1. **Resumen de Anyi zujey ARTÍCULO 11:** 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.

1. **Resumen de Aura maría fierro:** Hoy vamos a hablar sobre un enfoque clave en la ingeniería de software que ayuda a crear sistemas más eficientes, sostenibles y alineados con los objetivos de negocio. Este enfoque se llama Diseño Guiado por el Dominio o DDD. Y bueno ahora y que es eso de Diseño guiado por el dominio) es como una metodología de desarrollo de software que pone el foco principal en el dominio del negocio. Es decir, lo que importa es entender cómo funciona el negocio y reflejar eso en el sistema de software y tiene como objetivo crear modelos de negocio precisos que se integren directamente con el código. Esto también ayuda a simplificar la complejidad técnica, pues en estos encontramos dos arquitecturas, arquitectura limpia busca mantener el código organizado y arquitectura hexagonal, por otro lado, es una técnica que busca desacoplar la lógica de negocio del sistema.
2. **Resumen de Camilo Andrés bautista:** 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.
3. **Resumen de Carolina Martínez Cortes:** 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.
4. **Resumen de Cristian Fernando Narváez: Articulo 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

1. **Resumen de Cristian Jean pool:** 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.
2. **Resumen de John Sebastian Penna Arias:** La arquitectura hexagonal, también conocida como arquitectura de puertos y adaptadores, es un patrón de diseño de software que promueve la separación de responsabilidades en una aplicación. facilita el desarrollo, mantenimiento y prueba de sistemas. Su idea principal es dividir el sistema en una capa central que contiene la lógica de negocio y varias "interfaces" externas que se comunican con ella a través de puertos y adaptadores.
3. **Resumen de Jose Manuel Gasca: Articulo 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.
4. **Resumen de Julián David Fierro: Articulo 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 Observar 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.
5. **Resumen de Laura Ariza:** **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.
6. **Resumen de Maria del Mar Artunduaga: 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.
7. **Resumen de Maria jose Murcia:** El artículo habla sobre cómo el proyecto STATUS mejora la usabilidad del software desde el principio, en lugar de esperar hasta el final para corregir problemas. Normalmente, la usabilidad se evalúa cuando el sistema ya está terminado, pero STATUS propone incluir opciones útiles como "deshacer", "cancelar" y soporte para varios idiomas desde la etapa de diseño. Esto hace que el software sea más fácil de usar y evita retrabajos costosos. Además, evalúan la usabilidad en cada paso del proceso, haciendo ajustes cuando es necesario. Este enfoque ayuda a los desarrolladores a crear un sistema que sea funcional y agradable para los usuarios desde el inicio.
8. **Resumen de Mariana Charry:** Artículo 18:

Está siendo de gran impacto la evolución de los sistemas operativos, porque cada vez son más estables y robustos, las aplicaciones móviles ahora son de mayor tamaño y complejidad, lo cual permite a los desarrolladores crearlas. Se definió una arquitectura móvil, la cual comparte algunos de los principios más reconocidos de la arquitectura de software. Porque de esa forma permite ayudar a estandarizar y adaptar metodologías, métodos, procesos y enfoques. Entonces 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. Para lo cual, dicho 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.

1. **Resumen de Mariana González: 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.
2. **Resumen de Maydy Viviana Conde:** **Semifallo:** Un singlón 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 puedas 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
3. **Resumen de Maira Tamayo 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.
4. **Resumen de Patricia Sarmiento 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.
5. **Resumen de Valentina Silva:** **Análisis comparativo de Patrones de Diseño de Software:** El artículo analiza varios patrones populares como MVC, MVP, Front Controller, Template Method y MVVM. Se comparan estos patrones en base a su escalabilidad, facilidad de implementación, complejidad, uso, seguridad y compatibilidad con diferentes lenguajes de programación.
6. **Resumen de William cortes: Buenas prácticas en la construcción del software:** Este artículo de reflexión explota diferentes arquitecturas de software y metodologías de desarrollo utilizadas para crear soluciones eficientes 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 dé tiene patrones y tecno logias 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.
7. **Resumen de Yordy Núñez: Introducción a los Patrones de Diseño:** 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.