tanto estructural como dinámicamente. Esta integración entre estructura y comportamiento permite una verificación más exhaustiva. El uso de UML para formalizar patrones y OCL para restricciones es una metodología poderosa para garantizar la calidad del software desde las primeras fases. La validación automatizada con herramientas como RSA mejora la consistencia y evita errores costosos en proyectos grandes.

## Bibliografía

Cortez, A., Riesco, D. E., & Garis, A. G. (2012b). Perfiles UML para la definición de patrones de diseño de comportamiento. <https://n9.cl/ye1m6>

# Maria José Murcia Martínez

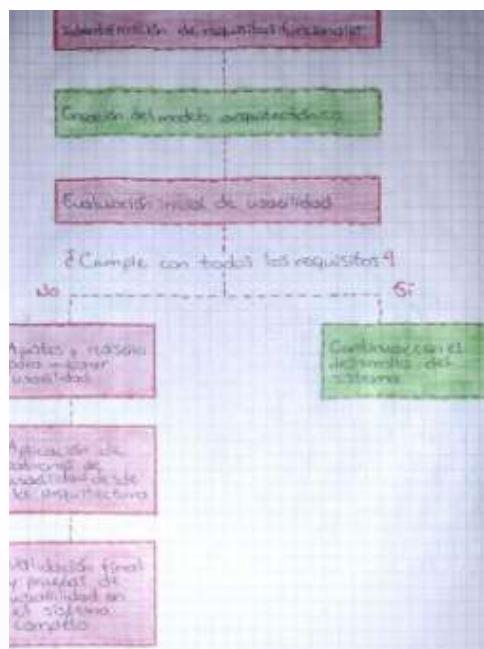


Soy María José Murcia Martínez, nací el 24 de agosto de 2005 en Campoalegre, Huila. Desde temprana edad, me interesé por las tecnologías y la creatividad, lo que me llevó a realizar un técnico en Diseño e Integración de Multimedia en la Institución Educativa Eugenio Ferro Falla. Al terminar el grado 11, egresé como bachiller técnico con esta formación complementaria. Posteriormente, ingresé al SENA, regional Huila, en el Centro de la Industria, la Empresa y los Servicios en Neiva, donde actualmente estudio el tecnólogo en Análisis y Desarrollo de Software. En este momento, me encuentro realizando mis prácticas, aplicando y fortaleciendo los conocimientos adquiridos durante la formación.

En mi tiempo libre, he aprovechado la plataforma Sofía Plus del SENA para realizar diferentes cursos virtuales que me han permitido adquirir nuevas habilidades y expandir mis conocimientos.

# Patrones de Usabilidad en la Arquitectura de Software

El proyecto STATUS propone integrar la usabilidad en el software desde la fase de diseño, utilizando patrones como "deshacer," "cancelar" y "múltiples idiomas" para mejorar la interacción del usuario. A diferencia de los enfoques tradicionales, que evalúan la usabilidad solo al final, STATUS utiliza un método inductivo que permite ajustar el diseño de forma continua durante el proceso. Esta integración estructural garantiza que el sistema sea intuitivo y accesible desde el principio, sin grandes pérdidas de tiempo o recursos, mejorando la experiencia del usuario en cada fase del desarrollo.



## Reflexión

ofrece ventajas como un sistema menos acoplado, facilitando el mantenimiento y la actualización de servicios independientes sin afectar al sistema completo. También permite la reutilización de servicios, lo que mejora la flexibilidad y escalabilidad. Sin embargo, presenta desafíos técnicos, como la latencia en la comunicación entre servicios, que afecta el rendimiento, y la complejidad de asegurar la seguridad y la integridad de los datos. Gestionar las dependencias entre servicios es crucial, y una posible solución es un servicio central que coordine las acciones del sistema.

## 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://n9.cl/6s49x>

# Arquitectura de software, Esquemas y servicios

La arquitectura de software define el diseño básico de un sistema, organizando sus componentes y cómo se comunican. En sistemas grandes y complejos, se busca minimizar el "acoplamiento" entre componentes para facilitar el mantenimiento y la evolución. El uso de esquemas, que agrupan funcionalidades comunes, promueve la modularidad y reduce costos al evitar duplicación de código. En sistemas antiguos, adaptar o actualizar el código es un reto, y aquí la arquitectura orientada a servicios (SOA) resulta útil. SOA organiza el sistema en servicios independientes, facilitando la integración y ofreciendo flexibilidad y escalabilidad.

## Reflexión

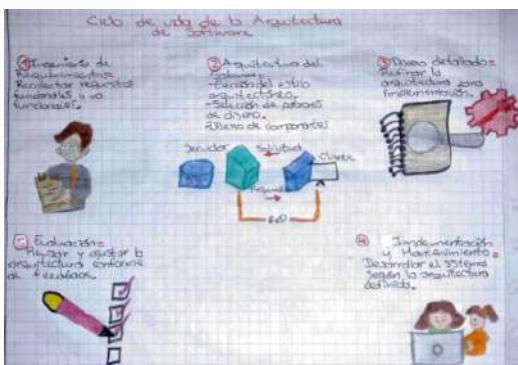
arquitectura orientada a servicios (SOA) ofrece ventajas como un sistema menos acoplado, facilitando su mantenimiento y permitiendo actualizaciones de servicios sin afectar al sistema completo. Además, promueve la reutilización de servicios, mejorando la flexibilidad y escalabilidad. Sin embargo, presenta desafíos como la latencia en la comunicación entre servicios, lo que puede afectar el rendimiento, y la dificultad de garantizar la seguridad e integridad de los datos. Gestionar dependencias entre servicios es crucial, y una solución es implementar un servicio central que coordine las acciones del sistema. A pesar de estos retos, SOA sigue siendo valiosa por su capacidad de adaptación y eficiencia.



## Bibliografía

# Introducción a la arquitectura de software

La arquitectura de software se consolidó como una disciplina independiente en los años 90, cuando se comprendió su importancia para estructurar sistemas a un nivel superior al diseño detallado. Aunque en los 70 ya se descomponían programas en partes más manejables, fue en los 90 cuando se organizó mejor gracias a contribuciones de expertos como Mary Shaw y David Garlan. A lo largo del ciclo de vida de un sistema, la arquitectura debe adaptarse a cambios en los requisitos, tanto funcionales como no funcionales, que influyen en su capacidad de respuesta y calidad, asegurando que el sistema siga siendo relevante para los usuarios y el negocio.



## Reflexión

Es clave no solo por su estructura técnica, sino también por alinear los requisitos sociales y de negocio, lo que la convierte en una disciplina compleja. El éxito de un sistema depende de cómo las decisiones de diseño impactan tanto en los aspectos técnicos como en los objetivos del negocio. Separar arquitectura de diseño permite tener una visión global antes de enfocarse en los detalles, lo que facilita un desarrollo más claro y reduce errores. Esto asegura un sistema funcional, adaptable y capaz de evolucionar a lo largo del tiempo, ajustándose a cambios en el negocio o la tecnología.

## Bibliografía

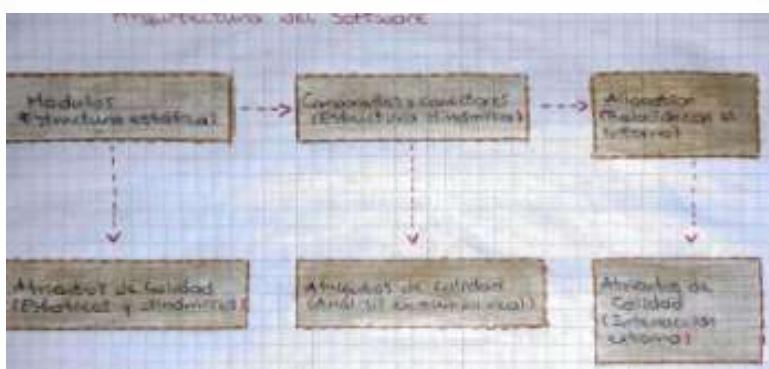
Romero, P. Á. (2006). Arquitectura de software, esquemas y servicios. Ingeniería Industrial, 27(1).<https://n9.cl/vzvqm>

# Integración de arquitectura en metodologías

La arquitectura de software es fundamental en el desarrollo de sistemas modernos, ya que define cómo estará estructurado el sistema. Facilita la documentación de las decisiones de diseño, permitiendo que desarrolladores e interesados comprendan la estructura y el mantenimiento a largo plazo. Diseñar una buena arquitectura implica equilibrar atributos de calidad como mantenibilidad, portabilidad, seguridad y rendimiento, analizados desde diversas perspectivas (estática, dinámica y externa). Las vistas arquitectónicas permiten un análisis detallado, y documentar la arquitectura asegura que el sistema cumpla con las expectativas del proyecto.

## Reflexión

Una buena arquitectura de software es crucial porque permite que el sistema se adapte a cambios sin comprometer su calidad. No se trata solo de crear una solución funcional, sino de pensar en el mantenimiento, la escalabilidad y la seguridad a largo plazo. El mantenimiento asegura que el sistema se actualice sin dificultades, la escalabilidad permite que crezca según las demandas, y la seguridad protege contra ataques. La arquitectura también facilita la comunicación entre los equipos de desarrollo y las partes involucradas, asegurando que el sistema cumpla con los objetivos del proyecto y de los usuarios.



## Bibliografía

- Bastarrica, M. C. (2005). Atributos de Calidad y Arquitectura del Software.  
<https://n9.cl/9gjrb>

# Atributos de calidad y Arquitectura de software

Este artículo explora la integración de la arquitectura de software (AS) y las metodologías ágiles (MA) en el desarrollo de sistemas de información, enfocándose en los requisitos de importancia arquitectónica (RSA) como punto de conexión clave. Mientras las MA favorecen la adaptabilidad y colaboración, la AS establece una estructura rígida difícil de modificar. El enfoque de "Arquitectura Ágil" (AA) propone usar los RSA para lograr una arquitectura flexible que mantenga la robustez y se ajuste a los principios ágiles, permitiendo un equilibrio entre estructura y flexibilidad.

Elemento	Descripción
Metodologías Ágiles (MA)	Enfoque basado en adaptabilidad, colaboración y fluidez ante cambios en requisitos.
Arquitectura de Software (AS)	Conjunto de decisiones trascendentales que afectan el diseño del sistema y condicionan futuras decisiones de desarrollo.
Sofío de Integración	Los MA suelen hacer diseños más leves y extensibles, lo cual puede generar conflictos con las AS, que dependen de diseños trascendentales.
Arquitectura Ágil (AA)	Propuesta para integrar MA y AS con un enfoque en los Requisitos Significantes para la Arquitectura (RSA).
Requisitos Significantes (RSA)	Requisitos que tienen un impacto profundo en la arquitectura del sistema y afectan su viabilidad y efectividad.
Infinito de Integración	Mejora en la adecuación de la arquitectura a las necesidades del sistema y en la adaptabilidad ante cambios de requisitos.

## 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://n9.cl/qjynz7>

## Reflexión

La integración de la arquitectura de software (AS) y las metodologías ágiles (MA) presenta un desafío, ya que ambos enfoques siguen principios opuestos. Mientras la AS busca una estructura estable desde el inicio, las MA promueven la adaptabilidad ante cambios. Sin embargo, los requisitos de importancia arquitectónica (RSA) permiten conciliar ambos métodos, manteniendo elementos clave de la arquitectura estables mientras se adaptan a los cambios. Integrar los RSA ayuda a crear una arquitectura flexible que evoluciona sin comprometer la estabilidad, logrando un balance entre robustez y adaptabilidad.

# Patrones GOF en el Desarrollo Web

El estudio de los patrones de diseño GOF (Gang of Four) en el desarrollo de aplicaciones web en Colombia revela que los patrones creacionales, como Singleton, Factory Method y Builder, son los más utilizados. En cuanto a los patrones estructurales, Decorator y Facade son comunes, mientras que los patrones de comportamiento más aplicados son Iterator, Template Method y Strategy. Aunque su uso no es masivo, se reconoce su importancia para mantener la calidad y facilitar el mantenimiento del software. La principal barrera para su adopción es la falta de conocimiento y experiencia entre los desarrolladores.

## Reflexión

Los patrones de diseño GOF son útiles en la creación de software, especialmente en aplicaciones web, donde la calidad y la claridad del código son esenciales. Sin embargo, su aplicación depende del conocimiento y experiencia de los desarrolladores. No todos los problemas requieren su uso, por lo que es crucial que los desarrolladores evalúen cuándo aplicarlos. Además, deben entender que los patrones no solo resuelven problemas, sino que mejoran la eficiencia y la expansibilidad del software. Para su adopción efectiva, las empresas deben ofrecer capacitación y prácticas en contextos reales.

Proceso de Identificación de Patrones de Diseño GOF	
Paso	Descripción
1	Identificación del entorno de desarrollo del proyecto: -CD: Tamaño del equipo de desarrollo -E: Uso del OML -C: Nivel de complejidad del sistema -C: Modularidad del sistema -C: Modelos de validación
2	Selcción de muestra de proyectos
3	Examen del código fuente de los proyectos
4	Identificación de patrones GOF
5	Resultados: Patrones más usados: Singleton, Factory Method, Decorator
6	
7	

## 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://n9.cl/bxew6>

# Análisis comparativo de Patrones de Diseño de Software

Este artículo analiza y compara cinco patrones de diseño clave en el desarrollo de software: Template Method, MVC, MVP, Front Controller y MVVM. Cada patrón ofrece soluciones a problemas comunes, facilitando la organización del código y la reutilización. Template Method permite definir un algoritmo flexible, MVC organiza en modelo, vista y controlador, MVP reduce el acoplamiento entre vista y modelo, Front Controller centraliza la gestión de solicitudes y MVVM optimiza la modularidad. La conclusión es que no hay un patrón superior, sino que se deben aplicar según las necesidades del proyecto.

## Reflexión

El conocimiento de patrones de diseño es esencial para estudiantes y desarrolladores, ya que permite organizar mejor el código y facilitar su mantenimiento. Comprender patrones como MVC y MVP ayuda a reducir el acoplamiento entre componentes, facilitando actualizaciones sin introducir errores. Además, no todos los patrones son adecuados para todas las aplicaciones; por ejemplo, el Template Method es útil cuando el algoritmo base es constante, y MVVM es ideal para interfaces de usuario complejas. Esta comprensión estratégica mejora la calidad del software final y permite elegir las mejores herramientas para cada proyecto.

Patrón	Descripción	Vantajes	Desventajas	Uso ideal
Template	Define esqueleto de un algoritmo permitiendo combinarlo.	Reutilización ésta evitando duplicaciones.	Mantenimiento complejo.	Algoritmos con variaciones menores.
MVC	Separación de datos, lógica y vista entre tres componentes.	Separación de lógica, visual y controlador.	Ciclo de aprendizaje alto.	Aplicaciones web y de escritorio.
MVP	Variante de MVC que desacopla vista y modelo de lógica.	Fácil de aprender, separación vista y modelo.	Compleja para aplicaciones simples.	Aplicaciones lógica compleja.
Front Ctrl	Controlador encarga toda la lógica.	Seguridad y control centralizado.	Escalabilidad limitada.	Aplicaciones con muchas solicitudes.
MVVM	Unifica lógica y vista para mejorar mantenimiento y productividad.	Módularidad, fácil mantenimiento.	Compleja de aprender y dominar.	Interfaz intuitiva y adaptativa.

## 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://n9.cl/d9ms3>

# Revisión sobre generadores de código y patrones de arquitectura

Este trabajo revisa los generadores de código fuente (GCF) y los patrones de arquitectura en el desarrollo de software, especialmente en aplicaciones web. Los GCF automatizan tareas repetitivas, como crear interfaces y conectar bases de datos, lo que ahorra tiempo, reduce costos y minimiza errores. Se destacan patrones como MVC y MVVM, que organizan el código de manera estandarizada, facilitando el mantenimiento y la expansión del software. Además, se analizan herramientas y lenguajes populares, así como las prácticas actuales en el uso de GCF y patrones de arquitectura.

Importancia	El diseño organiza y facilita el mantenimiento del software a largo plazo mediante cambios y asegurando su adaptabilidad.
Principios Clave	
Tecnología para el cambio	Proporciona que el software se adapte a cambios rápidos sin cambiar su estructura.
Diseño calculado	Uso de reglas claras para organizar las complejidades, facilitando su comprensión y mantenimiento.
Niveles del diseño	
1- Estilo Arquitectónico	Estructura general del sistema (como el modelo cliente/servidor).
2- Patrones de Diseño	Soluciones comunes para problemas recurrentes.
3- Definición de Componentes	Especificación detallada de cada parte del sistema.
	El diseño debe de ser organizado.

## Reflexión

El diseño de software no es solo una fase inicial, sino una inversión que facilita el proceso a largo plazo. El concepto de "diseño para el cambio" es clave, ya que un software bien diseñado no necesita rehacerse cada vez que surjan nuevos requisitos, ahorrando tiempo y dinero. Seguir reglas claras y usar patrones de diseño comunes mejora la organización, evita errores y facilita la comunicación en el equipo. Un buen diseño hace que el software sea flexible, fácil de mantener y evolucione con el tiempo, asegurando la inversión a largo plazo.

## Bibliografía

CCristiá, M. (2021). Una Teoría para el Diseño de Software.<https://n9.cl/dr0m7>

# Una teoría para el diseño de software

El artículo aborda la teoría del diseño de software, enfatizando la importancia de tener una estructura clara en cada proyecto. El diseño de software no es solo una fase inicial, sino un proceso continuo durante todo el ciclo de vida del sistema. La teoría sugiere dividir el software en componentes pequeños para facilitar su comprensión y adaptación, reduciendo costos de mantenimiento. Destaca principios como "diseño para el cambio" y "diseño calculado", que anticipan modificaciones sin alterar la estructura. Además, divide el diseño en tres niveles: arquitectónico, patrones de diseño y definición detallada de componentes.

## Bibliografía

Casas, M. R. H. (2020). Revisión sistemática sobre generadores de código fuente y patrones de arquitectura (Master's thesis, Pontificia Universidad Católica del Perú). <https://n9.cl/69c366>

Concepto	Descripción
Generadores de Código Fuente (GCF)	Herramientas que automatizan la creación repetitiva, como formularios o sets de datos.
Ejemplo	JHipster crea aplicaciones web con estructuras sólidas.
Beneficio	Ahorra tiempo y reduce errores al automatizar tareas comunes.
Patrón de arquitectura	Descripción
MVC (Modelo-Vista-Controlador)	Organiza el código en tres partes: Modelo (datos), Vista (interfaz) y Controlador (lógica).
Beneficio	Mejora la organización y facilita la expansión del sistema.
Tarea	Beneficio
Uso de GCF	Automatizar el código repetitivo, ahorrando tiempo y errores.
Uso de MVC	Estructura clara del código para facilitar su mantenimiento y evolución.

## Reflexión

Esta revisión destaca la importancia de usar herramientas como los GCF y los patrones de arquitectura en el desarrollo de software, especialmente para aplicaciones rápidas y complejas. Estos recursos permiten a los programadores trabajar de manera más eficiente, reduciendo el esfuerzo manual y mejorando el resultado final. Elegir adecuadamente estas herramientas es clave para cumplir con los objetivos del cliente y asegurar la facilidad de mantenimiento a largo plazo. Además, el conocimiento continuo de estas herramientas permite que el software sea adaptable y relevante incluso con cambios en las necesidades.

# Perfiles UML para definición de Patrones de Diseño

Los perfiles UML expanden las capacidades del lenguaje unificado de modelado (UML) al permitir la definición y visualización de patrones de diseño de forma más precisa. A través de estereotipos, etiquetas y restricciones, los perfiles personalizan UML para representar patrones de diseño específicos, como el patrón estructural "composite". Esta técnica facilita la creación de bibliotecas reutilizables y promueve la estandarización, optimizando el trabajo colaborativo y el diseño. Los perfiles UML mejoran la eficiencia y permiten una comunicación más clara en proyectos de programación orientada a objetos.

## Reflexión

La propuesta de usar perfiles UML para definir patrones de diseño mejora la flexibilidad y expresividad de UML, adaptándolo a problemas específicos en proyectos. Este enfoque optimiza el flujo de trabajo al eliminar la necesidad de herramientas externas, creando un vocabulario visual común y adaptado para cada tipo de proyecto. Facilita la colaboración entre desarrolladores, simplificando la comunicación en proyectos complejos. Además, promueve un desarrollo más dinámico y eficiente, mejorando la calidad del diseño y su alineación con las necesidades del cliente.

Objetivo de los Perfiles UML	¿Qué son los Perfiles UML?	Beneficios de los Perfiles UML	Patrón Composite
• Ayudar a los desarrolladores a manejar patrones complejos y abstractos de forma más sencilla y eficiente.	• Extensiones del UML mediante estereotipos, etiquetas y restricciones.	• <b>Flexibilidad:</b> Adaptación de lenguaje a cada proyecto.	• Elementos = Componentes generales del patrón.

## 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://n9.cl/csaft>

# Lenguajes de Patrones de Arquitectura

El texto explora los lenguajes de patrones de arquitectura de software, herramientas esenciales para resolver problemas comunes en el desarrollo de sistemas. Estos lenguajes permiten crear soluciones reutilizables y adaptables, mejorando la eficiencia del diseño. Destaca la composición de patrones, las secuencias de soluciones reutilizables y las colecciones de patrones relacionados. Se presentan ejemplos prácticos, como su uso en seguridad informática y e-business, además de enfrentar retos como arquitecturas orientadas a servicios y en la nube. Concluye que estos lenguajes son fundamentales para mejorar la calidad y eficiencia del software.



## Bibliografía

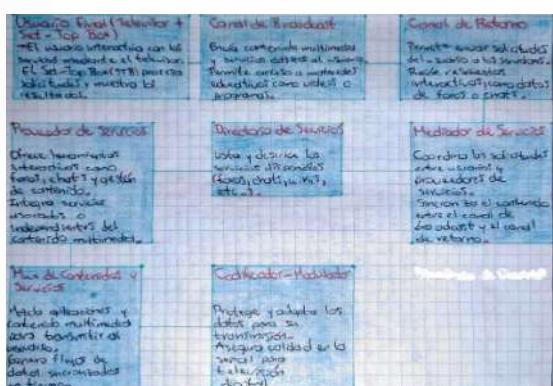
- Jimenez-Torres, V. H., Tello-Borja, W., & Rios-Patiño, J. I. (2014). Lenguajes de patrones de arquitectura de software: una aproximación al estado del arte. *Scientia et technica*, 19(4), 371-376. <https://n9.cl/ahapn>

## Reflexión

Los lenguajes de patrones en arquitectura de software son esenciales para estudiantes, ya que mejoran la eficiencia, calidad y sostenibilidad del software. Aprenderlos enseña a abordar problemas complejos de forma sistemática, desarrollando enfoques estratégicos para diseñar sistemas sólidos y adaptables. Además, fomenta el trabajo colaborativo y la documentación de buenas prácticas. Patrones en seguridad informática y e-business destacan su relevancia. Aplicarlos mejora habilidades técnicas y prepara para enfrentar desafíos con creatividad y soluciones efectivas centradas en el usuario.

# Arquitectura de Software para Comunidades Académicas

El artículo propone una arquitectura de software para comunidades académicas virtuales (CAV) utilizando televisión digital interactiva (TDi). El sistema integra herramientas de Web 2.0 como foros y chats con tecnología REST-JSON, permitiendo acceso a personas sin Internet. Incluye un directorio de servicios, un mediador de conexiones, un Set-Top Box (STB) y una plataforma multimedia. Los servicios piloto implementados demostraron su efectividad. Esta arquitectura flexible ofrece soluciones para educación y otros sectores, reduciendo la brecha digital.



## Reflexión

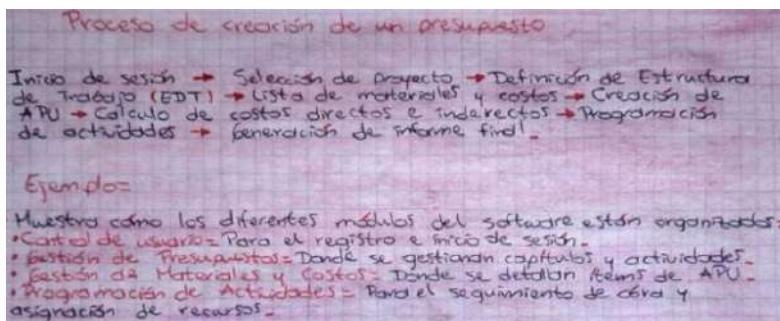
La arquitectura presentada utiliza televisión digital interactiva (TDi) para resolver problemas educativos en lugares con acceso limitado a Internet. Integrando herramientas como foros y chats mediante REST-JSON, fomenta la interactividad, el aprendizaje y la colaboración. Sin embargo, requiere inversión en infraestructura, capacitación y contenido educativo atractivo. Este modelo no solo es útil en educación, sino también en áreas como salud y gobierno, contribuyendo a reducir la brecha digital y democratizar el aprendizaje mediante soluciones tecnológicas inclusivas e innovadoras.

## 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://n9.cl/lbj1l>

# Arquitectura de Software para Costos y Programación

Este proyecto presenta una herramienta educativa para estudiantes de Ingeniería Civil, enfocada en costos, presupuestos y programación de obras. Usando un entorno simulado, permite crear análisis de precios unitarios, gestionar salarios y programar actividades mediante métodos como la ruta crítica. Desarrollada con arquitectura cliente-servidor y patrón MVC, la plataforma es accesible sin licencias costosas. Los estudiantes aprenden a gestionar proyectos de construcción, optimizando costos y tiempos, preparándolos para su futura carrera profesional.



## Reflexión

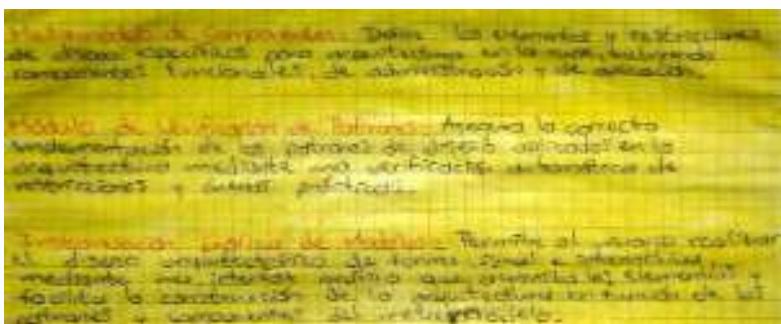
Esta herramienta educativa representa un avance en la enseñanza de ingeniería civil, proporcionando simulaciones de proyectos reales sin los costos y riesgos de los proyectos auténticos. Permite a los estudiantes aplicar conocimientos, optimizar costos y gestionar recursos en proyectos de construcción. Además, fomenta la toma de decisiones informadas, la comprensión de la interconexión de las partes de un proyecto y mejora la colaboración entre disciplinas. En resumen, mejora la formación académica y prepara a los estudiantes para su desarrollo profesional en ingeniería civil.

## 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://n9.cl/c7q9l>

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

Este trabajo propone una herramienta para modelar y verificar arquitecturas de software en la nube, asistiendo a los arquitectos en la implementación precisa de patrones de diseño. Basada en un metamodelo con elementos y restricciones específicas para la computación en la nube, la herramienta incluye un módulo de verificación automática que garantiza que los diseños cumplan con los requisitos. Esto asegura que las arquitecturas sean robustas, eficientes y alineadas con las mejores prácticas de la industria, optimizando recursos y reduciendo riesgos operativos.



## **Reflexión**

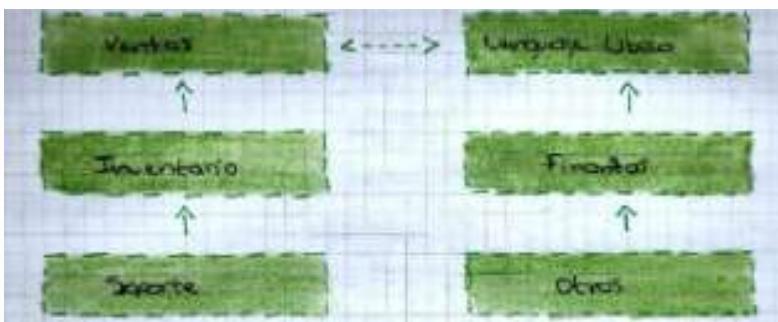
La verificación automática en el diseño arquitectónico de software es un avance clave para la computación en la nube, ya que facilita la validación continua de los diseños. En entornos donde la estructura de las aplicaciones y el consumo de recursos deben adaptarse constantemente, contar con una herramienta que verifique los patrones de diseño es esencial. Esto reduce riesgos, optimiza tiempos de desarrollo, mejora la calidad del software y permite cumplir plazos, al tiempo que facilita la innovación y el desarrollo sin preocupaciones sobre fallos arquitectónicos.

## **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://n9.cl/vzj9b>

# Arquitectura de Software Guiada por el Dominio

El Diseño Dirigido por el Dominio (DDD) coloca el conocimiento del negocio en el centro del desarrollo de software, creando un modelo de dominio que describe procesos y reglas empresariales. Utilizando un "lenguaje ubicuo", asegura que todos compartan la misma comprensión del sistema. Organiza el software en "contextos delimitados", permitiendo evolución independiente de las partes sin afectar otras áreas. DDD facilita la flexibilidad, escalabilidad y mantenimiento del software, adaptándose fácilmente a nuevos requerimientos del negocio.



## Reflexión

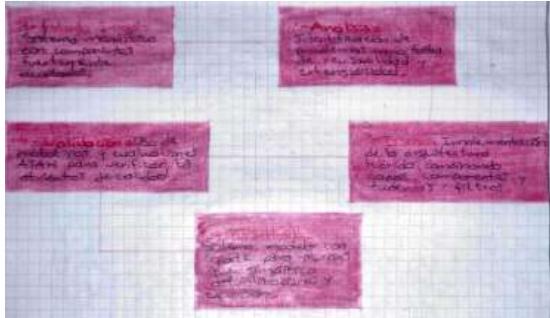
reduce la brecha entre el negocio y las soluciones tecnológicas, enfocándose en un entendimiento profundo del negocio. Fomenta la colaboración continua entre desarrolladores y expertos mediante un "lenguaje ubicuo", asegurando que todos estén alineados. La organización del software en "contextos delimitados" facilita la independencia y flexibilidad, permitiendo que el sistema evolucione sin afectar otras partes. DDD mejora la escalabilidad, mantenibilidad y adaptación a nuevos requisitos, garantizando soluciones sostenibles.

## 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://n9.cl/dch02>

# Arquitectura de software para Vismedic

El artículo presenta mejoras en la arquitectura del sistema de visualización médica Vismedic, utilizando un enfoque por capas, basado en componentes y con un modelo de tuberías y filtros para procesar datos eficientemente. La incorporación de un sistema de plugins permite añadir funciones sin modificar el núcleo. Esta arquitectura mejora la escalabilidad, facilita el mantenimiento y reduce costos a largo plazo. Las pruebas con prototipos y el método ATAM confirman su efectividad, aumentando la extensibilidad y reutilización del sistema.



## Reflexión

El artículo destaca que diseñar software es más que cumplir requisitos; es planificar para el futuro. Una arquitectura bien diseñada hace el sistema flexible y adaptable a cambios, facilitando la incorporación de nuevas funciones. La modularidad mejora el mantenimiento y reduce costos a largo plazo. Invertir tiempo en el diseño y validar lo con métodos como ATAM asegura que el sistema sea confiable y evolutivo. Un buen diseño garantiza que el software siga siendo útil y relevante con el tiempo.

## 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. <https://n9.cl/y2kte>

# Desarrollo de software con patrones de diseño OOP

El documento describe el desarrollo del sistema "Intranet Industrial" para la Facultad de Ingeniería Industrial de la UNMSM, utilizando patrones de diseño orientados a objetos. Se implementaron tres capas: presentación, negocio y datos, con patrones como Informador y Sensor para optimizar consultas y manejo de errores. Usando la metodología RUP, se lograron ahorrar costos y tiempo de desarrollo. El sistema incluye módulos como Usuarios, Noticias y Biblioteca, mejorando la comunicación y organización interna.

Capacidad	Arquitectura Propuesta	Arquitectura Tradicional
Evolucionabilidad	Baja. Cambiar tendremos que modificar muchos módulos.	Alta - Soporte para plug-in's independientemente de la arquitectura.
Mantenibilidad	Dificultad - Alta. Aprendizaje -	Simplificado - Componentes claros y claros.
Reutilización	Limitada o lenta.	Compartible con multiples plataformas.
Escalabilidad	Fácil a otras modificaciones en el código.	Relacionada a componentes existentes en otras plataformas.
Resiliencia	Incremento el rendimiento al redimensionar la memoria.	Precisa mantenimiento y resiliencia a cambios significativos en los sistemas.

## Reflexión

El proyecto resalta la importancia de los patrones de diseño en sistemas complejos como la Intranet Industrial. Estos patrones organizan el código, facilitando su modificación y reutilización, lo que ahorra tiempo y esfuerzo. La separación en capas mejora la comprensión y mantenimiento del sistema, mientras que patrones como Informador y Sensor optimizan la gestión de bases de datos y el manejo de errores. La metodología RUP reduce riesgos y asegura un sistema eficiente, escalable y preparado para el futuro, demostrando buenas prácticas en el desarrollo de software.

## Bibliografía

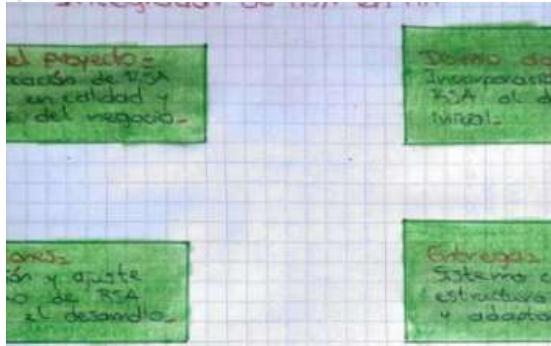
Lazo, L., & Paul, J. (2004). Desarrollo de sistemas de software con patrones de diseño orientado a objetos.<https://n9.cl/ch7fg>

# Arquitectura de software en desarrollo

El artículo explora cómo integrar la Arquitectura de Software (AS) con las Metodologías Ágiles (MA), que parecen incompatibles. La AS se enfoca en decisiones estables al inicio del proyecto, mientras que las MA priorizan la adaptabilidad. Para superar esto, se propone el modelo "Requisitos Significativos para la Arquitectura" (RSA), que identifica elementos clave que impactan la estructura del sistema desde el principio. Así, se combina la flexibilidad de las MA con la estabilidad necesaria en la AS.

## 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. <https://n9.cl/qbptus>

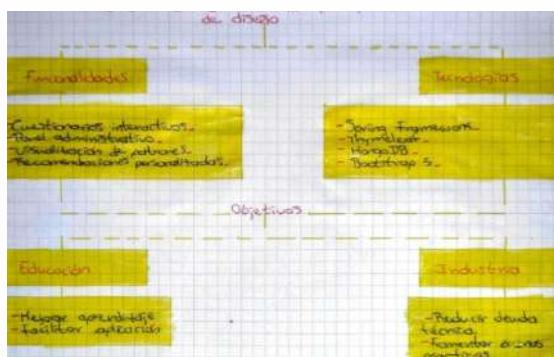


## Reflexión

La integración de la Arquitectura de Software (AS) y las Metodologías Ágiles (MA) mejora el desarrollo de software al combinar sus ventajas. Identificar los Requisitos Significativos para la Arquitectura (RSA) al inicio asegura una base sólida, cubriendo aspectos clave como seguridad y rendimiento. A la vez, se respeta la flexibilidad de las MA, permitiendo que el sistema evolucione y se adapte. Este enfoque crea software robusto, adaptable y alineado con los objetivos del negocio, respondiendo a las demandas del entorno.

# Desarrollo de una herramienta para el aprendizaje de patrones

El artículo presenta una herramienta web para facilitar el aprendizaje de patrones de diseño orientados a objetos, utilizando tecnologías como Spring, Thymeleaf y MongoDB. La plataforma interactiva permite a los usuarios experimentar con patrones mediante ejemplos prácticos y teóricos, promoviendo una comprensión profunda. Con cuestionarios interactivos y interfaces intuitivas, la herramienta es extensible y útil tanto para estudiantes como profesionales, mejorando la calidad del código y fomentando buenas prácticas de desarrollo.



## Reflexión

La herramienta propuesta aborda la necesidad de soluciones educativas prácticas en ingeniería informática, enfocándose en el aprendizaje de patrones de diseño para mejorar la calidad del código. A medida que el software se hace más complejo, la comprensión y aplicación de estos patrones es esencial. La herramienta debe ser intuitiva para usuarios novatos, flexible para futuras actualizaciones y fomentar la colaboración entre desarrolladores. Su objetivo es reducir la deuda técnica y promover un desarrollo más ágil y mantenible.

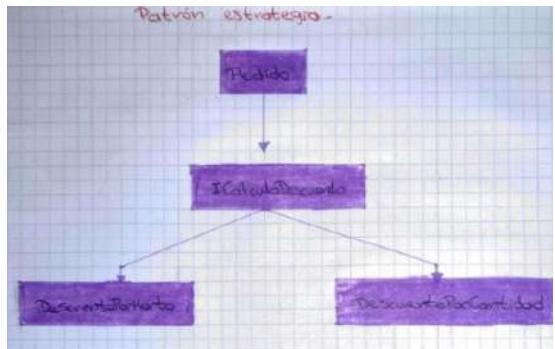
## Bibliografía

- Ferrandis Homsi, A. (2021). Desarrollo de una herramienta para el aprendizaje de patrones de diseño software (Doctoral dissertation, Universitat Politècnica de València).<https://n9.cl/i2em3j>

# Herramienta para reusó de código JavaScript orientado a patrones

El artículo presenta ReusMe, una herramienta para reutilizar código JavaScript en el diseño de interfaces web mediante patrones de interacción. Esta plataforma permite a los usuarios seleccionar y personalizar patrones, generando código listo para implementar. ReusMe usa metodologías como UML y PUD para garantizar un diseño estructurado y adaptable. Facilita la reutilización de soluciones probadas, ahorrando tiempo y mejorando la calidad, y promueve la usabilidad, asegurando interfaces intuitivas y eficientes para los usuarios.

## Reflexión



## 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://n9.cl/dyk2n>

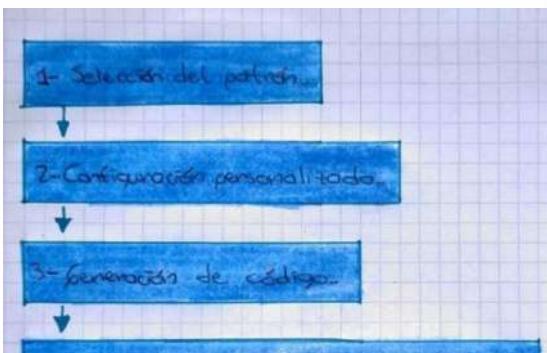
ReusMe impulsa la innovación en el diseño de interfaces al integrar usabilidad y reutilización de código. Esta herramienta permite a los desarrolladores personalizar patrones de interacción, mejorando la eficiencia y creatividad en sus proyectos. Aunque plantea desafíos como la actualización de patrones y la colaboración para ampliar la biblioteca, ReusMe democratiza el acceso a soluciones avanzadas, empoderando a diseñadores de diferentes niveles. Promueve un balance entre automatización y personalización en el desarrollo web.

# Aplicación de patrones de diseño para garantizar alta flexibilidad

El artículo resalta la importancia de los patrones de diseño para crear aplicaciones flexibles y adaptables. Utilizando un ejemplo de un sistema de pedidos en un restaurante, aplica los patrones estrategia, compuesto y fábrica para gestionar reglas de negocio variables, como políticas de descuento. Estos patrones promueven bajo acoplamiento, alta cohesión y el principio abierto-cerrado, lo que facilita la extensibilidad sin modificar el código base. Se recomienda el uso de frameworks como Spring para optimizar la organización y desarrollo del código.

## Reflexión

Son esenciales para manejar la complejidad y el dinamismo en aplicaciones empresariales. En el caso del sistema de pedidos, principios como bajo acoplamiento y alta cohesión aseguran estabilidad y escalabilidad frente a cambios. El uso de una interfaz para estrategias de descuento permite incorporar nuevas reglas sin modificar el código existente, reduciendo costos de mantenimiento y tiempo de desarrollo. El uso de frameworks como Spring optimiza estos procesos, garantizando un software adaptable y competitivo.



## 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://n9.cl/obnv8>

# Arquitectura de Software basada en Microservicios

El artículo aborda los problemas de la arquitectura monolítica en el desarrollo de aplicaciones web en la Asamblea Nacional del Ecuador, como la dificultad de mantenimiento, implementación de cambios y escalabilidad. Se propone adoptar una arquitectura de microservicios, que divide las aplicaciones en servicios autónomos, mejorando la escalabilidad, el mantenimiento y la resiliencia. Aunque presenta desafíos, como la gestión de redes distribuidas y seguridad, los microservicios ofrecen soluciones más flexibles y efectivas para el desarrollo de software.

## Bibliografía

- López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.<https://n9.cl/0tbddk>

Aspecto	Arquitectura Monolítica	Arquitectura de Microservicios
Consistencia	Toda la funcionalidad implementada en una sola unidad.	Aplicación dividida en servicios separados e independientes.
Complejidad	Compleja, requiere actualizar toda la aplicación.	Modular, se puede desplegar cada servicio de forma individualizada.
Mantenibilidad	Es necesario actualizar toda la aplicación.	Escalable (se actualiza solo el servicio necesario).
Contaminación	Difícil, cambios en una parte afectan toda la aplicación.	Seguro, los servicios son autoinformantes y se pueden actualizar sin impacto global.
Flexibilidad de diseño	Flexible, un módulo puede afectar toda la aplicación.	Falta limitadas y rigurosas especificaciones jerárquicas y jerárquicas.
Portabilidad	Limitada por la transcodificación entre las unidades compatibles.	Cada servicio puede usar la tecnología más adecuada para su funcionamiento.

## Reflexión

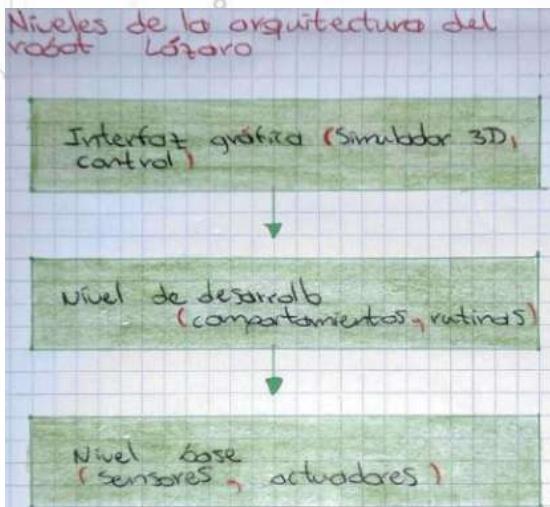
Ofrece modularidad y autonomía, mejorando la escalabilidad y el mantenimiento en aplicaciones críticas, como las de la Asamblea Nacional. Aunque introduce complejidad técnica, sus beneficios superan los retos, permitiendo actualizaciones sin afectar a los usuarios. Para una transición exitosa, es crucial contar con una estrategia clara, capacitación adecuada y herramientas apropiadas. Este modelo no solo mejora la eficiencia técnica, sino que también fomenta agilidad organizacional, adaptándose a las necesidades cambiantes.

# Arquitectura de software para el robot móvil Lázaro

El artículo presenta la arquitectura de software diseñada para el robot móvil "Lázaro", desarrollado en la Universidad Nacional Experimental del Táchira. La arquitectura consta de tres niveles: el nivel base, que conecta el hardware con el software y gestiona sensores y actuadores; el nivel de desarrollo, que permite crear rutinas y comportamientos del robot; y una interfaz gráfica que facilita la interacción del usuario mediante un panel de control y un simulador 3D. El robot, controlado remotamente a través de módulos XBee, puede operar en terrenos irregulares gracias a sus motores y sensores avanzados.

## Bibliografía

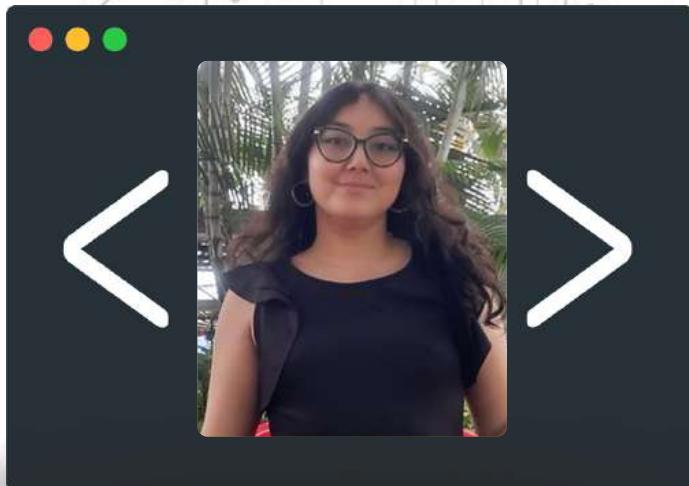
García, JM, Gil, Á. E., & Sánchez, E.A. (2018). Development of a software architecture for the Lázaro mobile robot. *Ingeniare. Chilean Journal of Engineering*, 26 (3), 376-390. <https://n9.cl/xueyi>



## Reflexión

La arquitectura del robot "Lázaro" destaca por su diseño modular, dividido en niveles que simplifican el desarrollo y la operación. Este enfoque permite a los usuarios interactuar fácilmente con el robot mediante una interfaz gráfica, mientras que los desarrolladores pueden integrar nuevas funcionalidades a través de la capa de desarrollo. Esta separación por niveles promueve la escalabilidad, facilitando futuras mejoras y adaptaciones del sistema.

# Mariana Charry Prada



Soy Mariana Charry Prada, comprometida con mi crecimiento profesional y personal. Actualmente, estudio "Análisis y Desarrollo de Software" en el SENA, Neiva, Huila, después de obtener mi título de Bachiller Técnico en Sistemas en el "Colegio Adventista Baluarte Interamericano".

En 2022, culminé el Técnico en "Programación de Software" en el SENA, iniciando mi camino en el desarrollo de software.

Me considero responsable, dinámica y con un fuerte deseo de superación.

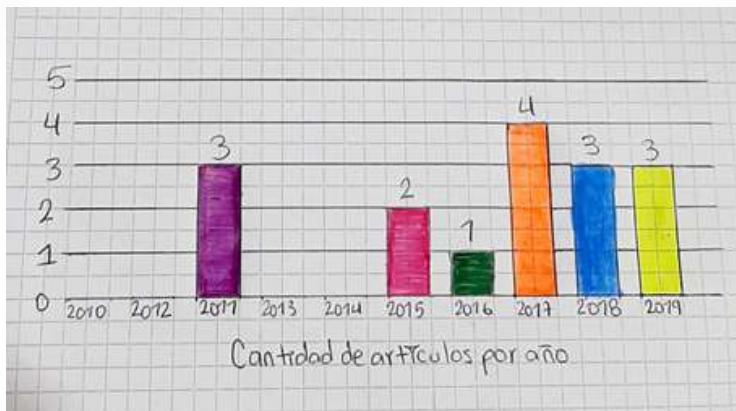
Poseo una excelente capacidad de trabajo en equipo, así como autonomía y organización. Valoro el trabajo colaborativo y siempre busco cumplir mis objetivos con calidad y compromiso.

# Análisis comparativo de patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad: Una revisión sistemática de la literatura

En el inmenso mundo de las aplicaciones móviles existen diversos patrones de diseño que nos brindan facilidades de crear un desarrollo más eficiente y organizado. El objetivo de la revisión es el de encontrar los principales estudios sobre patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad, y posteriormente determinar criterios de identificación que servirán como herramienta de selección de patrones de diseño de calidad.

## Reflexión

Los patrones de diseño han mostrado un mayor impacto en el desarrollo de software. Dado que los patrones de diseño proporcionan soluciones comprobadas, el desarrollo de aplicaciones móviles puede ser un proceso tedioso, ya que cada aplicación debe pasar por los ciclos de desarrollo para garantizar que la aplicación se ajuste a los atributos de calidad estándar. Luego de que el desarrollador compare los principales patrones de diseño de software, finalmente podrá implementar la más conveniente para el proyecto.



## Bibliografía

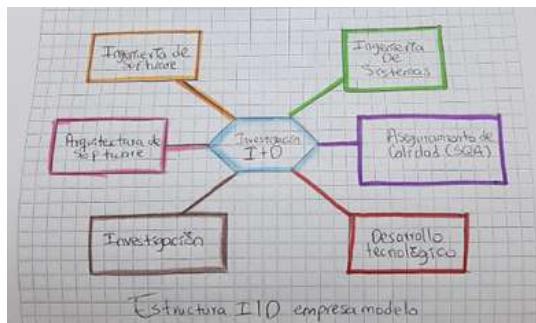
Abanto Cruz, J. A., & Gonzales Ramírez, O. F. (2019). Análisis comparativo de patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad: Una revisión sistemática de la literatura.

# Recomendaciones para la Formación de una Empresa de Desarrollo de Software Competitiva en un País como Colombia

Algunas de las recomendaciones para ser una empresa competitiva, son, buscar que la organización sea reconocida como una compañía transparente con los clientes, la estructura de la organización, debe estar fortalecida con un organismo que oriente el desarrollo tecnológico tener mecanismos que les permitan medir y saber si se están cumpliendo las metas establecidas en los procesos, los productos y proyectos de la compañía.

## Reflexión

Al tener una empresa dirigida al desarrollo de software, o al querer formarla, es necesario tener ciertos criterios importantes para de esa manera poder ser una empresa competitiva en todo este campo del desarrollo de software, más en un país como lo es Colombia, el cual en estos tiempos a tomado fuerza en el tema de la tecnología y sus componentes. Todo el artículo hablaba sobre recomendaciones, las nombraban y luego explicaban cada una de ellas, para que la organización de una compañía sea buena y resalte.

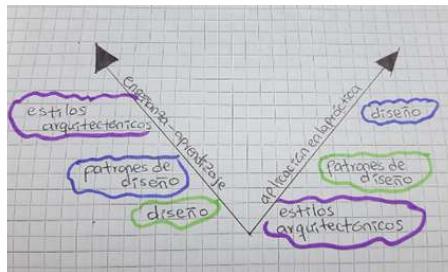


## Bibliografía

- Londoño, L. F. L. (2005). Recomendaciones para la Formación de una Empresa de Desarrollo de Software Competitiva en un País como Colombia. *Avances en Sistemas e Informática*, 2(1), 41-52.

# Introducción a la Arquitectura de Software

En los primeros años de la construcción de software no existía el diseño del sistema como una etapa independiente a la programación. A principios de la década del 90 algunos investigadores, comenzaron a ver la necesidad de investigar y desarrollar un nivel de abstracción superior al del diseño, al que llamaron "Arquitectura de software". La arquitectura del sistema es el resultado de combinar decisiones técnicas, sociales y del negocio.



## Reflexión

A partir de la creación de la arquitectura de software, se crearon de igual manera diferentes y variados modelos de desarrollo, como el "Modelo de Cascada", la arquitectura del sistema, la cual es la que se le sigue a la Ingeniería de Requerimientos. La fase de la arquitectura del sistema, se subdivide en tres etapas: elección del estilo arquitectónico, selección de los patrones de diseño y diseño de componentes; lo cual es importantes de aprender para llevarla a la práctica, también es importante conocer los estilos arquitectónicos, estos son una generalización de los patrones de diseño, como abstracción.

## Bibliografía

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

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación al Estado del Arte

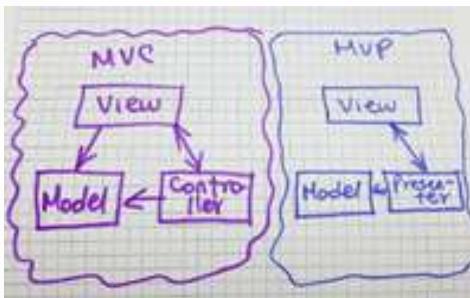
Mostrar el estado del arte en un área de la arquitectura de software llamada "Lenguajes de Patrones", desde sus orígenes los avances actuales y sus aplicaciones en la construcción de arquitecturas de software en diferentes dominios de aplicación. La extensibilidad y aplicabilidad de los lenguajes de patrones, se convierten en una herramienta importante para diseñadores e implementadores de todo tipo de sistemas de información. Es importante evitar la programación al estilo vaquero. Parches de último minuto o trabajo de última noche que pueden traer graves consecuencias.

## Reflexión

Para que un software sea bueno y tenga una funcionalidad correcta, es importante evitar hacer las cosas demasiado rápido, porque se pueden presentar especificaciones incompletas, lo cual puede resultar en un re-proceso, haciendo perder tiempo en la elaboración. La creación de nuevas formas de hacer las cosas ayuda en la productividad y una mayor calidad en los productos de software.

## Bibliografía

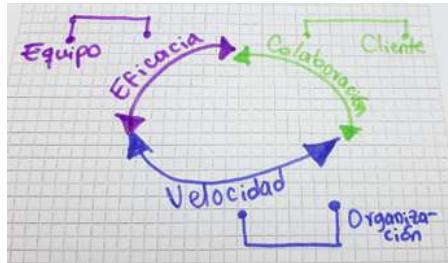
- Jimenez-Torres, V. H., Tello-Borja, W., & Rios-Patiño, J. I. (2014). Lenguajes de patrones de arquitectura de software: una aproximación al estado del arte. *Scientia et technica*, 19(4), 371-376.



# Integración de arquitectura de software en el ciclo de vida de las metodologías ágiles

Las metodologías ágiles se centran en el trabajo en equipo, la adaptabilidad y colaboración dentro del grupo de software y también entre los miembros del grupo y los usuarios finales. El uso de las metodologías (MA), ha marcado una tendencia en su adopción al desarrollo de proyectos de software (AR), en tanto, es una manifestación de decisiones de etapas muy tempranas del diseño sobre un sistema.

El tratamiento con enfoques diferentes en las primeras etapas, ha sido uno de los factores que ha causado la sensación de las MA y la AS.



## Reflexión

Las metodologías ágiles (MA) han transformado la manera en que se desarrollan proyectos de software, al centrado en la adaptabilidad, la colaboración y el trabajo en equipo. Su enfoque en las primeras etapas del diseño permite ajustar el rumbo del proyecto de manera temprana, lo cual es crucial en entornos dinámicos donde los requisitos pueden cambiar rápidamente. La flexibilidad de las MA permite que los equipos respondan de manera más eficiente a las necesidades del cliente, lo que genera productos finales más alineados con las expectativas del usuario. Este cambio en el enfoque, que prioriza la interacción constante y la retroalimentación temprana, ha sido fundamental para reducir los riesgos y mejorar la calidad del software.

## 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).

# Evaluación de una Arquitectura de Software

Una arquitectura de software es clave para que las organizaciones puedan avanzar y concentrarse en su función misional, solo si esta está bien diseñada. Es importante realizar evaluaciones tempranas a la arquitectura, mediante algún método, porque eso ayuda a la elección de una arquitectura, mejora la comunicación y permite una mejor interpretación de las historias de usuario. También tener en cuenta el análisis para el atributo de calidad modificabilidad.



## Reflexión

La arquitectura de software es el nacimiento sobre el que se construye el sistema, y su calidad tiene un impacto directo en la capacidad de una organización para alcanzar sus objetivos de negocio. Una buena arquitectura no solo asegura que los sistemas sean funcionales, sino que también facilita la adaptación, el mantenimiento y la escalabilidad a medida que las necesidades evolucionan. En este contexto, realizar evaluaciones tempranas y continuas de la arquitectura es crucial. Cuando se elige una arquitectura adecuada desde el principio, se pueden evitar costosos rediseños a medida que el sistema crece o se modifica.

## Bibliografía

Sanabria, E.R., & Rodríguez, S.V. (2021). Evaluación de una arquitectura de software. *Prospectiva*, 19(2).

# Arquitectura de software académica para la comprensión del desarrollo de software en capas

El desarrollo de software implica considerar una cantidad variada de aspectos tecnológicos. Entre los más destacados podemos mencionar los relacionados con el acceso a datos, las interfaces, los procesos funcionales, el control de las transacciones, la accesibilidad y la seguridad. Lograr un diseño coherente con los requerimientos planteados, niveles aceptables de flexibilidad, extensibilidad y usabilidad, así como facilitar las actividades de mantenimiento lleva a pensar la concepción del software en capas.



## Reflexión

El diseño en capas permite separar claramente las diferentes responsabilidades del sistema, como la capa de acceso a datos, la capa de lógica de negocio y la capa de presentación (interfaz de usuario). Esta separación facilita la modificación y evolución de cada capa de forma independiente. Por ejemplo, si se necesita cambiar el tipo de base de datos o actualizar un framework, podemos hacerlo sin alterar la lógica de negocio ni la interfaz de usuario.

## Bibliografía

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

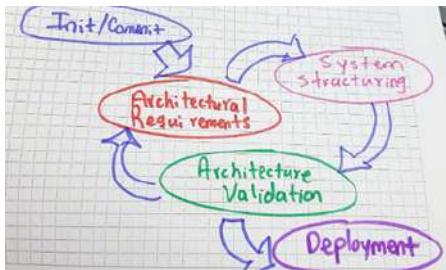
# **Arquitectura de software en el proceso de desarrollo ágil: una perspectiva basada en requisitos significantes para la arquitectura.**

Recopilar, comprender y gestionar los requisitos es un aspecto crítico en todos los métodos de desarrollo. Esto también es cierto para las metodologías ágiles en la que la captura de requisitos es realizada en todo el proceso de desarrollo, con requisitos que van evolucionando y cambiando a lo largo del ciclo de vida.

Este proceso es opuesto al enfoque de la arquitectura del software, donde los requerimientos deben ser identificados, recabados y comprendidos.

## **Bibliografía**

Navarro, M.E., Moreno, M.P., Aranda, J., Parra, L., Rueda, J.R., & 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).



## **Reflexión**

La reflexión sobre la gestión de requisitos en las metodologías ágiles frente al enfoque tradicional de la arquitectura de software revela una de las tensiones más fundamentales en el desarrollo de software. En un enfoque tradicional o "en cascada", la captura y definición de requisitos es una fase crítica inicial. Una vez que estos requisitos son comprendidos y documentados, la arquitectura del software se diseña y se ajusta para cumplir con esos requisitos. En este escenario, los requisitos se entienden como algo fijo, que debe ser meticuloso.

# **La Arquitectura de Información (AI) en el proceso de desarrollador de Software.**

La AI se ha convertido para la producción de software en un proceso determinante con vistas a que los productos alcancen la calidad requerida. El objetivo de la investigación fue demostrar la importancia del rol del arquitecto de información en el proceso de desarrollo de software, para la cual se analizan teóricamente los elementos que identifican el proceso de AI y se describen las Metodologías Rup y XP en cuanto a roles.

## **Reflexión**

En un entorno cada vez más dominado por la automatización y el uso de la inteligencia artificial, el rol del arquitecto de información no solo sigue siendo relevante, sino que se ha intensificado. Este profesional no solo se encarga de estructurar la organización del conocimiento y la información dentro del sistema, sino que también es esencial para garantizar que el software sea escalable, sostenible y funcional a lo largo del tiempo.



## **Bibliografía**

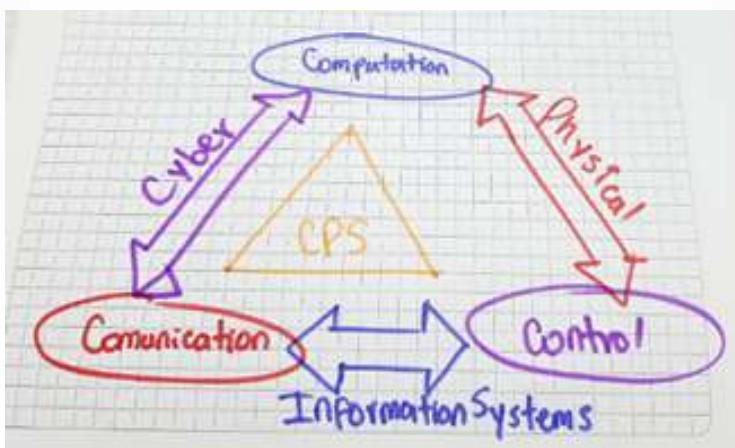
Moyares, Y.; & Lorenzo, D.B.; (2021). La Arquitectura de información (AI) en el proceso de desarrollo de software. *Bibliotecas Anuales de Investigación*, 6, 97- 102.

# Arquitectura de Software para los actuales sistemas ciberfísicos.

La próxima generación de sistema ciber-físicos, plantea grandes desafíos en el diseño de software. No se trata sólo de diseñar un sistema entorno a los plazos de ejecución, lo más importante es maximizar la utilización de recursos. En el artículo se proponía un sistema base para la arquitectura de software en el que los servicios se puedan diseñar e implementar y componer fácilmente de acuerdo con la demanda.

## Reflexión

Los CPS, al integrar componentes computacionales y físicos de manera estrecha, no solo requieren sistemas de software robustos, sino que deben maximizar la eficiencia en el uso de los recursos disponibles. Esta tarea se complica aún más cuando se deben tener en cuenta factores como la latencia, la confiabilidad, la seguridad y la escalabilidad.



## Bibliografía

Ting, J.S.(2011). Arquitectura de software para los actuales sistemas ciberfísicos. Revista Ingenierías USBmed, 2(1), 29.

# Arquitectura de software dinámica basada en la reflexión.

La tesis sobre el artículo, se centraba en el campo de la arquitectura de software, una rama de la Ingeniería de software dedicada al estudio de la estructura de los sistemas software complejos. Se trata y estudia concretamente uno de los problemas pendientes del campo: la especificación y descripción de arquitecturas de software dinámicas, es decir, aquellas cuya estructura puede variar. Con esto se elabora de manera informal un modelo reflexivo de descripción arquitectónica de software.

## Reflexión



La tesis sobre el artículo que aborda el campo de la arquitectura de software, una rama crucial dentro de la Ingeniería de Software, resalta la importancia de la estructura de los sistemas software complejos. Este campo se encarga de analizar cómo se organizan y se interrelacionan los componentes dentro de un sistema para garantizar su efectividad, escalabilidad, y flexibilidad.

## Bibliografía

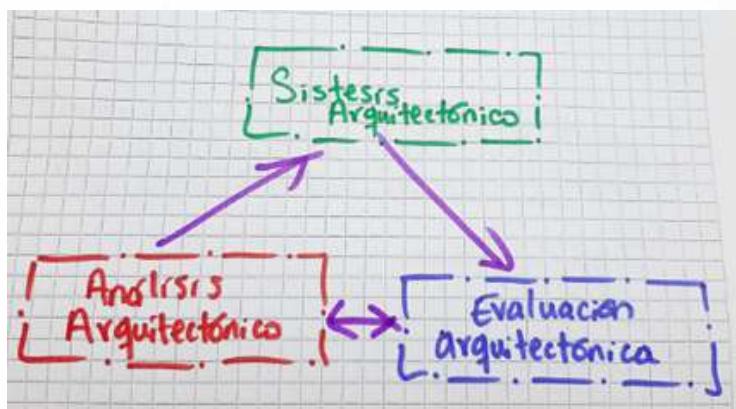
CUESTA QUINTERO, C.E.(2002). Arquitectura de software dinámica basada en reflexión (Doctoral dissertation, Universidad de Valladolid).

# Representación y razonamiento sobre las decisiones de diseño de arquitectura de software.

El objetivo general de la tesis estipulada en el artículo, estaba directamente vinculada a brindar soporte al arquitecto de software durante el diseño arquitectónico. Partiendo de la hipótesis: los arquitectos de software reutilizan soluciones conocidas en nuevos diseños, se identifica la necesidad de contar con una herramienta, tanto conceptual como computacional, que pueda colaborar con los arquitectos de software.

## Reflexión

La reutilización de soluciones conocidas. Esta práctica, que consiste en aplicar patrones, diseños o componentes previamente probados y validados, es una estrategia fundamental para mejorar la eficiencia, reducir los costos de desarrollo y aumentar la calidad del software. La hipótesis planteada —que los arquitectos de software reutilizan soluciones conocidas en nuevos diseños— es completamente válida y refleja una realidad observada en la mayoría de los proyectos de desarrollo de software a gran escala.



## Bibliografía

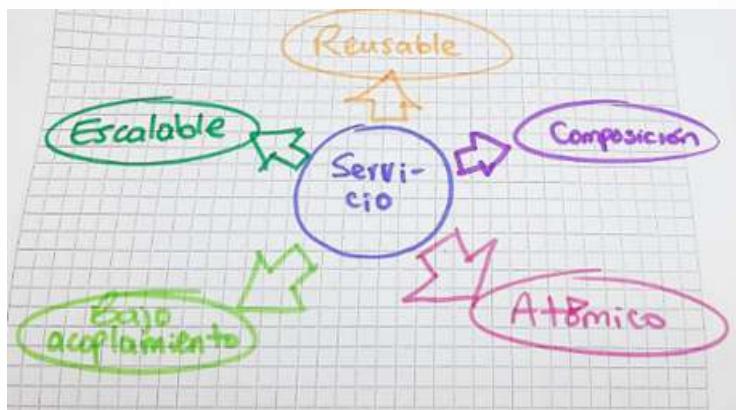
Carignano, M.C.(2016, June). Representación y razonamiento sobre las decisiones de diseño de arquitectura de software. In XVIII Workshop de Investigadores en Ciencias de la computación.) (WICC 2016, entre Ríos, Argentina).

# Arquitectura de software. Arquitectura orientadas a servicios.

Dentro del proceso de desarrollo de software de un sistema de los temas importantes a tratar es: la arquitectura de software, es la parte de la ingeniería de software que se dedica al estudio, análisis y descripción para formalizar el esquema global de un sistema; poniendo énfasis en el estudio de las interacciones entre sus elementos básicos, denominados componentes. En el artículo se realizó un estudio general de la arquitectura de software haciendo énfasis en la arquitectura orientada a servicios.

## Reflexión

La arquitectura de software es un componente fundamental dentro del proceso de desarrollo de software, ya que establece las bases para la estructura general del sistema. Su objetivo es organizar los diversos elementos que conforman el sistema de forma eficiente, asegurando que interactúen correctamente entre sí y con otros sistemas externos. Se encarga de describir, analizar y formalizar el esquema global del sistema, lo que incluye la distribución de tareas, la asignación de responsabilidades y las relaciones entre los diferentes componentes del sistema.



## Bibliografía

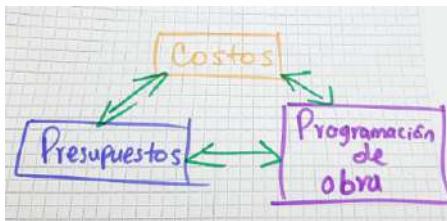
Martín, Y.E(2012). Arquitectura de Software. Arquitectura orientada a servicios. Serie científica de la Universidad de las Ciencias informáticas, 5(1), 1-10.

# **Arquitectura de Software para el desarrollo de herramienta tecnológica de costos, presupuestos y programación de obra.**

Durante todo el artículo trataron temas relacionados a la arquitectura de software, como los casos de uso, diagramas de flujo del software, diagramas de despliegue, diagramas de paquetes del software. Por último, nombraron todo sobre la cronología, metodología de desarrollo de costos, presupuestos y programación de obra que se debe llevar a cabo. En calidad, establece listas, establece valores de los costos indirectos, calcular el costo total del proyecto y realizar la programación de obra.

## **Bibliografía**

Cárdenas- Gutiérrez, J.A., Barrientos- Monsalves, E, S, & Molina- Salazar L.(2022). Arquitectura de software para el desarrollo de herramientas tecnológicas de costos, presupuestos y programación de obra. I+D Revista de investigaciones, 17(1), 89-100.



## **Reflexión**

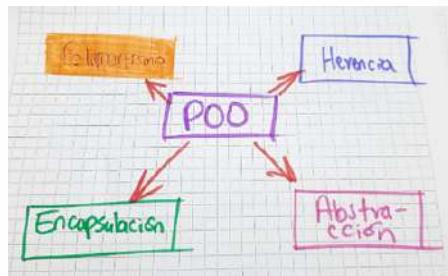
La arquitectura de software es la base sobre la cual se construye el sistema, y como tal, debe ser cuidadosamente diseñada y documentada. Los diagramas de flujo, los diagramas de despliegue, los casos de uso y los diagramas de paquetes permiten representar de forma visual y estructurada cómo interactúan los diferentes componentes del sistema. Estos artefactos son fundamentales para garantizar que todos los elementos del software estén bien definidos y que sus interacciones sean claras.

# Arquitectura de software con programación orientada a objetos.

El objetivo del artículo fue describir la arquitectura de software con programación orientada a objetos a partir de la revisión de fuentes de documentales actualizadas. Se encontró que la POO, agrupa un conjunto de técnicas que permiten desarrollar, y mantener mucho más fácilmente, programas de una gran complejidad. La evolución de la arquitectura de software, escala cada vez más posiciones superiores, que permiten a los profesionales resolver diversos problemas.

## Bibliografía

Vera, J.B.V.(2023). Arquitectura de software con programación orientada a objetos. Polo del conocimiento: Revista científico-profesional, 8(12), 1497-1508.

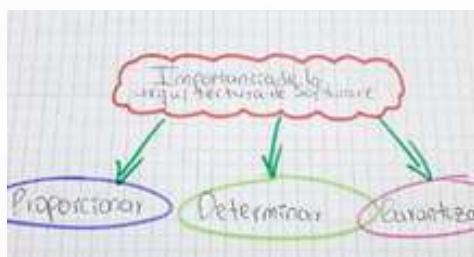


## Reflexión

La arquitectura de software con programación orientada a objetos (POO) ha demostrado ser una de las metodologías más poderosas y efectivas para el desarrollo de software, especialmente en el contexto de sistemas complejos. A través de la revisión de fuentes documentales actualizadas, se ha comprobado que la POO agrupa un conjunto de técnicas y principios que facilitan no solo el desarrollo, sino también el mantenimiento y evolución de programas con un alto grado de complejidad.

# La importancia de la Arquitectura de software

El software es un conjunto de programas de cómputo, procedimientos, estructura de datos, reglas documentación y datos asociados que forman parte de las operaciones de un sistema de computación. Se necesita tener un panorama completo del sistema, como el ciclo de vida de la arquitectura, en donde haya requisitos, un diseño, documentación, evaluación e implementación; en general, donde haya una buena práctica de la arquitectura.



## Bibliografía

Mamani, S.M.(2023). La importancia de la Arquitectura de software. In Memorias del Congreso Nacional de Informática (No.1, pp.41-50).

## Reflexión

La definición de software como un conjunto de programas, procedimientos, estructuras de datos, reglas, documentación y datos asociados resalta la complejidad integral de lo que implica crear y mantener sistemas de computación. Es crucial reconocer que, más allá de escribir código, el software es un sistema holístico que requiere un enfoque organizado y sistemático para garantizar su funcionalidad, eficiencia y calidad.

# Arquitectura de software

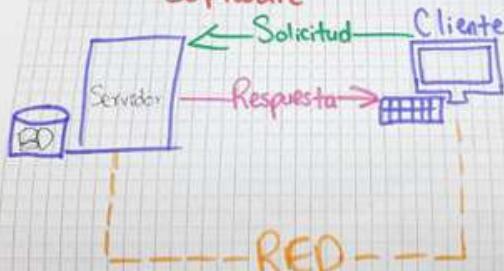
La arquitectura de software permite evaluar la solución de un software desde las primeras fases de su desarrollo, apartando numerosos beneficios a cualquier proyecto de software que se realice. Permite la evaluación de la solución desde fases muy tempranas de desarrollo.

Determina la viabilidad de un proyecto, si tiene grandes y complejos sistemas.

## Bibliografía

Nuñez, G. & Camacho, E. (2004). Arquitectura de software. Guía de estudio.

### Arquitectura de Software

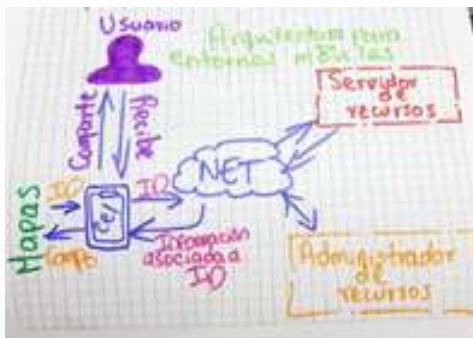


## Reflexión

La arquitectura de software juega un papel fundamental en el éxito de cualquier proyecto de desarrollo de software, y su capacidad para evaluar la viabilidad de una solución desde las primeras fases del proyecto es uno de sus beneficios más valiosos. La arquitectura no es simplemente un conjunto de estructuras y componentes del sistema, sino un marco estratégico que define cómo se organizará y evolucionará todo el sistema.

# Arquitectura de software para entornos móviles

La evaluación de los sistemas operativos está causando un gran impacto, porque cada vez son más estables y robustos, lo que permite a los desarrolladores de software, crear aplicaciones móviles de mayor tamaño y complejidad. El objetivo del artículo fue definir una solución arquitectónica móvil que compartiera algunos de los principios más reconocidos de la arquitectura de software en general, y de esa forma, ayudar a estandarizar y adaptar metodologías, métodos, procesos y enfoques.



## Bibliografía

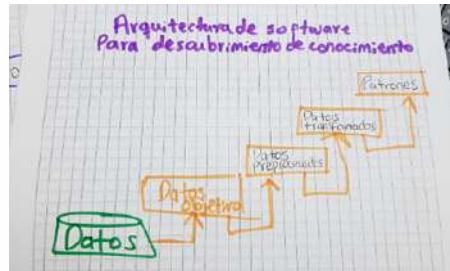
- Granada, E.Z., Rodríguez, L.E.S., Montoya, C.E.G., & Uribe, C.A.C. (2014). Arquitectura de software para entornos móviles. Revista de Investigaciones Universidad del Quindío, 25 (1), 20-27.

## Reflexión

La evaluación de los sistemas operativos móviles y su evolución hacia soluciones más estables y robustas ha tenido un impacto significativo en la capacidad de los desarrolladores para crear aplicaciones móviles de mayor tamaño y complejidad. Este avance no solo ha mejorado el rendimiento y la fiabilidad de las aplicaciones, sino que también ha proporcionado una base más sólida sobre la cual construir software innovador y de alta calidad.

# Arquitectura de software para descubrimiento de conocimiento

La mayoría de las organizaciones cuentan con sistemas de gestión de bases de datos. Se puede descubrir conocimientos en grandes volúmenes de datos que se encuentran almacenados, ya sea en archivos o en bases de datos. Es un proceso interactivo e iterativo que implica integrar en una herramienta interrogaciones, análisis y métodos de visualización que le facilitan al usuario. El proceso de descubrimiento de conocimiento se puede aplicar en diferentes dominios.



## Reflexión

El proceso de descubrimiento de conocimiento en bases de datos ha ganado una gran relevancia en las organizaciones actuales, especialmente con el volumen masivo de datos generados y almacenados en sistemas de gestión de bases de datos. Estos grandes volúmenes de datos, provenientes de diversas fuentes y plataformas, contienen patrones, tendencias y relaciones que, cuando son correctamente analizados, pueden proporcionar un valor significativo para la toma de decisiones, la optimización de procesos y la innovación empresarial.

## Bibliografía

De Abadía, M.E.V., Cuevas, C.M.G., González, M.E.M., & Bueno, J.A.A., (2001). Arquitectura de software para descubrimientos de conocimiento. Energía y Computación, 10 (1), 6-13.

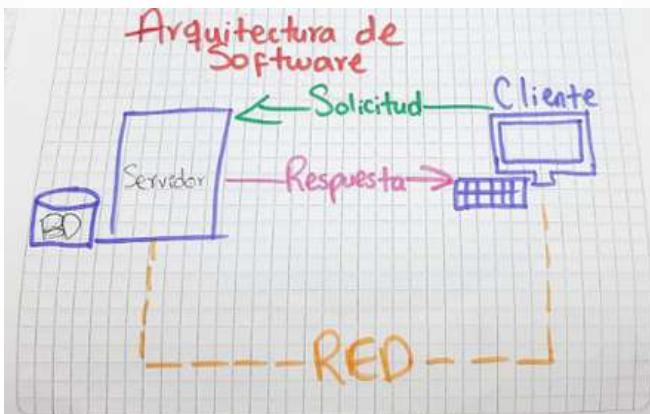
# Diseño de un mundo virtual para la enseñanza de arquitectura de software

El objetivo principal del proyecto hablado en el artículo fue crear un mundo virtual en el que se enseñe Arquitectura de Software a nivel universitario. Algo importante para lograr este fin fue la investigación sobre las tecnologías, lo que hizo que dicha idea fuera viable.

Se estudiaron herramientas que pudieron ayudar a la consecución del proyecto y a la implantación del mismo, contando con las diferentes posibilidades que se plantearon. Definir los pasos que se deberían realizar para la construcción del sistema.

## Reflexión

La creación de un mundo virtual para enseñar Arquitectura de Software a nivel universitario es una iniciativa innovadora que destaca la importancia de integrar nuevas tecnologías en el proceso educativo. La idea de utilizar un entorno virtual para enseñar conceptos complejos como la arquitectura de software no solo hace que el aprendizaje sea más interactivo, sino que también ofrece un espacio dinámico y flexible donde los estudiantes pueden experimentar y aplicar los principios de la arquitectura en un contexto más inmersivo.



## Bibliografía

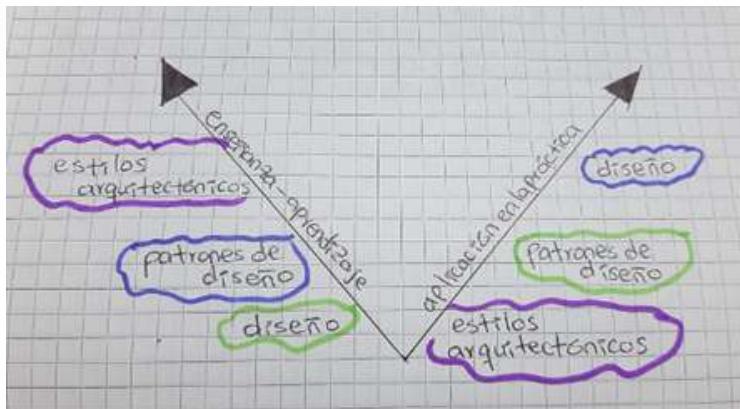
Casado Manzanero, V. (2009), Diseño de un mundo virtual para la enseñanza de arquitectura de software.

# Una teoría para el diseño de software.

El resultado de diseñar un sistema es una de las cuatro descripciones fundamentales que debe realizar un ingeniero de software. A esta descripción se la conoce como diseño de software, arquitectura de software o estructura de software. Es conveniente dedicar tiempo y esfuerzo en diseñar un sistema de software porque es más barato desarrollarlo y mantener, que hacerlo como se lo hace habitualmente. El mantenimiento consiste esencialmente en cambiar, agregar o eliminar líneas de código al software.

## Reflexión

El diseño de software es una de las actividades más críticas en el desarrollo de sistemas, y su importancia no solo radica en la creación de un sistema funcional, sino también en la sostenibilidad y la eficiencia a largo plazo del software. Al reflexionar sobre la afirmación de que es más barato diseñar un sistema adecuadamente que simplemente comenzar a programar sin un diseño previo, podemos comprender mejor la justificación económica y estratégica de un buen proceso de diseño.



## Bibliografía

Cristián, M. (2021). Una teoría para el diseño de software.

# Definición de arquitectura para una línea de producto de software

El artículo el cual trataba sobre el arte en la definición de arquitecturas en las líneas de producto de software, por lo que en él se trataron diversos conceptos de Arquitectura de software encontrados en la literatura revisada. El modelado de arquitectura es el esfuerzo de capturar y documentar las decisiones de diseño que conforman la arquitectura del sistema. Una línea de producto es un conjunto de sistemas que comparten un conjunto de características para satisfacer las necesidades específicas de un segmento de mercado particular.

## Reflexión

El artículo sobre arquitectura de software en líneas de producto pone de manifiesto un enfoque muy interesante y estratégico en la ingeniería de software. Al combinar el arte de la definición de arquitecturas con el concepto de líneas de producto, se aborda un desafío importante: cómo diseñar sistemas que sean lo suficientemente flexibles y reutilizables para adaptarse a diferentes contextos y necesidades del mercado, mientras se mantienen eficientes y coherentes a lo largo de su ciclo de vida.



## Bibliografía

Romo Moreno, A. (2009). Definición de Arquitectura para una línea de productos de software.

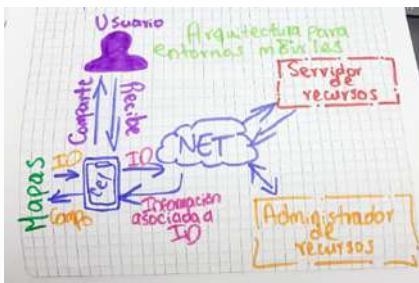
# SERVICIOS WEB EN TELEFONÍA CELULAR

La evolución de la telefonía móvil ha permitido el desarrollo de estándares con el fin de facilitar la adquisición y conocimiento de servicios Web a los usuarios de móviles. La llegada de nuevas tecnologías como SOA (Service Oriented Architecture) abre un amplio campo de posibilidades para el desarrollo de diversas aplicaciones cuyo propósito es facilitar el acceso a contenido informativo e interactivo.

## Reflexión

La evolución de la telefonía móvil ha transformado profundamente la forma en que interactuamos con el mundo, tanto en términos de comunicación como de acceso a la información. En este proceso, los avances en tecnologías como SOA (Arquitectura Orientada a Servicios) han jugado un papel crucial, al permitir una integración más eficiente y flexible de diversos servicios a través de dispositivos móviles.

SOA, como paradigma arquitectónico, se basa en la creación de servicios independientes y reutilizables que pueden ser accedidos por diferentes aplicaciones y sistemas.



## Bibliografía

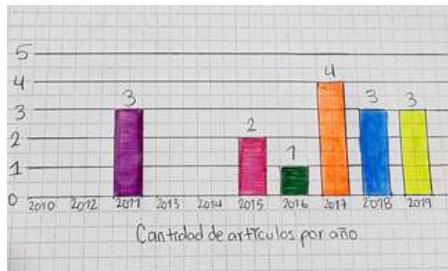
- Santos, L. M., Rico, D. W., & Rincón, A. A. (2009). Servicios web en telefonía celular. *Scientia et technica*, 15(42), 363-368.

# Protección de Datos Personales en Ecuador y Colombia: Principios, Ética y Desafíos Actuales

Se analizó en profundidad el concepto de protección de datos y del uso de los datos personales en países como Ecuador y Colombia donde se hace referencia a la forma de tratamiento de datos donde se examina por categorías y se especifica los principios que se deben aplicar para realizar el proceso de tratamiento de datos los cuales son: legalidad, libertad, veracidad, transparencia, acceso y circulación restringida, seguridad y confidencialidad. Por otro lado, también manifiesta al perfil ético que tiene el responsable para no cometer actos ilícitos con la información entregada y examina los tratamientos que impliquen el uso de datos biométricos.

## Bibliografía

Quishpe, M. V., Moreano, J. C., Guanoluisa, A. G., & Atavallo, C. C. (2023). Protección de Datos Personales en Ecuador y Colombia: Principios, Ética y Desafíos Actuales. Revista Científica de Informática ENcriptar-ISSN: 2737-6389, 6(11), 19-34.



## Reflexión

La protección de datos y el uso de los datos personales son elementos esenciales para garantizar la privacidad y la seguridad de los individuos en un contexto digital y globalizado. En países como Ecuador y Colombia, donde las normativas sobre protección de datos personales están claramente definidas, el marco legal se orienta a preservar los derechos fundamentales de los ciudadanos frente al uso de su información personal. La reflexión que surge en torno a este tema involucra tanto los principios legales y éticos como los desafíos que presentan los avances tecnológicos y la globalización.

# Criterios de evaluación de plataformas de desarrollo de aplicaciones empresariales para ambientes web

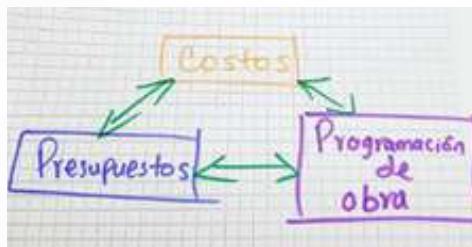
Las aplicaciones web han estado tomando fuerza en los últimos años, esto debido a la practicidad de las mismas, entre otras razones, por la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales y son accesibles desde cualquier lugar del mundo gracias a la red de redes, Internet. Los mecanismos de desarrollo de aplicaciones Web, recogen elementos comunes al desarrollo de aplicaciones empresariales, pero tienen características propias en análisis, diseño, e implementación. Estos elementos serán independientes del estilo arquitectónico que se decida implementar y también de la arquitectura de software.

## Reflexión

Una de las principales ventajas de las aplicaciones web es su capacidad de operar de manera independiente del sistema operativo del usuario. A diferencia de las aplicaciones tradicionales, que deben ser adaptadas y distribuidas para cada plataforma (Windows, macOS, Linux, etc.), las aplicaciones web se ejecutan en navegadores. Esto les permite ser accesibles desde cualquier dispositivo con acceso a Internet, lo que aumenta enormemente su alcance y flexibilidad. La independencia del sistema operativo también facilita la adopción de nuevas tecnologías, ya que los usuarios no se ven limitados por las especificaciones o compatibilidades de sus dispositivos.

## Bibliografía

Trejos Arroyave, M. H., & Zamora Cardona, D. F. (2012). Criterios de evaluación de plataformas de desarrollo de aplicaciones empresariales para ambientes web.

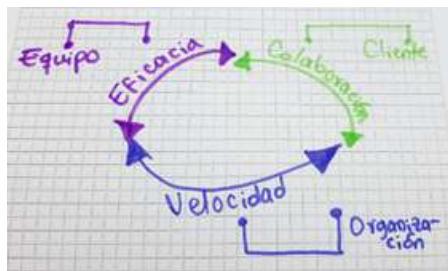


# Análisis comparativo de Patrones de Diseño de Software

Los patrones de diseño brindan soluciones a problemas que se presentan durante el desarrollo de software, evitan duplicaciones de código y facilitan su reutilización. En el presente artículo se detallan la estructura, componentes, ventajas y desventajas de los patrones de diseño: Template Method, Model-View-Controller, Model-View-Presenter, Model Front Controller y Model-View-View-Model MVVM. La investigación se realizó a través de una revisión bibliográfica en bases de datos científicas y consecuentemente se determinaron las métricas que permitieron comparar los patrones en estudio. Mediante el análisis comparativo de métricas y parámetros entre los patrones se establece que no existe un patrón superior a nivel general, pues cada patrón tiene su propósito definido y el desarrollador de software es quien debe identificar cuando un patrón se adapta mejor a la solución que desea desarrollar.

## 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.



## Reflexión

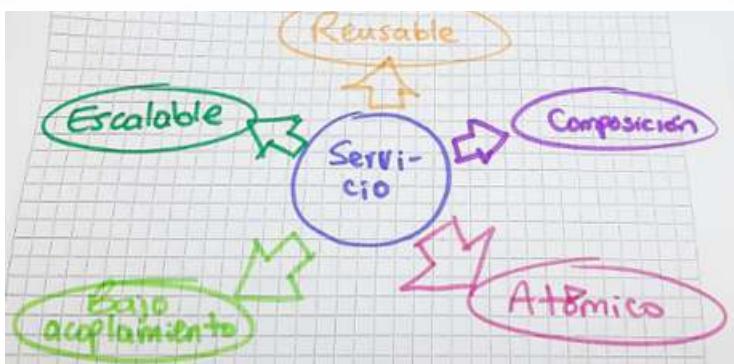
Es interesante cómo los patrones de diseño funcionan como herramientas fundamentales para mantener la calidad y modularidad del código. Aunque no existe un patrón "superior" universalmente, el verdadero desafío radica en identificar cuál se adapta mejor a las necesidades del proyecto en particular. Este enfoque flexible y adaptativo permite a los desarrolladores crear soluciones más eficientes, manteniendo la integridad y la claridad del código a largo plazo. Al final, la habilidad de seleccionar el patrón adecuado se convierte en un arte que mejora tanto la mantenibilidad como la escalabilidad del software.

# Arquitectura de software, esquemas y servicios

Cuando en la industria de software los productos tienen requerimientos cada vez más complejos y dinámicos, y los tiempos para desarrollarlos son cada vez menores; la reutilización y el bajo acoplamiento entre los componentes cobran vital importancia. En el artículo de investigación, partiendo de las definiciones de arquitectura de software y esquema, se tratan las características del paradigma de arquitectura orientada a servicios y se exponen algunos elementos significativos que muestran cómo los servicios son la evolución natural de los componentes de software. También se comentaban algunas cuestiones a tener en cuenta a la hora de diseñar orientado a servicios.

## Reflexión

En un mundo donde los tiempos de desarrollo son cada vez más cortos y las expectativas sobre las funcionalidades de los sistemas son cada vez más altas, los métodos tradicionales de desarrollo de software tienden a volverse insuficientes. Los equipos de desarrollo se enfrentan al reto de crear soluciones más rápidas sin comprometer la calidad o la capacidad de adaptación. Aquí, la reutilización de componentes y el bajo acoplamiento entre los diferentes módulos o servicios permiten crear arquitecturas de software más rápidas.

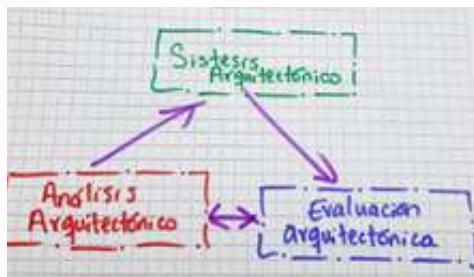


## Bibliografía

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

# Esquema basado en UML para representaciones de arquitectura de software

En los últimos años se ha llevado a cabo una importante cantidad de investigaciones en el campo de la arquitectura de software. El objetivo de muchos de estos estudios es encontrar un sistema de representación capaz de ir más allá de la informalidad del diagrama tradicional de "cajas y líneas", pero manteniendo un nivel de complejidad bajo, de modo que pueda utilizarse como herramienta de comunicación entre todos los interesados en un proyecto de software.



## Bibliografía

Gil, S. V. H. (2003). Representación de la arquitectura de software usando UML. *Sistemas y Telemática*, 1(1), 63-75.

## Reflexión

la evolución de la arquitectura de software y su representación en los últimos años apunta a una necesidad cada vez más clara: encontrar un equilibrio entre la formalidad y la accesibilidad. En el pasado, los diagramas de "cajas y líneas" eran, y en muchos casos siguen siendo, una herramienta visual útil para representar componentes y relaciones dentro de un sistema. Sin embargo, su simplicidad también ha sido uno de sus límites con el crecimiento de los sistemas de software y su integración en entornos más complejos y cambiantes.

# Arquitectura software de sistemas abiertos

El aumento de la complejidad de los sistemas software ha puesto de manifiesto la importancia que tiene la Arquitectura del Software en todo el proceso de desarrollo y mantenimiento del mismo. Frente a una visión tradicional centrada fundamentalmente en lo que se conoce como diseño arquitectónico, enmarcado dentro del campo más amplio de la Ingeniería del Software, los trabajos más recientes están orientados a considerar la Arquitectura del Software como un nuevo campo de interés por sí mismo. En este campo, los principales objetivos siguen siendo la especificación y diseño de los aspectos estructurales del software, pero haciendo un mayor énfasis en la especificación de las interrelaciones entre componentes y en el diseño de lenguajes de descripción de arquitecturas bien definidos.

## Bibliografía

- Linero, J. M. T. (1996). Arquitectura software de sistemas abiertos. In II Jornadas de informática. Actas: Almuñécar (Granada), 15 al 19 de julio 1996 (p. 3).



## Reflexión

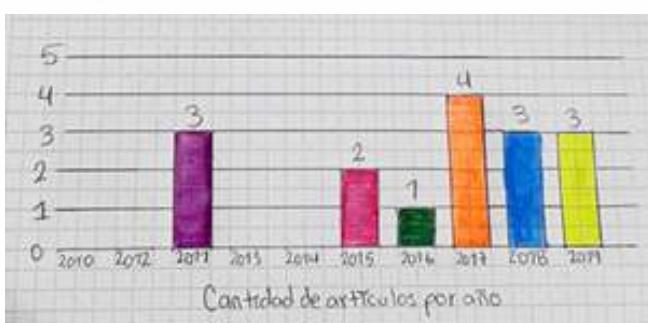
La evolución de la Arquitectura del Software refleja el creciente reconocimiento de su papel fundamental en el éxito de los sistemas software. Tradicionalmente, la arquitectura se entendía como una fase del diseño dentro del proceso global de desarrollo, con énfasis en la creación de un conjunto de componentes y la definición de sus relaciones. Sin embargo, al aumentar la complejidad de los sistemas, el enfoque ha pasado a ser mucho más holístico, reconociendo que la arquitectura no solo debe ser una fase técnica, sino también un campo de estudio y práctica en sí.

# Título: Visión de las competencias en arquitectura de software integrando las perspectivas de la industria y la academia

La formación de un arquitecto de software es una labor compleja que requiere de una mezcla de experiencia y conocimiento especializado que es difícil lograr en el contexto universitario. Este artículo busca determinar las competencias mínimas que debe lograr un arquitecto de software cubriendo la expectativa de la industria, así como el contexto formativo de las universidades e instituciones de educación superior. Para identificar y documentar estas competencias, se realizó un ciclo de investigación-acción, en el cual se diseñó un estudio basado en encuestas y talleres en el que participaron ingenieros de software de la industria y profesores universitarios que imparten cursos relacionados con el diseño y evaluación de la arquitectura.

## Reflexión

La formación de un arquitecto de software es, efectivamente, un desafío complejo debido a la naturaleza multidisciplinaria de esta profesión y la rápida evolución de la tecnología. Como señala, las universidades y las instituciones de educación superior habituales no pueden ofrecer la experiencia práctica completa que los arquitectos de software necesitan para enfrentar los desafíos reales de la industria. En este sentido, es crucial identificar y definir las competencias mínimas que debe tener un arquitecto de software, ya que estas habilidades no solo deben estar alineadas con las exigencias del mercado laboral, sino también con la capacidad de adaptación constante a nuevas tecnologías y metodologías.



## Bibliografía

- Yépez, W. L. P., Alegría, A. F. S., Bandi, A., & Alegría, J. A. H. (2024). Visión de las competencias en arquitectura de software integrando las perspectivas de la industria y la academia. REVISTA COLOMBIANA DE TECNOLOGIAS DE AVANZADA (RCTA), 1(43), 9-23.

# Metodologías Ágiles en el Desarrollo de Software

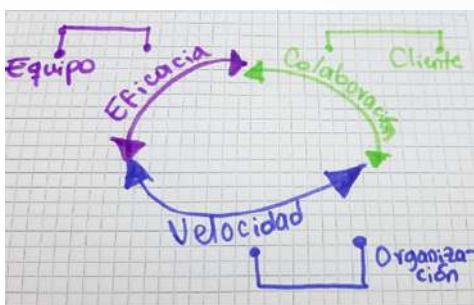
El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados.

## Bibliografía

Canós, J. H., Letelier, P., & Penadés, M. C. (2003). Metodologías ágiles en el desarrollo de software. Universidad Politécnica de Valencia, Valencia, 1-8.

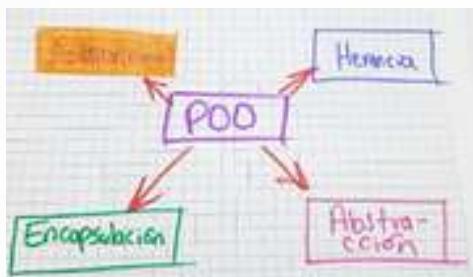
## Reflexión

El desarrollo de software es una disciplina compleja y multifacética que involucra una gran cantidad de factores técnicos, humanos y organizacionales. Como bien menciona, las metodologías tradicionales han sido fundamentales en la estructuración del proceso de desarrollo, proporcionando un marco claro que ayuda a gestionar las actividades, los artefactos y las herramientas. Este enfoque ha demostrado ser eficaz en muchos contextos, especialmente cuando se trata de proyectos grandes y bien definidos, donde el control y la predictibilidad son esencia.



# RADS: una herramienta para reutilizar estrategias en diseños de arquitecturas de software

El diseño de arquitecturas de software es un proceso altamente creativo que aún no ha sido estandarizado, por lo que las actividades llevadas a cabo para construir la arquitectura de un sistema son aquellas que los arquitectos involucrados consideran convenientes y pertinentes según el método de diseño que utilicen, su experiencia, conocimientos y habilidades personales. La reutilización es una práctica habitual dentro de dicha actividad. Sin embargo, no existen herramientas que asistan a los arquitectos para llevarla a cabo.



## Bibliografía

- Carignano, M. C., Gonnet, S. M., & Leone, H. P. (2016, November). RADS: una herramienta para reutilizar estrategias en diseños de arquitecturas de software. In Simposio Argentino de Ingeniería de Software (ASSE 2016)-JAIIO 45 (Tres de Febrero, 2016).

## Reflexión

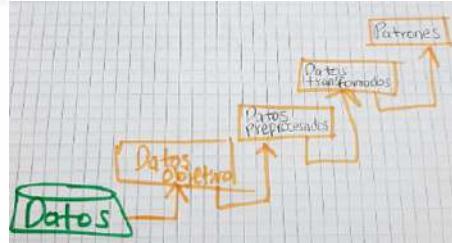
La reflexión sobre el diseño de arquitecturas de software como un proceso creativo, y la falta de herramientas específicas para la reutilización, revela varia. En primer lugar, el hecho de que la creación de arquitecturas de software no esté estandarizada resalta la naturaleza profundamente personalizada y subjetiva de este proceso. Cada arquitecto, dependiendo de su experiencia y enfoque, puede abordar un problema de una manera única. Esto es un reflejo de la complejidad inherente a la arquitectura de software, donde los desafíos son tan diversos y cambiantes que un enfoque rígido o universal no suele ser efectivo.

# Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software

las metodologías de Desarrollo de Software (DS.) han experimentado un proceso histórico y evolutivo que inicia en los años 40 con la aparición de las primeras computadoras, entonces no se contaban con parámetros ni estándares, el DS. Era prácticamente empírico y artesanal lo que llevó a que una buena parte de los proyectos fallaran en cubrir las exceptivas de los usuarios, así como en entregas extemporáneas y presupuestos excedidos, sobreviniendo la "crisis del Software" la respuesta para superarla fue la adopción de modelos y metodologías clásicas que progresivamente fueron incorporando estándares, controles y formalidades al DS. En un afán que llegó a ser definido como "triángulo de hierro."

## Bibliografía

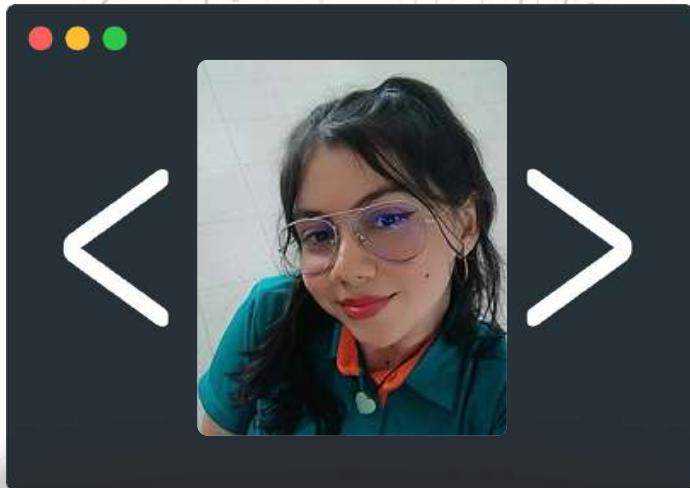
Gamboa, J. Z. (2018). Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software. INNOVA Research Journal, 3(10), 20-33.



## Reflexión

La evolución de las metodologías de Desarrollo de Software (DS) refleja un aprendizaje colectivo que nace de los fracasos y dificultades en los primeros días de la informática. En sus inicios, el desarrollo de software carecía de las estructuras y prácticas que hoy consideramos esenciales. Esto daba lugar a un proceso artesanal, donde la creatividad y las habilidades individuales eran claves, pero también a una gran incertidumbre en cuanto a la calidad, el tiempo y el presupuesto. Los proyectos eran propensos al fracaso porque no se contaban con los marcos necesarios para garantizar que los objetivos del cliente se cumplan.

# Mariana González Calderón



Soy desarrolladora full stack con una gran pasión por la tecnología y un sólido conocimiento en diversas áreas del sector.

Desde que comencé en este mundo, he sido constante en mi aprendizaje y dominio de distintos lenguajes de programación y herramientas, tales como Java, JavaScript, PHP, Kotlin, bases de datos relacionales, desarrollo para Android, entre otros.

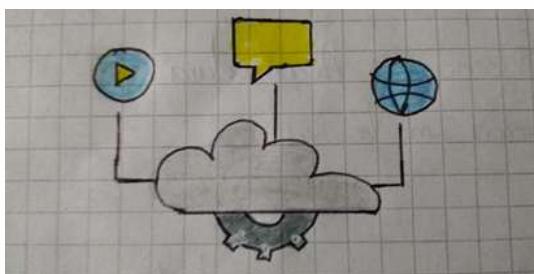
En cada proyecto en el que participo, me enfoco en ofrecer lo mejor de mí para crear aplicaciones eficientes y funcionales. Además, me gusta mantenerme actualizada con los cambios y avances en el campo de la tecnología, para poder aprender e implementar las novedades en mi trabajo diario.

# Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

El artículo propone un entorno de diseño integral para arquitecturas de software orientadas a aplicaciones web, especialmente en contextos de computación en la nube. Este entorno aborda los principales desafíos del diseño arquitectónico, utilizando un metamodelo de componentes arquitectónicos como base. Incluye herramientas gráficas para la creación y verificación de patrones de diseño, asegurando su correcta implementación y promoviendo diseños de calidad. Además, se enfoca en facilitar el trabajo de los arquitectos al proporcionar guías y sugerencias basadas en decisiones arquitectónicas previas. El objetivo es optimizar la productividad y garantizar que las arquitecturas cumplan con los atributos de calidad requeridos.

## Reflexión

El diseño arquitectónico es una disciplina compleja que equilibra requisitos funcionales y no funcionales para crear sistemas robustos y eficientes. La computación en la nube añade nuevos desafíos debido a su naturaleza dinámica y la necesidad de adaptar patrones de diseño existentes. Este artículo destaca la importancia de combinar herramientas prácticas y marcos teóricos sólidos para enfrentar estas dificultades. La propuesta presentada es valiosa porque no solo ayuda a construir arquitecturas eficientes, sino que también fomenta un enfoque sistemático hacia la calidad del software, algo esencial en un entorno tecnológico.

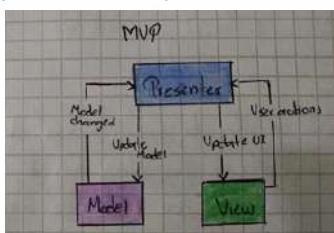


## Bibliografía

Blas et al. (2019)

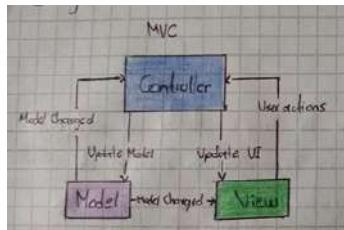
# Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software

El artículo aborda la importancia de la arquitectura de software en el desarrollo de proyectos tecnológicos. Presenta un marco de trabajo para seleccionar patrones arquitectónicos que mejoran la calidad, rendimiento, mantenimiento y adaptabilidad del software. Este marco guía a desarrolladores y arquitectos para tomar decisiones basadas en características del proyecto (como tipo de desarrollo y requisitos clave). Se describen patrones como MVC, MVP, Microservicios y Arquitectura en la Nube, y su idoneidad según el contexto y necesidades del software. Finalmente, se valida el marco mediante un caso práctico en el que el usuario selecciona opciones para obtener recomendaciones arquitectónicas personalizadas.



## Bibliografía

(Marco de Trabajo Para Seleccionar un Patrón - ProQuest, s. f.)

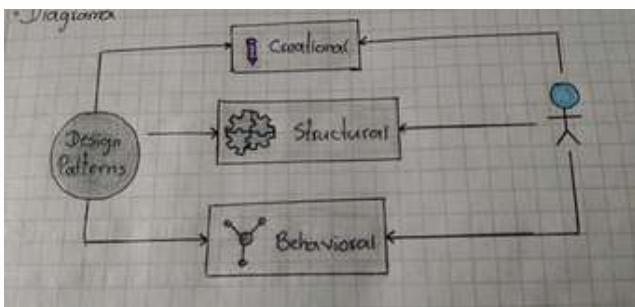


## Reflexión

La arquitectura de software es crucial para el éxito de los proyectos, ya que influye directamente en su escalabilidad, mantenibilidad y rendimiento. Este marco de trabajo no solo organiza el conocimiento sobre patrones arquitectónicos, sino que también empodera a los desarrolladores para tomar decisiones más informadas y eficientes. La propuesta resalta la necesidad de planificar antes de codificar, evitando reprocesos y promoviendo prácticas sostenibles. En un entorno donde las tecnologías avanzan rápidamente, adoptar metodologías como esta es esencial para cumplir con las demandas del mercado y garantizar la calidad del producto final.

# Impact of Design Patterns on Software Maintainability

El artículo examina los patrones de diseño de la "Gang of Four" (GoF), que incluyen 23 patrones clasificados en propósitos (creacionales, estructurales, comportamentales) y alcances (clases u objetos). Se discute su impacto en la mantenibilidad del software a través de estudios empíricos, con resultados inconsistentes. Aunque algunos hallazgos destacan beneficios en ciertos contextos, otros indican que los patrones no mejoran la mantenibilidad o pueden introducir complejidad innecesaria. Además, se proponen herramientas y enfoques para ayudar a los diseñadores a seleccionar patrones adecuados basados en atributos de calidad y tamaño del sistema.



## Reflexión

El uso de patrones de diseño GoF ofrece ventajas como reusabilidad y estructura, pero su impacto en la mantenibilidad varía según el contexto y la experiencia del diseñador. Esto subraya la importancia de comprender profundamente los patrones y evaluar su aplicabilidad en cada caso. Diseñadores deben equilibrar sus decisiones considerando tanto los beneficios como las posibles complejidades que puedan introducir. El uso adecuado de estos patrones puede mejorar la cohesión y reducir el acoplamiento entre los componentes, facilitando la comprensión del sistema a largo plazo.

## Bibliografía

Alghamdi y Qureshi (2014)

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del no Arte

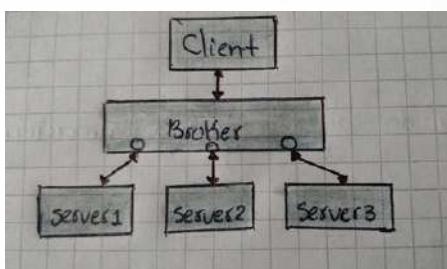
El artículo explora la evolución de la ingeniería de software, enfocándose en los principios fundamentales que han guiado el desarrollo y la maduración del campo. Se examina el estado del arte en los patrones de diseño y lenguajes de patrones, destacando cómo estos conceptos han evolucionado y su aplicación en diversos dominios. La investigación analiza cómo los patrones de diseño han ayudado a estandarizar soluciones para problemas recurrentes en el desarrollo de software, proporcionando marcos y directrices para crear sistemas más eficientes y mantenibles. Además, se discute el impacto de los lenguajes de patrones en la comunicación y documentación de estas soluciones, facilitando su adopción y adaptación en diferentes contextos.

## Bibliografía

(2008). Redalyc

## Reflexión

El artículo ofrece una visión profunda sobre la evolución de la ingeniería de software, subrayando la relevancia de los patrones de diseño y los lenguajes de patrones en la práctica actual. Al abordar cómo estos conceptos han transformado la manera en que los desarrolladores enfrentan problemas recurrentes, se destaca su papel crucial en la creación de soluciones estandarizadas que mejoran la eficiencia y mantenibilidad de los sistemas. La estandarización no solo facilita la implementación de prácticas recomendadas, sino que también fomenta la colaboración y comunicación entre equipos, lo cual es esencial en entornos de desarrollo ágiles y dinámicos.

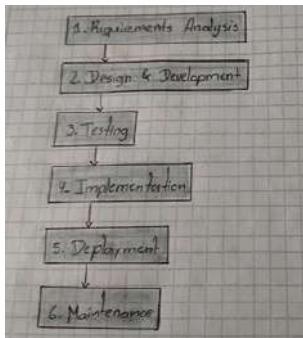


# Buenas prácticas en la construcción de software

La guía presentada ofrece un conjunto de consejos clave para lograr un código limpio y de alta calidad durante el desarrollo de software, destacando la importancia de seguir buenas prácticas y adoptar arquitecturas efectivas. A través de recomendaciones sobre metodologías y estilos arquitectónicos, la guía busca proporcionar directrices que mejoren la legibilidad, mantenibilidad y eficiencia del código. Se abordan aspectos fundamentales como la implementación de principios sólidos de diseño, la elección de arquitecturas adecuadas y el empleo de metodologías ágiles que faciliten un desarrollo más organizado y flexible. Además, se enfatiza la importancia de la consistencia en el estilo de codificación y la aplicación de prácticas que promuevan un desarrollo sostenible y colaborativo.

## Bibliografía

(2021). Tecnología Investigación y Academia.



## Reflexión

La guía sobre la importancia de un código limpio y de alta calidad me invita a reflexionar sobre un aspecto fundamental en el desarrollo de software que a menudo se pasa por alto: la calidad del código no es solo un objetivo técnico, sino una estrategia esencial para el éxito a largo plazo de cualquier proyecto. Al ofrecer consejos sobre buenas prácticas y arquitecturas efectivas, la guía se convierte en un recurso valioso para desarrolladores que buscan mejorar la legibilidad y mantenibilidad de su trabajo. La implementación de principios sólidos de diseño y la elección de metodologías ágiles no solo optimizan el proceso de desarrollo, sino que también fomentan un ambiente de trabajo más colaborativo y eficiente.

# Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua

El proyecto SIGAP tuvo como objetivo implementar una arquitectura basada en microservicios y principios DevOps para mejorar los procesos de desarrollo de software y aumentar la productividad del equipo. La arquitectura de microservicios permitió dividir la aplicación en servicios independientes, lo que facilitó su mantenimiento y escalabilidad. Además, este enfoque permitió realizar entregas de software de manera ágil y controlada, adaptando los flujos de trabajo a un entorno más eficiente. La integración de DevOps promovió la colaboración entre desarrollo y operaciones, automatizando procesos y mejorando la calidad del software. Se logró una mayor flexibilidad para adaptarse a cambios y requerimientos del cliente, lo que permitió ciclos de entrega más rápidos y confiables. El enfoque arquitectónico utilizado no solo mejoró el rendimiento de SIGAP, sino que también demostró ser aplicable a otros dominios.

## Reflexión

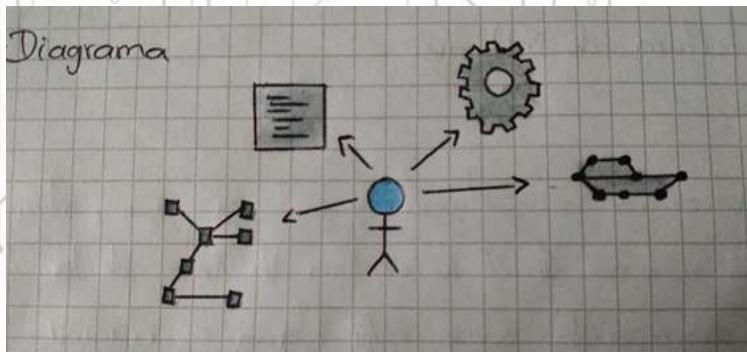
La experiencia del proyecto SIGAP me ha hecho reflexionar sobre la importancia de adoptar una arquitectura de microservicios y los principios de DevOps en el desarrollo de software. Al dividir aplicaciones complejas en servicios independientes, se mejora la escalabilidad y el mantenimiento, lo que permite implementar cambios más rápidamente y con menor riesgo. He visto cómo esta estructura no solo optimiza el rendimiento del software, sino que también fomenta la innovación continua.

Además, he apreciado cómo la integración de prácticas de DevOps transforma la colaboración entre los equipos de desarrollo y operaciones. Este enfoque crea un flujo de trabajo más eficiente y alineado.

## Bibliografía

- (2021). Revista Iberoamericana Ciencia, Tecnología y Sociedad.

# Introducción a la Arquitectura de Software



## Reflexión

El artículo sobre la arquitectura de software me lleva a reflexionar sobre el papel crucial que desempeña en el ciclo de vida del sistema. La fase de arquitectura es fundamental, ya que establece las bases sobre las cuales se construirá el software, asegurando que la estructura y los componentes se alineen con los objetivos del proyecto. Me parece notable cómo cada etapa del ciclo de la arquitectura, desde la planificación hasta la evaluación continua, es interdependiente y contribuye a la evolución del sistema. Este enfoque integrado permite realizar ajustes y mejoras a lo largo del desarrollo, lo que resulta esencial en un entorno en constante cambio.

## Bibliografía

Blancarte, O. (2016). Introducción a los Patrones de Diseño. México, México DF. <https://n9.cl/btd5y>

El artículo aborda la arquitectura de software y destaca su importancia en el ciclo de vida del sistema. En él se explora la fase de la arquitectura del sistema, subrayando cómo esta etapa es crucial para definir la estructura y los componentes del software de manera que se alineen con los objetivos del proyecto. Se analiza el ciclo de la arquitectura, que incluye la planificación, el diseño, la implementación y la evaluación continua, destacando cómo cada fase contribuye a la evolución y adaptación del sistema. Además, el artículo enfatiza cómo el entorno en el que opera el sistema y los requerimientos no funcionales, como el rendimiento y la escalabilidad, influyen significativamente en la definición de la arquitectura.

# Integración de arquitectura de software en el ciclo de vida de las metodologías ágiles

La investigación explora cómo las arquitecturas de software y las metodologías ágiles se interrelacionan y se integran en el desarrollo de software, analizando cómo ambas pueden colaborar para optimizar el proceso de creación. Se enfoca en el alcance de esta integración, destacando la importancia de considerar los "requisitos significativos para la arquitectura". Estos requisitos son cruciales para garantizar que la arquitectura sea flexible y adaptable a los cambios frecuentes que caracterizan a los entornos ágiles. La investigación busca demostrar cómo una arquitectura bien definida puede facilitar la implementación de metodologías ágiles, permitiendo un desarrollo más eficiente y adaptado a las necesidades cambiantes del proyecto.

## Reflexión

La investigación sobre la interrelación entre arquitecturas de software y metodologías ágiles me lleva a reflexionar sobre la importancia de una integración efectiva para optimizar el desarrollo de software. La colaboración entre estos dos aspectos es fundamental, ya que una arquitectura bien diseñada no solo sienta las bases para un desarrollo eficiente, sino que también permite a los equipos adaptarse rápidamente a los cambios y requisitos que surgen en entornos ágiles. Considero esencial prestar atención a los "requisitos significativos para la arquitectura", ya que estos garantizan la flexibilidad necesaria para responder a las dinámicas del proyecto.

## Bibliografía

(2020). Revista de Tecnología y Software, 15(3), 45-60.

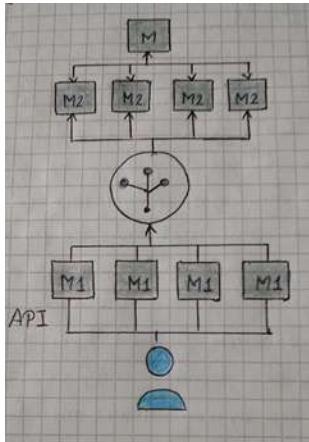


# Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software

El artículo examina las distintas formas de representar arquitecturas de software, subrayando la importancia de descomponer y definir sus elementos esenciales para lograr claridad. Se analizan los métodos más comunes, destacando el uso del lenguaje natural como una herramienta accesible que facilita la comunicación entre los participantes en el desarrollo de software. Sin embargo, se señala que, a pesar de su accesibilidad, el lenguaje natural tiene limitaciones en términos de precisión y estandarización, lo que puede llevar a ambigüedades y malentendidos en aspectos técnicos complejos. Por ello, se recomienda complementarlo con representaciones más formales para permitir un análisis más riguroso.

## Bibliografía

(2019). Revista Cubana de Ciencias Informáticas.

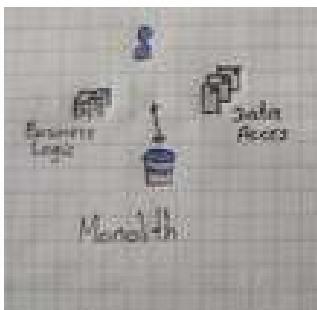


## Reflexión

Al leer el artículo, reflexioné sobre la importancia de representar adecuadamente las arquitecturas de software. Descomponer y definir claramente sus elementos esenciales es vital para el éxito de cualquier proyecto. El lenguaje natural, al ser accesible, facilita la comunicación entre todos los involucrados, permitiendo que tanto desarrolladores como no desarrolladores comprendan la estructura general del sistema. Sin embargo, también es cierto que puede generar ambigüedades que, en un entorno técnico, pueden resultar problemáticas. Por eso, me parece acertado complementar el lenguaje natural con representaciones más formales.

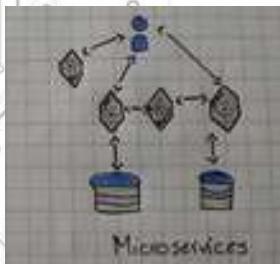
# From Monolithic Systems to Microservices: A Comparative Study of Performance

El artículo aborda la transición desde los sistemas monolíticos hacia los microservicios, explorando las ventajas y desventajas de ambos enfoques y los retos que surgen en este proceso. Se realiza un estudio de caso que compara el desempeño de una aplicación web implementada en una arquitectura monolítica con otra en una arquitectura de microservicios. Los resultados evidencian que los microservicios presentan una mayor eficiencia en términos de consumo de CPU, utilización de memoria y velocidad de escritura en disco. Sin embargo, se identifican ciertas desventajas en los microservicios, como la complejidad de la gestión y una mayor probabilidad de fallos en las comunicaciones.



## Bibliografía

<https://www.proquest.com/docview/2437268966/F05CEE9B819045B5PQ/1?accountid=31491&sourcetype=Scholarly%20Journals>



## Reflexión

El artículo destaca la transición de arquitecturas monolíticas a microservicios, subrayando que, aunque los microservicios ofrecen mejoras en eficiencia y escalabilidad, también introducen desafíos como la complejidad de gestión y fallos en la comunicación. Esto resalta la importancia de evaluar las necesidades específicas de cada proyecto. Mientras que los microservicios son ideales para aplicaciones de alta demanda, la arquitectura monolítica puede ser más adecuada para sistemas menos complejos. La elección debe basarse en el contexto y los objetivos del software, enfatizando que no hay una solución única para todos los casos.

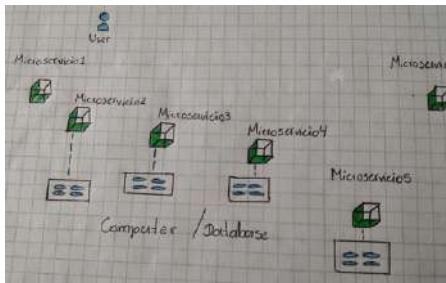
# What are microservices? Your next software architecture

Este artículo explica los microservicios, una arquitectura de software que ha ganado popularidad para crear aplicaciones web modernas. A continuación, se presenta un desglose de los conceptos clave:

- Los microservicios son unidades de código pequeñas e independientes que realizan tareas específicas y se comunican entre sí a través de API.
- Son adecuados para la implementación en la nube.
- Contrastan con las arquitecturas monolíticas.
- Los microservicios son populares en la comunidad Java, con marcos como Spring Boot y Spring Cloud que facilitan su desarrollo.
- Los contenedores, como Docker, permiten una implementación y gestión más eficientes de los microservicios.

## Bibliografía

Blas et al. (2019)

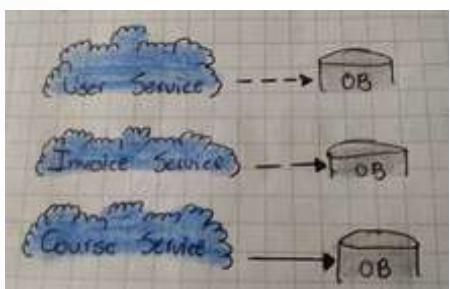


## Reflexión

Este artículo destaca cómo los microservicios han revolucionado el desarrollo de aplicaciones web al permitir un enfoque más modular y flexible. Su capacidad para escalar y adaptarse a entornos en la nube, junto con el uso de tecnologías como Docker, facilita la creación de aplicaciones complejas. La comparación con las arquitecturas monolíticas resalta la necesidad de evolucionar en las prácticas de desarrollo para abordar las demandas actuales del mercado. Este enfoque modular no solo mejora la eficiencia en el desarrollo, sino que también facilita el mantenimiento y la actualización de las aplicaciones sin afectar a todo el sistema. Además, la posibilidad de implementar microservicios de manera independiente permite un ciclo de vida de desarrollo más ágil y dinámico.

# Assessing the Impact of Migration from SOA to Microservices Architecture

El Artículo explora la transición de aplicaciones basadas en SOA (Arquitectura Orientada a Servicios) a microservicios, un estilo arquitectónico adoptado por empresas como Netflix y Twitter. Aunque el enfoque de microservicios presenta ventajas significativas, como la escalabilidad y flexibilidad, también plantea desafíos relacionados con el rendimiento y la complejidad durante la migración. El estudio utiliza métricas como la probabilidad de propagación de cambios y la estabilidad arquitectónica para evaluar el impacto de esta migración. Los resultados muestran que, a pesar de los desafíos, los microservicios ofrecen beneficios sustanciales y son especialmente adecuados para aplicaciones empresariales grandes.



## Bibliografía

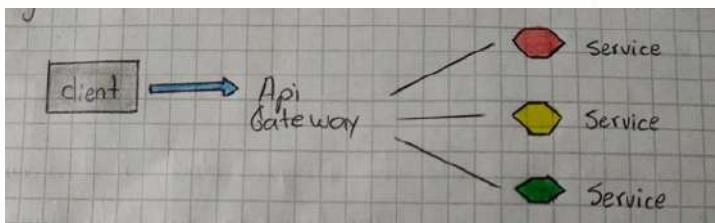
<https://www.proquest.com/docview/2921192054/72CF03D124D94837PQ/1?accountid=31491&sourcetype=Scholarly%20Journals>

## Reflexión

El artículo destaca la evolución de las aplicaciones de SOA a microservicios, un cambio adoptado por líderes como Netflix y Twitter. Aunque los microservicios ofrecen ventajas claras en escalabilidad y flexibilidad, también conllevan retos en rendimiento y complejidad durante la migración. Al evaluar esta transición mediante métricas específicas, los resultados indican que, a pesar de los desafíos, los beneficios de los microservicios son significativos, haciéndolos particularmente adecuados para grandes aplicaciones empresariales. Esta reflexión subraya la importancia de sopesar tanto los pros como los contras al considerar una migración arquitectónica.

# Monolitos vs. Microservicio en Arquitectura de Software: Perspectivas para un Desarrollo Eficiente

Este estudio analiza exhaustivamente dos enfoques clave en el desarrollo de software: los monolitos y los microservicios. Examina sus estructuras, interacciones de componentes, desafíos, ventajas y su implementación práctica en entornos reales. También evalúa casos de estudio de empresas líderes que han adoptado o migrado entre ambas arquitecturas. El análisis incluye el despliegue, mantenimiento y escalabilidad a largo plazo, proporcionando una visión completa para que los profesionales puedan tomar decisiones informadas sobre qué arquitectura utilizar, considerando las implicaciones técnicas, estratégicas y de costos a largo plazo.



## Reflexión

Este estudio ofrece un análisis profundo de dos enfoques en el desarrollo de software: monolitos y microservicios. Al examinar sus estructuras, ventajas y desafíos, se proporciona un contexto práctico mediante casos de estudio de empresas que han migrado entre ambas arquitecturas.

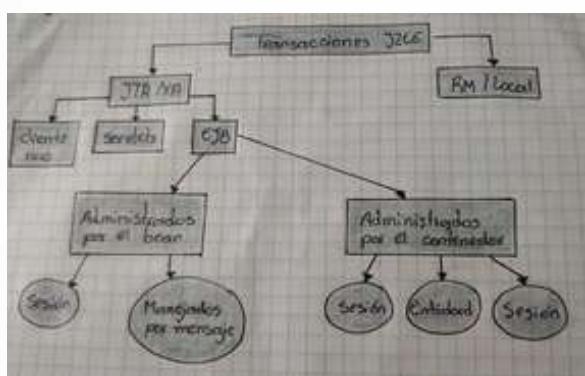
El enfoque en despliegue, mantenimiento y escalabilidad permite a los profesionales evaluar las implicaciones técnicas, estratégicas y de costos a largo plazo. En resumen, este análisis es esencial para que los desarrolladores tomen decisiones informadas y elijan la arquitectura más adecuada según las necesidades de sus proyectos.

## Bibliografía

Colombero et al. (2024)

# Arquitectura de Software y Entornos de Trabajo (Frameworks) de Contenedores Ligeros

El objetivo del trabajo radica explorar con respecto a las arquitecturas de software J2EE y los contenedores ligeros, dedicando especial atención a las características de los POJOs (Plain Old Java Objects). Pese a que no serían considerados como una nueva tecnología, los POJOs llegarían a ser considerados como una filosofía que permite a los desarrolladores que trabajen en la lógica de negocio, a la que determinan que ni la infraestructura ni el framework influyan en la determinación del diseño. Por otro lado, se establece como importante el garantizar que el dominio se mantenga aislado del resto del sistema.



## Reflexión

Este trabajo pone de manifiesto la manera en la que las modernas arquitecturas de software están comenzando a adoptar así un menor intrusismo y mayor flexibilidad de manera que los desarrolladores puedan centrarse en lo importante, la lógica de negocio. La filosofía de los POJOs pone de manifiesto una tendencia que va en aumento de querer reducir la complejidad de los OO frameworks y tecnologías más pesados para poder dejar al dominio de la aplicación algo aislado y desacoplado, aspecto este que ayudara a conservar la mantenibilidad del sistema o su escalabilidad.

## Bibliografía

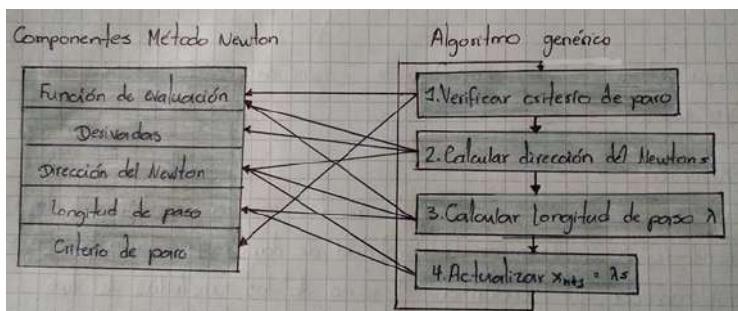
Tesis\_Damian\_Ciocca.pdf

# Arquitectura de software flexible y genérica para métodos del tipo Newton

La tesis aborda el análisis y diseño de una arquitectura de tipo orientado a objetos con la finalidad de dar respuesta a la problemática que suscita el reuso de los métodos del tipo Newton, correspondiendo dicha arquitectura a la abstrae de las principales características de los mismos métodos mediante la aplicación de patrones de tipo arquitectura y de diseño de la misma arquitectura. Dicha arquitectura está pensada para su uso en diversos problemas no lineales que juegan en calidad de interfaz genérica entre los distintos métodos del tipo Newton. En el documento se exponen las razones que sustentan las decisiones de diseño y representación de las dificultades que se presentan cuando se debe construir una arquitectura del tipo orientado a objetos.

## Reflexión

Esta tesis desarrolla un reto habitual en la elaboración de software: la reutilización del código. En concreto, se preocupa de cómo conseguir mejorar la flexibilidad y la extensibilidad de los métodos numéricos, como por ejemplo los métodos de Newton, mediante una arquitectura orientada a objetos. Con la adopción de los patrones de diseño y de la arquitectura, es posible conseguir una estructura que responde la situación existente, y que al mismo tiempo permite integrar nuevos métodos o problemas, y no hace necesario reestructurar el sistema completo. Un aspecto determinante de esta solución consiste en la creación de una interfaz genérica que abstraiga los detalles relacionados con cada uno de los métodos de Newton.



## Bibliografía

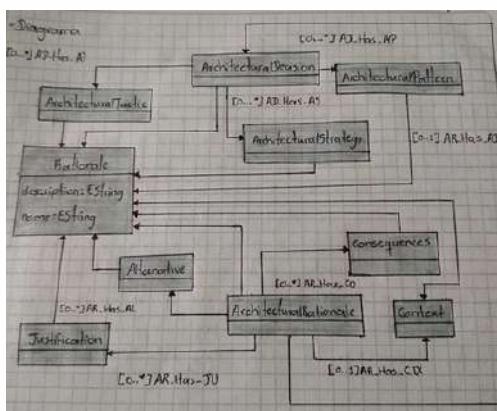
Item 1009/730 | Repositorio INAOE

# Un lenguaje de modelado para representar visualmente las decisiones de diseño arquitectónico y su rationale: Rationale

El trabajo explica qué significa documentar el rationale arquitectónico en el diseño de sistemas de software. Se hace especial hincapié en el trabajo ágil que a menudo ignora ese tipo de documentación. El rationale arquitectónico es el conjunto de razones que guían las decisiones adoptadas durante el diseño arquitectónico, y que careciendo de los niveles de documentación adecuados puede tener efectos negativos en la mantenibilidad del software a través del tiempo. A menudo el rationale queda en la cabeza de los creadores del sistema y eso, por tanto, puede hacer más difíciles las decisiones en los estados más adelante de la evolución de la aplicación.

## Reflexión

La documentación del rationale arquitectónico se convierte en algo crítico en su desarrollo, en particular cuando se están empleando paradigmas ágiles donde la práctica de la documentación más exhaustiva se antepone a una flexibilidad y rapidez del trabajo a realizar. El rationale arquitectónico actúa como una base compositiva que permite conocer las decisiones que han sido tomadas en el diseño, y que si están bien documentadas ayudan a que el sistema se mantenga a largo plazo y siga siendo sostenible. El presente trabajo muestra un reto habitual en contextos ágiles.



## Bibliografía

Un lenguaje de modelado para representar visualmente las decisiones de diseño arquitectónico y su rationale | Informador Técnico

# Componentes MDA para patrones de diseño

La Arquitectura de Modelos (MDA), provista en el modelo del Object Management Group (OMG), es una concepción para abordar el desarrollo del software que aboga por elevar el nivel de abstracción del desarrollo de sistemas complejos. Su esencia principal consiste en desacoplar la especificación funcional de dicho sistema de su implementación en una plataforma determinada. La MDA aboga por el uso de modelos y transformaciones de modelos a lo largo del proceso de desarrollo.

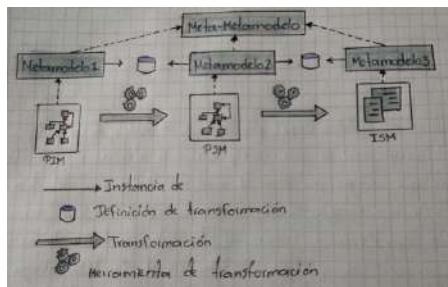
Los cuatro tipos de modelos son:

Modelo Independiente de la Computación (CIM)

Modelo Independiente de la Plataforma (PIM)

Modelo Específico a la Plataforma (PSM)

Modelo Específico a la Implementación (ISM)



## Reflexión

La propuesta de la Arquitectura Model-Driven MDA es una parte significativa de la automatización del desarrollo de software, así como de la mejora de la productividad en el desarrollo de software. Al separar la funcionalidad del sistema de su implementación tecnológica, MDA permite a los desarrolladores centrarse en el diseño de soluciones de alto nivel. La separación de preocupaciones no solo mejora la claridad, además de la mantenibilidad del sistema, sino que también facilita su adaptación a diferentes plataformas y distintas tipologías de entornos a lo largo del tiempo.

## Bibliografía

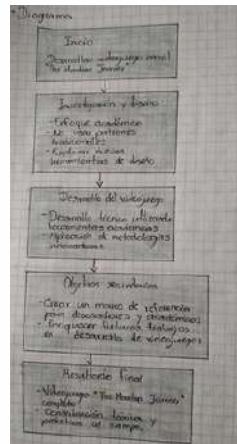
Moreno, J. C., Marciszack, M. M., & Groppo, M. A. (2020). Patrones de Usabilidad Temprana en el Modelo Conceptual. AJEA, (5). <https://n9.cl/ewi5om>

# Herramientas de diseño : exploración y construcción de caminos que aporten a los procesos de desarrollo de videojuegos

El presente documento expone el proceso investigativo, el diseño y el desarrollo del videojuego móvil *The Martian Journey*, un proyecto innovador en el desarrollo de videojuegos. En lugar apoyarse en patrones y modelos de diseño tradicionales del sector doméstico, el proyecto se sustentó en la teoría así como en la experiencia académica acumulada en la Maestría en Diseño de la Universidad de los Andes. El principal objetivo fue experimentar nuevas formas de apoyar los marcos de trabajo y los procesos existentes en el sector del entretenimiento en relación con herramientas diseñadas. Adicionalmente el proyecto se orientó a proporcionar un marco de referencia útil tanto para diseñadores y desarrolladores de videojuegos como para académicos e interesados con el fin de enriquecer sus futuros trabajos y proyectos.

## Bibliografía

Herramientas de diseño : exploración y construcción de caminos que aporten a los procesos de desarrollo de videojuegos

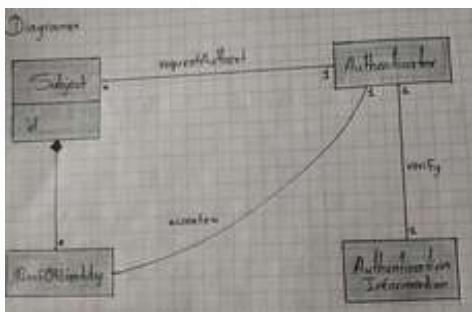


## Reflexión

La inducción de *The Martian Journey* es interesante por la forma en que concibe a los videojuegos como algo más que una forma de entretenimiento. Aparte de ello, el estudio de los videojuegos puede representar una forma en la que las herramientas de diseño y las teorías académicas sean fundamentales a la hora de crear experiencias interactivas o de tipo videojuegos, de algún modo, además, el hecho de que este proyecto haya escapar a los estándares del diseño de videojuegos en la industria, decantándose por un diseño de videojuegos fundamentado en la teoría o experiencia académica, de alguna manera muestra una tentativa de introducir tendencias actuales en el proceso de diseño de videojuegos.

# Relación entre patrones de seguridad Core Security Patterns (CSP) y Security Patterns Practice (SPP)

En la actualidad, la seguridad de las aplicaciones empresariales ha cobrado una especial relevancia debido al aumento de la amenaza que comportan los ataques maliciosos, lo cual ha llevado a que se utilicen catálogos de patrones de seguridad, que recogen buenas prácticas e implementaciones de la seguridad para aquellos sistemas a proteger. En este trabajo de investigación se realiza un análisis de dos de estos catálogos de patrones de seguridad: Core Security Patterns (CSP) y Security Patterns in Practice (SPP) que abordan una serie de patrones de seguridad para los desarrolladores de los sistemas que incrementan la seguridad de las aplicaciones empresariales.



## Bibliografía

Relación entre patrones de seguridad Core Security Patterns (CSP) y Security Patterns Practice (SPP)

## Reflexión

La seguridad en el desarrollo del software no es un enfoque trivial ni tampoco es una labor trivial. Las amenazas son cambiantes y obliga a los desarrolladores utilizar patrones de seguridad sobre los que pueden adelantarse y a partir de los cuales pueden realizar aplicaciones más extensibles.

Los catálogos sobre patrones de seguridad como CSP y SPP pasan a ser recursos imprescindibles de los equipos de desarrollo ya que aportan soluciones.

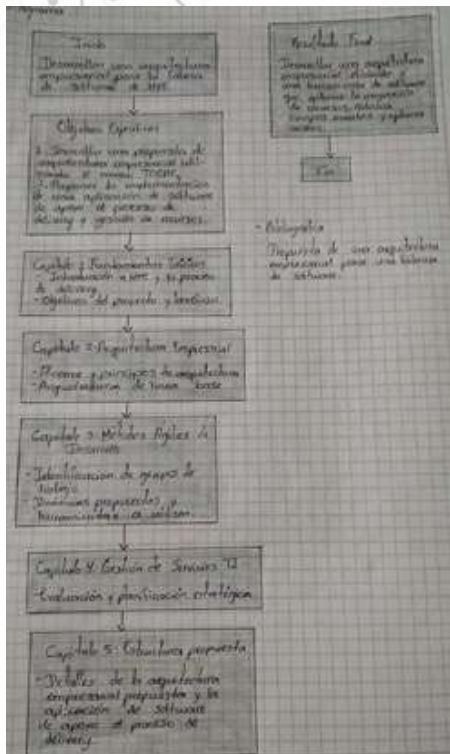
# Propuesta de una arquitectura empresarial para una fábrica de software

El presente proyecto tiene como fin el diseño de una arquitectura empresarial para la fábrica de software de Hewlett Packard Enterprise (HPE) con el firme propósito de resolver el principal problema que en ella se presenta en lo que se refiere a los procesos de entrega de servicios, que es la mala asignación de recursos humanos a los proyectos de software que se ofrecen a los clientes, problema que incide en el proceso de delivery y por lo tanto en la eficiencia y calidad del servicio. Para ello, el objetivo principal de este proyecto es poder llevar a cabo el diseño de una arquitectura empresarial que garantice la optimización.

## Reflexión

El enfoque propuesto a lo largo del presente proyecto tiene una fuerte implicancia sobre la gestión de recursos humanos en una fábrica de software, algo esencial en la industria del suministro de software, donde el capital humano se considera, en muchas ocasiones, uno de los recursos más importantes de la misma. Una mala asignación de recursos puede ya ser en sí misma un motivo de retraso, sobrecostes o baja calidad del producto final, lo que necesariamente repercute sobre el cliente final y la propia empresa, ya que afecta tanto a su imagen como a su cuenta de resultados.

Siguiendo el marco de trabajo TOGAF, que es conocido por su enfoque estructurado y exhaustivo en la creación de arquitecturas empresariales.

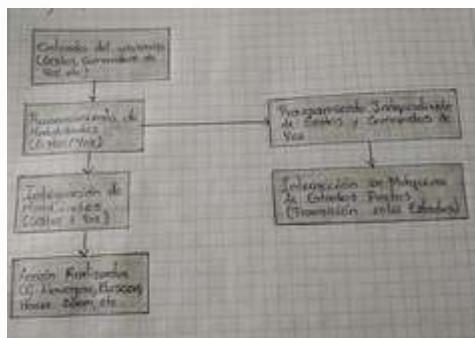


## Bibliografía

Propuesta de una arquitectura empresarial para una fábrica de software

# Arquitectura de interacción multimodal aplicado a navegación en mapas

El objetivo de este proyecto es diseñar una arquitectura que permita desarrollar interfaces de interacción multimodal. A través de ella, modalidades de diálogo distintas, como los gestos y los comandos de voz, son interpretadas y procesadas de manera simultánea, para luego unirse e interpretarse, fusionadas, para presentar una sola experiencia de usuario coherentemente integrada. Esta modalidad de interacción se modela usando una FSM en la que se muestran todos los movimientos de los usuarios como transiciones de estado. Este enfoque facilita la organización y agrupación de movimientos multimodales y los prepara para su manejo.



## Bibliografía

Arquitectura de interacción multimodal aplicado a navegación en mapas

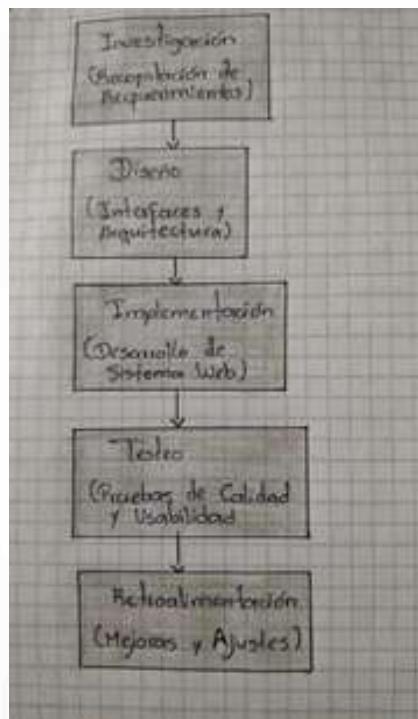
## Reflexión

La arquitectura multimodal para el desarrollo de interfaces de usuario es un paso significativo hacia la creación de experiencias más accesibles. Las interfaces multimodales no solo mejoran la interacción con dispositivos, sino que también permiten que los usuarios elijan el modo de interacción más fácil para ellos, como gestos, voz o ambos.

El uso de máquinas de estados finitos (FSM) para modelar la interacción del usuario es una estrategia eficaz, porque permite organizar las transiciones de las diferentes acciones en un flujo controlado. Este enfoque facilita el manejo de interacciones complejas.

# **Desarrollo de un sistema web para la gestión de historias clínicas y asistencia telemática mediante un chatbot utilizando la metodología Lean Thinking Developer en la clínica veterinaria "Mundo Mascotas"**

Este proyecto busca implementar un sistema web para gestionar historias clínicas y ofrecer asistencia telemática en la clínica veterinaria "Mundo Mascotas". Para ello, se empleó la metodología Lean Thinking Developer, que se centra en mejorar continuamente y entregar rápidamente (similar a la metodología scrum). Esta metodología se dividió en cuatro fases clave:



## **Reflexión**

Finalmente, el desarrollo de MediVen destaca la importancia de las metodologías ágiles, como en este caso Lean Thinking Developer, en el campo del software. Esta metodología no solo se trata de cómo crear un producto funcional rápidamente, sino de mejorar constantemente con la ayuda de la retroalimentación. La fase de prueba, durante la cual los usuarios finales validan el sistema creado, es opcional pero crucial en proyectos ágiles. Demuestra que el producto final será el uno que realmente satisface sus necesidades nativas y que también el productor trabajará de manera intuitiva y estable.

## **Bibliografía**

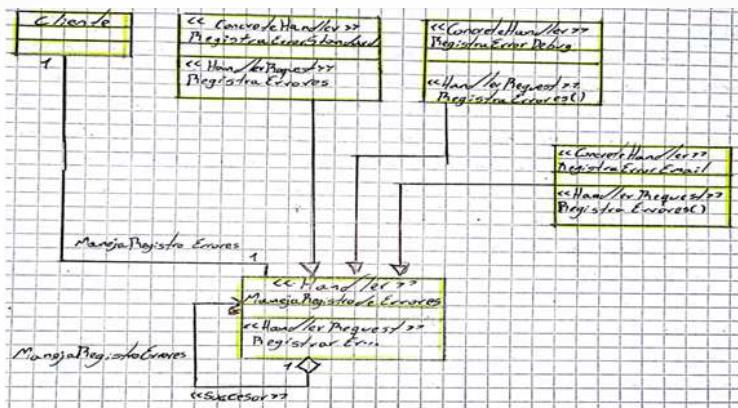
Desarrollo de un sistema web para la gestión de historias clínicas y asistencia telemática mediante un chatbot utilizando la metodología Lean Thinking Developer en la clínica veterinaria "Mundo Mascotas" del Cantón Guaranda

# Una Propuesta de Implementación para Especificaciones de Patrones de Comportamiento

El artículo nos presenta una propuesta para implementar especificaciones de patrones de comportamiento (PDC) utilizando perfiles UML y restricciones OCL. En el proceso se definen 3 niveles de perfiles UML para representar patrones de diseño. Eso permite modular tantos aspectos, tanto estructurales como dinámicos, eso lo aplica diagrama de clase y secuencia en una herramienta de software. Eso permite a los desarrolladores reutilizar soluciones de diseño, lo cual hace todo más fácil en proyectos complejos, permitiendo un mejor desarrollo.

## Reflexión

Nos muestra la importancia de estructurar patrones de comportamiento en el desarrollo de software, más cuando se manejan sistemas complejos. La incorporación de UML y restricciones OCL, facilita a los desarrolladores el mantenimiento y las actualizaciones del sistema. Lo cual permite la reutilización, sino también la mejora en la colaboración y claridad del diseño. Por último, el artículo nos invita a reflexionar sobre las especificaciones y las herramientas adecuadas en la creación de arquitectura adaptable y escalables.



## Bibliografía

Cortez, A. A., & Naveda, C. A. Una propuesta de implementación para especificaciones de patrones de comportamiento. <https://n9.cl/x6wz7>

# Estudio sobre el diseño estructurado y modular de video juegos utilizando Flash

El objetivo principal del proyecto es estudiar y analizar la metodología en el desarrollo, diseño e implementación de videojuegos utilizando Flash. Para ello, se han definido varios objetivos secundarios que abarcan aspectos clave de la programación orientada a objetos, notación UML, el lenguaje ActionScript y los patrones de diseño aplicables al desarrollo de videojuegos. Además, se identifican los elementos fundamentales en Flash; la cual fue una plataforma multimedia ampliamente utilizada para la creación de aplicaciones interactivas, animaciones, videojuegos y contenido web, por ende facilitan la estructuración y modularización del diseño de los videojuegos.

## Reflexión

Este proyecto mostró la importancia de comprender y aplicar una combinación de conceptos de programación, herramientas de diseño y metodologías de desarrollo. Cualquier proyecto de software, incluidos los videojuegos, requiere POO para organizar el código de una manera escalable y reutilizable. Del mismo modo, UML desempeña un papel fundamental al brindarles a los desarrolladores un lenguaje estandarizado de diseño con el que planificar y permitir de la mejor manera posible que las ideas se expresen en estructuras funcionales antes de la implementación.



## Bibliografía

[Estudio sobre el diseño estructurado y modular de video juegos utilizando Flash](#)

# Plataforma de Gestión de Seguridad Informática. Módulo Pruebas de Intrusión

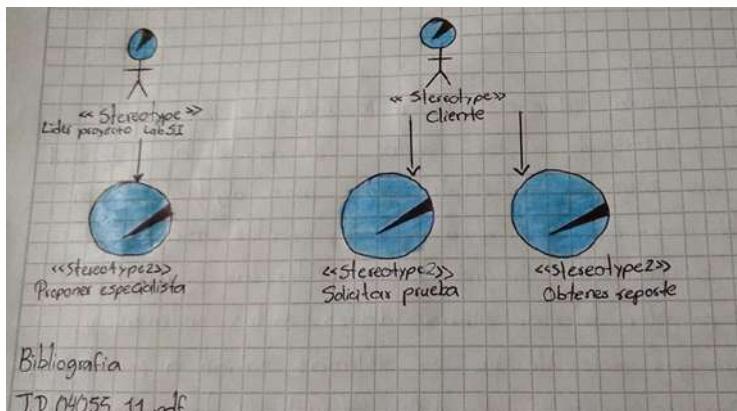
Este proyecto habla sobre el desarrollo de una aplicación para la gestión de pruebas de intrusión en los sistemas de la Universidad antes de ser entregados al cliente y tiene como objetivo principal mejorar la seguridad informática de los proyectos, asegurando que sean confidenciales, íntegros y disponibles. Las pruebas de intrusión permiten detectar vulnerabilidades en los sistemas y reducir riesgos antes de que los sistemas salgan a producción.

La aplicación se desarrolló con el fin de gestionar todo el proceso de pruebas de intrusión que se llevan a cabo en el Laboratorio de Seguridad Informática (LabSI).

## Reflexión

La seguridad informática es un área fundamental para proteger la confidencialidad, la integridad y la disponibilidad de la información, especialmente en sistemas que manejan datos delicados. Las pruebas de intrusión son una herramienta clave para detectar vulnerabilidades antes de que los sistemas se encuentren en producción. Este tipo de pruebas simulan posibles ataques y ayudan a identificar puntos débiles que podrían ser explotados por atacantes maliciosos.

El desarrollo de una aplicación para gestionar las pruebas de intrusión mejora significativamente la eficiencia.



## Bibliografía

[TD 04055 11.pdf](#)

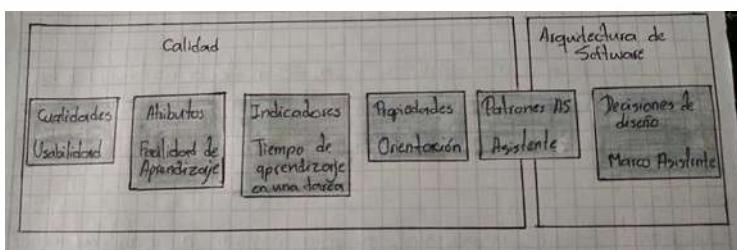
# **Procedimiento para el aseguramiento y evaluación de la usabilidad basado en patrones arquitectónicamente sensibles para los sistemas de gestión del Centro de Informatización de la Gestión de Entidades.**

Este proyecto describe un procedimiento para mejorar la usabilidad de los sistemas de gestión utilizados por el Centro de Informatización de la Gestión de Entidades (CEIGE). La metodología está basada en tres fases, tomadas de la Ingeniería de Usabilidad, que se alinean con el modelo de desarrollo del CEIGE.

Tiene como propósito identificar y dar solución a problemas de usabilidad en las primeras etapas del desarrollo de los sistemas, cuando es más fácil y barato corregirlos. Este procedimiento propone detectar los errores desde el principio, lo que permite hacer mejoras antes de que el sistema salga a producción.

## **Reflexión**

La usabilidad es uno de los aspectos más importantes de cualquier sistema informático, especialmente cuando se trata de plataformas usadas por muchos usuarios, como en el caso de los sistemas de gestión. La facilidad con la que los usuarios interactúan con el sistema no solo afecta la experiencia del usuario, sino también la eficiencia y productividad del trabajo diario. Por ello, detectar problemas de usabilidad en las primeras fases del desarrollo es clave para evitar que se conviertan en grandes obstáculos una vez que el sistema esté en producción.



## **Bibliografía**

[Repositorio Digital:Procedimiento para el aseguramiento y evaluación de la usabilidad basado en patrones arquitectónicamente sensibles para los sistemas de gestión del Centro de Informatización de la Gestión de Entidades.](#)

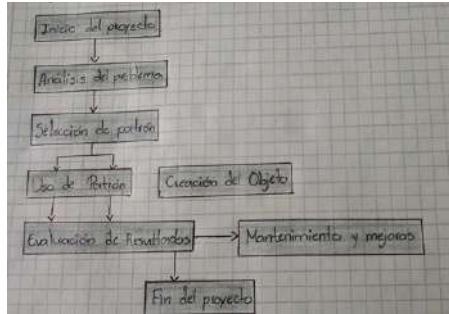
# Construyendo software innovador implementando patrones de diseño creacionales

La programación orientada a objetos (POO) es un enfoque ampliamente utilizado en el desarrollo de software que organiza el código en objetos, los cuales modelan entidades del mundo real. Esta técnica permite crear programas más fáciles de mantener y expandir a medida que el software crece.

Con el tiempo, los programadores enfrentaron problemas comunes durante el desarrollo de aplicaciones, por lo que se crearon patrones de diseño, que son soluciones reutilizables para problemas recurrentes en la programación. Los patrones de diseño, se empezaron a documentar en los años 90 y se han vuelto esenciales en la programación moderna.

## Bibliografía

2023-12-06 construyendo software innovador.pdf

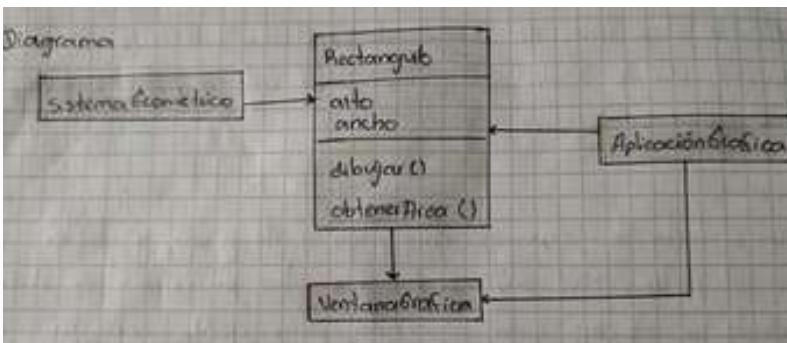


## Reflexión

La programación orientada a objetos es una de las mejores prácticas que ha permitido transformar el desarrollo de software en algo más estructurado y organizado. Gracias a este enfoque, los programadores pueden representar conceptos del mundo real de una manera mucho más natural y comprensible, lo que facilita tanto la creación como el mantenimiento de las aplicaciones. Por otro lado, los patrones de diseño son como "recetas probadas" que nos permiten resolver problemas comunes de manera eficiente, evitando reinventarlo.

# **Principios y patrones de diseño de software en torno al patrón compuesto modelo vista controlador para una arquitectura de aplicaciones interactivas.**

La tesis trata de exponer conceptos y de guiar al lector sobre el patrón MVC así como sobre los principios de diseño orientado a objetos. Está enfocada para gente con un conocimiento básico de desarrollo orientado a objetos y conocimiento UML. La tesis quiere hacer un hincapié claro de la importancia de una comunicación clara así como la de manejar la terminología técnica en el desarrollo de software que debe ser para conseguir un diseño flexible, mantenible y reutilizable. La tesis no es generadora de nuevos principios o patrones pero se espera que sirva la forma de enseñar.



## **Reflexión**

La presente tesis es un exponente de una metodología pedagógica en el desarrollo de software al priorizar la claridad en la exposición de conceptos así como en la comunicación técnica, no para proponer innovaciones, sino para consolidar el conocimiento existente sobre la técnica MVC y principios de diseño orientado a objetos, destacando, además, la importancia de una comunicación clara y precisa, así como de un vocabulario técnico común, subrayando un aspecto clave en la colaboración y el aprendizaje en este campo: la comprensión mutua. Esta tesis no solo trata de entender, sino también de enseñar y comunicar.

## **Bibliografía**

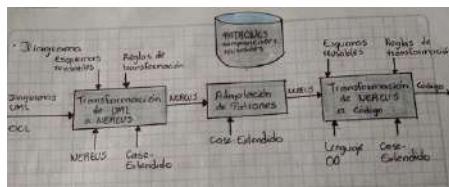
<https://acortar.link/nrL2n7>

# Una creación de patrones de diseño en procesos de ingeniería forward de modelos estáticos UML

los patrones de diseño ofrecen respuestas a problemas recurrentes existentes en el diseño de software, aunque no hay consenso ni acuerdo sobre la forma que deben adoptar para ser aplicados(si deben ser aplicados mediante un determinado lenguaje, plataforma o herramienta, etc), sí hay acuerdo en que su aplicación debería ser automática o en las óptima atención incorporando técnicas para obtener un buen rendimiento, porque su aplicación manual conlleva mucho tedium y una cierta propensión a errores. La industria ha aceptado la creencia de que los patrones de diseño automatizados o asistidos mejoran la comunicación y permiten un desarrollo más rápido.

## Bibliografía

<https://sedici.unlp.edu.ar/handle/10915/21285>



## Reflexión

Los patrones de diseño son cruciales en el desarrollo de forma material, pues ofrecen soluciones consolidadas para problemas recurrentes. Pero bien es cierto que, aunque su valor se ha sabido, los desacuerdos sobre su práctica, es decir, hoy si debería ser gracias a herramientas, lenguajes concretos o plataformas, son un inconveniente. Lo que está claro es que su práctica manual es lenta y propensa a errores. Por lo tanto, automatizar o asistir su uso es, por regla, hoy 1 de los elementos claves para explotar sus ventajas, mejor comunicación entre los desarrolladores y tiempos de desarrollo más cortos.

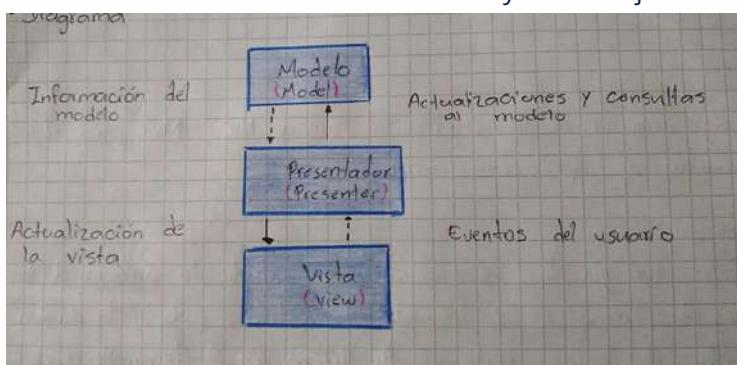
# Análisis comparativo de patrones de diseño MVC y MVP para el rendimiento aplicaciones web.

En este trabajo de investigación, se estudió la eficacia de los patrones de diseño en el desarrollo de software web y la elección adecuada a estos patrones. Para ello se encontraron varios patrones, se evaluaron los resultados obtenidos y se seleccionaron los dos siguientes: MVC y MVP. Cada 1 de ellos se aplicó a los proyectos de software web y se evaluó utilizando los siguientes criterios: Tiempo de desarrollo, líneas de código, uso de la memoria RAM, uso de la CPU y tiempo de respuesta.

Nuestros resultados muestran que el patrón MVC es significativamente más eficaz que el patrón MVP.

## Reflexión

El trabajo explica la importancia de realizar una correcta elección de los patrones de diseño del software para aplicaciones web. La comparación que se establece entre MVC y MVP se explica en el sentido de que ambos patrones de diseño persiguen de alguna manera la mejora y la organización del código, aunque el grado de pertinencia de uno y otro patrón dependerá del contexto de aplicación y de los objetivos de cada proyecto de software. Lee evidente superioridad de MVC que ha logrado cosechar en esta investigación sugiere, de cierta manera, que no todos los patrones son apropiados para todos los proyectos y que, por lo tanto, es importante estudiar y analizar sus beneficios y desventajas.



## Bibliografía

<https://repositorio.uss.edu.pe/handle/20.500.12802/11182>

# Un patrón de software en seguridad a la arquitectura de un microservicio

En este artículo se han analizado los riesgos de seguridad en la arquitectura de microservicios, que se ha vuelto más popular por su concepto moderno el desarrollo de software, que se centra en la modularidad y la escalabilidad. A pesar de los beneficios aportados por los microservicios como la entrega continua y el cambio en las tecnologías, la estructura descentralizada aumenta el riesgo de ser atacado, debido al mayor número de puntos vulnerables. La principal conclusión de la investigación es que es especialmente crítico optar un enfoque integral para la seguridad de los microservicios y tomar medidas que protejan los datos y el sistema.



## Bibliografía

<http://51.143.95.221/handle/TecNM/6574>

## Reflexión

Esta se hace pensar en el equilibrio que es necesario mantener entre innovación y seguridad llevando el desarrollo del software moderno. Si bien la creación de las aplicaciones de esta forma se ha beneficiado de la adopción de microservicios ya que mejoran la escalabilidad, flexibilidad y otros aspectos. Al mismo tiempo, hoy es un recordatorio de que traslapar el sistema en piezas más pequeñas y separadas conlleva un riesgo de seguridad diferente o generalmente mayor. La idea clave es que no debemos quedarnos atrás en términos de avances tecnológicos, si no de cuidar la seguridad. Además, deben avanzar juntos.

# Directorio de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia.

Un sistema RFID, el cual permite la identificación mediante etiquetas, ayuda a realizar un control de inventario actualizando la información sobre la cantidad y localización de bienes. Esta tesis estableció una arquitectura de software para desarrollar una aplicación móvil que forme parte de un sistema RFID, el cual respalda el inventario el Instituto tecnológico de Orizaba y también se presentan las aplicaciones generadas a partir de la arquitectura de software propuesta. Esta solución busca optimizar la gestión de recursos y reducir errores en el seguimiento de activos, facilitando una administración más eficiente el inventario.



## Reflexión

La integración de datos RFID como tecnología en la gestión del inventario está en contradicción con la forma en que la tecnología puede en último extremo cambiar procesos corrientes, un método que puede facilitar un control más efectivo y preciso de los recursos. Esta mejora no se reduce a una utilización óptima de los bienes, también puede disminuir las dificultades humanas y los tiempos de las operaciones, en la medida que las instituciones educativas y empresariales buscan la máxima eficacia. En otras palabras la tecnología debe adaptarse a las necesidades específicas para impulsar el cambio.

## Bibliografía

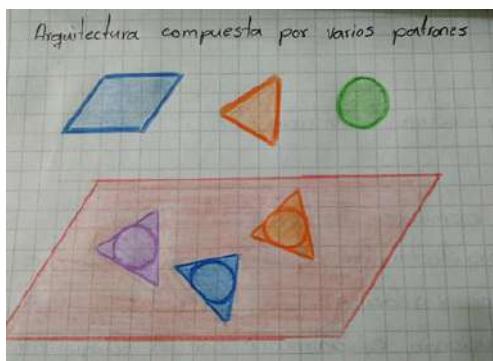
<https://www.redalyc.org/pdf/5122/512251501004.pdf>

# Especificación de la arquitectura de software

En la presente lectura, se examinarán los componentes básicos y fundamentales de una arquitectura de software y cómo los patrones arquitecturales estructuran y relacionan esos elementos. A medida que las tecnologías evolucionan, especialmente con el surgimiento de las plataformas web y móviles, las necesidades de infraestructura de diseño cambian; cada vez se impone mayores soluciones mucho más rápidas en términos de tiempos de entrega y procesamiento de datos y más seguras en términos de transacciones. El artículo trata sobre los elementos importantes de una arquitectura de software y su contexto interpretativo desde la perspectiva de la actualidad.

## Reflexión

Esta lectura no se enfatiza lo importante que es tener una buena base en el desarrollo de software, más ahora que las aplicaciones web y móviles están creciendo tanto. A medida que la tecnología avanza, hoy el software debe ser cada vez más rápido, seguro y capaz de manejar grandes cantidades de información sin perder eficiencia. Y los patrones de diseño son como una guía que ayuda a organizar todos los componentes del sistema de forma clara y coherente, permitiendo que el software funcione bien, se adapte y mejore con el tiempo.



## Bibliografía

<https://revistas.udistrital.edu.co/index.php/tia/article/view/18076>

# Arquitectura de un módulo I/O para objetos 3D

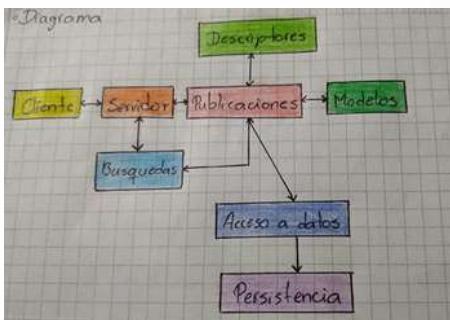
La presente investigación tiene por objeto el problema de la persistencia de los objetos 3D, el cual es una gestión de grandes volúmenes de datos en memoria y disco, así como de procedimientos en la comparación de objetos 3D, los cuales están basados en el teorema matemático que le sirve de base, un teorema que supone un alto coste computacional. Para ello era necesario realizar un exhaustivo análisis de requisitos, de diseño arquitectónico y de investigación de la gráfica por ordenador, y de arquitecturas de software. La solución que se propone responde a un enfoque integral, el cual intenta combinar teoría y práctica para garantizar su efectividad. El producto resultante supone una arquitectura independiente que asegura la funcionalidad requerida, mientras que el prototipo se ha desarrollado dentro de la arquitectura VITRAL, del grupo TAKINA, como módulo de la capa de servicios.

## Bibliografía

<https://core.ac.uk/download/pdf/71418995.pdf>

## Reflexión

El trabajo destaca por el hecho de que se vive junto a retos técnicos importantes en el manejo de los objetos 3D. Lo anterior confirma la importancia del diseño arquitectónico cuidado, de una planificación metódica para abordar la problemática, y la incorporación del prototipo a una arquitectura cooperativa como VITRAL pone de manifiesto la importancia del trabajo modular y de la investigación aplicada, es una interesante y real mejora en la construcción de herramientas para el manejo gráfico, y también es un reconocimiento de que hay que tener presentes las soluciones escalables que podemos esperar para responder, por ejemplo, a problemas de almacenamiento y comparación de sistemas complejos.



# **Arquitectura desoftware para la construcción de un sistema de cuadro de mando integral como herramienta de inteligencia de negocios.**

Cuando las organizaciones han conseguido asumir diversas herramientas y metodologías de toma de decisiones, almacenan información errónea. La Inteligencia de Negocios (IB) actúa ante esta problemática, yuxtaponiendo una serie de herramientas que extraen, depuran y analizan datos, siendo apta para organizaciones de cualquier tipo o tamaño. Dentro de estas herramientas destaca el Cuadro de Mando Integral (CMI) por permitir la posibilidad de un seguimiento constante de la estrategia y la mejora de los procedimientos de gestión decisiva; por ello es necesario construir un modelo de CMI basándose en técnicas o metodologías actuales que de forma sistemática incorporen las buenas prácticas en el desarrollo del software y, por lo tanto, los sistemas y procedimientos organizacionales.

## **Bibliografía**

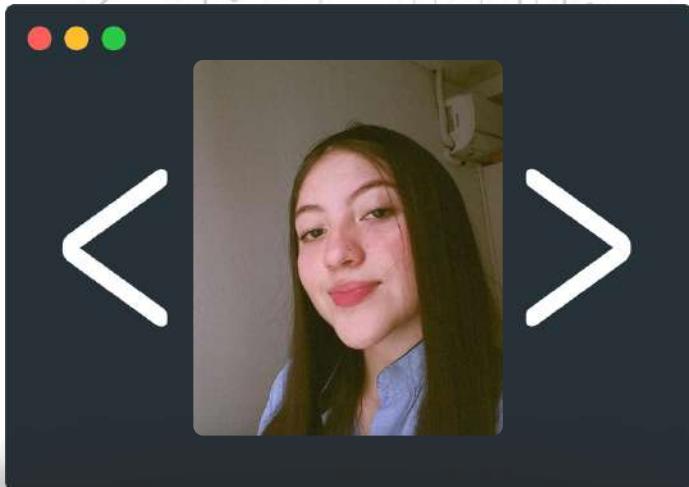
(Vista de Arquitectura de Software Para la Construcción de un Sistema de Cuadro de Mando Integral Como Herramienta de Inteligencia de Negocios, s. f.-b)

## **Reflexión**

El escrito destaca cómo las herramientas de Inteligencia de Negocios, como el CMI, pueden transformar datos en decisiones clave para la estrategia de una empresa. Al aplicar buenas prácticas de desarrollo de software, las organizaciones no solo optimizan sus procesos, sino que también ganan en precisión y flexibilidad frente a un mercado cada vez más competitivo. Es un recordatorio de que la combinación de tecnología y estrategia es fundamental para tomar decisiones más informadas y acertadas. Además, al integrar estas herramientas, las empresas logran ser más ágiles, transparentes y orientadas a resultados. En resumen, los datos se convierten en un activo invaluable para cualquier organización que quiera crecer.



# Maydy Viviana Conde Ladino



¡Hola! Me llamo Maydy Viviana Conde Ladino , tengo 18 años, Soy estudiante del Tecnólogo en Análisis y Desarrollo de Software, con un enfoque en la creación y mantenimiento de soluciones tecnológicas. A lo largo de mi formación, he desarrollado habilidades técnicas en programación, destacando en lenguajes como Java y JavaScript, con un sólido enfoque en el desarrollo backend. Me considero una persona responsable, comprometida y adaptable, capaz de trabajar de manera eficiente tanto de forma independiente como en equipo.

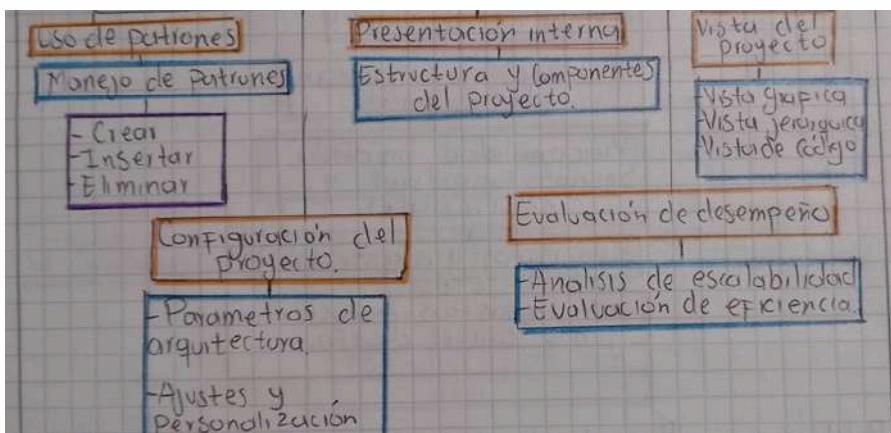
He tenido la oportunidad de participar en diversos proyectos pequeños, donde he aplicado mis conocimientos para diseñar e implementar soluciones software escalables y de alta calidad. Además, disfruto dedicar mi tiempo libre a jugar voleibol y leer, actividades que complementan mi vida personal y profesional al fomentar el trabajo en equipo, la concentración y la creatividad.

# Metodologías ágiles (Arquitectura de software)

El artículo traza la historia de la arquitectura de software, desde los años 60 hasta su formalización en los 90, cuando Perry y Wolf definieron una estructura compuesta por elementos, formas y restricciones. La arquitectura de software se consolidó como esencial para diseñar sistemas complejos, mejorar la comunicación y hacer soluciones escalables, aunque aún no tiene una definición universal.

## Reflexión

La historia de la arquitectura de software muestra que, a medida que los sistemas se volvieron más complejos, surgió la necesidad de un diseño estructurado para manejarlos mejor. Los arquitectos de software permiten que los sistemas sean escalables, sostenibles y comprensibles, alineando necesidades y soluciones técnicas. Aunque aún no tiene una definición fija, la arquitectura es clave para lograr claridad y colaboración en proyectos complejos.



## Bibliografía

Perry & Wolf (1992). "Foundations for the study of software architecture".

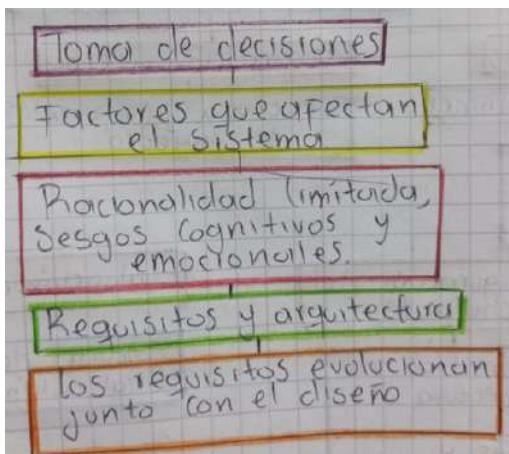
[https://ozarate.net/articulos/arquitectura\\_sw\\_sg\\_2006.pdf](https://ozarate.net/articulos/arquitectura_sw_sg_2006.pdf)

# Toma de decisiones en la arquitectura de software

El artículo examina cómo la arquitectura de software ha pasado de centrarse en la solución final a enfocarse en las decisiones de diseño. Analiza si la toma de decisiones es racional o influenciada por sesgos y destaca la importancia de las primeras decisiones en el proceso. Propone líneas de investigación para mejorar el diseño de software a través de una mejor comprensión de las decisiones involucradas.

## Reflexión

La arquitectura de software no solo debe enfocarse en la solución final, sino también en las decisiones tomadas durante su diseño. Estas decisiones, influenciadas por factores cognitivos y emocionales, pueden afectar la calidad del producto final. Comprender y mejorar el proceso de toma de decisiones es clave para lograr una arquitectura más efectiva y adaptable.



## Bibliografía

- Bass, L., Clements, P., & Kazman, R. (2013). Software Architecture in Practice (3rd ed.). Addison-Wesley.  
<https://www.sciencedirect.com/science/article/abs/pii/S0164121216000157>

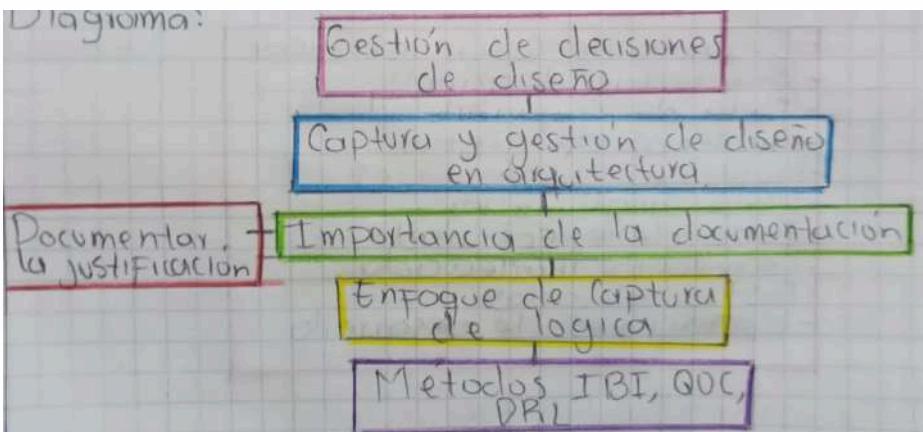
# Decisiones de diseño y fundamentos en la arquitectura de software

El artículo aborda la importancia de capturar y gestionar las decisiones de diseño en la arquitectura de software. A pesar de los avances en las últimas décadas, la comunidad de ingeniería de software aún lucha por desarrollar herramientas y técnicas efectivas para documentar la lógica detrás de estas decisiones. Se revisan enfoques históricos como IBIS, QOC, y DRL, y se destacan artículos recientes sobre cómo mejorar la documentación y gestión del conocimiento arquitectónico.

## Reflexión

Este artículo subraya la relevancia de la gestión explícita de las decisiones de diseño en la arquitectura de software para garantizar la trazabilidad y la coherencia en el desarrollo de sistemas complejos. La investigación muestra que, aunque se han propuesto varios enfoques, la adopción de estas prácticas sigue siendo limitada, lo que destaca la necesidad de más avances en herramientas y métodos para capturar y reutilizar el conocimiento arquitectónico de manera efectiva.

Diagrama:



## Bibliografía

Tyree, J., & Akerman, A. (2005). Managing Architectural Design Decisions. *IEEE Software*, 22(2), 30-37.

<https://www.sciencedirect.com/science/article/abs/pii/S0164121209001241>

# Desafíos críticos del diseño de la arquitectura de software

La arquitectura de software es clave para el éxito de los sistemas, especialmente en la Internet de las Cosas (IoT), ya que un mal diseño puede causar fallos. Este capítulo identifica varios desafíos que enfrentan los equipos de arquitectura de IoT, como la falta de desarrollo común, problemas de confiabilidad, costos elevados, y falta de tecnología adecuada. Estos desafíos se han analizado a nivel global, mostrando similitudes y diferencias entre los continentes.

## Desafíos en la arquitectura de Software IoT

- Desarrollo de común insuficiente
- Arquitectura deficiente
- Falta de confiabilidad
- Problemas de gestión
- Problemas ambientales
- Mala programación
- Falta de efectividad
- Tecnología inadecuada
- Costos elevados
- Limitaciones de desarrollo

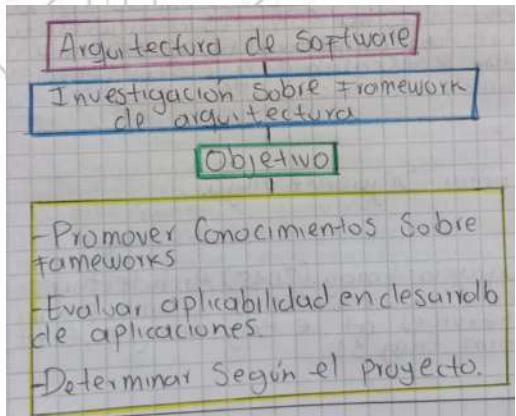
## Reflexión

La arquitectura de software bien diseñada es fundamental para evitar problemas en sistemas IoT. Superar los desafíos mencionados, como la falta de estándares comunes y tecnologías adecuadas, es crucial para garantizar sistemas más confiables y eficientes. Además, una arquitectura sólida debe ser flexible y escalable, permitiendo integrar nuevos dispositivos y tecnologías a medida que el ecosistema IoT evoluciona. La interoperabilidad entre diferentes plataformas y dispositivos también es esencial para asegurar que los sistemas puedan comunicarse de manera efectiva, independientemente de los fabricantes o protocolos utilizados.

## Bibliografía

<https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119821779.ch11>

# Análisis de los principales frameworks de arquitectura de software



Este documento se enfoca en un tema común hoy en día en el ambiente tecnológico y empresarial, el cual es la arquitectura de software y su aplicabilidad a través de frameworks a proyectos empresariales. Este documento de investigación servirá de base para obtener un conocimiento y entendimiento de los frameworks de arquitectura de software más usados en el desarrollo de aplicaciones empresariales, determinando su aplicabilidad según el proyecto que se esté abordando.

## Reflexión

La arquitectura de software es clave en el desarrollo de aplicaciones empresariales, ya que permite crear soluciones escalables y eficientes. La elección de un framework adecuado depende del tipo de proyecto y sus necesidades específicas. Conocer y aplicar los principios de arquitectura adecuados asegura que el software sea flexible y sostenible a largo plazo, siendo una herramienta estratégica fundamental para el éxito de los proyectos empresariales.

## Bibliografía

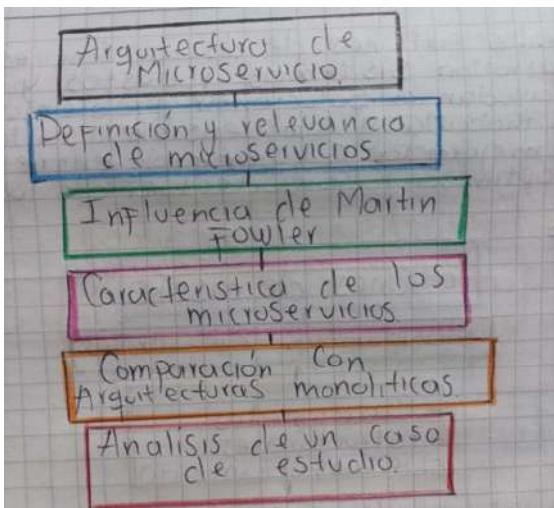
<https://sedici.unlp.edu.ar/handle/10915/52183>

# Arquitectura de microservicios

En este documento se presentará una introducción a las arquitecturas de microservicios, término que cada día toma más relevancia, liderado por personajes de la talla de Martin Fowler. Se verán características referentes a los microservicios, así como su comparación con arquitecturas monolíticas, con el fin de facilitar la comprensión; por último, se analizará un caso de estudio que tiene como objetivo mejorar la competitividad para tiendas de barrio en Colombia, para ello, se tendrá en cuenta cómo se podría iniciar con la definición de un microservicio.

## Reflexión

La adopción de arquitecturas de microservicios ofrece mayor flexibilidad y escalabilidad frente a las monolíticas, permitiendo a las empresas adaptarse rápidamente a cambios. El caso de estudio sobre tiendas de barrio en Colombia muestra cómo esta arquitectura puede mejorar la competitividad, ofreciendo una solución modular y escalable que favorece el crecimiento y la implementación de nuevas funcionalidades.

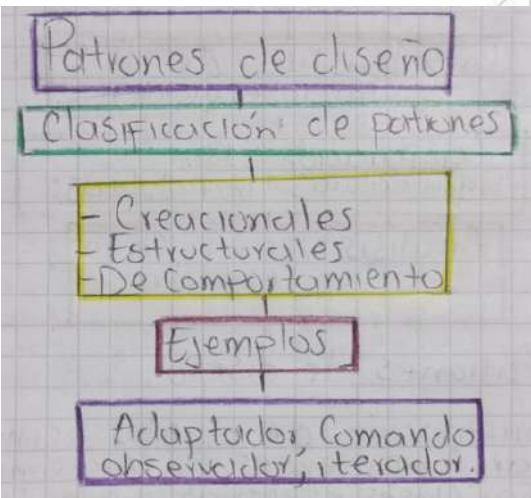


## Bibliografía

EMag. (2015). Architectures you always wondered about: Lessons learnt from adopting microservices at eBay, Google, Gilt, Hailo and Nearform.  
EMag,  
<https://revistas.udistrital.edu.co/index.php/tia/article/view/9687>

31.

# Patrones de diseño



## Reflexión

Los patrones de diseño son esenciales para desarrollar sistemas robustos y flexibles, ya que ayudan a solucionar problemas comunes de forma probada y comprensible. Usarlos con criterio mejora la colaboración entre desarrolladores y optimiza la arquitectura de los sistemas, aunque deben aplicarse de manera estratégica para evitar una complejidad innecesaria.

## Bibliografía

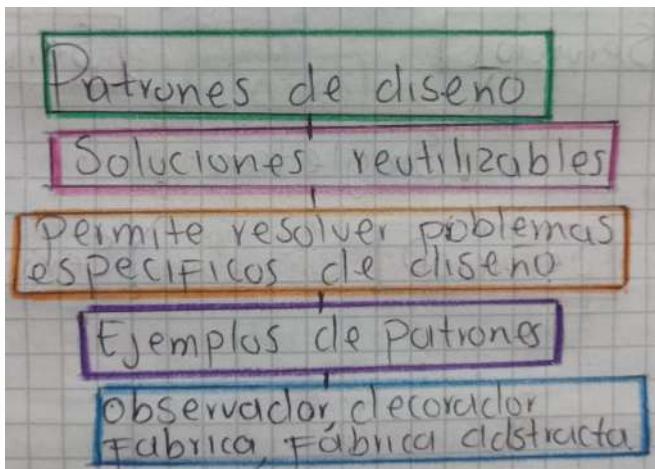
<https://www.infor.uva.es/~felix/datos/priii/tema7.pdf>

La orientación a objetos permite organizar y reutilizar componentes en sistemas complejos, pero diseñar clases efectivas sigue siendo un desafío. Los patrones de diseño ofrecen soluciones estándar a problemas de estructura y comportamiento en el desarrollo, ayudando a crear sistemas flexibles y sostenibles. Los patrones se dividen en:

- Creacionales:** Facilitan la creación de objetos.
- Estructurales:** Simplifican la composición de objetos.
- De comportamiento:** Manejan la interacción entre objetos. Ejemplos de patrones incluyen el Adaptador para hacer compatibles interfaces, el Comando para encapsular solicitudes, el Observador para sincronizar objetos, y el Iterador para recorrer estructuras sin exponer detalles internos.

# Patrones de diseño en el desarrollo de software

Los patrones de diseño son soluciones reutilizables para problemas comunes en el desarrollo de software. Ayudan a los desarrolladores a crear soluciones eficientes, a comunicar ideas arquitectónicas y a evitar errores costosos. Algunos ejemplos incluyen el Patrón de Observador, Decorador, Método de Fábrica y Fábrica Abstracta. Permiten una mayor flexibilidad y escalabilidad en las aplicaciones, facilitando el mantenimiento y la evolución del software.



## Reflexión

Los patrones de diseño son esenciales para crear software modular, escalable y fácil de mantener. Facilitan la comunicación entre desarrolladores y ayudan a evitar problemas comunes, mejorando la calidad y eficiencia del código. Su correcta implementación también promueve la consistencia en el diseño de aplicaciones, al seguir estos patrones, los desarrolladores pueden crear soluciones más robustas y preparadas para el cambio.

## Bibliografía

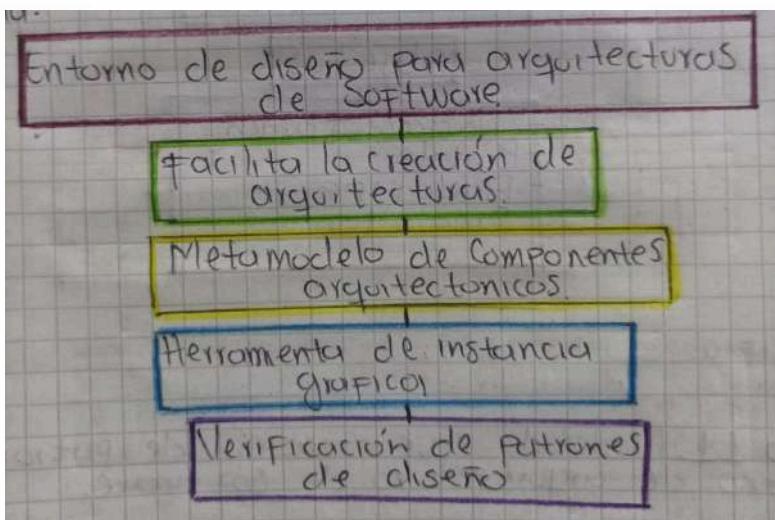
<https://ieeexplore.ieee.org/abstract/document/5982228>

# Modelado de patrones de diseño de arquitectura de software

Este trabajo presenta un entorno de diseño integral para la creación de arquitecturas de software orientadas a aplicaciones web. El entorno abstrae problemas clave en el diseño y ofrece módulos que ayudan a los arquitectos a generar diseños de calidad. Se basa en un metamodelo de componentes arquitectónicos que identifica elementos comunes en estas arquitecturas. Además, el entorno incluye una herramienta gráfica para la instanciación de modelos y verifica la correcta aplicación de patrones de diseño, garantizando así que los diseños sean apropiados y efectivos.

## Reflexión

El enfoque de este trabajo subraya la importancia de contar con herramientas de diseño que faciliten la creación de arquitecturas de software robustas y escalables, especialmente en el contexto de aplicaciones web. El uso de un metamodelo y la verificación de patrones de diseño son clave para asegurar que los arquitectos sigan buenas prácticas y produzcan diseños coherentes y de alta calidad. Este tipo de herramientas no solo optimiza el proceso de diseño, sino que también reduce los errores humanos, promoviendo la eficiencia y la fiabilidad en el desarrollo de software.



## Bibliografía

<https://ri.conicet.gov.ar/handle/11336/125130>

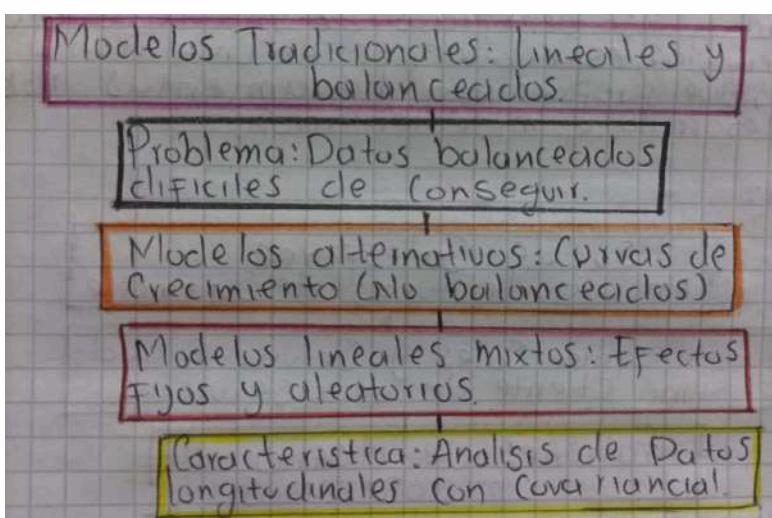
# Estudios longitudinales de medidas repetidas.

## Modelos de diseño

Los modelos tradicionales para datos de medidas repetidas son lineales y requieren datos balanceados. Los modelos lineales mixtos, en cambio, no necesitan datos balanceados y permiten estimar tanto los efectos fijos como los aleatorios, lo que los hace adecuados para datos longitudinales, que pueden ser correlacionados o incompletos. Estos modelos son especialmente útiles en situaciones donde las observaciones no son independientes, como en estudios médicos o sociales donde se sigue a los mismos individuos a lo largo del tiempo.

## Reflexión

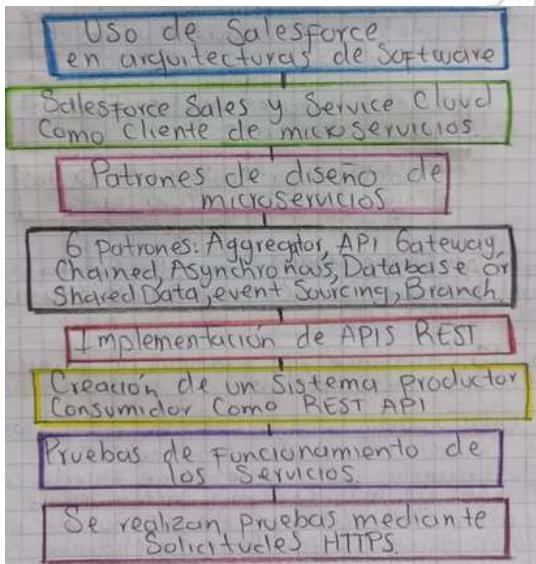
Los modelos lineales mixtos son más flexibles y eficaces para analizar datos longitudinales, ya que manejan la correlación entre observaciones y la falta de datos, ofreciendo un análisis más preciso y completo. Además, permiten incorporar tanto efectos fijos, que representan factores comunes a todos los sujetos, como efectos aleatorios, que capturan la variabilidad individual. Esta capacidad de manejar estructuras de datos más complejas los hace ideales para estudios en los que las medidas repetidas dependen de factores individuales que pueden variar a lo largo del tiempo.



## Bibliografía

- Albert, P. S. (1999). Longitudinal data analysis (repeated measures) in clinical trials. *Statistics in Medicine*, 18, 1707-1732.  
[https://scielo.isciii.es/scielo.php?pid=S198938092008000300005&script=sci\\_arttext](https://scielo.isciii.es/scielo.php?pid=S198938092008000300005&script=sci_arttext)

# Salesforce API framework para patrones de diseño de microservicios



## Reflexión

El proyecto resalta cómo Salesforce puede ser una plataforma eficiente para la implementación de microservicios, aprovechando su capacidad con Apex REST. La integración de patrones de diseño de microservicios facilita la creación de sistemas modulares, escalables y fáciles de mantener. Esto pone de relieve la flexibilidad y los beneficios de una arquitectura de microservicios, especialmente en un entorno como Salesforce, que proporciona herramientas robustas para su desarrollo e implementación.

Este trabajo explora el uso de Salesforce Sales y Service Cloud en arquitecturas de microservicios, destacando cómo Salesforce puede actuar como cliente de microservicios a través de Apex REST. Se describen seis patrones de diseño de microservicios (PDMS): Aggregator, API Gateway, Chained, Asynchronous, Database or Shared Data, Event Sourcing y Branch. El proyecto implementa estos patrones en un sistema productor-consumidor, compuesto por varias REST APIs, y se realizan pruebas mediante solicitudes HTTPS para demostrar su funcionalidad.

## Bibliografía

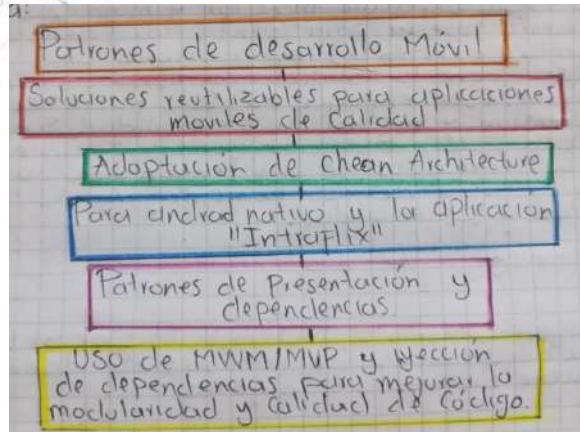
Milbradt Rodríguez, Daniel (2023). Salesforce API framework para patrones de diseño de microservicios.

<https://oa.upm.es/75035/>

# Arquitectura limpia : impacto en el rendimiento y la capacidad de mantenimiento de proyectos nativos de Android

Clean Architecture ha ganado popularidad en el desarrollo nativo de Android, promoviendo un código modular, escalable y fácil de mantener. Esta arquitectura, adaptada en proyectos como "InstaFlix", permite que los desarrolladores

trabajen simultáneamente en diferentes partes del sistema, mejorando la eficiencia. Su enfoque en patrones como MVVM y MVP, junto con la inyección de dependencias, facilita modificaciones sin afectar otros componentes.



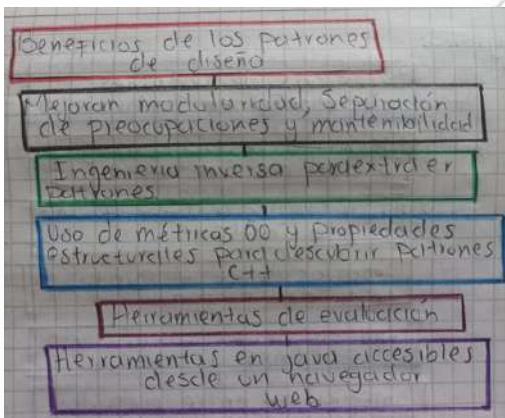
## Reflexión

La implementación de Clean Architecture en Android destaca el valor de una estructura bien organizada, que favorece tanto la colaboración en equipo como la calidad del software. Al fomentar el modularidad, esta arquitectura contribuye a construir aplicaciones robustas y adaptables, esenciales en el entorno móvil actual.

## Bibliografía

Rachovski, T., Hadzhikoleva, S., Hadzhikolev, E., Lengerov, A.: Uso de principios de arquitectura limpia para mejorar el diseño y la implementación de la plataforma móvil en línea.  
[https://doi.org/10.1007/978-981-16-7657-4\\_2](https://doi.org/10.1007/978-981-16-7657-4_2)

# Recuperación de patrones de diseño orientados a objetos



## Reflexión

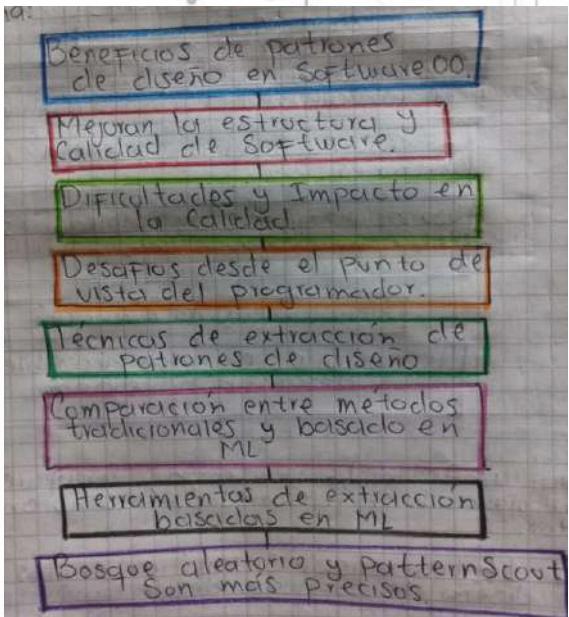
Los patrones de diseño son esenciales para crear software modular y mantible. Utilizar ingeniería inversa para identificar estos patrones en proyectos existentes proporciona una visión valiosa sobre la calidad y la estructura del software. Este enfoque no solo facilita la comprensión y mejora la trazabilidad, sino que también puede asegurar que los patrones de diseño adecuados se mantengan a lo largo del ciclo de vida del software, mejorando su calidad y sostenibilidad.

El texto aborda los patrones de diseño orientados a objetos (OO) como bloques reutilizables que mejoran el modularidad, separación de preocupaciones y mantenibilidad en el software. Se destaca la utilidad de la ingeniería inversa para identificar estos patrones en diseños o códigos existentes, lo que facilita la comprensión del programa, la trazabilidad entre diseño y código, y la evaluación de la calidad. Se presenta un enfoque experimental que usa métricas de software OO y propiedades estructurales para extraer patrones del código C++ y se evalúa con herramientas desarrolladas en Java en proyectos industriales.

## Bibliografía

Antoniol, G., Fiutem, R., Cristoforetti, L., 1998a. Recuperación de patrones de diseño en software orientado a objetos.  
<https://www.sciencedirect.com/science/article/abs/pii/S0164121201000619>

# Patrones de diseño de software desde la perspectiva actual



## Reflexión

Aunque los patrones de diseño mejoran la calidad del software, las técnicas tradicionales de extracción son limitadas. El uso de ML mejora significativamente la precisión en la identificación de patrones, lo que podría optimizar el desarrollo de software en sistemas complejos. Además, al aplicar algoritmos de aprendizaje automático, es posible manejar grandes volúmenes de código y detectar patrones que podrían pasar desapercibidos con métodos convencionales. Esto no solo acelera el proceso de desarrollo, sino que también permite identificar oportunidades para mejorar la arquitectura del software de manera más efectiva.

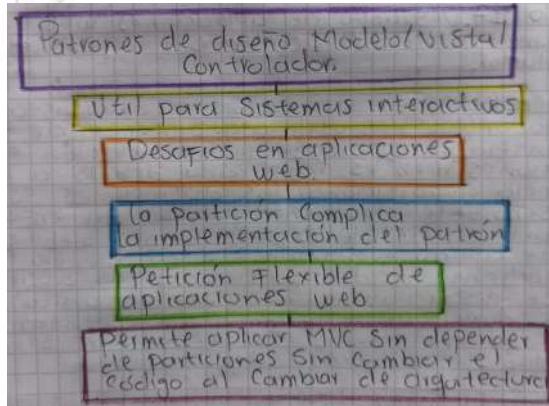
El artículo destaca los beneficios de los patrones de diseño en sistemas OO grandes y revisa estudios sobre su extracción, incluyendo enfoques con aprendizaje automático (ML). Compara herramientas convencionales con modelos como el de bosque aleatorio y PatternScout, mostrando que estas últimas ofrecen mayor precisión y efectividad en la extracción de patrones. Además, se analiza cómo el uso de técnicas de ML permite identificar patrones de diseño de manera más eficiente, incluso en sistemas complejos y de gran escala. Se discuten los desafíos asociados con la extracción automática, como la variabilidad en el código y la ambigüedad en la identificación de patrones.

## Bibliografía

<https://ieeexplore.ieee.org/abstract/document/10253758>

# Desarrollo de aplicaciones web utilizando el patrón de diseño Modelo/Vista/Controlador

El patrón de diseño Modelo/Vista/Controlador (MVC) es útil para sistemas de software interactivos, pero su implementación en aplicaciones web es complicada debido a la tendencia a particionar las aplicaciones durante el diseño. Este documento propone un enfoque de "Partición Flexible de Aplicaciones Web", que permite aplicar el patrón MVC de manera independiente de las particiones. Las aplicaciones se desarrollan y prueban en un único espacio de direcciones y luego se implementan en diversas arquitecturas cliente/servidor sin necesidad de modificar el código.



## Reflexión

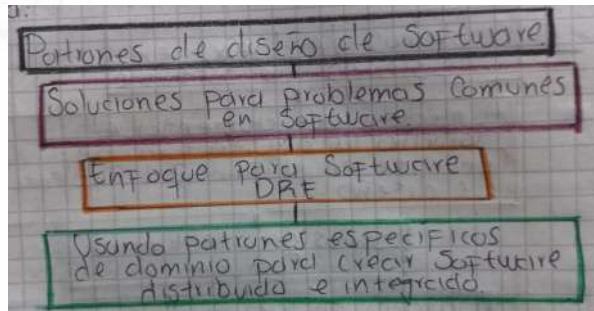
El patrón MVC es fundamental para el modularidad y la organización de sistemas interactivos, pero la partición temprana en las aplicaciones web puede generar complicaciones. La propuesta de Partición Flexible ofrece una solución interesante, permitiendo que el patrón MVC se aplique de manera más flexible y escalable, sin los inconvenientes de tener que reestructurar el código ante cambios de arquitectura.

## Bibliografía

M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams y J. Shuster, "UIML: un lenguaje de interfaz de usuario XML independiente del dispositivo", Actas de la octava conferencia internacional sobre la World Wide Web , págs. 617-630, mayo de 1999.  
<https://ieeexplore.ieee.org/abstract/document/950428>

# **Creación de arquitecturas de software específicas de cada dominio a partir de patrones de diseño arquitectónico de software**

Esta investigación propone un enfoque sistemático para diseñar software distribuido, en tiempo real e integrado (DRE), específico de un dominio, utilizando patrones de diseño de arquitectura de software. Para manejar la variabilidad en un dominio DRE, se aplican conceptos de línea de productos de software para organizar las características y patrones. Las arquitecturas generadas se validan mediante simulaciones en el diseño.



## **Reflexión**

El uso de patrones de diseño de arquitectura en software distribuido y en tiempo real es fundamental para abordar problemas complejos, pero su aplicación práctica puede ser desafiante debido a su generalidad. Este enfoque sistemático mejora la adaptabilidad y reutilización de los patrones, y al validarlos en simulaciones, asegura que las soluciones sean efectivas en contextos específicos, como el software de vuelo espacial

## **Bibliografía**

Mzid R Selvi S Abid M(2024) Panorama de la investigación de patrones en ingeniería de software: taxonomía, estado del arte y direcciones futuras SN Computer Science 10.1007/s42979-024-02767-8 5 :4Fecha de publicación en línea: 8 de abril de 2024  
<https://dl.acm.org/doi/abs/10.1145/1985793.1986026>

# Introducción a la arquitectura de software

## Problemas de diseño en Sistemas de Software

A medida que aumenta el tamaño, la arquitectura se convierte en el principal problema de diseño.

### Estilos arquitectónicos comunes

Basado en Sistemas circulares, combinando diferentes estilos

Problemas pendientes y direcciones de investigación

Área de investigación promete más de investigación

## Reflexión

La arquitectura de software es crucial para gestionar la complejidad en sistemas grandes. Combinando estilos arquitectónicos, se pueden crear soluciones más claras y escalables. Este enfoque destaca la importancia de investigar nuevas formas de integrar estos estilos para optimizar el diseño de sistemas.

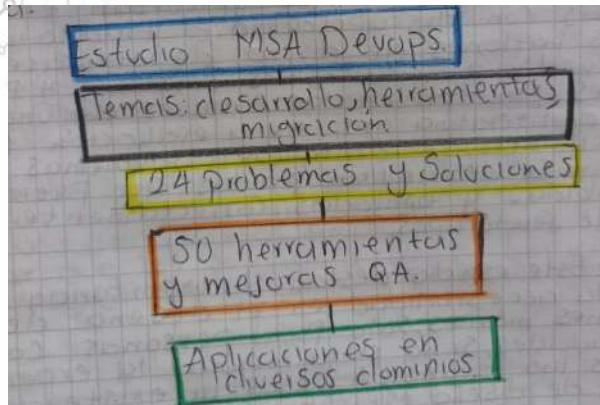
## Bibliografía

[https://www.worldscientific.com/doi/abs/10.1142/9789812798039\\_0001](https://www.worldscientific.com/doi/abs/10.1142/9789812798039_0001)

Este artículo presenta la arquitectura de software como solución a los problemas de diseño en sistemas grandes. Explora estilos arquitectónicos comunes y cómo combinarlos, utilizando seis estudios de caso para mostrar cómo las representaciones arquitectónicas mejoran la comprensión de sistemas complejos. También aborda problemas abiertos y futuras direcciones de investigación. Además, destaca cómo la elección adecuada de un estilo arquitectónico puede facilitar la escalabilidad, modularidad y mantenimiento de los sistemas, optimizando su rendimiento a largo plazo. Se discuten los desafíos de integrar diferentes enfoques arquitectónicos en sistemas existentes y cómo las técnicas emergentes, como los modelos basados en inteligencia artificial, podrían revolucionar la toma de decisiones arquitectónicas.

# Comprender y abordar los atributos de calidad de la arquitectura de microservicios

Este estudio revisa la literatura sobre la implementación de microservicios (MSA) en DevOps entre 2009 y 2018. Identifica tres áreas clave: desarrollo y operaciones, herramientas de soporte y experiencias de migración. También se destacan 24 problemas con sus soluciones, 50 herramientas que apoyan MSA y cómo MSA mejora la calidad del software. Los resultados ayudan a investigadores y profesionales a abordar los desafíos de MSA en DevOps.



## Reflexión

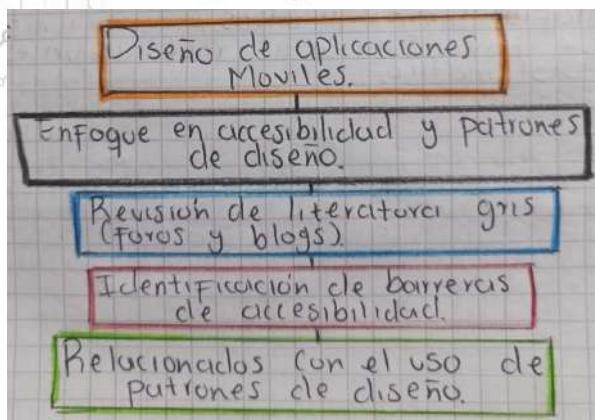
La combinación de MSA (Microservices Architecture) y DevOps mejora la flexibilidad y agilidad en el desarrollo de software. Este estudio ofrece soluciones prácticas y herramientas para superar los desafíos de implementar MSA, lo que facilita la adopción de estas prácticas en diversas industrias. Además, resalta cómo la integración de ambos enfoques permite una mejor gestión del ciclo de vida del software, desde la planificación hasta la implementación y el mantenimiento, optimizando la colaboración entre equipos de desarrollo y operaciones.

## Bibliografía

Arquitecturas de software para sistemas robóticos: un estudio de mapeo sistemático  
<https://www.sciencedirect.com/science/article/abs/pii/S0950584920301993>

# Prevención de barreras de accesibilidad: pautas para el uso de patrones de diseño de interfaz de usuario en aplicaciones móviles

El artículo aborda la importancia de la accesibilidad en el diseño de interfaces de usuario para aplicaciones móviles. Aunque los dispositivos móviles han ganado popularidad, siguen existiendo barreras que dificultan el acceso a personas con discapacidades. El trabajo investiga cómo los patrones de diseño de interfaces de usuario pueden generar estas barreras y propone directrices para prevenirlas. A través de una revisión de literatura gris y una evaluación con 60 participantes, se identificaron problemas comunes y se desarrollaron pautas que fueron bien recibidas y aplicadas correctamente en el diseño de prototipos.



## Reflexión

Este estudio destaca la importancia de integrar la accesibilidad en el diseño de aplicaciones móviles, no solo para personas con discapacidades, sino para mejorar la experiencia de todos los usuarios. A pesar de la existencia de patrones de diseño, estos no siempre consideran las barreras de accesibilidad. La integración de principios accesibles desde las fases iniciales del diseño puede beneficiar a un público más amplio, incluyendo personas mayores o usuarios con diferentes necesidades cognitivas y físicas. Además, se resalta que la accesibilidad no solo cumple con normativas legales y éticas, sino que también amplía el alcance de las aplicaciones, mejorando la satisfacción y fidelización de los usuarios.

## Bibliografía

Análisis comparativo de Android y iOS desde el punto de vista de la seguridad  
<https://www.sciencedirect.com/science/article/abs/pii/S0164121221002831>

# Ejemplos de patrones de diseño de software

Enfoques de despliegue de patrones de diseño.

- Encapsulado en máquinas Virtuales (MV)
- Encapsulado en Contenedores Virtuales

Enfoque 1: Máquinas Virtuales (MV)

- Virtualización de hardware
- Recursos reservados incluso cuando la MV no está activa.
- Uso de hipervisor para gestionar la MVS.

Enfoque 2: Contenedores Virtuales (CV)

- No utiliza virtualización de hardware
- Recursos disponibles bajo demanda
- Uso de plataformas de contenedores en lugar de hipervisor.

## Reflexión

La principal ventaja de los contenedores sobre las máquinas virtuales es su eficiencia en el uso de recursos. Los contenedores, al compartir el mismo núcleo del sistema operativo, son más ligeros y rápidos de iniciar, lo que permite un uso más eficiente del hardware y un mayor número de instancias en una misma infraestructura. Esto resulta especialmente beneficioso en entornos de microservicios y aplicaciones distribuidas, donde la rapidez de despliegue y la escalabilidad son esenciales. Ambos enfoques deben asegurar que las soluciones sean confiables y escalables, adaptándose a diferentes necesidades. Sin embargo, las máquinas virtuales siguen siendo relevantes en situaciones donde se requiere un aislamiento más fuerte o cuando se necesita ejecutar sistemas operativos diferentes en el mismo hardware.

Existen dos enfoques para desplegar patrones de diseño: máquinas virtuales (MV), que reservan recursos permanentemente, y contenedores virtuales (CV), que asignan recursos solo cuando están activos. Los contenedores son más eficientes al no requerir virtualización de hardware y no necesitan hipervisor. Ambos enfoques deben garantizar aspectos cualitativos como eficiencia y confiabilidad. Los contenedores, al ser más ligeros, permiten una mayor densidad de aplicaciones en el mismo entorno, lo que reduce el uso de recursos y mejora la escalabilidad. Por otro lado, las máquinas virtuales ofrecen un aislamiento más fuerte entre las aplicaciones, lo que puede ser crucial para ciertos escenarios de seguridad.

## Bibliografía

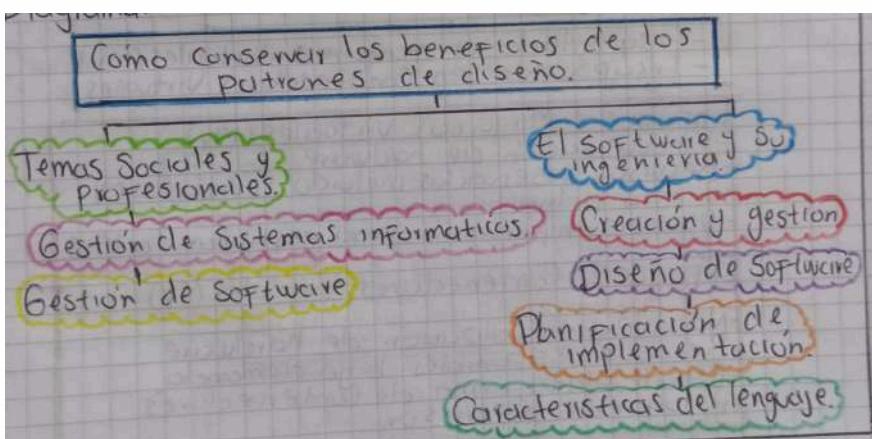
<https://repositorio.cinvestav.mx/bitstream/handle/cinvestav/3200/SSIT0016546.pdf?sequence=1>

# Cómo conservar los beneficios de los patrones de diseño

El aumento en el número de patrones de diseño ha dificultado su uso efectivo, creando problemas de vocabulario común y de documentación confusa. Este análisis propone reducir el número de patrones fundamentales y utilizar abstracciones sólidas en el lenguaje para resolver estos problemas, mejorando así la documentación y facilitando su aplicación. Al simplificar y estandarizar los patrones, se facilita la comunicación entre los desarrolladores, lo que reduce la ambigüedad y mejora la comprensión en equipos grandes o interdisciplinares.

## Reflexión

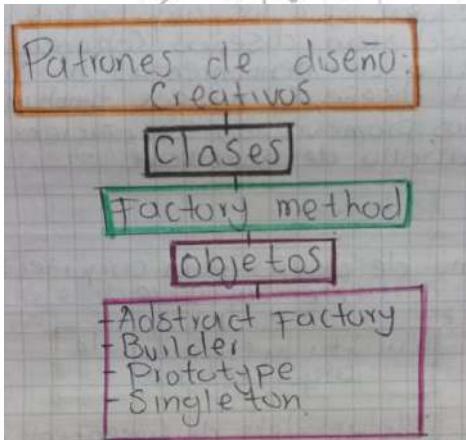
Este texto destaca cómo la proliferación de patrones de diseño ha complicado su implementación y comprensión. Aunque los patrones de diseño son útiles, su exceso dificulta la gestión de un vocabulario común y oscurece la documentación. La propuesta de reducir el número de patrones fundamentales y utilizar abstracciones más sólidas en el lenguaje puede ser clave para mejorar la claridad y la efectividad de la documentación, facilitando su uso y comprensión.



## Bibliografía

<https://dl.acm.org/doi/abs/10.1145/286936.286952>

# Patrones de diseño: Creativos



## Reflexión

Adoptar patrones de diseño y buenas prácticas de arquitectura es clave para el éxito a largo plazo en proyectos de software a gran escala. La consistencia en la estructura del código no solo mejora la calidad y la legibilidad, sino que también acelera el proceso de incorporación de nuevos miembros al equipo, permitiendo que todos se concentren en tareas importantes en lugar de perder tiempo entendiendo un código desorganizado.

Se explica cómo los principios de los patrones de diseño y los patrones de arquitectura de código para aplicaciones JavaScript a gran escala pueden ayudar a mantener el código organizado, fácil de entender y sencillo de mantener. Al aplicar estos patrones y técnicas, los desarrolladores encontrarán que los archivos de código tienen una estructura similar, lo que facilita el trabajo en equipo y la integración de nuevos desarrolladores en diferentes proyectos. Además, el uso consistente de estos patrones permite que el código sea más modular y reutilizable, reduciendo la duplicación y mejorando la escalabilidad de las aplicaciones.

## Bibliografía

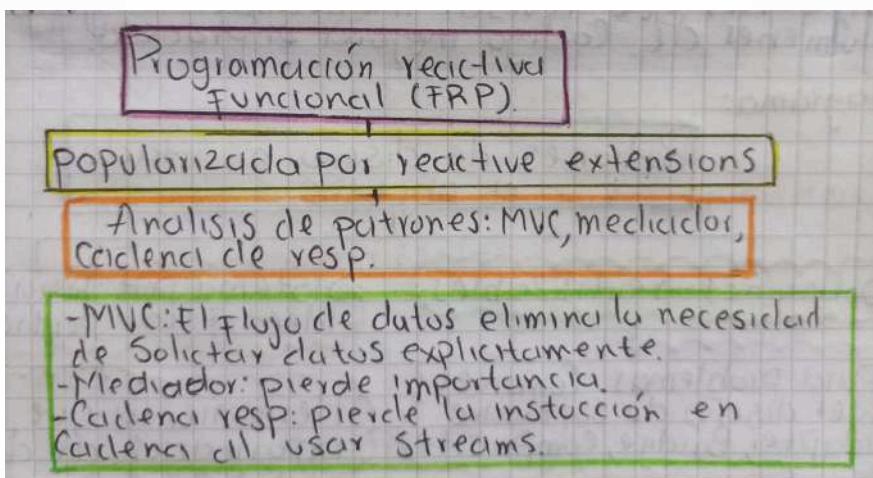
- Odell, D. (2014). Patrones de diseño: creacional. En: Pro-JavaScript Development. Apress, Berkeley, CA  
[https://link.springer.com/chapter/10.1007/978-1-4302-6269-5\\_5](https://link.springer.com/chapter/10.1007/978-1-4302-6269-5_5)

# Patrones en programación reactiva: de transformadora a reactiva

La programación reactiva funcional (FRP), que surgió en 1997, ha ganado popularidad gracias a tecnologías como Reactive Extensions y Elm. Este enfoque facilita la construcción de programas que responden a eventos externos de manera automática. Sin embargo, al aplicar FRP a patrones de diseño clásicos como MVC, Cadena de Responsabilidad y Mediador, se observan cambios importantes. Por ejemplo, en MVC, la necesidad de solicitar datos actualizados desaparece, y en el patrón Mediador, la clase mediadora pierde relevancia. En la Cadena de Responsabilidad, el uso de Streams elimina la estructura en cadena original.

## Reflexión

La programación reactiva mejora la gestión de flujos de datos, pero altera la forma en que los patrones tradicionales funcionan. Aunque muchos patrones se adaptan bien a FRP, otros pierden su efectividad, lo que requiere una reconsideración de cómo se aplican en un entorno reactivo. Esto muestra que, al usar FRP, es esencial ajustar los patrones de diseño para aprovechar sus beneficios sin perder la claridad y eficiencia del código.



## Bibliografía

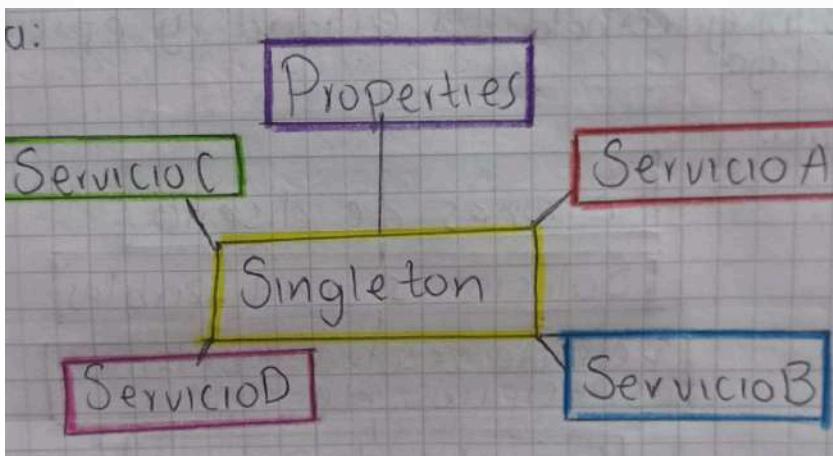
<https://oulurepo.oulu.fi/handle/10024/40749>

## Semifallo

Un singleton es un patrón en software que asegura que una clase tenga una única instancia y proporcione un acceso global a esa instancia. Se usa cuando un recurso debe ser único y no puede tener múltiples copias, como el caso del dispositivo GPS en un iPhone. Aunque puedes intentar crear múltiples instancias de la clase que gestiona el GPS, siempre estarás trabajando con un solo GPS, ya que solo hay uno físico en el dispositivo. Este patrón es útil para gestionar recursos compartidos, como conexiones a bases de datos, impresoras o configuraciones globales, donde la creación de múltiples instancias podría generar conflictos o un consumo innecesario de recursos.

## Reflexión

El patrón singleton es útil cuando un recurso debe ser único en todo el sistema, evitando la creación de copias innecesarias. Aunque en teoría podrías querer manejar varias instancias de un recurso como el GPS, en la práctica solo hay un recurso físico disponible, por lo que el singleton garantiza coherencia y control sobre ese recurso compartido. Al asegurar que solo exista una instancia, el patrón ayuda a mantener la consistencia en el acceso y uso del recurso, evitando posibles errores derivados de la manipulación concurrente. El patrón facilita la gestión de recursos donde tener múltiples instancias podría llevar a conflictos o resultados impredecibles.



## Bibliografía

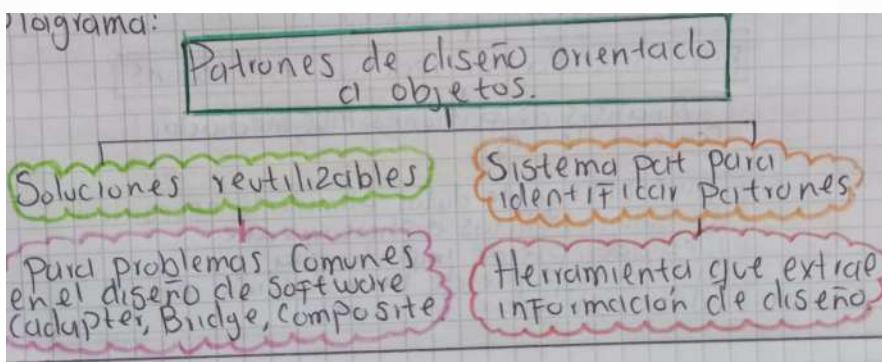
Chung, C. (2011). Singleton. En: Pro Objective-C Design Patterns for iOS. Apress. [https://link.springer.com/chapter/10.1007/978-1-4302-3331-2\\_7](https://link.springer.com/chapter/10.1007/978-1-4302-3331-2_7)

# Recuperación de diseño mediante búsqueda automatizada de patrones de diseño estructural en software orientado a objetos

En el diseño orientado a objetos, los patrones de diseño como Adapter, Bridge, Composite, Decorator y Proxy son soluciones reutilizables para problemas comunes en la construcción de software. La herramienta "Pat" permite identificar estas instancias de patrones en software existente, incluso si no fueron aplicados explícitamente. Funciona extrayendo información de los archivos de encabezado de C++ y traduciéndola en reglas PROLOG para buscar patrones en el código. Al aplicar esta técnica a cuatro aplicaciones, los autores encontraron que la precisión para identificar patrones es de aproximadamente un 40%.

## Reflexión

El uso de herramientas automáticas como Pat para identificar patrones de diseño en código existente es valioso, ya que facilita la comprensión y el mantenimiento del software. Aunque la precisión no sea perfecta, la capacidad de encontrar patrones estructurales sin tener que revisar manualmente grandes volúmenes de código mejora la eficiencia del proceso de mantenimiento. Sin embargo, el filtrado manual sigue siendo necesario para obtener resultados más precisos.



## Bibliografía

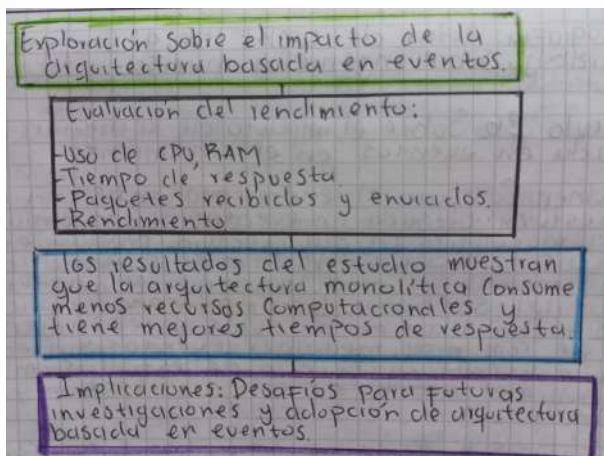
<https://ieeexplore.ieee.org/abstract/document/558905>

# Sobre el impacto de la arquitectura basada en eventos en el rendimiento

Este estudio exploratorio compara el rendimiento entre una arquitectura basada en eventos y una arquitectura monolítica. La arquitectura basada en eventos es popular debido a su capacidad para mejorar el modularidad y el mantenimiento del software, descomponiendo aplicaciones monolíticas en módulos independientes que se comunican mediante eventos. Sin embargo, la literatura carecía de estudios empíricos sobre el impacto de esta arquitectura en el rendimiento. Se compararon dos versiones de la misma aplicación: una con arquitectura basada en eventos y otra con arquitectura monolítica.

## Reflexión

Este estudio subraya la importancia de contar con pruebas empíricas antes de adoptar una arquitectura en el desarrollo de software, especialmente en entornos donde el rendimiento es crucial. Aunque la arquitectura basada en eventos ofrece ventajas como el modularidad y la facilidad de mantenimiento, sus beneficios en términos de rendimiento no siempre son evidentes.



## Bibliografía

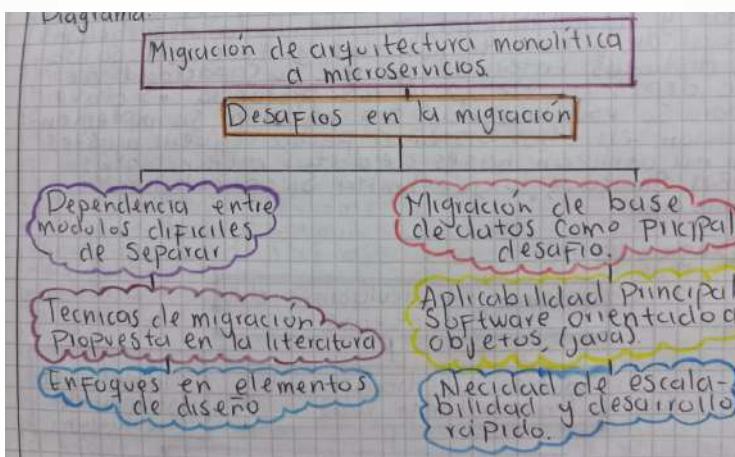
Brcode: un enfoque de ingeniería basado en modelos interpretativos para aplicaciones empresariales(2018)  
<https://www.sciencedirect.com/science/article/abs/pii/S0167739X23003977>

# Migración de una arquitectura monolítica a microservicios

La arquitectura de microservicios ha ganado popularidad debido a las limitaciones de las arquitecturas monolíticas en cuanto a escalabilidad y rapidez en los ciclos de desarrollo. Sin embargo, migrar de una arquitectura monolítica a una de microservicios no es un proceso sencillo, ya que los sistemas suelen tener demasiadas dependencias entre módulos, lo que dificulta su separación. Este estudio analiza 20 técnicas de migración propuestas en la literatura, destacando que la mayoría de ellas se enfocan en enfoques de diseño y se aplican principalmente a software orientado a objetos, siendo Java el lenguaje más utilizado.

## Reflexión

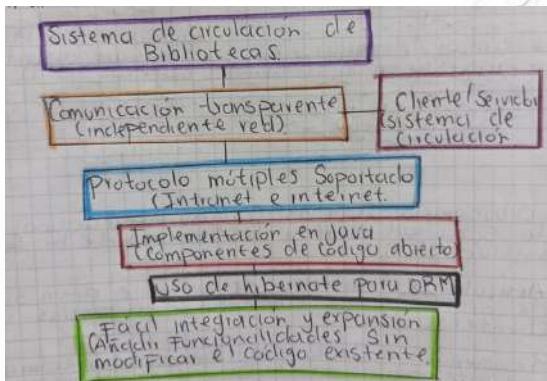
El cambio de arquitectura de monolítica a microservicios es una tendencia en la industria del software que responde a la necesidad de mayor flexibilidad y escalabilidad. Sin embargo, la migración no es trivial y requiere una planificación cuidadosa, especialmente cuando se trata de gestionar las dependencias entre módulos y la base de datos. Este estudio subraya la importancia de contar con estrategias bien definidas y basadas en principios de diseño para realizar una migración efectiva.



## Bibliografía

<https://ieeexplore.ieee.org/abstract/document/8990350>

# Arquitectura de software de un sistema de circulación de bibliotecas distribuidas cliente/servidor



## Reflexión

Esta arquitectura destaca por su flexibilidad y adaptabilidad, permitiendo la integración fácil de nuevas funciones y su uso en distintos entornos de red. Su capacidad de no depender de un único protocolo es clave para su escalabilidad. Sin embargo, su implementación en otros sistemas podría requerir ajustes si no se utilizan bases de datos o tecnologías ORM similares, lo que limita su aplicabilidad universal.

## Bibliografía

- Milosavljević, B. y Tešendić, D. (2010), "Arquitectura de software de un sistema de circulación de bibliotecas distribuidas cliente/servidor", The Electronic Library , vol. 28, núm. 2, págs. 286-299.  
<https://www.emerald.com/insight/content/doi/10.1108/02640471011033648/full.html>

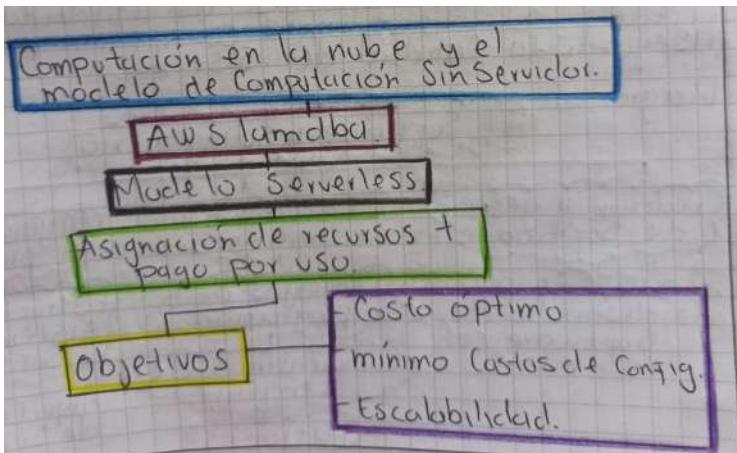
El artículo describe una arquitectura de software para permitir la comunicación transparente entre el cliente y el servidor en un sistema de circulación de bibliotecas. La arquitectura, modelada con UML y desarrollada en Java, es independiente de la red y soporta múltiples protocolos, funcionando tanto en intranet como en Internet. Se integra con el sistema BISIS y utiliza Hibernate para el mapeo de objetos. La solución es flexible y permite añadir nuevas funcionalidades sin modificar el código existente.

# Arquitectura sin servidor: una revolución en la computación en la nube

La computación sin servidor (serverless) ha emergido como un modelo clave en la computación en la nube, en el que el proveedor de servicios gestiona la asignación de recursos, permitiendo a los usuarios pagar solo por los recursos que consumen, en lugar de por capacidad fija previamente adquirida. Este enfoque busca optimizar costos y mejorar la escalabilidad de las aplicaciones en la nube. AWS Lambda es un ejemplo prominente de este modelo, que ofrece flexibilidad y eficiencia en la gestión de recursos. El artículo presenta una revisión exhaustiva de la arquitectura de computación sin servidor y explora las principales vías de investigación en este campo.

## Reflexión

La computación sin servidor representa una evolución significativa en la forma en que las aplicaciones se ejecutan y gestionan en la nube, proporcionando eficiencia y flexibilidad a través de la asignación dinámica de recursos. Este modelo tiene el potencial de transformar la manera en que las empresas gestionan sus infraestructuras, permitiendo un escalado más eficiente y costos más controlados



## Bibliografía

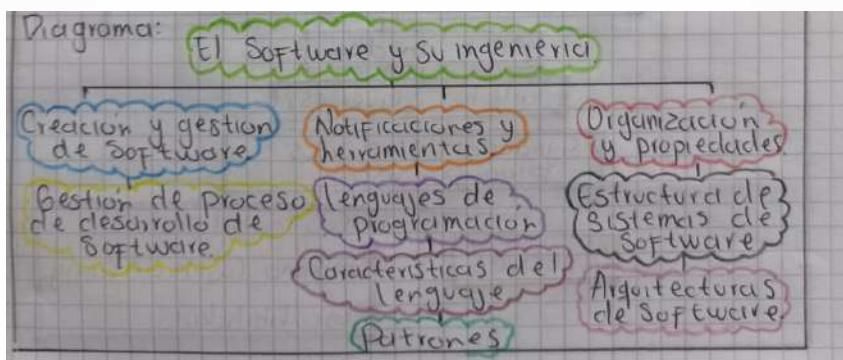
<https://ieeexplore.ieee.org/abstract/document/8939081>

# Construir la arquitectura del software basándose en patrones de diseño

En base a las palabras sobre por qué debemos estudiar la arquitectura de software, este artículo analizó cómo actúa la idea de construir una arquitectura de software basada en patrones de diseño en el proceso de desarrollo de software, discutió la tendencia de desarrollo de la arquitectura de software y los patrones de diseño, y señaló los problemas a los que debe prestar atención durante el período de construcción de una arquitectura de software basada en patrones de diseño. Además, este artículo describió cómo funciona el patrón MVC en J2EE y explicó qué responsabilidades tienen las distintas partes de MVC.

## Reflexión

Destaca la importancia de utilizar patrones de diseño, como el MVC en J2EE, para crear arquitecturas de software más modulares y escalables. Aunque estos patrones mejoran el desarrollo y mantenimiento, es crucial gestionar adecuadamente las responsabilidades entre las distintas capas. La reflexión es que, si bien los patrones son útiles, deben aplicarse con cuidado para evitar problemas durante su implementación y asegurar la calidad del sistema.

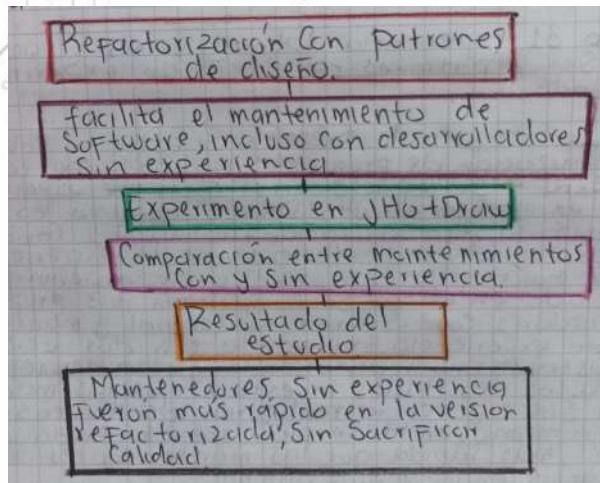


## Bibliografía

<https://dl.acm.org/doi/abs/10.5555/2452568.2452730>

# Experiencia laboral versus refactorización según patrones de diseño: un experimento controlado

Este artículo presenta un estudio sobre la refactorización de programas mediante patrones de diseño y su impacto en el mantenimiento de software, específicamente en el sistema JHotDraw. El objetivo es evaluar si la refactorización facilita los cambios anticipados, independientemente de la experiencia de los mantenedores. Se realizó un experimento controlado en el que se compararon dos grupos: mantenedores con experiencia laboral y mantenedores sin experiencia, trabajando con dos versiones del sistema: una refactorizada y otra no refactorizada.



## Reflexión

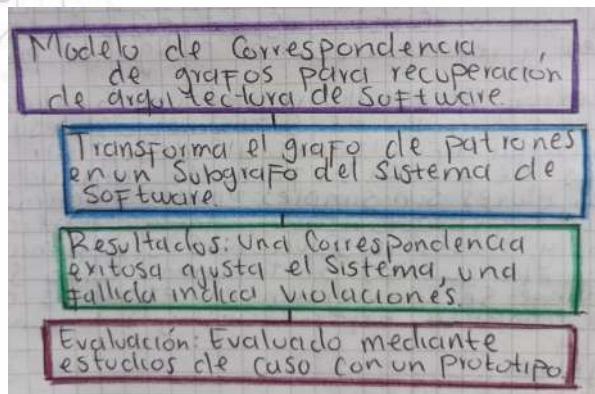
Este estudio muestra que la refactorización de código mediante patrones de diseño puede ser beneficiosa no solo para expertos, sino también para desarrolladores con menos experiencia. Los resultados indican que los mantenedores sin experiencia realizaron tareas de mantenimiento más rápido en una versión refactorizada que los expertos en la versión original, sin que ello afectara la calidad del código. Esto sugiere que los patrones de diseño ayudan a organizar y clarificar el código, facilitando el trabajo incluso para quienes no tienen mucha experiencia. La refactorización, por lo tanto, no solo mejora la calidad del software, sino que también aumenta la eficiencia del mantenimiento.

## Bibliografía

ELA Baniassad, GC Murphy y C. Schwanninger, "Gráficos racionales de patrones de diseño: vinculando el diseño con el código fuente", en Actas de la 25.<sup>a</sup> <https://dl.acm.org/doi/abs/10.1145/1181775.1181778>

# Sobre el modelado de la recuperación de la arquitectura de software como correspondencia de grafos

Este artículo propone un modelo de correspondencia de grafos para la recuperación de la arquitectura de software. Utiliza grafos para representar tanto el sistema de software como su arquitectura conceptual, buscando identificar cómo transformar el grafo de patrones (representación de la arquitectura de alto nivel) en un subgrafo del sistema real. Una correspondencia exitosa reestructura el sistema para que se ajuste al patrón dado, mientras que una correspondencia fallida señala áreas donde el sistema no cumple con las restricciones.



## Reflexión

El modelo de correspondencia de grafos propuesto ofrece un enfoque interesante para recuperar y reestructurar arquitecturas de software de manera más precisa y organizada. Permite identificar de forma clara los puntos de mejora y las violaciones de restricciones en el sistema. Sin embargo, la efectividad de este enfoque depende de la calidad y la generación adecuada del grafo de patrones, así como de la capacidad del proceso para manejar grandes sistemas de forma incremental.

## Bibliografía

MA Eshera y KS Fu. Una medida de similitud entre gráficos relativos atribuidos para el análisis de imágenes. En Séptima Conferencia Internacional sobre Reconocimiento de Patrones, páginas 75-77, 1984. <https://ieeexplore.ieee.org/abstract/document/1235425>

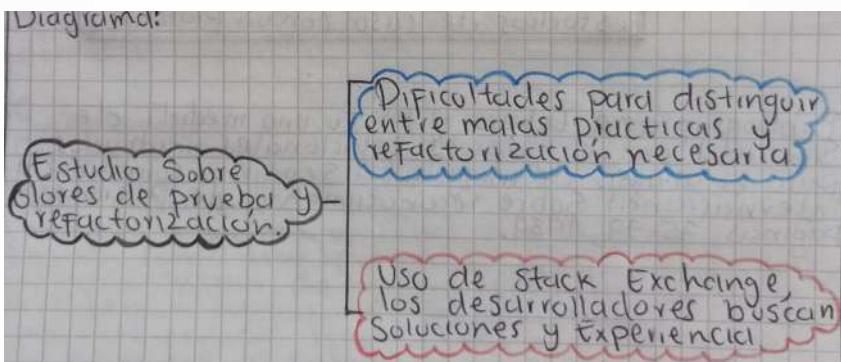
# Escuchando la voz de los expertos: revelando el conocimiento de las comunidades de Stack Exchange sobre los olores de las pruebas

Este artículo presenta un estudio sobre cómo los desarrolladores discuten los olores de prueba y las refactorizaciones de prueba en la plataforma Stack Exchange. Los olores de prueba son indicios de malas prácticas en el código de prueba, y las refactorizaciones buscan mejorar el diseño sin cambiar la funcionalidad. El estudio muestra que los desarrolladores frecuentemente recurren a Stack Exchange para obtener consejos sobre cómo manejar estos olores y refactorizar el código de prueba.

## Reflexión

El estudio resalta una realidad común en el desarrollo de software: los desarrolladores enfrentan dificultades al identificar y corregir olores de prueba. La falta de claridad sobre si estos olores son simples "malas prácticas" o si requieren refactorización puede generar confusión y llevar a soluciones subóptimas. Es relevante que las plataformas como Stack Exchange se conviertan en espacios clave para compartir experiencias y soluciones, pero también es necesario que se promuevan enfoques más sistemáticos y guiados para abordar estos problemas.

Diagrama:



## Bibliografía

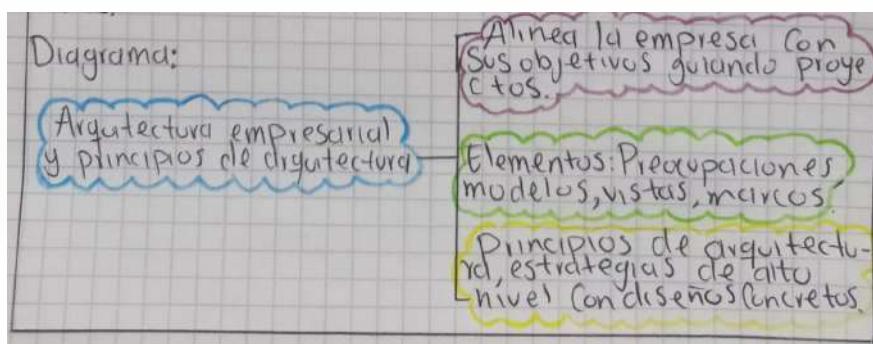
<https://ieeexplore.ieee.org/abstract/document/10164665>

# El papel de la arquitectura empresarial

Este capítulo explora el papel de la arquitectura empresarial y los principios que la sustentan. La arquitectura empresarial busca alinear la empresa con sus objetivos esenciales, actuando como una restricción normativa que guía el diseño de proyectos y programas de transformación. Los elementos clave de la arquitectura incluyen las preocupaciones, los modelos, las vistas, los principios y los marcos. Los principios de arquitectura son fundamentales, ya que conectan las intenciones estratégicas de alto nivel con los diseños concretos, proporcionando una base sólida en medio del cambio constante.

## Reflexión

La arquitectura empresarial desempeña un papel crucial en garantizar que los esfuerzos de transformación de una organización estén alineados con sus objetivos estratégicos. Los principios de arquitectura actúan como un ancla, ofreciendo directrices claras en un entorno de constante cambio. Sin estos principios, los proyectos de transformación pueden perder enfoque o desviarse de las metas fundamentales.

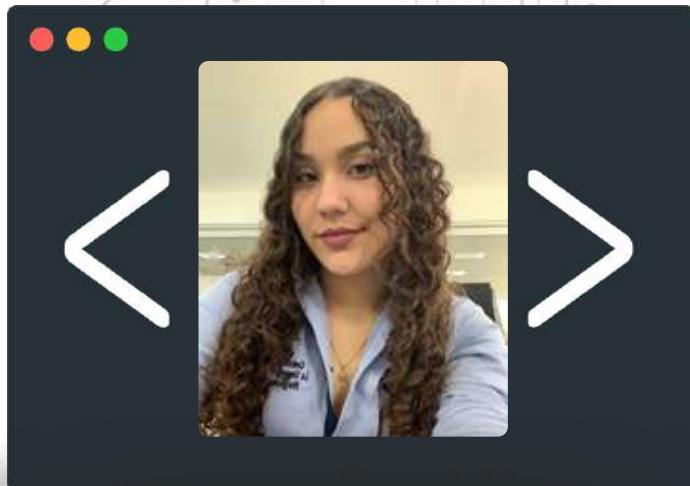


## Bibliografía

El término sistema debe entenderse aquí en su sentido original (Ashby 1956 ; Bunge 1979 ), incluidos los "sistemas de sistemas", como las empresas en su conjunto. El campo de la TI parece haber secuestrado el término sistema, al tiempo que lo ha convertido en sinónimo de sistema de aplicación o de software.

[https://link.springer.com/chapter/10.1007/978-3-642-20279-7\\_2](https://link.springer.com/chapter/10.1007/978-3-642-20279-7_2)

# **Mayra Alejandra Tamayo Perdomo**



Mi nombre es Mayra Alejandra Tamayo, tengo 18 años y soy aprendiz del Servicio Nacional de Aprendizaje (SENA), cursando el tecnólogo en Análisis y Desarrollo de Software. Tengo experiencia en Java con Spring Boot para desarrollar aplicaciones robustas, así como conocimientos básicos en C# y manejo de Python utilizando Django para construir aplicaciones web organizadas.

En bases de datos, domino MySQL y SQL Server, gestione bases de datos relacionales. Me apasiona la parte visual del desarrollo de software, trabajando con HTML, CSS y JavaScript para crear interfaces intuitivas y atractivas.

Estoy en constante aprendizaje, enfocándome en aportar valor a los proyectos con soluciones funcionales y visualmente destacadas.

# Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

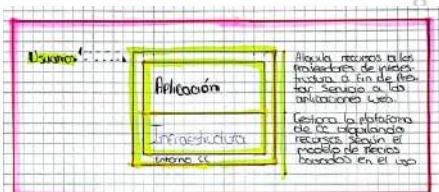
presenta un enfoque integral para el diseño de arquitecturas de software, especialmente para aplicaciones web. Se destaca la importancia de utilizar arquitecturas genéricas que proporcionen una estructura común para resolver problemas específicos, lo que ayuda a los arquitectos a evitar conflictos derivados de la falta de experiencia. El trabajo introduce un metamodelo de componentes arquitectónicos que identifica elementos clave en el diseño, y se complementa con una herramienta gráfica para la instanciación de estos componentes. Además, se aborda la verificación de patrones de diseño para asegurar su correcta aplicación, lo que contribuye a la calidad del software. El modelo UML se utiliza para describir diferentes aspectos de la arquitectura, incluyendo la aplicación en la nube y la descomposición en capas. Finalmente, se enfatiza que la experiencia del arquitecto es crucial para seleccionar las instancias adecuadas que cumplan con los requerimientos funcionales y de calidad.

## Bibliografía

- Blas, M. J., Leone, H., & Gonnet, S. (2019). Modelado y verificación de patrones de diseño de arquitectura de software para entornos de computación en la nube. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, 35, 1-17. [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://n9.cl/9e1no1](https://n9.cl/9e1no1)

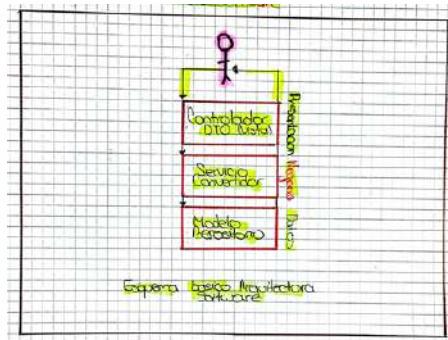
## Reflexión

El enfoque presentado en el documento es altamente relevante en el contexto actual de la computación en la nube, donde la complejidad de las aplicaciones web está en constante aumento. La propuesta de un entorno de diseño integral que combina un metamodelo con herramientas de verificación es una contribución valiosa, ya que no solo facilita el trabajo de los arquitectos de software, sino que también promueve la creación de aplicaciones más robustas y de calidad. Además, la atención a la experiencia del arquitecto resalta la necesidad de formación continua en un campo que evoluciona rápidamente. En general, este trabajo puede servir como una guía útil para profesionales y académicos interesados en mejorar sus prácticas de diseño en entornos de computación en la nube.



# Desarrollo de una herramienta para el aprendizaje de patrones de diseño software

Trata sobre el desarrollo de una herramienta para ayudar a estudiantes a aprender patrones de diseño de software. La herramienta está diseñada para ser fácil de usar y se centrará en patrones importantes que los estudiantes deben conocer. Se menciona que el aprendizaje continuo es clave, ya que muchos de los conceptos son nuevos para los alumnos. Además, se destaca la importancia de planificar y gestionar el tiempo durante el desarrollo del proyecto. Se utilizaron varias herramientas tecnológicas para crear la aplicación, como Microsoft Office y herramientas de gestión de proyectos. También se realizaron encuestas para entender las dificultades que enfrentan los estudiantes al aplicar estos patrones. La herramienta busca facilitar la incorporación de los alumnos al mundo laboral, ayudándoles a aplicar lo aprendido de manera efectiva.



## Reflexión

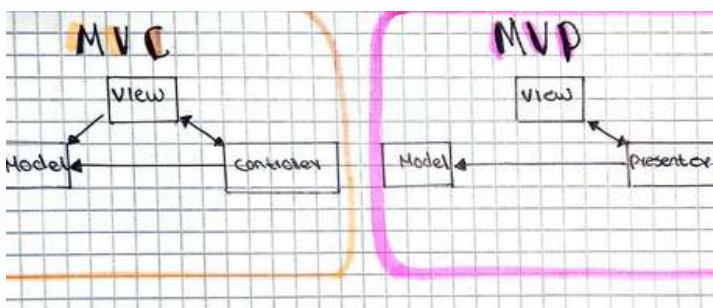
La herramienta propuesta es una contribución valiosa para enseñar patrones de diseño en software, combinando teoría y práctica de manera interactiva. Facilita el aprendizaje, mejora la comprensión de conceptos complejos y prepara a los estudiantes con habilidades prácticas para su futuro profesional. Este enfoque puede impulsar un desarrollo de software más eficiente y de calidad, transformando positivamente la enseñanza en el ámbito académico.

## Bibliografía

- Ferrandis, (2021). Desarrollo de una herramienta para el aprendizaje de patrones de diseño software. RIUNET. Sanabria, F. R., & Rodríguez, S. V. (2021). Evaluación de una Arquitectura de Software. Prospectiva, 19(2). <https://n9.cl/p47y4>

# Patrones de diseño de software aplicado a las aplicaciones web

El documento revisa los patrones de diseño de software aplicados a aplicaciones web, resaltando la creciente complejidad en su desarrollo debido a las demandas de los usuarios. Se establece la Ingeniería Web como subdisciplina de la Ingeniería de Software, enfocada en sistemas interactivos de alta calidad. Se analizan casos como el de Turkish Radio Television Corporation, que mostró deficiencias por no usar patrones de diseño. Se propone el uso de patrones como MVC para mejorar la estructura y funcionalidad del software. La investigación concluye que la implementación de patrones mejora la usabilidad, el mantenimiento y la organización del desarrollo.



## Reflexión

La revisión presentada en el documento es fundamental en el contexto actual del desarrollo web, donde la complejidad y las expectativas de los usuarios están en constante aumento. La identificación de problemas en sistemas que no utilizan patrones de diseño resalta la importancia de estas prácticas en la creación de software eficiente y funcional. Además, la propuesta de rediseñar aplicaciones utilizando patrones como MVC es un enfoque acertado que puede mejorar significativamente la calidad del software. La investigación también destaca la relevancia de una metodología rigurosa para seleccionar artículos, lo que refuerza la validez de los hallazgos. En mi opinión, este trabajo no solo contribuye al campo académico, sino que también ofrece valiosas lecciones prácticas para los desarrolladores de software, promoviendo un enfoque más estructurado y efectivo en la creación de aplicaciones web.

## Bibliografía

Universidad Señor de Sipán (USS). (2020). Patrones de diseño de software aplicado a las aplicaciones web. Repositorio USS. Cristiá, M. (2021). Una Teoría para el Diseño de Software. <https://n9.cl/dr0m7>

# Desarrollo de una arquitectura de software para el robot móvil Lázaro

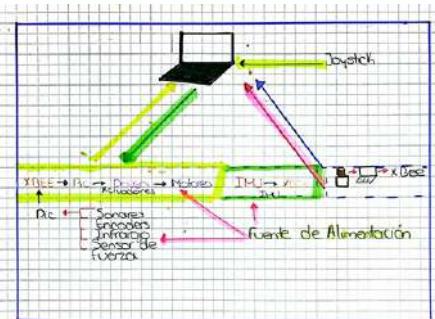
Presenta una innovadora estructura de software diseñada para optimizar el control y la operación del robot Lázaro. La arquitectura se compone de tres niveles que permiten la gestión eficiente de los actuadores y el monitoreo de los sensores, utilizando librerías implementadas en lenguaje C#. Se incorporan diversos sensores, como sonares y un telémetro láser, para la detección de obstáculos y la medición de fuerzas. La comunicación entre el robot y un computador remoto se realiza a través de módulos XBee®, permitiendo un control tanto autónomo como teleoperado. Además, se discuten las ventajas de arquitecturas deliberativas y reactivas, así como la implementación de herramientas de inteligencia artificial para mejorar la navegación en entornos no estructurados. La evaluación cualitativa de la arquitectura muestra resultados positivos en términos de flexibilidad, reactividad y facilidad de uso, aunque se identifican áreas de mejora, como el aprendizaje de tareas. En conclusión, la arquitectura propuesta es escalable y amigable para los usuarios, lo que la hace adecuada para las necesidades operativas del robot.

## Bibliografía

Scielo. (2018). Desarrollo de una arquitectura de software para el robot móvil Lázaro. Revista de Ingeniería. [https://www.scielo.cl/scielo.php?pid=S0718-33052018000300376&script=sci\\_arttext](https://www.scielo.cl/scielo.php?pid=S0718-33052018000300376&script=sci_arttext)

## Reflexión

La propuesta de una arquitectura de software para el robot Lázaro es un avance significativo en el campo de la robótica móvil. La integración de múltiples sensores y la capacidad de control remoto ofrecen un enfoque versátil y adaptable a diferentes entornos. Además, la combinación de arquitecturas deliberativas y reactivas, junto con el uso de inteligencia artificial, demuestra un entendimiento profundo de los desafíos que enfrentan los robots en situaciones dinámicas. Sin embargo, sería interesante ver cómo se pueden abordar las limitaciones en el aprendizaje de tareas, lo que podría abrir nuevas posibilidades para la autonomía del robot. En general, este trabajo no solo contribuye al desarrollo del robot Lázaro, sino que también sienta las bases para futuras investigaciones en arquitecturas de software para robots móviles.

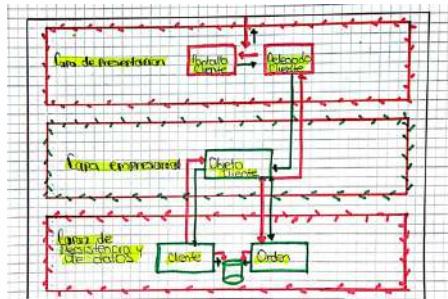


# Arquitectura de software, esquemas y servicios

El texto resalta la importancia de la arquitectura orientada a servicios (SOA) en el desarrollo de aplicaciones empresariales, facilitando la integración entre sistemas y ofreciendo flexibilidad y seguridad. Sin embargo, plantea desafíos como la gestión de conexiones y la prevención de fallos operativos. Se destaca la necesidad de diseñar servicios autónomos, con políticas de seguridad y manejo de excepciones. Además, se enfatiza la reusabilidad y el bajo acoplamiento de componentes, y la importancia de soluciones estandarizadas para mejorar la eficiencia. Finalmente, se proponen estrategias para gestionar dependencias, como la centralización del monitoreo de rendimiento.

## Bibliografía

- Uniroja. (2006). Arquitectura de software, esquemas y servicios. Revista Dialnet. Cambiarieri, 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://n9.cl/tdp9p>

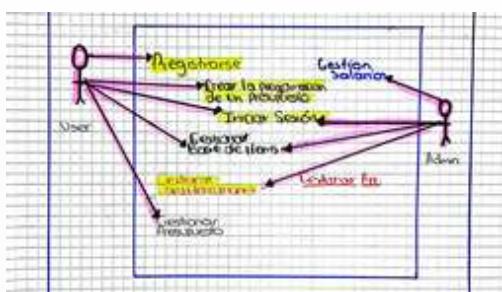


## Reflexión

Me parece interesante porque explica bien los beneficios de la arquitectura orientada a servicios (SOA), como la flexibilidad y la integración de sistemas. También me parece acertado que hable de los desafíos, como la complejidad en la gestión y los fallos, porque eso muestra una visión equilibrada. Además, la propuesta de centralizar el monitoreo de rendimiento me parece muy práctica para mejorar la eficiencia en sistemas distribuidos. En general, creo que es un enfoque completo y útil para entender el impacto de SOA en el desarrollo de software.

# Arquitectura de software para el desarrollo de herramienta Tecnológica de Costos, Presupuestos y Programación de obra

El artículo "Arquitectura de software para el desarrollo de herramienta Tecnológica de Costos, Presupuestos y Programación de obra" de Cárdenas-Gutiérrez, Barrientos-Monsalve y Molina-Salazar, se centra en el diseño de un software académico para la gestión de costos y presupuestos en Ingeniería Civil. Se organiza la información relacionada con materiales, mano de obra, maquinaria y transporte en grupos de procesos, buscando aumentar la productividad y facilitar la elaboración de presupuestos. La metodología incluye la creación de una Estructura de División del Trabajo (EDT) y el uso del método de la ruta crítica para la programación de obras. Se presentan casos de uso del software, que incluyen la gestión de presupuestos y salarios. La propuesta tiene como objetivo modernizar la enseñanza y mejorar la calidad educativa, ofreciendo una herramienta accesible y gratuita para los estudiantes. Los autores no tienen conflictos de intereses y el artículo concluye que esta herramienta puede preparar mejor a los estudiantes para el mercado laboral.



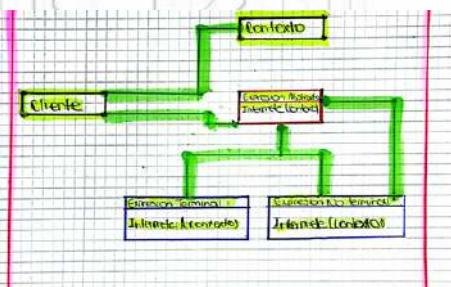
## Bibliografía

- Redalyc.org. (2022). Arquitectura de software para el desarrollo de herramienta tecnológica de costos, presupuestos y programación de obra. Revista Tecnológica, Guarin, J. S. V. (2023). Especificando una arquitectura de software: Software architecture specification. Tecnología Investigación y Academia, 11(2), 170-181. <https://n9.cl/d1v0w>

## Reflexión

Me parece interesante porque propone un software práctico y accesible, pensado para facilitar la gestión de costos, presupuestos y programación. Me gusta que utilice metodologías como la EDT y el método de la ruta crítica, ya que ayudan a conectar la teoría con la práctica. Además, que sea gratuito lo hace aún más valioso. Solo me pregunto cómo se asegura su actualización y si ya ha sido probado con usuarios.

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte



Examina el estado actual de los lenguajes de patrones en la arquitectura de software. Se analizan sus orígenes, avances y aplicaciones en la construcción de arquitecturas en diversos dominios, destacando su importancia para diseñadores y desarrolladores. Se revisa la evolución de la ingeniería de software desde los años 60, enfatizando la necesidad de un enfoque arquitectónico para mejorar la calidad y mantenibilidad del software. El artículo menciona a Christopher Alexander y Frank Buschmann, quienes contribuyeron al desarrollo de patrones en la arquitectura civil y de software, respectivamente. Se presentan diferentes formas de categorizar patrones y ejemplos de su aplicación en áreas como la gestión de identidades y el desarrollo de aplicaciones e-business. Se concluye que los lenguajes de patrones son herramientas valiosas para resolver problemas arquitectónicos de manera eficiente y se sugiere investigar más sobre metodologías en su creación.

## Reflexión

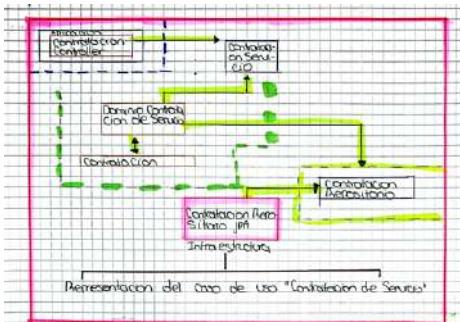
El artículo ofrece una visión clara sobre los lenguajes de patrones en la arquitectura de software, destacando su evolución y relevancia en el desarrollo de sistemas mantenibles y de alta calidad. Conecta sus orígenes, desde los aportes de Christopher Alexander en la arquitectura civil hasta su adaptación por Frank Buschmann al ámbito del software, lo que proporciona una base histórica sólida. Además, presenta ejemplos prácticos, como su aplicación en e-business y gestión de identidades, lo que refleja la versatilidad de estos patrones. La invitación a investigar metodologías para su creación fomenta el avance en su uso sistemático, resaltando la importancia de estos lenguajes como soluciones eficientes para problemas arquitectónicos complejos.

## Bibliografía

- Redalyc.org. (2014). Lenguajes de patrones de arquitectura de software: Una aproximación al estado del arte. Redalyc. Blancarte, O. (2016). Introducción a los Patrones de Diseño. México, México DF. <https://n9.cl/btd5y>

# Implementación de una Arquitectura de Software guiada por el Dominio

Presenta un enfoque de implementación de arquitecturas de software guiadas por el dominio, enfatizando el diseño dirigido por el dominio (DDD). Se propone transformar una arquitectura de software típica en una arquitectura hexagonal, centrada en el dominio del negocio. Esto permite separar la complejidad del negocio de la lógica técnica, promoviendo una mejor comunicación entre expertos en dominio y desarrolladores. Se discuten los conceptos de contextos delimitados y lenguaje ubicuo, fundamentales para el DDD. Además, se valida el enfoque a través de un caso de estudio sobre una plataforma de empleo, donde se ejemplifica cómo aplicar estos principios en un entorno real. La investigación concluye que esta transformación mejora la mantenibilidad y escalabilidad del software.



## Reflexión

El enfoque presentado en el artículo es muy relevante en el contexto actual del desarrollo de software, donde la complejidad y la rapidez de cambio son constantes. La adopción de una arquitectura guiada por el dominio no solo facilita una mejor alineación con los objetivos de negocio, sino que también promueve una colaboración más efectiva entre diferentes partes interesadas. La transformación hacia una arquitectura hexagonal, que favorece la independencia de tecnologías, es una estrategia inteligente para asegurar la adaptabilidad a futuros cambios y mejorar la calidad del software. Sin embargo, la implementación de estos conceptos puede ser desafiante y requiere un compromiso significativo por parte de todo el equipo de desarrollo.

## Bibliografía

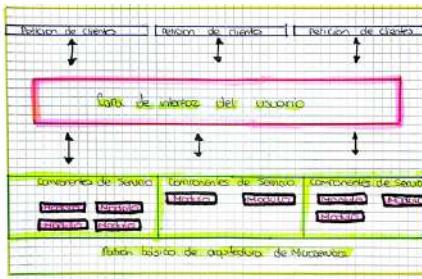
Universidad Nacional de La Plata (UNLP). (2022). Implementación de una arquitectura de software guiada por el dominio. Repositorio UNLP. Prieto, C. A., & Madrid, D. A. (2022). *Buenas prácticas en la construcción de software: Best practices in software construction*. Tecnología Investigación y Academia, 10(2), 149-166. <https://n9.cl/hnmdef>

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

El documento analiza la transición de la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador de una arquitectura monolítica a una basada en microservicios. La CGTIC enfrenta problemas de mantenimiento y escalabilidad debido a su enfoque monolítico. El estudio busca identificar tecnologías y metodologías que faciliten esta transición, destacando las ventajas de los microservicios, como el desarrollo flexible y autónomo. Se abordan servicios web como SOAP y REST, integración continua (CI), y contenedores como Docker. La conclusión enfatiza la necesidad de adoptar arquitecturas modernas para mejorar la calidad y modularidad de las aplicaciones.

## Bibliografía

2019). Arquitectura de software basada en microservicios para desarrollo de aplicaciones web. Repositorio Institucional. Carignano, M. C. (2016, June). Representación y razonamiento sobre las decisiones de diseño de arquitectura de software. In XVIII Workshop de Investigadores en Ciencias de la Computación (WICC 2016, Entre Ríos, Argentina). <https://n9.cl/ug8br>



## Reflexión

El estudio es relevante, ya que aborda un cambio crucial en la forma de desarrollar software en la CGTIC, proponiendo soluciones que pueden mejorar significativamente la eficiencia y flexibilidad de sus aplicaciones. La transición a microservicios representa una oportunidad no solo para resolver problemas actuales, sino también para alinearse con prácticas modernas que fomentan la innovación y agilidad en el desarrollo. Considero que la adopción de tecnologías como Docker y la implementación de integración continua son pasos esenciales para asegurar un proceso de desarrollo más robusto y adaptado a las necesidades del entorno dinámico actual.

# Análisis comparativo de Patrones de Diseño de Software

Aborda la importancia de los patrones de diseño como soluciones estandarizadas a problemas comunes en el desarrollo de software, destacando su capacidad para evitar la duplicación de código y facilitar la reutilización. Se analizan cinco patrones clave: Template Method, Model-View-Controller (MVC), Model-View-Presenter (MVP), Model Front Controller y Model-View-ViewModel (MVVM), evaluando sus componentes, ventajas y desventajas, así como su aplicabilidad en diferentes contextos. La investigación revela que no existe un patrón superior, dado que cada uno cumple un propósito específico, y concluye que los patrones de diseño son fundamentales para mejorar la organización, el mantenimiento y la calidad del código en sistemas de software, al permitir una estructura coherente que facilita su comprensión y evolución en el tiempo. Este análisis comparativo proporciona a los desarrolladores un recurso valioso para seleccionar el patrón que mejor se adapte a las necesidades de sus proyectos, promoviendo así prácticas de programación más efectivas y sostenibles.

## Reflexión

El estudio presenta una perspectiva valiosa sobre los patrones de diseño, resaltando su relevancia en la creación de software modular y mantenable. Considero que esta investigación es útil para desarrolladores, ya que les proporciona un marco claro para elegir el patrón adecuado según las necesidades del proyecto. La claridad en las ventajas y desventajas de cada patrón ayuda a entender cuál puede ser más efectivo en contextos específicos, promoviendo prácticas de programación más eficientes y menos propensas a errores.

## Bibliografía

- Danilo, O., Patricia, N., & Vinicio, M. (2022). Análisis comparativo de patrones de diseño de software. *Polo del Conocimiento: Revista Científico-Profesional*, 7(7), 2146-2165. <https://dialnet.unirioja.es/descarga/articulo/9042927.pdf>

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

El estudio analiza 68 proyectos de desarrollo de software, de los cuales el 47.1% fue eliminado por no cumplir con los estándares. En los 36 proyectos restantes, se identificaron patrones de diseño de la "Gang of Four" (GoF), siendo los patrones creacionales como Singleton (67%) y Factory Method (50%) los más comunes. Entre los patrones estructurales, Decorator y Facade fueron utilizados en el 39% y 36%, respectivamente, y el patrón de comportamiento más frecuente fue Iterator (56%). Se reconocieron 8 de los 23 patrones GoF, lo que representa un 35% de uso. Se propone una metodología para identificar estos patrones y se sugiere crear un catálogo más accesible para facilitar su comprensión y aplicación.

## Bibliografía

Scielo. (2021). Patrones de diseño GOF (The Gang of Four) en el contexto de procesos de desarrollo de aplicaciones orientadas a la web. Revista INFOTEC. Valbuena, D. C. Modelo de gestión de servicios PKI. <https://n9.cl/55lwus>

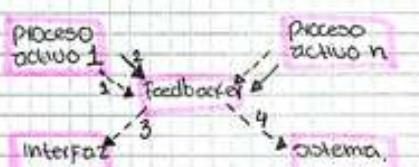
no	Patrón de diseño	Categoría
1	Factory	Creacionales
2	Factory Method	Creacionales
3	Singleton	Creacionales
4	Decorator	Estructurales
5	Iterator	Comportamiento
6	Visitor	Comportamiento
7	Strategy	Comportamiento
8	Composite	Estructurales

## Reflexión

La investigación muestra lo importante que son los patrones de diseño en el desarrollo de software. Aunque un 35% de los patrones de la "Gang of Four" se utilizan, hay un gran desconocimiento que impide que se usen más. Además, el hecho de que el 47.1% de los proyectos no cumplan con los estándares indica que necesitamos una cultura de calidad más fuerte en la industria. Patrones como Singleton y Factory Method son muy conocidos, pero para usarlos bien, los desarrolladores necesitan formación práctica. Crear un catálogo que sea fácil de entender es fundamental para ayudar a la gente a aprender y aplicar estos patrones. Si mejoramos la educación y el acceso a recursos, los profesionales podrán hacer soluciones más eficientes y de mejor calidad. En resumen, promover el uso de patrones de diseño podría cambiar la manera en que se desarrollan los proyectos de software.

# Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico

El texto "Patrones de Usabilidad" de Ana M. Moreno y M. Isabel Sánchez analiza cómo la arquitectura del software se relaciona con la usabilidad. Propone un método que incluye aspectos de usabilidad desde el principio del desarrollo. En el documento se presentan patrones de usabilidad, que son ideas generales para mejorar cosas como el aprendizaje, la eficiencia y la satisfacción del usuario. Esta investigación forma parte del proyecto europeo STATUS y sugiere un ciclo de evaluación y mejora que se realiza antes de construir el sistema. También se identifican características de usabilidad que ayudan a aplicar estos patrones en el diseño del software. Al prever las necesidades de usabilidad, este enfoque intenta disminuir la cantidad de cambios necesarios más adelante, lo que hace que el desarrollo de aplicaciones sea más efectivo y centrado en el usuario.



## Bibliografía

- Moreno, A. (2024). Patrones de usabilidad: Mejora de la usabilidad del software desde el momento arquitectónico. ResearchGate. [López, D., & Maya, E. \(2017\). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web. <https://n9.cl/0tbddk>](#)

## Reflexión

El texto anterior presenta una forma nueva y esencial de ver el desarrollo de software, al resaltar lo importante que es la usabilidad desde el principio. Incluir la usabilidad en el diseño no solo hace que la experiencia del usuario sea mejor, sino que también ayuda a encontrar y solucionar problemas antes de que se vuelvan costosos. Mostrar patrones de usabilidad ofrece herramientas útiles que pueden ayudar a los desarrolladores a crear aplicaciones más fáciles de usar y efectivas.

# Revisión sistemática sobre generadores de código fuente y patrones de arquitectura

El texto es un estudio detallado titulado "Estudio sistemático sobre generadores de código y patrones arquitectónicos", realizado por María Rosario Huari Casas y presentado como una tesis en la Pontificia Universidad Católica del Perú en 2020. Este trabajo se centra en los generadores de código y los patrones arquitectónicos, analizando su importancia y uso en el desarrollo de software. La revisión sistemática tiene como objetivo reunir y evaluar la información existente sobre estos temas, ofreciendo una perspectiva completa de las tendencias y prácticas actuales. También se examinan las ventajas y desventajas de usar generadores de código y patrones arquitectónicos en proyectos de software. Este documento puede ser útil para investigadores y profesionales en el área de la ingeniería de software. El texto completo puede estar disponible en bibliotecas o se puede comprar a través de ProQuest.

## Bibliografía

ProQuest. (2024). Revisión sistemática sobre generadores de código fuente y patrones de arquitectura. ProQuest. <https://acortar.link/CYBZpK>

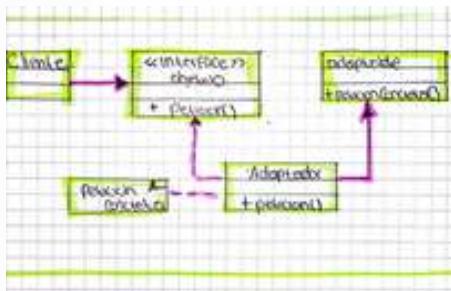


## Reflexión

Una de las cosas más interesantes del estudio es cómo se centra en los generadores de código. Estas herramientas ayudan a hacer tareas repetitivas de manera automática, lo que no solo ahorra tiempo, sino que también disminuye la posibilidad de cometer errores. Esto es especialmente útil en proyectos grandes, donde es muy importante ser consistente y preciso. La revisión que hizo Huari Casas da una buena idea de cómo se pueden usar estas herramientas de manera efectiva en diferentes situaciones. Los patrones ofrecen soluciones que ya han funcionado para problemas que se repiten, lo que ayuda a que los desarrolladores se entiendan mejor y fomenta buenas prácticas en el diseño.

# Perfiles UML para definición de Patrones de Diseño

El texto habla sobre cómo se utilizan los perfiles UML para definir y documentar patrones de diseño en la programación orientada a objetos. Los patrones de diseño son soluciones que se repiten para problemas comunes en el desarrollo de software. Los perfiles UML ayudan a ampliar la sintaxis y el significado de UML, lo que hace más fácil modelar elementos específicos de diferentes áreas. Mediante estereotipos, valores etiquetados y restricciones, estos perfiles crean un lenguaje común para describir los patrones de diseño de una manera más clara. El enfoque principal del artículo es la definición de patrones estructurales, comenzando con el patrón "Composite". Se menciona que las herramientas UML que ya existen son suficientes para usar estos perfiles sin necesidad de herramientas adicionales. También se propone que en el futuro se especifiquen más patrones y se busquen herramientas UML que hagan esta tarea más sencilla.



## Bibliografía

Universidad Nacional de La Plata (UNLP). (2024). Perfiles UML para definición de Patrones de Diseño. Repositorio UNLP. Capel, M. I., Grimán, A. C., & Garví, E. Calidad Ágil: Patrones de Diseño en un contexto de Desarrollo Dirigido por Pruebas. : <https://n9.cl/6wk3a>

## Reflexión

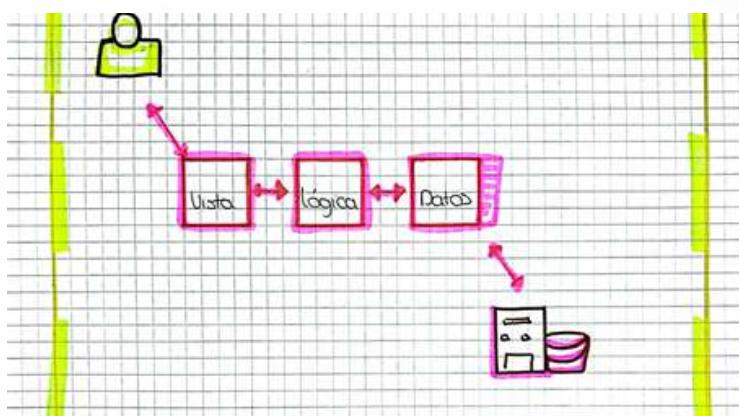
El anterior texto nos habla de la importancia de los perfiles UML, los cuales son herramientas muy útiles para definir y documentar patrones de diseño en la programación orientada a objetos. Ayudan a crear soluciones que ya han sido probadas y que son eficientes, lo que mejora la calidad del software. Al ampliar lo que se puede hacer con el lenguaje de modelado, los desarrolladores pueden mostrar sus proyectos de forma clara y detallada. Usar estereotipos, valores etiquetados y restricciones mejora la comunicación en los equipos, especialmente en proyectos complicados. Además, es fácil de adoptar, ya que las herramientas UML modernas permiten su uso sin necesidad de software extra.

# Artículo de software académica para la comprensión del desarrollo de software en capas

El modelo propuesto utiliza una arquitectura de software en capas para prevenir problemas a corto y largo plazo. Se presentan dos estructuras: una de tres capas (Vista, Lógica y Datos) y una más detallada de siete capas. Con el patrón Modelo-Vista-Controlador (MVC), se gestionan las interacciones entre la interfaz y las capas, promoviendo la separación de responsabilidades. Se describen clases como Cliente y TelefonoDatos, que implementan operaciones sobre la base de datos. El artículo concluye que adoptar arquitecturas de software mejora la flexibilidad y sostenibilidad, sugiriendo mejoras como la automatización de mapeadores y control de transacciones.

## Reflexión

El texto resalta la importancia de las arquitecturas de software en capas, utilizando el patrón Modelo-Vista-Controlador (MVC) para organizar y mejorar la reutilización y el mantenimiento del código. Presenta ejemplos prácticos, como las clases Cliente y TelefonoDatos, para ilustrar su aplicación. Aunque es un enfoque sólido, se sugiere profundizar en la automatización de mapeadores y el control de transacciones para mejorar la flexibilidad del sistema. En general, el texto subraya cómo una arquitectura bien estructurada favorece la creación de sistemas más sostenibles y fáciles de mantener.

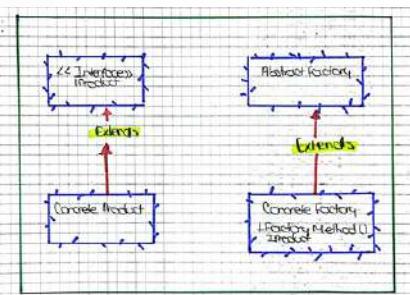


## Bibliografía

Google Scholar. (2024). Arquitectura de software académica para la comprensión del desarrollo de software en capas. Google Scholar. Garnero, A. B., & Horenstein, N. Utilización de antipatrones y patrones en el análisis de software. <https://n9.cl/ae8jq6>

# Introducción a los patrones de diseño

El texto contextualiza históricamente el origen de los patrones, desde la arquitectura hasta su integración en la programación orientada a objetos. Los patrones se dividen en tres categorías: creacionales, estructurales y de comportamiento, con ejemplos prácticos para facilitar su aplicación. Se destaca la importancia de usar patrones como soluciones comprobadas, evitando la reinvenCIÓN. El libro incluye un repositorio en GitHub para descargar y practicar con el código fuente. Combinando teoría y práctica, la obra es útil tanto para principiantes como para programadores experimentados, ayudando a mejorar la calidad y eficiencia del software.



## Reflexión

La obra de Oscar Blancarte representa un recurso invaluable para aquellos que buscan profundizar en los patrones de diseño, ya que integra teoría con ejemplos prácticos y aplicables. Su enfoque en situaciones del mundo real es particularmente pertinente, dado que muchos textos sobre este tema suelen emplear ejemplos abstractos que pueden resultar confusos. Además, el autor consigue transmitir conceptos complejos de forma clara, lo que facilita la comprensión tanto para principiantes como para expertos. La adición de un repositorio en GitHub es un excelente complemento, permitiendo a los lectores interactuar con el código y fortalecer su aprendizaje. En conclusión, se trata de un libro bien estructurado que aporta de manera significativa a la formación en ingeniería de software.

## Bibliografía

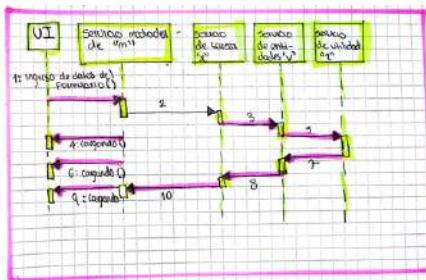
- Academia.edu. (2019). Introducción a los patrones de diseño.
- Academia.edu. Marticorena, R., López, C., García Osorio, C. I., & Pardo, C. (2002). Aprendizaje práctico de patrones de diseño en asignaturas de programación de nivel III. <https://n9.cl/4cuslk>

# Identificación y clasificación de patrones de diseño de servicios web para mejorar QoS

La tesis de Lic. Pablo Aguerre, dirigida por Dr. Gustavo Rossi, aborda la identificación y clasificación de patrones de diseño de servicios web para mejorar la Calidad de Servicio (QoS). Se destaca la importancia de los servicios web en la arquitectura de software, ya que permiten la interoperabilidad entre aplicaciones diversas. La investigación propone un inventario de patrones de diseño que facilite metodologías asegurando estándares de calidad en la implementación de servicios. La tesis se organiza en capítulos que incluyen definiciones, componentes de servicios web, trabajos relacionados y un catálogo de patrones aplicables. El objetivo final es optimizar la calidad en la nube y mejorar la eficiencia de las APIs, contribuyendo así a un desarrollo más efectivo y alineado con las necesidades del negocio.

## Bibliografía

Scielo. (2024). Identificación y clasificación de patrones de diseño de servicios web para mejorar QoS. Repositorio UNLP.  
Moreno, J. C., Marciszack, M. M., & Groppe, M. A. (2020). Patrones de Usabilidad Temprana en el Modelo Conceptual. AJEA, (5), <https://n9.cl/ewi5om>



## Reflexión

La tesis aborda un tema crucial en la ingeniería de software moderna, dado el creciente uso de servicios web y su impacto en los negocios. La sistematización de patrones de diseño es una contribución valiosa que puede guiar a desarrolladores y arquitectos en la creación de soluciones más robustas y eficientes. Además, la atención a la QoS es especialmente pertinente en un contexto donde la experiencia del usuario y la fiabilidad son fundamentales. Sin embargo, sería interesante ver cómo se aplican estos patrones en casos prácticos y qué resultados se obtienen en escenarios reales.

# Desarrollo de sistemas de software con patrones de diseño orientado a objetos

aborda la implementación de patrones de diseño orientados a objetos en la creación de un sistema de software denominado "Intranet Industrial", desarrollado en la Facultad de Ingeniería Industrial de la Universidad Nacional Mayor de San Marcos. Se analiza el impacto económico de la adopción de estos patrones en comparación con un enfoque que no los considera. En la primera sección, se introducen conceptos teóricos y se clasifican los patrones más reconocidos en el ámbito industrial. A continuación, se detallan los patrones aplicados en el desarrollo del sistema y se presentan de manera formal. La sección final ofrece una descripción del sistema, sus módulos y herramientas, así como una estimación de costos utilizando el patrón "Informador" y el método COCOMO. Se concluye que la implementación de patrones contribuye a mejorar la eficiencia y a disminuir los costos en el desarrollo de software.



## Reflexión

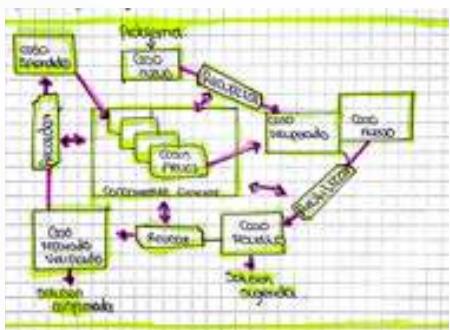
El enfoque de utilizar patrones de diseño en el desarrollo de software es una estrategia altamente efectiva que no solo optimiza el proceso de creación, sino que también promueve la reutilización y la escalabilidad. Este trabajo es un excelente ejemplo de cómo integrar teoría y práctica, mostrando que la implementación de patrones no solo es beneficiosa desde el punto de vista técnico, sino que también tiene repercusiones positivas en la gestión de costos. La investigación resalta la importancia de adoptar estándares en el desarrollo de sistemas complejos, ofreciendo un modelo que puede servir de referencia para futuros proyectos en el ámbito académico e industrial.

## Bibliografía

Desarrollo de sistemas de software con patrones de diseño orientado a objetos". (2024, 11 de noviembre). Cybertesis UNMSM. Rojas, M., & Montilva, J. (2011). Una arquitecturaology and Computational

# Módulo de recomendación de patrones de diseño para EGPat

Los problemas de diseño pueden dificultar la comprensión de estos recursos, haciendo que no cumplan con sus objetivos. Para abordar esto, se recomienda el uso de patrones de diseño, que mejoran la estructura y reusabilidad de los recursos educativos. Sin embargo, la selección de patrones es complicada debido a la falta de mecanismos adecuados en los repositorios que almacenan estos recursos. Muchos diseñadores no son conscientes de la existencia de estas fuentes, y el proceso de búsqueda puede ser tedioso y consumir mucho tiempo, especialmente si deben consultar múltiples repositorios. Como respuesta a estos problemas, el Grupo de Tecnologías de Apoyo a la Educación (GITAE) de la Universidad de las Ciencias Informáticas ha desarrollado el Entorno para la Gestión de Patrones de Diseño (EGPat). A pesar de estos avances, todavía persisten desafíos en la búsqueda y adaptación de patrones a las necesidades específicas de los diseñadores.



## Bibliografía

- Scielo. (2024). Módulo de recomendación de patrones de diseño para EGPat. Revista Scielo.
- Paez, A. H., Falcón, J. D., & Cruz, A. A. P. (2018). Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D. Revista de I+D tecnológico, 14(1), 54-64  
<https://n9.cl/dbt4a>

## Reflexión

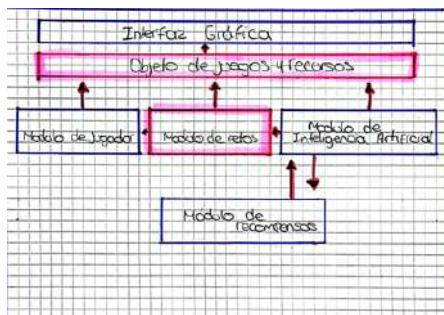
La creciente complejidad de los recursos educativos digitales subraya la importancia de implementar patrones de diseño efectivos, que mejoren la usabilidad y reduzcan el tiempo de búsqueda de soluciones. Herramientas como EGPat son un paso positivo al centralizar información y recomendar patrones. Sin embargo, es crucial que las soluciones no solo se enfoquen en la accesibilidad, sino también en la formación de diseñadores, ya que la falta de conocimiento y la complejidad de la búsqueda son barreras a superar. La inclusión de inteligencia artificial para personalizar recomendaciones es un avance, aunque plantea dudas sobre su adaptabilidad y calidad.

# Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D

El uso de las TIC ha impulsado la industria de videojuegos, abarcando disciplinas como programación, diseño y marketing. Motores como Unity 3D simplifican tareas complejas, pero en la UCI se identificaron problemas como la falta de organización y reutilización limitada de componentes. Para resolver esto, se diseñó una arquitectura de software que organiza las características funcionales básicas de los videojuegos. Un prototipo de videojuego de plataformas validó la propuesta, mejorando la reutilización y optimización de recursos. La solución cumple con atributos de calidad como reusabilidad y extensibilidad, y fue evaluada mediante el método ATAM y pruebas de escenarios.

## Bibliografía

Universidad de Ciencias Informáticas (UCI). (2018). Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D. Repositorio UCI. González, J. E. F., & García, C. A. O. Propuesta de diseño de arquitectura de software para la gestión, análisis y procesamiento de datos de neurociencia. <https://n9.cl/xz9s7>

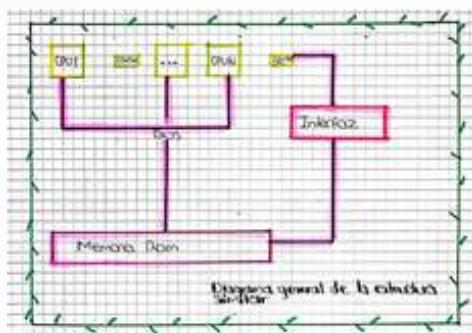


## Reflexión

El desarrollo de videojuegos combina tecnología y creatividad para generar soluciones innovadoras, pero enfrenta retos en la organización y estructuración del proceso. Contar con herramientas y metodologías sólidas es crucial para optimizar recursos y garantizar la calidad del producto final. El diseño de arquitecturas de software específicas facilita la reutilización de componentes, mejora la eficiencia y fomenta la sostenibilidad a largo plazo de los proyectos. Esta investigación es clave para formar desarrolladores capacitados y mantener la competitividad de la industria, asegurando productos que cumplan con las expectativas de los usuarios en un mercado exigente.

# Uso de patrones de diseño de software: un caso práctico

Los patrones de diseño son esenciales en la ingeniería de software, ya que ofrecen soluciones comprobadas a problemas comunes, promoviendo un desarrollo eficiente y estructurado. Sus beneficios incluyen la reutilización del código, reducción de la complejidad, desacoplamiento de componentes y mejora del mantenimiento del software. En un experimento en la Universidad de Costa Rica, estudiantes aplicaron patrones de diseño en un simulador de procesador multinúcleo MIPS, lo que mejoró la funcionalidad, mantuvo la fidelidad con arquitecturas reales y facilitó el mantenimiento y la extensibilidad. Los resultados mostraron que los patrones mejoran la calidad del software, aunque aumentan la complejidad del código y las pruebas.



## Bibliografía

- Redalyc. (2012). Uso de patrones de diseño de software: Un caso práctico. Revista Redalyc. Martín, Y. E. (2012). Arquitectura de software. Arquitectura orientada a servicios. Serie Científica de la Universidad de las Ciencias Informáticas, 5(1), 1-10. <https://n9.cl/p7pho>

## Reflexión

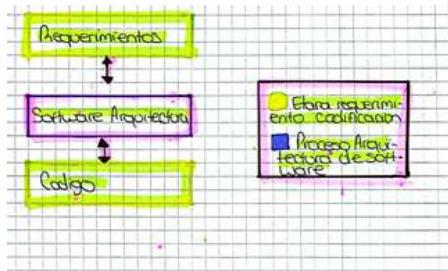
Los patrones de diseño son un pilar esencial en la formación de ingenieros de software y científicos de la computación. Incorporar su enseñanza desde etapas tempranas en las carreras universitarias es crucial, ya que promueve el desarrollo de habilidades prácticas que trascienden los métodos tradicionales de enseñanza. Este enfoque no solo capacita a los futuros profesionales para abordar problemas comunes de manera eficiente, sino que también fomenta una mentalidad orientada al diseño y a la adaptabilidad, características clave en un entorno tecnológico en constante cambio.

# Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software

La arquitectura de software se centra en diseñar y organizar sistemas, considerando componentes y sus relaciones. Los patrones ayudan a los desarrolladores a alcanzar objetivos comunes. Los lenguajes de descripción arquitectónica (ADL) como UniCon, Wright y Rapide permiten modelar y analizar arquitecturas antes de implementarlas, optimizando la reutilización y evolución del sistema. Los componentes y conectores son clave en las arquitecturas, facilitando un enfoque modular para el diseño, desarrollo y mantenimiento de sistemas complejos. Las arquitecturas de referencia (ARS), como marcos estándar, mejoran la reutilización y adaptabilidad, promoviendo prácticas más eficientes en la industria del software.

## Bibliografía

Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software". (2019). Scielo. <https://acortar.link/zeeKAc>



## Reflexión

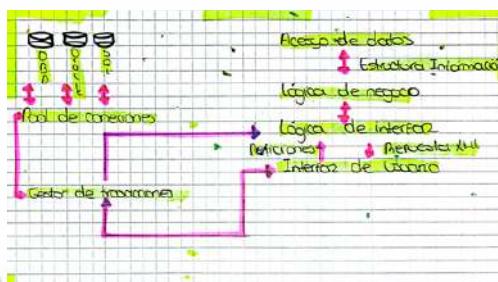
La arquitectura de software es clave para conectar las necesidades del negocio con la implementación técnica, proporcionando una visión modular del sistema. Es fundamental adaptar las herramientas de modelado y análisis a las necesidades del proyecto y usar arquitecturas de referencia para reducir costos y tiempos de desarrollo. Sin embargo, existen desafíos como la falta de mecanismos para la evolución dinámica y la reutilización efectiva en algunos lenguajes. Comprender los componentes y conectores es esencial para crear sistemas sólidos y sostenibles, fomentando la colaboración interdisciplinaria y enfrentando los desafíos de un entorno cambiante.

# Atributos de Calidad y Arquitectura de Software

En este artículo logramos construir una arquitectura de software efectiva, la cual requiere balancear múltiples atributos de calidad (como seguridad, mantenibilidad, portabilidad y rendimiento) sin comprometer desmedidamente otros. Las decisiones tomadas durante el diseño tienen un impacto significativo en el resultado final. Aplicar una metodología que considere las diferentes perspectivas, documente de manera integral y priorice las necesidades de los stakeholders permite crear arquitecturas que sean tanto funcionales como de calidad, aunque no garantiza estos atributos automáticamente. La práctica de este enfoque se valida mediante casos reales expuestos en el curso. Sin embargo, incluso con un diseño sólido, garantizar los atributos de calidad requiere validación constante y adaptabilidad durante todo el ciclo de vida del sistema.

## Reflexión

La arquitectura de software es un proceso estratégico que influye directamente en el éxito y la calidad de un sistema, enfrentando el desafío de equilibrar atributos como rendimiento, seguridad y mantenibilidad, que a menudo entran en conflicto. Este equilibrio requiere un enfoque sistemático, apoyado por metodologías y herramientas para analizar el sistema desde diversas perspectivas y documentar los compromisos asumidos. El uso de vistas arquitectónicas y estilos bien definidos facilita la comunicación con los stakeholders y guía las decisiones informadas, manteniendo la coherencia en el desarrollo. A pesar de un diseño sólido, garantizar los atributos de calidad requiere validación continua y adaptabilidad a lo largo del ciclo de vida del sistema.



## Bibliografía

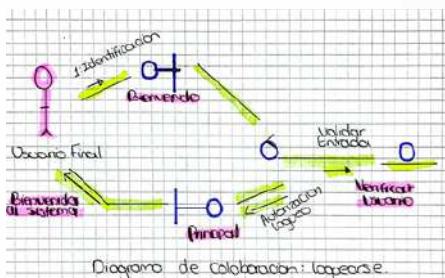
Universidad Politécnica de Madrid (UPM). (1905). Atributos de calidad y arquitectura del software. Repositorio UPM. Cortez, A. A., & Naveda, C. A. Una propuesta de implementación para especificaciones de patrones de comportamiento. <https://n9.cl/x6wz7>

# Herramientas para reusó de código JavaScript orientado a partir de interacción

El desarrollo de sistemas informáticos es una tarea compleja debido a los diversos problemas que pueden surgir a lo largo de todo el proceso, desde el análisis hasta la implementación. A lo largo del tiempo, han surgido diversas herramientas de diseño rápido de aplicaciones (RAD) que facilitan la creación de interfaces de usuario, como CBuilder, Visual Basic y Delphi. La usabilidad de una interfaz es crucial, ya que determina la efectividad y satisfacción del usuario al interactuar con la aplicación. La reutilización de estos patrones ayuda a acelerar el desarrollo, reducir costos y mejorar la calidad del software. Un ejemplo relevante de la aplicación de esta metodología es ReusMe, una herramienta que permite generar componentes web reutilizables en JavaScript basados en patrones de interacción. Esta aplicación facilita la creación de interfaces gráficas de usuario personalizadas, ofreciendo soluciones probadas y optimizadas para diseñadores y desarrolladores.

## Reflexión

El uso de patrones de interacción en el desarrollo de interfaces de usuario y aplicaciones web es clave para mejorar la eficiencia, reducir el tiempo de desarrollo y garantizar productos de calidad. Al adoptar patrones reutilizables, los diseñadores evitan "reinventar la rueda" y aprovechan soluciones probadas. Herramientas como ReusMe destacan el valor de la reutilización de código en interfaces gráficas, facilitando la personalización y adaptación de componentes sin comprometer la usabilidad ni funcionalidad. En resumen, emplear patrones y reutilizar código son pasos fundamentales para optimizar el desarrollo y ofrecer productos más robustos y accesibles.



## Bibliografía

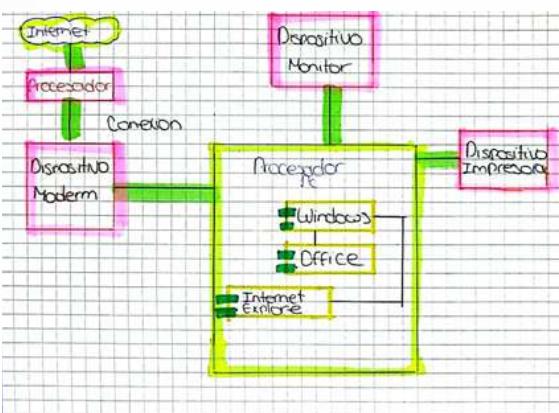
- Redalyc. (2024). Herramienta para reuso de código JavaScript orientado a patrones de interacción. Revista Redalyc.  
<https://www.redalyc.org/pdf/4277/427739438009.pdf>

# Una Teoría para el Diseño de Software

El diseño de software es fundamental porque al dividir un sistema en componentes que interactúan entre sí, facilita su desarrollo y mantenimiento. Este enfoque asigna funciones a cada elemento, establece sus relaciones y describe su estructura, reduciendo costos y complejidad a lo largo del ciclo de vida del software. El proceso se divide en dos etapas: desarrollo (33% del esfuerzo) y mantenimiento (67%), siendo esta última la más costosa debido a los cambios, correcciones y mejoras que requieren re-testeo. Un buen diseño anticipa estos cambios, permitiendo realizarlos con menor costo sin afectar la integridad del sistema.

## Reflexión

Refexionar sobre el diseño de software resalta la importancia de planificar y estructurar un sistema desde el inicio. Aunque es tentador comenzar directamente con la programación, hacerlo sin un diseño previo puede generar altos costos y problemas a lo largo del ciclo de vida del software. El principio de "Diseño para el Cambio" enfatiza que el software debe ser flexible y evolucionar con el tiempo. En resumen, un buen diseño no solo es una cuestión técnica, sino una estrategia económica que marca la diferencia entre un sistema costoso de mantener y uno adaptable a las necesidades futuras.



## Bibliografía

Universidad Nacional de Rosario (UNR). (2024). Una teoría para el diseño de software. Repositorio UNR, <https://www.fciai.unr.edu.ar/ingsoft/intro-diseno.pdf>

# Lenguajes de patrones de diseño de software bajo una perspectiva cognoscitivista

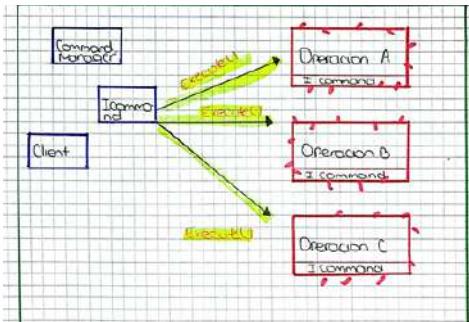
Este artículo nos ayuda analizar la aplicación del concepto de "lenguaje de patrones" en el diseño de software, basándose en las ideas de Alexander (1979) y su relación con la estructura de sistemas de patrones. Aunque el término "lenguaje de patrones" se ha utilizado ampliamente, aún persisten dudas teóricas y prácticas sobre su significado y aplicación. El trabajo propone un marco teórico cognoscitivo para interpretar y mejorar el uso de estos lenguajes. Se establece una distinción clara entre los conceptos de "lenguaje de patrones", "estructura de lenguaje de patrones", "catálogos de patrones" y "sistemas de patrones". Además, se sugiere que el diseño de un lenguaje de patrones debe enfocarse en su organización y evolución para mejorar su eficacia en el campo del desarrollo de software.

## Bibliografía

Redalyc. (2023). Lenguajes de patrones de diseño de software bajo una perspectiva cognoscitivista. Revista Redalyc. <https://www.redalyc.org/articulo.oa?id=66640712>

## Reflexión

Este artículo destaca la complejidad y la relevancia de los lenguajes de patrones en el diseño de software, señalando que, aunque ampliamente utilizados, aún carecen de una base teórica sólida. Esta falta de claridad puede dificultar su implementación efectiva. La propuesta de un marco cognoscitivo para organizar y mejorar estos lenguajes refleja la necesidad de estructurar mejor el conocimiento sobre patrones. Además, se enfoca en la evolución del conocimiento sobre patrones, lo que puede ser clave para avanzar en el desarrollo de software. El trabajo también sugiere nuevas direcciones para futuras investigaciones, resaltando la importancia de seguir innovando en la ingeniería de software.

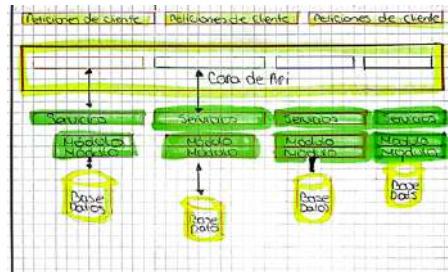


# Patrones de diseño para mejorar la accesibilidad y uso de aplicaciones sociales para adultos mayores.

Este artículo propone un conjunto de 36 modelos incompletos para el diseño de interacción en aplicaciones sociales dirigidas a personas mayores, basados en esfuerzos previos y normas de diseño. Su objetivo es identificar problemas de usabilidad en estas interfaces y fortalecer los métodos existentes al incluir descripciones de anomalías y soluciones alternativas. Se utilizan técnicas de "evaluación heurística" para obtener la percepción del usuario sobre el diseño. El enfoque aborda tanto la perspectiva técnica como la del grupo social de personas mayores, y los resultados demuestran que el modelo facilita la creación de interfaces bien diseñadas, mejorando la experiencia de usuario y la calidad de vida de los mayores.

## Bibliografía

Redalyc. (2015). Patrones de diseño para mejorar la accesibilidad y uso de aplicaciones sociales para adultos mayores. Revista Redalyc.  
<https://www.redalyc.org/articulo.oa?id=15839609009>

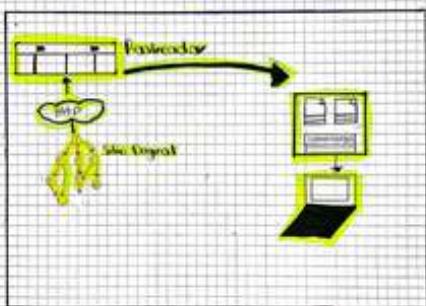


## Reflexión

Este artículo destaca la importancia de diseñar interfaces accesibles y usables para personas mayores, quienes a menudo enfrentan barreras cognitivas y físicas al utilizar aplicaciones sociales. Al integrar modelos de interacción y técnicas de evaluación heurística, se propone un enfoque que considera tanto la perspectiva técnica como las necesidades sociales de este grupo, promoviendo la adopción tecnológica y mejorando su calidad de vida. Este enfoque no solo optimiza la experiencia de usuario, sino que también favorece el bienestar emocional y social, facilitando la conectividad y participación en la sociedad digital, reduciendo la exclusión digital y empoderando a los mayores.

# Una arquitectura basada en software libre para archivos web

La información en la web es extensa pero puede ser eliminada. Un documento web está compuesto por archivos HTML y otros elementos. Desde los años 90 se han creado sistemas para preservar sitios web, conocidos como archivos web. En Latinoamérica y Venezuela hay falta de iniciativas de preservación web. Se busca diseñar una arquitectura que facilite la preservación web utilizando software libre. Se realizó un estudio de métodos y herramientas de preservación web a nivel mundial. Se seleccionaron los más adecuados para la arquitectura propuesta, priorizando el uso de software libre. Se está desarrollando un prototipo para probar la integración y rendimiento de estos componentes. La preservación de la información web es crucial para garantizar el acceso continuo a recursos digitales y proteger el patrimonio cultural. El objetivo final es proporcionar un marco de referencia para implementar sistemas de preservación web efectivos y sostenibles en Venezuela y Latinoamérica.



## Bibliografía

- Redalyc. (2013). Una arquitectura basada en software libre para archivos web. Revista Redalyc. <https://www.redalyc.org/articulo.oa?id=82326270005>

## Reflexión

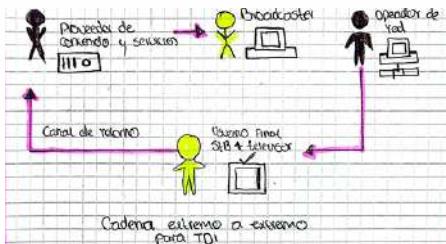
La preservación de la información web es esencial para garantizar el acceso continuo a recursos digitales y proteger el patrimonio cultural. Desde los años 90, existen sistemas de archivos web, pero en Latinoamérica y Venezuela hay una falta de iniciativas en este ámbito. Este artículo propone diseñar una arquitectura para la preservación web utilizando software libre, basada en un estudio global de métodos y herramientas de preservación. Se seleccionaron las mejores opciones para integrar en la arquitectura propuesta y se está desarrollando un prototipo para evaluar su integración y rendimiento. El objetivo es crear un marco de referencia para implementar sistemas de preservación efectivos y sostenibles en la región.

# Arquitectura de Software para el Soporte de Comunidades Académicas Virtuales en Ambientes de Televisión Digital Interactiva

El artículo presenta una arquitectura de software diseñada para apoyar comunidades académicas virtuales (CAV) en entornos de televisión digital interactiva (TDI). Esta arquitectura integra servicios de la Web 2.0 a través de un esquema REST-JSON, permitiendo una interacción dinámica entre los usuarios y los contenidos multimedia. Se exploran varios escenarios de uso, como foros y salas de chat, que facilitan la educación virtual y promueven la participación de los estudiantes. Además, se enfatiza la importancia del canal de retorno para mejorar la comunicación y el acceso a servicios, tanto relacionados como no relacionados con el contenido. La arquitectura ha sido implementada en un laboratorio, mostrando su potencial para optimizar la experiencia educativa y contribuir a la reducción de la brecha digital en la educación. En resumen, la propuesta busca transformar el aprendizaje en entornos digitales mediante la interactividad y la flexibilidad de los servicios ofrecidos.

## Bibliografía

- Scielo. (2024). Arquitectura de software para el soporte de comunidades académicas virtuales en ambientes de televisión digital interactiva. Revista Scielo. 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://n9.cl/6s49x>



## Reflexión

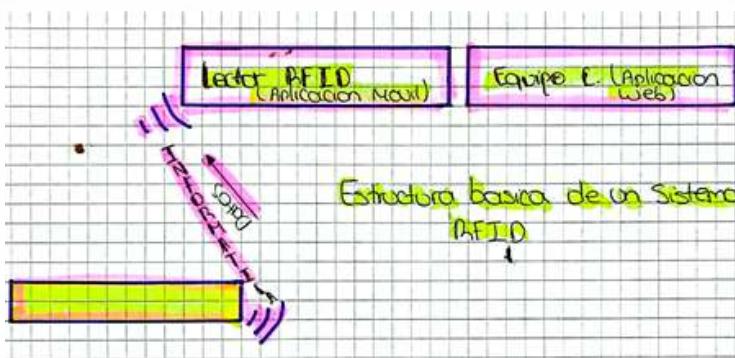
La iniciativa de integrar comunidades académicas virtuales con la televisión digital interactiva es muy pertinente, especialmente en contextos donde el acceso a Internet es limitado. La propuesta de una arquitectura que utilice estándares como REST-JSON promete flexibilidad y escalabilidad, lo que podría enriquecer la experiencia educativa. Además, la capacidad de interactuar de manera dinámica con los contenidos puede motivar a los usuarios a participar más activamente. Sin embargo, la implementación efectiva y la accesibilidad de los dispositivos necesarios seguirán siendo desafíos importantes a superar para garantizar que esta tecnología beneficie a un amplio espectro de estudiantes.

# Arquitectura de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia

El sistema RFID propuesto para el Instituto Tecnológico de Orizaba busca mejorar el control de inventarios al resolver problemas como la falta de concordancia entre los activos registrados y los reales, así como el tiempo de actualización lento. Utilizando tecnología RFID, el sistema facilita el registro, actualización y localización de los activos de manera más eficiente. Su arquitectura de software incluye una base de datos XML para integrar la aplicación web y la aplicación RFID, y un módulo RFID que detecta objetos etiquetados mediante señales de radiofrecuencia. Esto incrementa la seguridad, actualiza el inventario en tiempo real y reduce errores humanos y robos.

## Reflexión

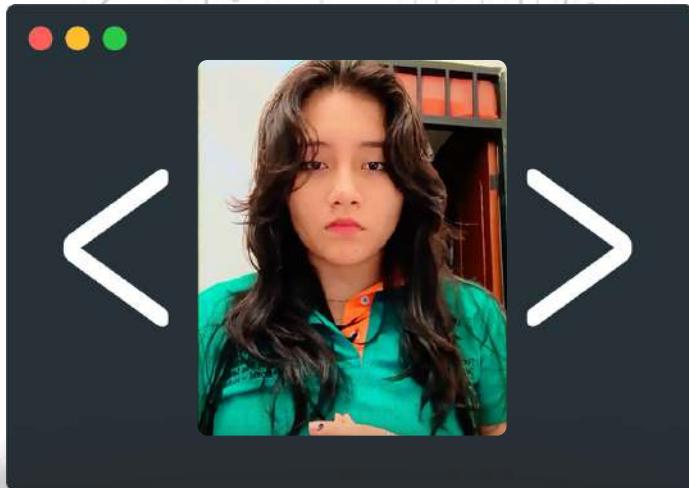
El uso de la tecnología RFID en el Instituto Tecnológico de Orizaba mejora la gestión de inventarios al permitir una actualización, consulta y localización más precisa de los productos, reduciendo errores humanos. Esta tecnología incluye aplicaciones web y móviles, automatizando procesos antes manuales y mejorando la seguridad y eficiencia. Además de optimizar la administración de materiales, también facilita una mejor organización interna y un uso más efectivo de los recursos. Sin embargo, su implementación presenta desafíos como los costos iniciales y la necesidad de capacitar al personal, factores clave para su adopción exitosa en instituciones.



## Bibliografía

- Redalyc. (2015). Arquitectura de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia. Revista Redalyc. <https://www.redalyc.org/articulo.oa?id=512251501004>

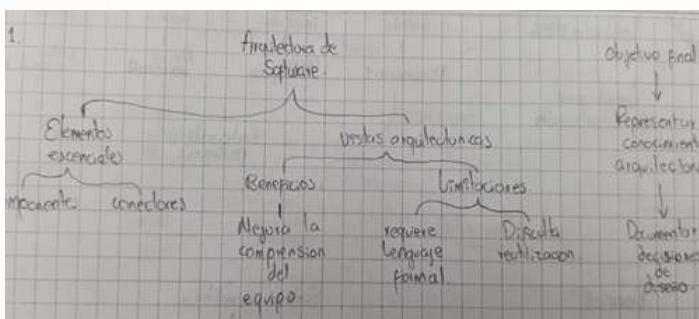
# Patricia Sarmiento



Mi nombre completo es Patricia del Rocio Sarmiento Ruiz, tengo 18 años, cumple el 3 de abril, me encuentro actualmente cursando un tecnólogo en Análisis y Desarrollo de Software , en lo que a mí respecta, siento que no tengo las habilidades o la lógica suficiente para continuar estudiando software, por lo cual me eh decidido a estudiar en la universidad algo que siempre quise estudiar, que de momento es innecesario comentar, el desarrollo de software me parece muy útil , ya que actualmente la tecnología es lo que casi domina a la humanidad , y saber manejar esa tecnología, es casi manejar al ser humano en sí, que suele ser ignorante, o es lo que me han comentado bastante , ya que mientras las personas se la pasan horas y horas en redes sociales y en aplicaciones o páginas web, los creadores de esas páginas y aplicativos , les muestran la información que quieren que vean para tenerlos entretenidos y ocupados en cosas sin importancia y que no dan productividad, pero algo que yo si digo es, que el desarrollo de software no es para todos.

# Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software

El artículo resalta cómo la Arquitectura de Software es esencial en el desarrollo, definiendo la estructura general de un sistema y mostrando cómo interactúan sus componentes. Según el Instituto de Ingeniería de Software, es como un mapa que muestra las partes y sus conexiones sin meterse en detalles. La investigación encontró que los elementos más importantes en estas arquitecturas son los componentes y los Conectores, que se repiten en muchos modelos y lenguajes. También se descubrió que las "vistas arquitectónicas" son la herramienta clave para ayudar a los equipos a entenderse mejor.



## Bibliografía

Bibliografía Miguel Angel Sánchez Palmero, Nemury Silega Martínez , Olga Yarisbel Rojas Grass, 26/10/2018

## Reflexión

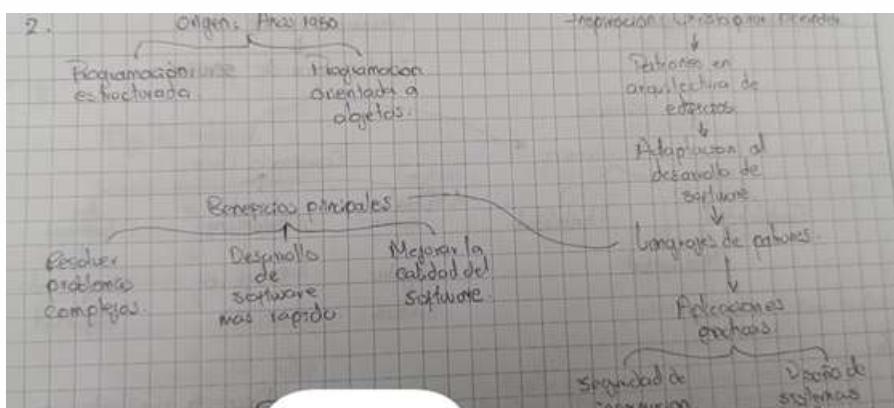
La arquitectura de software es crucial para estructurar sistemas complejos. Incorporar vistas arquitectónicas fomenta la comunicación entre equipos, aunque su implementación requiere lenguajes formales que pueden complicar la automatización. Superar estas limitaciones permitirá documentar decisiones de diseño con mayor precisión.

# Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte

En el artículo "Lenguajes de Patrones de Arquitectura de Software: Una Aproximación al Estado del Arte" leemos cómo los lenguajes de patrones han cambiado la forma en que se piensa la arquitectura de software. Desde los inicios de la ingeniería de software en los años 1950 con la programación estructurada y la orientación a objetos hasta ahora... contribuyó a crear la base de los patrones de diseño que usamos hoy. Inspirados por el arquitecto Christopher Alexander (que habló de "patrones" en la arquitectura de edificios) estos conceptos se adaptaron al mundo del software para encontrar soluciones que se pudieran repetir.

## Reflexión

Los lenguajes de patrones son herramientas esenciales para resolver problemas recurrentes en diseño de software. Su influencia histórica demuestra cómo estas soluciones sistemáticas mejoran la calidad y rapidez del desarrollo, particularmente en áreas como seguridad y e-Business.



## Bibliografía

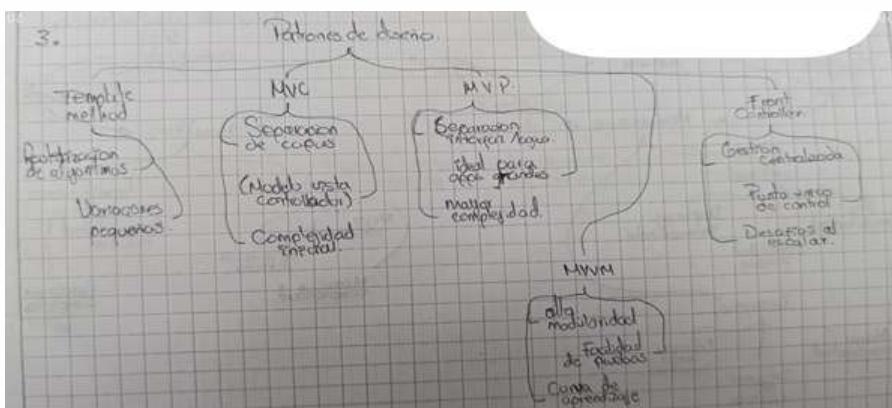
Bibliografía, Victor Hugo Jimenez-Torres , Wilman Tello-Borja , Jorge Iván Rios-Patiño, 4/12/2014

# Análisis comparativo de Patrones de Diseño de Software

El documento compara cinco patrones de diseño de software (Template Method, MVC, MVP, Front Controller y MVVM) y explica cómo ayudan a organizar y mantener el código, evitando duplicaciones. No hay un "mejor" patrón; cada uno tiene sus pros y contras. Por ejemplo, Template Method es ideal para reutilizar algoritmos con pequeñas variaciones. MVC es popular por separar bien la lógica, la vista y el modelo (aunque puede ser un poco difícil al principio). MVP separa la interfaz y la lógica (genial para apps grandes) pero es más complejo. Front Controller gestiona todas las solicitudes desde un solo punto, mejorando así la seguridad.

## Reflexión

Mi conclusión es que existen varios patrones de diseño que nos pueden ayudar a organizar el código de nuestro proyecto, pero al final, debemos analizar bien, cuáles son las necesidades de nuestro programa o proyecto, pasa así mismo poder elegir un patrón para usar.



## Bibliografía

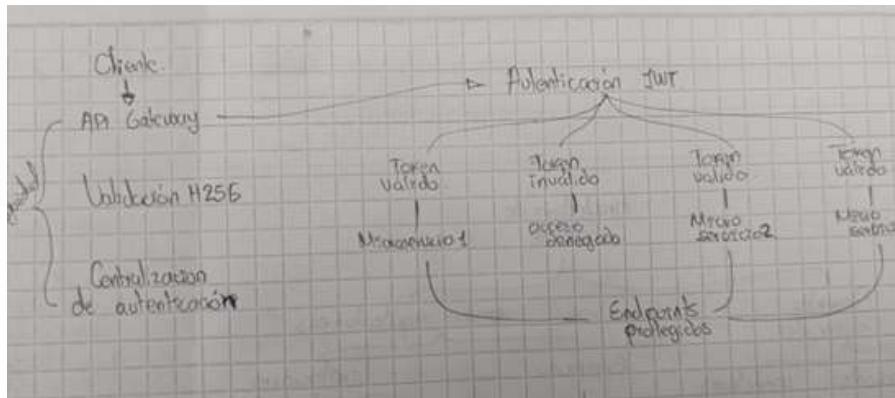
Bibliografía, Gavilánez Álvarez Oscar Danilo, Layedra Natalia, Ramos Vinicio, 28/07/2022

# Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software

La tesis aborda los desafíos de seguridad en arquitecturas de microservicios, proponiendo el patrón Microservice Security Pattern API Gateway (MSPAG), que utiliza JSON Web Tokens (JWT) y el algoritmo H256 para centralizar la autenticación y proteger los endpoints a través de un API Gateway. Incluye un marco teórico, análisis del estado del arte e implementación práctica en lenguajes como C#, Python y Java, evaluando su eficacia mediante pruebas. Concluye que la adaptación de patrones de seguridad como MSPAG fortalece la resistencia de los sistemas basados en microservicios, garantizando integridad y confidencialidad.

## Reflexión

Seleccionar patrones arquitectónicos adecuados es clave para optimizar tiempos, costos y calidad. Proveer herramientas prácticas y bien estructuradas facilita establecer bases sólidas para proyectos desde su inicio.



## Bibliografía

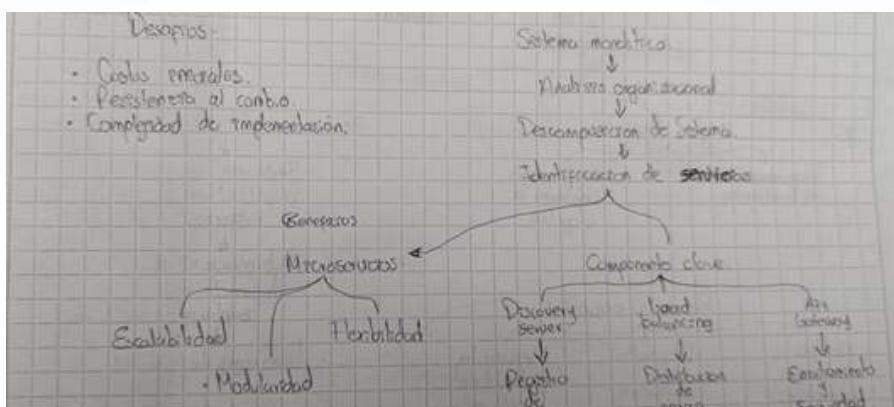
Hernandez Guzman, Karen Monica; 2023-10-27

# Propuesta metodológica para migración de sistemas web con arquitectura monolítica hacia una arquitectura basada en microservicios.

El documento propone una metodología para migrar sistemas web de arquitectura monolítica a microservicios, destacando beneficios como escalabilidad y modularidad, pero subrayando la necesidad de una descomposición cuidadosa. Introduce el esquema SMMicro, que aborda análisis organizacional, descomposición de sistemas, y la construcción de un ecosistema de microservicios con componentes como Discovery Server y Load Balancing. La metodología es validada en un caso de estudio en la Escuela Politécnica Nacional, evidenciando mejoras en flexibilidad y escalabilidad.

## Reflexión

Migrar de arquitecturas monolíticas a microservicios aporta escalabilidad y modularidad, pero requiere planificación estratégica. Este enfoque es ideal para sistemas grandes, aunque implica costos iniciales y resistencia al cambio.



## Bibliografía

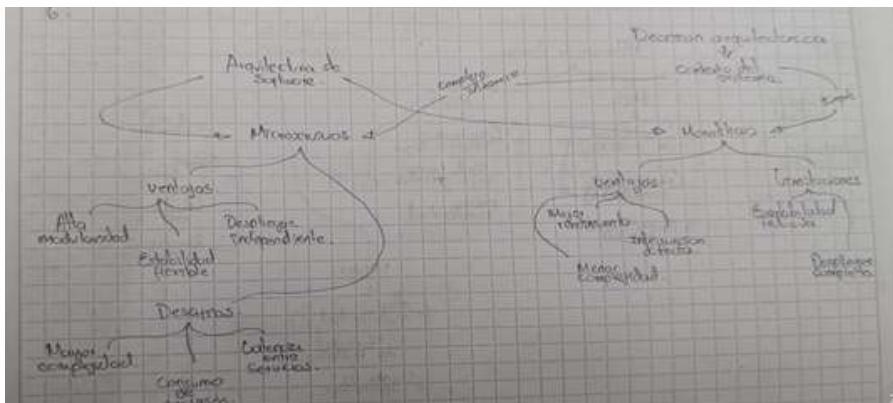
Arboleda Cola, Carlos Augusto; 11-dic-2017

# Son los microservicios la mejor opción? Una evaluación de su eficacia y eficiencia frente a los monolitos

El documento compara la eficacia y eficiencia de arquitecturas monolíticas y de microservicios, destacando que los monolitos ofrecen mejor rendimiento en contextos simples debido a su integración directa, mientras que los microservicios destacan en modularidad, escalabilidad y despliegue independiente, aunque implican mayor complejidad y consumo de recursos. Utilizando herramientas como Docker y JMeter, las pruebas mostraron que los monolitos tienen tiempos de respuesta más rápidos y menor tasa de error, mientras que los microservicios son más adecuados para sistemas grandes y dinámicos.

## Reflexión

La elección depende del contexto. Los monolitos ofrecen simplicidad y eficiencia para sistemas pequeños, mientras que los microservicios son mejores para proyectos grandes y dinámicos, priorizando la escalabilidad y flexibilidad.



## Bibliografía

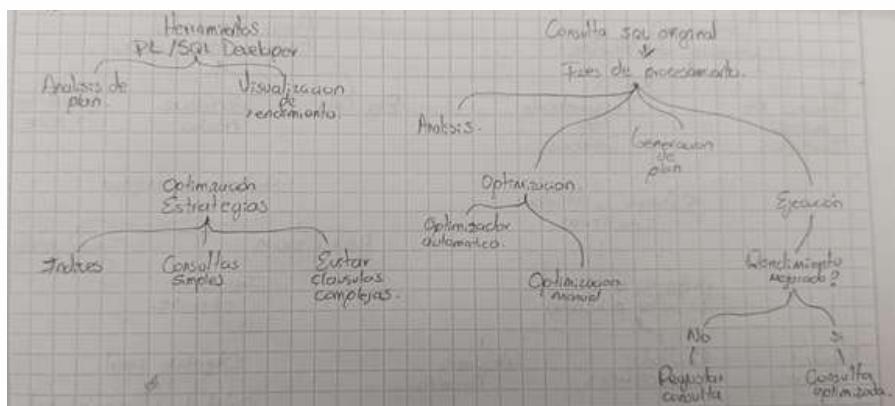
Rodrigo Alejandro Gonzalez; Sergio Agustín Giménez; Numa Roy Molina Pualuk; Rodrigo Zalazar; 2024-09-02

# Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube

El documento aborda la optimización de consultas SQL en bases de datos relacionales, destacando cómo las consultas mal codificadas pueden afectar el rendimiento. Se enfatiza el uso de herramientas como PL/SQL Developer para analizar y mejorar las consultas a través de planes de ejecución. Se describen las fases del procesamiento SQL (análisis, optimización, generación de registros y ejecución), y se señala que la optimización puede ser automática o manual, requiriendo conocimientos de SQL y la arquitectura de la base de datos.

## Reflexión

Optimizar consultas SQL es vital para el rendimiento de bases de datos en la nube. Implementar herramientas adecuadas y técnicas de mejora constante garantiza aplicaciones eficientes y escalables.



## Bibliografía

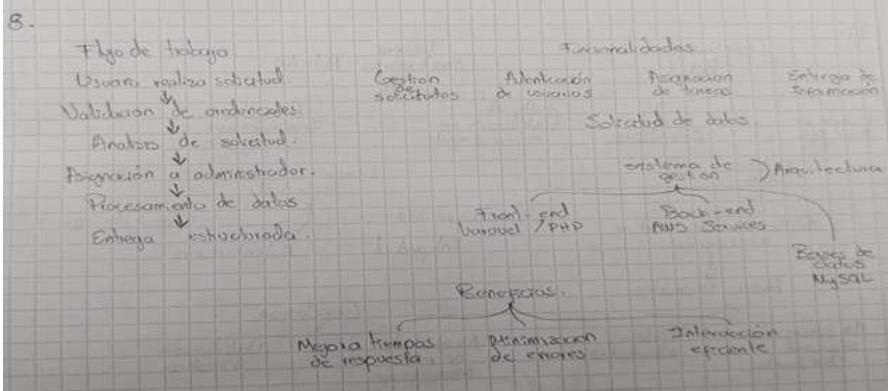
Manzanilla Vidal, Wilbert; Feliciano Morales, Severino; Alvarez Hilario, Valentin; Solis Carmona, Edgardo; Molina, Felix; 2018-11

# Solución para gestión y control de información para las bases de datos

El documento presenta una solución para la gestión de información en bases de datos, necesaria debido al aumento de datos impulsado por tecnologías como la inteligencia artificial, el internet de las cosas y la digitalización. La propuesta consiste en un sistema de gestión de solicitudes de datos, que captura, procesa y entrega información de manera estructurada y segura, evitando la saturación de las bases de datos. El sistema utiliza tecnologías como PHP con Laravel y MySQL, y permite la gestión de solicitudes, asignación de tareas, autenticación de usuarios y entrega eficiente de datos.

## Reflexión

: Sistemas robustos para gestionar datos en entornos modernos son esenciales. La implementación de metodologías adecuadas mejora la interacción entre usuarios y administradores, optimizando recursos y reduciendo errores.



## Bibliografía

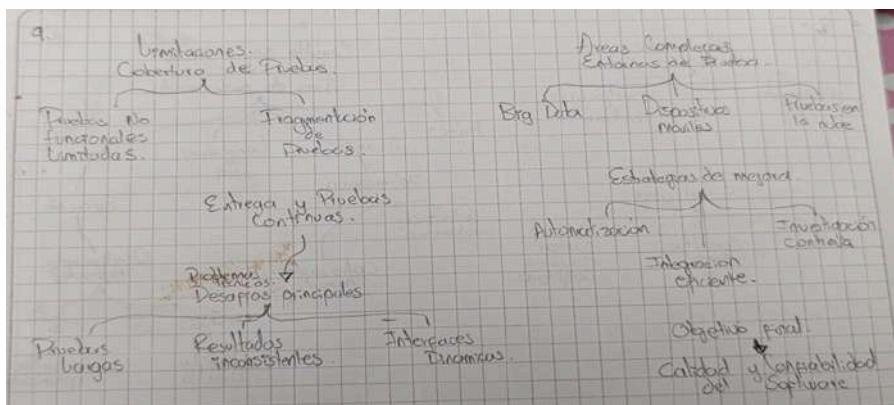
: Fuentes Pardo, Yudy Camila; 2024

# Problemas que afectan a la Calidad de Software en Entrega Continua y Pruebas Continuas

El documento examina los problemas que afectan la calidad del software en \*\*entrega continua (CD)\*\* y \*\*pruebas continuas (CT)\*\* en entornos ágiles, como pruebas largas, resultados inconsistentes, y desafíos con interfaces dinámicas, Big Data y dispositivos móviles. Destaca que las CT son esenciales para detectar defectos críticos rápidamente, aunque persisten retos como la cobertura limitada de pruebas no funcionales y en la nube. Se concluye que, pese a avances, se necesita investigación adicional para integrar eficientemente las pruebas continuas en los procesos de CD, asegurando calidad y confiabilidad.

## Reflexión

La integración de pruebas continuas en entrega continua asegura calidad, pero enfrenta retos significativos. Superarlos exige más investigación y técnicas avanzadas para mejorar la confiabilidad.



## Bibliografía

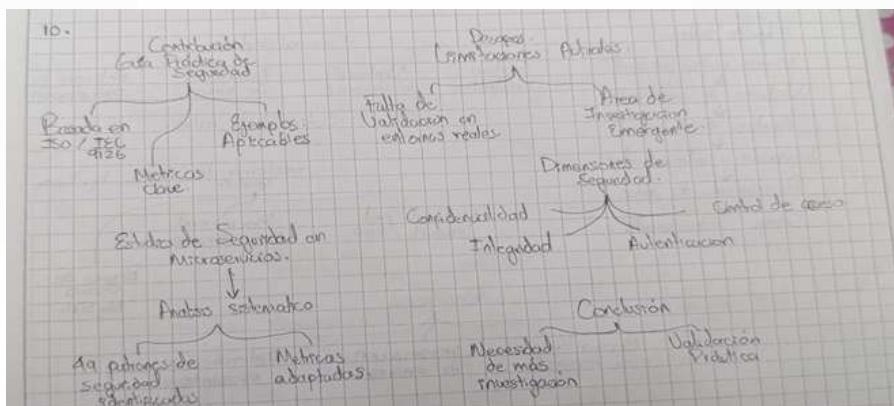
: Mascheroni, Maximiliano Agustín; Irrazábal, Emanuel; 2018

# Estudio de Métricas y Patrones de seguridad en Microservicios.

La tesis "Estudio de Métricas y Patrones de Seguridad en Microservicios" de Juana Victoria Penagos Sánchez aborda los retos de seguridad en esta arquitectura, destacando la falta de estudios sistemáticos. A través de un mapeo, analiza 49 patrones de seguridad y adapta métricas de otras arquitecturas, aunque muchas carecen de validación en entornos reales. Como aporte, propone una guía práctica basada en estándares de calidad (ISO/IEC 9126), con métricas clave y ejemplos aplicables. Concluye que la seguridad en microservicios es un área emergente que necesita más investigación y validación práctica.

## Reflexión

La seguridad en microservicios es un campo emergente que requiere mayor validación práctica. Las métricas y patrones deben adaptarse para garantizar sistemas robustos y confiables.



## Bibliografía

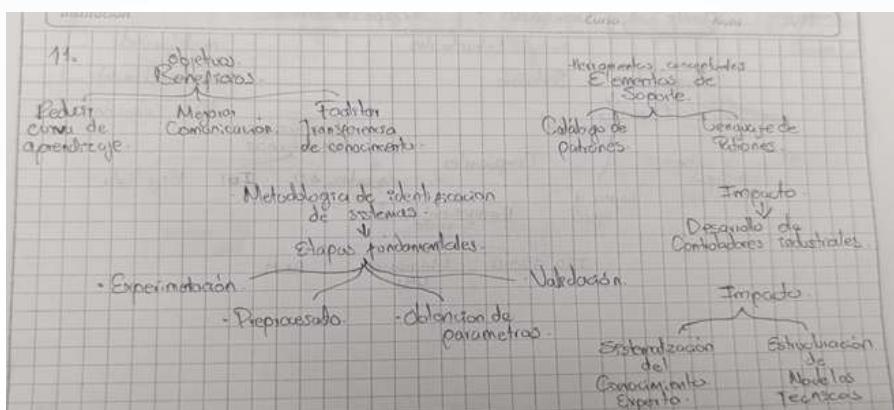
Juana Victoria Penagos Sánchez; Enero de 2023

# Metodología para la identificación de sistemas a través de patrones

La tesis "Metodología para la identificación de sistemas a través de patrones" de Francisco Sánchez Canto presenta una metodología innovadora para sistematizar y transmitir el conocimiento experto en la identificación de sistemas, especialmente en el desarrollo de controladores industriales. Basada en la teoría de patrones, organiza el proceso en cuatro etapas: experimentación, preprocesado, obtención de parámetros y validación. Incluye un catálogo y un lenguaje de patrones que guían el desarrollo de modelos, lo que contribuye a reducir la curva de aprendizaje, mejorar la comunicación entre expertos y principiantes, y facilitar la aplicación de patrones en áreas complejas como la ingeniería de control.

## Reflexión

Sistematizar la identificación de sistemas mediante patrones organiza procesos complejos, reduce la curva de aprendizaje y facilita la comunicación entre expertos y principiantes.



## Bibliografía

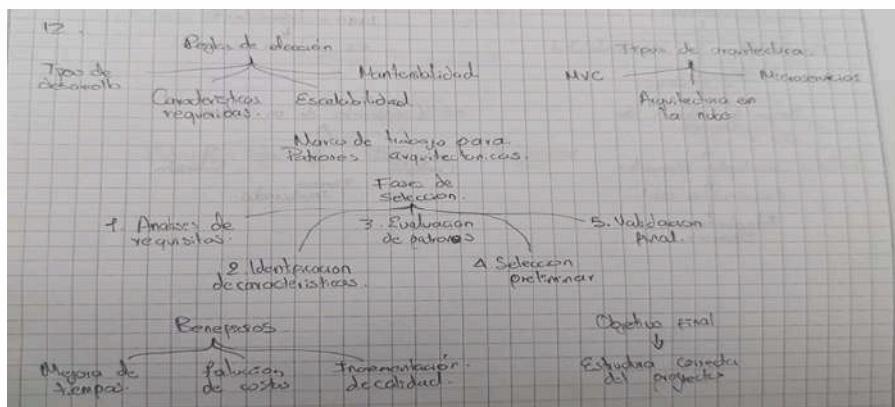
Francisco Sánchez Canto; Febrero 2009

# Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software

El artículo presenta un marco de trabajo para seleccionar patrones arquitectónicos en el desarrollo de software, abordando problemas de desacoplamiento y falta de conocimiento arquitectónico que afectan la calidad del producto. Basado en cinco fases, identifica patrones clave (como MVC, Microservicios y Arquitectura en la Nube) y establece reglas para elegirlos según el tipo de desarrollo y las características requeridas. Validado mediante un prototipo, el marco mejora tiempos, calidad, escalabilidad y mantenibilidad. Concluye que la solución es una herramienta práctica para guiar a arquitectos y desarrolladores en la selección de patrones adecuados, estableciendo estructuras desde el inicio del proyecto.

## Reflexión

Un marco práctico para elegir patrones arquitectónicos mejora la estructura del software desde el diseño inicial, promoviendo mantenibilidad, escalabilidad y menor costo a largo plazo.



## Bibliografía

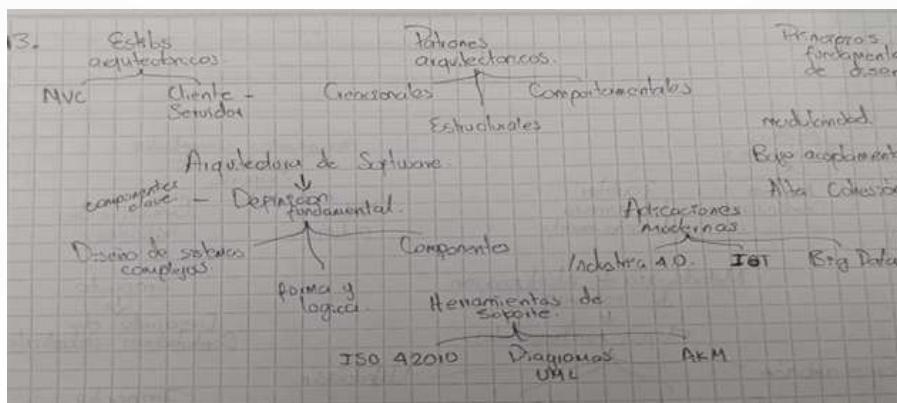
Mejía, Juan Camilo Giraldo; Agudelo, Fabio Alberto Vargas; Gil, Kelly Garzón; 25/07/2021

# Concepto de Arquitectura Software y Principios de Diseño

El documento es un material de estudio sobre Arquitectura de Software (AS), dirigido a estudiantes de Ingeniería en Software. Define la AS como el diseño de sistemas complejos, destacando principios como modularidad y bajo acoplamiento. Aborda el uso de patrones arquitectónicos (creacionales, estructurales y de comportamiento) y estilos como MVC y Cliente-Servidor. Introduce el estándar ISO 42010 para describir sistemas complejos e intensivos y presenta herramientas de gestión del conocimiento arquitectónico (AKM). También analiza aplicaciones en la Industria 4.0.

## Reflexión

La arquitectura de software combina principios, herramientas y decisiones iterativas. Su aplicación correcta permite desarrollar sistemas modernos y adaptables, esenciales en la Industria 4.0.



## Bibliografía

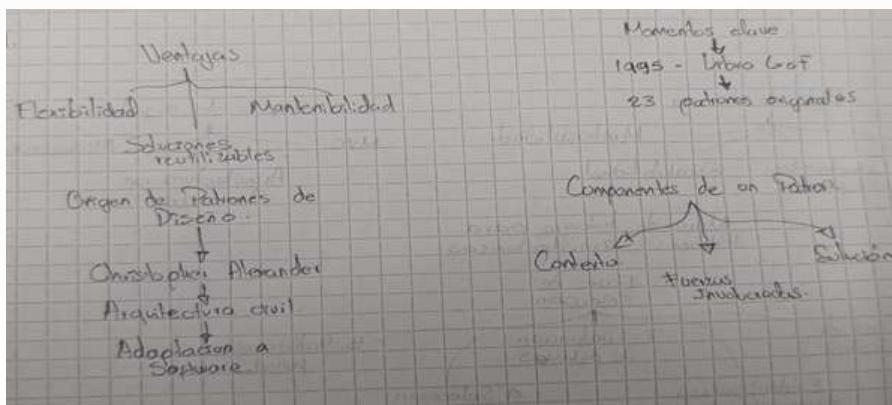
Rafael Capilla Sevilla; 01/01/2022

# Patrones. De Alexander a la Tecnología de Objetos

El texto explica los "patrones de diseño" en software, que son soluciones reutilizables a problemas comunes en el desarrollo de aplicaciones. Estos patrones surgieron a partir de las ideas del arquitecto Christopher Alexander, adaptadas al campo del software, especialmente en la programación orientada a objetos. El concepto se popularizó en 1995 con el libro Design Patterns de los "Cuatro de la Banda" (GoF), que presentó 23 patrones para resolver problemas recurrentes de diseño. Los patrones ayudan a crear sistemas más flexibles y fáciles de mantener, describiendo el contexto del problema, las fuerzas involucradas y la solución.

## Reflexión

Los patrones de diseño inspiran soluciones flexibles y reutilizables, destacándose como un pilar para construir sistemas eficaces y fáciles de mantener en entornos orientados a objetos.



## Bibliografía

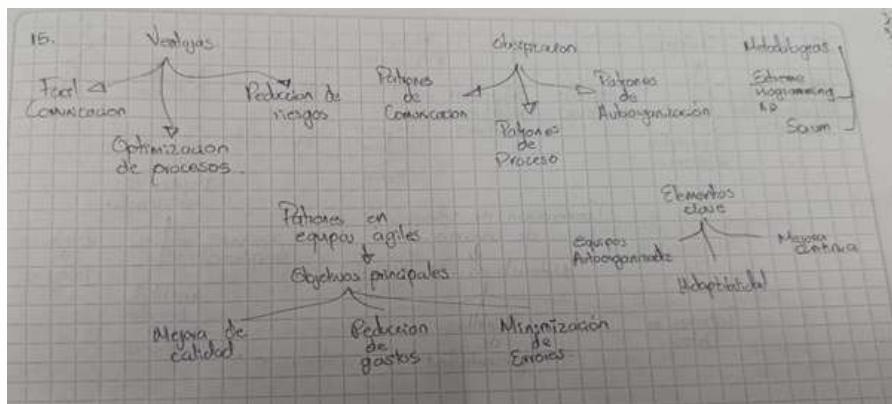
García-Peñalvo, Francisco J.; 1998-11

# Patrones y anti-patrones en equipos ágiles: una revisión sistemática

En la ingeniería del software, los patrones son soluciones reutilizables a problemas recurrentes, aceptadas por los profesionales para mejorar la calidad y reducir costes y errores. Su uso facilita la comunicación en los equipos y optimiza los procesos. La agilidad en el desarrollo de software busca reducir riesgos y garantizar la calidad y el cumplimiento de plazos y costes, promoviendo equipos autoorganizados. Los patrones y la agilidad comparten el objetivo de minimizar esfuerzos y asegurar calidad, y se integran en marcos ágiles como XP y Scrum. Este trabajo propone una investigación para desarrollar un marco de trabajo de patrones ágiles que apoyen la auto-organización de equipos a lo largo de su evolución.

## Reflexión

Los patrones ágiles optimizan la autoorganización de equipos, fomentando la calidad y eficiencia en el desarrollo de software, especialmente en marcos como Scrum y XP.



## Bibliografía

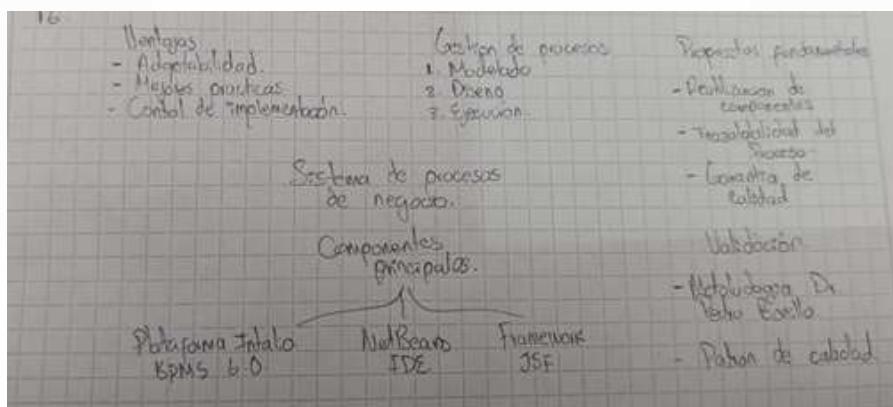
Ramos Vega, Cristina; 2019

# Sistema para el Control y Seguimiento de la Implementación de la Metodología de Gestión de Procesos de Negocio sustentada en el Uso de Patrones

Este Trabajo de Grado propone un sistema para gestionar el seguimiento de métodos, herramientas y técnicas en el desarrollo de software, facilitando la reutilización de componentes a través de un modelo de calidad. Basado en la metodología de Gestión de Proceso de Negocio con patrones del Dr. Pedro Bonillo, la solución utiliza Intalio BPMS 6.0 y NetBeans IDE con JSF para garantizar la calidad del producto final y asegurar la trazabilidad y éxito de cada fase del proceso.

## Reflexión

Implementar sistemas basados en patrones para la gestión de procesos empresariales asegura la trazabilidad y reutilización de componentes, mejorando la calidad final del producto.



## Bibliografía

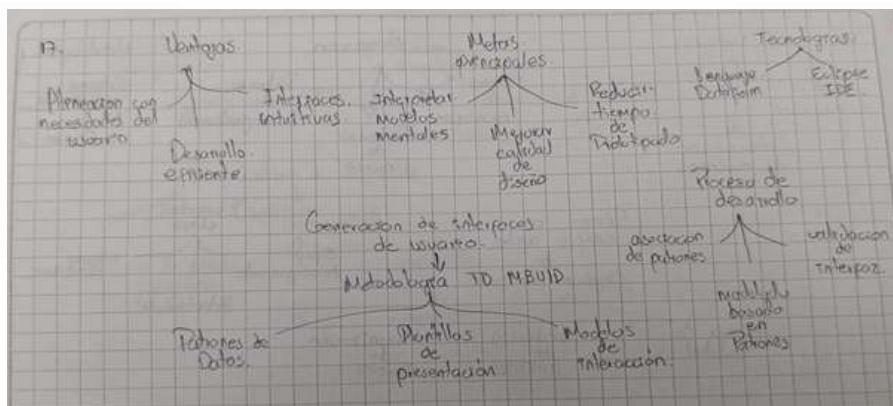
González, Deiby G.; Mendoza, Mayerling D.; 2-Dec-2014

# Generación de la interfaz de usuario de negocio a partir de patrones de negocios basada en los fundamentos metodológicos de td-mbuid

El documento propone un proceso basado en modelos para desarrollar interfaces de usuario de negocio, utilizando la metodología TD-MBUID. A través de la asociación de patrones de datos, plantillas de presentación y modelos de interacción, busca interpretar mejor los modelos mentales de los usuarios sobre los datos. Este enfoque mejora la calidad y el tiempo de diseño de prototipos de interfaces, utilizando el lenguaje DataForm y herramientas de Eclipse. Su objetivo es optimizar la creación de interfaces alineadas con las necesidades del usuario final, aplicando patrones de diseño y metodologías basadas en modelos para el desarrollo eficiente de software empresarial.

## Reflexión

Un enfoque basado en modelos para desarrollar interfaces de usuario mejora la alineación con las necesidades del usuario y optimiza el diseño eficiente en proyectos empresariales.



## Bibliografía

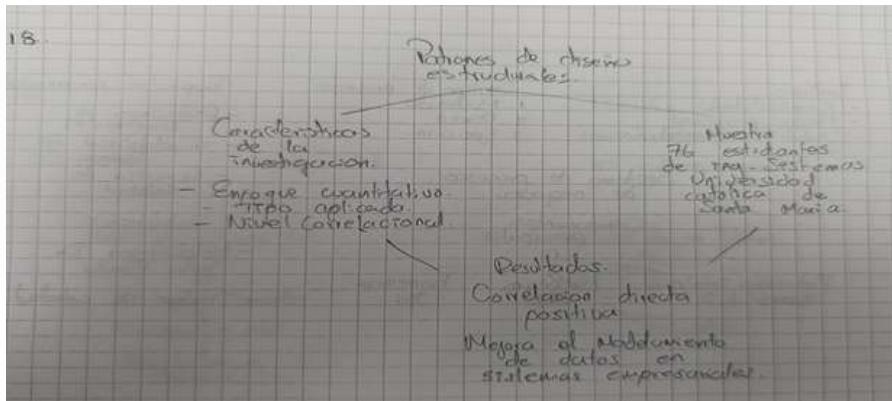
Triviño Arbeláez, Jorge Iván; 2016

# Aplicación de patrones de diseño estructurales para el modelamiento de clases de los sistemas empresariales

La tesis de Manuel Mariano Zúñiga Carnero, "Aplicación de patrones de diseño estructurales para el modelamiento de clases de los sistemas empresariales" (2020), estudia cómo los patrones de diseño estructurales mejoran el modelado de datos en sistemas de información administrativos. A través de una investigación cuantitativa con 76 estudiantes, los resultados muestran una relación positiva entre la aplicación de estos patrones y la mejora en el diseño de sistemas empresariales, destacando su efectividad en la resolución de problemas de diseño.

## Reflexión

El uso de patrones estructurales facilita el modelamiento de sistemas complejos, asegurando calidad y permitiendo abordar desafíos específicos en sistemas empresariales.



## Bibliografía

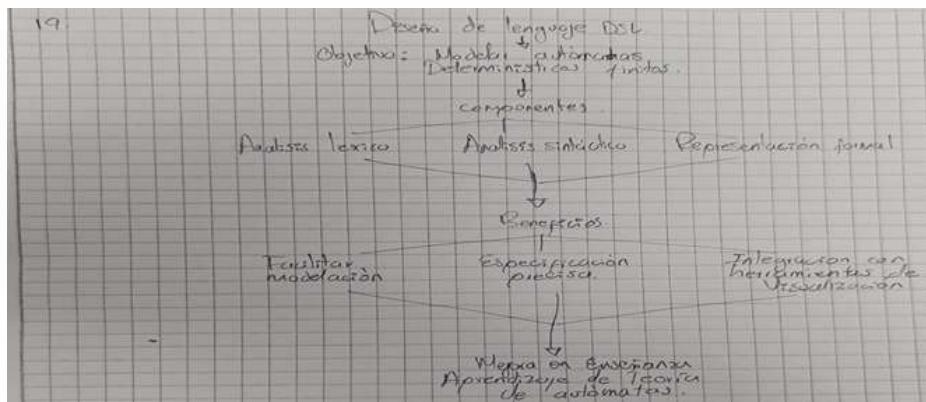
: Zúñiga Carnero, Manuel Mariano; 2020

# Diseño e implementación de un lenguaje para modelar autónomas determinísticos finitos

El trabajo de grado "Diseño e implementación de un lenguaje para modelar autómatas determinísticos finitos" propone un lenguaje de dominio específico (DSL) que facilita a los estudiantes modelar y especificar autómatas finitos determinísticos. Este DSL ofrece una representación formal y precisa, integrándose con herramientas de visualización para mejorar la comprensión de los conceptos. Implementado con técnicas de análisis léxico y sintáctico, el lenguaje se muestra efectivo en entornos educativos, contribuyendo al aprendizaje de la teoría de autómatas y mejorando la representación y análisis de estos modelos.

## Reflexión

Diseñar lenguajes específicos para modelar autómatas fortalece la enseñanza de teoría computacional, brindando herramientas prácticas y efectivas para su representación.



## Bibliografía

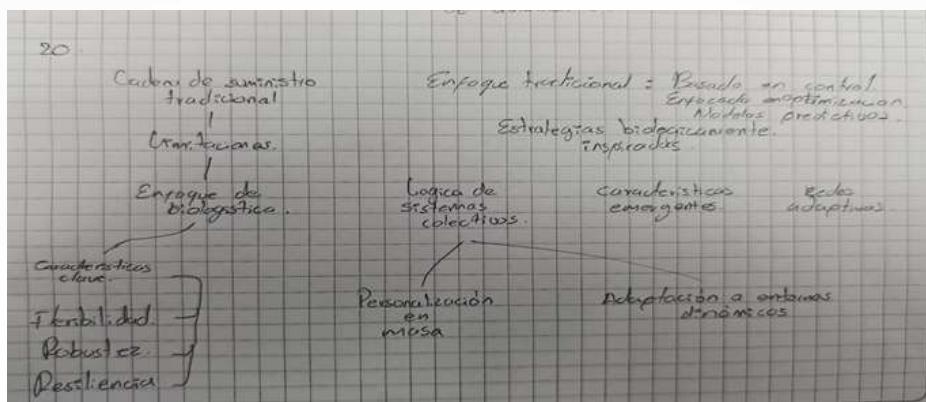
Zúñiga Carnero, Manuel Mariano; Julio 2007

# Biológica: Estrategias computacionales bioinspiradas para la gestión de las redes de suministro complejas

El artículo "Biológica: Estrategias computacionales bioinspiradas para la gestión de redes de suministro complejas" propone un modelo basado en sistemas biológicos para mejorar la gestión de redes de suministro. Los autores, Nelson Alfonso Gómez-Cruz y John Leonardo Vargas Mesa, argumentan que los modelos tradicionales, enfocados en control y predicción, no son suficientes para enfrentar desafíos como la personalización en masa y la adaptabilidad en entornos cambiantes. Al tratar las redes de suministro como sistemas colectivos, se busca aprovechar características como flexibilidad, robustez y resiliencia, proponiendo que enfoques bioinspirados mejorarán su eficiencia y adaptabilidad.

## Reflexión

Aplicar estrategias bioinspiradas a las redes de suministro potencia su flexibilidad, resiliencia y adaptabilidad, abordando desafíos complejos con enfoques innovadores.



## Bibliografía

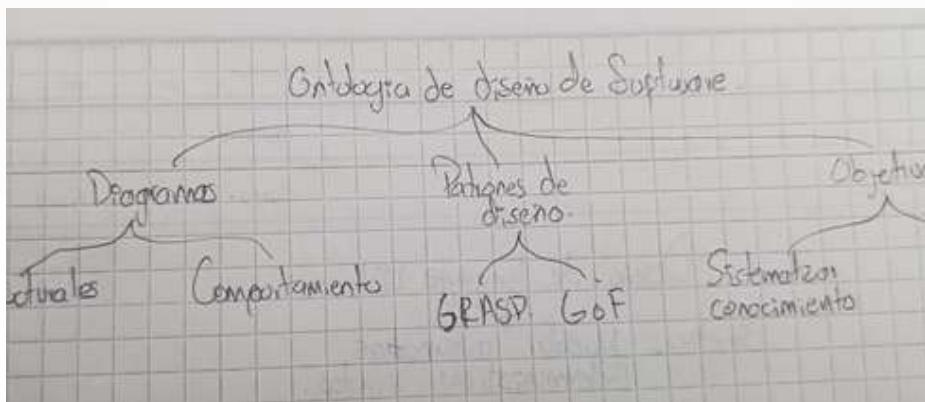
Giraldo G., Gloria L.; Acevedo O., Juan F.; Moreno N., David A.; 10 de noviembre de 2011

# Desarrollo de framework web para el desarrollo dinámico de aplicaciones.

El documento propone una ontología para organizar el conocimiento en el diseño de software, utilizando diagramas y patrones esenciales para esta fase del desarrollo. Define la ontología como una representación formal que mejora la comunicación y reutilización del conocimiento. Incluye diagramas estructurales y de comportamiento, junto con patrones de diseño como GRASP y GoF. La ontología busca sistematizar conceptos dispersos, facilitando su enseñanza y aplicación profesional, mejorando la calidad del software y promoviendo buenas prácticas. También se enfoca en el diseño de sistemas colaborativos para enseñar estos conceptos a estudiantes de ingeniería.

## Reflexión

La implementación del Framework Web optimiza procesos clave como autenticación y gestión de usuarios, mejorando la eficiencia y personalización. Un modelo pequeño validó los requerimientos antes de escalar. Establecer estándares de codificación y una arquitectura Cliente/Servidor asegura calidad y sostenibilidad. A pesar de los retos técnicos, como la compatibilidad de librerías, se enfatiza la necesidad de pruebas exhaustivas y mejoras continuas para garantizar un producto robusto.



## Bibliografía

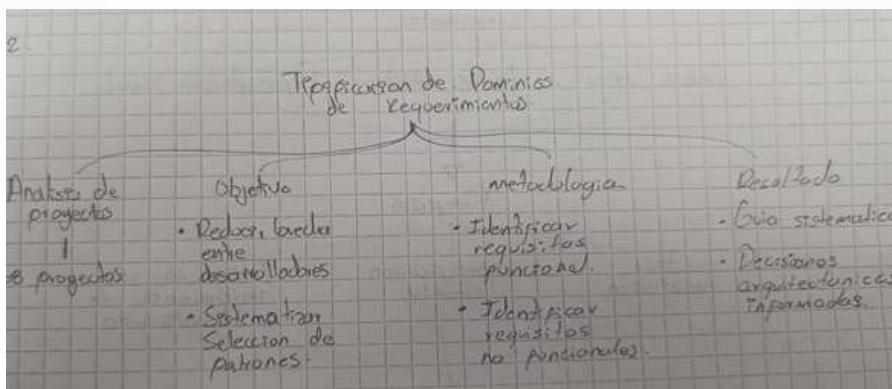
Martínez Villalobos, Gustavo; Camacho Sánchez, Germán Darío; Biancha Gutiérrez, Daniel Alberto; abril 2010.

# Tipificación de Dominios de Requerimientos para la Aplicación de Patrones Arquitectónicos

El artículo propone una metodología para seleccionar patrones arquitectónicos en el desarrollo de software, simplificando el proceso mediante un "dominio de requerimientos" basado en el análisis de 68 proyectos. Esta guía sistemática busca reducir la brecha entre ingenieros en formación y profesionales experimentados, facilitando la elección de patrones arquitectónicos adecuados según los requisitos del sistema. La validación de la metodología se realizó en un proyecto de evaluación docente, demostrando su efectividad en la toma de decisiones arquitectónicas más informadas.

## Reflexión

El uso de patrones arquitectónicos en software es crucial para asegurar calidad y eficiencia, aunque su implementación no siempre está documentada. Son esenciales para organizar el diseño y cumplir con los requisitos funcionales y no funcionales, pero requieren sensibilización en los equipos de desarrollo y estándares claros para fomentar buenas prácticas. La integración de lenguajes de modelado como UML facilita la selección adecuada de patrones, mejorando la navegabilidad del código, optimizando procesos y asegurando soluciones sostenibles y adaptables.



## Bibliografía

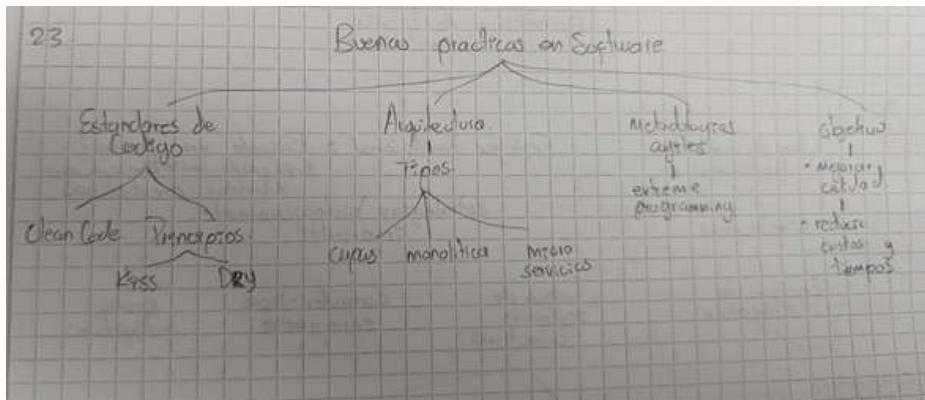
Johanna M. Suárez; Luz E. Gutiérrez; 2016

# Buenas prácticas en la construcción del software.

El artículo reflexiona sobre las buenas prácticas en la construcción de software en la cuarta revolución industrial, destacando desafíos en el desarrollo eficiente de sistemas sin exceder costos. Se subrayan estándares como Clean Code, y principios como KISS y DRY, junto con la importancia de una arquitectura adecuada que cubra requisitos funcionales y no funcionales. Los patrones de diseño se destacan como herramientas para estructurar el código, mientras que metodologías como XP y herramientas tecnológicas optimizan procesos, mantienen flexibilidad y aseguran entregables funcionales. La integración de estas prácticas mejora la calidad, mantenibilidad y reduce costos en las organizaciones.

## Reflexión

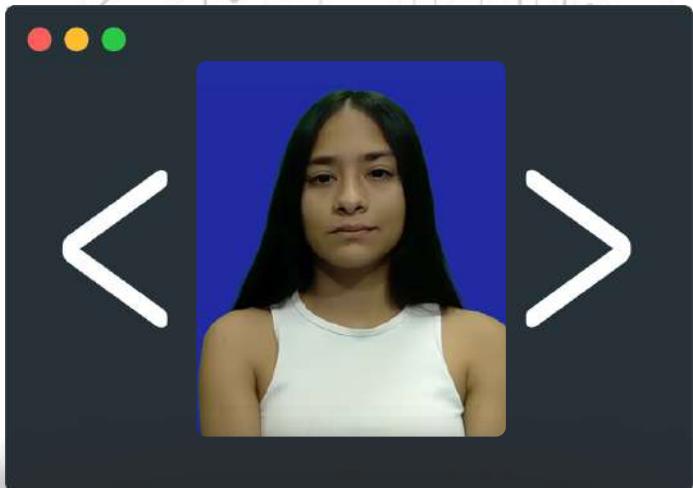
Promover una cultura organizacional basada en buenas prácticas de desarrollo es clave para crear software eficiente y sostenible. La adopción de estándares como Clean Code mejora la comprensión y mantenibilidad del código, mientras que herramientas tecnológicas y marcos de trabajo optimizan el rendimiento del equipo. Una arquitectura adecuada asegura soluciones alineadas con las necesidades del negocio, fortaleciendo la calidad y el impacto del software en las organizaciones.



## Bibliografía

Camilo Alberto Prieto; Diego Alejandro Madrid; 2024-01-23

# Valentina Silva Garrido



Valentina Silva Garrido es una joven profesional en constante crecimiento, egresada del Servicio Nacional de Aprendizaje (SENA) como Técnica en Diseño e Integración en Multimedia. Actualmente cursa el Tecnólogo en Análisis y Desarrollo de Software en la misma institución, complementando su formación con una pasantía en el área.

Conocedora de herramientas de diseño gráfico y desarrollo web, Valentina demuestra un alto nivel de compromiso y habilidades para la resolución de problemas.

Su experiencia en servicio al cliente y su capacidad para trabajar en equipo la posicionan como una candidata valiosa para roles que requieran creatividad, pensamiento lógico y adaptabilidad.

# Análisis comparativo de Patrones de Diseño de Software

Los patrones de diseño son herramientas esenciales para construir software robusto y escalable. Cada patrón ofrece una estructura y solución específicas, pero la elección correcta depende del contexto y los requisitos del proyecto. En este análisis, comparamos patrones populares como Template Method, MVC, MVP, Front Controller y MVVM, considerando aspectos como lenguaje de programación, complejidad, seguridad y uso. Al comprender las fortalezas y debilidades de cada patrón, los desarrolladores pueden tomar decisiones informadas y crear aplicaciones más eficientes y sostenibles.

## Reflexión

Los patrones de diseño representan un avance significativo en la ingeniería de software, pues proporcionan un marco estructurado para resolver problemas recurrentes en el desarrollo. Al mismo tiempo, evidencian la importancia de la flexibilidad y la adaptabilidad en la elección de la solución más adecuada para cada situación. No existe un patrón perfecto y cada uno de ellos tiene su contexto y uso específico, y elegir el adecuado depende de factores como los requisitos del proyecto, el equipo de desarrollo, la escalabilidad, el mantenimiento y las características principales del software que se está realizando.



## Bibliografía

Gavilánez Alvarez, O. D., Layedra, N., & Ramos, V. (2022). Análisis comparativo de patrones de diseño de software, 20 páginas.  
<https://dialnet.unirioja.es/servlet/articulo?codigo=9042927>

# Framework de Evaluación para Modelos Formales de Patrones de Diseño

¿Cómo podemos asegurar que los patrones de diseño se utilicen de forma correcta y consistente en nuestros proyectos? Las descripciones informales de los patrones limitan su potencial. Este estudio presenta un marco para evaluar la eficacia de los lenguajes formales en la definición precisa de patrones. Al comprender mejor estas herramientas, podemos desarrollar procesos de diseño más eficientes y confiables, aprovechando el poder de la automatización. Ante la diversidad de modelos formales disponibles, se diseñó un marco de evaluación de dos niveles. El primer nivel evalúa 14 características de los lenguajes formales utilizados para modelar patrones orientados a objetos. El segundo nivel se centra en analizar 10 aspectos propios de los modelos resultantes.

## Bibliografía

- Flores, A. P., & Fillottrani, P. R. (2003). Framework de Evaluación para Modelos Formales de Patrones de Diseño. Documento en línea. Recuperado de [https://sedici.unlp.edu.ar/bitstream/handle/10915/22886/Documento\\_completo.pdf?sequence=1&isAllowed=y](https://sedici.unlp.edu.ar/bitstream/handle/10915/22886/Documento_completo.pdf?sequence=1&isAllowed=y).



## Reflexión

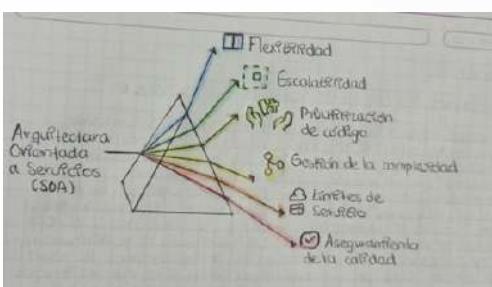
La búsqueda de descripciones precisas para los patrones de diseño nos lleva a explorar el potencial de los lenguajes formales. Estos lenguajes, al ofrecer una representación rigurosa y no ambigua, facilitan la comunicación entre desarrolladores y promueven la reutilización de soluciones probadas. Un diseño basado en patrones más sólidos y precisos no solo mejora la calidad del software, sino que también abre nuevas posibilidades para la automatización del desarrollo, acelerando los procesos y reduciendo los márgenes de error.

# Arquitectura de software, esquemas y servicios

El artículo explora la evolución de la arquitectura de software, desde modelos monolíticos hacia la arquitectura orientada a servicios (SOA). Se destaca cómo los esquemas, conjuntos de clases que encapsulan funcionalidades comunes, y la SOA, que descompone las aplicaciones en servicios independientes, ofrecen soluciones a los problemas de las arquitecturas tradicionales, como la duplicación de código, el acoplamiento y la dificultad de mantenimiento. La SOA promueve la reutilización, la flexibilidad y la escalabilidad de las aplicaciones, al permitir que los servicios se comuniquen entre sí de manera independiente. Sin embargo, la implementación de SOA presenta desafíos como la complejidad en la gestión de servicios y la necesidad de un diseño cuidadoso.

## Reflexión

La evolución hacia la arquitectura orientada a servicios representa un cambio de paradigma en el desarrollo de software. Al descomponer las aplicaciones en servicios independientes, se logra una mayor flexibilidad, escalabilidad y reutilización del código. Sin embargo, esta transición implica desafíos como la gestión de la complejidad, la definición de límites claros entre servicios y la garantía de la calidad de servicio. La SOA ofrece un camino prometedor para construir sistemas de software más ágiles y adaptables, pero su implementación requiere una planificación cuidadosa y una comprensión profunda de los principios arquitectónicos.



## Bibliografía

Romero, P.A., (2006). Arquitectura de software, esquemas y servicios. 3 Páginas. Recuperado de <https://dialnet.unirioja.es/servlet/articulo?codigo=4786655>

# Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web

En el artículo analiza la problemática actual en el desarrollo de software de la Asamblea Nacional del Ecuador, la cual se basa en una arquitectura monolítica que dificulta el mantenimiento, la escalabilidad y la entrega de nuevas funcionalidades. Esta arquitectura monolítica implica que todas las funcionalidades de la aplicación están agrupadas en un solo bloque, lo que genera dificultades cuando se necesita hacer cambios o actualizaciones. Como solución, se propone adoptar una arquitectura de microservicios, donde la aplicación se divide en pequeños servicios independientes que se comunican entre sí. Esta nueva arquitectura ofrecería mayor flexibilidad, escalabilidad y facilidad de mantenimiento.

Transición a Microservicios		
Pros	vs	Cons
 Modernización		 Cambio cultural
 Eficiencia mejorada		 Requerido
 Agilidad en la respuesta		 Nuevas metodologías requeridas
 Mejor calidad de servicio		 Adopción de herramientas
 Cultura de innovación		 Mejora continua

## Bibliografía

- López, L., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web., 12 Páginas. Recuperado De <http://138.59.13.30/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf>

## Reflexión

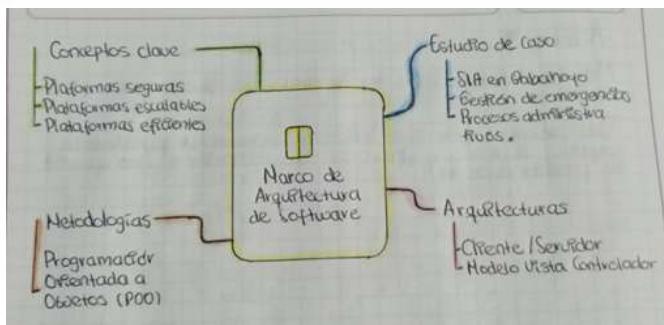
Resalta la importancia de la evolución tecnológica en el ámbito gubernamental, específicamente en la Asamblea Nacional del Ecuador. La arquitectura monolítica, aunque tradicional, presenta limitaciones en términos de escalabilidad, mantenimiento y adaptación a las nuevas demandas. La propuesta de migrar a una arquitectura de microservicios es un paso audaz hacia la modernización. Esta transición no solo implica un cambio técnico, sino también una transformación cultural en la forma de desarrollar y gestionar software.

# Marco de referencia de arquitectura de software para aplicaciones web y móviles

Este artículo destaca la importancia de desarrollar plataformas seguras y escalables para aplicaciones web y móviles, utilizando arquitectura Cliente/Servidor y metodologías como POO y MVC. Se presenta un marco de referencia que abarca los requisitos técnicos y funcionales, integrando tecnologías libres para reducir costos. Además, se describe el diseño de un Sistema Integral de Aplicaciones (SIA) implementado en dos instituciones públicas en Babahoyo, mejorando la gestión de emergencias y procesos administrativos, mediante herramientas como Sencha Touch y ExtJS para aplicaciones modulares y escalables.

## Reflexión

El artículo revela la importancia de construir sistemas que no solo sean funcionales, sino que también sean escalables, modulares y fáciles de mantener. Esto es crucial en un mundo en constante cambio, donde las necesidades tecnológicas evolucionan rápidamente. El uso de arquitecturas flexibles y herramientas que favorecen la reutilización de código permite a las organizaciones adaptarse mejor a nuevos desafíos, reduciendo el tiempo y los costos asociados con el desarrollo y la actualización de sistemas.



## Bibliografía

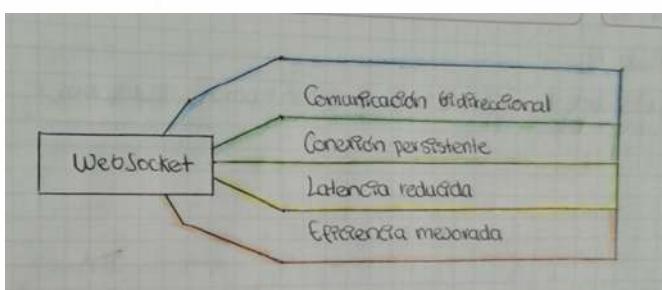
- Maliza Martinez, C. A., Lopez Mendizabal, V. L. & Mackiff Peñafiel, V. V, (2016). Marco de referencia de arquitectura de software para aplicaciones web y móviles., 4 Páginas. Recuperado de <https://revistas.utb.edu.ec/index.php/sr/article/view/123/pdf>

# Arquitectura de Software con websocket para aplicaciones web multiplataforma

El artículo analiza la integración de WebSocket en Java EE para habilitar comunicación bidireccional en tiempo real en aplicaciones web. WebSocket supera las limitaciones de tecnologías como AJAX polling y Comet, ofreciendo una conexión persistente y eficiente entre cliente y servidor. Es ideal para aplicaciones que requieren actualizaciones instantáneas, como chats, juegos en línea y sistemas de votación. La especificación JSR-356 facilita la implementación en Java EE, permitiendo a los desarrolladores crear aplicaciones en tiempo real de manera más ágil. El artículo incluye un ejemplo de un sistema de votación para ilustrar sus beneficios.

## Reflexión

El artículo destaca a WebSocket como una tecnología clave para aplicaciones web en tiempo real. Superando las limitaciones de enfoques tradicionales, WebSocket permite una interacción más dinámica y bidireccional entre cliente y servidor. Esta capacidad ha transformado la experiencia de usuario en aplicaciones web. Sin embargo, su implementación exitosa requiere un profundo entendimiento de los protocolos de red y la arquitectura de las aplicaciones. A medida que las aplicaciones web se vuelven más complejas, tecnologías como WebSocket se consolidarán como esenciales en el desarrollo de software.



## Bibliografía

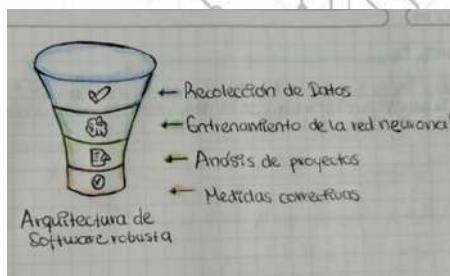
Luis Vivas, Horacio Muñoz Abbate, Mauro Cambarieri, Nicolás García Martínez, Marcelo Petroff, (2014). Arquitectura de Software con websocket para aplicaciones web multiplataforma., 10 Páginas. Recuperado de <https://rid.unrn.edu.ar/bitstream/20.500.12049/148/1/CACIC%202014%20-%20Ar>

# **Modelo Teórico para la Identificación del Antipatrón "Stovepipe System" en la Etapa de la Implementación de una Arquitectura de Software**

El artículo propone una solución innovadora para detectar y prevenir el antipatrón "Stovepipe System", caracterizado por la falta de integración entre componentes, lo que dificulta el mantenimiento y actualización del sistema. La propuesta consiste en desarrollar un modelo teórico que utiliza redes neuronales para identificar este antipatrón durante la fase de implementación. Al entrenar la red con datos de proyectos previos, se busca detectar patrones asociados con el problema y prevenirlo en futuros proyectos, asegurando una arquitectura de software más sólida y escalable.

## **Bibliografía**

Candia Peñaloza, J. C., (2014). Modelo Teórico para la Identificación del Antipatrón "Stovepipe System" en la Etapa de la Implementación de una Arquitectura de Software., 7 Páginas. Recuperado de [http://www.revistasbolivianas.ciencia.bo/scielo.php?pid=S3333-7777201400100023&script=sci\\_arttext&tlang=es](http://www.revistasbolivianas.ciencia.bo/scielo.php?pid=S3333-7777201400100023&script=sci_arttext&tlang=es)

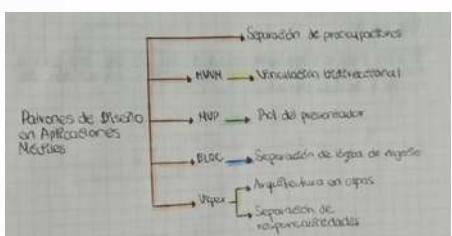


## **Reflexión**

El artículo presenta una propuesta innovadora para abordar un problema recurrente en el desarrollo de software: la identificación temprana de antipatrones. Al usar redes neuronales para detectar el antipatrón Stovepipe System, se abre un camino hacia soluciones más eficientes y precisas. Esta aproximación puede mejorar la calidad de las arquitecturas de software y reducir los costos de corrección de errores y refactorización. Sin embargo, su éxito dependerá de la calidad y cantidad de datos para entrenar la red neuronal y de la capacidad para capturar las características clave del antipatrón.

# Análisis comparativo de patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad: Una revisión sistemática de la literatura

Este artículo explora los patrones de diseño en el desarrollo de aplicaciones móviles. Se inician explicando conceptos básicos de la programación orientada a objetos y luego se profundiza en cómo los patrones de diseño mejoran la calidad y eficiencia del software. Se analizan en detalle patrones populares como MVC (Model-View-Controller), MVVM (Model-View-ViewModel), MVP (Model-View-Presenter), BLOC (Business Logic Component) y Viper, luego se presenta una revisión exhaustiva de la literatura para identificar los más efectivos. En conclusión, el estudio ofrece una guía práctica para desarrolladores que buscan seleccionar y aplicar los patrones de diseño adecuados en sus proyectos móviles.



## Bibliografía

Jesús Alberto Abanto Cruz, Omar Fernando Gonzales Ramírez,(2019). Análisis comparativo de patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad: Una revisión sistemática de la literatura., 15  
Páginas Recuperado de <https://repositorio.upeu.edu.pe/server/api/core/bitstreams/2107746c-809f-4619-8152-4eb578af9b27/content>

## Reflexión

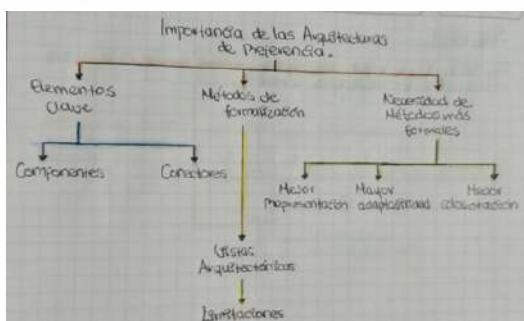
El artículo profundiza en la importancia de los patrones de diseño en el desarrollo de aplicaciones móviles. Explica cómo estos patrones ayudan a organizar el código de manera eficiente, facilitando la comprensión, modificación y mantenimiento de las aplicaciones. Además, destaca que la selección adecuada de patrones puede mejorar significativamente la seguridad de las aplicaciones, especialmente en el manejo de datos sensibles. La revisión exhaustiva de estudios previos le otorga al artículo una solidez científica y permite identificar los patrones más utilizados y efectivos en el desarrollo móvil. El artículo presenta una guía práctica para desarrolladores móviles que buscan construir aplicaciones de alta calidad y seguras.

# Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software

El artículo destaca la importancia de las arquitecturas de referencia en el desarrollo de software, ya que sirven como guías para crear nuevos sistemas, promoviendo la reutilización de componentes y garantizando la coherencia en los proyectos. Se enfoca en identificar los elementos clave de estas arquitecturas, como los componentes y conectores, que son esenciales. Además, analiza los métodos utilizados para formalizarlas, observando que las vistas arquitectónicas son las más comunes, aunque limitan el uso de herramientas automatizadas. El artículo subraya la necesidad de desarrollar métodos más formales para superar estas limitaciones.

## Reflexión

El artículo presenta una visión clara sobre la importancia de las arquitecturas de referencia en el desarrollo de software, pero podría profundizar en ciertos aspectos. Sería útil explorar cómo estas arquitecturas se adaptan a los cambios constantes en los requisitos y tecnologías. También sería interesante discutir cómo pueden mejorar la colaboración entre equipos de desarrollo y facilitar la transición hacia nuevas tecnologías. Dada la creciente complejidad de los sistemas, las arquitecturas de referencia se vuelven más esenciales, ya que no solo mejoran la calidad del software, sino que también reducen los costos de desarrollo y mantenimiento.



## Bibliografía

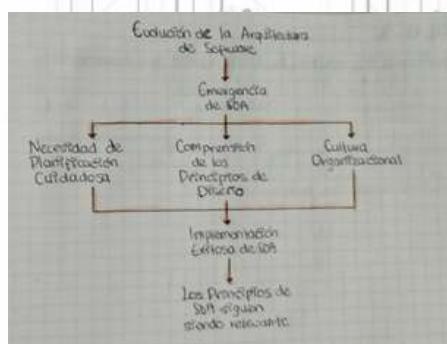
Sanchez Palmero, M. A., Silega Martinez, N., & Rojas Grass, O. Y. (2018). Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software., 15 Páginas. Recuperado de [http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci\\_arttext](http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci_arttext)

# Arquitectura de software. Arquitectura orientada a servicios

El artículo traza un recorrido histórico por la evolución de la arquitectura de software, desde los primeros intentos de estructurar sistemas de manera ordenada hasta la consolidación de la Arquitectura Orientada a Servicios (SOA). Se destaca la importancia de figuras como Dijkstra, Wirth y Parnas en el desarrollo de conceptos fundamentales como la modularidad y el ocultamiento de información. La SOA emerge como una respuesta a la necesidad de sistemas más flexibles y adaptables, ofreciendo una serie de beneficios como la reutilización de componentes, la agilidad en los cambios y la mejora de la integración entre sistemas heterogéneos. Los servicios web y la gestión de procesos de negocio (BPM) se presentan como tecnologías clave para implementar la SOA.

## Bibliografía

Espinal Martín, Y., (2012). Arquitectura de software. Arquitectura orientada a servicios., 10 Páginas. Recuperado de [https://dialnet.unirioja.es/servlet/\\_articulo?codigo=8590088](https://dialnet.unirioja.es/servlet/_articulo?codigo=8590088)



## Reflexión

El artículo evidencia cómo la arquitectura de software ha evolucionado de una disciplina incipiente a una rama fundamental de la ingeniería de software. La SOA, como última evolución presentada, demuestra la búsqueda constante de la industria por crear sistemas más eficientes, escalables y alineados con las necesidades de negocio. Sin embargo, es importante destacar que la implementación exitosa de una arquitectura SOA requiere una planificación cuidadosa, una comprensión profunda de los principios de diseño y una cultura organizacional que fomente la colaboración y la innovación. A medida que las tecnologías continúan avanzando, es probable que surjan nuevas arquitecturas y enfoques, pero los principios fundamentales de la SOA seguirán siendo relevantes para el desarrollo de sistemas de software de alta calidad.

# Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software

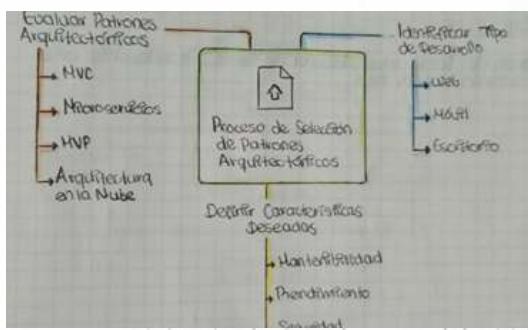
El artículo presenta un marco de trabajo diseñado para ayudar a desarrolladores y arquitectos de software a seleccionar el patrón arquitectónico más adecuado para sus proyectos. Aborda el problema de la falta de conocimiento sobre arquitectura de software en muchos proyectos, lo que puede resultar en productos de baja calidad y dificultades de mantenimiento. La solución propuesta es un marco que guía al usuario en el proceso de selección, basado en las características del proyecto y las necesidades del cliente. Este marco considera factores como el tipo de desarrollo (web, móvil, escritorio), las características deseadas (mantenibilidad, rendimiento, seguridad) y los patrones arquitectónicos más comunes (MVC, microservicios, MVP, arquitectura en la nube).

## Bibliografía

Giraldo Mejia, J. C., Vargas Agudelo, F. A., & Garzon Gil, K., (2021). Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software,. 14 Páginas. <https://n9.cl/umwh8c>

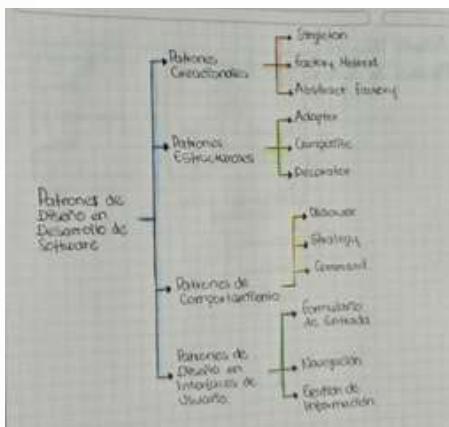
## Reflexión

El artículo presenta una propuesta valiosa para el desarrollo de software al introducir un marco de trabajo para seleccionar patrones arquitectónicos. Esta iniciativa aborda un problema común: la elección adecuada de la arquitectura al inicio de un proyecto, lo cual impacta directamente en la calidad, mantenibilidad y escalabilidad del producto final. La propuesta es útil porque ofrece una guía sistemática para tomar decisiones informadas sobre la arquitectura. Al considerar factores como el tipo de desarrollo, las características deseadas y los patrones arquitectónicos más comunes, el marco ayuda a los desarrolladores a evitar errores y elegir la arquitectura más adecuada.



## Patrones de diseño

El artículo explora los patrones de diseño, tanto en el desarrollo de software como en el diseño de interfaces de usuario. Los patrones de diseño son soluciones pre establecidas y probadas para problemas comunes en la programación y el diseño. En el desarrollo de software, estos patrones ayudan a organizar el código, mejorar su legibilidad y facilitar el mantenimiento. Se clasifican en creacionales, estructurales y de comportamiento, cada uno con sus propias características y aplicaciones. Por otro lado, en el diseño de interfaces de usuario, los patrones sirven para crear interfaces intuitivas y consistentes, mejorando la experiencia del usuario. Estos patrones se utilizan para resolver problemas comunes como la entrada de datos, la navegación y la gestión de información. El artículo destaca las ventajas y desventajas de utilizar patrones de diseño y enfatiza la importancia de elegir los patrones adecuados para cada contexto.



## Bibliografía

- Murillo Alpizar, J. C., Oconitrillo Rodriguez, C., & Equivel Bolaños, L. (2017). Patrones de diseño, 17 Páginas. Recuperado de <https://n9.cl/zmv3i>

## Reflexión

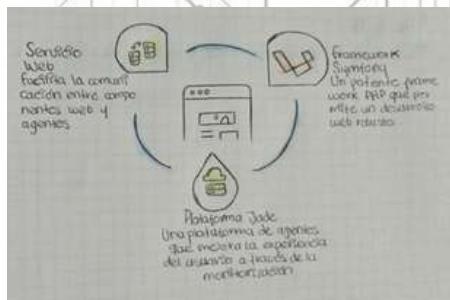
Los patrones de diseño son herramientas poderosas que pueden transformar la forma en que desarrollamos software y diseñamos interfaces. Al proporcionar soluciones probadas y reutilizables, los patrones de diseño nos permiten construir aplicaciones más robustas, escalables y mantenibles. Sin embargo, es esencial comprender que los patrones de diseño no son una panacea. Su uso inadecuado puede llevar a soluciones sobre ingeniería difíciles de entender. Es fundamental elegir los patrones adecuados para cada problema y considerar las trade-offs involucrados. Además, es importante mantenerse actualizado sobre los nuevos patrones y las mejores prácticas en la industria. En última instancia, los patrones de diseño son una herramienta que debe ser utilizada con juicio y experiencia para lograr resultados óptimos.

# Arquitectura de Comunicación entre Frameworks JadeSymfony

El artículo describe una solución arquitectónica para integrar un sistema web desarrollado en Symfony con una plataforma de agentes basada en Jade. El objetivo es crear una aplicación web que se adapte a las necesidades de cada usuario. La arquitectura se divide en dos partes: la comunicación entre los componentes del sistema web y la comunicación entre el sistema web y la plataforma de agentes. Se utiliza un servicio web como intermediario para facilitar el intercambio de información. Los agentes son responsables de monitorear el comportamiento del usuario y adaptar la interfaz en consecuencia. Esta solución ofrece una forma flexible y eficiente de crear aplicaciones web personalizadas. La arquitectura propuesta en el artículo permite una adaptación dinámica de la interfaz al usuario, mejorando así la experiencia de usuario.

## Bibliografía

Paola J. Rodríguez C., Santiago Gómez R., (2007). Arquitectura de Comunicación entre Frameworks JadeSymfony., 6 Páginas. Recuperado de <https://revistas.unal.edu.co/index.php/avances/article/view/9720/10250>



## Reflexión

El artículo nos presenta una solución innovadora para crear experiencias de usuario más personalizadas en aplicaciones web. Al integrar una plataforma de agentes como Jade con un framework web como Symfony, se logra que las aplicaciones se adapten a las necesidades y preferencias individuales de cada usuario. Esta integración permite que las interfaces sean más intuitivas y eficientes, mejorando significativamente la interacción entre el usuario y la aplicación. Sin embargo, es importante considerar que la implementación de esta arquitectura requiere de un profundo conocimiento tanto de desarrollo web como de sistemas multiagente, lo que podría representar un desafío para algunos equipos de desarrollo.

# Análisis de secuencias discretas para la detección de Patrones de Diseño de Software

El artículo explora una nueva forma de asistir a los programadores durante el desarrollo de software. Mediante el uso de Modelos de Markov de Orden Variable (VOM), se analiza la secuencia de acciones que realiza un programador al construir una aplicación. Estos modelos permiten identificar patrones en el código y predecir qué patrón de diseño está intentando implementar el programador. Al reconocer estos patrones, el sistema puede ofrecer sugerencias personalizadas, como snippets de código o advertencias sobre posibles errores, lo que agiliza el desarrollo y mejora la calidad del software. La investigación demuestra que esta técnica es efectiva y precisa, con un alto porcentaje de aciertos en la identificación de patrones de diseño.

## Bibliografía

Silva Lograño, J. F., Berdún, L., Armentano, A., & Amandi, A. (2010). Análisis de secuencias discretas para la detección de Patrones de Diseño de Software, 12 Páginas. Recuperado de <https://sedici.unlp.edu.ar/bitstream/handle/10915/152663/Documento%20completo.pdf-PDFA.pdf?sequence=1&isAllowed=y>



## Reflexión

Este trabajo representa un avance significativo en el campo de la ingeniería de software, al acercarnos a un futuro donde las herramientas de desarrollo sean capaces de comprender y anticipar las necesidades de los programadores. La aplicación de modelos de inteligencia artificial, como los VOM, en el contexto de la programación, abre un abanico de posibilidades para mejorar la productividad y la eficiencia en el desarrollo de software. Sin embargo, es importante considerar que esta tecnología aún está en desarrollo y que su implementación a gran escala requerirá de una mayor investigación y adaptación a diferentes lenguajes de programación y entornos de desarrollo. Además, es fundamental garantizar que estas herramientas no limiten la creatividad de los programadores, sino que la complementen y potencien.

# Lenguajes de patrones de diseño de software bajo una perspectiva cognoscitivista

El artículo explora el concepto de "lenguajes de patrones", una herramienta tomada de la arquitectura y aplicada al diseño de software. Se centra en la idea de "completitud" de estos lenguajes, es decir, qué tan completos y exhaustivos son. El autor propone un nuevo marco teórico para evaluar esta completitud, considerando que los lenguajes de patrones son sistemas vivos que evolucionan con el tiempo y están influenciados por la forma en que los programadores piensan. Se presentan criterios específicos para evaluar la completitud de un lenguaje de patrones, como la riqueza de las conexiones entre los patrones, la claridad de sus descripciones y la existencia de patrones prototípicos. Al aplicar estos criterios, se busca mejorar la calidad y efectividad de los lenguajes de patrones en el desarrollo de software.



## Bibliografía

- Calderon Castro, A., (2011). Lenguajes de patrones de diseño de software bajo una perspectiva cognoscitivista, 23 Páginas. Recuperado de <https://revistas.ucr.ac.cr/index.php/intersedes/article/view/829/890>

## Reflexión

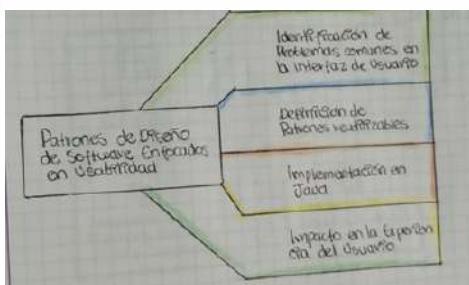
El artículo nos invita a repensar la forma en que concebimos y utilizamos los lenguajes de patrones en el desarrollo de software. Al introducir un enfoque más profundo y detallado para evaluar la completitud de estos lenguajes, nos desafía a ir más allá de una simple catalogación de patrones. La propuesta de considerar los lenguajes de patrones como sistemas vivos y en constante evolución, estrechamente vinculados a los procesos cognitivos de los desarrolladores, abre nuevas posibilidades para mejorar la calidad y la eficacia del diseño de software. Al aplicar los criterios de evaluación propuestos, podemos construir lenguajes de patrones más ricos, coherentes y adaptables, lo que a su vez facilita la colaboración, la resolución de problemas y la creación de sistemas de software más robustos y mantenibles.

# Ambientes de desarrollo de software basados en patrones de usabilidad

El artículo presenta un proyecto de investigación cuyo objetivo es desarrollar patrones de diseño de software enfocados en la usabilidad de interfaces. Estos patrones proporcionarán soluciones predefinidas para mejorar la interacción entre usuarios y aplicaciones. El proyecto busca crear herramientas y conocimientos que ayuden a los desarrolladores a crear interfaces más intuitivas y fáciles de usar. Para ello, los investigadores identificarán y clasificarán problemas comunes en interfaces de usuario, y definirán patrones con soluciones efectivas y reutilizables. Los resultados pueden mejorar significativamente la calidad y la experiencia del usuario en diversas aplicaciones.

## Reflexión

Este proyecto de investigación aborda un aspecto clave en el desarrollo de software: la experiencia del usuario. Al enfocarse en patrones de diseño para mejorar la usabilidad, los investigadores contribuyen a la tendencia creciente en la industria tecnológica hacia interfaces más intuitivas y accesibles. El uso de patrones no solo agiliza el desarrollo, sino que también asegura mayor coherencia y calidad en las interfaces. Sin embargo, dado que la usabilidad está en constante evolución, estos patrones deberán adaptarse con el tiempo a nuevas tecnologías y tendencias. Además, será crucial evaluar su impacto real en la experiencia del usuario mediante pruebas de usabilidad a gran escala.



## Bibliografía

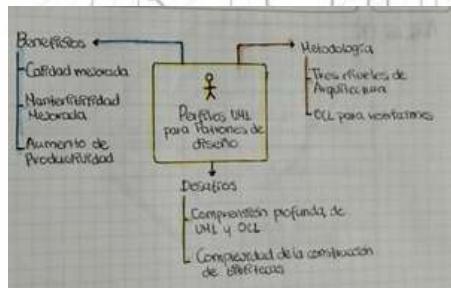
- Merlino, H., Vranić, A., Rodríguez, D., Pytel, P., García-Martínez, R. (2011). Ambientes de desarrollo de software basados en patrones de usabilidad., 4 Páginas. Recuperado de <https://n9.cl/1il1r>

# Formalización de patrones de diseño de comportamiento

Este artículo presenta una propuesta innovadora para especificar y documentar patrones de diseño de comportamiento utilizando Perfiles UML. Al definir una arquitectura de patrones en tres niveles y emplear el lenguaje OCL para establecer restricciones, los autores logran una representación precisa y formal de estos patrones. Esta metodología no solo facilita la comprensión y comunicación de los patrones, sino que también permite su validación y reutilización en diferentes proyectos. Además, el trabajo sienta las bases para futuras investigaciones en áreas como la definición de antipatrones, la detección automática de patrones y la creación de frameworks para su validación.

## Bibliografía

- AA.Cortez, A. Garis, D. Riesco. (2011). Formalización de patrones de diseño de comportamiento., 4 Páginas. Recuperado <https://n9.cl/b2m9m>



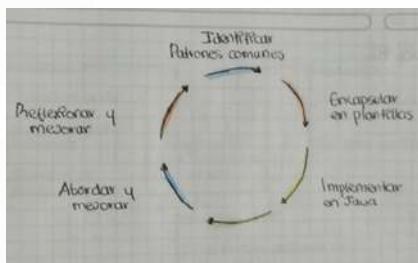
## Reflexión

La utilización de Perfiles UML para especificar patrones de diseño representa un avance significativo en la ingeniería de software. Al proporcionar un mecanismo estándar y formal para definir y documentar estos patrones, se contribuye a mejorar la calidad y la mantenibilidad del software. Sin embargo, es importante destacar que la adopción de esta metodología requiere una comprensión profunda tanto de UML como de OCL. Además, la construcción de una biblioteca completa de perfiles para todos los patrones existentes puede ser una tarea compleja y demandante en términos de tiempo y recursos. No obstante, los beneficios a largo plazo en términos de productividad y calidad de software justifican plenamente este esfuerzo.

# Implementación de Patrones de Diseño Paralelos en JAVA como una biblioteca de clases de Objetos Paralelos

El artículo propone un enfoque para diseñar y desarrollar algoritmos paralelos de manera más eficiente y reutilizable. Los autores argumentan que muchos algoritmos paralelos comparten estructuras de control comunes, a pesar de resolver problemas diferentes. Estos patrones de diseño comunes pueden ser encapsulados en plantillas o esqueletos algorítmicos, facilitando la creación de nuevos algoritmos paralelos. El objetivo principal es reducir la complejidad de la programación paralela al identificar y reutilizar estas estructuras de control comunes. Para lograrlo, se propone una metodología basada en la identificación de paradigmas de programación paralela, como el "divide y vencerás". Estos paradigmas son luego implementados en Java como plantillas genéricas que pueden ser especializadas para resolver problemas específicos.

## Reflexión



## Bibliografía

M. Rossainz López, J.J. Varela Toledo , I. Pineda Torres, M. Capel Tuñón., (2012). Implementación de Patrones de Diseño Paralelos en JAVA como una biblioteca de clases de Objetos Paralelos., 6 Páginas. Recuperado de [https://jornadassarteco.org/js2012/papers/paper\\_68.pdf](https://jornadassarteco.org/js2012/papers/paper_68.pdf)

El artículo nos invita a repensar la forma en que abordamos la programación paralela. Al identificar y encapsular patrones de diseño comunes, se nos ofrece una herramienta poderosa para construir software más eficiente y escalable. Sin embargo, esta propuesta también plantea nuevos desafíos. ¿Cómo garantizamos la portabilidad de estas plantillas a diversas arquitecturas? ¿Cómo descubrimos y categorizamos nuevos patrones? Y quizás lo más importante, ¿cómo integramos estos conceptos en la educación de las futuras generaciones de programadores? Estas preguntas nos recuerdan que, aunque hemos avanzado significativamente, el camino hacia una programación paralela verdaderamente intuitiva y accesible aún está en construcción.

# Clasificación de los patrones de diseño idóneos en programación Android

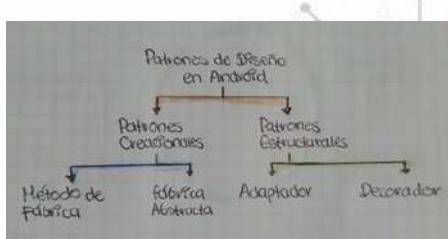
El artículo profundiza en la importancia de los patrones de diseño en el desarrollo de aplicaciones para Android, especialmente para aquellos programadores que se inician en esta plataforma. Explica que los patrones de diseño son como plantillas o soluciones predefinidas para problemas comunes en la programación, lo que permite escribir código más limpio, eficiente y fácil de mantener. Se exploran diferentes tipos de patrones, como los creacionales (que se ocupan de la creación de objetos) y los estructurales (que organizan las clases y objetos), y se detallan algunos ejemplos concretos como el Factory Method, Abstract Factory y Adapter. El artículo concluye que el uso de patrones de diseño es una práctica recomendada para cualquier desarrollador de Android, ya que contribuye a mejorar la calidad general de las aplicaciones y a facilitar el trabajo en equipo.

## Bibliografía

- Osuna Tirado J. L., Ibarra Astorga J. A., Lepe Mendoza J. C., Reyes Ramírez R.  
U. & Peraza Garzon A., (2019). Clasificación de los patrones de diseño idóneos en programación Android., 8 Páginas. Recuperado de <https://redtis.org/index.php/Redtis/article/view/33/53>

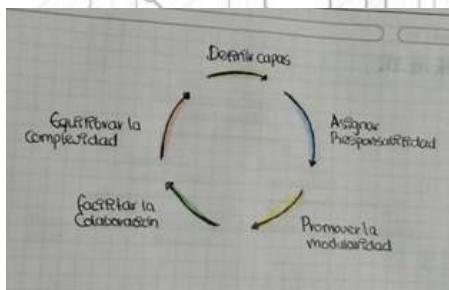
## Reflexión

Este artículo resalta la importancia de aprender y aplicar los patrones de diseño en el desarrollo de software, especialmente en un entorno tan dinámico y exigente como el desarrollo de aplicaciones móviles. Al utilizar patrones de diseño, los desarrolladores pueden construir aplicaciones más robustas, escalables y mantenibles, lo que a su vez se traduce en una mejor experiencia de usuario. Sin embargo, es fundamental comprender que los patrones de diseño no son una solución mágica para todos los problemas. Es necesario elegir el patrón adecuado para cada situación y tener una comprensión profunda de los principios de la programación orientada a objetos para poder aplicarlos de manera efectiva. En definitiva, la inversión de tiempo en aprender y dominar los patrones de diseño es una inversión en el futuro profesional de cualquier desarrollador.



# Arquitectura de software académica para la comprensión del desarrollo de software en capas

La arquitectura en capas es un patrón de diseño de software que organiza un sistema en niveles funcionales bien definidos, con el objetivo de mejorar la modularidad, la mantenibilidad y la escalabilidad. Cada capa se encarga de una tarea específica, como la interfaz de usuario, la lógica de negocio o el acceso a datos. Esta estructura permite una mayor separación de responsabilidades, facilitando la comprensión, el desarrollo y la modificación del software. Al encapsular funcionalidades en capas, se promueve la reutilización de código, se simplifican las pruebas y se facilita la adaptación a nuevos requerimientos. La arquitectura en capas es ampliamente utilizada en el desarrollo de aplicaciones modernas, ya que ofrece una base sólida para construir sistemas complejos y robustos.



## Reflexión

La arquitectura en capas es una herramienta valiosa para cualquier desarrollador de software que busque crear aplicaciones sólidas y mantenibles. Al dividir un sistema en capas bien definidas, se promueve una mejor organización del código, lo que facilita la colaboración en equipos y reduce la probabilidad de errores. Sin embargo, es importante destacar que la elección del número de capas y la complejidad de cada una dependerá de las características específicas del proyecto. Un diseño excesivamente detallado puede resultar en una sobre ingeniería, mientras que una arquitectura demasiado simple puede limitar la escalabilidad del sistema. En última instancia, el éxito de una arquitectura en capas radica en encontrar un equilibrio entre la flexibilidad y la complejidad, siempre teniendo en cuenta los requisitos del proyecto y las tecnologías disponibles.

## Bibliografía

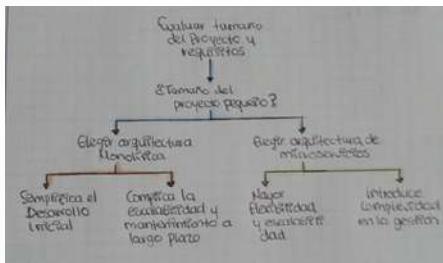
Darío G. Cardacci., (2015). Arquitectura de software académica para la comprensión del desarrollo de software en capas., 23 Páginas. Recuperado de <https://www.econstor.eu/bitstream/10419/130825/1/837816424.pdf>

# Monolitos vs. Microservicios en Arquitectura de Software: Perspectivas para un Desarrollo Eficiente

El artículo compara y contrasta dos enfoques principales para diseñar aplicaciones: arquitecturas monolíticas y de microservicios. Las arquitecturas monolíticas agrupan todas las funcionalidades de una aplicación en un solo bloque de código, lo que simplifica el desarrollo inicial pero dificulta la escalabilidad y el mantenimiento a largo plazo. Por otro lado, las arquitecturas de microservicios descomponen la aplicación en servicios más pequeños e independientes, lo que ofrece mayor flexibilidad y escalabilidad, pero introduce complejidad en la gestión. La elección entre una arquitectura y otra depende de factores como el tamaño del proyecto, los requisitos y las capacidades del equipo de desarrollo. En resumen, las arquitecturas monolíticas son ideales para proyectos pequeños y con requisitos estables, mientras que las arquitecturas de microservicios son más adecuadas para proyectos grandes y complejos que requieren mayor flexibilidad y escalabilidad.

## Reflexión

La elección entre una arquitectura monólica y una de microservicios es una decisión estratégica que puede marcar la diferencia en el éxito a largo plazo de un proyecto de software. Ambas arquitecturas tienen sus fortalezas y debilidades, y la mejor opción dependerá de factores específicos como el tamaño del proyecto, los requisitos de escalabilidad, la experiencia del equipo y la tolerancia al riesgo. Si bien las arquitecturas monolíticas ofrecen una implementación más sencilla, las arquitecturas de microservicios brindan una mayor flexibilidad y escalabilidad, adaptándose mejor a los entornos tecnológicos actuales en constante evolución. Es fundamental realizar un análisis detallado de las necesidades del proyecto y sopesar cuidadosamente los pros y los contras de cada enfoque antes de tomar una decisión.

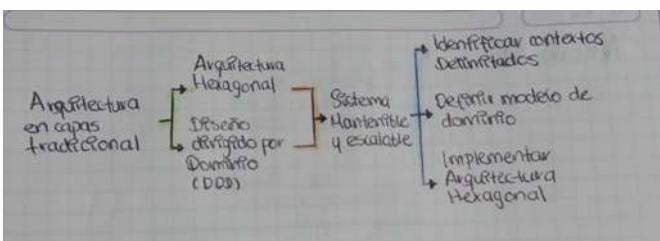


## Bibliografía

- (2020). Monolitos vs. Microservicios en Arquitectura de Software: Perspectivas para un Desarrollo Eficiente, 13 Páginas, M. Rossainz López, J.J. Varela Toledo, I. Pineda Torres, M. Capel Tuñón.,  
(2012). Implementación de Patrones de Diseño Paralelos en JAVA como una biblioteca de clases de Objetos Paralelos, 6 Páginas. Recuperado de [https://jornadassarteco.org/jsp2012/papers/paper\\_68.pdf](https://jornadassarteco.org/jsp2012/papers/paper_68.pdf)

# Implementación de una Arquitectura de Software guiada por el dominio:

El artículo explora la transformación de una arquitectura de software tradicional en capas a una arquitectura hexagonal, utilizando Diseño Dirigido por Dominio (DDD). La arquitectura hexagonal ofrece una estructura más robusta y adaptable al separar la lógica de negocio de otros componentes, permitiendo cambios tecnológicos sin afectar el núcleo. El DDD, que modela el dominio del negocio con precisión, facilita la comunicación entre desarrolladores y expertos. Combinando ambas técnicas, se obtiene una arquitectura más mantenible y escalable. El artículo presenta un caso de estudio práctico de esta transformación aplicada a una plataforma de empleo, detallando la identificación de contextos delimitados, el modelo de dominio y la implementación de la arquitectura hexagonal con puertos y adaptadores.



## Reflexión

El artículo ofrece una visión profunda sobre cómo la arquitectura hexagonal, combinada con el diseño dirigido por dominio (DDD), puede transformar el desarrollo de software. Al separar la lógica de negocio de otros componentes y modelar el dominio con precisión, estas técnicas brindan mayor flexibilidad, mantenibilidad y capacidad de adaptación a cambios. El caso práctico de una plataforma de empleo muestra cómo estos conceptos teóricos se aplican a soluciones concretas. Esta combinación no solo mejora la calidad del software, sino también facilita la colaboración entre equipos y mejora la comprensión del negocio. En resumen, el artículo nos invita a adoptar estas prácticas para construir sistemas más robustos y escalables.

## Bibliografía

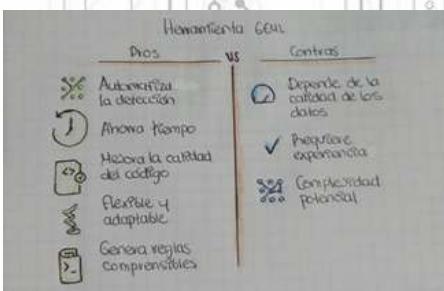
Mauro Germán Cambarieri, Federico Difabio, Nicolás García Martínez, (2020). Implementación de una Arquitectura de Software guiada por el dominio 18 Páginas. Recuperado de [https://sedici.unlp.edu.ar/bitstream/handle/10915/115198/Documento\\_completo.pdf-PDFA.pdf?sequence=1&isAllowed=true](https://sedici.unlp.edu.ar/bitstream/handle/10915/115198/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=true).

# Detección de patrones de diseño con GEML: discusión y enfoque práctico

El artículo presenta GEML, una nueva técnica de aprendizaje automático diseñada para detectar patrones de diseño en código fuente de manera automática. A diferencia de métodos tradicionales que requieren una definición manual de los patrones, GEML aprende a reconocerlos a partir de ejemplos existentes en un repositorio de código. Esta herramienta genera reglas comprensibles para humanos que describen los patrones, lo que facilita su interpretación. GEML ha demostrado ser flexible, adaptable a diferentes estilos de programación y preciso en la detección de diversos patrones. Además, ofrece la posibilidad de ser configurada por el ingeniero de software para ajustarse a necesidades específicas. En conclusión, GEML es una herramienta prometedora para mejorar la calidad del software al facilitar la comprensión, mantenimiento y evolución del código.

## Bibliografía

Jose Raul Romero, Rafael Barbudo, Aurora Ramirez , Francisco Servant, (2019). Detección de patrones de diseño con GEML: discusión y enfoque práctico, 14 Páginas. Recuperado de [https://fservant.github.io/papers/Romero\\_Barbudo\\_Ramirez\\_Servant\\_JISBD21.pdf](https://fservant.github.io/papers/Romero_Barbudo_Ramirez_Servant_JISBD21.pdf)



## Reflexión

El artículo sobre GEML nos presenta una herramienta prometedora para el futuro del desarrollo de software. Al automatizar la detección de patrones de diseño, GEML no solo ahorra tiempo a los desarrolladores, sino que también contribuye a mejorar la calidad y la consistencia del código. La capacidad de aprender de ejemplos y generar reglas comprensibles hace de GEML una herramienta versátil y adaptable a diversos proyectos. Sin embargo, es importante considerar que la efectividad de GEML dependerá en gran medida de la calidad y cantidad de datos de entrenamiento disponibles. Además, aunque las reglas generadas son más interpretables que los modelos de caja negra típicos en el aprendizaje automático, todavía pueden requerir cierta experiencia para ser completamente comprendidas.

# Patrón de Diseño BeaconAction Manager para comunicar Aplicaciones Móviles(IoT)

El artículo explora el mundo en constante evolución del Internet de las Cosas (IoT) y el papel fundamental que desempeñan los beacons en esta transformación. Los beacons, pequeños dispositivos que transmiten señales de Bluetooth de bajo consumo, permiten a las aplicaciones móviles determinar la ubicación de un usuario y ofrecer experiencias personalizadas basadas en su entorno. El artículo presenta un nuevo patrón de diseño llamado "Beacon Action Manager", diseñado para estandarizar el desarrollo de aplicaciones móviles que interactúan con beacons. Este patrón ofrece una estructura clara y eficiente para manejar los eventos que ocurren cuando un dispositivo móvil entra en contacto con un beacon. A través de varios ejemplos de aplicaciones reales, el artículo demuestra la utilidad y versatilidad de este patrón de diseño en diferentes contextos.

## Reflexión



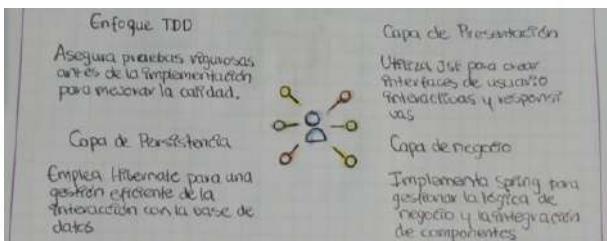
## Bibliografía

- (2023). Patrón de Diseño Beacon Action Manager para comunicar Aplicaciones Móviles (IoT). Recuperado de <https://n9.cl/7myp1>

El artículo nos invita a reflexionar sobre cómo la tecnología, en este caso los beacons y el IoT, está redefiniendo nuestra interacción con el entorno. Al permitir que los objetos cotidianos se comuniquen entre sí y con nuestros dispositivos móviles, los beacons abren un abanico de posibilidades para crear experiencias más personalizadas y eficientes. El patrón de diseño propuesto, "Beacon Action Manager", es un paso importante hacia la estandarización de este tipo de desarrollo, lo que podría acelerar la adopción de estas tecnologías en diversos sectores. Sin embargo, es fundamental considerar las implicaciones éticas y de privacidad que conlleva esta creciente interconexión. ¿Hasta qué punto estamos dispuestos a compartir nuestra ubicación y nuestros hábitos con las aplicaciones? ¿Cómo podemos garantizar la seguridad de los datos que se generan a través de estos sistemas?

# Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo

El artículo explora la sinergia entre la arquitectura de software y las metodologías ágiles, con un enfoque particular en el desarrollo guiado por pruebas (TDD). Los autores proponen un marco de trabajo innovador que combina tecnologías consolidadas como JSF, Spring y Hibernate para construir una arquitectura en capas robusta y adaptable. Al aplicar TDD de manera rigurosa en cada una de las capas de la aplicación (presentación, negocio y persistencia), se garantiza un desarrollo iterativo y de alta calidad. Este enfoque permite que el software evolucione de forma incremental, respondiendo de manera ágil a los cambios en los requisitos del cliente. Además, al escribir las pruebas antes de implementar el código, se reduce significativamente la cantidad de errores y se mejora la calidad del producto final.



## Reflexión

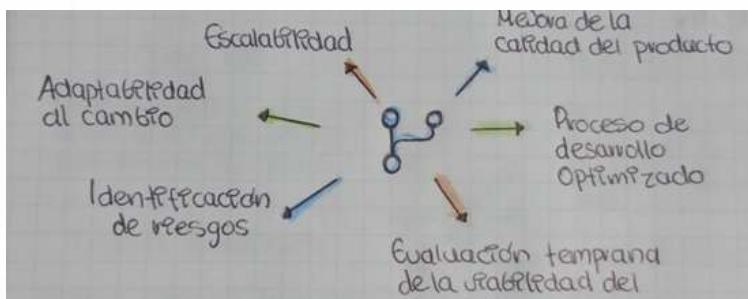
Este artículo presenta una visión interesante sobre cómo combinar la estructura y planificación de la arquitectura de software con la flexibilidad y adaptabilidad de las metodologías ágiles. La propuesta de utilizar TDD en cada capa es particularmente atractiva, ya que permite un desarrollo más granular y enfocado en la calidad. Sin embargo, es importante considerar que la implementación exitosa de este enfoque requiere de un equipo de desarrollo con una sólida comprensión tanto de la arquitectura como de las prácticas ágiles. Además, la elección de las tecnologías y herramientas adecuadas es fundamental para garantizar el éxito del proyecto. A pesar de los beneficios evidentes, es crucial evaluar si este enfoque es el más adecuado para cada proyecto, considerando factores como el tamaño del equipo, la complejidad del sistema y los requisitos del cliente.

## Bibliografía

Luis Vivas, Mauro Cambarieri, Nicolás García, Marcelo Martínez, Horacio Petroff, Muñoz Abbate. (2013). Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo, 10 Páginas. Recuperado de <https://n9.cl/rldhdj>

# Arquitectura de Software

El artículo explora la sinergia entre la arquitectura de software y las metodologías ágiles, con un enfoque particular en el desarrollo guiado por pruebas (TDD). Los autores proponen un marco de trabajo innovador que combina tecnologías consolidadas como JSF, Spring y Hibernate para construir una arquitectura en capas robusta y adaptable. Al aplicar TDD de manera rigurosa en cada una de las capas de la aplicación (presentación, negocio y persistencia), se garantiza un desarrollo iterativo y de alta calidad. Este enfoque permite que el software evolucione de forma incremental, respondiendo de manera ágil a los cambios en los requisitos del cliente. Además, al escribir las pruebas antes de implementar el código, se reduce significativamente la cantidad de errores y se mejora la calidad del producto final.



## Reflexión

La arquitectura de software, a menudo subestimada, es el cimiento sobre el cual se construyen sistemas complejos. Este artículo resalta de manera convincente la importancia de invertir tiempo y esfuerzo en definir una arquitectura sólida. Es como diseñar los planos de una casa antes de comenzar a construir: una buena planificación evita costosas modificaciones y garantiza que la estructura final sea funcional y resistente. En un mundo donde la tecnología evoluciona rápidamente, la capacidad de adaptar y escalar los sistemas es crucial. Una arquitectura bien diseñada facilita estos cambios, haciendo que el software sea más duradero y adaptable a las necesidades cambiantes de los usuarios. Sin embargo, es importante recordar que la arquitectura no es estática; debe evolucionar junto con el sistema a medida que se adquieran nuevos conocimientos y se enfrentan nuevos desafíos.

## Bibliografía

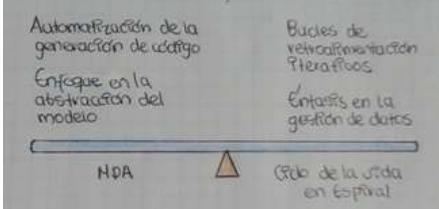
Cañete Núñez, L., (2014). Arquitectura de Software. Recuperado de <https://revista.jovenclub.cu/arquitectura-de-software/>

# La Arquitectura de Software en el Proceso de Desarrollo: Integrando MDA al Ciclo de Vida en Espiral

El artículo explora cómo combinar la Arquitectura Dirigida por Modelos (MDA) con el ciclo de vida en espiral para mejorar el desarrollo de software. La MDA propone crear modelos abstractos del sistema que evolucionan de forma automática hacia código, mientras que el ciclo de vida en espiral es un enfoque iterativo que enfatiza la gestión de riesgos. Al integrar ambas metodologías, se logra una mayor trazabilidad entre los requisitos y el código, lo que facilita la adaptación a cambios y reduce errores. Además, la MDA permite separar los aspectos de negocio de la tecnología, haciendo el software más portátil y adaptable. Los autores proponen un proceso iterativo donde se crean y refinan modelos en cada ciclo, evaluando los riesgos y obteniendo retroalimentación de los usuarios. En resumen, la combinación de MDA y el ciclo de vida en espiral ofrece una forma más eficiente y efectiva de desarrollar software, al permitir una mayor flexibilidad, calidad y control sobre el proceso de desarrollo.

## Bibliografía

Valeria S. Meaurio, Eric Schmieder, (2013). La Arquitectura de Software en el Proceso de Desarrollo: Integrando MDA al Ciclo de Vida en Espiral , 5 Páginas. Recuperado de <https://revistas.unla.edu.ar/software/article/view/103>



## Reflexión

La integración de MDA y el ciclo de vida en espiral representa un avance significativo en las metodologías de desarrollo de software. Al combinar la capacidad de la MDA de generar código a partir de modelos abstractos con la naturaleza iterativa y centrada en riesgos del ciclo de vida en espiral, se obtiene un enfoque más ágil y adaptable. Esta sinergia permite a los equipos de desarrollo responder de manera más efectiva a los cambios en los requisitos y reducir el tiempo de comercialización. Sin embargo, es importante destacar que la implementación exitosa de esta combinación requiere una sólida comprensión tanto de MDA como del ciclo de vida en espiral, así como herramientas y procesos adecuados para gestionar la transformación de modelos. Además, la adopción de esta metodología puede requerir una inversión inicial en capacitación y adaptación de los equipos de desarrollo.

# Arquitectura de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia

El artículo presenta una solución innovadora para optimizar el control de inventario en el Instituto Tecnológico de Orizaba mediante la implementación de un sistema RFID. Esta solución combina una aplicación web y una aplicación móvil para dispositivos RFID, permitiendo una gestión más precisa y eficiente de los activos. La arquitectura de software del sistema se basa en una estructura de tres niveles: presentación, aplicación y persistencia. La aplicación web, que actúa como el cerebro del sistema, utiliza una arquitectura MVC (Modelo-Vista-Controlador) para separar las distintas responsabilidades del software. Por otro lado, la aplicación móvil, diseñada para dispositivos Handheld CS101-2, se estructura en capas, con una interfaz de usuario intuitiva y una lógica de negocio que interactúa directamente con las etiquetas RFID.

## Reflexión

Implementación de un sistema RFID		
Pros	vs	Contras
Reducción de Pérdidas		\$ Costos Involucrados
Mejora de Eficiencia		Infraestructura tecnológica
Estado preciso		Seguridad de datos

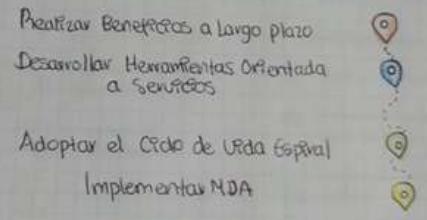
## Bibliografía

(Lagunes García, Gerardo; López Martínez, Ignacio; Peláez Camarena, Gustavo S; Abud Figueroa, María Antonieta; Olivares Zepahua, Beatriz Alejandra., (2015). Arquitectura de software de una aplicación móvil para desarrollar un sistema de identificación por radiofrecuencia., 17 Páginas. Recuperado de <https://www.redalyc.org/pdf/5122/512251501004.pdf>

La implementación de sistemas RFID para el control de inventario representa un avance significativo en la gestión de activos. Sin embargo, es fundamental considerar los costos involucrados, la necesidad de una infraestructura tecnológica adecuada y la importancia de garantizar la seguridad de los datos. Además, es crucial evaluar la escalabilidad de la solución para adaptarse a entornos más grandes y complejos. A pesar de estos desafíos, los beneficios de la tecnología RFID, como la reducción de pérdidas, la mejora de la eficiencia y la toma de decisiones basada en datos, hacen que sea una inversión atractiva para muchas organizaciones. Es importante destacar que esta tecnología no solo es aplicable a instituciones educativas, sino que también puede ser utilizada en diversos sectores, como la logística, la manufactura y el retail.

# Arquitectura orientada a servicios para software de apoyo para el proceso personal de software

El artículo presenta una propuesta para desarrollar una herramienta de software que facilite la implementación del Proceso Personal de Software (PSP). El PSP es un método estructurado para que los desarrolladores mejoren su desempeño individualmente, a través de la medición y el análisis de sus procesos. La herramienta propuesta busca automatizar la captura de datos como el tiempo empleado, los defectos encontrados y el tamaño del código, lo que reduce la carga de trabajo del desarrollador. Además, ofrece funcionalidades para generar reportes personalizados, realizar estimaciones y administrar proyectos. La arquitectura de la herramienta se basa en servicios web, lo que permite una gran flexibilidad y escalabilidad. Los datos se recolectan a través de plugins que se integran en los entornos de desarrollo integrados (IDEs) más populares.



## Reflexión

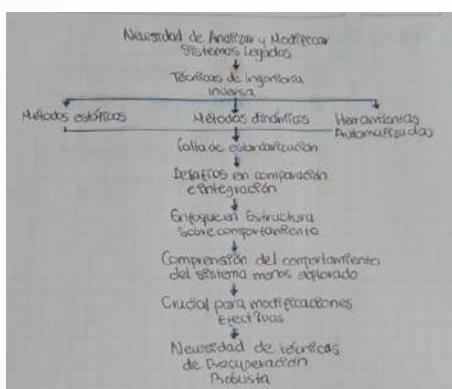
El artículo explora la creación de una herramienta que revoluciona la manera en que los desarrolladores de software abordan sus proyectos. Al automatizar la recolección de datos y proporcionar un marco de trabajo estructurado, esta herramienta facilita la implementación del Proceso Personal de Software (PSP). Esto permite a los desarrolladores obtener una visión más clara de su desempeño, identificar áreas de mejora y tomar decisiones más informadas. La arquitectura basada en servicios web de la herramienta la hace adaptable y escalable, lo que la convierte en una solución prometedora para mejorar la calidad y eficiencia en el desarrollo de software. Sin embargo, es importante destacar que la efectividad de esta herramienta dependerá en gran medida de la adopción por parte de los desarrolladores y de la calidad de los datos recopilados.

## Bibliografía

Erick Salinas, Narciso Cerpa, Pablo Rojas, (2011). Arquitectura orientada a servicios para software de apoyo para el proceso personal de software., 13 Páginas. Recuperado de [https://www.scielo.cl/scielo.php?pid=s0718-33052011000100005&script=sci\\_arttext](https://www.scielo.cl/scielo.php?pid=s0718-33052011000100005&script=sci_arttext)

# Recuperación de Arquitecturas de Software: Un Mapeo Sistemático de la Literatura

El artículo explora el desafío de comprender y modificar sistemas de software heredados a través de la ingeniería inversa. Se centra en la recuperación de la arquitectura, es decir, en reconstruir la estructura y el comportamiento de un sistema a partir de su código fuente. El estudio revela una amplia variedad de técnicas y enfoques para lograr esto, pero destaca la necesidad de una mayor estandarización en la representación de los resultados. Además, se observa una tendencia a centrarse en la estructura del sistema, dejando aún un espacio para mejorar en la recuperación del comportamiento. Los autores concluyen que la recuperación de la arquitectura es un campo de investigación activo con un gran potencial para mejorar el mantenimiento y la evolución de los sistemas de software.



## Reflexión

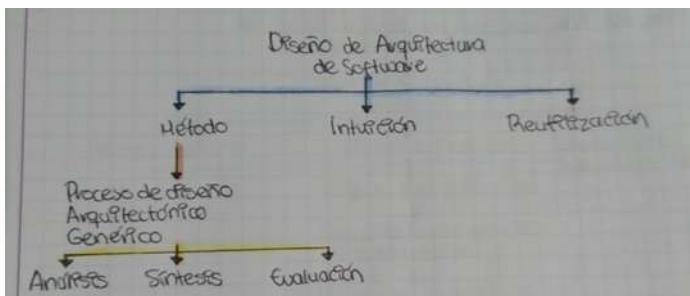
La recuperación de la arquitectura es una herramienta fundamental para hacer frente a la complejidad creciente de los sistemas de software. Sin embargo, el artículo pone de manifiesto que aún queda mucho camino por recorrer. La falta de un enfoque unificado y la dificultad para recuperar el comportamiento completo de un sistema son desafíos que deben abordarse. A medida que los sistemas se vuelven más distribuidos, complejos y basados en microservicios, la necesidad de técnicas de recuperación de arquitectura sólidas se hará aún más evidente. Es crucial continuar investigando en este campo para desarrollar herramientas y métodos que permitan a los desarrolladores comprender y modificar sistemas de software de manera más eficiente y segura.

## Bibliografía

- Martín E. Monroy, José L. Arciniegas y Julio C. Rodríguez, (2016). Recuperación de Arquitecturas de Software: Un Mapeo Sistemático de la Literatura., 20 Páginas. Recuperado de [https://www.scielo.cl/scielo.php?pid=S0718-07642016000500022&script=sci\\_arttext](https://www.scielo.cl/scielo.php?pid=S0718-07642016000500022&script=sci_arttext)

# Representación y razonamiento sobre las decisiones de diseño de arquitectura de software

El artículo propone un nuevo enfoque para el diseño de arquitecturas de software basado en la reutilización de experiencias pasadas. Mediante la técnica de Razonamiento Basado en Casos, se busca mejorar la calidad y eficiencia en el diseño de sistemas. La idea es almacenar información sobre diseños arquitectónicos previos (casos) y utilizarlos como referencia para resolver nuevos problemas. Al identificar similitudes entre casos pasados y el nuevo problema, los arquitectos pueden reutilizar soluciones probadas y tomar decisiones más informadas. Esto permite reducir el tiempo de desarrollo, minimizar errores y mejorar la calidad del producto final. El artículo presenta un modelo detallado para representar estos casos, una metodología para su aplicación y una herramienta (RADS) para facilitar su uso en la práctica.



## Reflexión

La propuesta de utilizar el Razonamiento Basado en Casos para el diseño de arquitecturas de software presenta un enfoque prometedor para mejorar la calidad y eficiencia en el desarrollo de software. Al aprender de experiencias pasadas y reutilizar soluciones probadas, los arquitectos pueden tomar decisiones más informadas y reducir el tiempo de desarrollo. Sin embargo, para garantizar su éxito a gran escala, es necesario abordar desafíos como la escalabilidad de la base de datos de casos, la adaptabilidad de la técnica a diferentes contextos y la integración con otras herramientas de desarrollo. Además, es fundamental mantener un equilibrio entre la automatización y la intervención humana en el proceso de diseño.

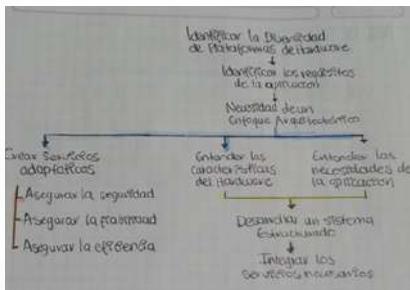
## Bibliografía

María Celeste Carignano, (2017). Representación y razonamiento sobre las decisiones de diseño de arquitectura de software, 10 Páginas. Recuperado de [https://sedici.unlp.edu.ar/bitstream/handle/10915/53411/Documento\\_completo.pdf-PDFA.pdf?sequence=1&isAllowed=true](https://sedici.unlp.edu.ar/bitstream/handle/10915/53411/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=true)

# Arquitectura de software para los actuales sistemas ciber físicos

El artículo aborda los desafíos y oportunidades que presenta el diseño de software para los sistemas ciber-físicos del futuro. Estos sistemas, que combinan componentes físicos y digitales, requieren una mayor flexibilidad y adaptabilidad debido a la diversidad de plataformas de hardware y las demandas específicas de cada aplicación. El texto propone una arquitectura de software más modular y adaptable, donde los servicios puedan ser combinados y personalizados de acuerdo a las necesidades. Además, destaca la importancia de desarrollar interfaces de programación más precisas y de considerar la heterogeneidad del hardware para lograr sistemas más eficientes y seguros.

## Reflexión



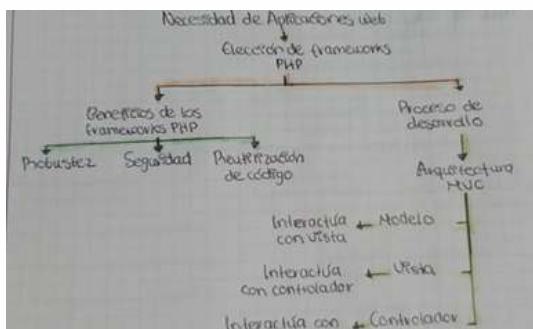
## Bibliografía

- Jeannette S. Ting, (2011). Arquitectura de software para los actuales sistemas ciber físicos, 4 Páginas. Recuperado de <https://dialnet.unirioja.es/servlet/articulo?codigo=3692798>

Este artículo nos presenta una visión desafiante pero prometedora del futuro del software. La creciente complejidad de los sistemas ciber-físicos exige una evolución en la forma en que diseñamos y desarrollamos software. La idea de una arquitectura de software más flexible y adaptable es un paso fundamental para hacer frente a esta complejidad. Sin embargo, implementar esta visión implica resolver desafíos técnicos importantes, como la creación de mecanismos eficientes para la composición y adaptación de servicios, así como el desarrollo de herramientas y metodologías que permitan a los desarrolladores crear software de manera más eficiente y confiable. Además, es fundamental considerar las implicaciones de estos avances en términos de seguridad y privacidad, ya que los sistemas ciber-físicos cada vez más sofisticados serán objetivos atractivos para los ciberataques. En resumen, este artículo nos invita a reflexionar sobre la necesidad de una mayor colaboración entre ingenieros de software, expertos en hardware y otros profesionales para dar forma al futuro de los sistemas ciber-físicos.

# Frameworks PHP basados en la arquitectura Modelo-Vista-Controlador para desarrollo de aplicaciones web

El artículo realiza una exhaustiva comparación de los principales frameworks PHP basados en la arquitectura MVC, destacando las características, ventajas y desventajas de cada uno. Se analiza en profundidad Laravel, Symfony, CodeIgniter, Zend, CakePHP y Yii, considerando aspectos como facilidad de uso, rendimiento, seguridad y compatibilidad con bases de datos. El estudio concluye que la elección del framework ideal depende de las necesidades específicas del proyecto, siendo Laravel y Symfony los más populares para proyectos grandes y complejos, mientras que CodeIgniter y Yii son más adecuados para proyectos más pequeños y desarrolladores principiantes. Todos los frameworks ofrecen sólidas características de seguridad y una amplia gama de herramientas para agilizar el desarrollo.



## Reflexión

Este análisis evidencia la importancia de los frameworks PHP MVC en el desarrollo web moderno, simplificando tareas y promoviendo buenas prácticas de programación. La diversidad de opciones disponibles demuestra la madurez del ecosistema PHP y la constante evolución de estas herramientas. Al elegir un framework, es fundamental considerar no solo las características técnicas, sino también la curva de aprendizaje, la comunidad de desarrolladores y la disponibilidad de recursos. En última instancia, el framework ideal es aquel que se adapta mejor a las necesidades del proyecto y al estilo de trabajo del desarrollador. Este tipo de estudios comparativos resultan invaluable para tomar decisiones informadas y optimizar el proceso de desarrollo web.

## Bibliografía

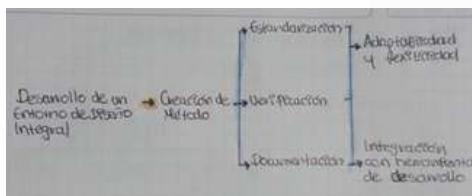
- Carlos Castillo, Andrés Marjorie Coronel Yagual Alexandra Suárez, (2023). Frameworks PHP basados en la arquitectura Modelo-Vista-Controlador para desarrollo de aplicaciones web, 9 Páginas. Recuperado de <http://scielo.senescyt.gob.ec/pdf/rctu/v10n1/1390-7697-rctu-10-01-00070.pdf>

# **Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en Nube**

El artículo aborda la complejidad de diseñar arquitecturas de software robustas y eficientes para entornos de computación en la nube (CC). Se identifica la falta de estándares y herramientas adecuadas para modelar y verificar estos diseños como un obstáculo significativo. La propuesta del artículo es un entorno de diseño integral que, a través de un metamodelo y módulos complementarios, facilita la creación y evaluación de arquitecturas de CC. Este entorno ofrece un vocabulario común para describir los componentes de una arquitectura, permite verificar la correcta aplicación de patrones de diseño y documenta las decisiones arquitectónicas tomadas. En esencia, el objetivo es proporcionar a los arquitectos una herramienta que les permita diseñar arquitecturas de CC de manera más eficiente y con mayor confianza en su calidad.

## **Reflexión**

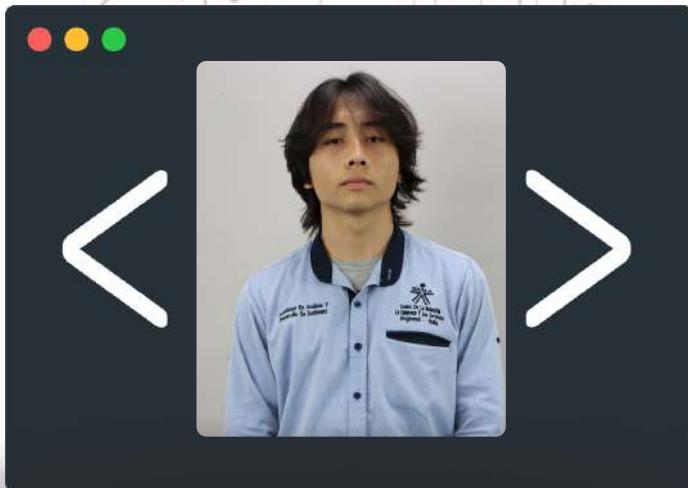
El artículo presenta un valioso aporte al campo de la ingeniería de software al abordar la creciente complejidad del diseño de arquitecturas en la nube. La propuesta de un metamodelo y un entorno de diseño integrado representa un paso significativo hacia la estandarización y automatización de este proceso. Sin embargo, para maximizar su potencial y garantizar su adopción a largo plazo, es necesario considerar varios aspectos adicionales. En primer lugar, la rápida evolución de las tecnologías y los patrones de diseño en la nube exige que la herramienta sea altamente adaptable y flexible. Un metamodelo estático podría rápidamente quedar obsoleto, por lo que es fundamental desarrollar mecanismos para actualizar y extender el modelo de manera ágil.



## **Bibliografía**

María Julia Blas, Horacio Leone, Silvio Gonnet, (2019). Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en Nube, 17 Páginas.  
Recuperado de  
[https://ri.conicet.gov.ar/bitstream/handle/11336/125130/CONICET\\_Digital\\_Nro.8053e909-ab36-4e05-a990-4d92bb067f45\\_A.pdf?sequence=2&isAllowed=y](https://ri.conicet.gov.ar/bitstream/handle/11336/125130/CONICET_Digital_Nro.8053e909-ab36-4e05-a990-4d92bb067f45_A.pdf?sequence=2&isAllowed=y)

# Willian Steban González Cortes



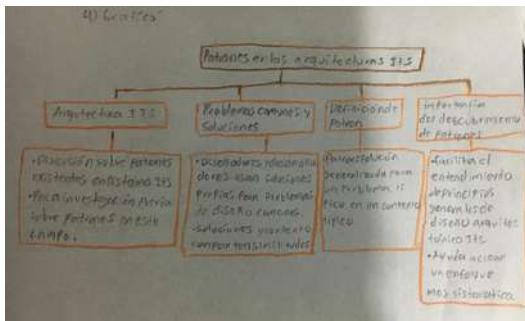
Soy desarrollador de software mi pasión es la tecnología tengo un sólido conocimiento del sector del software y un poco en electrónica. Comencé en este mundo con un constante aprendizaje empezando con HTML 5 y después me intereso la carrera el software decidí estudiarla donde allí me encontré con otros lenguajes como: java, JavaScript, MySQL, C#, Kotlin y documentación del software. Siempre busco que las ideas para nuevos programas sean eficientes y funcionales siempre estando actualizado al cambio constante de las tecnologías y el desarrollo desoftware, me gusta siempre hacerme preguntas cada que hago un código y tratar de buscar las respuestas y comunicarlas, siempre aprovecho los eventos y muestra de proyectos de algunas instituciones escuchando y estando al tanto, también me gusta mucho investigar herramientas que me ayuden en mi día a día como programador a aventurarme más en este mundo.

# Patrones de software arquitectura ITS

El artículo analiza el uso de patrones en las arquitecturas de Sistemas tutores inteligentes (ITS), destacando su escasa atención por los investigadores. A pesar de los problemas comunes entre sistemas, diseñadores suelen crear soluciones vascas. Sin embargo, estas comparten similitudes que permiten identificar Patrones, entendidos como soluciones generales a problemas recurrentes en contextos específicos. Descubrir estos patrones facilita sistematizar principios arquitectónicos.

## Reflexión

El artículo resalta la importancia de identificar Patrones en el diseño de sistemas tutores inteligentes (ITS). Para optimizar su desarrollo. Aunque los diseñadores suelen crear soluciones únicas para problemas comunes, reconocer patrones permite reutilizar enfoques efectivos y mejorar la calidad y coherencia de los sistemas. La falta de atención a estos patrones limita el potencial de los ITS. Incorporarlos al diseño arquitectónico sistematiza el proceso, haciéndolo más eficiente y efectivo.



## Bibliografía

Autor: Devedzic, vladan Harrier Andreas  
Año: 2005

# Arquitectura de componentes para el modelo de redes en SIG

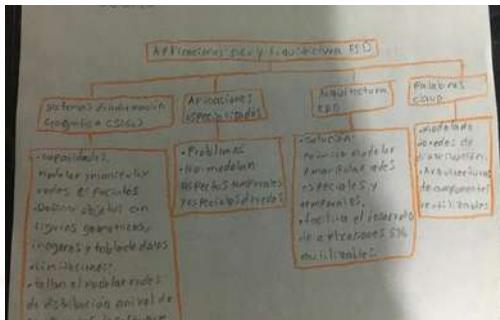
Nos habla de la arquitectura de dominio RED Permite modelar redes de distribución en sistemas de información Geográfica, aprovechando el concepto de grafo espaciotemporal. RED se enfoca en subdominios como redes hidráulicas, eléctricas y oleoductos, y cuenta con Componentes para modelado, calculo y optimización. Permite agregar nuevos atributos a las entidades en tiempo de Ejecución, y utiliza una arquitectura en capas que integra elementos middle ware y visualización. La implementación esta organiza en paquetes que agrupan componentes genéricos y específicos. RED fue evaluada mediante un prototipo y aporta una solución genérica para el desarrollo de aplicaciones S16 en este dominio. El artículo también describe el método utilizado para su desarrollo.

## Reflexión

La arquitectura de dominio RED aborda la necesidad de contar con soluciones integrales para el desarrollo de aplicaciones S16 en redes de distribución. Su enfoque en subdominios específicos y el uso de componentes reutilizables para modelado, cálculo y optimización, la convocatoria en un valioso aporte. El método de desarrollo aplicado, que separa la ingeniería de dominio de la ingeniería de aplicación, destaca la importancia de contar con procesos adecuados para el diseño de arquitecturas de software. Esto facilita el reusó y evolución de la arquitectura. En general el artículo ilustra como una arquitectura de dominio bien concebida puede simplificar significativamente el desarrollo de aplicaciones en áreas específicas. Esta representa una lectura valiosa para la ingeniería de software orientada a dominios.

## Bibliografía

Autor: Jose Rivas, Jonas Montilva  
Año: 2003



# Patrones de diseño GOF (the Gang of Four) en el contexto de procesos de desarrollo de aplicaciones orientadas a la web

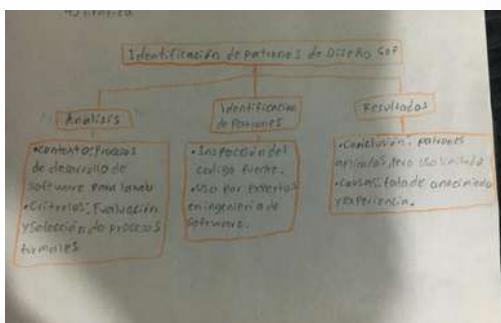
El estudio investigó que patrones de diseño GOF se usan en el desarrollo de aplicaciones web en Colombia. Se analizaron varios proyectos y se encuestó a expertos del tema. Los resultados mostraron que los patrones creacionales son los más populares, seguidos por los estructurales y de Comportamiento. El uso de estos patrones mejora la calidad y eficiencia del software. Sin embargo, se encontró que muchos desarrolladores desconocen o no aplican estos patrones de manera consistente. El estudio recomienda fomentar la educación en Patrones de diseño y el uso de herramientas que faciliten su implementación.

## Bibliografía

Autor: Carlos Guerrero,  
johanna Suarez y Luz  
Gutiérrez  
Año: 2013

## Reflexión

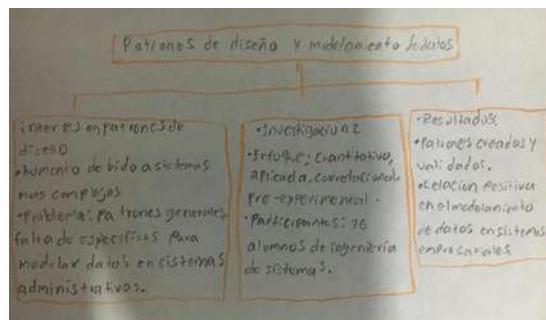
Los patrones de diseño son como los manuales de Construcción que usan los mejores arquitectos para crear edificios super resistentes y bonitos. En la programación, estos Manuales nos ayudan a crear videojuegos y apps increíbles. ¡Es como tener un super kit de construcción para crear videojuegos y apps increíbles! Al igual que tu castillo de lego, tus programas Serán más fuertes y divertidas si usas estos manuales especiales. Este estudio nos dice que muchos programadores colombianos ya están usando estos manuales y que hacen que sus programas colombianos ya están usando estos manuales y que hacen que sus programas sean mejores. ¡Así que la próxima vez que crees algo en la computadora, recuerda que puedes usar estos trucos para hacerlo aún más genial!



# Aplicación de patrones de diseño estructurales para el modelamiento de clases de los sistemas empresariales

El artículo se centra en la mejora de la calidad del diseño de software a través de la aplicación de Patrones de diseño estructurales.

Estos patrones son como plantillas o recetas Predefinidas que ofrecen soluciones probadas para problemas comunes en el desarrollo de software. Al utilizar estos Patrones los programadores pueden construir sistemas mas robustos, flexibles y fáciles de mantener. Con el objetivo principal del estudio es determinar si el uso de estos patrones de diseño en sistemas empresariales.



## Reflexión

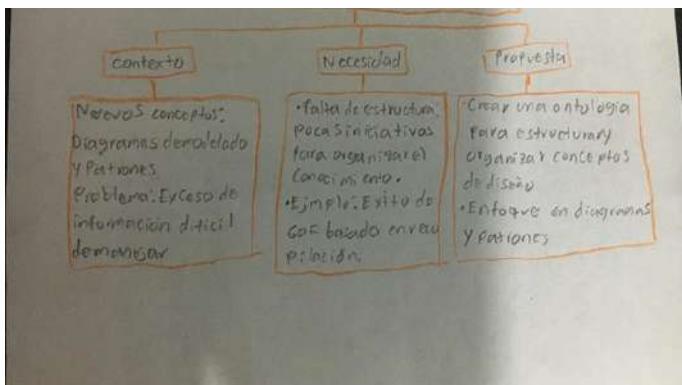
Este estudio nos muestra como los Patrones de diseño pueden hacer una gran diferencia en la calidad Software. Básicamente, estos patrones funcionan como soluciones Prácticas para problemas comunes, permitiendo que el código sea más robusto, flexible y fácil de Mantener. La investigación destaca que aplicar estos Patrones ayuda a que los modelos de datos sean más claros y coherentes, facilita encontrar y corregir errores Y hace que, en el futuro, sea más sencillo modificar el sistema.

## Bibliografía

Autor: Manuel Mariano Zúñiga  
Año: 2020

# Una ontología para la representación de conceptos de diseño de software

Este trabajo apunta a Crear una herramienta Poderosa para los desarrolladores de software, proporcionando una base Sólida para la toma de decisiones de diseño y la creación de sistemas de alta calidad, lo que felicitara la colaboración entre equipos de desarrollo. estandarización Para solo utilizar un lenguaje común Para hablar del diseño del software, utilizando la ontología Permitiendo una estructura formal que relacione todos estos conceptos, permitiendo búsquedas y consultas Precisas.



## Reflexión

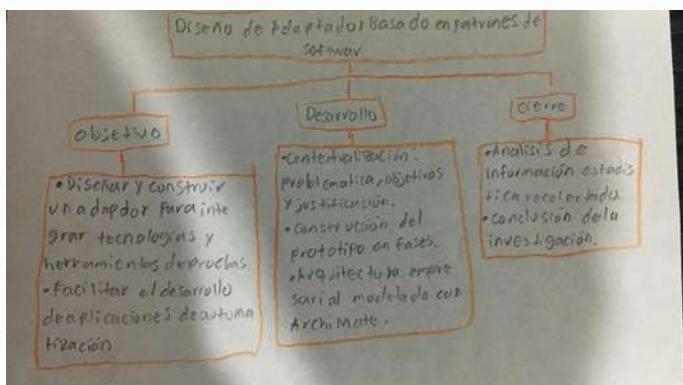
La propuesta de crear una ontología para el desarrollo. Al unificar los conceptos y las mejores prácticas en un solo lugar, se facilita la comunicación entre desarrolladores, se promueve la reutilización de código se asegura una mayor calidad en los proyectos. Esta iniciativa no solo beneficiara a los programadores experimentados, Sino que también sirve como una herramienta de Software para principiantes

## Bibliografía

Autor: Gloria Giraldo, Juan Acevedo, David Moreno  
Año:2011

# Paquete java para la integración de herramientas de pruebas de software basado en patrones de software

Este artículo de investigación explora varios, Patrones de diseño que ayudan a estructurar y organizar mejor el desarrollo de software. Los patrones Creacionales facilitan la creación controlada de objetos, como el singleton, que asegura una única instancia, o el factory, que permite crear objetos de una familia de clases. Los Patrones estructurales como el adapter y el facade, optimizan las relaciones entre clases para crear sistemas más flexibles y extensibles.



## Reflexión

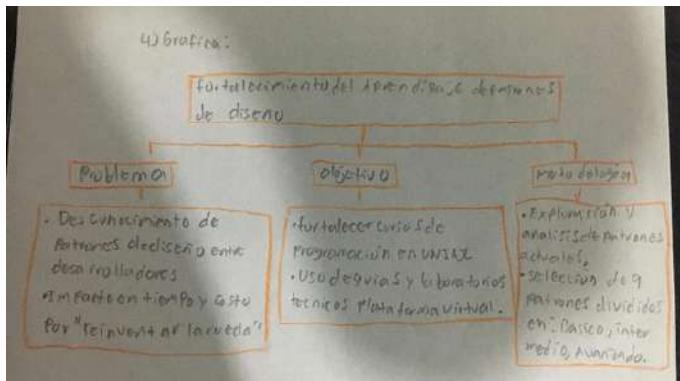
en este artículo se destaca la importancia de los patrones de diseño para optimizar el desarrollo de software, a bordando la creación, Estructuración y comportamiento de objetos en un sistema. Los patrones Creacionales, como el singleton o el factory, ofrecen mecanismos para manejar la creación de instancias de manera controlada, lo cual resulta útil en contextos donde la creación directa puede generar problemas de rendimiento o estructura

## Bibliografía

Autor: Yhon Edinson Estevez Mendoza.  
Año: 2019

# Entorno virtual para la formación de tecnologías e ingenieros de sistemas en patrones de diseño de software

Este artículo está dirigido a estudiantes de ingeniería en sistemas de la UNIAJC que desean profundizar en patrones de diseño de software. A lo largo del texto, se presentan nueve patrones esenciales del reconocido libro "Gang of Four", organizados en tres niveles: básico, intermedio y avanzado. Cada nivel se enfoca en distintos tipos de patrones de diseño: los Creacionales, como el 'Factory Method', los estructurales, como 'Composite', y las de comportamiento como 'composite' y los de comportamiento, como 'observer'.



## Reflexión

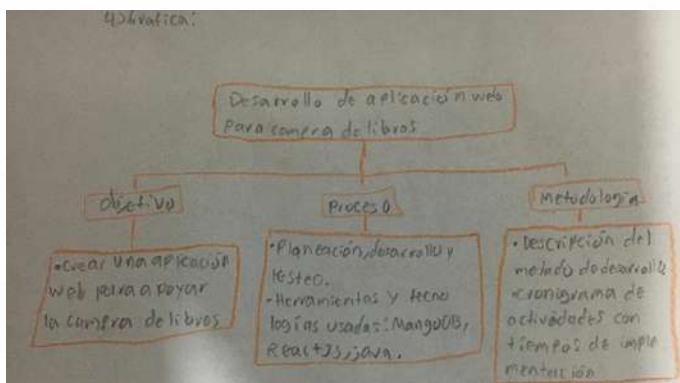
Este artículo es una guía útil para estudiantes de ingeniería en sistemas que desean aprender Patrones de diseño. Estos Patrones ayudan a resolver problemas comunes en el desarrollo de software, mejorando la calidad y facilitando el trabajo en equipo. Usar JavaScript en los ejemplos hace que los conceptos sean aplicables en proyectos actuales, sobre todo en el desarrollo web. La inclusión de moodle como plataforma permite un aprendizaje flexible y autónomo. Este enfoque práctico, basado en resolver problemas reales, prepara a los estudiantes para enfrentar desafíos en la industria y fomenta la innovación en tecnología.

## Bibliografía

Autor: Ramiro Andres Escobar y Tania Isadora Mora Pedreros  
Año: 2021

# Aplicación web para recomendación de libros para apoyar la elección de compra

Este artículo aborda temas claves en el desarrollo de software, como las API REST, que facilitan la comunicación entre sistemas mediante HTTP, y los principios SOLID- que ayudan a estructurar código más mantenible Se discuten también en patrones de diseño, útiles para resolver problemas comunes, y loases de datos no relaciones Eficaces para manejar grandes volúmenes de datos. Herramientas como git yjira mejoran la gestión de versiones y tareas en equipo, mientras que Node.js facilita la creación de aplicaciones en tiempo real.



## Reflexión

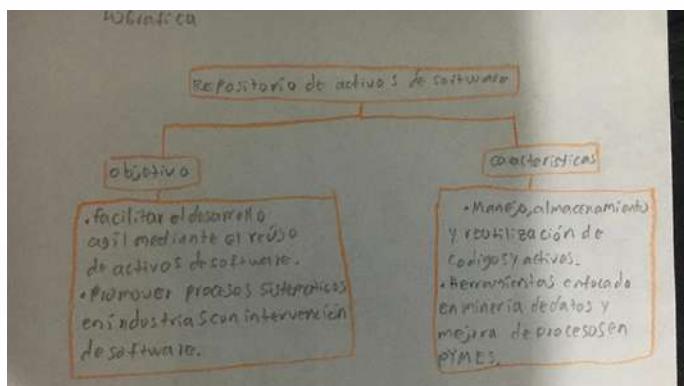
El articulo destaca la importancia de aplicar buena S prácticas y herramientas en el desarrollo de Software. Las API REST y los principios SOLID Permiten Crear sistemas escalables y fáciles de mantener, mientras que los Patrones de diseño ayudan resolver problemas Comunes de forma efectiva. Las bases de datos no relacionadas ofrecen flexibilidad para manejar grandes volúmenes de datos de manera ágil. Además, herramientas como git yjira facilitan la organización y colaboración en equipo.

## Bibliografía

Autor: Juan Pablo. Gomez Quintero  
Año: 2021

# Buenas prácticas en la construcción del software

Este artículo de reflexión explora diferentes arquitecturas de software y metodologías de desarrollo utilizadas para crear soluciones eficientes y flexibles en sistemas de ti. Primero, detalla la arquitectura de solución, que guía el diseño de estructuras integradas para responder a necesidades actuales y futuras, y la arquitectura de software, que tiene patrones y tecnologías específicas. Luego, describe estilos arquitectónicos como capas, monolítico, microservicio S, EDA (Arquitectura Orientada a eventos) y cliente-servidor, que organizan la comunicación y estructura de aplicaciones.



## Reflexión

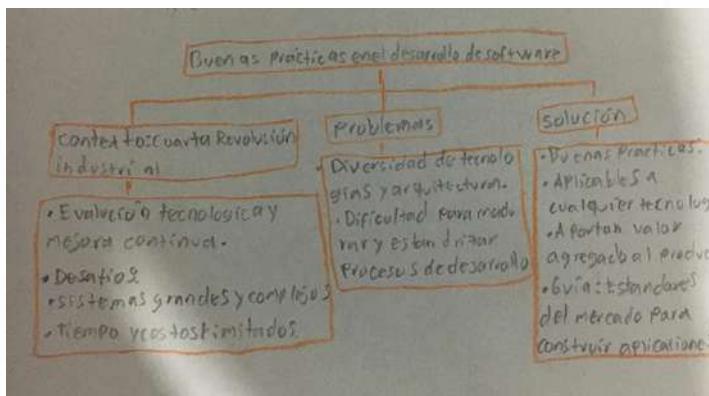
Este artículo destaca la importancia de elegir bien la arquitectura y la metodología de desarrollo para crear software que sea adaptable, modular y eficiente. Cada estilo arquitectónico - como capas, monolítico o microservicios - ofrece ventajas según las necesidades de la aplicación, ya sea flexibilidad, independencia o escalabilidad. La metodología XP, por su parte, fomenta la comunicación constante entre desarrolladores y clientes, permitiendo cambios rápidos y manteniendo el equipo enfocado en el cliente.

## Bibliografía

Autor: Camilo Andrés prieto y Diego Alejandro Madrid  
Año:2022

# **Construcción de un repositorio de activos de software para el desarrollo ágil de aplicaciones un método para el reusó**

Este artículo aborda la problemática de la reutilización de software en la industria del desarrollo, enfocándose en como los repositorios de activos pueden mejorar este proceso. A lo largo del tiempo, metodologías como la programación estructurada y la programación orientada a objetos sean desarrollado para mejorar la eficiencia. La investigación incluye un repositorio llamado "Activos" que almacena artefactos reutilizables. Como código y esquemas de base de datos, y permite el acceso a estos recursos para proyectos futuros.



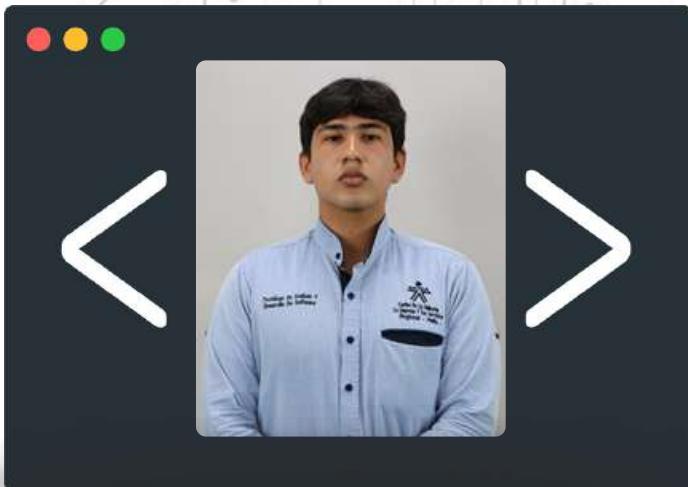
## **Reflexión**

El reusó de activos de software es clave Para ahorrar tiempo y recursos en el desarrollo. Al reutilizar Patrones y componentes, los equipos logran aplicaciones más consistentes y Eficaces, además de fomentar la colaboración. Los repositorios de software organizados facilitan este proceso, evitando empezar desde cero y permitiendo construir sobre el trabajo de otros. Esta práctica no solo mejora la productividad, sino que impulsa una cultura más innovadora y Colaborativa en la industria del software.

## **Bibliografía**

Autor: yeimar Alonso Castro, Javier Dario Fernandez, julian alberto Rivera, Eder Acevedo Marin.  
Año:2017

# **Yordy Erik Nuñez Pineda**



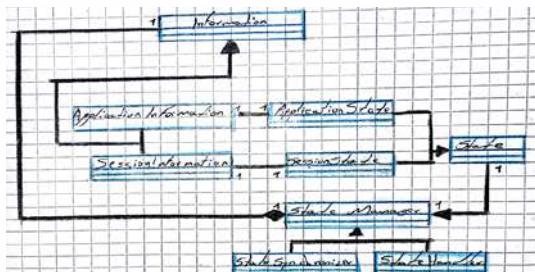
Soy Yordy Erik Nuñez Pineda, un apasionado por el mundo de la tecnología, el deporte y la música, con experiencia en diversas herramientas y lenguajes de programación como Java, Spring, JavaScript, MySQL, HTML y CSS. Manejo GitHub de forma eficiente y disfruto involucrarme en la solución de problemas, aplicando mis conocimientos para crear soluciones efectivas. Estoy firmemente en el aprendizaje continuo, lo que me motiva a mejorar constantemente y afrontar nuevos desafíos. Además, me gusta trabajar en equipo, ya que considero que la colaboración es clave para alcanzar metas compartidas e impulsar la innovación en el desarrollo tecnológico.

# Modelado y verificación de patrones de diseño de arquitectura de software para entornos de computación en la nube.

El artículo examina el diseño y la validación de patrones de arquitectura del software en el contexto de la computación en la nube. En este se presenta una herramienta fundamentada en un metamodelo de un componente que ayuda a los arquitectos en la creación y evaluación de diseños arquitectónicos lo cual facilita la representación gráfica y garantiza la correcta aplicación de los patrones de diseño, lo cual ayuda a mejorar la calidad de software. El metamodelo propuesto abarca módulos que definen y validan los patrones, así ayudando a los arquitectos en la documentación. También el artículo resalta el desafío de los arquitectos, debido a la ausencia de patrones estandarizados para entornos en la nube y menciona la relevancia de contar con una estructura bien definida.

## Reflexión

El artículo es de gran relevancia en el contexto actual, donde la computación en la nube desempeña un papel importante en la arquitectura de aplicaciones. El desarrollo de un entorno de diseño que facilita la verificación de patrones en aplicaciones basadas en la nube es un avance significativo, dado a la complejidad de estos entornos. Mediante un metamodelo que posibilita la instantación y verificación de componentes, el artículo nos invita a considerar la importancia de la automatización en el diseño arquitectónico. Esto no sólo minimiza errores, sino que también ayuda a la adopción de prácticas estandarizadas, lo que mejora la eficiencia y la calidad del software.



## Bibliografía

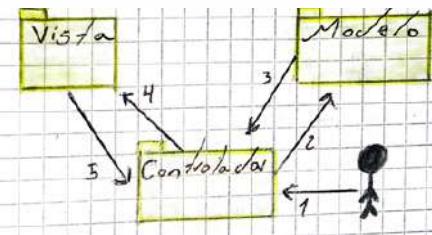
- Blas, M. J., Leone, H. P., & Gonnet, S. M. (2019). Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube. <https://n9.cl/9e1no1>

# Evaluación de una arquitectura de software

El artículo nos muestra la evaluación de una arquitectura de software destinada a la secretaría de salud de Villavicencio (SMSV) centrada en el cumplimiento de la resolución 4505 de Colombia, que nos habla del reglamento del registro y la presentación de información de salud pública. Se diseñó un sistema de software utilizando el marco de Modelo - Vista -Controlador (MVC) y se analizaron diversas arquitecturas a través del método SAAM, en esta se dio prioridad a la modificabilidad debido a los cambios de la normativa. La metodología ágil Scrum, con un sprint 0, facilita la evaluación anticipada de la arquitectura, lo cual permitió la adaptabilidad del sistema. Lo cual mejoró la comunicación, ayudó a la identificación de posibles modificaciones y optimizar la implementación de la arquitectura que eligieron.

## Bibliografía

Sanabria, F. R., & Rodríguez, S. V. (2021). Evaluación de una Arquitectura de Software. *Prospectiva*, 19(2). <https://n9.cl/p47y4>

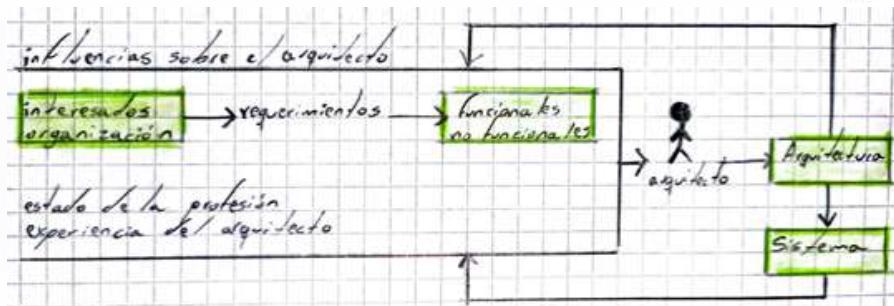


## Reflexión

En el artículo elegir una arquitectura bien fundamentada y flexible como lo es el MVC resultó importante en proyectos de gestión pública, más cuando éstos cambian constantemente, al seleccionar la arquitectura MVC ofrece una clara división de responsabilidades, sino, que también permitirá la realización de actualizaciones futuras sin poner en riesgo la estabilidad del sistema. También la evaluación temprana en un entorno no sólo mejora la eficiencia al desarrollar, sino que también asegura que el software se mantenga eficaz conforme avanza el tiempo, mostrándonos la relevancia de prácticas de arquitectura adaptativa.

# Una Teoría para el Diseño de Software

El artículo aborda los principios del diseño de software, destacando la importancia de organizar los sistemas para facilitar modificaciones y reducir costos de mantenimiento. Describe el diseño como una etapa crucial donde el software se divide en componentes con funciones y conexiones definidas. Se presentan tres niveles de estructura: estilos de arquitectura, diseño de patrones y componentes, explicando cómo cada uno contribuye al desarrollo del sistema. También se discute la evolución del diseño, desde los patrones arquitectónicos hasta el ciclo de vida arquitectónico, y se diferencia arquitectura y diseño.



## Reflexión

El artículo resalta la importancia del diseño de software con una inversión que mejora la capacidad para la adaptación y durabilidad del sistema. Adaptar un diseño orientado al cambio brinda a los desarrolladores para prever y administrar modificaciones de manera eficiente, asegurando la integridad del sistema. Esto resulta importante dentro de un entorno donde las exigencias a cuanto el software está en constante cambio y cualquier modificación todavía implica gastos adicionales. La distinción entre los niveles estructurales y la importancia de los requisitos facilitan la comunicación. Ya que un diseño bien estructurado se convierte en una herramienta que no sólo soluciona problemas presentes, sino que alista el software a futuras necesidades.

## Bibliografía

- Cristiá, M. (2021). Una Teoría para el Diseño de Software. <https://n9.cl/dr0m7>

# Arquitectura de Software con Programación Orientada a Objetos

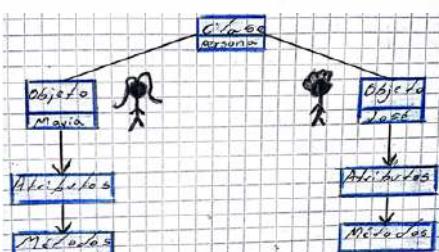
El artículo nos habla de la arquitectura de software dentro del ámbito de la programación orientada a objetos (POO) donde resalta la simplicidad de la creación y actualización de sistemas complicados. La arquitectura de software consiste en estructurar los componentes y sus interacciones para satisfacer los requerimientos técnicos. La programación orientada a objetos posibilita la agrupación de datos, lo que facilita la reutilización y estructuración del código. La arquitectura orientada a objetos no permite desarrollar componentes independientes y reutilizables, lo que mejora la escalabilidad y flexibilidad.

## Reflexión

El artículo destaca cómo la arquitectura de software incorporando programación orientada a objetos ha transformado la creación de sistema, ofreciendo una estructura que favorece a la flexibilidad y durabilidad del software. Al estructurarse en partes separadas, la programación orientada a objetos nos facilita la gestión de sistemas complicados tanto en su diseño como en el mantenimiento. Esto ayuda a la eficiencia del desarrollo, sino que también fomenta a una programación más intuitiva y centrada a la resolución de problemas reales. La programación orientada a objetos con sus principios facilita la creación de sistemas escalables que pueden ajustarse al cambio.

## Bibliografía

Vera, J. B. V. (2023). Arquitectura de software con programación orientada a objetos. Polo del Conocimiento: Revista científico-profesional, 8(12), 1497-1508. <https://n9.cl/3qemz>

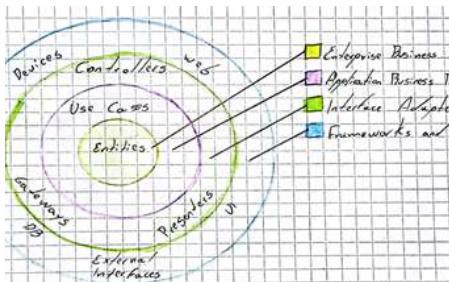


# Implementación de una Arquitectura de Software Guiada por el Dominio.

El artículo presenta la implementación de una arquitectura de software centrada en el dominio mediante el diseño guiado por el dominio (DDD) y la arquitectura hexagonal. Este enfoque se utiliza para desarrollar sistemas complejos donde la lógica empresarial es clave. DDD propone un modelo de dominio que facilita la comunicación entre expertos, permitiendo la evolución independiente de las distintas partes del sistema. El estudio muestra cómo se transformó una arquitectura de 3 capas a una hexagonal en una plataforma para la contratación y evaluación de servicios, lo que facilita la independencia y el mantenimiento del núcleo empresarial.

## 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://n9.cl/tdp9p>



## Reflexión

El artículo resalta la relevancia de enfocar el diseño de software en la comprensión del negocio lo que da la capacidad de evolución y adaptación del sistema ante cambios tecnológicos. La arquitectura hexagonal optimiza la modularidad y escalabilidad del sistema al separar la lógica de negocio de las interfaces y las herramientas externas. La flexibilidad dada por DDD y la arquitectura hexagonal, permite que los desarrolladores se enfoquen en la funcionalidad principal. Este enfoque resalta la importancia de una estructura de software claramente definida para abordar desafíos complejos.

# Especificación de la Arquitectura de Software

El artículo nos muestra los componentes y diseños arquitectónicos que conforman una arquitectura de software detallando la importancia en la creación y permanencia de sistemas tecnológicos modernos. Resalta la contribución de los patrones arquitectónicos en la organización del software para facilitar la comunicación seguridad y flexibilidad en entornos móviles y web. Hace mención a los patrones POSA y PEAA los cuales son importantes en esta situación. Al proponer estructuras arquitectónicas en grupos como sistemas distribuidos e interactivos. Así mismo nos presenta conceptos como patrones complementarios y secuenciales, los cuales mejoran y combinan las soluciones arquitectónicas de forma coherente.

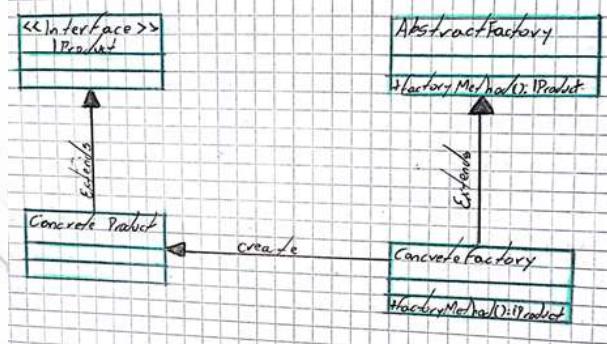
## Reflexión

El artículo nos muestra la importancia de los patrones arquitectónicos como herramientas claves en el diseño de software, se plantea una arquitectura modular y adaptable que simplifica la creación y actualización de aplicaciones con capacidad de escalabilidad. Los patrones hablan sobre desafíos de diseño como lo que facilita la comunicación entre desarrolladores, además permite reutilizar estructuras probadas. También al combinar patrones complementarios o compuestos, aborda las necesidades de software, mejorando la eficiencia del diseño y la capacidad de adaptarse. Lo cual, al emplear patrones arquitectónicos bien documentados es de gran ayuda para desarrollar un sistema confiable y flexible.

## Bibliografía

- Guarin, J. S. V. (2023). Especificando una arquitectura de software: Software architecture specification. *Tecnología Investigación y Academia*, 11(2), 170-181. <https://n9.cl/d1v0w>

# Introducción a los Patrones de Diseño



## Reflexión

El artículo resalta la importancia que conlleva el empleo de patrones de diseño, ya que nos muestra lo efectivo que es para abordar los desafíos habituales en la creación de software. El seleccionar y utilizar un patrón apropiado según cada situación, nos permite desarrollar sistemas flexibles lo cual es importante ya que es la calidad del código y la escalabilidad son vitales. Igualmente nos invita a cómo los patrones no limitan la creatividad, sino que nos permite desarrollar con mayor seguridad. Esto nos permite tener una mentalidad de programación enfocada a la resolución de problemas.

## Bibliografía

Blancarte, O. (2016). Introducción a los Patrones de Diseño. México, México DF. <https://n9.cl/btd5y>

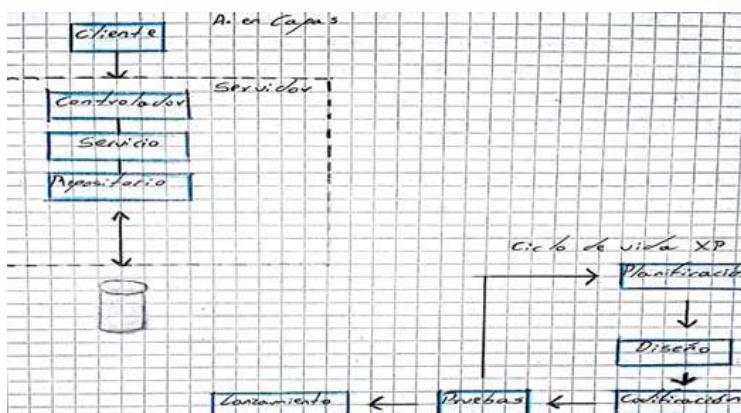
El artículo nos muestra la introducción a los patrones de diseño, resaltando su procedencia, importancia y clasificación en creacionales, estructurales y de comportamiento. Muestra la forma en que estos patrones abordan problemas típicos de diseño con soluciones probadas, reutilizando código y haciendo una gestión eficaz de los sistemas. Cada patrón se describe detalladamente con ejemplos de su aplicación en situaciones reales, empleando UML y programación orientada a objetos (POO). El artículo nos brinda una guía práctica, destacando la importancia del uso correcto de patrones para mejorar la estructura, flexibilidad y durabilidad del software.

# Buenas Prácticas en la Construcción de Software.

El artículo nos habla de las prácticas más efectivas para desarrollar software debido al rápido avance tecnológico debido a la cuarta Revolución Industrial. La aplicación de estas prácticas ayuda a afrontar el reto de elaborar sistemas complejos con rapidez y de manera eficiente. El artículo también muestra la importancia de principios como "Clean Code" junto a metodologías ágiles como XP "Extreme Programming" que facilita la rápida adaptación a los cambios. También se exploran conceptos de arquitectura y estilos arquitectónicos como la arquitectura en capas y los microservicios. Por último, hablando de la integración de estándares de codificación y el uso de herramientas DevOps para mejorar el desarrollo.

## Reflexión

Seguir buenas prácticas y estándares en el desarrollo del software no sólo influye en la calidad de nuestro código, sino que también nos ayuda para la sostenibilidad y eficiencia del proyecto. Por eso resulta esencial que los desarrolladores sigan principios como Clean Code y apliquen metodologías ágiles. Emplear herramientas automatizadas nos ayudará a los plazos de entrega. Tener todos estos principios y estas prácticas nos garantizará la capacidad de adaptabilidad de nuestro software ante los constantes cambios tecnológicos.

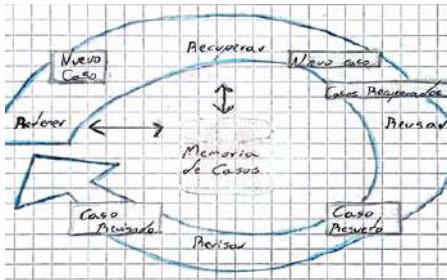


## Bibliografía

- Prieto, C. A., & Madrid, D. A. (2022). Buenas prácticas en la construcción de software: Best practices in software construction. *Tecnología Investigación y Academia*, 10(2), 149-166. <https://n9.cl/hnmdef>

# Representación y Razonamiento sobre las Decisiones de Diseño de Arquitectura de Software.

Este artículo es un modelo de ayuda para los arquitectos de software, éste emplea el Razonamiento Basado en Casos (RBC) para la planeación de diseños arquitectónicos. Aquí nos presentan una herramienta que favorece la utilización de experiencias basadas en el ámbito de la arquitectura, lo cual nos brinda acceso a soluciones las cuales ya han sido probadas y registradas. En este se nos detalla un proceso de cuatro etapas las cuales son (recuperación, reutilización, revisión, retención) denominada las "4Res" el cual facilita la organización del conocimiento arquitectónico con el propósito de ser reutilizado.



## Reflexión

En el artículo nos muestra la importancia para capturar, organizar y reutilizar conocimientos en el diseño arquitectónico del software. Al usar el razonamiento basado en casos no permite tomar decisiones efectivas más en situaciones complejas las cuales necesitan mayor experiencia para mejorar los resultados. También disminuye el tiempo y esfuerzo que se emplean en la etapa de diseño. Igualmente nos muestra un enfoque cuidadoso y la importancia de aplicar métodos que conviertan la creación de software en un procedimiento efectivo, lo cual puede reducir fallos.

## Bibliografía

Carignano, M. C. (2016, June). Representación y razonamiento sobre las decisiones de diseño de arquitectura de software. In XVIII Workshop de Investigadores en Ciencias de la Computación (WICC 2016, Entre Ríos, Argentina). <https://n9.cl/ug8br>

# **El Papel de las Tecnologías del Acuerdo en la Definición de Arquitecturas de Software Adaptativas**

El artículo investiga de qué manera las tecnologías del acuerdo y los sistemas multi - agente (SMA) pueden favorecer a la elaboración de arquitecturas de software adaptables. Conforme los sistemas de software se vuelven más complicados, es importante que sean capaces de adaptarse de forma automática. La atención se centra en un marco centrado en servicios, donde los agentes colaboran y se autogestionan según los protocolos y controles específicos. La plataforma THOMAS facilita la coordinación de agentes, como se evidencia en el caso de SUMMA 112, donde se logra reorganizar recursos entre distintas organizaciones para abordar diversas situaciones. Igualmente, los conceptos de protocolos, confianza y negociación son importantes en la arquitectura.



## **Bibliografía**

Pérez, J. S., Cuesta, C. E., & Ossowski, S. (2010). El papel de las tecnologías del acuerdo en la definición de arquitecturas de software adaptativas. In Simposio Argentino de Ingeniería de Software (ASSE 2010)-JAIIO 39 (UADE, 30 de agosto al 3 de septiembre de 2010). <https://n9.cl/kpvql3>

## **Reflexión**

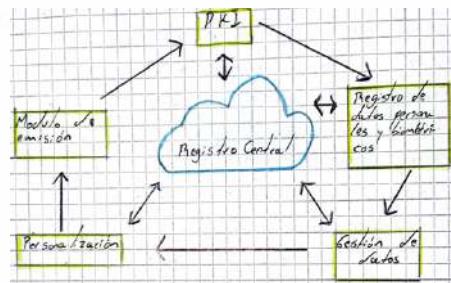
En este artículo se muestra un proceso importante en la creación de arquitecturas de software adaptativo, empleando tecnologías del acuerdo en un entorno multi - agente para facilitar la autoorganización y adaptabilidad de los sistemas. En este caso la plataforma THOMAS ilustra cómo estas tecnologías pueden ser aplicadas en escenarios de la vida real, demostrando así el potencial de la adaptabilidad automática en la gestión de recursos. El desarrollo de esas ideas podría conducir a innovadoras maneras de colaborar y atender eficazmente en entornos complejos.

# Modelo de Gestión de Servicios PKI

El artículo detalla la evolución de un modelo de gestión de servicios PKI (Infraestructura de Clave Pública) fundamentado en la arquitectura orientada a servicios (SOA), que se ha llevado a cabo en el Banco de la República de Colombia. Se fusionaron saberes de PKI, SOA, junto a tecnologías como web Services y J2EE. La arquitectura propuesta posibilita brindar servicios de seguridad, como la firma digital. Mediante una estructura distribuida con clientes y proveedores de servicios que trabajan en diferentes plataformas. Esto ofreció un marco flexible y escalable.

## Bibliografía

Valbuena, D. C. Modelo de gestión de servicios PKI.  
<https://n9.cl/55lwus>

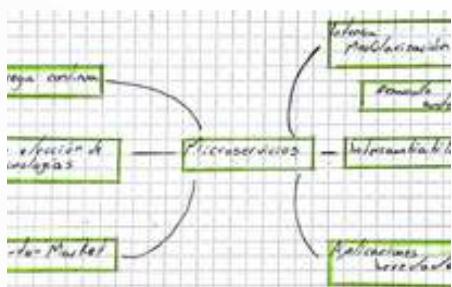


## Reflexión

La implementación de un modelo de gestión PKI con una arquitectura enfocada a servicios resalta la relevancia de elegir soluciones flexibles y seguras en un entorno donde la seguridad de los datos es importante. La integración de tecnologías como J2EE y web Services hoy nos ayuda a garantizar la flexibilidad del modelo, permitiendo su integración en diferentes sistemas y cumpliendo con todos los requisitos. Este modelo no sólo optimiza la gestión de servicios PKI, sino también simplifica la automatización, lo que nos ayuda a reducir riesgos.

# Arquitectura de Software Basada en Microservicios para Desarrollo de Aplicaciones Web

En este artículo se expone una arquitectura fundamentada en microservicios, destinada a la creación de aplicaciones web. La investigación describe las restricciones de aplicaciones monolíticas, como la complejidad en su mantenimiento, escalabilidad y tiempos de despliegue, lo que conduce a la importancia de contar con una arquitectura modular y distribuida. La arquitectura de microservicios consiste en dividir la aplicación en pequeños servicios autónomos que se interconectan mediante APIs, brindando beneficios como despliegue independiente, escalabilidad y la tolerancia a fallos.



## Bibliografía

- López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web. <https://n9.cl/0tbddk>

## Reflexión

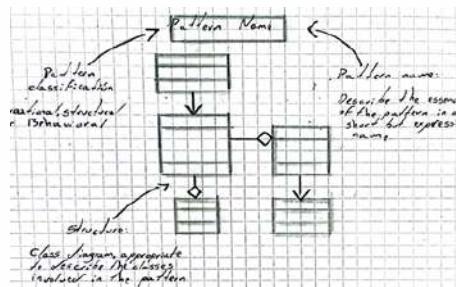
La evolución desde una arquitectura monolítica hacia una de microservicios supone un gran avance en la capacidad de adaptación y rendimiento de las aplicaciones web. El artículo nos invita a ver cómo los microservicios no sólo contribuyen a mejorar el mantenimiento, sino que también fomenta una cultura para el desarrollo más ágil y adaptable. Al separar la aplicación en componentes independientes cada uno con su propio funcionamiento y base de datos, lo cual disminuye la posibilidad de errores y se simplifica la colaboración en grupo, permitiendo que distintos equipos se enfoquen en servicios particulares sin afectar el conjunto de la aplicación.

# COFFEE CHALLENGE: UN JUEGO PARA EL APRENDIZAJE DE PATRONES DE DISEÑO DE SOFTWARE

El artículo nos presentó un juego educativo diseñado para enseñar patrones de diseño a estudiantes de ingeniería de software. El juego utiliza el aprendizaje basado en juegos para mejorar la comprensión de patrones como los por la "Gang of Four" (GoF) Y patrones de interfaz de usuario. Coffee Challenge simula el diseño de una página web de venta de café en la que los jugadores eligen y aplican los patrones de diseño adecuados para cumplir los requisitos del proyecto. Se hicieron pruebas evaluando su efectividad mediante encuestas. Los resultados indicaron que El juego facilitó el aprendizaje práctico y motivó a los estudiantes a participar más.

## Bibliografía

González-Castaño, L. E., Marroquín-Soto, S. V., Maturana-González, G. V., & Manjarrés-Betancur, R. A. (2021). Coffee Challenge: Un juego para la enseñanza de patrones de diseño de software. Revista Politécnica, 17(33), 34-46. <https://n9.cl/fm1ii>

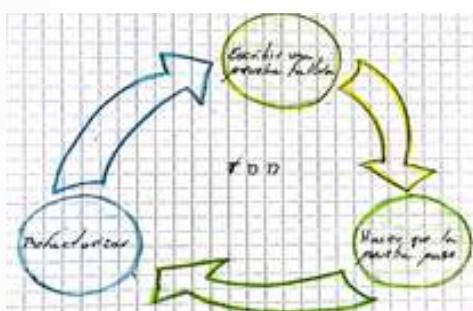


## Reflexión

El implementaron juegos interactivos como Coffee Challenge, representa una innovación importante en la enseñanza de conceptos como los patrones de diseño de software. Eso permite a todos aprender de una forma interactiva, aplicando conocimiento en situaciones simuladas que reflejan problemas reales. La estructura del juego, que utiliza un diseño de una página de comercio, lo cuál ayuda al aprendizaje, haciendo que todos entiendan no sólo cómo aplicar los patrones, sino también cuándo y por qué usarlos. Haciendo que este tipo de herramientas potencie el aprendizaje y resolución de problemas.

# Calidad Ágil: Patrones de Diseño en un Contexto de Desarrollo Dirigido por Pruebas

El artículo explora la aplicación de patrones de diseño en el desarrollo dirigido por pruebas (TDD) dentro de las metodologías ágiles. Este resalta la importancia de asegurar la testeabilidad de los patrones, haciendo pruebas unitarias, usando JUnit para evaluar patrones como observador, factoría abstracta y MVC. Estos patrones deben cumplir ciertas condiciones, como encapsulamiento y ausencia de dependencias ocultas, para ser efectivos en TDD. Se hizo una investigación del sistema automático de control de velocidad de un vehículo (SCACV). Los resultados indican que los patrones mejoran la calidad, pero algunos presentan limitaciones en términos de complejidad y mantenimiento.



## Bibliografía

Capel, M. I., Grimán, A. C., & Garví, E. Calidad Ágil: Patrones de Diseño en un contexto de Desarrollo Dirigido por Pruebas. : <https://n9.cl/6wk3a>

## Reflexión

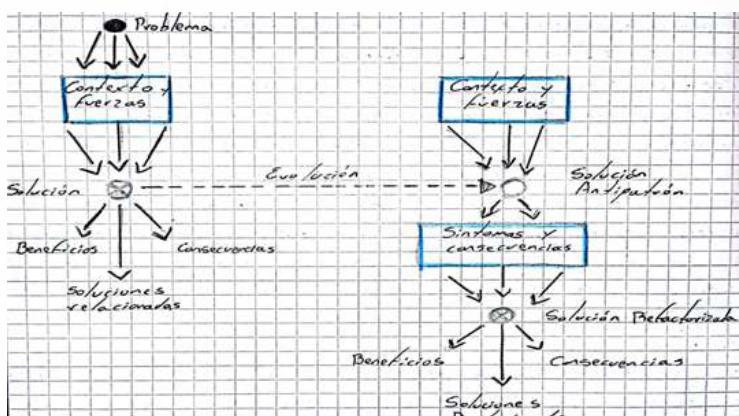
El artículo nos muestra los beneficios y desafíos de combinar patrones de diseño con TDD en el contexto ágil. La integración de patrones de diseño probado nos ayuda a estructurar el código de forma modular, facilitando su mantenimiento y reutilización. Igualmente, nos muestra que no todos los patrones se adaptan fácilmente al TDD, cheque algunos aumentan la complejidad del sistema y dificulta la creación de las pruebas unitarias, especialmente en las interfaces del usuario. Mostrándonos la importancia de seleccionar patrones de diseño con criterios de testabilidad en mente.

# Utilización de Antipatrones y Patrones en el Análisis de Software

El artículo explora el papel de los antipatrones y patrones en el desarrollo de software. Los antipatrones se definen como soluciones recurrentes, que no resuelven problemas, sino, que generan consecuencias negativas, mientras que los patrones representan buenas prácticas que se pueden aplicar en diferentes etapas del desarrollo. Los autores no muestran la importancia de identificar y evitar antipatrones para mejorar la calidad de software, al tiempo que nos sugieren que los patrones pueden acelerar la definición de modelos conceptuales y facilitar la validación de requerimientos.

## Reflexión

Reconocer los antipatrones no sólo permite a los desarrolladores evitar errores comunes, sino que también proporciona una valiosa lección sobre la importancia del conocimiento. Al mismo tiempo, la adopción de patrones como guías para la solución de problemas fomenta a la innovación y a la mejora continua en el proceso de desarrollo de software. Lo cual nos invita a reflexionar sobre nuestras decisiones y enfoques, lo cual promueve el aprendizaje y adaptación. Así que los antipatrones y los patrones son herramientas que conducen a la creación de software de alta calidad.



## Bibliografía

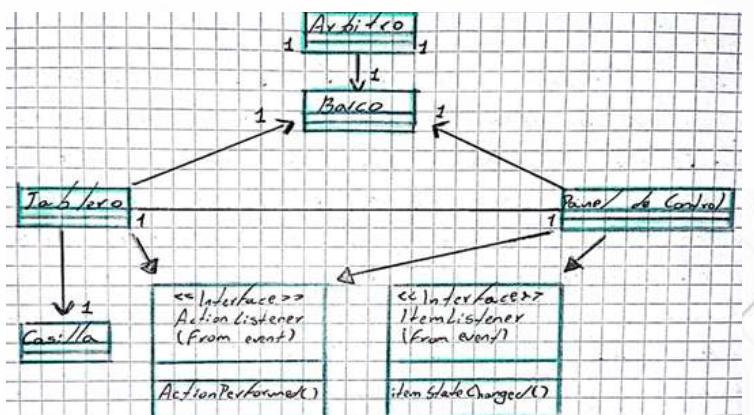
Garnero, A. B., & Horenstein, N. Utilización de antipatrones y patrones en el análisis de software. <https://n9.cl/ae8jq6>

# Aprendizaje Práctico de Patrones de Diseño en Asignaturas de Programación de Nivel III

El artículo nos presenta los resultados de una evaluación en la cual se aplicaron patrones de diseño a un curso de programación. A través de las encuestas que se realizaron, se pudo notar que estos no sólo comprenden la teoría, sino que también aprecian su utilidad práctica. Los resultados indican un alto porcentaje de estudiantes considera que aplicarán patrones a sus proyectos. Además, nos muestra los beneficios de utilizar patrones, la reducción de tiempo de desarrollo, mejora en la documentación de software y en su calidad.

## Reflexión

El artículo nos muestra la importancia de usar conceptos teóricos con práctica en la educación, más en áreas complejas como lo son la programación. El aplicar patrones de diseño demuestra que no sólo mejora la comprensión, sino que también nos prepara para enfrentar desafíos reales en el desarrollo de software. Nos demuestra también que la comunicación es crucial para un mayor aprendizaje. Más ahora donde la tecnología avanza rápidamente es importante adoptar métodos y herramientas de enseñanza donde sea notable la mejora continua del desarrollo de software.



## Bibliografía

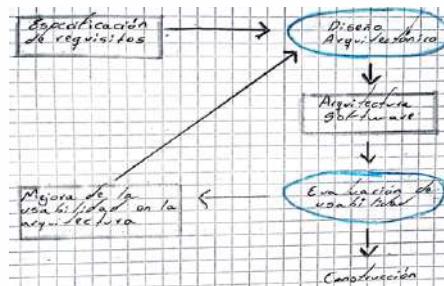
Marticorena, R., López, C., García Osorio, C. I., & Pardo, C. (2002). Aprendizaje práctico de patrones de diseño en asignaturas de programación de nivel III. <https://n9.cl/4cusk>

# Patrones de Usabilidad Temprana en el Modelo Conceptual

El artículo nos muestra la importancia de la usabilidad en el desarrollo de aplicaciones web como el artículo nos resalta como muchos sistemas fallan debido a deficiencias. Se nos menciona que la usabilidad es la que determina la satisfacción del usuario al interactuar con un sistema, por eso debe ser considerada desde las etapas iniciales del ciclo de vida del software. Nos proponen patrones de usabilidad que pueden ser incorporados en el modelo desde la fase de requisitos. Igualmente nos presentan patrones específicos que abordan distintos criterios de usabilidad, como la prevención de errores de entrada, etc.

## Bibliografía

Moreno, J. C., Marciszack, M. M., & Groppe, M. A. (2020). Patrones de Usabilidad Temprana en el Modelo Conceptual. AJEA, (5). <https://n9.cl/ewi5om>



## Reflexión

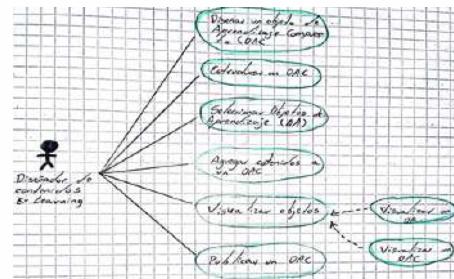
Incorporar patrones de usabilidad desde las etapas iniciales de nuestro desarrollo del software es algo valioso ya que ayuda a cómo se diseña y construye en aplicaciones. El establecer un patrón que guíe a los desarrolladores a la creación de interfaces más intuitivas y accesibles, asegura un desarrollo más centrado al usuario. Además, es importante que nosotros los desarrolladores estemos equipados e informados de las herramientas para que podamos implementar soluciones efectivas.

# Una Arquitectura de Software para la Integración de Objetos de Aprendizaje Basados en Servicios Web

El artículo nos presenta un modelo arquitectónico diseñado en el cual se facilita la integración de Learning Management Systems (LMS) y Repositories of Learning Objects (ROA). La arquitectura se compone de 5 vistas: funcional, estructural, de comportamiento, de implementación y de despliegue, cada una de estas aborda diferentes aspectos del sistema. La arquitectura presentada busca resolver problemas de interoperabilidad y la reutilización de objetos, para ofrecer una solución más práctica para el desarrollo.

## 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://n9.cl/4lhz>

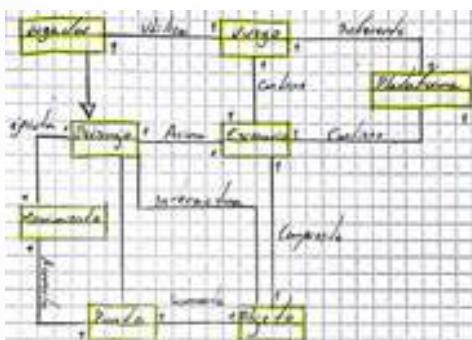


## Reflexión

Este nos resalta que una buena arquitectura bien definida en el ámbito de la educación, donde la integración de recursos de aprendizaje es crucial. Hoy la propuesta de que un modelo basado en servicios web no sólo aborda la necesidad de la interoperabilidad entre los diferentes sistemas, sino que también nos promueve a la reutilización de objetos de aprendizaje como también al ofrecernos un marco estructurado que facilita la búsqueda y el diseño de contenidos, nos permite crear materiales más efectivos y accesibles.

# Arquitectura de Software para el Desarrollo de Videojuegos sobre el Motor de Juegos Unity 3D

El artículo nos presenta una arquitectura de software diseñada específicamente para la creación de videojuegos utilizando Unity 3D. Esta arquitectura se organiza en capas como incluye un Game Manager, cada uno con funciones específicas, también incluye un State Machine, Sound Manager, Data Manager y Scene Manager. Se resalta la importancia de patrones de diseño como el singleton para asegurar la existencia de una única instancia de ciertos componentes. También discuten algunas restricciones arquitectónicas para garantizar la multiplataforma y la facilidad de actualización de los componentes.



## Bibliografía

- Paez, A. H., Falcón, J. D., & Cruz, A. A. P. (2018). Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D. Revista de I+D tecnológico, 14(1), 54-64 <https://n9.cl/dbt4a>

## Reflexión

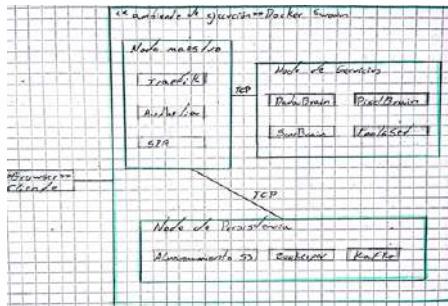
El artículo nos muestra la importancia de una arquitectura de software bien estructurada en el desarrollo de videojuegos, un campo el cual es bastante creativo. La arquitectura propuesta permite a los desarrolladores gestionar de manera más efectiva los diversos aspectos del juego desde la lógica hasta la interacción con el usuario. Al implementar patrones de diseño, hay creación de un código más limpio y mantenable, lo cual es crucial en este entorno donde los cambios son constantes, permitiendo a los desarrolladores innovar.

# Propuesta de Diseño de Arquitecturas Software para la Gestión, Análisis y Procesamiento de datos de Neurociencia

El artículo nos presenta un modelo arquitectónico destinado a abordar los desafíos de interoperabilidad, mantenimiento, escalabilidad y funcionalidad en el manejo de neurodatos. El artículo se centra en el diseño de una arquitectura que permite la adquisición automática y manual de datos en diversos formatos, así como la automatización del flujo de trabajo y los procesos de investigación. Ejemplar un método como el análisis documental y el modelado para desarrollar algo que facilite el procesamiento de datos y el aprendizaje automático, además que tiene un enfoque a microservicios, lo que permite estabilidad y eficiencia.

## Bibliografía

González, J. E. F., & García, C. A. O. Propuesta de diseño de arquitectura de software para la gestión, análisis y procesamiento de datos de neurociencia. <https://n9.cl/xz9s7>

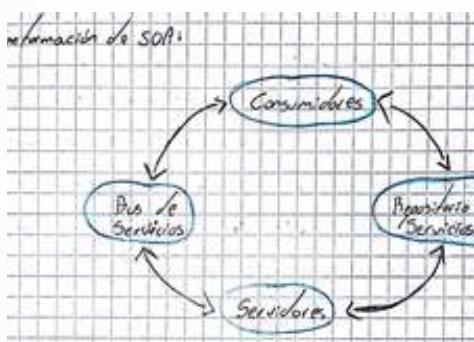


## Reflexión

El artículo nos destaca la importancia de una arquitectura de software bien diseñada en el campo de la ciencia, donde una buena gestión de datos es importante para la investigación. Al estar basado en microservicios no sólo nos permite la escalabilidad y mantenimiento, sino que también promueve la interoperabilidad entre distintos sistemas y formatos. También nos facilita la automatización de procesos en la integración de algoritmos de aprendizaje automático, lo cual nos da nuevas oportunidades para el análisis de datos complejos.

# Arquitectura de Software. Arquitectura Orientada a Servicios

El artículo nos muestra el concepto de Arquitectura Orientada a Servicios (SOA). Como un enfoque arquitectónico que nos permite la creación de sistemas de software flexibles y escalables. SOA se basa en que los servicios son componentes independientes, los cuales pueden comunicarse entre sí con interfaces bien definidas. El artículo también nos muestra los beneficios de implementar SOA, lo cual lleva a la mejora, la reducción de costos en mantenimiento y la capacidad de adaptarse rápidamente a los diferentes cambios.



## Bibliografía

- Martín, Y. E. (2012). Arquitectura de software. Arquitectura orientada a servicios. Serie Científica de la Universidad de las Ciencias Informáticas, 5(1), 1-10.  
<https://n9.cl/p7pho>

## Reflexión

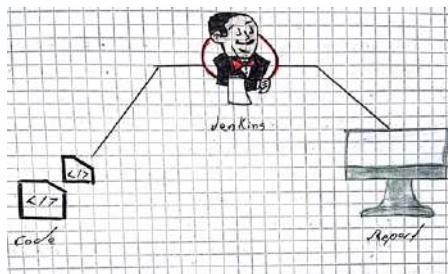
La arquitectura SOA representa un cambio significativo en la forma en la que diseñamos y gestionamos los sistemas de software actualmente. Ya que los servicios se comunican de manera independiente, SOA no sólo mejora la interoperabilidad, sino que también nos permite una mayor flexibilidad en el desarrollo de aplicaciones. La implementación de SOA en nuestros desarrollos puede reducir los costos y aumentar la eficiencia. Mostrándonos que la arquitectura orientada a servicios nos ayuda a mejorar en nuestros proyectos de software.

# Aprendiendo arquitectura de software a partir de proyectos de código abierto en GitHub

El artículo nos muestra cómo en la universidad de Sevilla, enseña una arquitectura de software a estudiantes mediante el análisis de proyectos de código abierto en GitHub. Se inspira en un método usado en la Delft University, esto les permite a los estudiantes aprender los conceptos arquitectónicos aplicándolos a proyectos reales. Los estudiantes trabajaron en grupos y utilizaron herramientas como SonarQube. Esto se evaluó con 258 estudiantes teniendo como resultado mejores calificaciones, lo que permitió desarrollar mejores habilidades y ayudó a un mayor conocimiento.

## Bibliografía

Sánchez, A. B., Parejo, J. A., Estrada-Torres, B., Márquez-Chamorro, A. E., del-Río-Ortega, A., & Segura, S. (2023). Aprendiendo arquitectura software a partir de proyectos de código abierto en GitHub. <https://n9.cl/x90ie>



## Reflexión

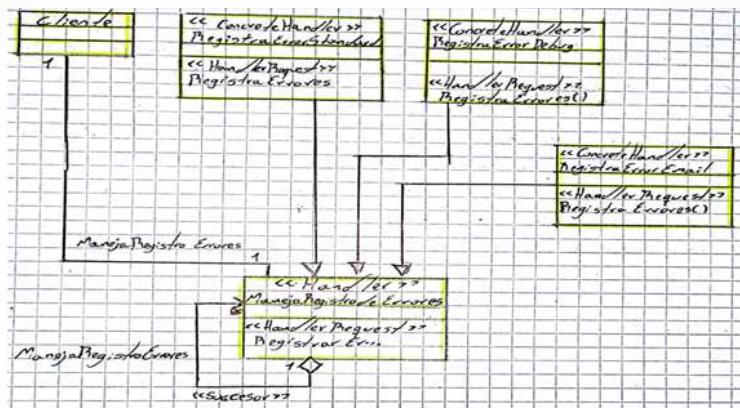
Utilizar ese tipo de prácticas en la arquitectura de software muestra una mejora en la educación. El aplicar este enfoque brinda una oportunidad de aplicar conceptos a proyectos reales, lo cual ayudó a la comprensión y retención de ese conocimiento. También el uso de herramientas mejora el desarrollo, lo cual nos prepara mejor para enfrentar entornos de desarrollo reales. Cabe mencionar que esto mejoró las habilidades colaborativas al trabajar en grupo siendo este un recurso valioso para fortalecer las competencias para el futuro.

# Una Propuesta de Implementación para Especificaciones de Patrones de Comportamiento

El artículo nos presenta una propuesta para implementar especificaciones de patrones de comportamiento (PDC) utilizando perfiles UML y restricciones OCL. En el proceso se definen 3 niveles de perfiles UML para representar patrones de diseño. Eso permite modular tantos aspectos, tanto estructurales como dinámicos, eso lo aplica diagrama de clase y secuencia en una herramienta de software. Eso permite a los desarrolladores reutilizar soluciones de diseño, lo cual hace todo más fácil en proyectos complejos, permitiendo un mejor desarrollo.

## Reflexión

Nos muestra la importancia de estructurar patrones de comportamiento en el desarrollo de software, más cuando se manejan sistemas complejos. La incorporación de UML y restricciones OCL, facilita a los desarrolladores el mantenimiento y las actualizaciones del sistema. Lo cual permite la reutilización, sino también la mejora en la colaboración y claridad del diseño. Por último, el artículo nos invita a reflexionar sobre las especificaciones y las herramientas adecuadas en la creación de arquitectura adaptable y escalables.



## Bibliografía

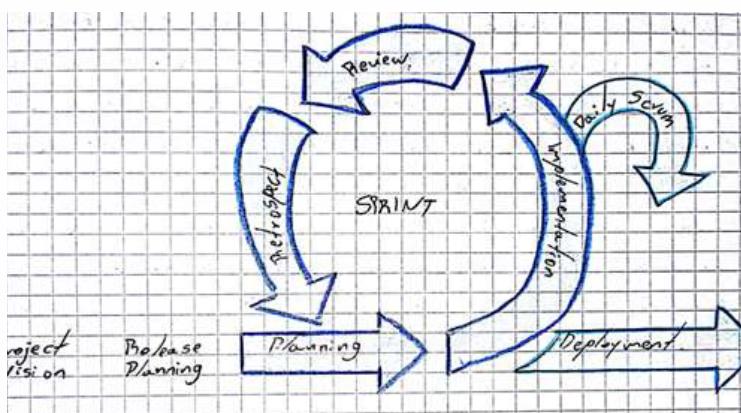
Cortez, A. A., & Naveda, C. A. Una propuesta de implementación para especificaciones de patrones de comportamiento. <https://n9.cl/x6wz7>

# Selección de Metodologías Ágiles e Integración de Arquitecturas de Software en el Desarrollo de Sistemas de Información

El artículo muestra las metodologías ágiles como ICONIX Y SCRUM. Estas metodologías ágiles priorizan la adaptabilidad y la respuesta rápida a los cambios. El artículo nos propone un modelo que permite integrar arquitecturas flexibles en proyectos, lo que permite mantener los principios de calidad y minimiza los riesgos y costos. Además, menciona que ICONIX Y SCRUM soportan beneficios para proyectos variables. Lo cual busca un modelo ágil, adaptable en cualquier momento y útil en la gestión de sistemas.

## Reflexión

El implementar una arquitectura de software representa un avance significativo para el desarrollo del proyecto. El Sprint 0 en SCRUM es innovador, ya que establece una arquitectura sólida sin afectar la flexibilidad. Esto es importante ya que garantiza que nuestro sistema pueda adaptarse a diferentes cambios sin que su calidad se vea afectada. Al combinarse con ICONIX, nos ofrece un camino viable para implementar soluciones ágiles sin que se pierda el control de toda la estructura del software, lo que permite maximizar la inversión.



## Bibliografía

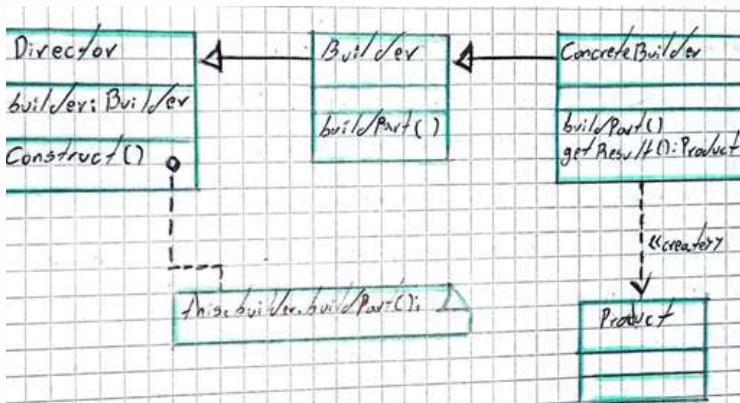
- Navarro, M. E., Moreno, M. P., Aranda, J., Parra, L., Rueda, J. R., & Pantano, J. C. (2017, September). Selección de metodologías ágiles e integración de arquitecturas de software en el desarrollo de sistemas de información. In XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017, ITBA, Buenos Aires). <https://n9.cl/53lyw>

# CLASIFICACIÓN DE LOS PATRONES DE DISEÑO IDÓNEOS EN PROGRAMACIÓN ANDROID

Este artículo muestra la importancia de usar patrones de diseño en el desarrollo de software, especialmente en las aplicaciones móviles de Android. Ya que los patrones de diseño son soluciones reutilizables para problemas comunes en programación que ayudan a no crear código espagueti. Se mencionó la evolución de los patrones de diseño desde su introducción. Nos muestran las 3 categorías de patrones: creacionales, estructurales y de comportamiento, aunque en el artículo se centra especialmente en los 2 primeros, mencionando también que esto lleva a un desarrollo más eficiente.

## Reflexión

El implementar patrones de diseño en la programación no sólo es recomendado, sino una necesidad en el desarrollo de software actual. Usar patrones de diseño es una herramienta esencial para nosotros los desarrolladores. Esto nos proporciona un marco para poder resolver problemas, promover la reutilización de código y mejorar la calidad. Evitar el código espagueti es importante, ya que un código bien estructurado es mucho más fácil de entender y mantener, también disminuye los riesgos a errores y fallos que éste pueda presentar.



## Bibliografía

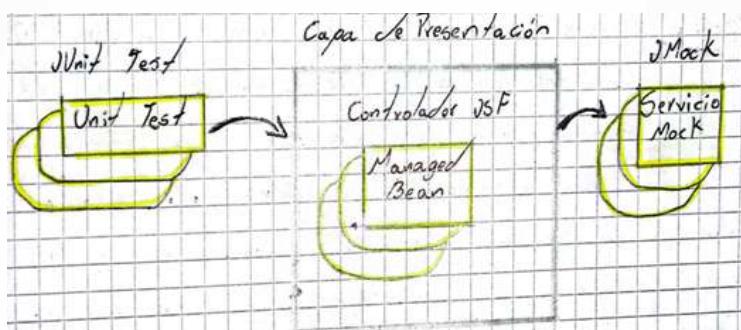
- Astorga, J. A. I., Tirado, J. L. O., Ramírez, R. U. R., Mendoza, J. C. L., & Garzón, A. P. (2019). Clasificación de los patrones de diseño idóneos en programación Android. Revista Digital de Tecnologías Informáticas y Sistemas, 3(1). <https://n9.cl/g02z3p>

# Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo

En este artículo se nos presenta un marco de trabajo para integrar arquitecturas de software en metodologías ágiles, utilizando Extreme Programming (XP) y guiado por pruebas TDD. Nos propone una arquitectura en capas, donde está la capa de presentación, negocio y persistencia, donde esta se testean mediante TDD para asegurar la calidad del software. También utilizan la herramienta JUnit, frameworks como JSF, Spring e Hibernate para poder implementar cada capa de la arquitectura. Esto ayuda a mantener la calidad del código.

## Reflexión

El combinar metodologías ágiles con una arquitectura sólida, proponiendo un marco de trabajo que emplea TDD dentro de la arquitectura, permite a todos los desarrolladores tener mejor control del sistema y una mejor respuesta al cambio de este. Lo cuál enseña como una estructura incapaz con pruebas unitarias, nos permite detectar errores mucho más fácil, lo cual nos facilita el mantenimiento. Usar las herramientas mencionadas igualmente facilita el desarrollo y nos permite crear un software más adaptable y escalable.

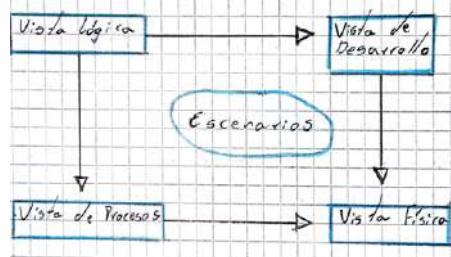


## Bibliografía

Vivas, H. L., Cambarieri, M. G., García Martínez, N., Muñoz Abbate, H., & Petroff, M. (2013). Un marco de trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo. <https://n9.cl/ebfpu>

# ARQUITECTURAS DE SOFTWARE PARA ENTORNOS MÓVILES

Este artículo nos presenta una arquitectura de software móvil (ASM) para aplicaciones Android colaborando con la universidad de Quindío y EtherealGF S.A.S. El objetivo que nos presentan es querer adaptar los principios de la arquitectura tradicional para entornos móviles, para sí mismo asegurar la calidad y la eficiencia de la aplicación. Esta se organizó en 3 tipos de capas: dependiente, independiente y conceptual, cada uno orientada a distintos módulos, como Google Maps y REST. También se usó el modelo de vistas cuatro uno para asegurar la calidad, rendimiento y la usabilidad.



## Reflexión

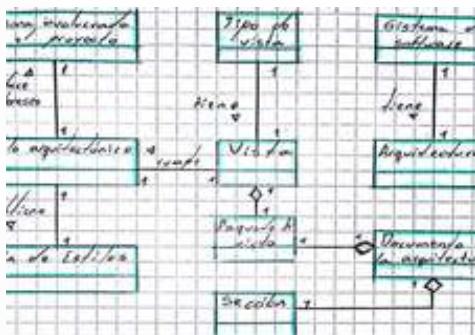
La arquitectura de software para las aplicaciones móviles son fundamentales y en este artículo nos muestran cómo adaptar arquitecturas tradicionales para la necesidad de las plataformas móviles, donde la calidad y eficiencia son importantes ya que hay dispositivos limitados en sus especificaciones técnicas. El artículo nos muestra la importancia de aplicar prácticas arquitectónicas sólidas para la creación de aplicaciones móviles, lo cual beneficia a la eficiencia en el desarrollo de estas aplicaciones móviles.

## Bibliografía

Granada, E. Z., Rodríguez, L. E. S., Montoya, C. E. G., & Uribe, C. A. C. (2014). Arquitecturas de software para entornos móviles. Revista de Investigaciones Universidad del Quindío, 25(1), 20-27. <https://n9.cl/1gp12>

# Documentando la Arquitectura de Software

El artículo nos habla de los principios básicos para documentar la arquitectura de software, nos presenta algunos conocidos, como el modelo 4 + 1 de Philippe Kruchten y el método de Robert L. Nord. Amo menciona en el uso de múltiples vistas como lo son: la lógica, proceso, desarrollo, física, estos representan varios aspectos del sistema. Nos muestran la propuesta del SEI (Vistas y más allá) y lo que es el estándar IEEE. Igual nos dan recomendaciones para optimizar la documentación, estilos arquitectónicos reutilizables, entre otros.



## Bibliografía

Gómez, O. Documentando la arquitectura de software.  
<https://n9.cl/wv1mw>

## Reflexión

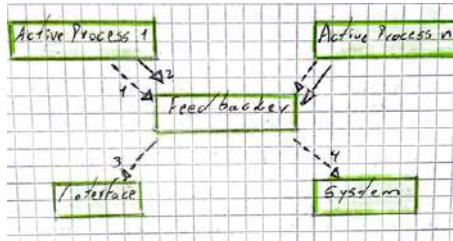
El artículo nos muestra cómo la buena documentación de la arquitectura de software es de gran ayuda para la comunicación, mantenimiento y actualizaciones de los sistemas. El tener diferentes enfoques y adaptar la documentación del software a las necesidades, facilita la toma de decisiones y asegura que todo cumpla con los requisitos de calidad. Lo cual nos permite a nosotros los desarrolladores no sólo cumplir con los requerimientos sino también facilitar la colaboración en los equipos de desarrollo de software.

# Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el Momento Arquitectónico

El artículo nos presenta como la usabilidad puede mejorar el sistema de software mediante la implementación de patrones arquitectónicos. Se presentan varios patrones específicos diseñados para resolver problemas de usabilidad, cada uno tiene una descripción, ventaja y beneficios en términos de la usabilidad. Un ejemplo es el patrón **Feedbacker**, este proporciona información al usuario sobre el estado del sistema, eso aumenta la satisfacción y reduce la carga de trabajo. El artículo nos muestra la importancia de la usabilidad desde el inicio del desarrollo.

## 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://n9.cl/6s49x>



## Reflexión

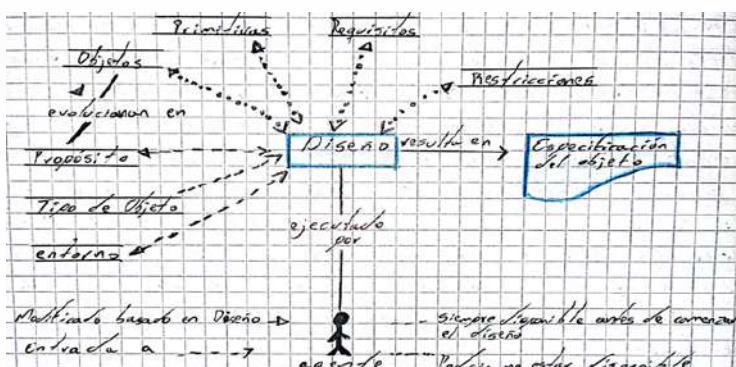
El integrar patrones de usabilidad en nuestro diseño arquitectónico es importante para crear un software mucho más intuitivo y mucho más eficiente, ya que, si abordamos las necesidades desde un inicio del desarrollo, se puede mejorar la experiencia al usuario y mejorar el rendimiento del software. Esto ayudaría en gran medida a la interacción del usuario con el sistema. Al usar la usabilidad en nuestros desarrollos de software, nos garantiza un software mucho más accesible y satisfactorio.

# Soporte a la Actividad de Diseño Basado en Patrones de Diseño

El artículo presenta la importancia de los patrones de diseño en la arquitectura de software, mostrándonos las especificaciones y el desarrollo de sistemas. Nos menciona que la descripción de un objeto y el proceso del diseño son importantes para generar soluciones efectivas y proponen la investigación para mejorar la especificación de patrones y sus aplicaciones en el desarrollo de software. En todo el documento nos plantean actividades de investigación y desarrollo que buscan avanzar en la interpretación y aplicación de patrones de diseño.

## Reflexión

El artículo muestra la importancia de patrones de diseño y herramientas para estructurar y optimizar bien nuestro desarrollo de software. Poniendo en práctica lo planteado en el artículo mejora la calidad del software al final y promueve una mejor colaboración de trabajo en equipo al momento de desarrollar, nos incentiva a la innovación en el diseño de software. Al avanzar rápidamente la tecnología, cuando la adaptación y la evolución de los patrones se vuelve importante para los desafíos a futuro.

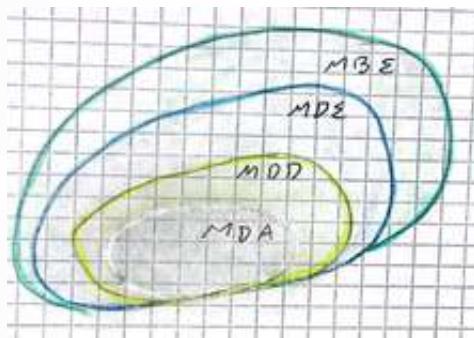


## Bibliografía

- Roqué Fourcade, L. E., & Arakaki, L. (2015, May). Soporte a la actividad de diseño basado en patrones de diseño. In XVII Workshop de Investigadores en Ciencias de la Computación (Salta, 2015). <https://n9.cl/3375h>

# **MODELO DE CASOS DE USO. SU APLICACIÓN EN UN ENTORNO DE DESARROLLO GLOBAL DE SOFTWARE**

El artículo analiza el modelo de casos de uso con una herramienta clave para definir requisitos funcionales de software. Aunque ampliamente utilizado en entornos de desarrollo co - localizados, su aplicación en contextos globales distribuidos presenta desafíos únicos. La investigación propone la plantilla CUPIDo, diseñada para documentar casos de uso de manera flexible, adaptándose a las necesidades de desarrollo colaborativo, ya sea sincrónico o asincrónico. También explora cómo estos pueden integrarse en procesos de Desarrollo Dirigido por Modelos (MDD), generando diagramas útiles para las etapas posteriores al desarrollo.



## **Bibliografía**

Lund, M. I., Chavez, S. B., Ormeño, E. G., Martin, A., & Matturro, G. MODELO DE CASOS DE USO. SU APLICACIÓN EN UN ENTORNO DE DESARROLLO GLOBAL DE SOFTWARE. <https://n9.cl/hr2oz>

## **Reflexión**

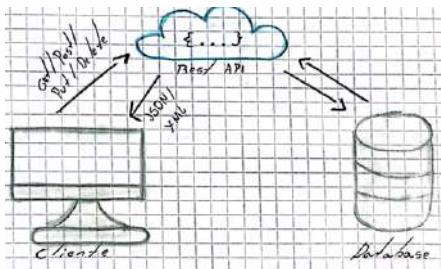
El artículo nos muestra cómo el desarrollo global transforma las prácticas tradicionales de ingeniería de software, exigiendo herramientas adaptadas a contextos multiculturales y distribuidos. La plantilla CUPIDo es un ejemplo de innovación que facilita la especificación de requisitos y la integración de modelos en procesos más amplios, como el MDD. La investigación refleja la importancia de la comunicación en equipos remotos y como el uso de tecnologías adecuadas pueden mitigar problemas. Esto refuerza la idea de que un software de calidad no sólo depende de la técnica, sino también de la capacidad de adaptación.

# Implementación de MVCC para la Generación de Aplicaciones Web en Tiempos de Entrega Reducidos

El artículo nos muestra una arquitectura innovadora para aplicaciones web llamada MVCC (Modelo - Vista- Controlador - Controlador), que optimiza tiempos de entrega al dividir la lógica de control en 2 controladores: uno en el cliente usando JavaScript con AJAX y otro en el servidor e implementando REST. Esto permite mejor compatibilidad con dispositivos móviles. La propuesta también incluye el uso de JSON para la comunicación entre controladores y el framework RedBeanPHP para facilitar el manejo de base de datos.

## Bibliografía

Rodríguez, I. Y. A., & Cortes, E. E. P. R. Implementación de MVCC para la generación de Aplicaciones Web en tiempos de entrega reducidos.  
<https://n9.cl/rctbb>



## Reflexión

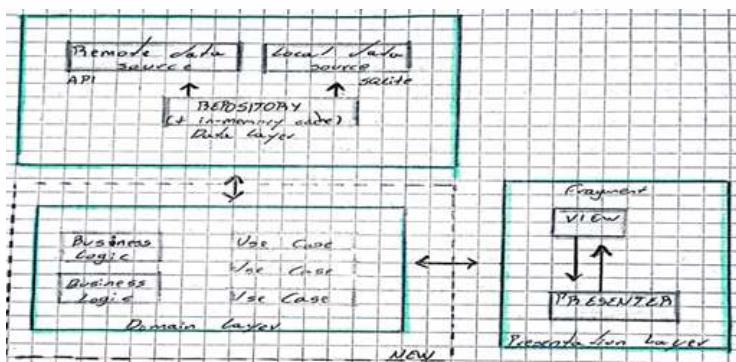
La arquitectura MVCC nos demuestra como la incorporación de tecnologías modernas puede transformar el desarrollo web, equilibrando la carga entre cliente y servidor, lo que permite aplicaciones más rápidas y versátiles. Actualizar framework y técnicas con una curva de aprendizaje baja, la propuesta resulta accesible para desarrolladores con diferentes niveles de experiencia, fomentando buenas prácticas de programación sin sacrificar la eficiencia. Esta metodología nos muestra la importancia de actualizar herramientas y paradigmas para satisfacer las necesidades actuales.

# Repercusión de Arquitectura Limpia y la Norma ISO/IEC 25010 en la mantenibilidad de Aplicativos Android

El artículo evalúa el impacto de la arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad del aplicativo móvil Educar Teacher, comparándolo con una aplicación convencional (CRM Distribución). La arquitectura limpia, propuesta por Robert Martin, organiza las aplicaciones en capas separadas por responsabilidades, promoviendo cohesión y bajo acoplamiento. La investigación emplea cuasi-experimental con métrica de analizabilidad, cambiabilidad, estabilidad y testeabilidad. Los resultados muestran mejoras significativas, lo cual evidencia que adoptar una arquitectura limpia mejora la capacidad de mantenimiento, etc.

## Reflexión

El artículo muestra la relevancia de la arquitectura limpia y las normas de calidad en el desarrollo de software, especialmente en aplicaciones móviles que requieren actualizaciones constantes. La mejora en las habilidades demuestra la eficiencia de dividir responsabilidades en capas, mientras que los incrementos modestos en otras métricas reflejan desafíos en la implementación completa de buenas prácticas. Este enfoque fomenta la creación de aplicaciones más duraderas y escalables, aunque existe un compromiso inicial de aprendizaje. Esto garantiza su sostenibilidad, reduce costos y aumenta la satisfacción del cliente.



## Bibliografía

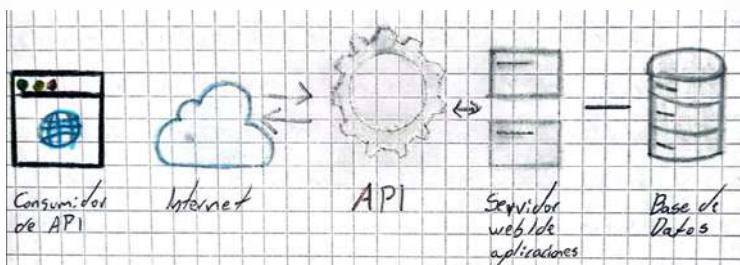
- Arias-Orezano, J. F., Reyna-Barreto, B. D., & Mamani-Apaza, G. (2021). Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android. *TecnoLógicas*, 24(52), 226-241. <https://n9.cl/s4de8>

# Metodología para la Enseñanza del Desarrollo de Software Modular

El artículo presenta una metodología educativa para la enseñanza del desarrollo de software modular, orientada a estudiantes de ingeniería en sistemas. La metodología integra el contexto como elemento clave para conectar teoría y práctica, fomentando habilidades de diseño y codificación escalables. A través de un enfoque iterativo, los estudiantes desarrollan una API web, siguiendo etapas como levantamiento de requisitos, diseño conceptual, implementación y pruebas. Se destacan conceptos claves como la programación orientada a objetos, patrones de diseño y el uso de herramientas como Swagger y DocAsCode.

## Reflexión

El artículo refleja como una metodología bien estructurada puede cerrar problemas en la formación de desarrolladores, mostrando que integrar el contexto y herramientas adecuadas facilita el aprendizaje. El énfasis en prácticas modernas, como código limpio y documentación integrada, prepara a los estudiantes para enfrentar desafíos reales. Ese enfoque resalta la importancia de enseñar no sólo técnicas de codificación, sino también habilidades críticas de diseño, validación y documentación, asegurando un buen futuro profesional.



## Bibliografía

Guevara, W. J. T. Metodología para la enseñanza del Desarrollo de Software Modular. <https://n9.cl/7hu2ba>



2694667