INGENIERÍA DE SOFTWARE 2: UNIDAD 1 TAREA EN GRUPO O INDIVIDUAL- DISEÑO DE SOFTWARE- PRIMER PARCIAL

UNIREMINGTON®
CORPORACIÓN UNIVERSITARIA REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996

Brayhan Stiven Orrego Valencia, Jimmis J Simanca
UNIREMINGTON Ingeniería de Software 2

Trabajo investigativo

1 Realice un mapa mental sobre lo que es el diseño de software, tenga en cuenta: características, tiempos relevantes, métodos que apoyan un buen diseño, entre otros. Usar la aplicación de miro:

https://miro.com/welcomeonboard/N2p6cldpV05RSGRSa0d5UjVOaDJPb3dmeFBpbTAxYWxHY3NpVDFCeUdLMjVDMGxOd0hCV3BMSG4zZU9RVWNTV3wzNDU4NzY0NTk4MTM1NjE2MzUwfDI=?share_link_id=310191795019

2. ¿Cuáles son las características o requisitos que distinguen un buen diseño? Presente un mapa mental con las características.

https://miro.com/welcomeonboard/MWc3Z2JaNHJiZVBKTlczMjNIRzA2VlNWN1Z5ZWtnbEloM3RPN2NFcXdpQVYyTmhsc1FUY3NYME4zcFNiaVd2NnwzNDU4NzY0NTk5MTAyNDY1NzY0fDI=?share_link_id=258701610360

3. ¿Qué es DDD? ¿Cómo se aplica en un proyecto de software?

DDD, o Domain Driven Design, es una metodología de desarrollo de software que implica modificar el software para adaptarlo al dominio específico de la organización o empresa en la que se crea. En 2003, El término fue acuñado por Eric Evans en su libro "Domain-Driven Design: Tackling Complexity in the Heart of Software", publicado en 2003.

DDD es particularmente útil en proyectos de software complejos que requieren un conocimiento profundo del dominio.

DDD principios fundamentales:

El dominio abarca el área de especialización y trabajo que realiza la empresa. El software es necesario para abordar las necesidades específicas del negocio o contexto.

El modelo de dominio es una representación conceptual de los conceptos y relaciones dentro de un dominio. Los desarrolladores y expertos en el dominio, normalmente partes interesadas del negocio deben comprender y perfeccionar el modelo que es fundamental para DDD.

El lenguaje Ubicuo es compartido por los desarrolladores y los expertos del dominio. Este lenguaje evita malentendidos y asegura que todos estén alineados con los términos y conceptos usados en el desarrollo.

Entidades y objetos de valor:

Entidades: Objetos que tienen una identidad única, que persiste a lo largo del tiempo y a través de diferentes estados.

Objetos de Valor: Objetos que no tienen identidad propia, sino que se identifican por sus atributos y pueden ser intercambiados.

Repositorio: Un patrón que proporciona acceso a las entidades. Es una abstracción sobre la capa de persistencia.

Servicios de Dominio: Operaciones que no pertenecen naturalmente a una entidad o un objeto de valor, pero son relevantes en el dominio.

Eventos de Dominio: Eventos que representan algo que ha ocurrido en el dominio que es relevante para el negocio.

Aplicación de DDD en un proyecto de software.

Comprensión del Dominio: Trabajar estrechamente con los expertos en el dominio para entender profundamente el negocio y los problemas que se quieren resolver. Crear un modelo de dominio que refleje este conocimiento.

Diseño del Modelo: Basándose en el conocimiento del dominio, diseñar un modelo de software que capture los conceptos clave. Este modelo debe estar alineado con el lenguaje ubicuo y ser validado constantemente con los expertos del dominio.

Definición de Bounded Contexts: Dividir el sistema en diferentes "bounded contexts" (contextos delimitados), cada uno con su propio modelo de dominio, lo que facilita la gestión de la complejidad. Dentro de un bounded context, los términos y conceptos son consistentes.

Implementación: Utilizar el modelo de dominio como base para la implementación, manteniendo una alineación estrecha entre el código y el modelo conceptual. Usar patrones como agregados, servicios de dominio, repositorios, y eventos de dominio para estructurar la implementación.

Iteración y Refinamiento: El modelo de dominio no es estático; debe evolucionar a medida que se descubre más sobre el dominio y cambian los requisitos del negocio. Esto implica una comunicación constante entre los desarrolladores y los expertos en el dominio.

4. ¿Qué es un patrón de diseño?

Un patrón de diseño es una técnica desarrollada para resolver problemas recurrentes en la programación, basándose en soluciones que han demostrado ser efectivas a lo largo del tiempo. Estos patrones aportan escalabilidad, abstracción y facilidad de mantenimiento, haciendo que el código sea más comprensible para otros desarrolladores que trabajen en el proyecto. Aunque los patrones de diseño no son algoritmos complejos, funcionan como un plano que guía cómo abordar un problema específico, sin dictar el orden exacto de las acciones a seguir. Al entender y aplicar estos patrones, puedes resolver problemas similares de manera más eficiente.

Los patrones de diseño se clasifican según su propósito y complejidad en tres grupos esenciales:

patrones creacionales, que se centran en la manera en que se crean los objetos; **patrones estructurales**, que tratan con la composición y organización de clases y objetos; y **patrones de comportamiento**, que abordan la interacción y responsabilidad entre objetos. Esta clasificación ayuda a seleccionar el patrón adecuado según el tipo de problema que enfrentas y facilita su aplicación en el desarrollo de software. (REFACTORIN GURU)

¿Cuándo surgieron?

Según (REFACTORING GURU).Los patrones de diseño surgieron inicialmente a partir del trabajo de Christopher Alexander, quien los describió en su libro "**El lenguaje de patrones**". Alexander utilizó estos patrones para crear entornos urbanos, especificando detalles como la altura de las ventanas, los niveles de los edificios y las zonas verdes en los barrios, entre otros aspectos.

Posteriormente, la idea de los patrones fue adaptada al campo de la ingeniería de software por Erich Gamma, John Vlissides, Ralph Johnson y Richard Helm, quienes en 1995 publicaron el libro "**Patrones de diseño**". En esta obra, presentaron 23 patrones de diseño que aplicaban los conceptos desarrollados por Alexander para resolver diversos problemas en el desarrollo de software. A lo largo del tiempo, se han identificado muchos más patrones de diseño, lo que ha contribuido a la popularización y expansión de esta metodología. (REFACTORING GURU)

¿Por qué surgieron?

Desde mi punto de vista, los patrones de diseño surgieron debido a la necesidad de estandarizar soluciones para problemas recurrentes en el desarrollo de software. A medida que se trabajaba en proyectos, se observaba que se utilizaban patrones similares, ya sean lógicos, de código o abstractos, para resolver problemas similares en diferentes casos. Esto llevó a la idea de crear soluciones abstractas que pudieran aplicarse en diversos contextos.

En esencia, un patrón de diseño muestra cómo resolver un problema específico de manera efectiva. La aplicación de estos patrones, que ya han sido probados y son eficaces, no solo facilita la resolución de problemas recurrentes, sino que también puede acelerar el tiempo de desarrollo de un proyecto al evitar reinventar la rueda cada vez. Por lo tanto, los patrones de diseño surgieron como una respuesta a la necesidad de soluciones reutilizables y comprobadas para problemas comunes en el desarrollo de software. J. J. Simanca (Comunicación personal, 08, de septiembre, 2024)

5. Mencione algunas técnicas de la evaluación y análisis de la calidad de un diseño de software.

¿Cuáles son los tipos de pruebas de software?

Los tipos más comunes de pruebas de software incluyen:

- Análisis estático
- Prueba unitaria
- Pruebas de integración
- Prueba del sistema
- Test de aceptación

A continuación, se mencionan algunas técnicas de evaluación de diseño de software:

- Análisis estático: Se analizan las estructuras del diseño sin ejecutar el código, utilizando herramientas automáticas para revisar la calidad, complejidad y consistencia del diseño.
- Revisión de pares: Un equipo de expertos revisa el diseño de software para detectar errores y sugerir mejoras.
- Análisis de requisitos: Verifica que el diseño de software cumpla con los requisitos del usuario.
- Simulaciones: Se simulan escenarios de uso o cargas para evaluar el comportamiento del diseño bajo diferentes condiciones operativas.
- Pruebas de diseño (Design Testing): Se realizan pruebas en partes del diseño antes de la implementación completa, para garantizar que cumple con los objetivos de rendimiento, usabilidad, escalabilidad, etc.

- Análisis de seguridad: Evalúa la seguridad del software para detectar vulnerabilidades.
- Pruebas de integración: Evalúa cómo los diferentes componentes del software interactúan entre sí.
- Prototipado: Crear versiones simplificadas o funcionales del sistema para validar las decisiones de diseño y evaluar cómo se comporta en escenarios reales.

Estas técnicas ayudan a garantizar que el diseño de software sea de alta calidad, cumpla con los requisitos del usuario y sea sostenible a lo largo del tiempo.

· Análisis de la calidad de un diseño de software.

La calidad del software es un elemento esencial que determina el éxito o fracaso de un proyecto de desarrollo de software. Un software de alta calidad se caracteriza por ser confiable, fácil de usar y sostenible a lo largo del tiempo, lo que significa que puede cumplir con los requisitos del usuario de manera efectiva y eficiente. Por otro lado, un software de baja calidad puede presentar problemas de confiabilidad, usabilidad y sostenibilidad, lo que puede generar retrasos y costos adicionales en el proyecto, además de afectar negativamente la satisfacción del usuario.

Es fundamental evaluar la calidad del software de manera adecuada para garantizar que cumpla con los requisitos del usuario y sea de alta calidad. Esto implica realizar pruebas y validaciones exhaustivas para detectar y corregir errores, así como asegurarse de que el software sea fácil de usar, escalable y sostenible. Algunos de los factores que contribuyen a la calidad del software incluyen:

- La claridad y comprensión de los requisitos del usuario

- La experiencia y habilidades del equipo de desarrollo

- La utilización de metodologías y herramientas adecuadas

- La realización de pruebas y validaciones exhaustivas

- La documentación y mantenimiento adecuados.

6.-Realice un diagrama de procesos para especificar la implementación de alguna de las técnicas:

https://lucid.app/lucidchart/4f826b9f-d2f4-4eec-a277-7cc5a5b3c91c/edit?viewport_loc=112%2C79%2C2601%2C1451%2C0_0&invitationId=inv_ad83b2a6-c6d7-44cf-bf13-7f12319150bd

7.-Los diferentes diagramas usados para el diseño, describen y representan los aspectos estructurales de un diseño de software, es decir, se utilizan para describir los componentes principales y cómo están interconectados (vista estática). ¿Cuáles son esos diagramas? mencionar como mínimo 5 diagramas con sus respectivas características y presentar un ejemplo de cada uno. No pegar ejemplos de internet sobre los diagramas y que estén en inglés. Los diagramas deben ser elaborados por el equipo de trabajo en español y a partir del siguiente tema: óptica, este punto se calificará solo si se trabaja sobre el tema dado:

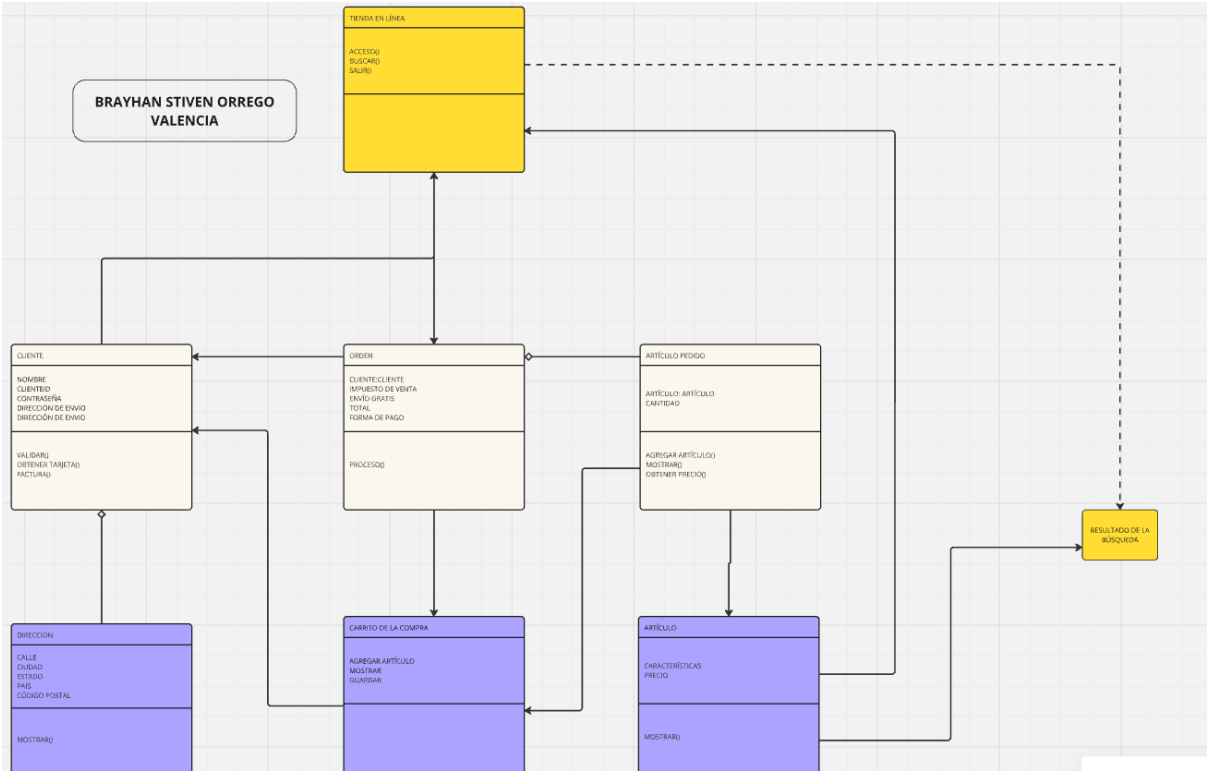
¿Cuáles son esos diagramas?

A las personas que no cuentan con experiencia les puede parecer que hay un número infinito de diagramas, pero en realidad, los estándares de UML identifican 13 tipos de diagramas que se dividen en dos grupos, definidos a continuación:

- **Diagramas UML estructurales**

Los diagramas UML estructurales, como su nombre lo indica, muestran cómo está estructurado el sistema, incluyendo las clases, objetos, paquetes, componentes, etc. del sistema y las relaciones entre esos elementos.

Ejemplo:



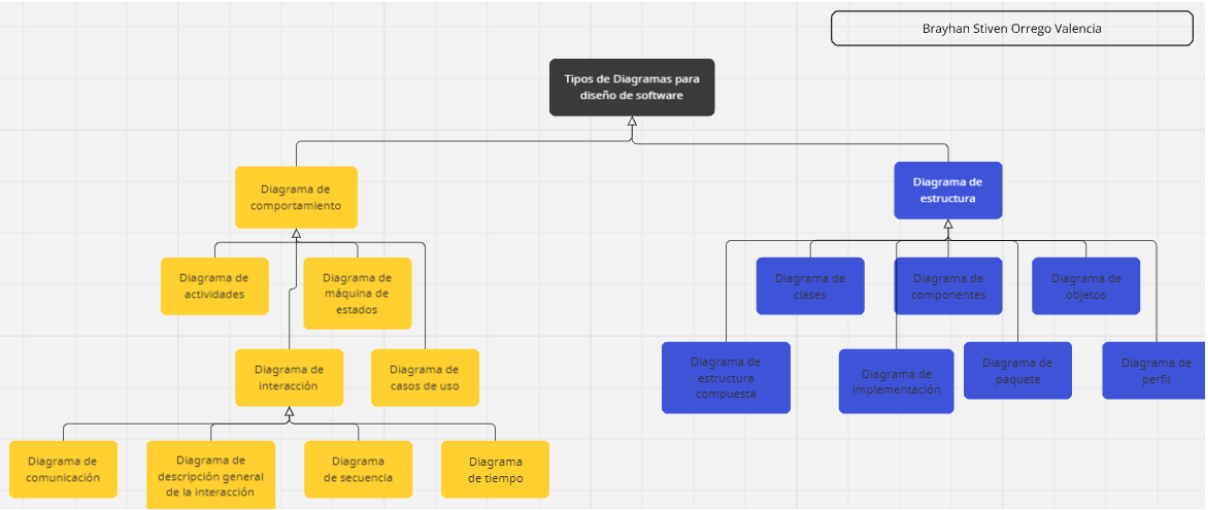
https://miro.com/welcomeonboard/aTZnaExnemJ5Zjg0MXpKaWNFNmNUaHA2RUZEbnpZRVNxRVJqMHZxZmpxWnZsdlJtbG5jRDlsUGNPbVJ6czBHdnwzNDU4NzY0NTIwODU1MjYxMzM2fDI=?share_link_id=274178604355

- **Diagrama de clases**

Dado que gran parte del software se basa en la programación orientada a objetos, en la que los desarrolladores definen tipos de funciones que se pueden utilizar, los diagramas de clases son el tipo de diagrama UML más comúnmente utilizado. Los diagramas de clases muestran la estructura estática de un sistema, incluyendo las clases, sus atributos y comportamientos, y las relaciones entre cada clase.

Una clase está representada por un rectángulo que contiene tres compartimientos apilados verticalmente: el compartimiento superior contiene el nombre de la clase y es obligatorio, pero los dos compartimientos inferiores muestran detalles sobre los atributos y las operaciones o comportamientos de la clase.

Ejemplo:



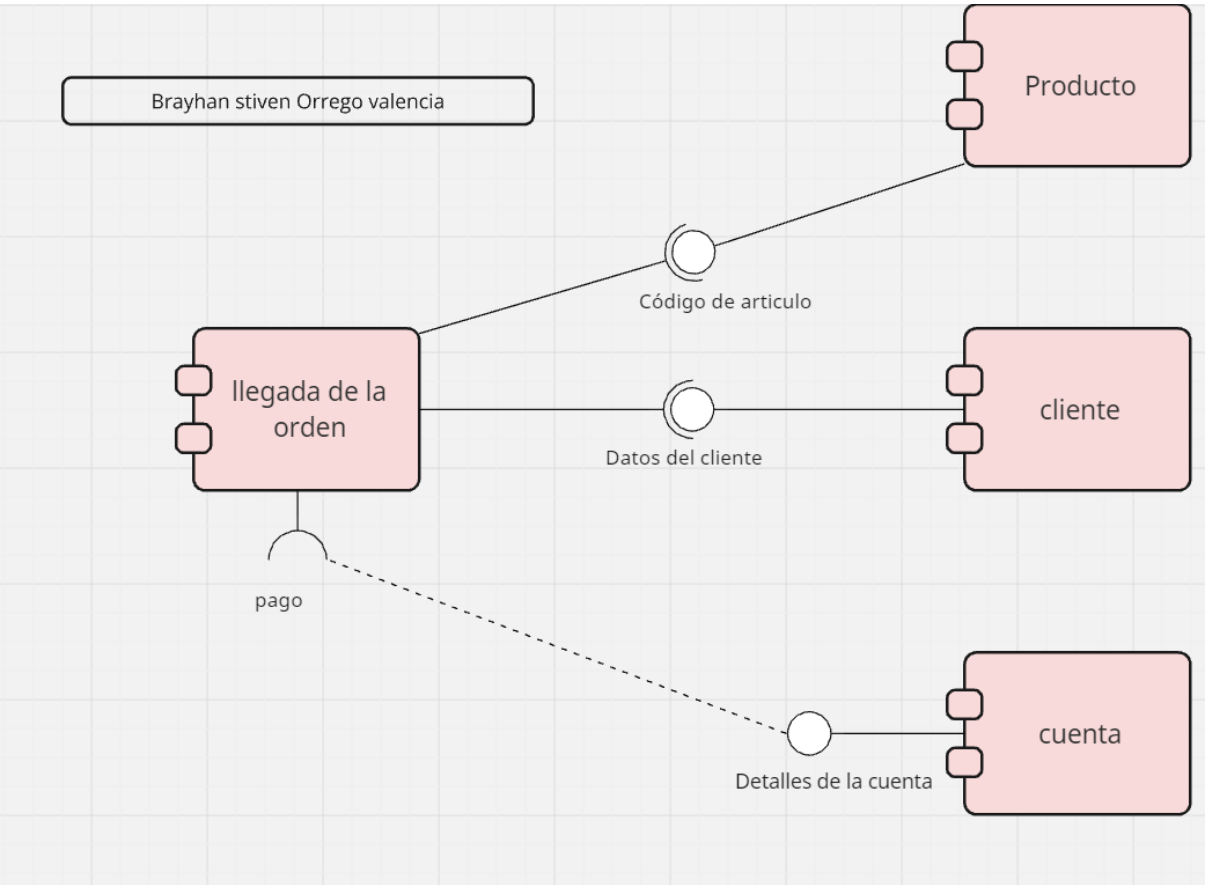
https://miro.com/welcomeonboard/cW1WbUxsVEMzMHlmYkRPVjhWazJ3d0t0akdlYUM4cngwcE5WOEhWajZWdHJ4MWJvUUV6RFhSZ1Vwd3k3S1ZQcHwzNDU4NzY0NTIwODU1MjYxMzM2fDI=?share_link_id=712444923987

- **Diagrama de componentes**

Un diagrama de componentes es esencialmente una versión más especializada del diagrama de clases: se aplican las mismas reglas de notación para ambos. Un diagrama de componentes descompone un sistema complejo en componentes más pequeños y visualiza la relación entre esos componentes.

Ejemplo:

https://miro.com/welcomeonboard/WlBrblU4RmJRcjF6OGV0YVFEQTFQcmJXaGg0Vm05dGwzYXBtRzkxSmlsWUxKTDByMEFjeU9QcTIwaFptMlZ2dHwzNDU4NzY0NTIwODU1MjYxMzM2fDI=?share_link_id=457880594196

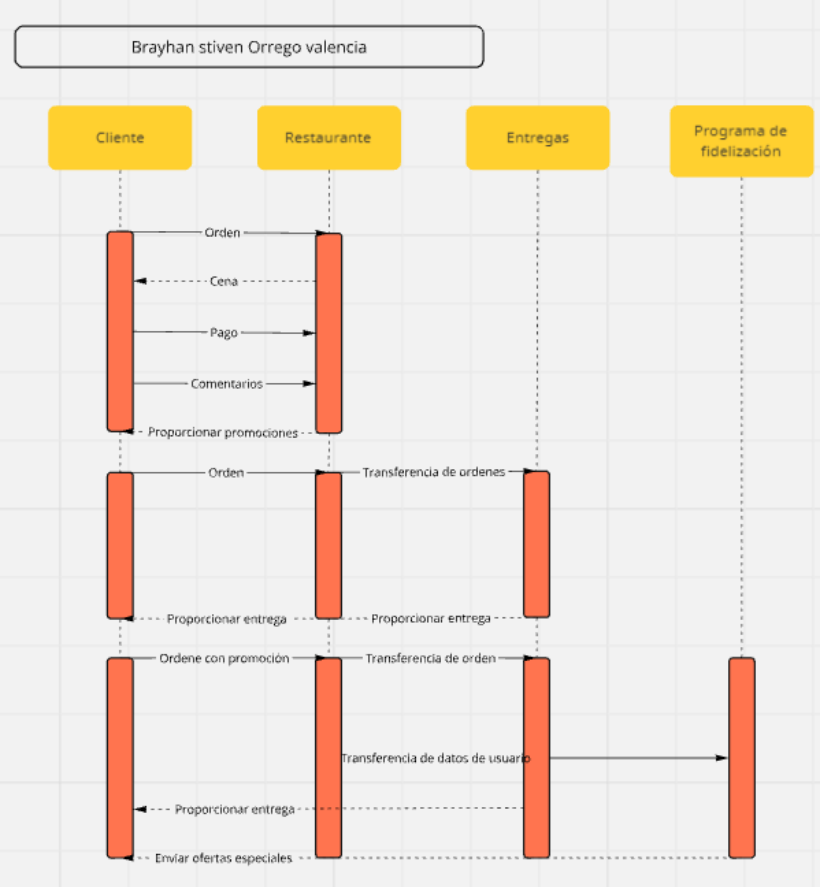

Brayhan stiven Orrego valencia
Producto
Código de articulo
llegada de la orden
cliente
Datos del cliente
pago
cuenta
Detalles de la cuenta

- **Diagrama de Secuencia**

Los diagramas de secuencia siguen la notación UML estandarizada para representar las interacciones en un comportamiento de sistema específico. Pueden usarse para mejorar la comprensión de interacciones de software complejas, potenciar la eficiencia del diseño y mejorar la comunicación dentro de un proyecto.

Ejemplo:

https://miro.com/welcomeonboard/aDYyQ2NzWHVtNlA3Q0lXQXE2MVBHVWZ4cDZ6UjJkMUdxdnJteXgzSlJXcXBQNEV3VGkzdlRhTFRrZXNvNWVQR3wzNDU4NzY0NTIwODU1MjYxMzM2fDI=?share_link_id=137974072608

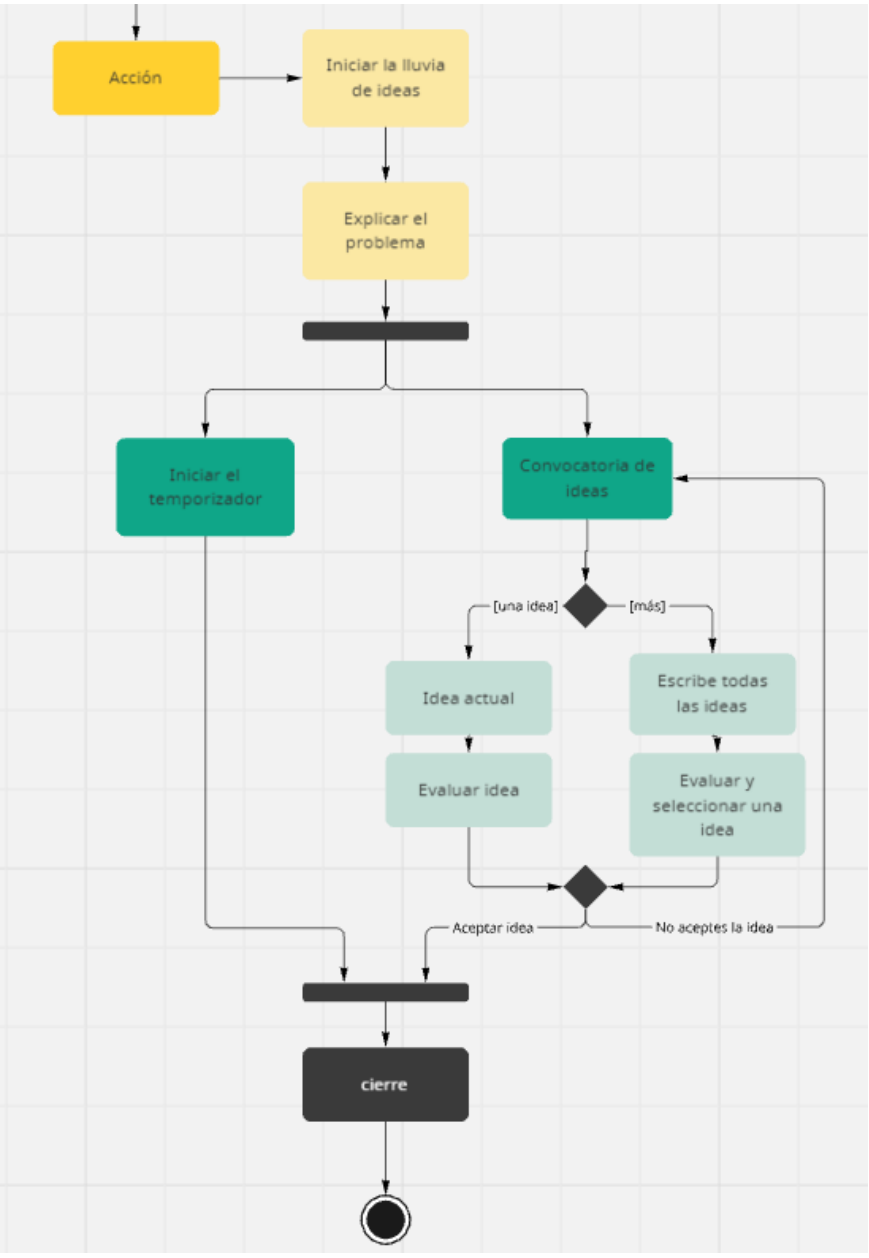


- **Diagrama de actividades UML**

Un diagrama de actividades es una representación visual de acciones, restricciones, requisitos y otros factores que intervienen en la realización de tareas. Puedes pensar en ellos como un diagrama de flujo detallado, mostrando cada paso y punto de decisión en un proceso. Mapear actividades de esta manera puede revelar nueva información, ayudar a identificar ineficiencias y proporcionar otros beneficios importantes.

Ejemplo:

https://miro.com/welcomeonboard/a3VjcHo3cWZPWFNMRTlQMFp2VlJrMEE2RVNReDd2NExsVVlOVmNXc0pTR1pSVE82UGpUMTFwMnNoaDRVamt4Y3wzNDU4NzY0NTIwODU1MjYxMzM2fDI=?share_link_id=172117408949



Referencia, bibliografías, citas y enlaces que fueron de apoyo para la realización de esta actividad:

· DDD: ¿Qué es y cómo se aplica? (2021, June 23). TECNOVA. https://www.tecnova.cl/2021/06/23/ddd-domain-driven-design/

· Domain Driven Design. (n.d.). Modyo.com. Retrieved September 6, 2024, from https://docs.modyo.com/es/architecture/patterns/ddd.html

· Dominguez, J. (2022, September 27). Domain-Driven Design, ¿Qué es y cómo aplicarlo en mi organización? Itequia. https://itequia.com/es/domain-driven-design-que-es-y-como-aplicarlo-en-mi-organizacion/

Camacho, R. (29 de Mayo de 2024). *PARASOFT*. Obtenido de https://es.parasoft.com/blog/software-testing-methodologies-guide-a-high-level-overview/#What_Are_the_Types_of_Software_Testing

Castro, David. (17 de Enero de 2023). *Medium*. Obtenido de Medium: https://davidcasr.medium.com/c%C3%B3mo-evaluar-la-calidad-de-un-software-10aa923f63aa

hdeleon.net (2022, agosto 11). *¿Qué son los PATRONES de DISEÑO?*

. [Vídeo]. ¿Qué son los PATRONES de DISEÑO? (youtube.com)

REFACTORIN GURU. (s.f.). *Patrones.* Abolmasova 7 Kyiv, Ukraine, 02002: FOP Olga Skobeleva.

Interactive Refactoring Course: Dive Into Refactoring

REFACTORING GURU. (s.f.). *Historia de los patrones.* Abolmasova 7 Kyiv, Ukraine, 02002: FOP Olga Skobeleva.

Historia de los patrones (refactoring.guru)

https://www.linkedin.com/advice/0/what-most-effective-types-software-design-nn6me?lang=es&originalSubdomain=es

https://www.lucidchart.com/blog/es/tipos-de-diagramas-uml

https://www.edrawsoft.com/es/types-software-diagrams.html

Piñeres, M. F. C., Miranda, R. E. T., Rozo, F. M. H., & Lobo, M. E. D. (2009). Diseño de software educativo basado en competencias. *Ciencia E Ingeniería Neogranadina*, *19*(1), 71-98. https://doi.org/10.18359/rcin.311