FACULTAD:	**Tecnología Informática**		
CARRERA:	**Analista Programador**		
ALUMNO/A:	**GERARDO RODOLFO TORDOYA**		
SEDE:	**Buenos Aires**	LOCALIZACIÓN:	**Distancia**
ASIGNATURA:	**Metodología de Desarrollo de Sistemas II**		
CURSO:	**T4-17-15 2-K-D**	TURNO:	**Distancia**
PROFESOR:	**Leonel Jiménez G.**	FECHA:	**27/05/2023**
TIEMPO DE RESOLUCIÓN:	**5 días**	EXAMEN PARCIAL NRO:	**2**
MODALIDAD DE RESOLUCIÓN:		**Domiciliario – Individual**	

Resultados de aprendizaje:

- [Diseña]+ [El sistema informático] + [Para representar la estructura estática y el comportamiento dinámico] + [Aplicando un Proceso de desarrollo] / [integrando todas las vistas de modelado] / [Aplicando el estándar UML]
- [Resuelve]+ [en posibles situaciones de conflicto]+ [para consensuar ideas comunes] +[considerando las características de cada participando del grupo]
- [Elabora] + [Informes técnicos]+ [para comunicar sus producciones] + [considerando la legibilidad y organización de la información]

Los estudiantes encontrará el examen habilitado en la plataforma ULTRA a partir de la fecha y hora indicada por el profesor.

Criterios de calificación: Para acreditar los saberes deberá tener aprobado, al menos, el 60% de los aspectos conceptuales, además de tener aprobado también, al menos, el 60% de los aspectos procedimentales. La calificación final se obtendrá luego de la defensa oral del trabajo presentado.

Criterios de evaluación: Se evaluará la claridad en el planteamiento de los aspectos conceptuales y procedimentales. La evaluación se hará a partir de la siguiente grilla:

Criterio	Calificación	Observaciones
Instancia oral		
Aspectos Conceptuales		
Pregunta 1		
Pregunta 2		
Pregunta 3		
Pregunta 4		
Pregunta 5		
Aspectos procedimentales		
Pregunta 1		
Pregunta 2		
Pregunta 3		
Pregunta 4		
Calificación final		

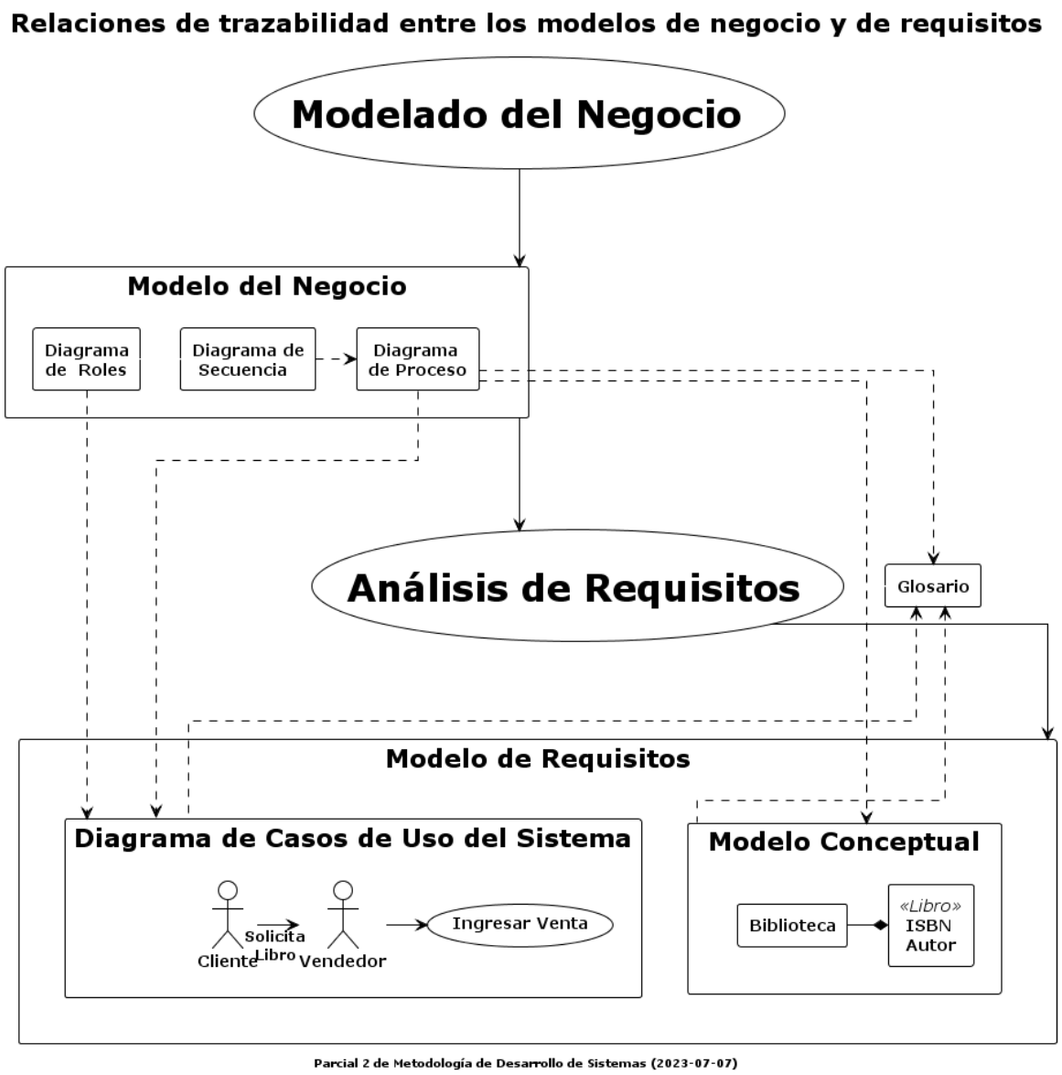
Forma de entrega del examen

- Se debera resolver el examen sobre el mismo archivo del examen original.
- Se deberá subir al ULTRA el examen ya resuelto en un archivo tipo ZIP o RAR.

Aspectos conceptuales

1. Enumere y describa los pasos implicados dentro de la técnica (Articulo) “De los Procesos de Negocio a los Casos de Uso” [30]

La técnica "De los Procesos de Negocio a los Casos de Uso" es un enfoque que permite traducir los procesos de negocio de una organización en requisitos funcionales para un sistema de información. Para producir los diagramas mencionados, los pasos implicados en esta técnica son los siguientes:



Para el Modelo de Negocio: Hay que identificar los procesos de negocio relevantes, esto es: comenzar por analizar y comprender los procesos de negocio existentes en la organización e identificar los procesos clave que se relacionan con el sistema que se va a desarrollar.

- **Diagrama de Roles:** Identifica los diferentes roles o actores involucrados en los procesos de negocio: el diagrama de roles representa las interacciones entre los diferentes actores y los procesos de negocio.
- **Diagrama de Secuencia:** Para cada proceso de negocio identificado, el diagrama de secuencia muestra la secuencia de interacciones entre los actores y el sistema (este diagrama ayuda a visualizar cómo los actores interactúan con el sistema en diferentes etapas del proceso).
- **Diagrama de Proceso:** el diagrama de proceso representa visualmente el flujo de trabajo y las actividades involucradas en cada proceso de negocio: este diagrama ayuda a comprender la lógica y las interdependencias entre las diferentes etapas del proceso.

Para el Modelo de Requisitos: Se deben identificar los casos de uso utilizando los diagramas de proceso y la información recopilada en los pasos anteriores, identificar los diferentes casos de uso del sistema. Un caso de uso representa una interacción entre un actor y el sistema para lograr un objetivo específico.

- **Diagrama de Casos de Uso:** el diagrama de casos de uso muestra los diferentes actores y los casos de uso identificados. Este diagrama ayuda a visualizar las interacciones entre los actores y el sistema, y proporciona una vista general de los requisitos funcionales del sistema.
- **Modelo Conceptual:** el modelo conceptual captura las entidades y las relaciones importantes relacionadas con el sistema. Este modelo puede ser representado mediante un diagrama de clases, que muestra las clases, sus atributos y las relaciones entre ellas. El modelo conceptual ayuda a establecer una comprensión clara de la estructura y los conceptos clave del sistema.

Estos pasos permiten traducir los procesos de negocio en requisitos funcionales y representarlos en diferentes diagramas, lo que facilita la comunicación entre los stakeholders y el equipo de desarrollo del sistema.

---

2. Explique para qué se utiliza el patrón Adaptador. (Explicación y Diagrama de Clases) [20].

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:

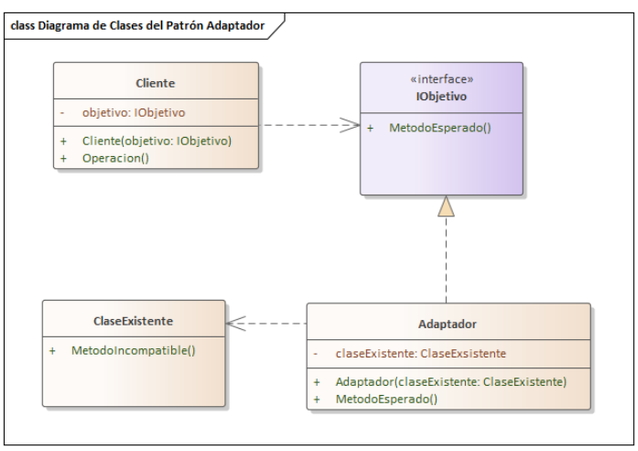
IObjetivo: Es la interfaz o clase esperada por el sistema o cliente.

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

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.

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 que ilustra la estructura del patrón Adaptador:



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.

En otras palabras (porque estas definiciones son muy abstractas sin un ejemplo), el adaptador se encarga de convertir la interfaz de uno de los objetos en otra interfaz que el cliente espera. En este sentido, el adaptador "mapea" o adapta los métodos de una interfaz a los métodos de otra interfaz. Esto permite que objetos con interfaces diferentes puedan colaborar y comunicarse entre sí sin necesidad de modificar el código fuente original.

Ejemplo:

```
// Interfaz esperada por la clase Cliente
public interface IElectrodomestico
{
    void Encender();
    void Apagar();
}

// Implementación original incompatible
public class Lavadora
{
    public void EncenderLavadora()
    {
        Console.WriteLine("Encendiendo la lavadora");
    }

    public void ApagarLavadora()
    {
        Console.WriteLine("Apagando la lavadora");
    }
}

// Adaptador que mapea la interfaz de la Lavadora a la interfaz esperada por Cliente
public class AdaptadorLavadora : IElectrodomestico
{
    private Lavadora lavadora;

    public AdaptadorLavadora(Lavadora lavadora)
    {
        this.lavadora = lavadora;
    }

    public void Encender()
    {
        lavadora.EncenderLavadora();// Aquí se hace la adaptación.
    }

    public void Apagar()
    {
        lavadora.ApagarLavadora();
    }
}
```

---

3. Explique las siguientes expresiones en OCL:
    - Context Proyecto inv: self.participa -> select (edad > 50) ->notEmpty() [5].
    - Context Job inv: self.empleado.edad > 21 [5].

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" que indica que 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.

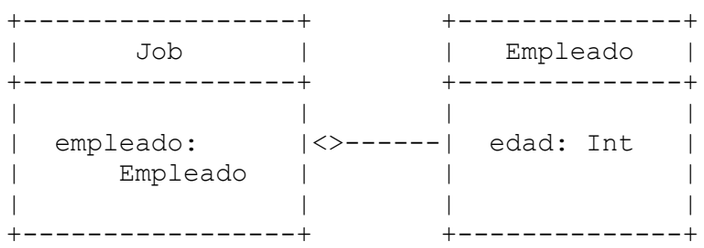
En resumen, esta expresión OCL verifica que en un objeto "Proyecto", haya al menos una instancia relacionada a través de la asociación "participa" cuya edad sea mayor a 50.

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) en la que la propiedad "edad" del objeto relacionado "empleado" debe ser mayor a 21.

Esta expresión 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.



---

4. Explique cómo se realiza paso a paso la transformación de un Diagrama de Clases de un Diagrama Entidad Relación. [20 Puntos]

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 (basado, considerablemente, en los apuntes de MDS1 y BBDD):

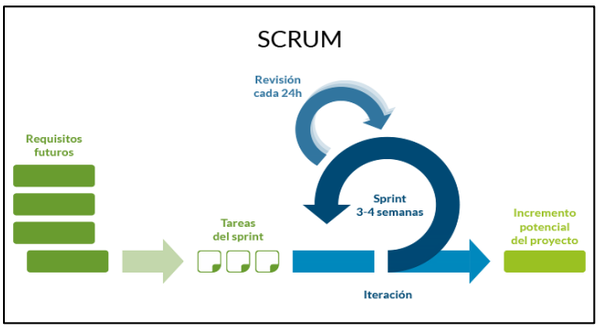
- **Identificar las clases:** Revisar el Diagrama de Clases y listar todas las clases presentes en él.
- **Identificar las entidades:** Cada clase en el Diagrama de Clases representa una posible entidad en el Diagrama Entidad-Relación. Identificar las clases que tienen atributos relevantes y que se deben convertir en entidades.
- **Mapear los atributos:** Para cada entidad identificada, mapear los atributos correspondientes. Cada atributo de una clase se convierte en un atributo de la entidad correspondiente en el DER (tener en cuenta los tipos de datos).
- **Identificar las relaciones:** Analizar 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.
- **Mapear las cardinalidades:** Observar las multiplicidades definidas en las asociaciones del Diagrama de Clases y traducirlas a cardinalidades en el DER, como "1", "0..1", "0..*" o "*".
- **Identificar las claves primarias:** Determinar las claves primarias de cada entidad en el DER. En el Diagrama de Clases, esto puede estar representado por atributos o combinaciones de atributos. (Esto es tema de debate, realmente. Yo adopto la postura de que las claves

primarias no deben basarse en datos del negocio, así que, en el pasaje al modelo conceptual o lógico, según el caso, cada entidad o tabla -según su propia lógica- tendrá su propia PK).

- **Mapear las herencias:** Si hay jerarquías de herencia presentes en el Diagrama de Clases, decidir 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). Por ejemplo, en el último punto de este parcial, lo que diferencia a la clase abstracta "Tarjeta" de las clases heredadas son los métodos, lo cual, en el modelo ER, no tiene representación. Ahí se adoptó el enfoque de Tabla Única.
- **Revisar y refinar:** Una vez que se haya realizado la transformación inicial, hay que realizar 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. Por ejemplo, las restricciones que se pidieron para el caso presentado en programa de viajeros frecuentes, último punto de este parcial. Tener en cuenta que algunos elementos del Diagrama de Clases, como métodos y operaciones, no tienen una representación directa en un Diagrama Entidad-Relación (que se centra en la estructura de los datos).

---

5. Explique el siguiente diagrama que se relaciona con las metodologías Ágiles [20].



LO QUE MUESTRA EL GRÁFICO

EL gráfico muestra cómo se desarrolla un proyecto de software utilizando la metodología Scrum. La secuencia ordenada de eventos y conceptos es la siguiente:

- **Cliente y Requisitos Futuros:** Un cliente tiene una necesidad de software y proporciona los requisitos iniciales para el proyecto.
- **Product Backlog:** El Product Backlog es una lista ordenada de todas las funcionalidades, requisitos y mejoras deseadas para el software. El cliente y el Project Owner (dueño del proyecto) trabajan juntos para crear y priorizar el Product Backlog.
- **Sprint Planning:** El Sprint Planning es una reunión en la que el equipo de desarrollo, el Scrum Master y el Project Owner planifican el próximo Sprint. Durante esta reunión, el equipo selecciona las tareas del Product Backlog que se abordarán en el próximo Sprint.
- **Tareas del Sprint:** Las tareas seleccionadas del Product Backlog se dividen en unidades más pequeñas y manejables llamadas Tareas del Sprint. Estas tareas representan el trabajo específico que el equipo de desarrollo necesita realizar durante el Sprint.
- **Sprint:** Un Sprint es un período de tiempo fijo, típicamente de 3-4 semanas, en el que se lleva a cabo el trabajo del equipo de desarrollo. Durante el Sprint, el equipo trabaja en las *Tareas del Sprint* y se esfuerza por completarlas dentro del plazo establecido.
- **Daily Scrum:** El Daily Scrum (reunión diaria) es una breve reunión de 15 minutos en la que el equipo de desarrollo se sincroniza. Durante esta reunión, cada miembro del equipo comparte qué hizo ayer, qué hará hoy y si tiene algún obstáculo.

- **Incremento:** El Incremento es el resultado tangible y potencialmente utilizable del trabajo del equipo de desarrollo al final de cada Sprint. Es una versión mejorada y ampliada del software que incorpora las funcionalidades completadas durante el Sprint.
- **Sprint Review:** Al final de cada Sprint, se lleva a cabo una reunión de Sprint Review en la que el equipo de desarrollo presenta el Incremento completado al cliente y a otras partes interesadas. Durante esta reunión, se recopilan comentarios y se realizan ajustes en el Product Backlog según sea necesario.
- **Sprint Retrospective:** Después de la reunión de Sprint Review, el equipo de desarrollo se reúne en la Sprint Retrospective para reflexionar sobre el Sprint anterior. Se discuten las lecciones aprendidas, se identifican áreas de mejora y se establecen acciones correctivas para el próximo Sprint.
- **Incremento Potencial del Proyecto:** El Incremento Potencial del Proyecto se refiere a la versión mejorada y ampliada del software que se entrega al final de cada Sprint. El Incremento Potencial es un resultado tangible que puede ser utilizado o mostrado al cliente, y representa el progreso hacia el producto final.

CONCEPTOS INVOLUCRADOS

- **Scrum Master:** El Scrum Master es el facilitador y defensor del uso correcto de la metodología Scrum. Ayuda al equipo de desarrollo a comprender y adoptar los principios y prácticas de Scrum, y también facilita las reuniones y la comunicación entre los miembros del equipo.
- **Project Owner:** El Project Owner es el representante del cliente o la organización que es responsable de maximizar el valor del producto desarrollado. Colabora con el equipo de desarrollo para establecer las prioridades del Product Backlog y garantizar que las necesidades del cliente se cumplan.
- **Development Team:** El Development Team (equipo de desarrollo) está compuesto por los programadores y otros especialistas necesarios para crear el software. Trabajan juntos para completar las Tareas del Sprint y entregar el Incremento al final de cada Sprint.
- **Artefactos:** Los artefactos en Scrum son elementos tangibles que se utilizan para facilitar la planificación, seguimiento y comunicación en el proyecto. Los principales artefactos en Scrum son el Product Backlog, el Sprint Backlog y el Incremento.
- **Backlog:** El Backlog es una lista de elementos o tareas pendientes de hacer. En Scrum, hay dos tipos de Backlogs: el Product Backlog y el Sprint Backlog.
- **Product Backlog:** El Product Backlog es una lista ordenada de todas las funcionalidades y requisitos deseados para el producto. El Product Owner es responsable de priorizar y mantener actualizado el Product Backlog.
- **Sprint Backlog:** El Sprint Backlog es una lista de las Tareas del Sprint seleccionadas para el Sprint actual. Es creado por el equipo de desarrollo durante la reunión de Sprint Planning.
- **Eventos Scrum:** Los eventos Scrum son actividades programadas que ayudan a estructurar y controlar el flujo de trabajo en Scrum. Los eventos Scrum incluyen el Sprint, Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective.
- **Programadores:** Los programadores son miembros del equipo de desarrollo y son responsables de implementar el software según las tareas asignadas. Participan activamente en las reuniones diarias (Daily Scrum) y colaboran con otros especialistas para completar las Tareas del Sprint.
- **Analistas Funcionales:** Los analistas funcionales son expertos que ayudan a definir y comprender los requisitos del software. Trabajan en estrecha colaboración con el cliente, el Project Owner y el equipo de desarrollo para garantizar que se cumplan los objetivos del proyecto.
- **Jefe de Proyecto:** En Scrum, el rol de Jefe de Proyecto se diluye, ya que el enfoque se basa en la colaboración y la autogestión del equipo. El Project Owner asume la responsabilidad

de la gestión del proyecto, mientras que el Scrum Master ayuda a coordinar y facilitar el proceso Scrum.

- **Mantenimiento y Soporte del Software:** El mantenimiento y soporte del software se consideran actividades continuas una vez que el proyecto inicial se completa. Estas actividades pueden incluir correcciones de errores, actualizaciones, mejoras y asistencia técnica para garantizar el funcionamiento continuo del software.

PROBLEMAS OBVIOS DE LA METODOLOGÍA ASÍ COMO ESTÁ PROPUESTA

- **Falta de enfoque en la calidad:** Scrum pone demasiado énfasis en la entrega rápida de incrementos de software, lo que puede conducir a una menor atención a la calidad y a un aumento de la deuda técnica[1]. Es importante equilibrar la entrega rápida con el mantenimiento de altos estándares de calidad.
- **Falta de roles definidos:** Scrum promueve equipos auto-organizados y no enfatiza las jerarquías tradicionales. Esto puede llevar a una falta de claridad en cuanto a los roles y responsabilidades individuales, lo que a veces puede causar confusión y conflictos dentro del equipo.
- **Limitaciones en proyectos grandes y complejos:** Aunque Scrum es adecuado para proyectos pequeños, tiene claras dificultades para escalar a proyectos grandes o complejos. La coordinación entre múltiples equipos y la gestión de dependencias pueden ser desafiantes en estas situaciones.

UNA ÚLTIMA ACLARACIÓN

Puede ser necesario aclarar algo: ¿qué significa Scrum? No es un acrónimo. ¿Entonces qué es? El término "Scrum" se deriva del rugby, donde se utiliza para describir una formación de equipo en la que los jugadores se agrupan juntos para reiniciar el juego después de una infracción menor.

Esto es Scrum. Así concibe Scrum el desarrollo de software.

[1] **Este concepto de "deuda técnica" lo acabo de conocer y es interesante:** La deuda técnica es un concepto que se refiere a las consecuencias negativas que surgen cuando se toman atajos o se toman decisiones que comprometen la calidad del software durante el proceso de desarrollo. Es decir, se refiere a las consecuencias negativas que surgen cuando se toman atajos o se toman decisiones que comprometen la calidad del software durante el proceso de desarrollo. El concepto de deuda técnica destaca la importancia de abordar y pagar esta deuda a lo largo del tiempo. Al igual que con una deuda financiera, cuanto más tiempo se posponga el pago, mayor será el costo y los riesgos asociados. Pagar la deuda técnica implica realizar actividades como refactorización del código, mejora de la documentación, pruebas adicionales y revisión de arquitectura para garantizar que el software sea más mantenible, confiable y escalable.

Aspectos procedimentales

Para el siguiente Diagrama de Caso:

Se necesita construir el sistema de gestión del programa de viajeros frecuentes de una línea aérea. El programa de viajero frecuente de la línea aérea permite a sus clientes acumular millas que luego pueden cambiar por premios. Para esto, a cada cliente se le otorga una tarjeta. La tarjeta puede ser de Nivel Plata o Nivel Oro (inicialmente es de nivel plata). El sistema permite cargar los viajes que hacen los usuarios y actualizar el nivel de las tarjetas de los usuarios.

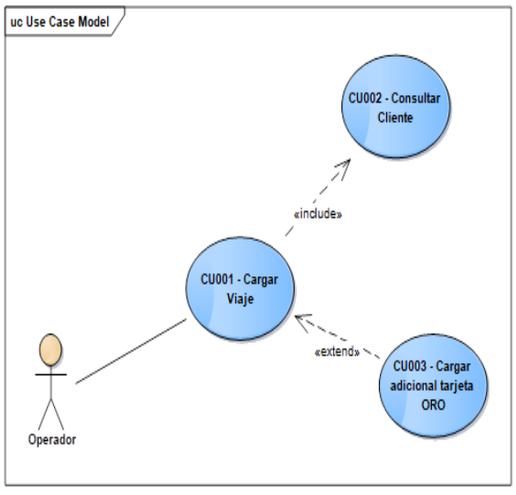
Registrar usuario. Cuando una persona se subscribe al programa, se registra de la misma: su nombre, nacionalidad, dirección postal y de email, y su número de pasaporte.

Carga de viajes. Cuando una persona hace un viaje, se registra del mismo el origen, el destino, el número de vuelo, la fecha, y las millas voladas. Además, cada viaje suma millas de acuerdo al siguiente criterio:

- **Calculo base:** Simplemente se suman las millas del viaje. Si la persona tiene tarjeta oro se duplica la cantidad de millas del viaje.
- **Adicionales:** Los viajes de distancia de más de 10.000 millas suman 1.000 millas adicionales para la tarjeta plata y 2.000 para la tarjeta oro. Para la tarjeta oro, se suman 2.000 millas adicionales para todo viaje dentro del país.

Cobro de premios. El sistema permite registrar el cambio de puntos por distintos premios (viajes, electrodomésticos, etc). Por cada cambio registraremos la fecha, el premio elegido, la cantidad de puntos utilizados.

Emisión de resúmenes. El sistema permite emitir el resumen de la cuenta de cada usuario. Del resumen puede obtenerse, la fecha de emisión del mismo, el total de millas acumuladas desde el último balance que no han sido cambiadas por premios, el total de millas cambiadas por premio desde el ultimo balance, el total de millas que vencerán en el próximo balance (es decir, a un año del último balance).



Se Requiere:

1 Diagrama de Secuencia del CU 001 [20].

2 Diagrama de Actividad del CU 001 [20].

3 Realizar el Diagramas de clase de toda la aplicación e implementar dos restricciones por medio de OCL [30].

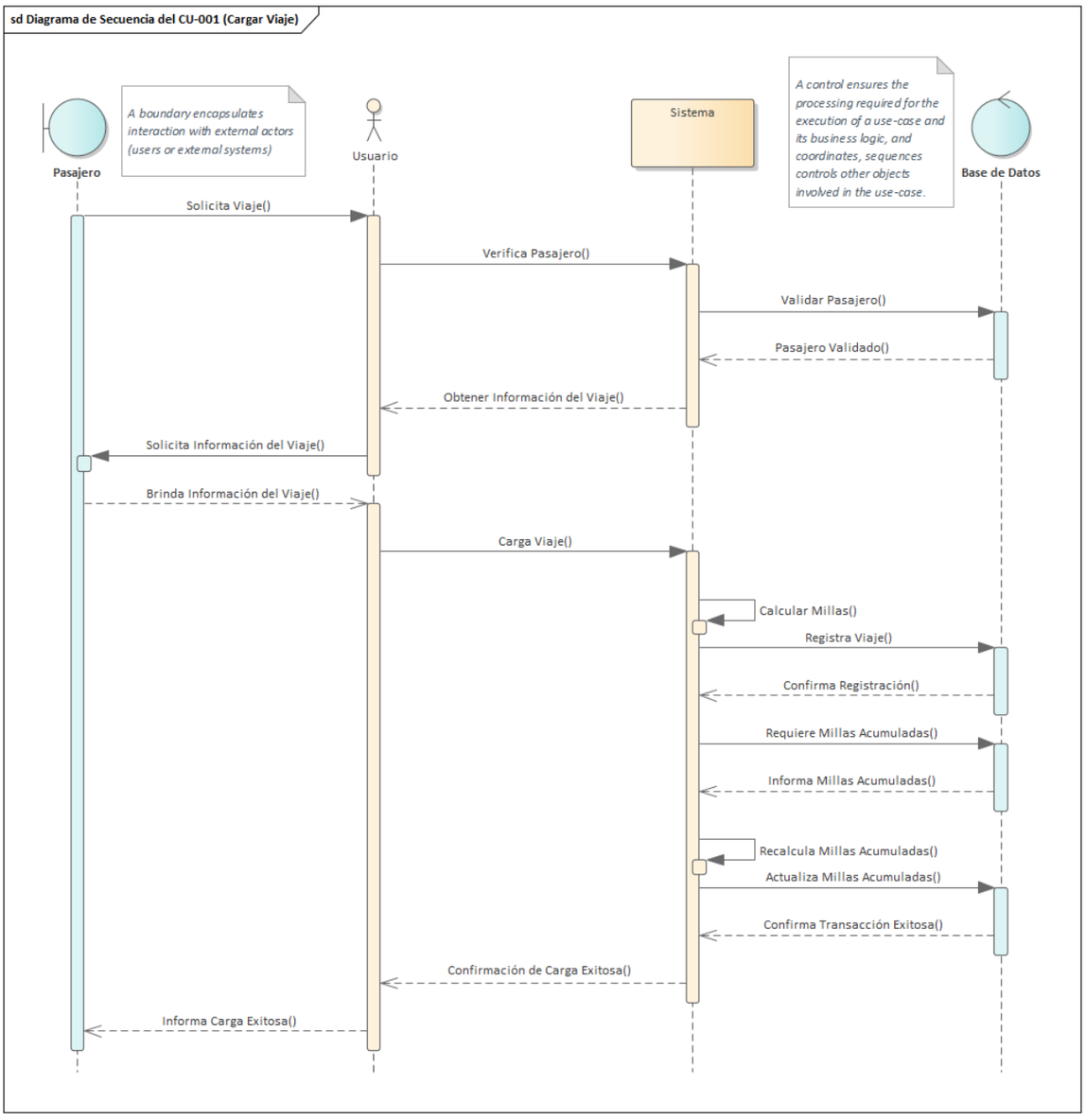
4 Realizar la conversión al modelo de datos correspondiente al punto 3 [30].

Se pueden adicionar más atributos de datos si se consideran convenientes, siempre y cuando se mantengan dentro de la lógica del tema y del sentido común. Cualquier asumisión adicional que se emplee debe ser descrita en el examen.

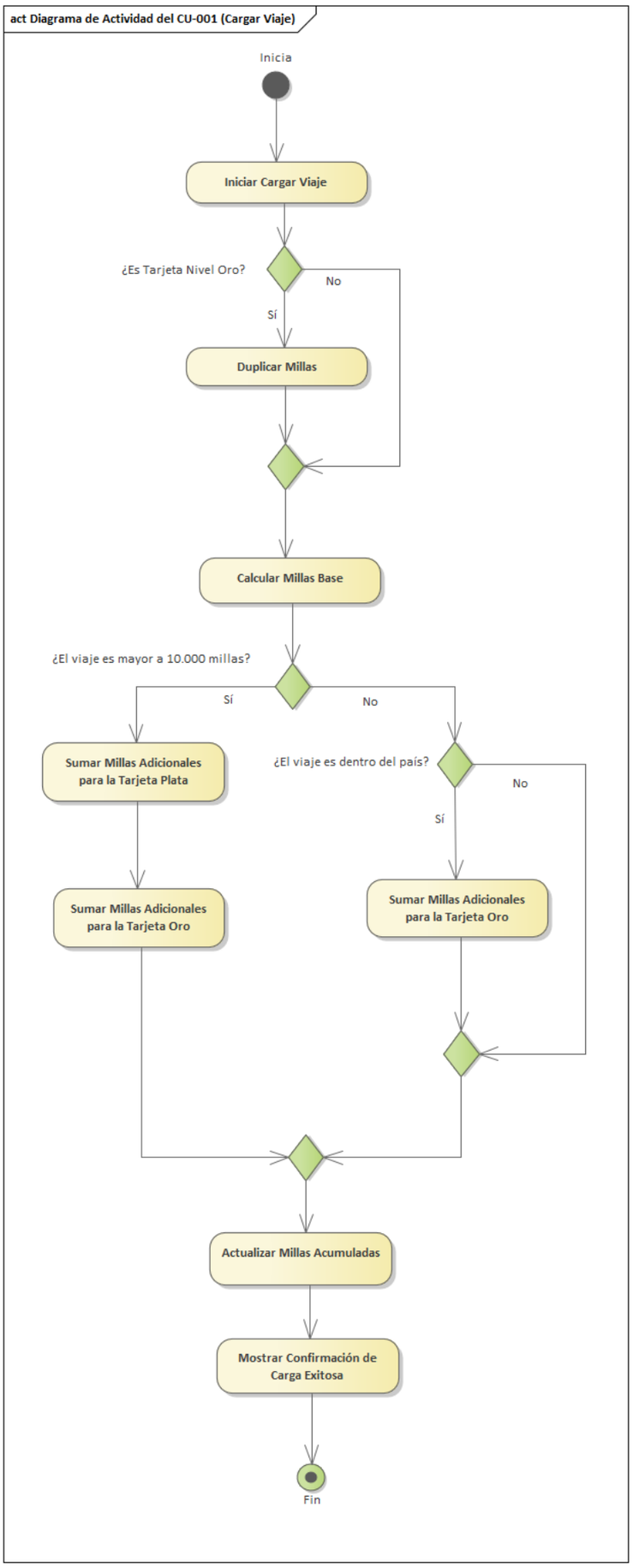
NOTA: Se adicionaron más atributos que se consideraron convenientes de acuerdo a lo cursado en BBDD. Luego se adecuó el Diagrama de Clases para que refleje y esté en sintonía con el DER resultante. Estos atributos, en base a su buena enseñanza y práctica, se mantienen dentro de la lógica y el sentido común, sobre todo cuando respecta a la atomicidad de los datos. No considero que, de esta manera, haya hecho ninguna asumición adicional (soy, naturalmente, enemigo de las suposiciones), y por tanto, no he detallado este agregado de atributos que entiendo no representa ninguna lógica adicional a la planteada.

1. Diagrama de Secuencia del CU-001

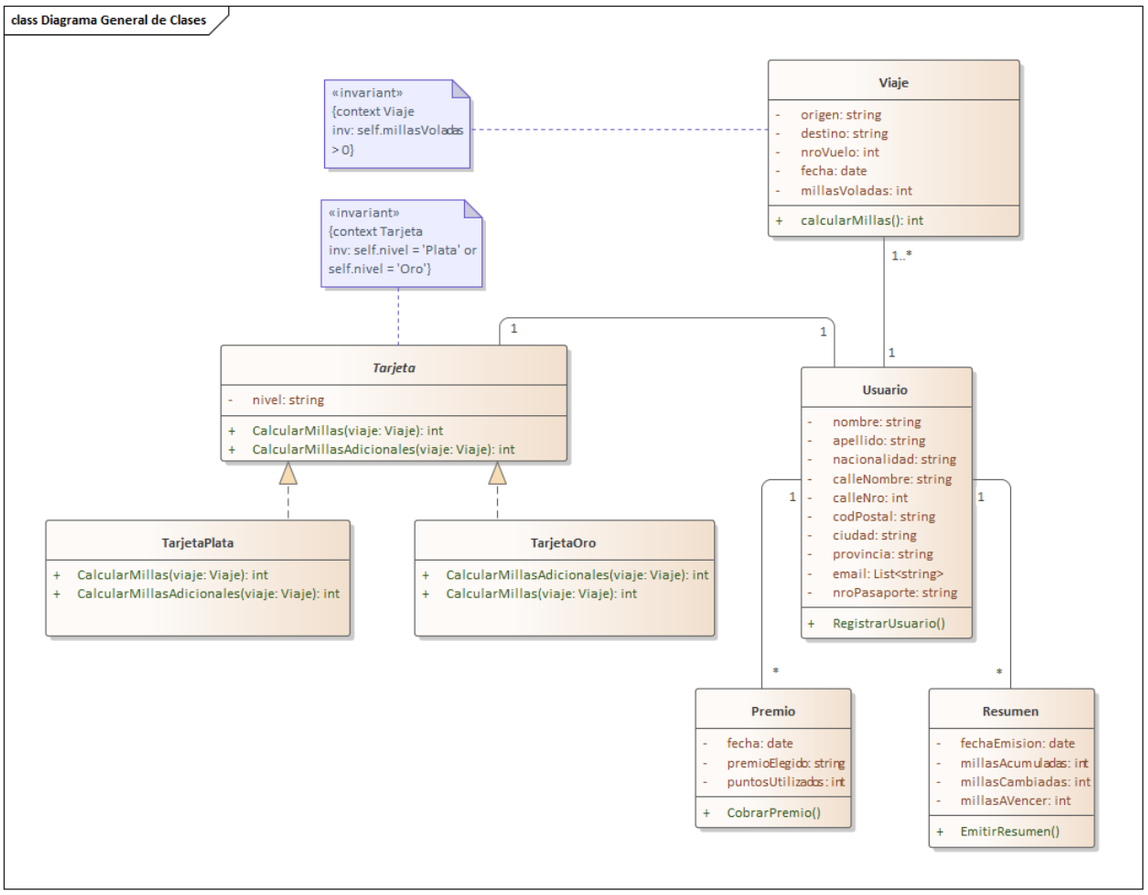
NOTA: Puse en los comentarios las razones pertinentes (sacadas de Wikipedia) por las que adopto esa simbología en particular tanto para “Pasajero” como para “Base de Datos”.



2. Diagrama de Actividad del CU-001

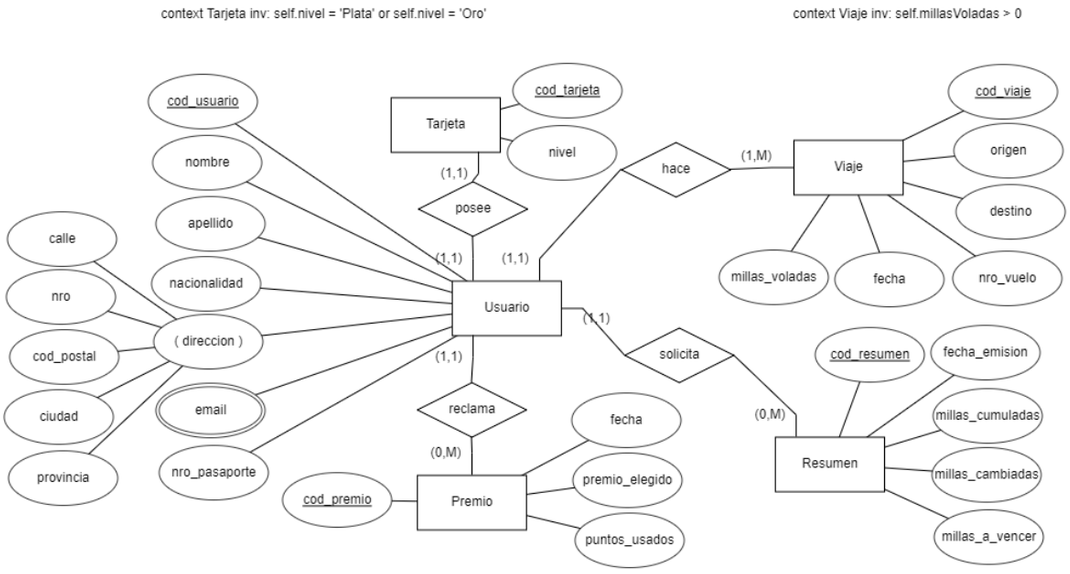


3. Realizar el Diagramas de clase de toda la aplicación e implementar dos restricciones por medio de OCL

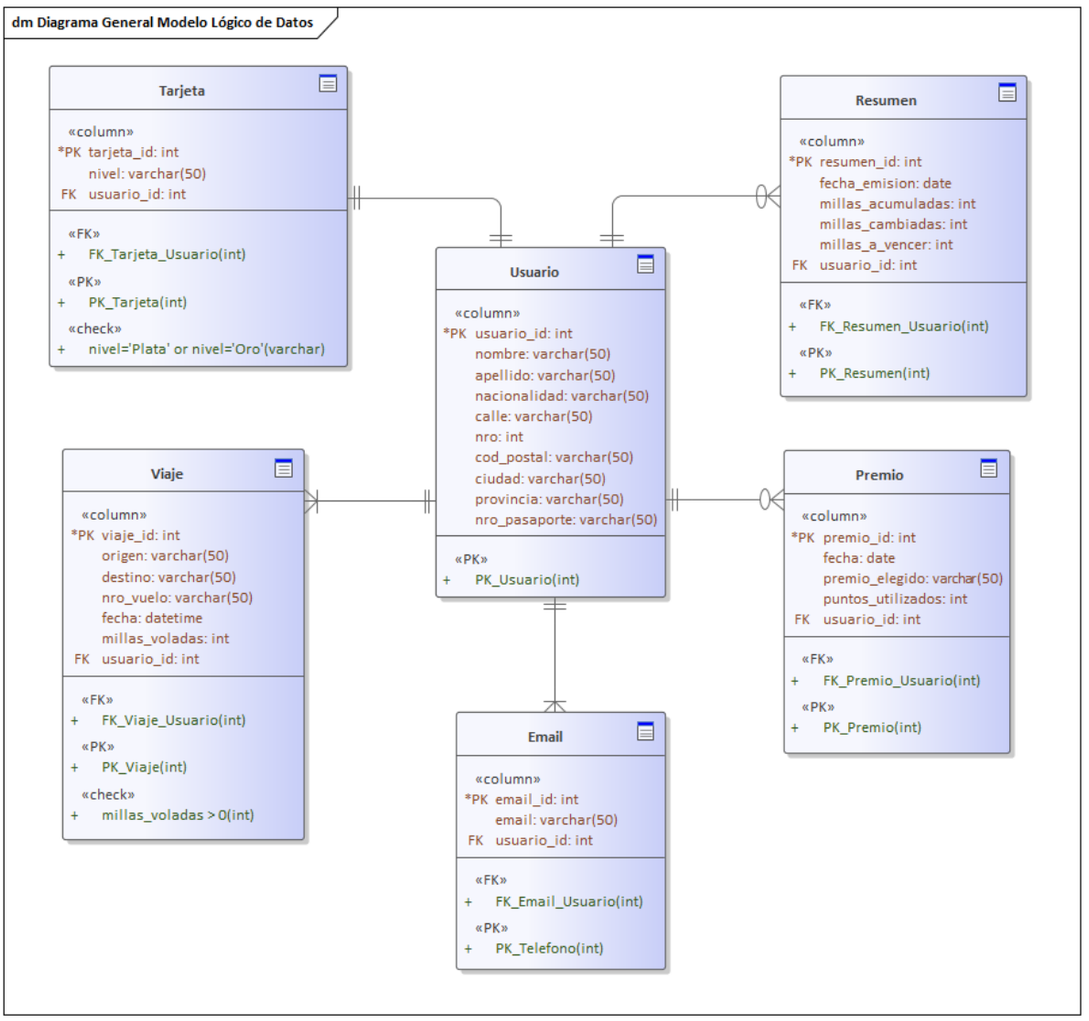


4. Realizar la conversión al modelo de datos correspondiente al punto 3

MODELO CONCEPTUAL



MODELO LÓGICO



MODELO RELACIONAL ASOCIADO

Usuario(**usuario_id, ...**)

Tarjeta(**tarjeta_id, ...,** usuario_id(Usuario))

Resumen(**resumen_id, ...,** usuario_id(Usuario))

Viaje(**viaje_id, ...,** usuario_id(Usuario))

Premio(**premio_id, ...,** usuario_id(Usuario))