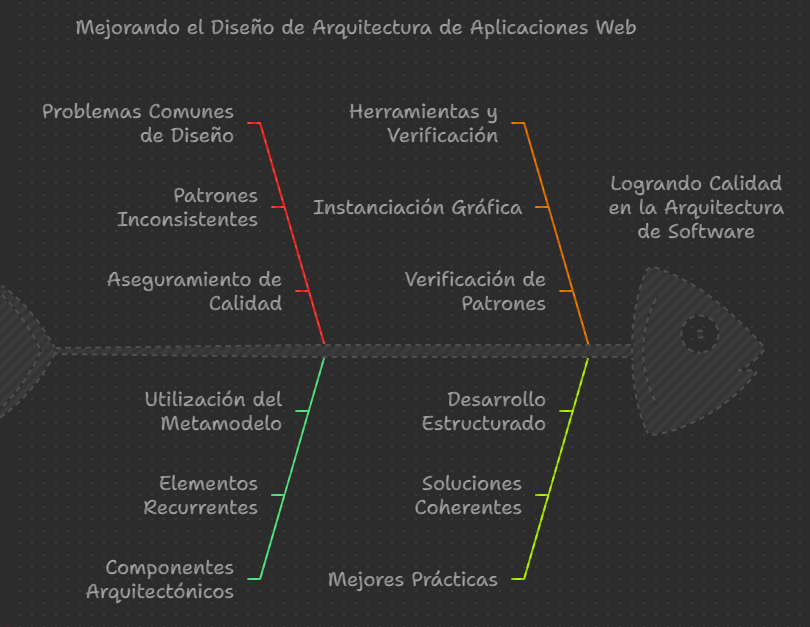
**ARTÍCULO 1 :** Modelado y verificación de patrones de diseño de arquitectura de software para entornos de computación en la nube.

**RESUMEN :** 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 instanciación gráfica y un sistema de verificación de patrones de diseño. Esto asegura la correcta aplicación de patrones, fomentando soluciones más coherentes y estructuradas. Como resultado, se promueve el desarrollo de aplicaciones web robustas, eficientes y alineadas con buenas prácticas.

**DIAGRAMA :   
**

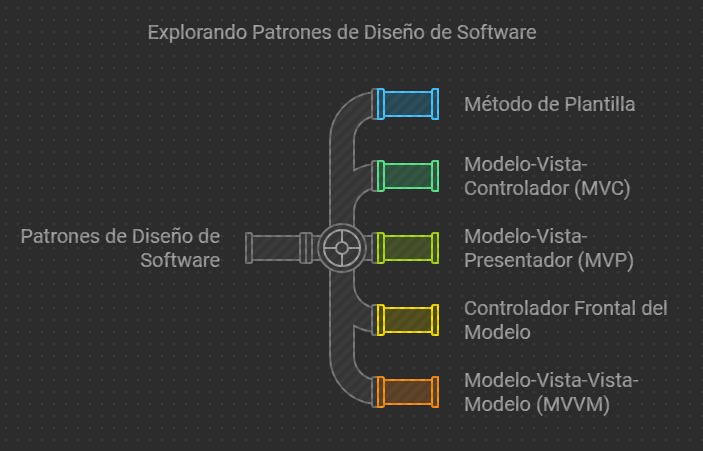
**BIBLIOGRAFÍA :Referencia: Asociación Ibérica de Sistemas y Tecnologías de Información** [**https://ri.conicet.gov.ar/handle/11336/125130**](https://ri.conicet.gov.ar/handle/11336/125130)**.**

**ARTÍCULO 2 :** Análisis comparativo de Patrones de Diseño de Software.

**RESUMEN :**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. Se concluye que los patrones de diseño son estructuras bien definidas que permiten mantener una lógica de organización en el código de un sistema, gracias a esto se puede crear software de calidad, con más facilidad de mantenimiento y con una mejor comprensión del código al buscar modularidad en el sistema.

**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.

**DIAGRAMA :**

****

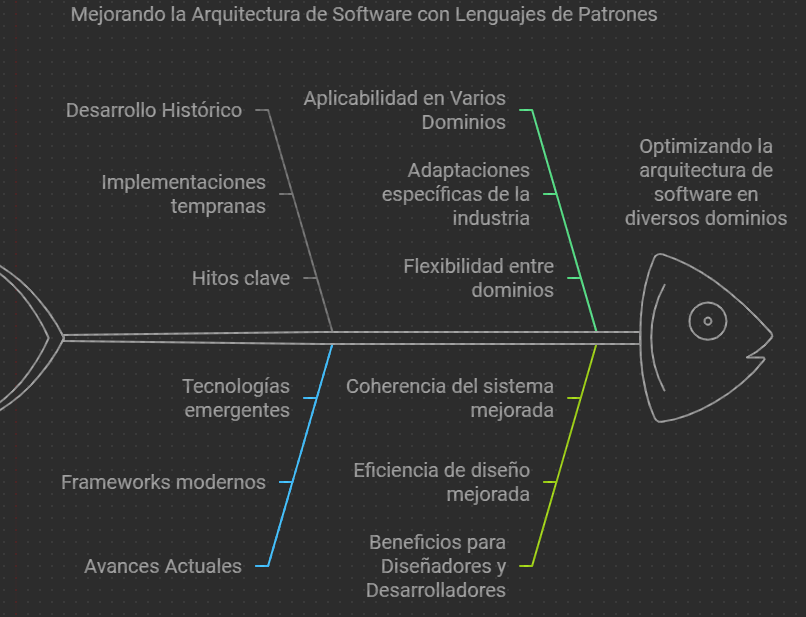
**BIBLIOGRAFÍA : Referencia :** Escuela Superior Politécnica de Chimborazo (ESPOCH), Riobamba, Ecuador <https://dialnet.unirioja.es/servlet/articulo?codigo=9042927>.

**ARTÍCULO 3 :** Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte.

**RESUMEN :** 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.

**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.

**DIAGRAMA :**

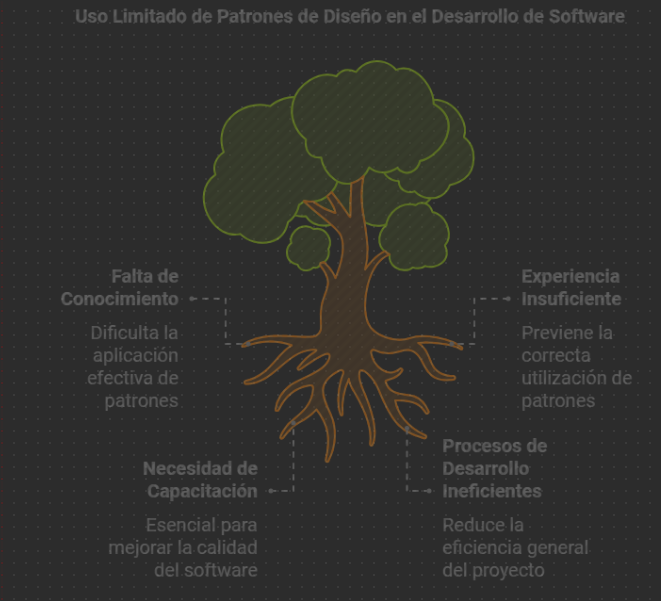


**BIBLIOGRAFÍA :**Jimenez-Torres,Victor Hugo, Tello-Borja, Wilman, Rios-Patiño Jorge Iván . Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte. Scientia Et Technica [en linea]. 2014, 19(4), 371-376[fecha de Consulta 11 de Noviembre de 2024]. ISSN: 0122-1701. Disponible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

**ARTÍCULO 4 :** Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web.

**RESUMEN :**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. Sin embargo, su uso es reducido por falta de conocimiento de la existencia de estos patrones o por falta de experiencia para lograr su correcta utilización.

**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.  
  
**DIAGRAMA :**

****

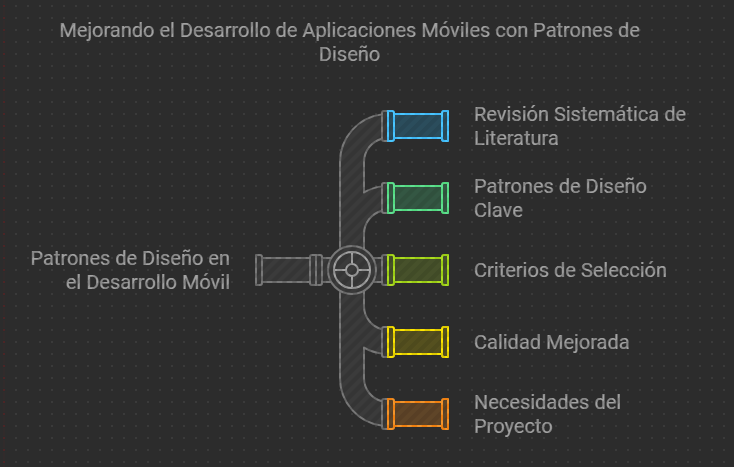
**BIBLIOGRAFÍA :** Grupo de Investigación en Ingeniería del Software-GRIIS, <https://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci_arttext&tlng=en>

**ARTÍCULO 5 :** Análisis comparativo de patrones de diseño de software para el desarrollo de aplicaciones móviles de calidad: Una revisión sistemática de la literatura.

**RESUMEN :**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.

**DIAGRAMA :**

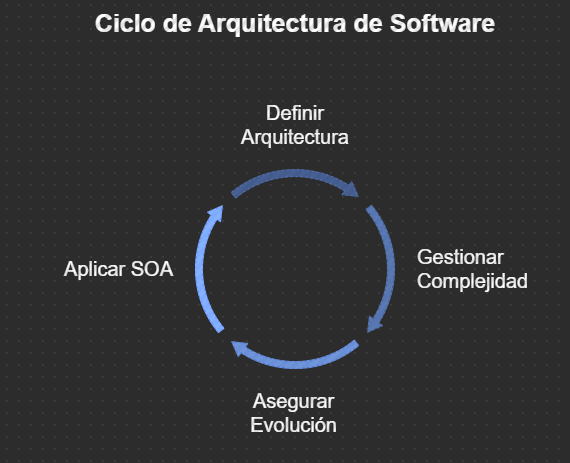


**BIBLIOGRAFÍA:**Universidad-Peruana-Unión <https://repositorio.upeu.edu.pe/items/11034d47-e68f-4b85-b9d2-83e9ceb0ef78>

**ARTÍCULO 6 :** Una breve encuesta de conceptos de arquitectura de software y arquitectura orientada a servicios.  
  
**RESUMEN:** 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. Finalmente, la arquitectura orientada a servicios (SOA) como una de las opciones útiles para que la arquitectura de software, desarrollo software y sistemas web se glosa en una encuesta.

**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.

**DIAGRAMA :**

****

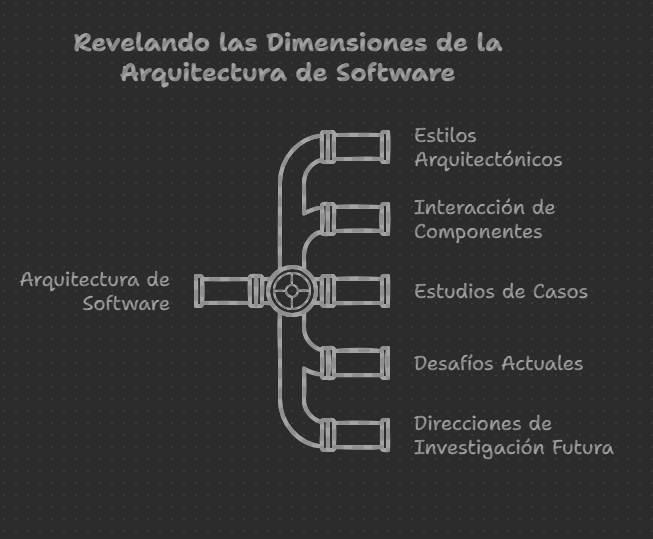
**BIBLIOGRAFÍA :** IEEE <https://ieeexplore.ieee.org/abstract/document/5235004>

**ARTÍCULO 7:** INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE.

**RESUMEN :** 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.

**DIAGRAMA :**

****

**BIBLIOGRAFÍA :** [INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE](https://www.worldscientific.com/doi/abs/10.1142/9789812798039_0001)

DAVID GARLAN (EE.UU.) y MARY SHAW (EE.UU.)

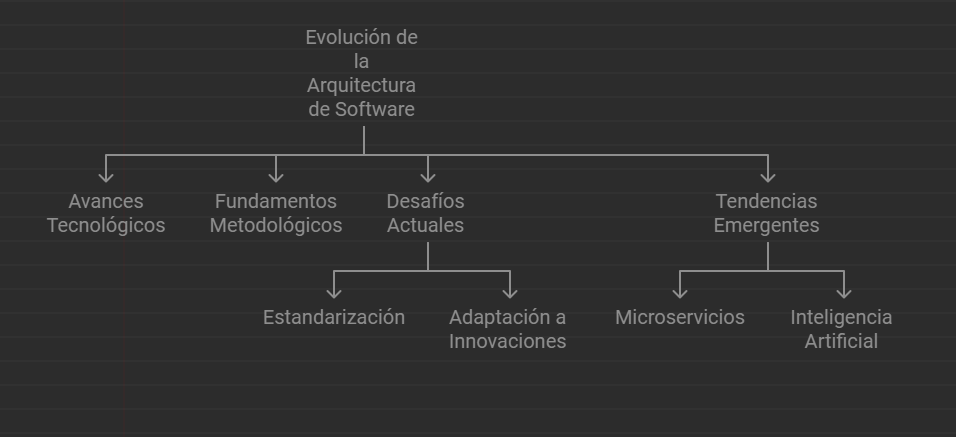
Avances en ingeniería de software e ingeniería del conocimiento. Diciembre de 1993 , 1-39

**ARTÍCULO 8 :** Arquitectura de software: un diario de viaje.

**RESUMEN :** 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.

**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.

**DIAGRAMA :**

****

**BIBLIOGRAFÍA :** Asociación de Maquinaria Computacional, Nueva York Estados Unidos <https://dl.acm.org/doi/abs/10.1145/2593882.2593886>.

**ARTÍCULO 9 :** Arquitectura de software y diseño de software.

**RESUMEN :** 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.

**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.

**DIAGRAMA :**

****

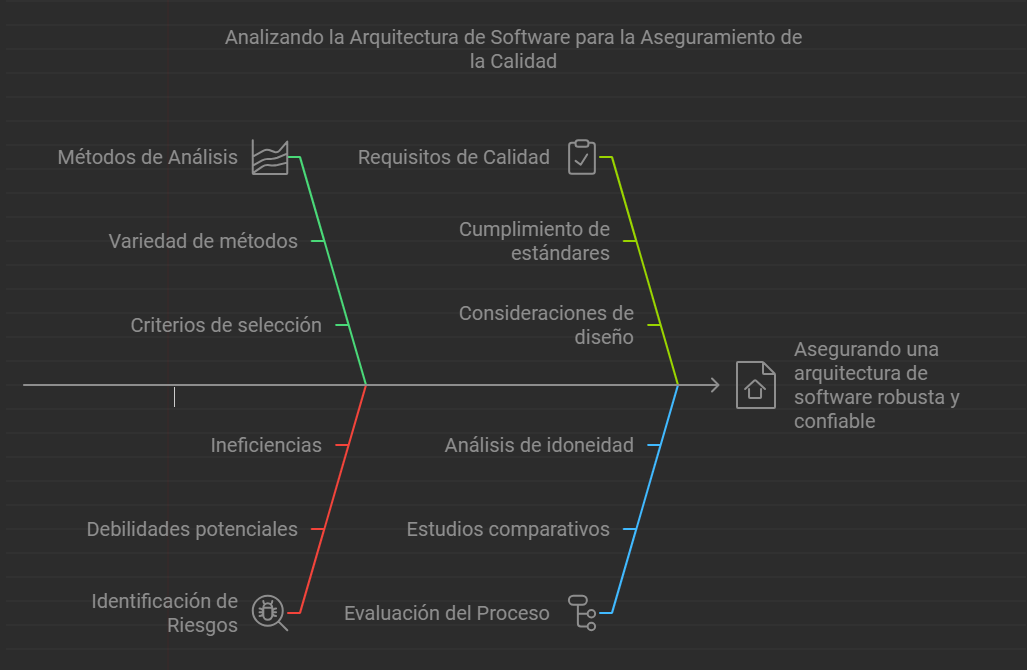
**BIBLIOGRAFÍA :** Jaiswal, Manishaben, Arquitectura de software y diseño de software (5 de noviembre de 2019). Revista internacional de investigación de ingeniería y tecnología (IRJET) e-ISSN: 2395-0056, p-ISSN: 2395-0072, Volumen: 06, Número: 11, s. no -303, pp. 2452-2454, noviembre de 2019. Disponible en: https://www.irjet.net/archives/V6/i11/IRJET-V6I11303.pdf, Disponible en SSRN: [https://ssrn.com/abstract=3772387 o http://dx.doi.org/10.2139/ssrn.3772387](https://ssrn.com/abstract=3772387%20o%20http://dx.doi.org/10.2139/ssrn.3772387)

**ARTÍCULO 10 :** Una encuesta sobre los métodos de análisis de la arquitectura de software.

**RESUMEN :** 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. El papel de la discusión es ofrecer pautas relacionadas con el uso del método más adecuado para un proceso de evaluación de la arquitectura. Nos concentramos en descubrir similitudes y diferencias entre estos ocho métodos disponibles mediante la realización de clasificaciones, comparaciones y estudios de idoneidad.

**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.

**DIAGRAMA :**

****

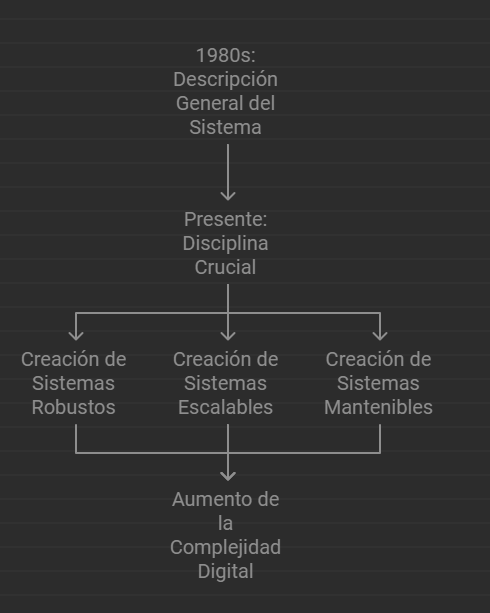
**BIBLIOGRAFÍA :** [IEEE Transactions on Software Engineering](https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=32) ( volumen: 28 , [número: 7](https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=21927&punumber=32) , julio de 2002 ).

**ARTÍCULO 11 :** La edad de oro de la arquitectura de software.

**RESUMEN :** 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.

**DIAGRAMA :**

****

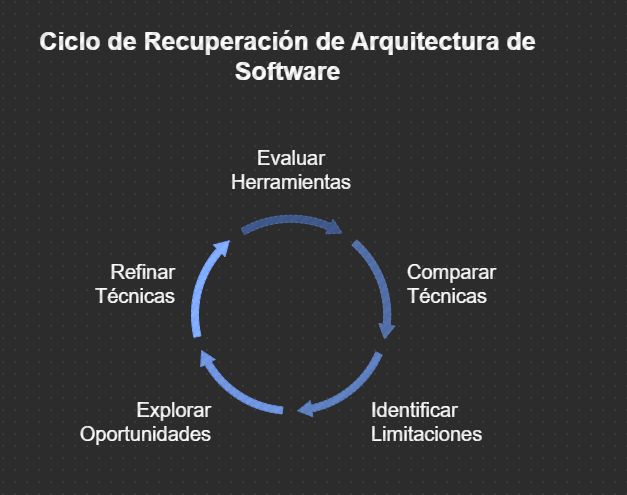
**BIBLIOGRAFÍA :** [IEEE Software](https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=52) ( Volumen: 23 , [Número: 2](https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=33727&punumber=52) , Marzo-Abril 2006 )

**ARTÍCULO 12 :** Un análisis comparativo de las técnicas de recuperación de la arquitectura de software.

**RESUMEN :** 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.

**DIAGRAMA :**

****

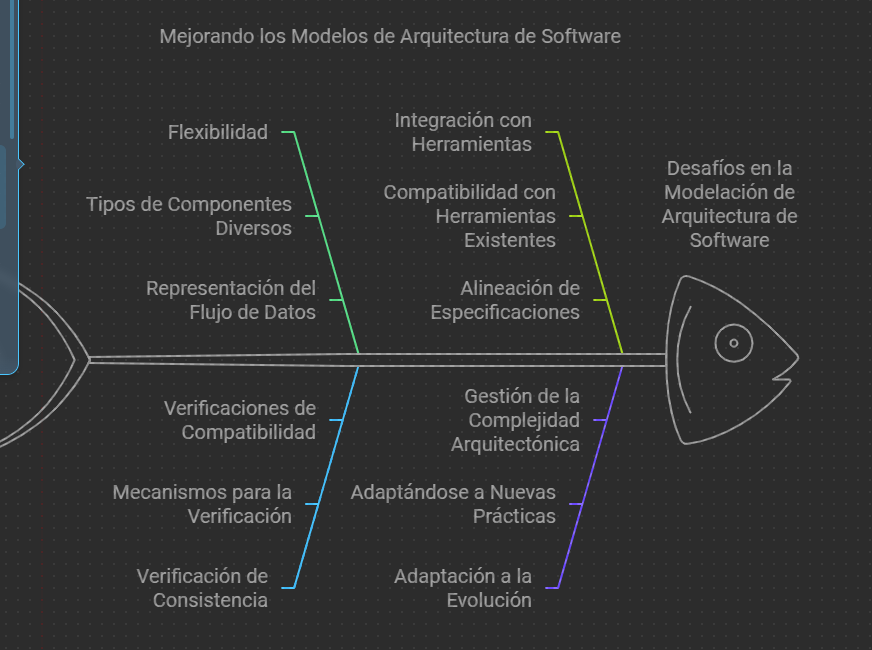
**BIBLIOGRAFÍA :** [28.ª Conferencia internacional IEEE/ACM sobre ingeniería de software automatizada (ASE) de 2013](https://ieeexplore.ieee.org/xpl/conhome/6684409/proceeding).

**ARTÍCULO 13 :** Abstracciones para la arquitectura de software y herramientas para respaldarlas.

R**ESUMEN :** 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. Además, incorpora mecanismos para verificar la consistencia y compatibilidad de los componentes, y facilita la integración con herramientas y especificaciones existentes. En esencia, el objetivo es proporcionar una base sólida para diseñar, analizar y comunicar arquitecturas de software de manera más rigurosa y efectiva.

**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. En resumen, este modelo ofrece una base sólida para futuras investigaciones y aplicaciones en el campo de la ingeniería de software.

**DIAGRAMA :**

****

**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.

keywords: {Software architecture;Computer languages;Computer architecture;Software engineering;Computer science;Software design;Software tools;Packaging;System testing;Scheduling},

**ARTÍCULO 14 :** Patrones de reconfiguración de software para la evolución dinámica de arquitecturas de software.

**RESUMEN :** 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. También se describe un modelo de gestión de cambios para la evolución dinámica de una línea de productos de software. El documento concluye con una descripción de estudios de casos y un prototipo de prueba de concepto.

**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.

**DIAGRAMA :**

****

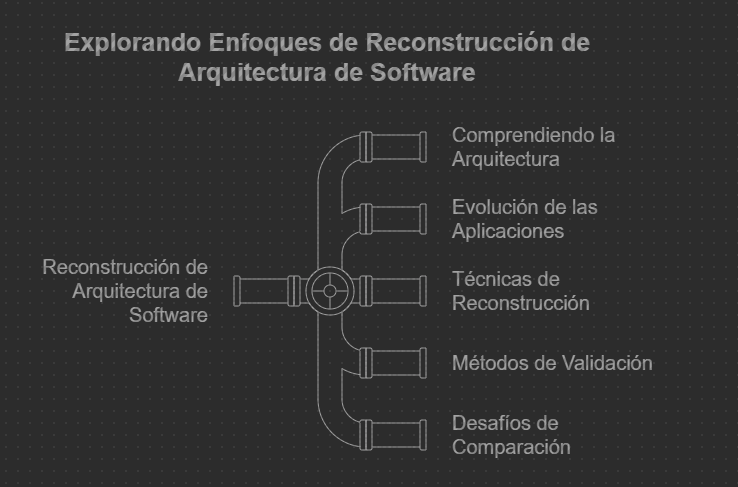
**BIBLIOGRAFÍA :** H. Gomaa and M. Hussein, "Software reconfiguration patterns for dynamic evolution of software architectures," Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004), Oslo, Norway, 2004, pp. 79-88, doi: 10.1109/WICSA.2004.1310692. keywords: {Computer architecture;Software systems;Application software;Software architecture;Software design;Runtime;LAN interconnection;Software engineering;Software prototyping;Prototypes},

**ARTÍCULO 15:** Reconstrucción de la arquitectura de software: una taxonomía orientada a procesos.

**RESUMEN :** 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. Este documento presenta un estado del arte en los enfoques de reconstrucción de la arquitectura de software.

**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. Este documento presenta una revisión exhaustiva de los enfoques actuales para reconstruir arquitecturas de software, contribuyendo así a llenar este vacío en el conocimiento.

**DIAGRAMA :**

****

**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.

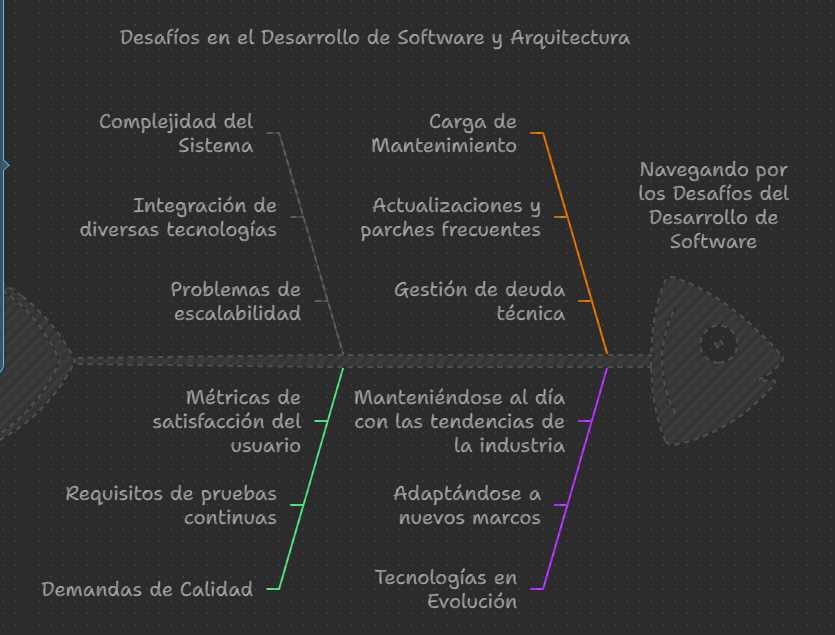
keywords: {Software architecture;Taxonomy;Computer architecture;Application software;Data mining;Programming;Europe;Packaging;Cognitive science;Bridges;Software architecture reconstruction.},

**ARTÍCULO 16 :** El papel de la vista de decisión en la práctica de la arquitectura de software

**RESUMEN :** 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. Varios años después de que comenzara la "crisis del software" a mediados de la década de 1970, la práctica de la arquitectura de software surgió como una disciplina madura (aunque todavía en crecimiento), capaz de abordar la creciente complejidad de los nuevos sistemas de software.

**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.

**DIAGRAMA :**

****

**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.

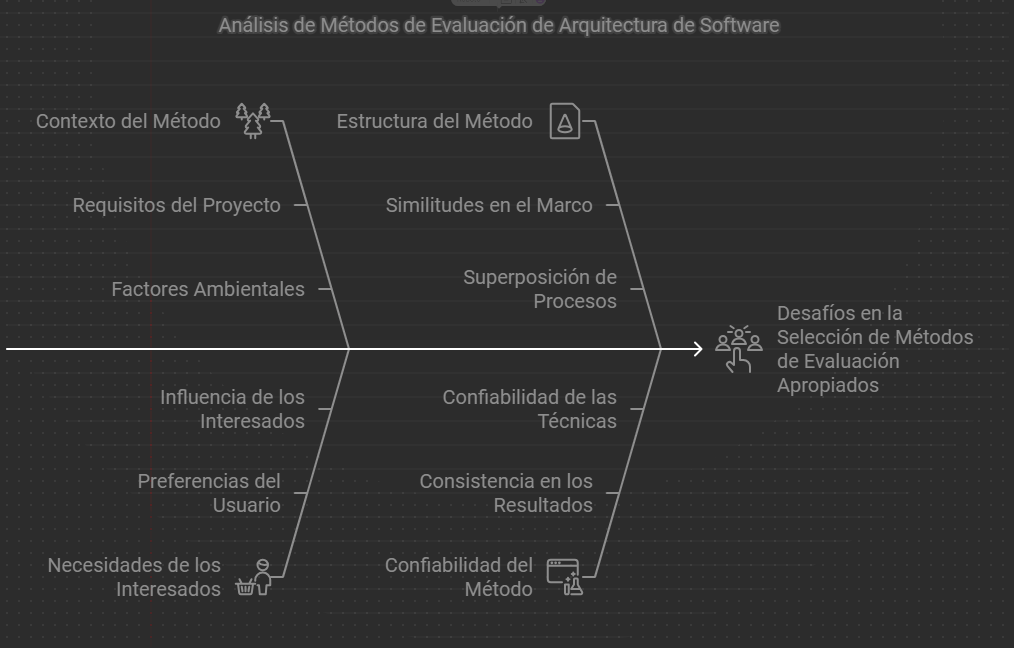
keywords: {Software architecture;Computer architecture;Context modeling;Software systems;Collaborative software;Standards development;Production systems;Software maintenance;Costs;Software design;software architecture;architectural design decision;architectural knowledge;architecture views;decision view},

**ARTÍCULO 17 :** Comparación de métodos de evaluación de arquitectura de software basados ​​en escenarios.

**RESUMEN :** 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. El marco considera cada método desde el punto de vista del contexto del método, las partes interesadas, la estructura y la confiabilidad. La comparación revela que la mayoría de los métodos estudiados son estructuralmente similares, pero existen varias diferencias entre sus actividades y técnicas. Por lo tanto, algunos métodos se superponen, lo que nos lleva a identificar cinco actividades comunes que pueden formar un modelo de proceso genérico.

**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.

**DIAGRAMA :**

****

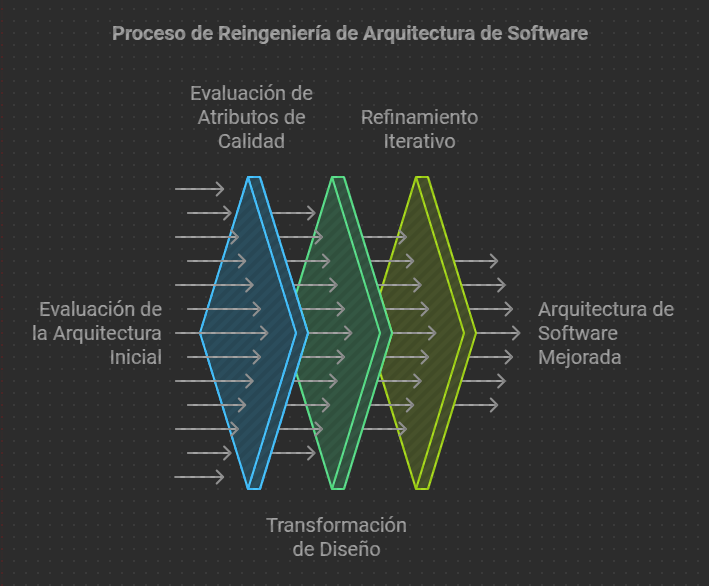
**BIBLIOGRAFÍA :** M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," 11th Asia-Pacific Software Engineering Conference, Busan, Korea (South), 2004, pp. 600-607, doi: 10.1109/APSEC.2004.38. keywords: {Software architecture;Software engineering;Computer architecture;Performance analysis;Australia;Terminology;Instruments},

**ARTÍCULO 18:** Reingeniería de arquitectura de software basada en escenarios.

**RESUMEN :** 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.

**DIAGRAMA :**

****

**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. keywords: {Software architecture;Software quality;Computer architecture;Design methodology;Software systems;Electrical capacitance tomography;Application software;Software prototyping;Prototypes;Software maintenance},

**ARTÍCULO 19 :** Descripción de estilos de arquitectura de software mediante gramáticas gráficas.

**RESUMEN :** 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. La evolución dinámica de una arquitectura se define de forma independiente mediante un "coordinador". Un estilo de arquitectura es una clase de arquitecturas especificadas por una gramática de grafos. La clase caracteriza un conjunto de arquitecturas que comparten un patrón de comunicación común. Las reglas del coordinador se verifican estáticamente para garantizar que conserven las restricciones impuestas por el estilo de 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.

**DIAGRAMA :**

****

**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.

keywords: {Software architecture;Computer architecture;Computer languages;Application software;Engineering drawings;Costs;Organizing;Software standards;Software design;Mathematical model},

**ARTÍCULO 20 :** En la práctica: descripción de diseño y arquitectura de software UML

**RESUMEN :** 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. Nuestro estudio se centró en el uso de UML y la calidad del modelo en proyectos reales, en lugar de en su idoneidad como notación o lenguaje.

**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.

**DIAGRAMA :**

****

**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.

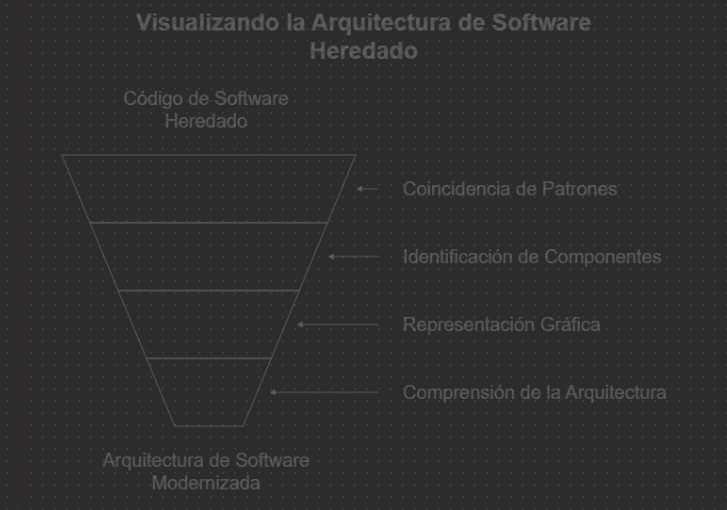
keywords: {Unified modeling language;Software architecture;Software design;Programming;Industrial control;Terminology;Inspection;Job shop scheduling;Demography;Unified Modeling Language (UML);software quality;current practice;industry survey},

**ARTÍCULO 21:** Recuperación de la arquitectura del software basada en la coincidencia de patrones.

**RESUMEN :** 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.

**DIAGRAMA:**

****

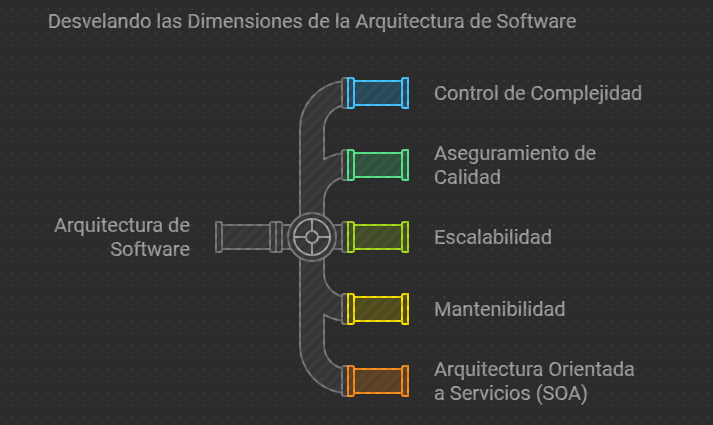
**BIBLIOGRAFÍA :** K. Sartipi, "Software architecture recovery based on pattern matching," International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., Amsterdam, Netherlands, 2003, pp. 293-296, doi: 10.1109/ICSM.2003.1235434. keywords: {Software architecture;Pattern matching;Computer architecture;Software systems;Data mining;Documentation;Pattern analysis;Reverse engineering;Software maintenance;Computer science},

**ARTÍCULO 22 :** Un breve estudio de los conceptos de arquitectura de software y arquitectura orientada a servicios.

**RESUMEN :** 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. Primero, se presentan las definiciones de arquitectura. Finalmente, se analiza en un estudio la arquitectura orientada a servicios (SOA) como una de las opciones útiles para la arquitectura de software para desarrollar software y sistemas web.

**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. En esencia, la arquitectura de software es el puente entre las ideas y la realidad digital, permitiendo que los sistemas evolucionen y se adapten al mundo en constante cambio.

**DIAGRAMA :**

****

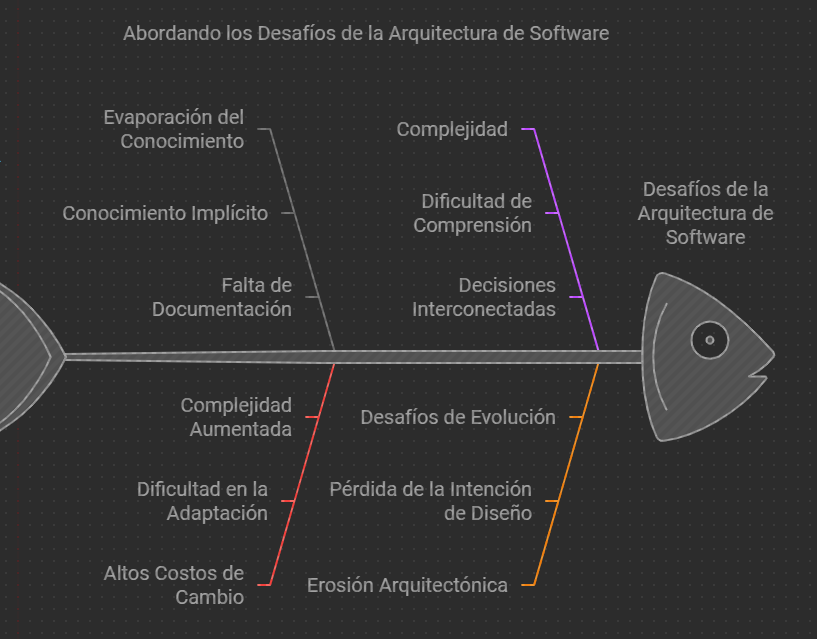
**BIBLIOGRAFÍA :**M. H. Valipour, B. Amirzafari, K. N. Maleki and N. Daneshpour, "A brief survey of software architecture concepts and service oriented architecture," 2009 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China, 2009, pp. 34-38, doi: 10.1109/ICCSIT.2009.5235004. keywords: {Software architecture;Service oriented architecture;Computer architecture;Algorithm design and analysis;Software systems;Stability;Programming;Control systems;Web services;History;Software Architecture;Service-Oriented Architecture;Web Services},

**ARTÍCULO 23** : La arquitectura de software como un conjunto de decisiones de diseño arquitectónico.

**RESUMEN :** 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. En este artículo, se presenta una nueva perspectiva sobre la arquitectura de software, que considera a la arquitectura de software como una composición de un conjunto de decisiones de diseño explícitas. Esta perspectiva hace que las decisiones de diseño arquitectónico sean una parte explícita de una arquitectura de software. En consecuencia, se reduce la evaporación del conocimiento, aliviando así algunos de los problemas fundamentales de la arquitectura de software.

**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. Como resultado, surgen dificultades para comprender y adaptar la arquitectura, lo que aumenta su complejidad y los costos de modificación. Una nueva perspectiva propone ver la arquitectura de software como un conjunto de decisiones de diseño específico. Al hacer visibles estas decisiones, se conserva mejor el conocimiento y se facilita la adaptación y evolución de la arquitectura, aliviando algunos de los problemas más comunes relacionados.

**DIAGRAMA :**

****

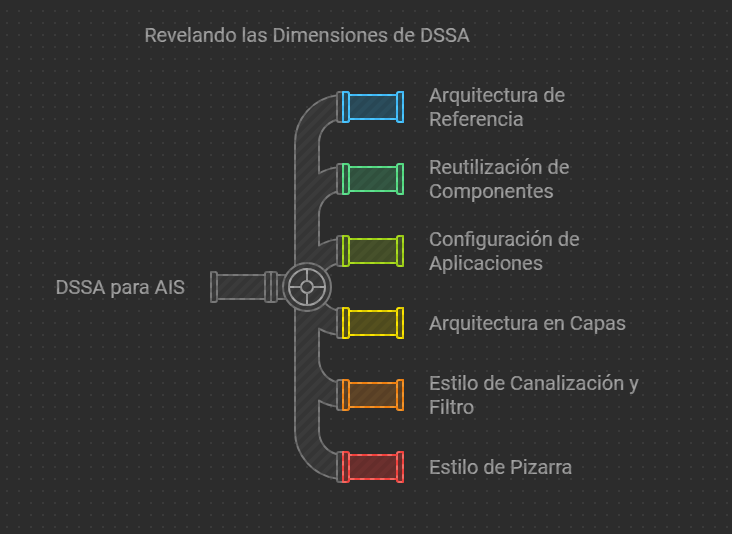
**BIBLIOGRAFÍA :** A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), Pittsburgh, PA, USA, 2005, pp. 109-120, doi: 10.1109/WICSA.2005.61. keywords: {Software architecture;Computer architecture;Costs;Software systems;Application software;Computer industry;Electrical equipment industry;Connectors;Stress control;Control systems},

**ARTÍCULO 24 :** Una arquitectura de software específica de dominio para sistemas inteligentes adaptativos

**RESUMEN :** 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. La arquitectura de referencia AIS incorpora características de arquitecturas en capas, de canalización y filtro y de estilo pizarra. Describimos tres estudios que demuestran la utilidad de nuestra arquitectura en el subdominio de los robots de oficina móviles e identificamos los principios de ingeniería de software incorporados en la arquitectura.

**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. Con características de arquitectura en capas y estilo pizarra, la DSSA es flexible y eficiente. Los estudios realizados en robots de oficina móviles demuestran su aplicabilidad práctica y reflejan principios claves como la reutilización y la escalabilidad en ingenio.

**DIAGRAMA :**

****

**BIBLIOGRAFÍA :** B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot 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.

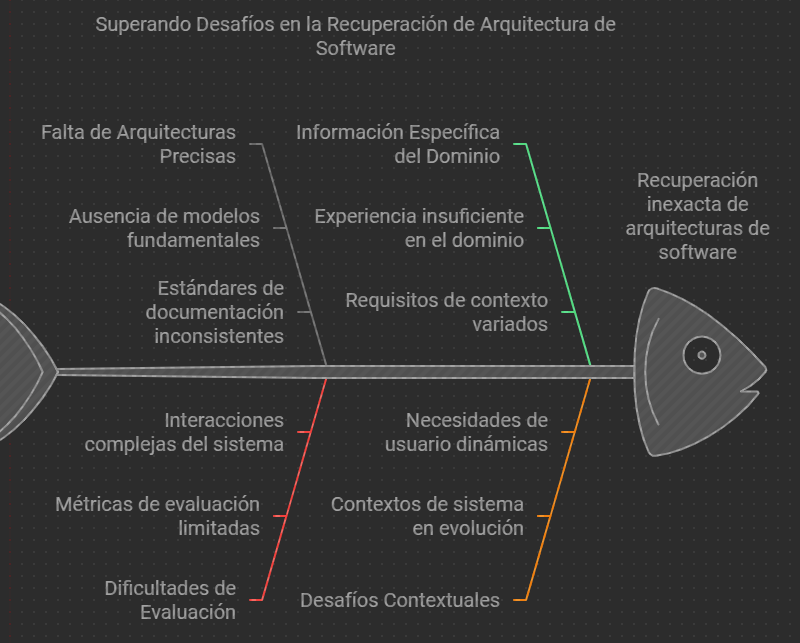
keywords: {Software architecture;Computer architecture;Application software;Decision support systems;Adaptive systems;Intelligent systems;Software libraries;Filters;Teleworking;Mobile robots},

**ARTÍCULO 25 :** Obtención de arquitecturas de software basadas en la verdad fundamental

**RESUMEN :** 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. En este artículo, presentamos nuestra experiencia al recuperar arquitecturas de verdad fundamentales de cuatro sistemas de código abierto, discutiendo los principales conocimientos obtenidos y analizando las características de estas arquitecturas. Los hallazgos sugieren que es viable obtener arquitecturas necesarias para sistemas grandes y alientan futuros esfuerzos para establecer un repositorio a gran escala de arquitectura.

**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. La experiencia con sistemas de código abierto demuestra que es posible recuperar arquitecturas precisas, incluso en sistemas grandes. Esto sugiere que, a largo plazo, se podrían crear repositorios a gran escala de arquitecturas precisas, facilitando el mantenimiento y la evolución de sistemas complejos.

**DIAGRAMA :**

****

**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. keywords: {Computer architecture;Documentation;Java;Protocols;Web servers;Software systems},

**ARTÍCULO 26 :** Un marco de clasificación y comparación para lenguajes de descripción de arquitectura de software.

**RESUMEN :** 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. Este artículo ofrece una definición y un marco para clasificar los ADL, demostrando su utilidad al compararlos con otras notaciones. A través de esta comparación, se identifican las fortalezas y debilidades de los ADL existentes, lo que abre un camino para futuros entornos.

**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. Este artículo propone una definición y clasificación de los ADL, ayudando a diferenciarlos y evaluarlos frente a otras notaciones. La comparación de ADL existente resalta sus fortalezas y debilidades, destacando la necesidad de continuar investigando y mejorando los ADL y los entornos de desarrollo.

**DIAGRAMA:**

****

**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.

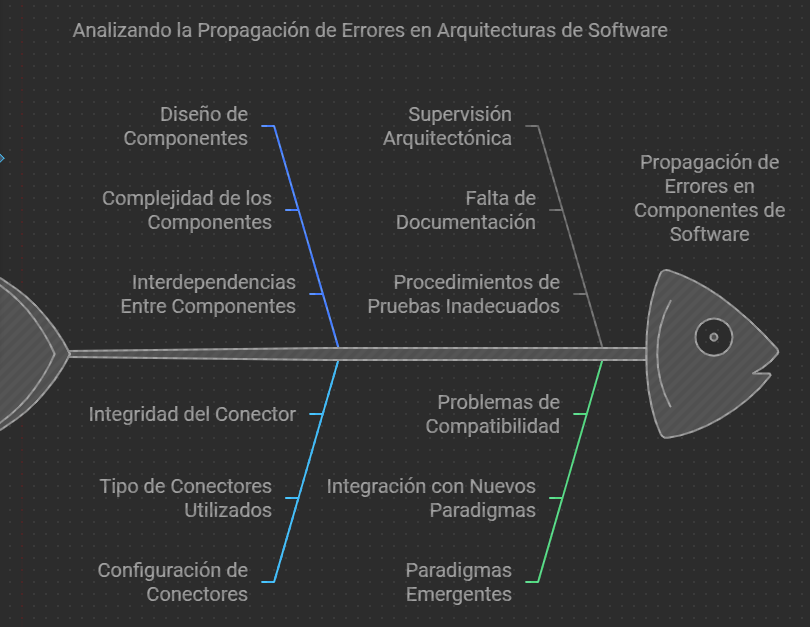
keywords: {Software architecture;Computer architecture;LAN interconnection;Architecture description languages;Application software;Formal specifications;Computer languages;Connectors;Computer science;Computer Society},

**ARTÍCULO 27:** Propagación de errores en arquitecturas de software

**RESUMEN :** 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. En este artículo, nos centramos en un atributo arquitectónico específico, que es la probabilidad de propagación de errores a lo largo de la arquitectura, es decir, la probabilidad de que un error que surge en un componente se propague a otros componentes. Presentamos, analizamos y validamos fórmulas para estimar estas probabilidades utilizando información a nivel de 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. Este artículo se enfoca en la probabilidad de propagación de errores, es decir, la posibilidad de que un error en un componente se propague a otros componentes. Se presentan, analizan y validan fórmulas para estimar estas probabilidades, utilizando información.

**DIAGRAMA :**

****

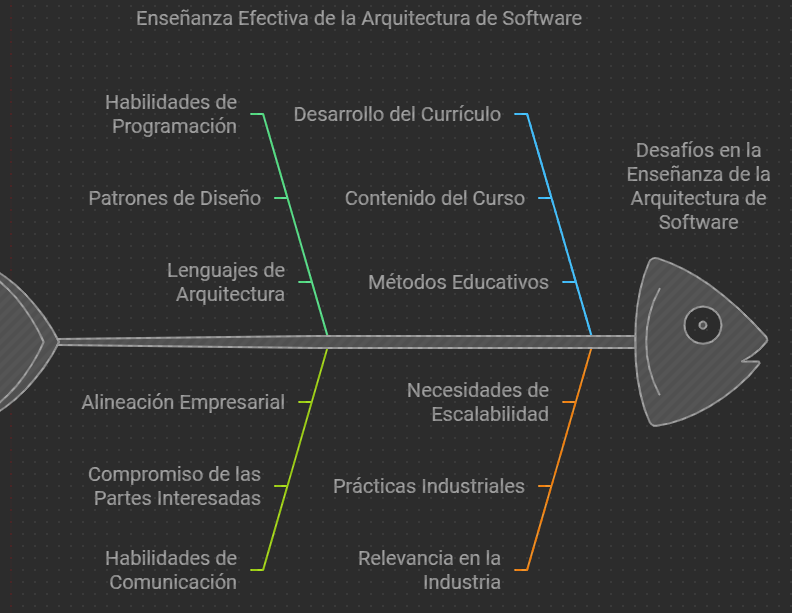
**BIBLIOGRAFÍA :**W. Abdelmoez et al., "Error propagation in software architectures," 10th International Symposium on Software Metrics, 2004. Proceedings., Chicago, IL, USA, 2004, pp. 384-393, doi: 10.1109/METRIC.2004.1357923. keywords: {Computer architecture;Software engineering;Software metrics;Computer errors;NASA;Software architecture;Information analysis;Programming;Computer science;Software safety},

**ARTÍCULO 28:** Impartición de un curso sobre arquitectura de software

**RESUMEN :**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. Se demuestra que, con el enfoque adecuado, es posible enseñar eficazmente estos aspectos clave de la arquitectura de software en un entorno académico, destacando su relevancia industrial.

**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. Las experiencias compartidas en los cursos de maestría evidencian que, con un enfoque adecuado, es posible enseñar los aspectos técnicos y comunicacionales de la arquitectura de software de forma exitosa, garantizando que los futuros ingenieros comprendan tanto los desafíos técnicos como la necesidad de adaptar las soluciones. a las necesidades del negocio. En un entorno industrial, estos conocimientos son esenciales para la creación de software robusto, escalable y alineado con las expectativas de todas las partes interesadas.

**DIAGRAMA :**

****

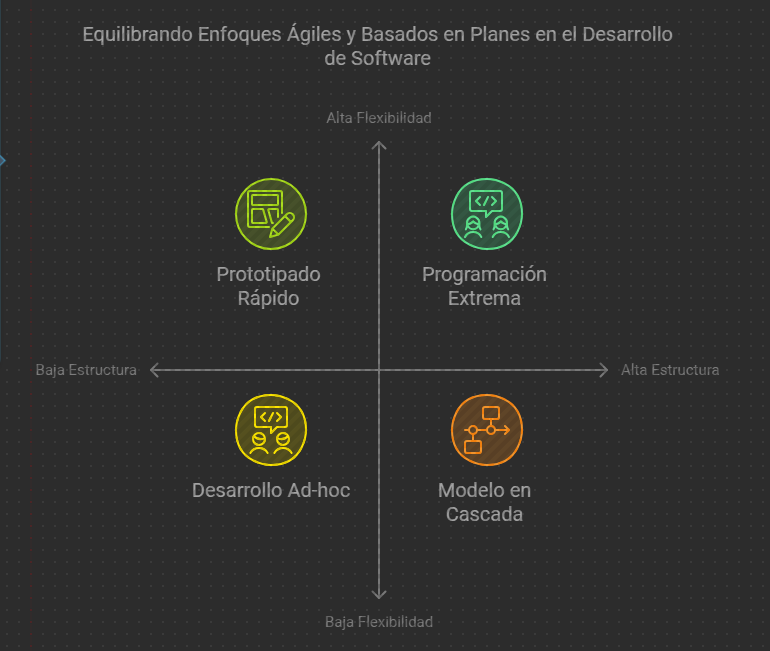
**BIBLIOGRAFÍA :** P. Lago and H. van Vliet, "Teaching a Course on Software Architecture," 18th Conference on Software Engineering Education & Training (CSEET'05), Ottawa, ON, Canada, 2005, pp. 35-42, doi: 10.1109/CSEET.2005.33. keywords: {Education;Software architecture;Computer architecture;Software engineering;Architecture description languages;Computer industry;Programming;Software standards;Maintenance engineering;Design engineering},

**ARTÍCULO 29 :** Métodos centrados en la arquitectura de software y desarrollo ágil

**RESUMEN :** 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. La arquitectura de software es parte de la calidad del producto y no está vinculada a un proceso, tecnología, cultura o herramienta en particular. Este artículo explora la relación y las sinergias entre los métodos de diseño y análisis centrados en la arquitectura y el marco de programación extrema. Elegimos centrarnos en XP porque es una de las prácticas ágiles más maduras y conocidas.

**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. Un desarrollo exitoso debe integrar diversos elementos, como el proceso, las personas, la tecnología y las herramientas. Este artículo explora cómo los métodos de diseño y análisis centrados en la arquitectura pueden integrarse con el marco de programación extrema (XP), que es una de las prácticas ágiles más maduras. La sinergia entre estos enfoques puede dar lugar a soluciones de software efectivas y de alta calidad.

**DIAGRAMA :**

****

**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.

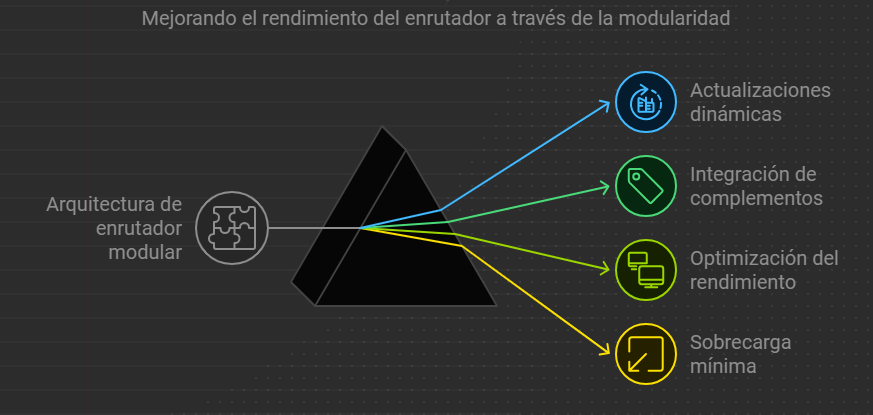
keywords: {Computer architecture;Software architecture;Design methodology;Software engineering;Process design;Risk analysis;Feedback;Prototypes;Programming;Design engineering;software architecture;architecture-centric methods;quality attributes;agile software development;Extreme Programming;XP},

**ARTÍCULO 30:** Plugins de enrutador: una arquitectura de software para enrutadores de próxima generación.

**RESUMEN :** 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. En comparación con un núcleo monolítico, nuestra implementación solo aumenta la sobrecarga en un 8% y puede reenviar paquetes.

**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. Además, el diseño modular optimiza el rendimiento al emplear un algoritmo eficiente de clasificación de paquetes y almacenamiento en caché, lo que aprovecha mejor las características del tráfico de Internet. Comparado con un núcleo monolítico, la implementación modular aumenta la sobrecarga solo un 8%, lo que permite reenviar paquetes hasta tres veces más rápido. Este enfoque demuestra cómo la modularidad no solo facilita la actualización, sino que también mejora el rendimiento y la eficiencia en el procesamiento de paquetes, adaptándose mejor a las demandas actuales.

**DIAGRAMA :**

****

**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.

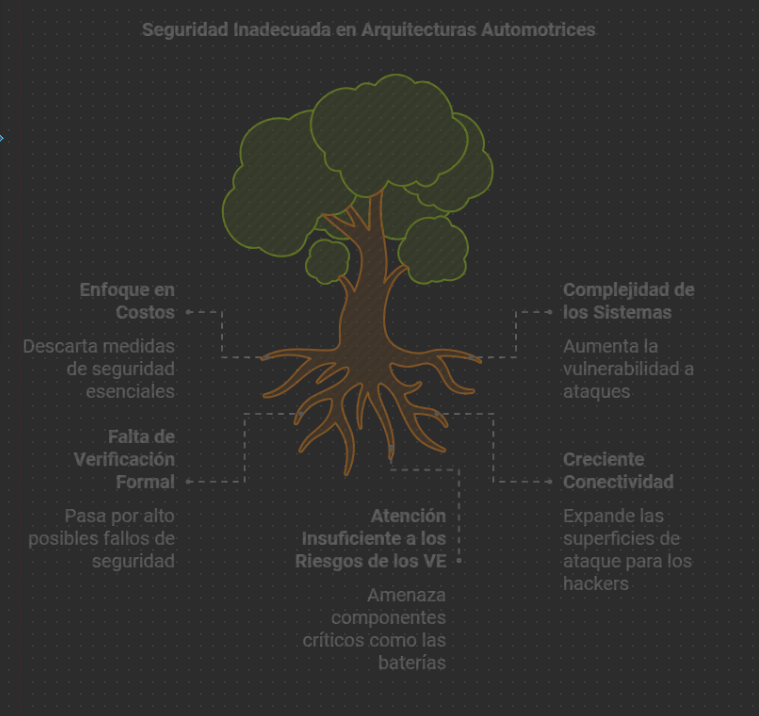
keywords: {Software architecture;Routing;Kernel;Electronic mail;Internet;Protocols;Laboratories;Paramagnetic resonance;Operating systems;Computer architecture},

**ARTÍCULO 31 :** Desafíos de seguridad en el diseño de arquitectura de hardware y software automotriz.

**RESUMEN :** 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. Finalmente, se analiza cómo las futuras arquitecturas automotrices basadas en Ethernet/IP podrían beneficiarse de métodos de verificación formal para mejorar la seguridad.

**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. A pesar de los riesgos, el artículo sugiere que las futuras arquitecturas automotrices basadas en Ethernet/IP podrían incorporar métodos de verificación formal, lo que podría ser un avance significativo para mí.

**DIAGRAMA:**

****

**BIBLIOGRAFÍA :**F. Sagstetter 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. keywords: {Security;Automotive engineering;Vehicles;Computer architecture;Batteries;Wireless communication;Plugs},

**ARTÍCULO 32:** Visualización de la arquitectura de software: un marco de evaluación y su aplicación

**RESUMEN :** 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. Las partes interesadas pueden aplicar el marco para determinar si una herramienta de visualización de la arquitectura de software en particular es adecuada para una tarea determinada. El marco se aplica en la evaluación de una colección de seis herramientas de visualización de la arquitectura de software. El marco también puede utilizarse como plantilla de diseño para una herramienta integral de visualización de la arquitectura de software.

**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. Además, se presenta un marco cualitativo para evaluar herramientas de visualización de la arquitectura de software, lo que permite a las partes elegir satisfactoriamente las más adecuadas según sus necesidades. Este marco, basado en siete áreas clave y 31 características, también puede servir de base para el diseño de herramientas de visualización más completas. Por último, el artículo sugiere que las arquitecturas basadas en Ethernet/IP se beneficiarían de la implementación de métodos de verificación formal para mí.

**DIAGRAMA :**

****

**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.

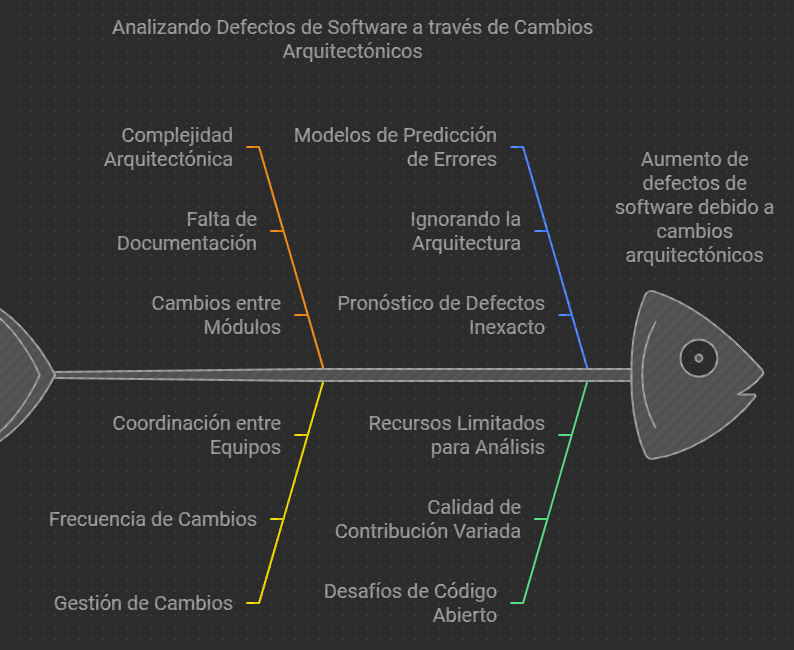
keywords: {Software architecture;Visualization;Application software;Computer architecture;Guidelines;Software systems;Computer Society;Performance evaluation;Navigation;Computer science;Software Architectures;Visualization techniques and methodologies;Software Architectures;Visualization techniques and methodologies},

**ARTÍCULO 33:** Un estudio sobre el papel de la arquitectura del software en la evolución y la calidad del software.

**RESUMEN :** 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. Los resultados muestran que los cambios que cruzan los límites de los módulos son más propensos a generar defectos, lo que sugiere que los predictores de errores deben tener en cuenta la arquitectura del software.

**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. Los resultados sugieren que los predictores de errores, que normalmente no consideran la arquitectura, deberían incorporar esta variable para mejorar la precisión y la confiabilidad de las predicciones. Esto abre nuevas perspectivas sobre cómo abordar la calidad del software, destacando la importancia de tener en cuenta la estructura interna del sistema al evaluar posibles defectos. De este modo, se refuerza la idea de que la arquitectura no solo es clave para el diseño, sino también para mantener la estabilidad y reducir los errores a largo plazo.

**DIAGRAMA :**

****

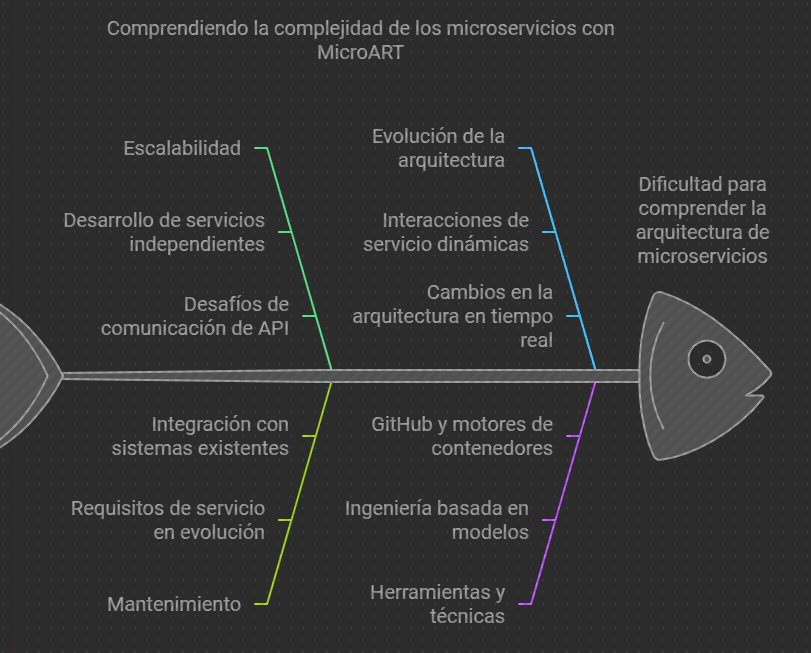
**BIBLIOGRAFÍA :** E. Kouroshfar, M. Mirakhorli, H. Bagheri, L. Xiao, S. Malek and Y. Cai, "A Study on the Role of Software Architecture in the Evolution and Quality of Software," 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, Florence, Italy, 2015, pp. 246-257, doi: 10.1109/MSR.2015.30. keywords: {Computer architecture;Measurement;Couplings;Software systems;Data mining;History;Software Repositories;Software Architecture;Defects},

**ARTÍCULO 34 :** Hacia la recuperación de la arquitectura de software de los sistemas basados ​​en microservicios.

**RESUMEN :**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. MicroART extrae automáticamente modelos de arquitectura usando un repositorio de GitHub y un motor de contenedores. La validación en un sistema de referencia.

**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. Al extraer automáticamente modelos de arquitectura mediante herramientas como GitHub y motores de contenedores, MicroART facilita la comprensión y gestión de estos sistemas complejos, demostrando ser una herramienta valiosa, como lo muestra su validación en un sistema de referencia.

**DIAGRAMA :**

****

**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," 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, Sweden, 2017, pp. 46-53, doi: 10.1109/ICSAW.2017.48. keywords: {Computer architecture;Containers;Data mining;Software;Reverse engineering;Software architecture;Ports (Computers);Microservices;Architecture recovery;Model-Driven Engineering},

**ARTÍCULO 35 :** La vista de la arquitectura del software en tiempo de compilación

**RESUMEN :** 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. A través de tres estudios de caso, se muestra cómo el modelado de estas vistas en tiempo de compilación puede mejorar la comprensión del sistema y su desarrollo. Finalmente, se introduce un nuevo estilo arquitectónico, el "robot de código", que se encuentra combinado en sistemas que tienen vistas de arquitecto.

**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. Además, la inclusión del concepto de "robot de código" abre nuevas posibilidades para diseñar sistemas más eficientes y adaptativos, combinando vistas arquitectónicas con enfoques innovadores.

**DIAGRAMA :**

****

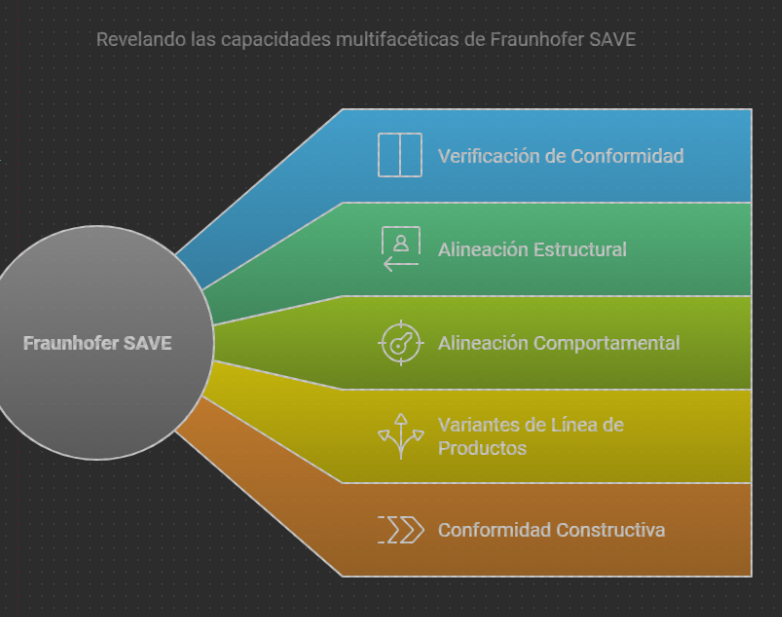
**BIBLIOGRAFÍA:**Q. Tu and M. W. Godfrey, "The build-time software architecture view," Proceedings IEEE International Conference on Software Maintenance. ICSM 2001, Florence, Italy, 2001, pp. 398-407, doi: 10.1109/ICSM.2001.972753. keywords: {Software architecture;Taxonomy;Software systems;Computer science;Application software;Concurrent computing;Computer industry;Java;Robots;Mathematical model},

**ARTÍCULO 36:** SAVE: Visualización y evaluación de la arquitectura de software

**RESUMEN :** 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. En particular, mostramos cómo funcionan las características de verificación de conformidad de SAVE para asegurar la conformidad con las vistas arquitectónicas estructurales y de comportamiento, para asegurar la conformidad entre variantes en un contexto de línea de productos e incorporar la verificación de conformidad constructiva para permitir la conformidad incorporada durante el desarrollo y la evolución.

**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. Esto no solo fortalece la calidad del sistema, sino que también optimiza su mantenimiento y adaptabilidad futura.

**DIAGRAMA :**

****

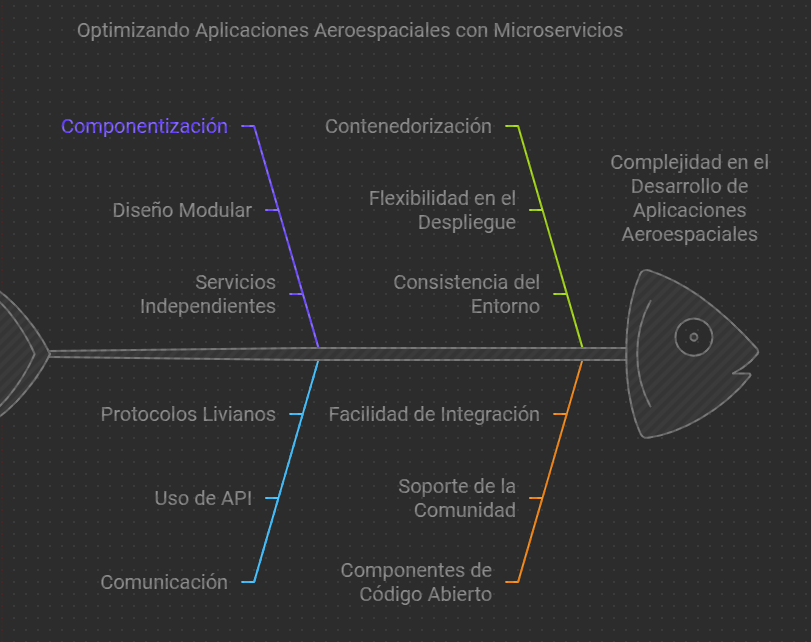
**BIBLIOGRAFÍA :** S. Duszynski, J. Knodel and M. Lindvall, "SAVE: Software Architecture Visualization and Evaluation," 2009 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany, 2009, pp. 323-324, doi: 10.1109/CSMR.2009.52. keywords: {Software architecture;Visualization;Software engineering;Computer architecture;Feedback;Software maintenance;Educational institutions;Software systems;Reverse engineering;Runtime;software architecture;visualization;reverse engineering;compliance checking;SAVE},

**ARTÍCULO 37:** Arquitectura y enfoques de software basados ​​en microservicios

**RESUMEN :** 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. Este documento explora características clave como la componenteización, organización, comunicación entre servicios, contenedorización, componentes de código abierto y casos de uso práctico aplicado.

**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. Este enfoque también aprovecha herramientas como la contenedorización y componentes de código abierto, promoviendo una estructura modular y organizada para atender casos de nosotros.

**DIAGRAMA:**

****

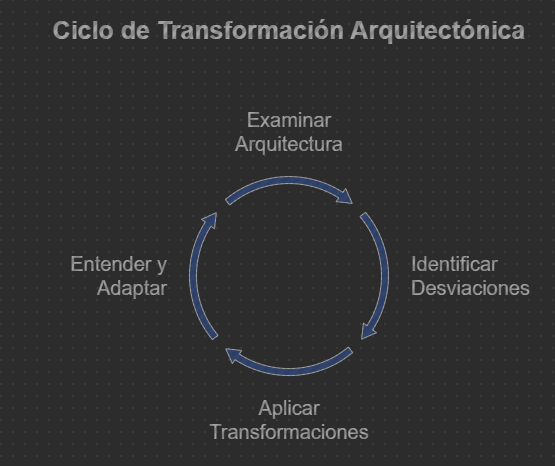
**BIBLIOGRAFÍA :** K. Bakshi, "Microservices-based software architecture and approaches," 2017 IEEE Aerospace Conference, Big Sky, MT, USA, 2017, pp. 1-8, doi: 10.1109/AERO.2017.7943959. keywords: {Computer architecture;Software;Databases;Production;Tools;Organizations},

**ARTÍCULO 38:** Transformaciones de la arquitectura de software

**RESUMEN :** 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. Clasificamos las transformaciones arquitectónicas útiles y las describimos dentro del marco de las transformaciones gráficas. Al describirlas de forma unificada, obtenemos una mejor comprensión de las transformaciones y, por lo tanto, podemos trabajar para modelarlas, especificarlas y automatizarlas.

**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.

**DIAGRAMA :**

****

**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},