Enumere y describa los pasos implicados dentro de la técnica del artículo “De los Procesos de Negocio a los Casos de Uso” (autores: Jesús García Molina, M. José Ortín, Begoña Moros, Joaquín Nicolás, Ambrosio Toval) de la Universidad de Murcia.

Lamentablemente, no tengo acceso al artículo específico que mencionas, ya que mi conocimiento se limita a la información recopilada hasta septiembre de 2021 y no tengo la capacidad de navegar por Internet en tiempo real. Sin embargo, puedo proporcionarte una descripción general de la técnica comúnmente utilizada para derivar casos de uso a partir de procesos de negocio. Ten en cuenta que esta descripción no estará basada directamente en el artículo mencionado, sino en una metodología general utilizada para tal fin.

La técnica de "De los Procesos de Negocio a los Casos de Uso" generalmente implica los siguientes pasos:

1. Identificar los procesos de negocio: Comienza identificando y documentando los procesos de negocio relevantes para el sistema que se está analizando. Estos procesos pueden representar actividades clave dentro de una organización o flujos de trabajo específicos.

2. Modelar los procesos de negocio: Utiliza una notación como BPMN (Business Process Model and Notation) para modelar los procesos de negocio de manera visual. Esto implica la creación de diagramas de flujo que representen las actividades, eventos y decisiones dentro de cada proceso.

3. Identificar actores y roles: Identifica los actores externos que interactúan con los procesos de negocio. Estos actores pueden ser personas, sistemas externos u otras entidades que participan en el sistema. Además, define los roles que desempeñan dentro del contexto de los procesos.

4. Identificar las interacciones entre actores y procesos: Analiza cómo interactúan los actores con los procesos de negocio. Identifica las entradas y salidas de información, así como las interacciones que ocurren a lo largo de cada proceso.

5. Derivar casos de uso: Utilizando la información recopilada en los pasos anteriores, puedes derivar casos de uso a partir de los procesos de negocio. Un caso de uso es una descripción detallada de una interacción específica entre un actor y el sistema. Cada caso de uso debe tener un objetivo claro y describir los pasos necesarios para alcanzar ese objetivo.

6. Refinar y documentar los casos de uso: Una vez derivados, los casos de uso pueden ser refinados y documentados en detalle. Esto implica identificar los pasos específicos que realiza el actor, las interacciones con el sistema y los resultados esperados.

Recuerda que esta descripción general no refleja necesariamente la metodología exacta descrita en el artículo que mencionaste. Si deseas obtener una descripción precisa de los pasos involucrados en dicha técnica, te recomendaría consultar el artículo original o cualquier otra fuente que lo describa en detalle.

Explique para qué se utiliza el patrón Adaptador (Explicación y Diagrama de Clases en PlantUML).

El patrón Adaptador, también conocido como Wrapper, es un patrón de diseño estructural que se utiliza para convertir la interfaz de una clase en otra interfaz esperada por el cliente. Permite que clases con interfaces incompatibles trabajen juntas al envolver la funcionalidad de una clase con otra interfaz.

El patrón Adaptador se aplica en situaciones en las que se necesita integrar una clase existente en un sistema que espera una interfaz diferente. En lugar de modificar directamente la clase existente, se crea un adaptador que actúa como un intermediario entre la clase y el sistema. El adaptador implementa la interfaz esperada por el sistema y delega las llamadas y conversiones necesarias a la clase subyacente.

El patrón Adaptador se compone de los siguientes elementos:

1. Objetivo: Es la interfaz o clase esperada por el sistema o cliente.

2. Cliente: Es el componente que interactúa con el sistema y utiliza la interfaz esperada. El cliente no es consciente de la existencia del adaptador.

3. Adaptador: Es la clase que implementa la interfaz esperada por el cliente y envuelve la clase existente. El adaptador se encarga de traducir las llamadas y convertir los datos para que sean compatibles entre la clase existente y el cliente.

4. Clase existente: Es la clase que ya existe y tiene una interfaz incompatible con la esperada por el cliente. Esta clase se envuelve mediante el adaptador.

A continuación se muestra un diagrama de clases en PlantUML que ilustra la estructura del patrón Adaptador:

```plantuml

En el diagrama, la clase "Objetivo" define la interfaz esperada por el cliente. La clase "ClaseExistente" es la clase con la interfaz incompatible. El adaptador, representado por la clase "Adaptador", implementa la interfaz "Objetivo" y envuelve la clase existente. El cliente interactúa con el adaptador a través de la interfaz esperada.

En resumen, el patrón Adaptador permite que clases con interfaces incompatibles trabajen juntas al proporcionar una interfaz común. Esto facilita la integración de componentes y promueve la reutilización de código existente sin modificarlo directamente.

Explique las siguientes expresiones en OCL (Object Constraint Language): a) Context Proyecto inv: self.participa -> select (edad > 50) -> notEmpty() b) Context Job inv: self.empleado.edad > 21

a) La expresión OCL "Context Proyecto inv: self.participa -> select (edad > 50) -> notEmpty()" se utiliza para definir una restricción (constraint) en el contexto de la clase "Proyecto".

La restricción indica que en el contexto de un objeto "Proyecto", la asociación "participa" debe tener al menos una instancia que cumpla la condición de tener una edad mayor a 50 años.

La expresión utiliza los siguientes operadores OCL:

- El operador "->" se utiliza para encadenar operaciones. En este caso, se aplica después de "self.participa" para realizar las operaciones siguientes en la colección resultante.

- El operador "select" se utiliza para filtrar los elementos de una colección según una condición. En este caso, filtra los elementos de la asociación "participa" donde la propiedad "edad" sea mayor a 50.

- El operador "notEmpty()" se utiliza para verificar que la colección resultante del filtro no esté vacía. En este caso, se asegura de que exista al menos una instancia que cumpla la condición de edad mayor a 50.

En resumen, esta expresión OCL verifica que en un objeto "Proyecto", haya al menos una instancia relacionada a través de la asociación "participa" cuya edad sea mayor a 50.

b) La expresión OCL "Context Job inv: self.empleado.edad > 21" se utiliza para definir una restricción en el contexto de la clase "Job" (trabajo).

La restricción establece que en el contexto de un objeto "Job", la propiedad "edad" del objeto relacionado "empleado" debe ser mayor a 21.

En esta expresión, se utiliza:

- "self" hace referencia al objeto "Job" en el contexto.

- "self.empleado" hace referencia al objeto relacionado "empleado" en la clase "Job".

- "self.empleado.edad" accede a la propiedad "edad" del objeto "empleado" relacionado.

- El operador ">" compara la propiedad "edad" con el valor 21, asegurando que sea mayor.

En resumen, esta expresión OCL establece una restricción que garantiza que la propiedad "edad" del objeto "empleado" relacionado en un objeto "Job" sea mayor a 21.

Explique cómo se realiza (paso a paso) la transformación desde un Diagrama de Clases a un Diagrama Entidad-Relación.

La transformación de un Diagrama de Clases a un Diagrama Entidad-Relación (DER) implica representar las estructuras y relaciones de objetos del Diagrama de Clases en términos de entidades, atributos y relaciones del DER. A continuación, se presenta un paso a paso general para realizar esta transformación:

1. Identificar las clases: Revisa el Diagrama de Clases y lista todas las clases presentes en él.

2. Identificar las entidades: Cada clase en el Diagrama de Clases representa una posible entidad en el Diagrama Entidad-Relación. Identifica las clases que tienen atributos relevantes y que se deben convertir en entidades.

3. Mapear atributos: Para cada entidad identificada, mapea los atributos correspondientes. Cada atributo de una clase se convierte en un atributo de la entidad correspondiente en el DER. Asegúrate de tener en cuenta los tipos de datos adecuados y cualquier restricción o cardinalidad asociada a los atributos.

4. Identificar las relaciones: Analiza las relaciones entre las clases en el Diagrama de Clases y determina cómo se traducen en relaciones en el DER. Las asociaciones entre clases se convierten en relaciones entre las entidades correspondientes. Identifica los tipos de relaciones, como uno a uno, uno a muchos o muchos a muchos.

5. Mapear cardinalidades: Para cada relación identificada, mapea las cardinalidades. Observa las multiplicidades definidas en las asociaciones del Diagrama de Clases y tradúcelas a cardinalidades en el DER, como "1", "0..1", "0..\*" o "\*".

6. Identificar las claves primarias: Determina las claves primarias de cada entidad en el DER. En el Diagrama de Clases, esto puede estar representado por atributos o combinaciones de atributos. Asegúrate de asignar claves primarias únicas a cada entidad.

7. Mapear herencia: Si hay jerarquías de herencia presentes en el Diagrama de Clases, decide cómo se representarán en el DER. Las opciones comunes son el enfoque de tabla única (un atributo discriminatorio) o el enfoque de tabla por clase (tablas separadas para cada subclase).

8. Revisar y refinar: Una vez que hayas realizado la transformación inicial, revisa el Diagrama Entidad-Relación resultante. Verifica que todas las entidades, atributos y relaciones se hayan mapeado correctamente. Realiza ajustes y refinamientos según sea necesario para asegurar que el DER capture correctamente la estructura y relaciones del Diagrama de Clases.

Es importante tener en cuenta que esta transformación puede variar dependiendo de las características específicas del Diagrama de Clases y de los requerimientos del sistema en cuestión. Además, ten en cuenta que algunos elementos del Diagrama de Clases, como métodos y operaciones, no tienen una representación directa en un Diagrama Entidad-Relación, ya que se centra en la estructura de los datos.

Describa (en forma ordenada, cronológica, sumaria) los ciclos de SCRUM: Requisitos futuros, Revisión cada 24 horas, Tareas del sprint, Iteración, Sprint 3-4 semanas, Incremento potencial del proyecto, etc.

Aquí tienes una descripción ordenada y cronológica de los ciclos de Scrum:

1. Planificación del Sprint: Al comienzo de cada sprint (generalmente de 2 a 4 semanas de duración), el equipo de Scrum se reúne en una sesión de planificación del sprint. Durante esta reunión, se seleccionan los elementos del backlog del producto que se abordarán en el sprint, se establecen los objetivos del sprint y se definen las tareas específicas que se realizarán para completar esos elementos.

2. Reuniones diarias de seguimiento (Daily Scrum): Durante el sprint, el equipo se reúne diariamente en reuniones de seguimiento rápidas y cronometradas llamadas Daily Scrum o Stand-up. En estas reuniones, cada miembro del equipo informa sobre lo que hizo el día anterior, lo que planea hacer ese día y si tiene algún impedimento. El objetivo es mantener a todos en el equipo actualizados sobre el progreso y los desafíos.

3. Trabajo en las tareas del sprint: A lo largo del sprint, los miembros del equipo trabajan en las tareas asignadas. Siguen el plan establecido durante la planificación del sprint y colaboran para completar las tareas de manera incremental.

4. Iteraciones y retrospectivas: Durante el sprint, el equipo tiene ciclos de iteración, donde se revisan y mejoran continuamente. Esto implica realizar revisiones periódicas del progreso, adaptar las tareas y ajustar el plan según sea necesario. Al final de cada sprint, se realiza una retrospectiva para analizar lo que funcionó bien, lo que no funcionó y cómo se puede mejorar en el próximo sprint.

5. Revisión del sprint: Al final de cada sprint, se lleva a cabo una reunión de revisión del sprint. En esta reunión, el equipo muestra los elementos completados del backlog del producto al cliente o al propietario del producto. Se discuten los resultados, se recopilan comentarios y se determina si se ha alcanzado el objetivo del sprint. Esta revisión también puede ayudar a identificar nuevas necesidades o cambios en los requisitos futuros.

6. Potencial incremento del proyecto: Después de cada sprint exitoso, el proyecto se considera potencialmente incrementado. Esto significa que se ha agregado valor al producto y que el equipo está cada vez más cerca de alcanzar los objetivos finales del proyecto.

Estos ciclos de Scrum se repiten continuamente a lo largo del desarrollo del proyecto, con cada sprint proporcionando un incremento funcional del producto y permitiendo una adaptación constante a medida que se van obteniendo retroalimentaciones y se realizan mejoras. El enfoque de Scrum se basa en la entrega incremental y la mejora continua a lo largo del tiempo.