
Área Informática

Sistemas III

Material elaborado por
Ing. Cecilia Savi

INSTITUCION CERVANTES



2006

Sistemas III

Carta al alumno

Dice Rodolfo PEÓN AGUIRRE, ya hace un tiempo, en 1998 , ...“en la actualidad en una buena parte de la sociedad se percibe el deseo de aprender constantemente nuevas cosas. Aprender para tener más y mejores oportunidades; aprender para tener mejor comunicación con el entorno; aprender para sentirnos realizados como seres humanos; pero poder hacerlo sin necesidad de afectar nuestra actividad en el trabajo, sin depender de tiempos fijos o necesitar de espacios especiales, aprender y saber más en tiempo y espacios adecuados a las posibilidades de cada quién”..... Esa es nuestra intención, como autora de este material y como integrante de la cátedra de Sistemas de la Institución Cervantes, deseamos que estos contenidos sean una síntesis que muestre los aspectos mas relevantes de una metodología de trabajo actualizada, de una metodología de sistemas útil al momento de analizar y diseñar sistemas de información, aplicables a todo tipo de organizaciones.

Se trabajó el módulo, priorizando los conceptos y ejemplos prácticos sobre los aspectos meramente teóricos. Se utilizaron diversos tipos de ejemplos , buscando mayor diversidad en las aplicaciones, mejor entendimiento por el uso de situaciones comunes, propias de la realidad actual.

Deseamos que el proceso “de aprender” sea comprometido y significativo, y esperamos una comunicación fluida que permita generar aportes y propuestas que seguramente serán para mejorar la idea original.

Fundamentación

Dentro de la estructura del plan de estudios de la carrera de Analista de Sistemas de computación, la materia se encuentra en el tercer cuatrimestre, que corresponde al segundo año de estudios.

En esta asignatura, usted aprenderá a realizar todas las acciones inherentes a la resolución de problemas a través del estudio, y desarrollo de sistemas de información eficientes en áreas de la organización; además, deberá proponer alternativas de solución a las necesidades y problemas detectados.

Es decir, desarrollará actividades que le permitan dimensionar, analizar e ingresar al diseño de sistemas de información en el contexto de las organizaciones o en casos de estudio.

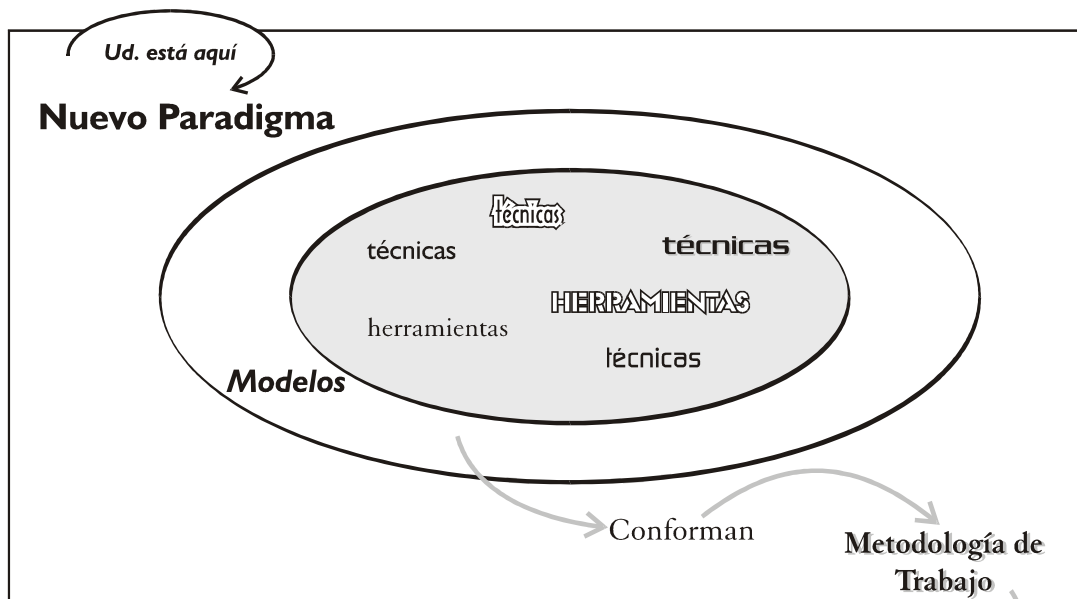
Para ello el módulo, proporciona una guía práctica que potencia las habilidades fundamentales que se requieren al analizar los sistemas de información de negocios actuales.

En él se encontrarán herramientas para la interpretación, análisis y creación de modelos representativos de la realidad a estudiar, en el ámbito de los sistemas de información.

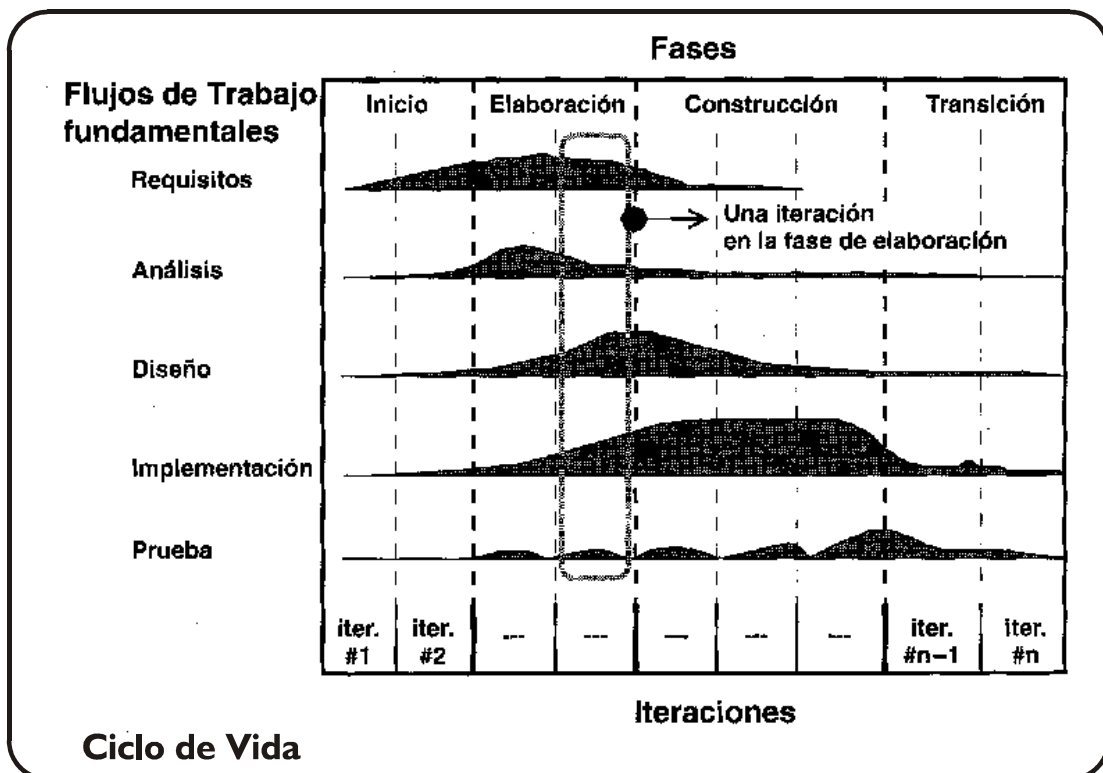
Objetivos Generales

- ▶ Conocer los fundamentos del surgimiento de la Tecnología de Objetos.
- ▶ Comprender el rol de la tecnología de objetos, dentro de la Ingeniería de Software.
- ▶ Entender las diferencias existentes entre los diferentes modelos adoptados históricamente para el ciclo de desarrollo de un sistema.
- ▶ Conocer el Proceso Unificado de Desarrollo de Software e identificar los flujos de trabajo, fases, artefactos o modelos básicos.
- ▶ Internalizar el significado de los procesos en la captura de requisitos dentro del paradigma a Objetos.
- ▶ Interpretar los modelos de sistemas construidos, como así también la vinculación de estos, en cada flujo de trabajo y en las fases.
- ▶ Aplicar los conceptos teóricos a ejemplos concretos.

Mapa Conceptual



Proceso Unificado de Desarrollo de Software



Unidad I

Introducción a la Tecnología de Objetos” Proceso Unificado de Desarrollo de Software- Ciclo de Vida

Tecnologías orientadas a objetos. Componentes reusables. El paradigma de la integración. Los factores que influyeron en el cambio de paradigma. Ingeniería de software (Un concepto de actualidad).

¿Qué es una metodología. ¿Qué es lo que hace que una metodología sea buena?

El Proceso Unificado de desarrollo de Software

El Proceso Unificado, dirigido por casos de uso

El Proceso Unificado está centrado en la arquitectura

El Proceso Unificado es iterativo e incremental

Relación entre los tres aspectos del Proceso de desarrollo Unificado

Ciclo de Vida del Proceso Unificado

Fases dentro de un ciclo. Iteraciones

Flujos de trabajo del proceso. Modelos.

UML

Unidad II

Captura de Requisitos: “Enumerar los requisitos candidatos”. “Comprender el contexto del sistema.

Visión general de la captura de requisitos

1- Enumerar los requisitos candidatos

2- Comprender el contexto del sistema

Comprensión del contexto del sistema mediante Modelo del Dominio

Comprensión del contexto del sistema mediante un Modelo del Negocio

Modelo de Caso de uso del Negocio

Diferentes categorías de Caso de uso de Negocio.

¿Están los Caso de uso del negocio relacionados siempre con actores?

Vistas del negocio. Nombre de un Caso de uso. Estructura de un Caso de uso

Actor.

Diagrama de Caso de uso en el modelo de UC de Negocio

Unidad III

Captura de Requisitos: “Capturar requisitos funcionales”.

“Capturar requisitos no funcionales

¿Qué es la captura de requisitos?. ¿Por qué la captura de requisitos es complicada?.

¿Cómo se consigue?. ¿Cuál es la finalidad?

3- Capturar requisitos funcionales

- Encontrar actores y casos de uso - Describir brevemente cada caso

- Priorizar casos de uso

- Detallar un caso de uso. Descripción de funcionalidad compartida

Generalización entre casos de uso. Una relación de inclusión

- Prototipar la interfaz del usuario

- Estructurar el modelo de casos de uso

4- Capturar requisitos no funcionales

Síntesis: Modelos y diagramas de la Captura de Requisitos en el ciclo de vida.

Unidad IV

Como construimos los modelos

¿Qué es entonces un modelo?. ¿Por qué modelamos?

¿Las herramientas son esenciales en el proceso?

Diagramas.

Caso práctico: Enunciado

1. Modelado del Sistema de Negocio

1.1. Modelo de Use case del Sistema de Negocio (diagrama y descripciones de los use cases).

1.2. Modelo de Objetos del Dominio.

2. Modelado del Sistema de Información

2.1. Modelo de use case (diagrama de use case y descripción de cada use case).

2.2. Descripción o Prototipo de Interfaz.

Unidad V

Objetos

Introducción básica a Objetos. ¿Qué es un objeto?

¿Qué es un tipo objeto?

Métodos. Encapsulado. Mensajes. Blob

¿Qué es una clase?

Herencia. Herencia de clase

Polimorfismo

Análisis de la estructura de objetos

Asociaciones de objetos

Jerarquías de generalización

Bases de datos orientadas a objetos

Independencia de datos versus encapsulado

Rendimiento

Diferencias entre las bases de datos por relación y las bases de datos orientadas a objetos.

Orientaciones Metodológicas

Propuesta Metodológica

Brevemente en este punto, se plantea la organización de los temas en el módulo, y recomendaciones sobre la forma de realizar su lectura.

El módulo se organiza:

En su **primer unidad**, una síntesis de las ideas fundamentales sobre la tecnologías de objeto, una conceptualización de metodología, técnicas, herramientas y modelos, dentro del enfoque de sistemas. Se describe un método de trabajo, el Proceso Unificado de Desarrollo de Software (PUDS), e introducen también los conceptos necesarios para

comprender el ciclo de vida, flujo de trabajo y fases, destacando cómo se lleva a cabo el trabajo en cada uno de ellos.

En la **unidad dos**, se trabaja, cómo desarrollar una visión de la nueva organización, definiendo los procesos, roles y responsabilidades de esa organización en un modelo de negocio.

La **unidad tres** trata el concepto de desarrollo dirigido por casos de uso con mayor detalle. Los casos de uso como un medio para determinar los requisitos correctos funcionales del Sistema de Información, se utilizan los flujos de trabajo fundamentales, y se los combinan de un modo ajustado a cada fase, construyendo cada modelo o diagrama, de manera que permitan alcanzar resultados deseados, en cualquier proyecto de sistemas.

Es decir que, las unidades 1-2-3- son el marco teórico práctico, necesario para comprender, la forma de trabajo, las visiones, la arquitectura o esqueleto de esta metodología (PUDS), mientras que **la unidad cuatro**, es la materialización en modelos y diagramas del PDUS, utilizando para ello un enunciado que simula un caso real .

La unidad V es una síntesis bibliográfica necesaria, para construir las clases, identificar los objetos, los métodos, atributos involucrados en todo proyecto de desarrollo de software

Formas de evaluación

Según la modalidad de regularización definida por la cátedra de Sistemas III, Ud., deberá rendir por lo menos un examen parcial escrito, (cualquiera sea la modalidad) en los que deberá desarrollar contenidos teóricos y prácticos.

En el examen final, la forma de evaluación es, en primera instancia, un escrito con un caso práctico para resolver, y en segunda instancia un escrito u oral, (según lo determine el tribunal), sobre aspectos Teóricos complementarios o que fundamenten lo realizado en la actividad práctica.

Material Soporte de Información de todas las unidades

Obligatoria

- ▶ Ivar Jacobson, Grady Booch, James Rumbaugh, *El proceso Unificado de Desarrollo de Software*, Edición 1999 o superior.
- ▶ Grady Booch, James Rumbaugh, Ivar Jacobson, *El lenguaje Unificado de Modelado*, Edición 2000 o superior
- ▶ James Rumbaugh, Ivar Jacobson, Grady Booch, *El Lenguaje Unificado de Modelado. Manual de Referencia* Edición 2000 o superior

Sugerida

- ▶ Joseph Schmuller, *Aprendiendo UML en 24 hs*
- ▶ James Martin – James J. Odell, *Análisis y Diseño Orientado a Objetos*
- ▶ Apuntes de Cátedra (aspectos teóricos). Cátedra Diseño de Sistemas – UTN. FRC – Judith Meles
- ▶ Ejercicios Prácticos de Cátedra . Cátedra Diseño de Sistemas – UTN. FRC – Cattaneo Marcela- Mac William MI – Velez G

Iconos de Referencia



Recomendación



Lectura en bibliografía



Conceptos para recordar

Evaluación Diagnóstica

1. ¿Qué es dato y qué es información?. Establezca diferencias utilizando un ejemplo
2. Cite ejemplos de decisiones de Nivel Estratégico, Táctico y Operativo en una Organización que presta servicio por ejemplo: Cyber café.
3. ¿Quiénes participan en los proyectos de sistemas y cuáles son las responsabilidades que le competen a cada uno?
4. Defina con sus palabras, cada una de las técnicas de relevamiento: Elabore un cuadro con las ventajas y desventajas de cada una.

Técnicas	Ventajas	Desventajas

5. ¿Con qué herramientas usted definiría el dimensionamiento de un proyecto de sistemas?
6. Determinar la factibilidad técnica operativa y económica, ¿es lo mismo que definir el proyecto y sus alcances?
7. ¿Qué herramienta es común utilizar, para planificar tiempos de desarrollo de un proyecto informático?
8. Mencione y explique algún concepto que haya sido significativo para usted en el cuatrimestre anterior y que considere relevante, al momento de comenzar el estudio de un proyecto informático.
9. Elabore un esquema o gráfico que sintetice lo aprendido en Sistemas II

► Unidad I

“Introducción a la Tecnología de Objetos” Proceso Unificado de Desarrollo de Software “Ciclo de vida”

Objetivos

- Conocer el por qué del surgimiento de la Tecnología de Objetos.
- Identificar el rol de la tecnología de objetos, dentro de la Ingeniería de Software.
- Comprender el concepto de Ingeniería de Software.
- Repasar los aspectos generales, de metodologías aplicables al desarrollo de sistemas .
- Interpretar las características y alcances de: metodología, modelo, técnica.
- Conocer de qué trata el Proceso Unificado de desarrollo de Software, y las tres ideas clave: Dirigido por casos de Uso, centrado en la arquitectura, iterativo.
- Interpretar y relacionar las fases y flujos en el entorno del ciclo de vida.
- Introducir el concepto de Lenguaje Unificado de modelado (UML).

Contenidos de la Unidad

Tecnologías orientadas a objetos. Componentes reusables. El paradigma de la integración. Los factores que influyeron en el cambio de paradigma. Ingeniería de software (Un concepto de actualidad).

Qué es una metodología. ¿Qué es lo que hace que una metodología sea Buena?

El Proceso Unificado de desarrollo de Software

El Proceso Unificado, dirigido por casos de uso

El Proceso Unificado está centrado en la arquitectura

El Proceso Unificado es iterativo e incremental

Relación entre los tres aspectos del Proceso de desarrollo Unificado

Ciclo de Vida del Proceso Unificado

Fases dentro de un ciclo. Iteraciones

Flujos de trabajo del proceso. Modelos.

UML

Tecnologías orientada a objetos

El término orientado a objetos pulula por todas partes. Hay entornos orientados a objetos, aplicaciones orientadas a objetos, bases de datos orientadas a objetos, arquitectura, interfaces con el usuario, análisis y diseño orientados a objetos. Y por supuesto, lenguajes de programación orientados a objetos, que incluyen desde el Ada al diferente Smalltalk, Con C++ y Objective-C, Java, y todos aquellos que nos permiten diseñar entorno WEB.

En algún lugar en medio de todos estos términos, uno puede preguntarse ¿qué es lo que tienen todas estas tecnologías en común?, ¿Qué es lo que la expresión orientado a objetos realmente significa?. Para atacar el problema del significado de la orientación a objetos, hay que comprender que la misma es:

- » Un fin, y no un medio, un objetivo mas de la tecnología para lograrlo.
- » Significa cambiar la manera en la que vemos el software, desplazando nuestro énfasis **hacia los objetos que construimos mas que hacia los procesos que utilizamos para hacerlo.**
- » Significa utilizar todas las técnicas disponibles, para hacer del software algo tan tangible, tan ameno y manejable al sentido común como lo son los objetos de la vida real.
- » Orientado a Objeto significa abandonar la visión centrada en el proceso, en el cual se enaltece la relación desarrollador - máquina, en favor de otra centrada en el producto, guiada por la relación productor - consumidor.

También se podría entender mas, si utilizamos otro término: **“Revolución industrial del software”**

Orientado a objetos siempre significó para mí transformar la programación de una artesanía, hecha a medida, a una actividad industrial como lo es hoy, la fabricación. Significa activar a los consumidores de software, haciéndoles posible resolver sus propios y especializados problemas de software, de la misma forma en que se resuelven los problemas caseros de plomería, ensamblando soluciones a partir de un gran mercado de componentes reusables y listos para usar, los cuales son a su vez provistos por múltiples productores de menor nivel.

El problema con el antiguo paradigma puede ilustrarse con una pregunta simple: ¿Al construir una casa, consideraría usted construir el sistema de cañerías a partir de partes hechas a su medida y únicas en su tipo?

Sin embargo, la comunidad del software, encaprichada con los procesos y su presunción de la perfección, espera que los programadores hagan exactamente eso, cada vez que construyen software. Esta actitud sería considerada como fuera de lugar en el campo de la plomería.

El *elemento clave* de la **revolución industrial del software** es la creación del mercado de partes standard, un lugar en el cual aquellos que se especialicen en la resolución de

problemas, puedan comprar componentes de bajo nivel para ensamblarlos, para lograr soluciones de mayor nivel. Ellos tendrán poco interés en el proceso usado para construir esos componentes.

Excepto por pequeños programas contruidos por programadores solitarios para uso personal, el desarrollo del software, es una actividad organizacional.



La meta de la revolución industrial del software en general y de las tecnologías orientadas a objetos en particular, es dotar de sentido común al desarrollo de software.

La revolución industrial del software involucra un cambio de paradigma, con un efecto sobre arraigados sistemas de valores, estructuras de poder y creencias acerca del rol de los desarrolladores, en relación a los consumidores. Está también motivada por necesidades prácticas que anteriores paradigmas han sido incapaces de resolver, resultando en un desesperado sentimiento de crisis.

La necesidad de software confiable para la era de la Información está intensificando la crisis del software hasta hacer que un cambio de paradigma no sea más una cuestión de conveniencia, sino una cuestión de cuándo y a través de quiénes.

Una de las preocupaciones actuales más urgentes de la industria de la computación es la de crear software y sistemas corporativos más pronto y de más bajo costo. Para hacer un buen uso del poder cada vez mayor de las computadoras, necesitamos un software de mayor complejidad, aparte de más complejo, también es necesario que dicho software sea más confiable. La alta calidad es esencial en el desarrollo del software, ya que una calidad pobre es un desperdicio de dinero.

La necesidad de crear un mejor software se aplica tanto en el interior de la propia industria del software como dentro de las empresas de todo tipo que crean sus propias aplicaciones de computación. Las organizaciones relacionadas con la tecnología de la información necesitan crear y modificar aplicaciones en forma mucho más rápida. Si el desarrollo de aplicaciones tarda de dos a tres años, con un retraso de creación de varios años, las empresas no pueden crear más aplicaciones ni reaccionar ante la competencia de manera rápida. Se pierde la capacidad vital para el cambio dinámico.

La historia muestra algunas consideraciones, sobre la crisis del Software



“El software es escaso, caro, de insuficiente calidad, difícil de planificar y casi imposible de mantener. “

Conferencia de Ingeniería del Software de la OTAN (1968)



“Pocas disciplinas [como la ingeniería de software] adolecen de una brecha tan marcada entre la mejor práctica y la práctica promedio actual. [...] Los principales problemas en grandes proyectos no son técnicos sino de gestión. “

Departamento de Defensa, USA (1987)



“Las herramientas, técnicas y abstracciones convencionales se convierten rápidamente en inadecuadas mientras los sistemas son cada vez mayores y su complejidad aumenta.”

“Software Orientado a Objetos”, Ann Winblad (1993)



“La complejidad de los problemas abordados por el software está creciendo más rápidamente, que nuestra habilidad para desarrollar y mantener ese software. La habilidad para desarrollar y entregar software usable y confiable, dentro del presupuesto y en los plazos acordados continua eludiendo a la mayoría de las organizaciones de software.”

“The Capability Maturity Jod” Software Engineering Institute (1995)



Pero también a través de la historia de la ingeniería, parece surgir un principio: “la gran ingeniería es la ingeniería sencilla”. Las ideas que se tornan muy rebuscadas, inflexibles y problemáticas tienden a ser reemplazadas por otras más nuevas y claras desde el punto de vista conceptual y con sencillez estética.

Cuando el diagrama de un programador parece telaraña, es hora de rediseñar todo el programa. El software corre el riesgo constante de convertirse en rebuscado, inflexible y problemático. Los enlaces entre las diversas partes tienden a multiplicarse si se añaden nuevas características y si cambian las necesidades de los usuarios. De no tener claros los conceptos, los diseñadores sólo producirán intrincadas telarañas.



Las técnicas orientadas a objetos se pueden utilizar como medios para el diseño sencillo de sistemas complejos.

El sistema se puede ver como una **colección de objetos**, donde cada uno de ellos puede llegar a tener varias posibilidades. Las operaciones que modifican el estado son relativamente sencillas.

El término revolución industrial en el software se ha utilizado para describir el paso hacia una era, en la que el software será compilado a partir de componentes de objetos reutilizables, con lo que se crearía una enorme biblioteca de componentes.

Debemos pasar de una era de paquetes monolíticos de software, donde un vendedor “construye” todo un paquete, hasta una era en la que el software sea ensamblado a partir de componentes y paquetes de muchos proveedores (de la misma forma en que las computadoras y los automóviles son ensamblados a partir de componentes de muchos proveedores). Los componentes serán cada vez más complejos desde el punto de vista interno, pero será más sencillo interactuar con ellos. Serán cajas negras donde no podremos mirar al interior.

Componentes reusables = Standard

Ciertamente, el universo del software es caótico hoy en día, con tecnologías orientadas a objetos luchando con tecnologías tradicionales, Ada luchando contra Smalltalk, C++ contra Objective-C, prototipado rápido contra métodos tradicionales, etc. Solo uno debe ganar y ser adoptado por toda la comunidad y cada nuevo contendiente debe competir con los anteriores por el codiciado **título de standard**.

Poniendo el foco en el producto en vez del proceso, un patrón más simple emerge.

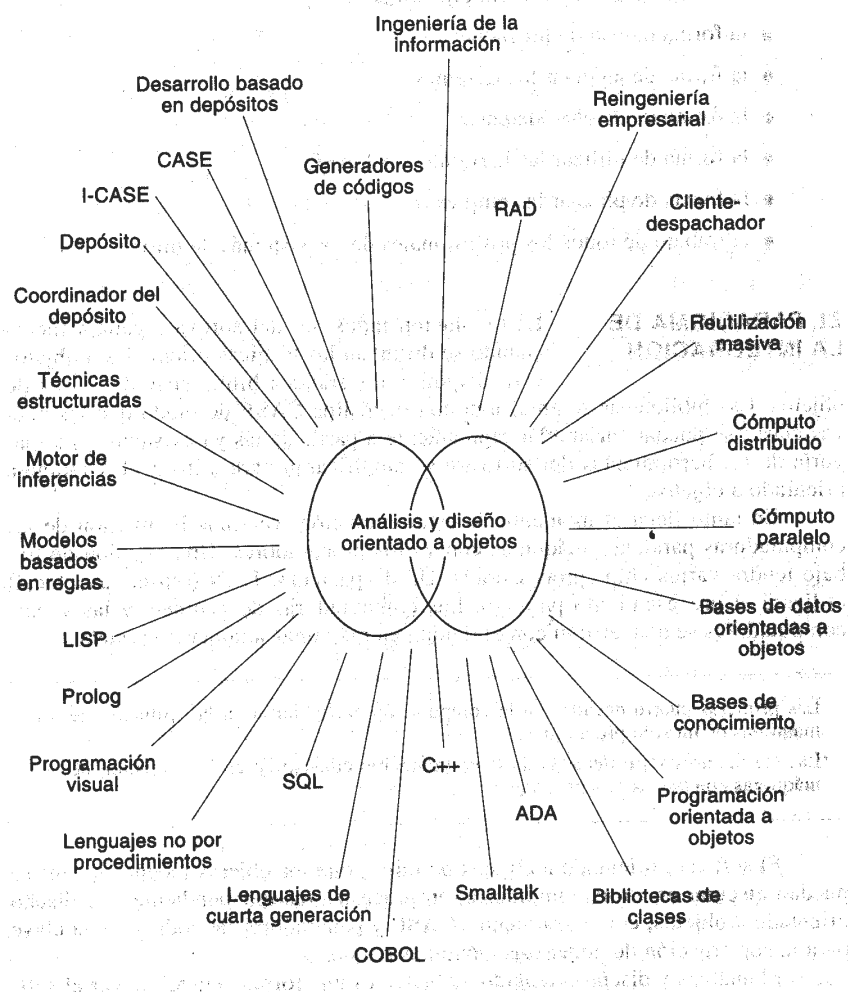
Como una reminiscencia de los diferentes niveles de integración de la ingeniería de hardware, por ejemplo al nivel de plaquetas electrónicas, pueden ensamblarse múltiples placas para obtener soluciones personalizadas de hardware sin necesidad de entender cómo funcionan los cables y chips. Al nivel de chips, los vendedores pueden construir plaquetas a partir de chips, sin necesidad de conocer los detalles de compuertas lógicas y bloques que quienes a su vez son sus proveedores deben conocer para poder construirlos. Cada tecnología encapsula niveles de complejidad, de manera que sus consumidores no necesitan preocuparse en como están implementados los componentes de nivel inferior, sino solamente en cómo deben ser usados para solucionar el problema que tienen entre manos.

El paradigma de la integración

No todo se basa en la programación, el análisis y diseño orientado a objetos es una forma general de ver el software, la que integrará todas las tecnologías, como se muestra en el gráfico. Esta integración provocará una revolución industrial del software.



El modelado y el diseño orientado a objetos es un paradigma de integración, que enlazará todas las herramientas y las técnicas poderosas para la creación de software.



El análisis y diseño orientado a objetos deberá integrar y ser utilizado por las demás tecnologías poderosas del software

Los factores que influyeron en el cambio de paradigma

Existen una serie de factores, que en sus diversos campos o enfoques, influyeron en el cambio de paradigma. Estos son:

► *Factor I: Evolución de la Tecnología*

- » El avance en los medios de comunicación y transporte nos lleva a una economía globalizada en donde el recurso estratégico es la información.
- » La tecnología nos brinda acceso a grandes masas de información en bruto que debe ser refinada para obtener lo relevante.
- » El recurso clave para este proceso es el Software.

- » La sociedad exige más software y depende en mayor medida de él.
- » Los sistemas son cada vez más complejos.
- » Computación ubicua Software como recurso estratégico
 Ambientes distribuidos
 Ambientes en tiempo real

► ***Factor 2: La Administración de Proyectos***

Organizaciones inmaduras:

- » Proceso de desarrollo improvisado
- » Continuamente "combatiendo incendios".
- » Estimaciones poco realistas hacen que se excedan rutinariamente los plazos y presupuestos.
- » No cuentan con bases objetivas para medir la calidad o predecirla.
- » No se miden los impactos de las decisiones sobre la calidad.
- » Las actividades de aseguramiento de la calidad (por ej.: pruebas y revisiones) se restringen o eliminan.

Organizaciones maduras:

- » Proceso de desarrollo standard a aplicar en todos los proyectos.
- » Consenso general sobre la importancia de aplicar un proceso de desarrollo consistente.
- » Existe la infraestructura para dar soporte al proceso de desarrollo.
- » Mejoría continua.
- » Introducción controlada de nuevas tecnologías (proyectos piloto, estudios de riesgo, etc.).
- » Roles y responsabilidades bien definidas.
- » Existen bases objetivas y cuantitativas para juzgar la calidad.
- » Planificación y presupuestación basada en estimaciones realistas según datos históricos.

► ***Factor 3: El Mantenimiento del Software***

Mantenimiento = Correcciones + Adaptaciones + Extensiones

El mantenimiento es una actividad altamente entrópica (tiende a destruir la estructura del sistema).

Causas:

- Altos niveles de acoplamiento: es la dependencia mutua que existe entre diferentes porciones de código de un sistema.
- Efecto secundario: la introducción de un cambio se propaga e introduce otros errores debido a ramificaciones poco obvias. Tarde o temprano, las correcciones no producen ninguna mejoría.

- ▶ Mucho esfuerzo (75%) destinado a mantener viejos sistemas: la fábrica de software envejece.

Ingeniería de software (Un Concepto de actualidad)

El software es un producto de tipo ingenieril, por lo tanto se debe aplicar criterios de ingeniería para obtener un producto de calidad; que sea anticipable, es decir, saber que frente a los mismos estímulos siempre va a dar la misma respuesta; que pueda tener acotados los márgenes de error y que en el caso de que esto ocurra, poder actuar con el menor tiempo de espera posible. Hablamos entonces de Ingeniería de Software.

El software, como todo producto industrial, tendrá un proceso de Desarrollo, acarreando un producto de subprocesos, cada uno de los cuales deja como resultado un subproducto terminado o inconcluso.

El análisis es uno de esas fases que conforman el proceso de desarrollo del software. Otros serán Diseño y algún tipo de construcción.

Hay que recalcar que no se habla de etapas (actividades con una cierta cronología), sino de fases, instancias. Esto implica definir cada uno, lo cual lleva a tener que saber: que productos entrega cada uno de ellos, con qué se entra y con que se sale, que condiciones iniciales hay y cuales condiciones finales se obtienen. Especificando todo esto, las fases pueden combinarse de la manera más conveniente, dependiendo de la organización de desarrollo, el software a desarrollar, etc. El proceso de desarrollo de software deberá adecuarse a las circunstancias particulares de los proyectos en los que se trabaja.

Se pueden clasificar las fases, dentro de varios de los siguientes términos: metodología de desarrollo de tipo cascada, metodologías iterativas y metodologías de tipo incremental. (Temas vistos en el módulo de sistemas II)

Las metodologías de desarrollo en cascada:

{ En realidad no es un proceso de desarrollo, es la típica metodología tradicional en la cual se dice la secuencia de actividades que hay que cumplir, en qué orden y se establece (casi como axioma) que no puede iniciarse una actividad hasta que no se complete la anterior. Tampoco se permiten vueltas hacia atrás.

Las metodologías de tipo iterativo

{ Son aquellos que permiten iterar en cada subproceso, tantas veces como se quiera. Se itera porque a medida que vamos comprendiendo el problema vamos modelando y tenemos que volver atrás para comprender, o cambiar algo que se entendió mal.

Las metodologías de tipo incremental

Está muy ligado al iterativo (en realidad es imposible que viva sin el). Permite ir haciendo las cosas por partes e ir avanzando en el desarrollo de cada una de ellas. Luego se deberá volver atrás para ir completando cada parte y haciendo otras. Esto permite el desarrollo en paralelo de distintas partes.

La ingeniería del software tradicionalmente planteó el desarrollo de software siguiendo alguno de estos tres esquemas (o bien combinándolos de alguna manera). Pero si observamos la realidad, encontraremos que ninguna de esas formas de trabajo aplican formas puras.

Desarrollar software es una actividad caótica debido a que el software mismo lo es por naturaleza. Lo que se tiene que hacer es no imponer un tipo de proceso, sino armarlo en base, a lo que se conoce de cómo se trabaja en la organización. Es decir definir la forma en la que se va a trabajar de acuerdo a cómo se trabaja realmente.

Qué es una metodología

Hay que tener en claro el concepto de metodología para poder hacer la diferenciación entre un modelo, una metodología, una técnica, herramientas, etc.



Entendemos por Metodología a un conjunto de conceptos, técnicas, notaciones y herramientas que ayuden y guíen el proceso de desarrollo de sistemas Informáticos.

Hablamos de:

- ▶ Conceptos básicos con los que se va a trabajar.
- ▶ Técnicas, es decir cómo utilizar esos conceptos para construir los modelos a realizar.
- ▶ Notaciones, es decir ¿cómo vamos a expresar esos modelos?, ¿cómo los vamos a dibujar o escribir?.
- ▶ Herramientas, es decir, ¿con qué cosas vamos a trabajar para construir esos modelos?.

Una metodología está formada por todos esos elementos; no es ninguno de ellos en forma aislada.

Entonces es necesario que el modelo este representado en términos de sistemas de información para que esto sea viable, es decir para que un diseñador pueda traducir esa especificación en algo implementable y ejecutable.



En síntesis una metodología de Ingeniería de software es el acomodo ordenado de las técnicas en un enfoque sistemático para la construcción o adquisición de sistemas de información.



Una técnica es un método estructurado y repetible para lograr una tarea específica.

Mientras que los analistas diseñadores y desarrolladores individuales son responsables del dominio y la ejecución de las técnicas, el gerente de proyectos sirve como la fuerza conductora para ordenar las tareas en una metodología coherente a fin de satisfacer los objetivos del proyecto.

Una buena metodología arma a sus practicantes con un juego de técnicas confiables y repetibles que se adecuan particularmente bien a los problemas que están tratando de resolver. Una metodología balanceada incluye técnicas que le dan a los analistas y diseñadores una amplia cobertura de todos los aspectos que deban modelar, pero les permiten desviar su énfasis de modelado para adaptarse a la tendencia del problema de negocios.

Una buena metodología de análisis o diseño debe tener cinco características principales:

1. Motivar la actividad pretendida
2. Ser completa
3. Ser modificable en su corrección
4. Producir productos contra los que se pueda medir el avance
5. Ser fácilmente aprovechable en la fase subsiguiente.

El Proceso Unificado de desarrollo de Software



Un proceso desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario, en un sistema software

Un proceso define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo. En la ingeniería del software el objetivo es construir un producto software o mejorar uno existente. Un proceso efectivo proporciona normas para el desarrollo eficiente de software de calidad, captura y presenta las mejores prácticas que el estado actual de la tecnología permite. En consecuencia, reduce el riesgo y hace el proyecto más predecible.

Es necesario un proceso que sirva como guía para todos los participantes clientes, usuarios, desarrolladores y directores ejecutivos. No nos sirve ningún proceso antiguo; necesitamos uno que sea el mejor proceso que la industria pueda reunir en este punto de su historia. Necesitamos un proceso que esté ampliamente disponible de forma que todos los interesados puedan comprender su papel en el desarrollo en el que se encuentran implicados.

El problema del software se reduce a la dificultad que afrontan los desarrolladores para coordinar las múltiples cadenas de trabajo de un gran proyecto de software. La comunidad de desarrolladores necesita una forma coordinada de trabajar. Necesita un

proceso que integre las múltiples facetas del desarrollo. Necesita un **método común**, un proceso que:

- ▶ Proporcione una guía para ordenar las actividades de un equipo.
- ▶ Dirija las tareas de cada desarrollador por separado y del equipo como un todo.
- ▶ Especifique los modelos que deben desarrollarse.
- ▶ Ofrezca criterios para el control y la medición de los productos y actividades del proyecto.



Sin embargo, el Proceso Unificado más que un simple proceso, **es un marco de trabajo genérico** que puede especializarse para: una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.

Un proceso de desarrollo de software debería también ser capaz de evolucionar durante muchos años. Ahora bien, durante esta evolución debería limitar su alcance, en un momento del tiempo dado, a consecuencia de las actualizaciones de **las tecnologías, las herramientas, las personas y los patrones de organización, en un momento.**

Tecnologías

El proceso debe construirse sobre las tecnologías, lenguajes de programación, sistemas operativos, computadoras, estructuras de red, entornos de desarrollo, etc., disponibles en el momento en que se va a emplear el proceso. Por ejemplo, hace veinte años el modelado visual no era realmente de uso general. Era demasiado caro y limitaba mucho el grado en el cual el creador del proceso podía establecer el modelado dentro del proceso.

Herramientas

Los procesos y las herramientas deben desarrollarse en paralelo. Las herramientas son esenciales en el proceso. Dicho de otra forma, un proceso ampliamente utilizado puede soportar la inversión necesaria para crear las herramientas que lo soporten.

Personas

Un creador del proceso debe limitar el conjunto de habilidades necesarias para trabajar en el proceso, a las habilidades que los desarrolladores actuales poseen, o apuntar aquellas que los desarrolladores puedan obtener rápidamente. Hoy es posible empotrar en herramientas de software, técnicas que antes requerían amplios conocimientos, como la comprobación de la consistencia en los diagramas del modelo.

Patrones de organización.

Aunque los desarrolladores de software no pueden ser expertos tan independientes como los músicos de una orquesta, están muy lejos de los trabajadores autómatas. El creador del proceso debe adaptar el proceso a las realidades del momento.

Los Analistas de Sistemas del proceso deben equilibrar estos cuatro conjuntos de circunstancias. El creador del proceso debe diseñar, el proceso de forma tal que pueda evolucionar, de igual forma que el desarrollador de software intenta desarrollar un sistema que no solo funcione este año, sino que evolucione en los años siguientes. Sin este equilibrio de tecnologías, herramientas, personas y organización, el uso del proceso sería bastante arriesgado.

No obstante, los verdaderos aspectos definitorios del Proceso Unificado se resumen en tres frases claves:

- ▶ 1) Dirigido por casos de uso,
- ▶ 2) Centrado en la arquitectura,
- ▶ 3) Iterativo e incremental.

Esto es lo que hace único al Proceso Unificado.

El Proceso Unificado, dirigido por casos de uso

Para construir un sistema con éxito debemos conocer lo que sus futuros usuarios necesitan y desean.



El término usuario hace referencia a usuarios humanos y a otros sistemas, que interactúan con el sistema que estamos desarrollando.



Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso representan los requisitos funcionales.

Por ejemplo : un Caso de uso puede ser “Registrando horarios” , “Emitiendo listado de encomiendas”, “Registrando facturas pendientes” “Cancelando reservas”.

Todos los casos de uso juntos constituyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema. Puede decirse que una especificación funcional contesta a la pregunta: ¿Qué debe hacer el sistema? La estrategia de los casos de uso puede describirse añadiendo tres palabras al final de esta pregunta: *¿...para cada usuario?* Estas tres palabras albergan una implicación importante. Nos fuerzan a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno tener.



Sin embargo, **los casos de uso** no son sólo una herramienta para especificar los requisitos de un sistema. También guían su diseño, implementación, y prueba; esto es, *guían el proceso de desarrollo*.

Basándose en el modelo de casos de uso, los desarrolladores luego crean una serie de modelos de diseño e implementación que llevan a cabo, y completan los casos de uso. Los desarrolladores revisan cada uno de los sucesivos modelos para que sean conformes al modelo de casos de uso.



Casos de uso no sólo inician el proceso de desarrollo sino que le proporcionan un hilo conductor. Dirigido por casos de uso quiere decir que el proceso de desarrollo sigue un hilo - avanza a través de una serie de flujo de trabajo que parten de los casos de uso. Los casos de uso se especifican, se diseñan, y los casos de uso finales son la fuente a partir de la cual los ingenieros de prueba construyen sus caso de prueba (programación).



Los casos de uso han sido adoptados casi universalmente para la captura de requisitos de sistemas software en general y de sistemas basados en componentes en particular. Pero los casos de uso son mucho mas que una herramienta para capturar requisitos. Dirigen el procesos de desarrollo en su totalidad.

¿Por qué casos de Uso?

Existen varios motivos por los cuales los casos de uso son buenos, se han hecho populares y se han adoptado universalmente. Las dos razones fundamentales son:

- » (a) Proporcionan un medio sistemático e intuitivo de capturar requisitos funcionales .
- » (b) Dirigen todo el proceso de desarrollo debido a que la mayoría de las actividades como el análisis, diseño y prueba se llevan a cabo partiendo de los casos de uso.
- » (c) Para idear la Arquitectura

Para capturar los requisitos

La captura de los casos de uso, afecta a los usuarios, a los clientes y a los desarrolladores. Los usuarios y los clientes son los expertos en los requisitos. El papel de los desarrolladores es el de facilitar las discusiones y ayudar a los usuarios y a los clientes a comunicar sus necesidades.



El modelo de casos de uso se utiliza para conseguir un acuerdo con los usuarios y clientes sobre qué debería hacer el sistema para los usuarios.

Podemos pensar en el modelo de casos de uso como en una especificación completa de todas las formas posibles de utilizar el sistema (los casos de uso). Esta especificación puede utilizarse como parte del contrato con el cliente. El modelo de casos de uso nos ayuda a delimitar el sistema definiendo todo lo que debe hacer para sus usuarios.

Por ejemplo, en un sistema de información de un cyber, podemos analizar y desarrollar los PROCESOS que tengan que ver con el área de atención al cliente y no de mantenimiento y reparación de maquinarias. Los Caso de uso nos ayudan a tomar la dimensión del trabajo a realizar, en consenso con el cliente.

Para dirigir el proceso

Como hemos dicho, el que un proyecto de desarrollo esté dirigido por los casos de uso significa que progresa a través de una serie de flujos de trabajo que se inician a partir de los casos de uso.

Los casos de uso ayudan a los desarrolladores a:

- ▶ Encontrar las clases (unidad 5). Las clases se recogen de las descripciones de los casos de uso a medida que las leen los desarrolladores, buscando clases que sean adecuadas para la realización de los casos de uso.
- ▶ También nos ayudan a desarrollar interfaces de usuario (unidad 3) que hacen más fácil a los usuarios el desempeño de sus tareas.
- ▶ Los casos de uso ayudan a los jefes de proyecto a planificar, asignar y controlar muchas de las tareas que los desarrolladores realizan. Por cada caso de uso, un jefe de proyecto puede identificar unas cuantas tareas. Cada caso de uso a especificar es una tarea, cada caso de uso a diseñar es una tarea, y cada caso de uso a probar es una tarea. El jefe de proyecto puede incluso estimar el esfuerzo y el tiempo necesario para llevar a cabo esas tareas. Las tareas identificadas a partir de los casos de uso ayudan a los jefes a estimar el tamaño del proyecto y los recursos necesarios.



Los casos de uso son un importante mecanismo para dar soporte a la **trazabilidad** a través de todos los modelos.. Un caso de uso en la captura de requisitos es trazable a su realización en el análisis y en el diseño, a todas las clases participantes en su realización, a componentes (indirectamente), y finalmente, a los casos de prueba que lo verifican.

Para idear la arquitectura y más...

Los casos de uso también nos ayudan a idear la arquitectura. Mediante la selección del conjunto correcto de casos de uso, los casos de uso significativos arquitectónicamente, para llevarlo a cabo durante las primeras iteraciones, podemos implementar un sistema con una arquitectura estable, que pueda utilizarse en muchos ciclos de desarrollo subsiguientes.

Por ejemplo, continuando con el ejemplo del cyber, si trabajamos casos de uso significativos, como atender el cliente, cobrarle, asignar máquina, el resto de los ciclos de trabajo, serán de fácil aplicación, porque se han tomado como base a los casos de uso pilares, a los esenciales, a los mas robustos, dejando para más tarde el análisis de otros casos de uso no tan importantes .

Los casos de uso también se utilizan como punto de partida para escribir el manual de usuario. Ya que cada caso de uso describe una forma de utilizar el sistema, son el punto de partida ideal para explicar cómo puede el usuario interactuar con el sistema.

Mediante la estimación de la frecuencia de uso de los diferentes caminos en los casos de uso, podemos estimar qué caminos necesitan el mejor rendimiento. Esas estimaciones

pueden utilizarse para dimensionar la capacidad de proceso del hardware subyacente o para optimizar el diseño de la base de datos para ciertos usos.



Completar con la lectura del texto de del texto de Jacobson, I., Booch, G. y Rumbaugh, J. *Proceso Unificado de desarrollo de Software*, de la página 38 a la 40.

El Proceso Unificado está centrado en la arquitectura

El papel de la arquitectura de software es parecido al papel que juega la arquitectura en la construcción de edificios. El edificio se contempla desde varios puntos de vista: estructura, servicios, conducción de la calefacción, cañería, electricidad, etc. Esto permite a un constructor ver un imagen completa antes de que comience la construcción. Análogamente, la arquitectura en un sistema software se describe mediante diferentes vistas del sistema en construcción.



El concepto de **arquitectura** incluye los aspectos estáticos y dinámicos más significativos del sistema.

La arquitectura surge de la necesidad de la empresa, cómo la perciben los usuarios y los inversores, y se refleja en los casos de uso.



La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado.

La arquitectura software abarca decisiones importantes sobre:

- » La organización del sistema software.
- » Los elementos estructurales que compondrán el sistema y sus interfaces.
- » La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.



Sin embargo, la arquitectura software está afectada no sólo por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos, y la estética

Por qué es necesaria la arquitectura

Un sistema software grande y complejo requiere una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común.

A medida que los sistemas se hacen más complejos, los problemas de diseño van más allá de los algoritmos y las estructuras de datos para su programación. Para diseñar y especificar una estructura del sistema global surgen nuevos tipos de problemas.

Se necesita una arquitectura para:

- ▶ (a) Comprender el sistema.
- ▶ (b) Organizar el desarrollo.
- ▶ (c) Fomentar la reutilización.
- ▶ (d) Hacer evolucionar el sistema.

Comprensión del sistema

Para que una organización desarrolle un sistema, dicho sistema debe ser comprendido por todos los que vayan a intervenir en él. El hacer que los sistemas modernos sean comprensibles es un reto importante por muchas razones:

- ▶ Abarcan un comportamiento complejo.
- ▶ Operan en entornos complejos.
- ▶ Son tecnológicamente complejos.
- ▶ A menudo combinan computación distribuida, productos y plataformas comerciales.
- ▶ Deben satisfacer demandas individuales y de la organización.

Por otra parte, estos factores cambian constantemente. Todo esto añade dificultad potencial para comprender la situación.

Organización del desarrollo

Cuanto mayor sea la organización del proyecto software, mayor será la sobrecarga de comunicación entre los desarrolladores para intentar coordinar sus esfuerzos. Esta sobrecarga se incrementa cuando el proyecto está geográficamente disperso. Dividiendo el sistema en subsistemas, con las interfaces claramente definidas y con un responsable o un grupo de responsables establecido para cada subsistema. Una "buena" arquitectura es la que define explícitamente estas interfaces, haciendo que sea posible la reducción en la comunicación. Una interfaz bien definida "comunica" eficientemente a los desarrolladores de ambas partes qué necesitan saber sobre lo que los otros equipos están haciendo.

Fomento de la reutilización

Los desarrolladores capaces de reutilizar conocen el **dominio del problema** y qué componentes específicos como adecuados, la arquitectura. Los desarrolladores piensan en cómo conectar esos componentes para cumplir con los requisitos del sistema, realizar el modelo de casos de uso. Cuando tienen disponibles componentes reutilizables, los usan, los componentes software reutilizables están diseñados y probados para encajar, y así el tiempo de construcción y el costo son menores. El resultado es predecible.

Una buena arquitectura ofrece a los desarrolladores un andamio estable sobre el que trabajar. El papel de los arquitectos es definir ese andamiaje y los subsistemas reutilizables que el desarrollador pueda utilizar. Se obtienen subsistemas reutilizables diseñándolos con cuidado para que puedan ser utilizados conjuntamente.

Evolución del sistema

Si hay algo de lo que podemos estar seguros, es de que cualquier sistema de un tamaño considerable evolucionará. Evolucionará incluso aunque aún esté en desarrollo. Más tarde, cuando esté en uso, el entorno cambiante provocará futuras evoluciones. Hasta que esto ocurra, el sistema debe ser fácil de modificar; esto quiere decir que los desarrolladores deberían ser capaces de modificar partes del diseño e implementación sin tener que preocuparse por los efectos inesperados que puedan tener repercusión en el sistema. En la mayoría de los casos, deberían ser capaces de implementar nuevas funcionalidades (es decir, casos de uso) en el sistema sin tener que pensar en un impacto dramático en el diseño e implementación existentes.

En otras palabras, el sistema debe ser en sí mismo flexible a los cambios o tolerante a los cambios.



Completar con la lectura del texto de Jacobson, I., Booch, G. y Rumbaugh, J. *Proceso Unificado de desarrollo de Software*, de la página 41 a la 45.

El Proceso Unificado es iterativo e incremental

El desarrollo de un producto software comercial supone un gran esfuerzo que puede durar varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes mas pequeñas o mini proyectos.

Cada mini proyecto es una **iteración** que resulta en un incremento.



Las iteraciones hacen referencia a pasos en el flujo de trabajo. Para una efectividad máxima, las iteraciones deber estar controladas; esto es, deben seleccionarse y ejecutarse de una forma planificada. Es por esto, por lo que son *mini proyectos*.

Las iteraciones sucesivas se construyen sobre los modelos (unidad 4) o artefactos de desarrollo tal como quedaron al final de la última iteración. Al ser mini proyectos, comienzan con los casos de uso y continúan a través del trabajo de desarrollo subsiguiente -análisis, diseño, implementación y prueba-, que termina convirtiendo en código ejecutable, los casos de uso que se desarrollaban en la iteración.



Los incrementos, son los que determinan el crecimiento del producto.

Por supuesto, un incremento no necesariamente es aditivo. Especialmente en las primeras fases del ciclo de vida, los desarrolladores pueden tener que reemplazar un diseño superficial por uno más detallado o sofisticado. En fases posteriores, los incrementos son típicamente aditivos.

Por ejemplo, en cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfagan los casos de uso. Si una iteración cumple con sus objetivos como suele suceder, el desarrollo continúa con la siguiente iteración. Cuando una iteración no cumple sus objetivos, los desarrolladores deben revisar sus decisiones previas y probar con un nuevo enfoque. Por supuesto, en la medida en que se añadan iteraciones o se altere el orden de las mismas por problemas inesperados, el proceso de desarrollo consumirá más esfuerzo y tiempo. Son muchos los beneficios de un proceso iterativo controlado.

Relación entre los tres aspectos del Proceso de desarrollo Unificado

Estos conceptos – los de desarrollo dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental- son de igual importancia. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración. La eliminación de una de las tres ideas reduciría drásticamente el valor del Proceso Unificado. Es como un taburete de tres patas. Sin una de ellas, el taburete se cae.



Completar con la lectura del texto de Jacobson, I.; Booch, G. y Rumbaugh, J.: *Proceso Unificado de desarrollo de Software*, páginas 61 a 70.

Ahora que hemos presentado los tres conceptos claves, es el momento de mirar, al proceso en su totalidad, su ciclo de vida, artefactos (modelos), flujos de trabajo, fases e iteraciones

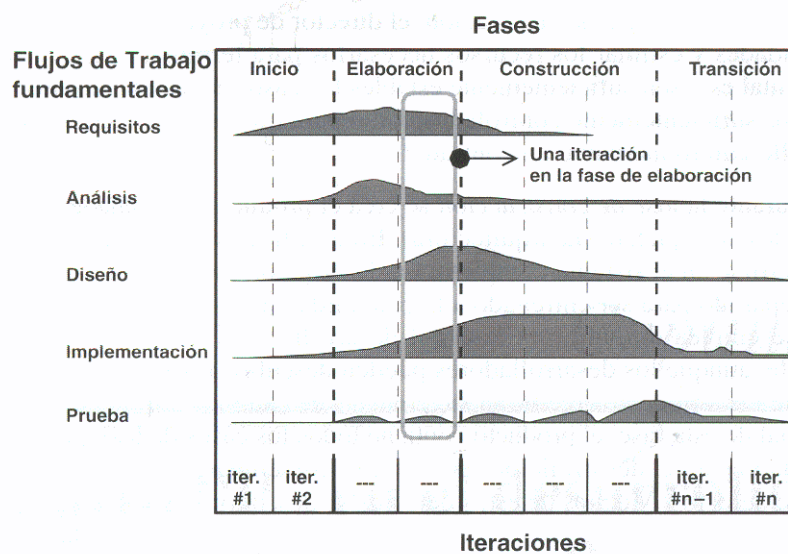
El objetivo del Proceso Unificado es guiar a los desarrolladores en la implementación y distribución eficiente de sistemas que se ajusten a las necesidades de los clientes. La eficiencia se mide en términos de costo, calidad, y tiempo de desarrollo. El paso desde la determinación de las necesidades del cliente hasta la implementación no es trivial. En primer lugar, las necesidades del cliente no son fáciles de discernir. Esto nos obliga a que tengamos algún modo de capturar las necesidades del usuario de forma que puedan comunicarse fácilmente a todas las personas implicadas en el proyecto. Después, debemos ser capaces de diseñar una implementación funcional que se ajuste a esas necesidades. Por último, debemos verificar que las necesidades del cliente se han cumplido mediante la prueba del sistema. Debido a esta complejidad, el proceso se describe como una serie de flujos de trabajo que construyen el sistema gradualmente.

Ciclo de Vida del Proceso Unificado

Un proceso es un conjunto de pasos ordenados parcialmente para alcanzar un objetivo. En la ingeniería del software, el objetivo es entregar un producto software que satisfaga las necesidades del usuario, de forma eficiente y predecible dentro de planificaciones y presupuestos estipulados.

El Proceso unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema.

Cada ciclo consta de cuatro fases: inicio, elaboración, construcción y transición. Cada fase se subdivide a su vez en iteraciones.



Fases dentro de un ciclo

Cada ciclo se desarrolla a lo largo del tiempo. Este tiempo, a su vez, se divide en cuatro fases. A través de una secuencia de modelos visualizan lo que está sucediendo en esas fases. Dentro de cada fase, los desarrolladores pueden descomponer adicionalmente el trabajo en iteraciones con sus incrementos resultantes.

La Figura anterior muestra en la columna izquierda los flujos de trabajo: requisitos, análisis, diseño, implementación, y prueba. Las curvas son una aproximación, de hasta donde se llevan a cabo los flujos de trabajo en cada fase. Recuerden que cada fase se divide normalmente en iteraciones o mini proyectos.

Una iteración típica pasa por los cinco flujos de trabajo, como se muestra en la Figura.



Una fase es el intervalo de tiempo entre dos hitos importantes del proceso durante la cual se cumple un conjunto bien definido de objetivos, se completan artefactos y se toman decisiones sobre si pasar a la siguiente fase.

El proceso Unificado consta de 4 fases:

I. Iniciación: Establece la planificación del proyecto.

2. Elaboración: Establece un plan para el proyecto y una arquitectura correcta.
3. Construcción: Desarrollar el sistema.
4. Transición: Proporciona el sistema a sus usuarios finales.

Las fases de iniciación y elaboración incluyen las actividades de análisis, diseño del ciclo de vida del desarrollo; la construcción y la transición constituyen su producción.

Dentro de cada fase hay varias iteraciones. Una *iteración* representa un ciclo de desarrollo completo, desde la captura de requisitos en el análisis hasta la implementación y pruebas, que produce como resultado la entrega al cliente o la salida al mercado de un proyecto ejecutable.

Cada fase e iteración se centra en disminuir algún riesgo y concluye con un hito bien definido. La revisión de hitos es el momento adecuado para evaluar cómo se están satisfaciendo los objetivos y si el proyecto necesita ser reestructurado de alguna forma para continuar.

Si resumiéramos en pocas palabras el objetivo de cada **fase**, podríamos expresar:

Durante **la fase de inicio**, se desarrolla una descripción del producto final a partir de una buena idea y se presenta el análisis de negocio para el producto. Esencialmente, esta fase responde a las siguientes preguntas:

<i>¿Cuáles son las principales funciones del sistema para sus usuarios más importantes?</i>	Esta pregunta se encuentra en el modelo de casos de uso
<i>¿Cómo podría ser la arquitectura del sistema?</i>	Esta pregunta esta en un esbozo (a partir del modelos de casos de Uso) de los subsistemas mas importantes.
<i>¿Cuál es el plan de proyecto y cuánto costará desarrollar el producto?</i>	Se deben identificar los riesgos, y planificar las fase de elaboración

Durante la fase de iniciación, se establece la planificación del proyecto y se delimita su alcance. La planificación del proyecto incluye los criterios de éxito, la evaluación del riesgo, estimaciones de recursos que se necesitarán y un plan de fases que muestre la planificación de los hitos principales. Durante la iniciación, es frecuente crear un prototipo ejecutable que sirva para probar los conceptos.

Al final de la fase de inicio se examinan los objetivos del ciclo de vida del proyecto y se decide si proceder con el desarrollo del sistema.

Durante **la fase de elaboración**, se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema. La relación entre la arquitectura del sistema y el propio sistema es primordial.

Por tanto, la arquitectura se expresa en forma de vistas de todos los modelos del sistema, los cuales juntos representan al sistema entero. Esto implica que hay vistas arquitectónicas del modelo de casos de uso, del modelo de análisis, del modelo de diseño, del modelo de implementación y modelo de despliegue.

Al final de la fase de elaboración, el director de proyecto está en disposición de planificar las actividades y estimar los recursos necesarios para terminar el proyecto. Aquí la cuestión fundamental es: ¿son suficientemente estables los casos de uso, la arquitectura y el plan, y están los riesgos suficientemente controlados como para que seamos capaces de comprometernos al desarrollo entero mediante un contrato?

Los objetivos de la fase de elaboración son analizar el dominio del problema, establecer una base arquitectónica sólida, desarrollar el plan del proyecto y eliminar los elementos de más alto riesgo del proyecto. Las decisiones arquitectónicas deben tomarse con una comprensión del sistema global. Esto implica que se deben describir la mayoría de los requisitos del sistema. Para verificar la arquitectura, se implementa un sistema que demuestre las distintas posibilidades de la arquitectura y ejecute los casos de uso significativos.

Al final de la fase de elaboración se examinan el alcance y los objetivos del sistema, la elección de la arquitectura y la resolución de los riesgos más grandes, y se decide si se debe pasar a la construcción.

Durante **la fase de construcción** se crea el producto. En esta fase, la línea base de la arquitectura crece hasta convertirse en el sistema completo. La descripción evoluciona hasta convertirse en un producto preparado para ser entregado a la comunidad de usuarios. El grueso de los recursos requeridos se emplea durante esta fase del desarrollo. Sin embargo, la arquitectura del sistema es estable, aunque los desarrolladores pueden descubrir formas mejores de estructurar el sistema, ya que los arquitectos recibirán sugerencias de cambios arquitectónicos de menor importancia. Al final de esta fase, el producto contiene todos los casos de uso que la dirección y el cliente han acordado para el desarrollo de esta versión. Sin embargo, puede que no está completamente libre de defectos. Muchos de estos defectos se descubrirán y solucionarán durante la fase de transición.

Durante la fase de construcción, se desarrolla de forma iterativa e incremental un producto completo que está preparado para la transición hacia la comunidad de usuarios. Esto implica describir los requisitos restantes y los criterios de aceptación, refinando el diseño y completando la implementación y las pruebas del software.

Al final de la fase de construcción se decide si el software, los lugares donde se instalará y los usuarios están todos preparados para empezar a funcionar.

La **fase de transición** cubre el periodo durante el cual el producto se convierte en versión beta. En la versión beta un número reducido de usuarios con experiencia prueba el producto e informa de defectos y deficiencias. Los desarrolladores corrigen los problemas e incorporan algunas de las mejoras sugeridas en una versión general dirigida a la totalidad de la comunidad de usuarios. La fase de transición conlleva actividades como la fabricación, formación del cliente, el proporcionar una línea de ayuda y asistencia, y la corrección de los defectos que se encuentren tras la entrega.

Durante la fase de transición, el software se despliega en la comunidad de usuarios. Una vez que el sistema ha sido puesto en manos de los usuarios finales, a menudo aparecen cuestiones que requieren un desarrollo adicional para ajustar el sistema, corregir algunos problemas no detectados o finalizar algunas características que habían

sido pospuestas. Esta fase comienza normalmente con una versión beta del sistema, que luego será reemplazada con el sistema de producción.

Al final de la fase de transición se decide si se han satisfecho los objetivos del ciclo de vida del proyecto, y se determina si se debería empezar otro ciclo de desarrollo. Este es también un punto en el que se asimilan las lecciones aprendidas en el proyecto para mejorar el proceso de desarrollo, que será aplicado al próximo proyecto.

Iteraciones



Una iteración es un ciclo completo de desarrollo que produce una versión (interna o externa) de un producto ejecutable, además de constituir un subconjunto del producto final en desarrollo, que luego se irá incrementando de iteración en iteración hasta convertirse en el sistema final.

Cada iteración pasa a través de varios flujos de trabajo del proceso, aunque con un énfasis diferente en cada uno de ellos, dependiendo de la fase en que se encuentre. Durante la iniciación, el interés se orienta hacia el análisis y el diseño. Durante la construcción, la actividad central es la implementación, y la transición se centra en el despliegue.

Ciclos de desarrollo

El paso a través de las cuatro fases principales constituye un ciclo de desarrollo, y produce una generación de software. La primera pasada a través de las cuatro fases se denomina ciclo de desarrollo inicial. A menos que acabe la vida del producto, un producto existente evolucionará a la siguiente generación repitiendo la misma secuencia de inicio, elaboración, construcción y transición. Esta es la evolución del sistema, así que los ciclos de desarrollo después del ciclo inicial son los ciclos de evolución.

Flujos de trabajo del proceso

El Proceso Unificado consta de nueve flujos de trabajo.

1. Modelado del negocio	Describe la estructura y la dinámica de la organización.
2. Requisitos	Describe el método basado en casos de uso para extraer los requisitos.
3. Análisis y diseño	Describe las diferentes vistas arquitectónicas
4. Implementación	Tiene en cuenta el desarrollo de software, la prueba de unidades y la integración.
5. Pruebas	Describe los casos de pruebas, los procedimientos y las métricas para evaluación de defectos
6. Despliegue	Cubre la configuración del sistema entregable
7. Gestión de configuración	Controla los cambios y mantiene la integridad de los artefactos de un proyecto.
8. Gestión del Proyecto	Describe varias estrategias de trabajo en un proceso iterativo.

9. Entorno

Cubre la infraestructura necesaria para desarrollar un sistema.

Dentro de cada flujo de trabajo del proceso hay un conjunto de artefactos y actividades relacionados.



Un *artefacto* es algún documento, informe o ejecutable que produce, se manipula o se consume.



Una *actividad* describe las tareas (pasos de concepción, realización y revisión) que llevan a cabo los trabajadores para crear o modificar artefactos, junto con las técnicas y guías para ejecutar las tareas, incluyendo quizá herramientas para ayudar a automatizar algunas de ellas.

Modelos



Los modelos son el tipo de artefacto más importante en el Proceso Unificado. Un modelo es una simplificación de la realidad, creada para comprender mejor el sistema que se está creando.

En el Proceso, hay nueve modelos que en conjunto cubren todas las decisiones importantes implicadas en la visualización, especificación, construcción y documentación de un sistema con gran cantidad de software.

1. **Modelo del negocio:** Establece una abstracción de la organización
2. **Modelo del dominio:** Establece el contexto del sistema.
3. **Modelo de casos de uso:** Establece los requisitos funcionales del sistema
4. **Modelo de análisis (opcional):** Establece un diseño de las ideas.
4. **Modelo de diseño:** Establece el vocabulario del problema solución.
5. **Mod. del proceso (opcional):** Establece los mecanismos de concurrencia y sincronización del sistema.
6. **Modelo de despliegue:** Establece la topología hardware sobre la cual se ejecutará el sistema.
7. **Modelo de implementación:** Establece las partes que se utilizarán para ensamblar y hacer disponible el sistema físico
8. **Modelo de pruebas:** Establece las formas de validar y verificar el sistema.

UML



El UML (lenguaje Unificado de Modelado), es una herramienta, que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas

Nos ofrece un modo estándar de visualizar, especificar, construir, documentar y comunicar los artefactos de un sistema muy basado en el software, un lenguaje como éste, debe utilizarse en el contexto de un proceso de software completo.

UML es un medio, y no un fin. El objetivo final es una aplicación software robusta, flexible y escalable.

El UML está compuesto de diversos diagramas gráficos, que se combinan para conformar diagramas. Debido a que UML es un lenguaje, cuenta con reglas para combinar tales elementos. La finalidad de los diagramas es presentar diversas perspectivas de un sistema al cual se conoce como modelo. El modelo UML de un sistema describe lo que supuestamente hará el sistema, pero no dice como implementar dicho sistema.

**Nota**

Cada una de las unidades que se desarrollan en este módulo, se basan completamente en la metodología presentada por Ivar Jacobson, Grady Booch, James Rumbaugh, por lo que es recomendable el uso de su bibliografía *El Proceso Unificado de desarrollo de Software*, como complemento.

Resumen de la unidad...

Los puntos que caracterizan el concepto “Orientación a Objeto”

- ▶ Es un cambio cultural más que un cambio tecnológico.
- ▶ Implica dejar de reinventar la rueda; hay que reutilizarla para así lograr mayor productividad.
- ▶ Significa construir aplicaciones a partir de componentes reutilizables y modelar aplicaciones de una manera más natural.
- ▶ Implica una reorganización del mercado de software: productores de sistemas y productores de componentes.
- ▶ Los objetos son unidades que combinan datos con las operaciones para accederlos, por lo que presentan un comportamiento bien definido.
- ▶ Los objetos representan entidades (tangibles o abstractas) observables directamente en el problema a ser resuelto.
- ▶ Esta visión es consistente con la forma en que las personas percibimos el mundo que nos rodea.



Beneficios de la Tecnología Orientada a objetos

- ▶ **El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel.** El encapsulado oculta los detalles y hace que las clases complejas sean fáciles de utilizar.
- ▶ **Nuevos mercados para el software.** Las compañías de software deben proporcionar bibliotecas de software para áreas específicas, que se adapten con facilidad a las necesidades de la organización que las utiliza.

- ▶ **Un diseño más rápido.** Las aplicaciones se crean a partir de componentes ya existentes. Muchos de los componentes están contruidos de modo que se puedan adaptar para un diseño particular.
- ▶ **Diseño de mayor calidad.** Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados, que han sido verificados y pulidos varias veces.
- ▶ **Integridad.** Las estructuras de datos sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-despachador y los sistemas distribuidos, en los que usuarios desconocidos podrían intentar el acceso al sistema.
- ▶ **Programación más sencilla.** Los programas se juntan a partir de piezas pequeñas, cada una de las cuales, en general, se puede crear fácilmente. El programador crea un método para una clase a la vez. El método cambia el estado de los objetos en formas que suelen ser sencillas cuando se les considera en sí mismas.
- ▶ **Mantenimiento más sencillo.** El programador encargado del mantenimiento cambia un método de clase a la vez. Cada clase efectúa sus funciones independientemente de las demás.
- ▶ **Modelado más realista.** El análisis OO modela la empresa o área de aplicación de manera que sea lo más cercana posible a la realidad de lo que se logra con el análisis convencional. El análisis se traduce de manera directa en el diseño y la implantación. En las técnicas convencionales, el paradigma cambia cuando pasamos del análisis al diseño y del diseño a la programación. Con las técnicas OO, el análisis, diseño e implantación utilizan el mismo paradigma y lo afinan de manera sucesiva.
- ▶ **Mejor comunicación entre los profesionales de los sistemas de información y los empresarios.** Los empresarios comprenden más fácilmente el paradigma OO. Piensan en términos de eventos, objetos y políticas empresariales que describen el comportamiento de los objetos. Las metodologías OO apoyan una mejor comprensión, ya que los usuarios finales y los creadores de software comparten un modelo común.
- ▶ **Una interfaz de pantalla sugestiva para el usuario.** Hay que utilizar una interfaz usuario gráfica, de modo que el usuario apunte a iconos o elementos de un menú desplegado, relacionados con los objetos. En determinadas ocasiones, el usuario puede, de hecho, ver un objeto en la pantalla. Ver y apuntar es más fácil que recordar y escribir.
- ▶ **Imágenes, video y expresión.** Se almacenan objetos binarios de gran tamaño (BLOB) que representan imágenes, video, expresiones, texto sin formato o algunos otros flujos de bits de gran tamaño.
- ▶ **Independencia del diseño.** Las clases están diseñadas para ser independientes del ambiente de plataformas, hardware y software. Utilizan solicitudes y respuestas con formato estándar. Esto les permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de redes, interfaces usuario gráficas, etc. El creador de software no tiene que preocuparse por el ambiente o esperar a que éste se especifique.
- ▶ **Interacción.** El software de varios proveedores puede funcionar como conjunto. Un proveedor utiliza clases de otros. Existe una forma estándar de localizar clases e interactuar con ellas.

- ▶ **Computación de distribución masiva.** Las redes a nivel mundial utilizarán directorios de software de objetos accesibles. El diseño orientado a objetos es la clave para la computación de distribución masiva. Las clases de una máquina interactuarán con las de algún otro lugar sin saber donde residen tales clases. Ellas envían y reciben mensajes 00 en formato estándar.
- ▶ **Mayor nivel de automatización de las bases de datos.** Las estructuras de datos en las bases de datos 00 están ligadas a métodos que llevan a cabo acciones automáticas. Una base de datos 00 tiene integrada una *inteligencia*, en forma de métodos, en tanto que una base de datos de relación básica no.

PDUS

El objetivo del Proceso Unificado es guiar a los desarrolladores en la implementación y distribución eficiente de sistemas que se ajusten a las necesidades de los clientes. Es un conjunto de pasos ordenados parcialmente para alcanzar un objetivo. En la ingeniería del software, el objetivo es entregar un producto software que satisfaga las necesidades del usuario, de forma eficiente y predecible dentro de planificaciones y presupuestos estipulados .

El Proceso unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema.

Cada ciclo consta de cuatro fases: inicio, elaboración, construcción y transición. Cada fase se subdivide a su vez en iteraciones.

El proceso Unificado consta de 4 fases:

- » Iniciación: Establece la planificación del proyecto.
- » Elaboración: Establece un plan para el proyecto y una arquitectura correcta.
- » Construcción: Desarrollar el sistema.
- » Transición: Proporciona el sistema a sus usuarios finales.

Flujos de trabajo del proceso

El Proceso Unificado consta de nueve flujos de trabajo.

» Modelado del negocio	Describe la estructura y la dinámica de la organización.
» Requisitos	Describe el método basado en casos de uso para extraer los requisitos.
» Análisis y diseño	Describe las diferentes vistas arquitectónicas
» Implementación	Tiene en cuenta el desarrollo de software, la prueba de unidades y la integración.
» Pruebas	Describe los casos de pruebas, los procedimientos y las métricas para evaluación de defectos
» Despliegue	Cubre la configuración del sistema entregable

-
- » **Gestión de Config.** Controla los cambios y mantiene la integridad de los artefactos de un proyecto.
 - » **Gestión del Proyecto** Describe varias estrategias de trabajo en un proceso iterativo.
 - » **Entorno.** Cubre la infraestructura necesaria para desarrollar un sistema.
-

Autoevaluación

1. Realice una lectura del siguiente enunciado, y teniendo en cuenta el concepto de Casos de Uso, intente descubrir y describir algunos de ellos.

Una empresa de colectivos emite bonos para viajes regulares a docentes y estudiantes. Para sacar el abono, los usuarios deben presentar un certificado de domicilio y comprobante de docente o alumno, además se llena una solicitud de abono donde constan los datos personales. Con esta solicitud se lo registra al usuario, también se registra el abono solicitado en otro almacenamiento aclarando el n° de viajes autorizados y se le entrega al cliente el n° de abono.

El área cobros receipta el pago y emite la factura que se le entrega al usuario. Una vez recibido el pago, se lo registra en el almacenamiento de abonos. Del chofer se recibe un listado de los números de abono que recibió en el colectivo y con esto se realiza la descarga correspondiente del registro. Diariamente se emite un listado de facturación al Contador.

2. Busque dos ejemplos, en los que se demuestre el concepto de Iterativo e incremental (trabaje ejemplos cotidianos, fuera del ámbito metodológico de sistemas).
3. Investigue la evolución en el tiempo de las diferentes metodologías y Ciclos de Vida (de Proyectos de Sistemas), que se han presentado, o que conviven actualmente.

► Unidad II

Captura de Requisitos: “Enumerar los requisitos candidatos”, “Comprender el contexto del sistema”.

Objetivos

- Entender la estructura y dinámica de la organización en la cual se desarrollará el sistema.
- Comprender los problemas reales en la organización e identificar potenciales mejoras.
- Asegurar que los clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización.
- Derivar los requerimientos, para desarrollar y analizar los sistemas de información que soportará la organización.

Contenidos de la Unidad

Visión general de la captura de requisitos

1- Enumerar los requisitos candidatos

2- Comprender el contexto del sistema

Comprensión del contexto del sistema mediante Modelo del Dominio

Comprensión del contexto del sistema mediante un Modelo del Negocio

Modelo de Caso de uso del Negocio

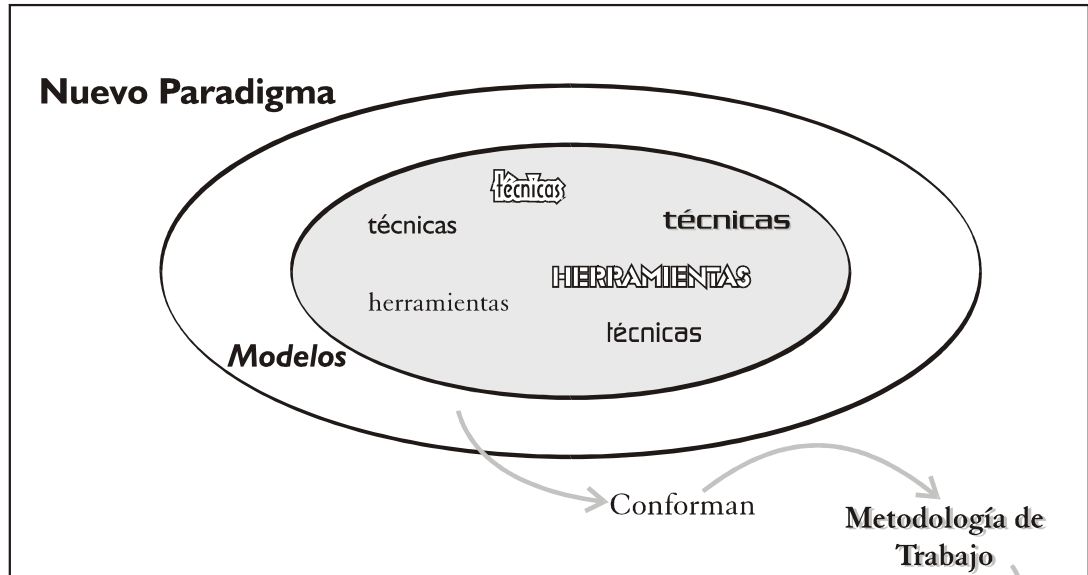
Diferentes categorías de Caso de uso de Negocio.

¿Están los Caso de uso del negocio relacionados siempre con actores?

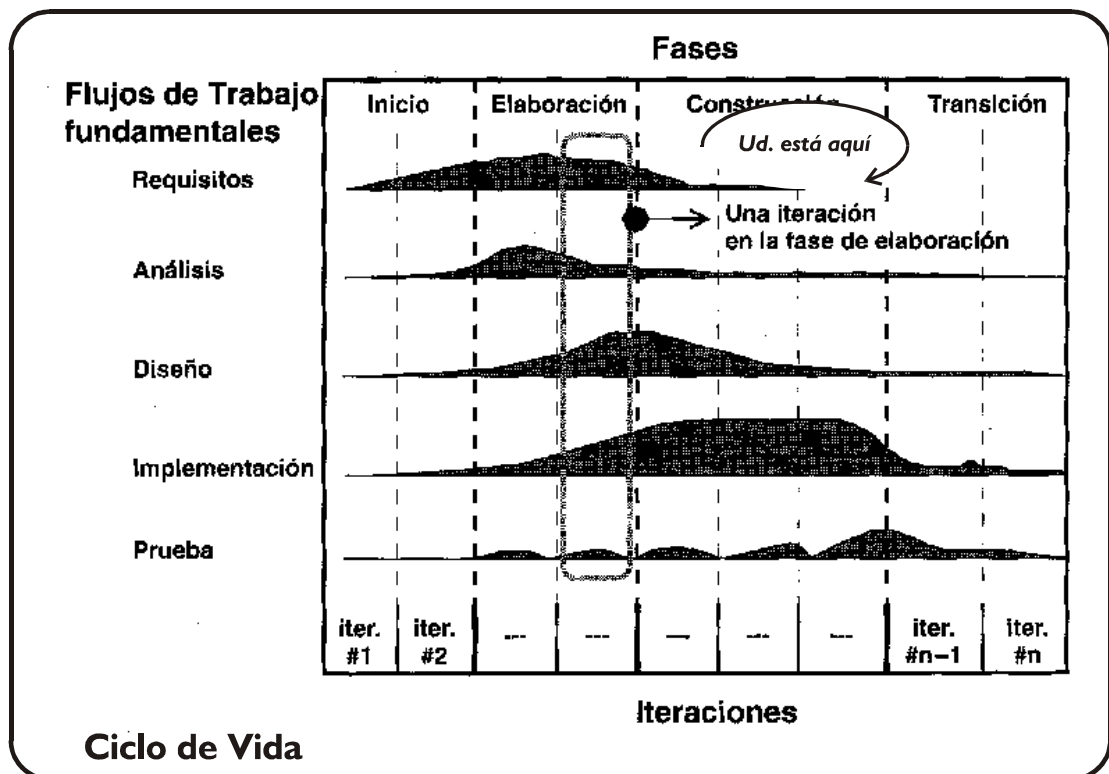
Vistas del negocio. Nombre de un Caso de uso. Estructura de un Caso de uso Actor.

Diagrama de Caso de uso en el modelo de UC de Negocio

Mapa Conceptual



Proceso Unificado de Desarrollo de Software



Introducción

Ahora que comprendemos qué es el Proceso de desarrollo Unificado presentaremos y describiremos cada flujo de trabajo en detalle, en esta unidad (II) y en la próxima unidad (III) se tratará, la captura de Requisitos.

La descripción separada de los flujos de trabajo fundamentales, uno detrás de otro, dan la impresión de que el proceso de desarrollo de software (en una mirada generalista), del comienzo al fin del proyecto, pasa por una secuencia de flujos de trabajo sólo una vez. Se podría llegar a pensar que los flujos de trabajo fundamentales son un proceso de una pasada, como el antiguo proceso en cascada, pero no es así, se recorren los flujos una vez por cada iteración, no una vez por el proyecto completo.

Se describen los flujos de trabajo en unidades separadas, dentro de este material, solamente con fines pedagógicos, pero se hace referencia permanentemente en el texto que la tarea no es secuencial, sino iterativa incremental.

Visión general de la captura de requisitos

La mayoría de las aplicaciones de software ya no son más aplicaciones construidas programando por medio de asistentes, por aficionados a las computadoras. Debemos entender que actualmente es necesario ayudar a las personas en sus tareas diarias, por ello el nuevo estándar es tratar de comprender el dominio del negocio antes, o en paralelo con el desarrollo de proyecto de ingeniería de software.

El modelado del negocio y la definición del sistema no solo son de interés para el área de sistemas, sino también de todos los involucrados en el desarrollo del negocio, desde los ejecutivos hasta los operativos y usuarios finales.

Pero aún teniendo mucha importancia, a veces no es recomendable el modelado de negocios para todos los proyectos de ingeniería de software. El modelado agrega más valor cuando existen muchas personas directamente involucradas, en el uso del futuro sistema de información.

El modelado del negocio puede tener diferentes alcances dependiendo del contexto y las necesidades, se puede desear construir un mapa simple de la organización y sus procesos, para obtener una mejor comprensión de cuales son los requerimientos de la aplicación que se está construyendo, es decir el modelado de negocio es parte de un proyecto de ingeniería del software, realizado durante la fase de inicio. También se construye si se están trabajando aplicaciones con el propósito principal de organizar y presentar información, se puede elegir construir un modelo de negocio de esta información. Por último cuando se está construyendo un sistema grande, el modelado de negocio servirá como entrada a varios proyectos de ingeniería de software.

Los modelos de negocios ayudarán a encontrar requerimientos funcionales, así como también servirán como entrada para construir la arquitectura de la familia de aplicaciones.

Cada proyecto software es diferente. Esta singularidad proviene de las diferencias en el tipo de sistema, en el cliente, en la organización de desarrollo, en la tecnología, etc. De igual forma, hay diferentes puntos de partida para la captura de requisitos. Tomaremos uno de estos puntos de partida, el de la "vaga noción", es decir el cliente tiene una vaga noción de lo que debe hacer el proyecto

Este flujo de trabajo incluye los siguientes pasos:

1. Enumerar los requisitos candidatos. (Temas a desarrollar en esta unidad)
2. Comprender el contexto del sistema. (Temas a desarrollar en esta unidad)
3. Capturar requisitos funcionales. (Temas a desarrollar en la siguiente unidad)
4. Capturar requisitos no funcionales. (Temas a desarrollar en la siguiente unidad)

Durante la vida del sistema, los clientes, usuarios, analistas y desarrolladores aparecen con muchas buenas ideas que podrían convertirse en verdaderos requisitos. Mantener una lista de estas ideas, que consideramos como un conjunto de requisitos candidatos hace que podamos decidir la implementación para una versión futura del sistema.

1- Enumerar los requisitos candidatos

Esta lista de características crece a medida que se añaden nuevos elementos y mengua, cuando algunas características se convierten en requisitos y se transforman en casos de uso. La lista de características se utiliza sólo para la planificación del trabajo.

Cada característica tiene un nombre corto y una breve explicación o definición, información suficiente para poder hablar de ella durante la planificación del producto. Cada característica tiene también un conjunto de valores de planificación que podríamos incluir:

Ideas	Estado	Costo	Prioridad	Nivel de riesgo
Ej Estadísticas para la toma de decisión. Informes y Gráficos	<input type="checkbox"/> Propuesto <input type="checkbox"/> Aprobado <input type="checkbox"/> Incluido	Tipo de recursos Horas - personal	<input type="checkbox"/> Critico <input type="checkbox"/> Importante <input type="checkbox"/> Secundario	<input type="checkbox"/> Critico <input type="checkbox"/> Significativo <input type="checkbox"/> Ordinario

Estos valores se utilizan para estimar el tamaño del proyecto y para decidir como dividir el proyecto en una secuencia de iteraciones.

2- Comprender el contexto del sistema

Para capturar los requisitos correctos y para construir el sistema correcto, los desarrolladores claves requieren un firme conocimiento del contexto en el que se emplaza el sistema.

Hay por lo menos dos aproximaciones para expresar el contexto de un sistema en una forma utilizable para desarrolladores de software:

- A - Modelado del dominio
- B - Modelado del negocio.

► *A - Comprensión del contexto del sistema mediante Modelo del Dominio*

» *¿Qué es un modelo del dominio?*



Un **modelo del dominio** describe los conceptos importantes del contexto como “objetos del dominio”, y enlaza estos objetos unos con otros.

La identificación y la asignación de un nombre para estos objetos nos ayuda a desarrollar un glosario de términos que permiten comunicarse mejor con todos los que están trabajando en el sistema. Más adelante, “los objetos del dominio” nos ayudarán a identificar algunas de las clases (unidad V) a medida que analizamos y diseñamos el sistema.

Un modelo del dominio captura los tipos más importantes de objetos en el contexto del sistema.



Los objetos del dominio representan las "cosas" que existen o los eventos que suceden en el entorno en el que trabaja el sistema

Muchos de los “objetos del dominio” o clases (unidad V) pueden obtenerse de una especificación de requisitos o mediante la entrevista con los expertos del dominio.

Las clases del dominio aparecen en tres formas típicas:

- Objetos del negocio que representan cosas que se manipulan en el negocio, como pedidos, cuentas y contratos.
- Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento, como la aviación enemiga, misiles y trayectorias.
- Sucesos que ocurrirán o han ocurrido, como la llegada de un avión, su salida y la hora de la comida.

El modelo del dominio se describe mediante diagramas de UML (especialmente mediante diagramas de clases). Estos diagramas muestran a los clientes, usuarios, revisores y a otros desarrolladores las clases del dominio y cómo se relacionan unas con otras mediante asociaciones.

» *Desarrollo de un modelo del dominio*

El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema.

El glosario y el modelo del dominio ayudan a los usuarios, clientes, desarrolladores, y otros interesados a utilizar un vocabulario común. La terminología común es necesaria para compartir el conocimiento con los otros.



Es muy importante recordar que el objetivo del modelado del dominio es contribuir a la comprensión del contexto del sistema, y por lo tanto también contribuir a la comprensión de los requisitos del sistema que se desprenden de este contexto.

En otras palabras, el modelado del dominio debería contribuir a una comprensión del *problema* que se supone que el sistema resuelve en relación a su contexto. El modo interno por el cual el sistema resuelve este problema se tratará en los flujos de trabajo de análisis, diseño, e implementación.

» *Construimos un Diagrama Clases de Dominio*

Basándonos en un caso hipotético de un negocio que comercializa productos, podemos identificar algunas de las clases del dominio: como por ejemplo Pedido, Factura, Artículo y Cuenta.

Si suponemos que "El sistema utilizará Internet para enviar pedidos, facturas y pagos, entre compradores y vendedores"....

El sistema ayudará

- a) Al comprador a confeccionar sus pedidos,
- b) Al vendedor a evaluar los pedidos y a enviar las facturas
- c) Al comprador a validar las facturas y hacer efectivos los pagos de sus cuenta a la del vendedor.

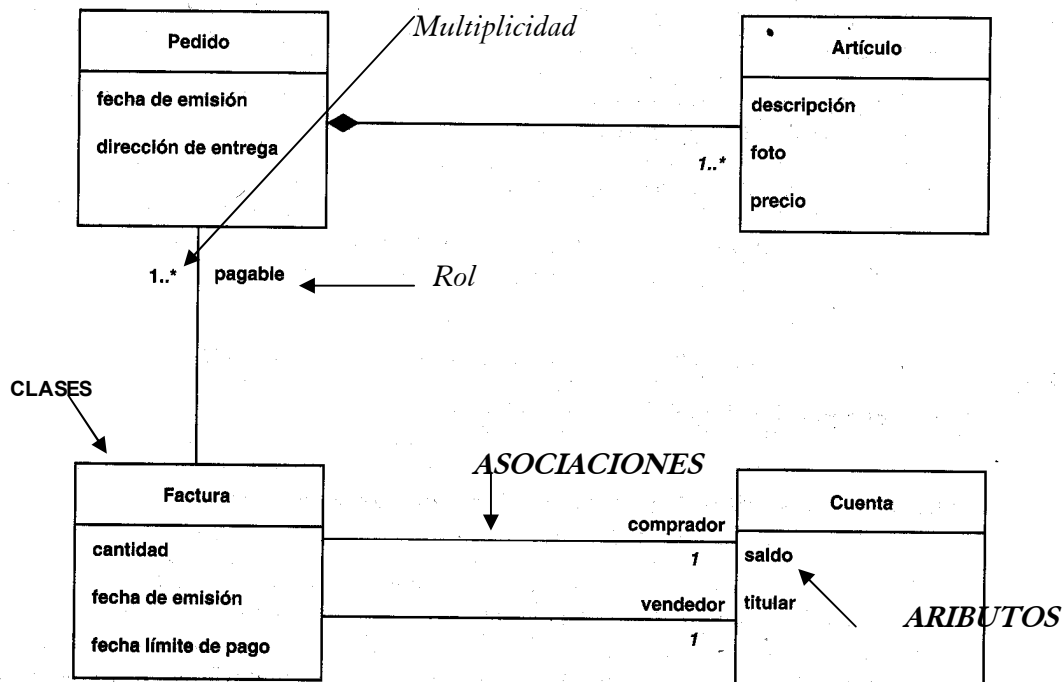
Si analizamos más en detalle estos conceptos, podemos señalar,

Primero: Un pedido es la solicitud de un comprador a un vendedor, de un artículo. Cada artículo ocupa una línea en el Pedido. Un pedido posee datos (atributos) como la fecha de emisión, y la dirección de entrega, etc. Hasta ahora hemos descubierto dos clases "PEDIDOS" y "ARTICULOS".

Segundo: Una factura es una solicitud de pago del vendedor al comprador, en respuesta a un pedido de bienes o servicios. Una factura posee datos (atributos) como la cantidad, fecha de emisión, y fecha límite de pago. Una factura puede ser la solicitud de pago de varios pedidos. Encontramos otra clase: "FACTURA"

Tercero: Las facturas se pagan mediante la transferencia de dinero de la cuenta del comprador a la del vendedor. Una Cuenta posee datos (atributos) como el saldo y el titular. El atributo titular identifica a la persona a la cual pertenece la cuenta.

El gráfico nos muestra, como se relacionan cada una de estas clases, formando el diagrama de Clases del Dominio.



Un diagrama de clases en un modelo del dominio, que captura los conceptos más importantes del contexto del sistema.

Clases (rectángulos), atributos (texto en la parte inferior de los rectángulos), asociaciones (las líneas entre los rectángulos de las clases). El texto al final de una línea de asociación explica el rol de la clase en relación con la otra, es decir, el rol de la asociación. Multiplicidad (los números y asteriscos al final de una línea de asociación) indican cuántos objetos de la clase de ese extremo pueden enlazarse a un objeto en el otro extremo.

Por ejemplo, la asociación que conecta las clases Factura y Pedido en la Figura tiene una multiplicidad 1..*, dibujada en el extremo de la clase Pedido. Esto significa que cada Factura puede ser una solicitud de pago de uno o más objetos Pedido, como se indica **rol** de la asociación pagable.

► B - Comprensión del contexto del sistema mediante un Modelo del Negocio

A medida que los analistas modelan el negocio aprenden mucho sobre el contexto del sistema software, y lo describen en un modelo del negocio.



El objetivo del modelado del negocio es describir los procesos --existentes u observados- con el objetivo de comprenderlos. **El modelo del negocio** especifica qué procesos de negocio soportará el sistema. Aparte de identificar los objetos del dominio implicados en el negocio, este modelado también establece las competencias requeridas en cada proceso.



El modelado del negocio es una técnica para comprender los procesos de negocio de la organización.

Pero, ¿qué pasa si tratamos con un sistema que no tiene nada que ver, con lo que la mayoría de nosotros considera un negocio? Por ejemplo, ¿qué deberíamos hacer en el desarrollo de un marcapasos, de un sistema de frenos, de un software educativo, o de un sistema de telecomunicaciones? En estos casos, también podemos modelar el sistema que rodea el sistema software que vamos a desarrollar. Este sistema (parte del cuerpo humano, parte de un coche, la cámara, el conmutador) es el "sistema de negocio" del sistema software empotrado. El objetivo es identificar los casos de uso del software y las entidades de negocio relevantes que el software debe soportar, de forma que podríamos modelar sólo lo necesario para comprender el contexto.

El modelo de casos de uso del negocio representa un sistema (en este caso, el negocio) desde la perspectiva de su uso, y esquematiza cómo proporciona valor a sus usuarios (en este caso, sus clientes y socios).

Modelo de Caso de uso del Negocio

Un modelo de Caso de uso de Negocio, describe procesos de un negocio y sus interacciones con el exterior.

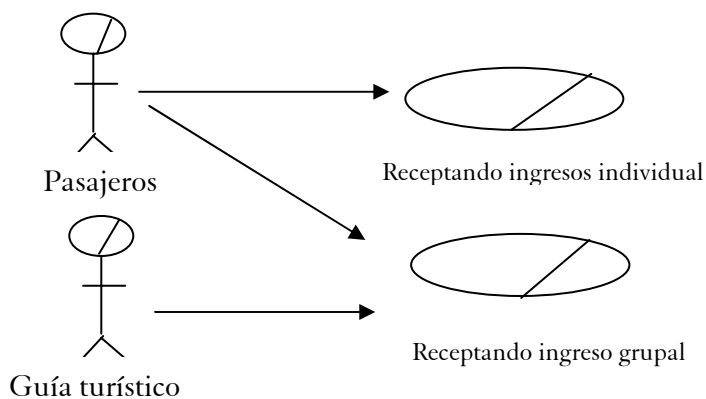
El diseño del modelo persigue ciertos objetivos principales

- ▶ Resumir el propósito de la organización
- ▶ Destacar los límites de las cosas que no se incluyeron en el modelo y las razones por las que no se incluyeron
- ▶ Definir los principales caso de uso del negocio.
- ▶ Cada actividad del negocio debería incluirse en al menos un caso de uso.
- ▶ La revisión del modelo de caso de uso debería dar una vista comprensiva de la organización.

Un ejemplo, de una porción del diseño del diagrama de caso de uso del negocio de una empresa turística, sería la grafica que se muestra

Los actores, se simbolizan con una figura de un hombre, con una línea en la cabeza, mientras que los procesos con una elipse también cruzada por una línea en uno de sus extremos. Esta simbología es solo para diferenciarlos del modelo

de sistema de información (tema que se vera en la próxima unidad)

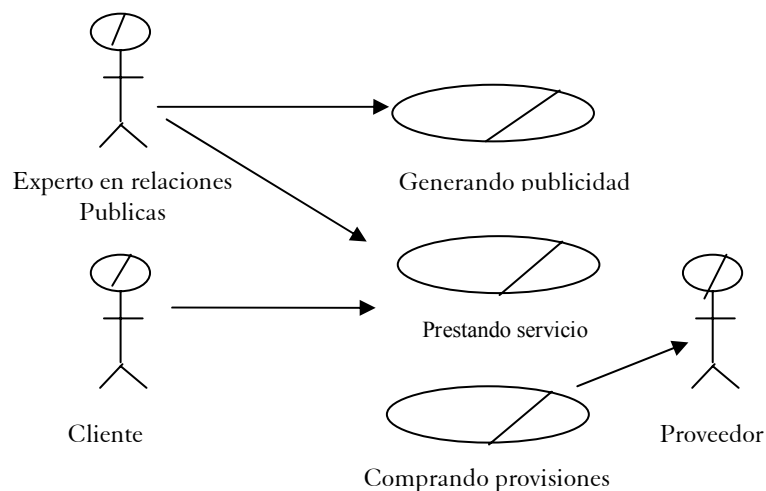


Diferentes categorías de Caso de uso de Negocio

Cuando se analizan las actividades del negocio es posible identificar al menos tres categorías de caso de uso de negocio:

- » Primero, las actividades comercialmente importantes, llamadas procesos de negocio.
- » Segundo, hay actividades que no son comercialmente importantes, pero deben realizarse de cualquier forma para hacer el trabajo del negocio, ej. Limpieza, seguridad, que serían caso de uso de soporte.
- » Tercero, los casos de uso de negocio de administración muestran el típico trabajo que afecta como otros procesos de negocio son administrados y las relaciones del negocio con sus dueños. Comúnmente, los casos de uso de administración describen en general las relaciones entre Gerentes

Si tomamos otro ejemplo, ahora un restaurante, los **caso de uso centrales** del negocio, son generando publicidad y prestando el servicio, mientras que, el **caso de uso de soporte** es comprando provisiones.



¿Están los Caso de uso del negocio relacionados siempre con actores?

Cada Caso de uso del negocio deberá tener una relación de comunicación de o hacia un actor de negocio. Los casos de uso del negocio tienen actores de negocio que comúnmente los inician y esperan diferentes servicios de ellos. De otra forma no serían parte del negocio. Otros casos de uso del negocio producirán resultados para un actor del negocio, aunque no sean explícitamente iniciados por el actor del negocio. Los casos de uso de negocio de administración y soporte, no necesariamente necesitan estar conectados a un actor de negocio. Los casos de uso abstractos del negocio, no necesitan un actor de negocio, nunca son instanciados por estos, son procesos necesarios de definir en el modelado de negocio, porque luego serán generadores de procesos en el modelo de sistema de información.

Vistas del negocio

En un proyecto de ingeniería conducido por Caso de uso, se desarrollan dos vistas del negocio

Una vista externa:

La cual define, lo que es esencial que el negocio realice para entregar el resultado deseado al actor.

Una vista interna:

La cual define como debería ser organizado y ejecutado el trabajo para alcanzar los resultados deseados. Una realización comprende a los trabajadores del negocio y a las entidades del negocio, que son invocadas en la ejecución de un caso de uso para hacer el trabajo.

Ambas vistas están desarrolladas para las personas que están relacionadas con el negocio, tanto la gente que esta fuera del negocio, como para las personas que trabajan dentro del negocio.

Nombre de un Caso de uso

El nombre, del caso de uso de negocio debería expresar que pasa cuándo una instancia del caso de uso es ejecutada. La forma del nombre deberá ser por lo tanto, activa, descriptiva comúnmente por la forma gerundia del verbo o un verbo y un sustantivo juntos.

Por ejemplo Generando permisos
Comprando provisiones
Inscribiendo pasajeros

Los nombres también pueden describir las actividades del negocio desde un punto de vista interno o externo.

Por ejemplo:
Enviando una orden o recibiendo una orden.

Estructura de un Caso de uso

La descripción de un caso de uso de negocio contiene en general las siguientes partes

- ▶ Nombre del caso de uso: debe ser breve y claros y fácil de comprender, aun para la gente que no forma parte del equipo de reingeniería
- ▶ Objetivo
- ▶ Flujo de eventos básico, debería cubrir lo que normalmente ocurre cuando el caso de uso es ejecutado.
- ▶ Flujo de eventos alternativos, cubren comportamiento de carácter excepcional u opcional en relación al comportamiento normal. Se pide pensar en los flujos alternativos como desviaciones del flujo básico de eventos.

Actor

Un actor de negocio, representa un rol jugado en relación al negocio, por algo o alguien en el entorno del negocio.

Para comprender completamente el propósito de un negocio se debe saber con quien interactúa el negocio, esto es, quien pone demandas sobre el, o esta interesado en sus salidas

Los siguientes tipos de usuarios del negocio son ejemplos de potenciales actores del negocio:

- » Clientes
- » Proveedores
- » Socios
- » Potenciales clientes
- » Autoridades locales
- » Colegas de partes del negocio no modeladas

Por lo tanto un actor normalmente corresponde a un ser humano, sin embargo, hay circunstancias donde por ejemplo un sistema de información juega un rol de actor.

Características de un actor de Negocio

- Todas las interacciones del entorno del negocio, son modeladas como actores.
- Cada actor modela algo fuera del negocio
- Cada actor esta involucrado al menos con un **caso de uso**, si no es así debería revisarse.
- Un actor específico no interactúa con el negocio de varias formas completamente diferentes. Si eso ocurre es probable que se le hayan asignado varios roles a un actor.

Diagrama de Caso de uso en el modelo de UC de Negocio

Un diagrama de caso de uso muestra actores, caso de uso de negocios, y sus relaciones.

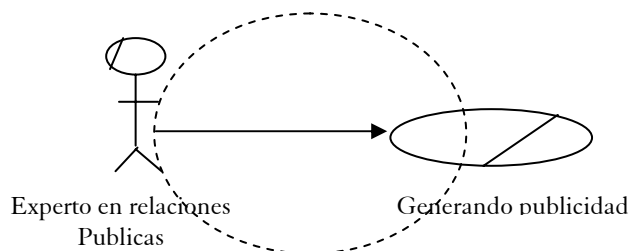
Asociaciones de comunicación en el modelo de caso de Uso de negocio

1- Una asociación de comunicación entre un caso de uso y un actor, indica que una instancia de un caso de uso y una instancia del actor van a interactuar.

Los actores interactúan enviando y recibiendo mensajes.

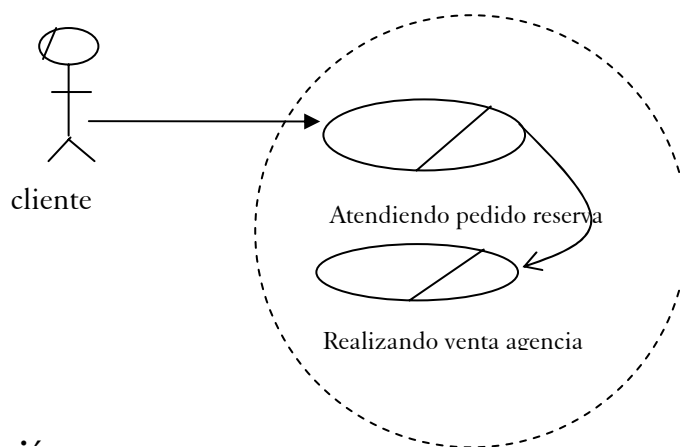
Cuando un actor y un caso de uso interactúan, pueden hacerlo usando diferentes criterios, por ejemplo teléfono, fax, correo, personalmente etc., pero hay una sola asociación de interacción entre los dos.

Gráficamente



2- Una asociación de comunicación entre un caso de uso y otro caso de uso, también indica que van a interactuar, por lo que es bueno que veamos las siguientes relaciones:

Gráficamente



Relación de Extensión



Una relación de extensión, es una relación desde un caso de uso de extensión a un caso de uso base, especificando como el comportamiento definido en el caso de uso de extensión puede ser insertado en el comportamiento definido por el caso de uso base.

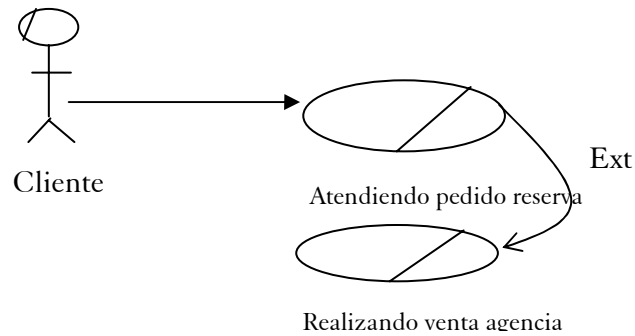
Es implícitamente insertado en el sentido que la extensión no es mostrada en el caso de uso base.

Una instancia de caso de uso de negocio que es opcionalmente extendida por otro caso de uso, primero sigue la descripción del caso de uso base y luego, si se cumple alguna condición sigue la descripción del caso de uso de extensión. Cuando el caso de uso de extensión alcanza el final, continua con la descripción de la base.

El caso de uso de negocio que está siendo extendido tiene que ser significativo y completo en si mismo, aún si el caso de uso adicional no se ejecuta.

Por ejemplo si observas la figura, se ha tomado una porción del diagrama de caso uso del negocio de una agencia de minibús, podemos observar que el caso de uso *Atendiendo pedido de reserva* es una extensión del caso de uso base *Realizando una venta en agencia*, cumpliendo con cada una de las recomendaciones que leíste.

Realizando una venta en agencia, es un caso de uso completo en si mismo, que no necesita que se haya hecho una reserva para poder vender un pasaje, es decir puedo o no acceder al caso de uso extendido, no es imprescindible hacerlo.



Relación de Inclusión

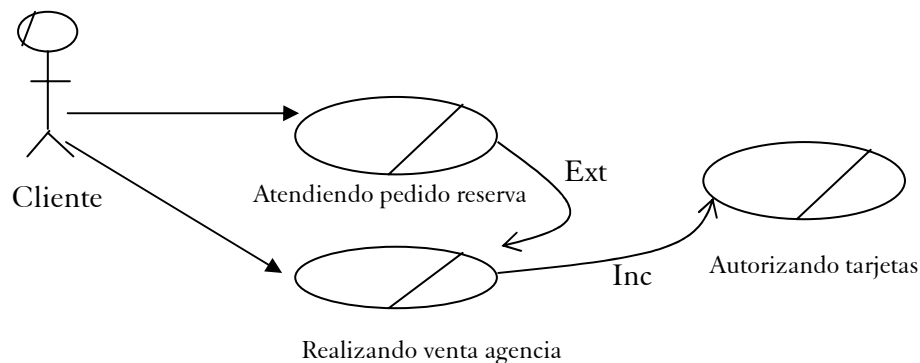


Una relación de inclusión es una relación desde un caso de uso de inclusión, especificando como el comportamiento definido en el caso de uso de inclusión es explícitamente insertado en el comportamiento definido en el caso de uso base.

Un caso de uso de negocio que sigue la descripción de un caso de uso base, también seguirá la descripción del caso de uso de inclusión. El flujo de trabajo completo descrito en el caso de uso de negocio incluido es incorporado. El caso de uso de inclusión de este tipo es siempre abstracto y no necesita tener una relación con un actor de negocio.

Un caso de uso de inclusión es siempre abstracto, es decir que no necesita tener una relación con un actor de negocio.

Gráficamente se presentaría.



Relación de Generalización

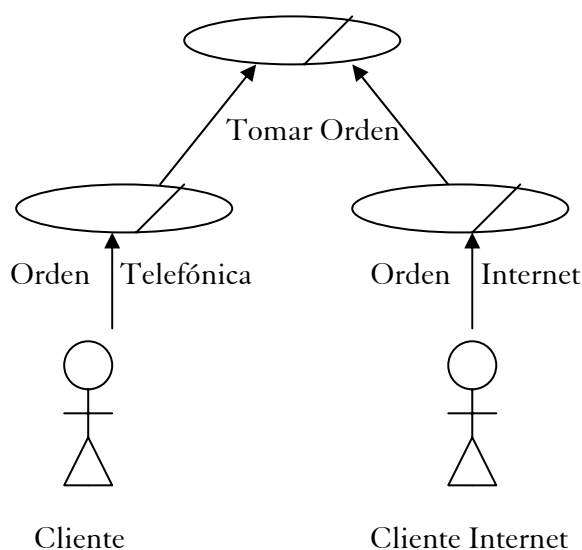


Una generalización de caso de uso, es una relación de un caso de uso hijo a un caso de uso padre, especificando como el hijo puede especificar todo el comportamiento y características descriptas por el padre.

La generalización es usada cuando se encuentran dos o mas caso de uso que tienen cosas en común en su comportamiento, estructura y propósito. Cuando esto ocurre, se puede describir las partes en un caso de uso nuevo, a menudo abstracto, que es entonces

completado con las especificaciones del proceso mas especializadas por los caso de usos hijo.

Gráficamente se presentaría



Una instancia de caso de uso hijo ejecutándose, seguirá el flujo de eventos descrito por el caso de uso padre insertando comportamiento adicional y modificando comportamiento como se define en el flujo de eventos del caso de uso hijo.

Autoevaluación

Analice las siguientes situaciones y desarrolle cada uno de los modelos.

I- Caso práctico Nº1

El Hotel Resplandor, se encuentra ubicado en la ciudad de Santa Fe. El mismo aloja pasajeros de excursiones como clientes particulares que se hospedan por algunos días. Cada vez que un pasajero llega al hotel debe registrarse en la recepción, en donde completa una ficha de ingreso al hotel. Esa ficha contiene los datos de los pasajeros (nombre, apellido, estado civil, documento, domicilio, etc.). Una vez que se ha completado el tramite el conserje verifica las habitaciones que tiene disponibles y le asigna una, entregándole la llave de la misma.

Diariamente el pasajero solicita alguno de los servicios que se brindan en el hotel (lavandería, cafetería, etc.), para eso, este debe dirigirse al conserje, el cual registrará la solicitud del servicio que desea, y se lo asignará a la habitación correspondiente, emitiéndole al pasajero un recibo como constancia del mismo. Al finalizar el día, se ordenan y registran en una ficha realizada por habitación todos los servicios brindados junto con el correspondiente comprobante y se lo emite a contaduría.

Cuando el pasajero cancela su estadía, se emite un listado y la factura correspondiente, detallando los importes del alojamiento como así también el de los servicios prestados. Mensualmente se realiza un listado que se emite a Gerencia informando de la cantidad de habitaciones ocupadas, los importes cobrados y las anormalidades surgidas en ese período.

1 -Defina el Modelo de Negocios, utilizando:

El Diagrama de Casos de Uso y La descripción de los C. de uso del negocio

2- Caso práctico N°2

En la Clínica XXX, los pacientes son atendidos de acuerdo al orden de llegada. Cuando el paciente ingresa, se le toman los datos con el objeto de buscar su historia clínica, en la cual el médico anotará: diagnóstico, terapia sugerida y análisis clínicos. En la clínica no se reciben mutuales, y el precio de cada consulta varía según la especialidad.

Cuando se cobra se emite un recibo para el paciente, y se registra el importe en un informe de ingreso a caja, que diariamente es enviado a Contaduría. Además del recibo, se le entrega al paciente una solicitud de consulta, que éste le entregará al médico al ser atendido.

Todos los meses deben enviarse a Personal un reporte que contenga la cantidad de consultas atendidas para cada médico, para que le realicen la respectiva liquidación de honorarios.

1 - Según su criterio, seleccione las herramientas necesarias y suficientes, para capturar en forma exitosa los requerimientos esenciales que se plantean en este caso práctico.

tutor1 ▶

Estos casos serán desarrollados en grupo o individualmente, corroborando las respuestas, con el docente en los encuentros presenciales.

► Unidad III

Captura de Requisitos: “Capturar requisitos funcionales”. “Capturar requisitos no funcionales”.

Objetivos

- ▶ Introducirse en las distintas herramientas de modelado.
- ▶ Desarrollar habilidades en el manejo de las herramientas, para modelar la verdadera esencia del sistema.
- ▶ Descubrir el comportamiento adecuado de un sistema, utilizando para ello el modelo de sistema de Información.
- ▶ Construir el diagrama de Casos de Uso del Sistema de Información
- ▶ Construir las plantillas de Caso de uso del Sistema de Información
- ▶ Diseñar los prototipos de interfaz
- ▶ Conocer cómo se relacionan las herramientas de distintos modelos vistos.
- ▶ Transferir los conocimientos adquiridos, en trabajos prácticos de aplicación.

Contenidos de la Unidad

¿Qué es la captura de requisitos?. ¿Por qué la captura de requisitos es complicada?.
¿Cómo se consigue?. ¿Cuál es la finalidad?

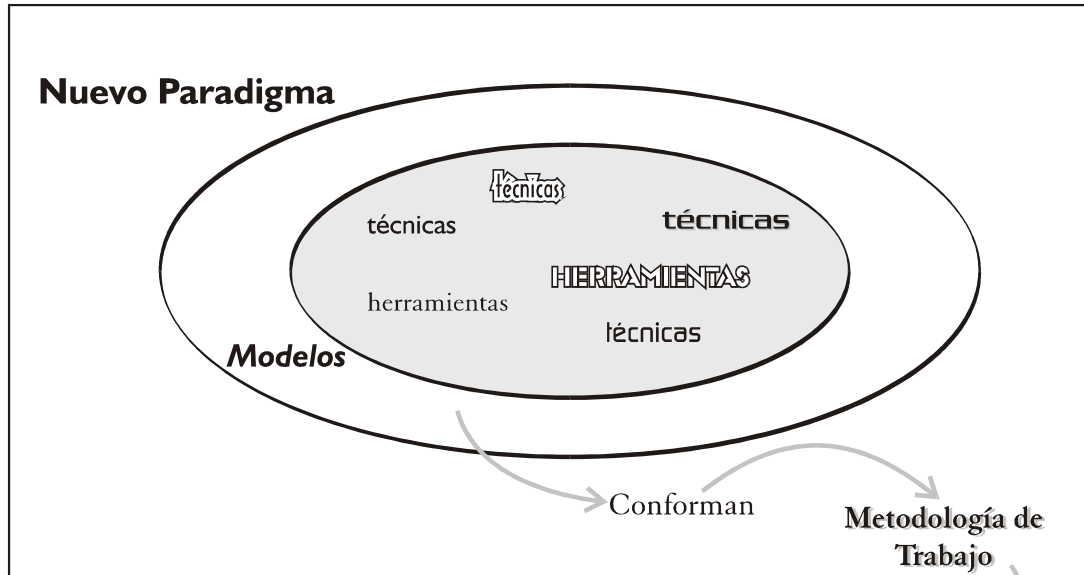
3- Capturar requisitos funcionales

- a - Encontrar actores y casos de uso - Describir brevemente cada caso
- b - Priorizar casos de uso
- c - Detallar un caso de uso.
 - Descripción de funcionalidad compartida
 - Generalización entre casos de uso. Una relación de inclusión
- d - Prototipar la interfaz del usuario
- e - Estructurar el modelo de casos de uso

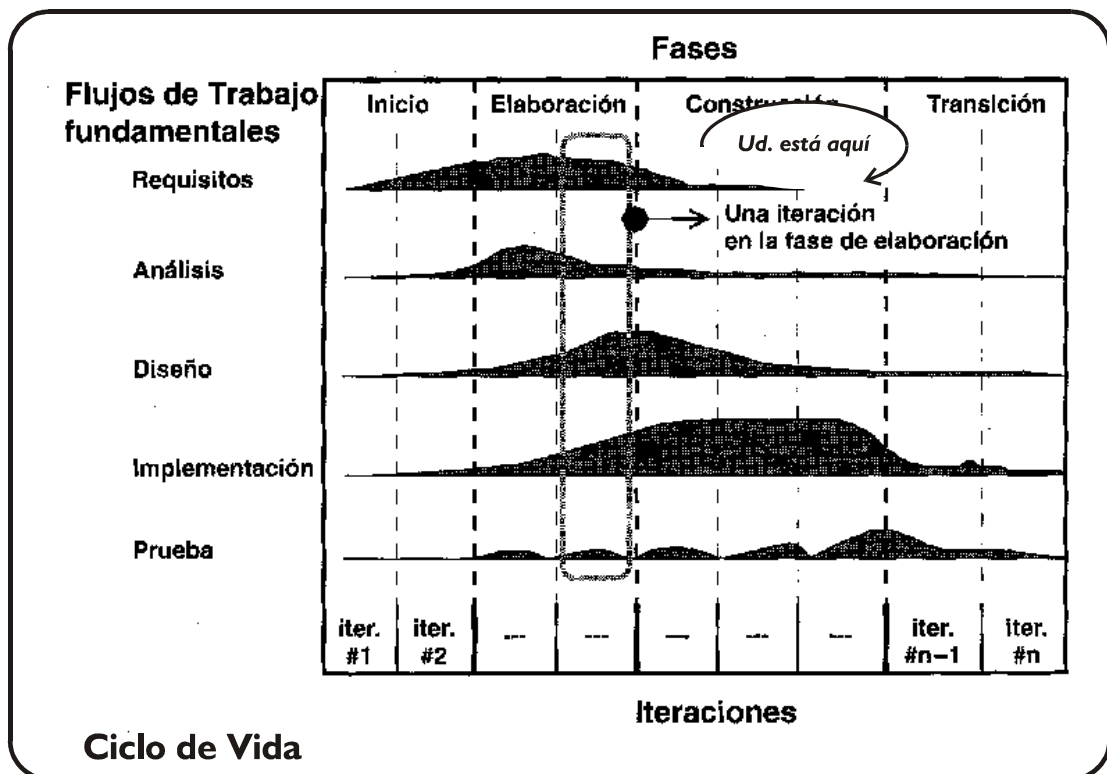
4- Capturar requisitos no funcionales

Síntesis: Modelos y diagramas de la Captura de Requisitos en el ciclo de vida.

Mapa Conceptual



Proceso Unificado de Desarrollo de Software



Introducción

Continuamos con la descripción del flujo de trabajo Captura de Requisitos, en esta unidad centraremos nuestra atención en definir los Requisitos Funcionales y No funcionales. Los modelos que lo componen, y los diagramas necesarios de construir.

Recordemos que dentro del material, se describen los flujos de trabajo, sus modelos y herramientas como herramientas aisladas, solamente con el fin de describir mejor su objetivo y sus reglas de construcción, pero en el contexto general de PDUS no es una tarea secuencial, sino iterativa incremental.

En la unidad anterior definimos los siguientes pasos como importantes en el flujo de trabajo: Captura de Requisitos

1. Enumerar los requisitos candidatos.
2. Comprender el contexto del sistema.
3. Capturar requisitos funcionales.
4. Capturar requisitos no funcionales.

En esta unidad retomaremos la temática abordada y continuaremos en primera instancia con la conceptualización de términos significativos, para luego proseguir con la descripción de la *Captura de Requisitos Funcionales (sus modelos y herramientas)* y *Captura de Requisitos No Funcionales*, puntos tres y cuatro de la clasificación del párrafo anterior.

¿Qué es la captura de requisitos?



Llamamos captura de requisitos a un acto de descubrimiento. Es el proceso de averiguar normalmente en circunstancias difíciles, lo que se debe construir.

¿Por qué la captura de requisitos es complicada?

Ya sabemos que los desarrolladores de software profesionales normalmente crean código para otros y no para sí mismos. De un modo u otro, normalmente cualquier sistema tiene muchos usuarios (o tipos de usuarios), y mientras que cada uno de ellos puede saber lo que él o ella hacen, ninguno tiene una visión global.

La mayoría de los usuarios no sabe qué partes de su trabajo pueden transformarse en software. Francamente, con frecuencia los usuarios no saben cuáles son los requisitos ni tampoco cómo especificarlos de una forma precisa, los usuarios no comprenden del todo lo que el sistema software debe hacer hasta que el sistema esta casi terminado.

Durante años nos hemos engañado a nosotros mismos creyendo que los usuarios saben cuáles son los requerimientos, que los tienen claros y que lo único que tenemos que hacer es entrevistarnos con ellos. Es cierto que los sistemas que construimos deberían dar

soporte a los usuarios, y que podemos aprender de ellos sobre sus interacciones. Sin embargo, es aún más importante que los sistemas den soporte a la *misión* para la cual se construyen.

Aún peor como reflejo del mundo real en constante cambio, durante la vida del proyecto, puede cambiar el negocio en sí, puede cambiar la tecnología disponible para construir el sistema, pueden cambiar los recursos (personas, dinero) disponibles para construir el sistema, etc.

Aún con estas ideas, la **captura de requisitos** sigue siendo difícil, y la industria lleva **buscando** un proceso bueno, sistemático para llevarla a cabo, desde hace mucho tiempo.



Por todo ello el propósito fundamental del flujo de trabajo de los requisitos es guiar el desarrollo hacia el sistema correcto.

¿Cómo se consigue?

Esto se consigue mediante una descripción de los requisitos del sistema (*es decir, las condiciones o capacidades que el sistema debe cumplir*) suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

¿Cuál es la finalidad?

Un reto fundamental para conseguirlo es que el cliente, que asumimos que la mayor parte de las veces será un especialista no informático, debe ser capaz de leer y comprender el resultado de la captura de requisitos. Para alcanzar este objetivo debemos utilizar *el lenguaje del cliente* para describir esos resultados.

3- Capturar requisitos funcionales



La técnica inmediata para identificar los requisitos del sistema se **basa en los casos de uso**. Estos casos de uso capturan tanto los requisitos funcionales como los no funcionales que son específicos de cada caso de uso.

► ¿Cómo los casos de uso nos ayudan a capturar los requisitos adecuados.?

Cada usuario quiere que el sistema haga algo para él o ella, es decir, que lleve a cabo ciertos casos de uso. Para el usuario, un caso de uso es un modo de utilizar el sistema.

En consecuencia, si los analistas pueden describir todos los casos de uso que necesita el usuario, entonces saben lo que debe hacer el sistema.



Cada caso de uso representa una forma de usar el sistema (de dar soporte a un usuario durante un proceso de negocio). Cada usuario necesita varios casos de uso distintos, cada uno de los cuales representa los modos diferentes en los cuales él o ella utilizan el sistema.

La captura de los casos de uso que realmente se quieren para el sistema, como aquellos que soportarán el negocio y que el usuario piensa que le permiten trabajar cómodamente, requieren que conozcamos en profundidad las necesidades del usuario y del cliente.

A continuación, vamos a describir las actividades que corresponden a la delimitación de los requisitos funcionales y que típicamente aparecen en la **iteración de la fase inicio y elaboración**.



Actividad

- (a) Encontrar actores y casos de uso – Describir brevemente cada caso.
- (b) Priorizar casos de uso
- (c) Detallar un caso de uso
- (d) Prototipar la interfaz del usuario
- (e) Estructurar el modelo de casos de uso

a- Encontrar actores y casos de uso - Describir brevemente cada caso

Primero, el analista de sistemas (una persona respaldada por un equipo de analistas) ejecuta la actividad de “Encontrar Actores y Casos de Uso” para preparar una primera versión del modelo de casos de uso, con los actores y casos de uso identificados. El analista de sistemas debe asegurar que el desarrollo del modelo de casos de uso captura todos los requisitos que son entradas del flujo de trabajo, es decir: la lista de características, el modelo de dominio y/o el modelo de negocio.

Es la actividad más decisiva para obtener adecuadamente los requisitos, y es responsabilidad del analista de sistemas

Búsqueda de casos de uso y de actores del Sistema de Información (SI) a partir de un Modelo del Negocio

Mediante la utilización de un modelo del negocio como entrada, un analista emplea una técnica sistemática para crear un modelo de casos de uso de SI.

1er Paso

El analista identifica **un actor del SI**, por cada trabajador y por cada actor del negocio, que se **convertirá en usuario del sistema** de información.

Tomemos el ejemplo del modelo de negocio de la empresa de minibuses, tenemos:

Actores del negocio	Trabajadores del negocio	Actores del S. de información
Taller mecánico	Mecánico	Mecánico o secretaria de administración
Secretaria de transporte	Empleada administrativa	Empleada administrativa
Cliente	Empleado de ventas	

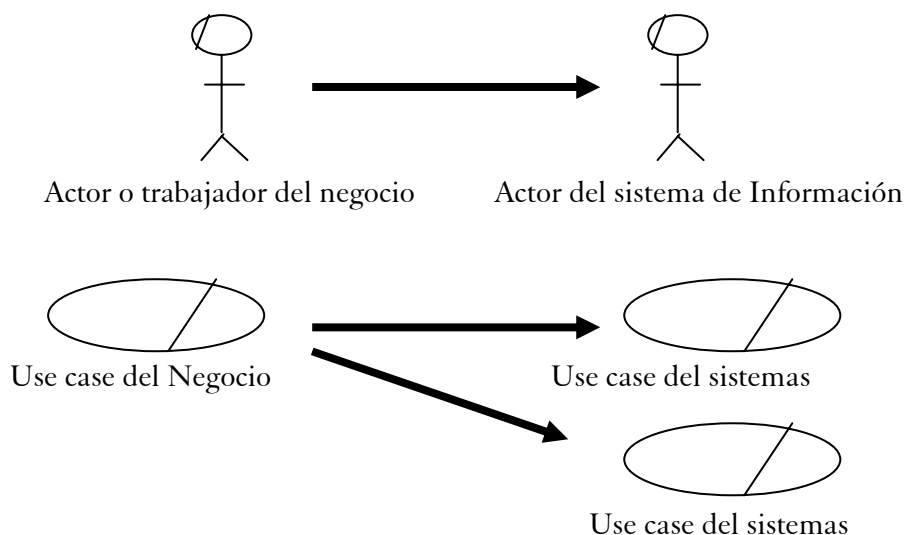
2do Paso

Cada trabajador y cada actor del negocio que vaya a ser usuario del sistema de información requerirá un soporte por parte del mismo. Para cada **trabajador**, **identificamos todas las realizaciones de casos de uso del negocio** diferentes en las que participa.

Actores del negocio	Trabajadores del negocio	Actores del S. de información
Cliente	Cliente	Cliente (autogestión Internet)
Cliente	Cliente	Empleado de ventas (agencia)

3er Paso

Una vez que hemos encontrado todos los roles de un trabajador o de un actor del negocio, uno por cada caso de uso del negocio en el que participa, podremos **encontrar los casos de uso de los actores del sistema de información**.



Por tanto la manera más directa de identificar un conjunto tentativo de casos de uso es crear un caso de uso para el actor correspondiente a cada rol de cada trabajador y de cada actor del negocio.

Sistema de Información	Como encontrar candidatos usando información en los modelos de negocio	Negocio
Actor	Actores candidatos se encuentran entre trabajadores de negocio	Trabajador de Negocio
Actor	Otros candidatos de actores se encuentran entre los diferentes actores de negocio (clientes, vendedores) que usarán el sistema directamente.	Actor de Negocio
Use Case	Los candidatos a use cases se encuentran entre las operaciones de los trabajadores de negocio. Busque operaciones (y áreas de responsabilidad) que implica interacciones con el sistema de información. Idealmente un use case del sistema de información debería soportar todas las operaciones del trabajador dentro de la realización del modelo de use case de negocio.	Operaciones de los trabajadores de negocio
Clase Entidad	Las clases de entidad candidatas se encuentran entre entidades de negocio. Busque entidades de negocio que deberían ser mantenidas o representadas en el sistema de información.	Entidad de Negocio
Clase Entidad	La clase entidad candidata encontrada entre atributos en el modelo de objetos de negocio. Busque atributos que deberían ser mantenidos o representados en el sistema de información.	Atributos
Relaciones entre clases de entidad	Las relaciones entre entidades de negocio a menudo indican una relación correspondiente entre las clases del modelo del sistema de información.	Relaciones entre entidades de negocio

b- Priorizar casos de uso

Entonces, el arquitecto/s (Analista de sistemas) identificará/n los casos de uso relevantes arquitectónicamente hablando, para proporcionar entradas a la priorización de los casos de uso (y posiblemente otros requisitos) que van a ser desarrollados en la iteración actual.

El propósito de esta actividad es proporcionar entradas a la priorización de los casos de uso para determinar cuáles son necesarios para el desarrollo (es decir, análisis, diseño, implementación, etc.) en las primeras iteraciones, y cuáles pueden dejarse para más tarde.

La vista de la arquitectura del modelo de casos de uso debe mostrar los casos de uso significativos desde el punto de vista de la arquitectura.

c- Detallar un caso de uso

El objetivo principal de detallar cada caso de uso, es describir su flujo de sucesos en detalle, incluyendo cómo comienza, termina e interactúan con los actores

Con el modelo de casos de uso y los diagramas de casos de uso asociados como punto de comienzo, el especificador de un caso de uso individual puede ya describir cada caso de uso en detalle. El especificador de casos de uso detalla paso a paso la descripción de cada caso de uso, en una especificación precisa de la secuencia de acciones.

En esta sección veremos:

- ▶ Cómo estructurar la descripción para especificar todas las vías alternativas del caso de uso.
- ▶ Cómo formalizar la descripción del caso de uso cuando sea necesario.

Cada especificador de casos de uso debe trabajar estrechamente con los usuarios reales de los casos de uso. El especificador de casos de uso necesita entrevistarse con los usuarios, quizás anotar su comprensión de los casos de uso y discutir propuestas con ellos, y solicitarles que revisen la descripción de los casos de uso.

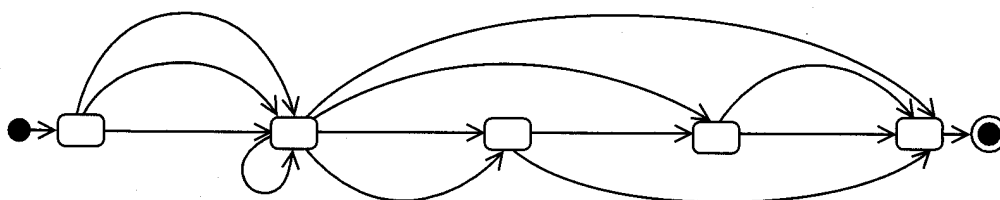
El resultado de esta actividad es la descripción detallada de un caso de uso en particular en forma de texto y diagramas.

Estructuración de la descripción o Especificación de casos de uso

Ya hemos mencionado que un caso de uso define los estados que las instancias de los casos de uso pueden tener y la posible transición entre estos estados.



Cada transición es una secuencia de acciones que se ejecutan en una instancia del caso de uso cuando ésta se dispara por efecto de un suceso, como podría ser un mensaje.



Un caso de uso puede imaginarse como si tuviera un estado de comienzo (el círculo de más a la izquierda), estados intermedios (los subsiguientes rectángulos redondeados), estados finales (el círculo de más a la derecha) y transiciones de un estado a otro. Las flechas rectas ilustran el camino básico, y las curvas, otros caminos.

El gráfico de transición de estados ilustrado en la Figura puede llegar a ser bastante intrincado. Por ello, debemos describir la posible transición de estados de manera simple y precisa. Una técnica probada es elegir un camino básico completo (las flechas rectas en la Figura desde el estado de inicio al estado final, y describir ese camino en una sección de la descripción. Entonces podemos describir en secciones separadas el resto de los caminos (flechas curvas) como caminos alternativos o desviaciones del camino básico.

Algunas veces, no obstante, las alternativas o desviaciones son lo suficientemente pequeñas como para explicarlas "en línea" como parte de la descripción del camino básico. El sentido común determina si la descripción debe ser "en línea" o se debe crear una sección separada para ella. Hay que recordar que el objetivo es hacer una descripción precisa pero fácil de leer.

Con cualquier técnica que elijamos, tendremos que describir todas las alternativas, sino, no habremos especificado el caso de uso.

Las alternativas, desviaciones, o excepciones del camino básico pueden ocurrir por muchas razones:

- » El actor puede elegir entre diferentes caminos en el caso de uso. Por ejemplo, durante el caso de uso Pagar Factura, el actor puede decidir pagar una factura o rechazarla-
- » Si está implicado más de un actor en el caso de uso, las acciones de uno de ellos pueden influenciar el camino de las acciones del resto de actores.
- » El sistema puede detectar entradas erróneas de los actores.

El camino básico elegido debe ser el camino "normal", esto es, el que el usuario percibe como el que más habitualmente va a seguir y aquél que proporciona el valor más obvio al actor. Generalmente, cada camino básico debe abarcar un par de excepciones y un par de peculiaridades que el sistema raramente necesita manejar.

► Descripción de casos de uso del SI

(General - Detallada)

Ejemplo

Descripción general

La descripción general para el modelo de casos de uso del Sistema de Facturación y Pagos podría parecerse a la siguiente narración.

El comprador utiliza el caso de uso Solicitar Bienes y Servicios para buscar los productos y precios, para realizar un pedido y después enviarlo.

Tarde o temprano, los bienes o servicios le llegarán al comprador junto con una factura.

El comprador activa el caso de uso Pagar Factura para dar el visto bueno a la factura recibida y planificar el pago requerido. En la fecha planificada, el caso de uso Pagar Factura transferirá automáticamente el dinero desde la cuenta del comprador a la cuenta del vendedor (comentario 1).

Es más, el caso de uso Pagar Recargos Saldo Deudor extiende al caso de uso Pagar Factura si se produce un descubierto en el saldo (comentario 2).

Vamos a pasar ahora a cómo utiliza el sistema el vendedor. El vendedor puede estudiar, proponer cambios y confirmar los pedidos recibidos utilizando el caso de uso Confirmar Pedidos. Un pedido confirmado estará seguido de la entrega de los bienes o servicios (no descritos en nuestro modelo de casos de uso de ejemplo; de hecho se hace fuera del Sistema de Facturación y Pagos).

Más tarde, cuando hayan sido entregados los bienes o servicios, el vendedor facturará al comprador a través del caso de uso Enviar Factura al Comprador. Al llevar a cabo la facturación,

el vendedor puede que tenga que aplicar un descuento y puede que también elija combinar varias facturas en una.

Si el comprador no ha pagado en la fecha de vencimiento, se informará al vendedor y éste puede utilizar el caso de uso Enviar Aviso. El sistema podría enviar avisos automáticamente, pero tenemos que elegir una solución en la que el vendedor tenga la oportunidad de revisar los avisos antes de que sean enviados para evitar violentar a algún cliente (comentario 3).

Descripción Detallada

Caminos (flujos de decisión) en el caso de uso: “Pagar Factura”

Este camino refleja cómo detallamos los casos de uso a medida que los modelamos, aunque las descripciones en detalle de los casos de uso, en realidad seguro que son más grandes y que cubren más caminos.

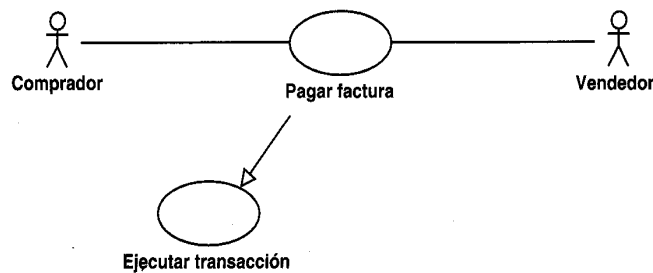
Precondición: el comprador ha recibido los bienes y servicios y al menos una factura del sistema. El comprador ahora planifica el pago de las facturas.	
Flujo de sucesos Camino básico 1. El comprador invoca al caso de uso para comenzar a hojear las facturas recibidas del sistema. El sistema verifica que el contenido de cada factura es consistente con las confirmaciones de pedido recibidas anteriormente (como parte del caso de uso Confirmar Pedido) y así indicárselo al comprador. La confirmación de pedido describe qué elementos serán enviados, cuándo, dónde y a qué precio. 2. El comprador decide planificar una factura para pagarla por banco, y el sistema genera una petición de pago para transferir el dinero a la cuenta del vendedor. Nótese que un comprador no puede planificar el pago de la misma factura dos veces. 3. Más tarde, si hay suficiente dinero en la cuenta del comprador, se hace un pago mediante transacción en la fecha planificada. Durante la transacción, el dinero se transfiere de la cuenta del comprador a la cuenta del vendedor, como se describe en el caso de uso abstracto Realizar Transacción (que es utilizado por Pagar Factura). Tanto el comprador como el vendedor tienen notificación del resultado de la transacción. El banco recoge unos cargos por la transacción, que se retiran de la cuenta del comprador. 4. La instancia del caso de uso finaliza.	Caminos alternativos 2. El comprador puede, en cambio, pedir al sistema que devuelva un rechazo de factura al vendedor. 3. Si no hay suficiente dinero en la cuenta, el caso de uso cancelará el pago y se lo notificará al comprador.
Poscondición: la instancia del caso de uso termina cuando la factura ha sido pagada o cuando el pago de la factura ha sido cancelado y no se ha hecho ninguna transferencia.	

► Descripción de funcionalidad compartida

Los casos de uso pueden organizarse especificando relaciones de generalización, inclusión y extensión entre ellos. Estas relaciones se utilizan para factorizar el comportamiento común (extrayendo ese comportamiento de los casos de uso en los que incluye) y para factorizar variantes (poniendo ese comportamiento en otros casos de uso, que lo extienden).

► Generalización entre casos de uso

Tomamos como ejemplo el caso de uso Pagar Factura, en el cual generalizamos el caso de uso mencionado, con el caso de uso Ejecutar Transacción. La secuencia de acciones descrita en el caso de uso Ejecutar Transacción es, por tanto, heredada en la secuencia descrita en Pagar Factura.



La relación de generalización entre los casos de uso Pagar Factura y Ejecutar Transacción.

La generalización entre casos de uso se representa con una línea continua con una punta de flecha vacía, al igual que la generalización entre clases

La generalización se emplea para simplificar la forma de trabajo y la comprensión del modelo de casos de uso y para re utilizar casos de uso "semifabricados" cuando reunimos casos de uso terminados requeridos por el usuario.

Cada caso de uso terminado se denomina



Caso de uso concreto

Los inicia un actor, y sus instancias constituyen una secuencia de acciones completa ejecutada por el sistema.



El caso de uso "semifabricado" existe solamente para que otros casos de uso lo re utilicen y se les llama casos de uso abstractos. Un caso de uso abstracto no puede instanciarse por sí mismo, pero una instancia de un caso de uso concreto también exhibe el comportamiento especificado por un caso de uso abstracto que lo (re)utiliza.

Si completamos un poco mas el concepto podríamos decir que....

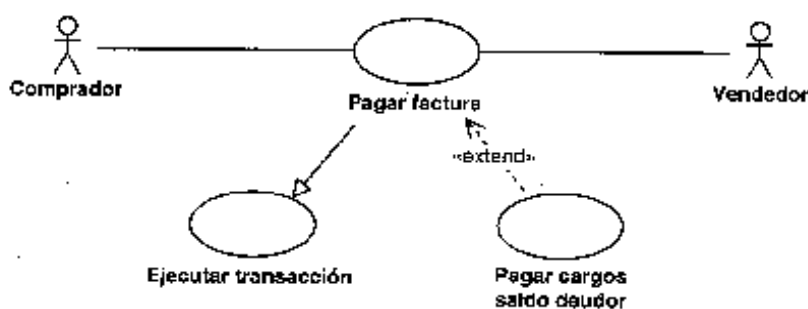
La generalización entre casos de uso es como la generalización entre clases. Aquí significa que el caso de uso hijo hereda el comportamiento y el significado del caso de padre; el hijo puede añadir o redefinir el comportamiento del padre; el hijo puede ser

colocado en cualquier lugar donde aparezca el padre (tanto el padre como el hijo pueden ser instancias concretas).

Por ejemplo, en un sistema bancario puede tenerse el caso uso Validar Usuario, responsable de verificar la identidad del usuario. Además, podría haber dos hijos especializados de este caso de uso (Comprobar clave y examinar retina), los cuales se comportarían como Validar Usuario y podrían utilizarse dondequiera que apareciera Validar Usuario, aunque ambos tengan su propio comportamiento (el primero comprobando una clave textual, el segundo comprobando los patrones diferenciadores en la retina del usuario).

» Relación de extensión entre casos de uso

En este tipo de relación, el caso de uso Pagar Factura es extendido por el caso de uso Pagar Recargos Saldo Deudor. La secuencia de las acciones descritas en el caso de uso Pagar Recargos Saldo Deudor se inserta de esta forma en la secuencia, descrita en Pagar Factura, si se da un descubierto en la cuenta (ésta es la condición para la extensión). De no ser así, no se activa.



La relación de extensión entre los casos de uso Pagar Factura y Pagar Cargos Saldo Deudor.

Debemos identificar criterios separados para realizar buenos casos de uso concretos, buenos casos de uso abstractos y buenas extensiones de casos de uso.



Los casos de uso concretos no deben describir comportamientos (significativos) que son compartidos con otros casos de uso concretos. Un caso de uso abstracto hace más fácil de especificar los casos de uso concretos ofreciendo comportamiento re utilizable y compartido. Una extensión de un caso de uso especifica comportamiento adicional. para otros casos de uso, independientemente de si su comportamiento es opcional u obligatorio.

» Una relación de inclusión

Este tipo de relación entre casos de uso significa, que un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso en el lugar especificado del caso base. El caso de uso incluido nunca se encuentra aislado, sino que es instanciado sólo

como parte de algún caso de uso base más amplio que lo incluye. La inclusión puede verse como que el caso de uso base toma el comportamiento del caso de uso proveedor.

La relación de inclusión se usa para evitar describir el mismo flujo de eventos repetidas veces, poniendo el comportamiento común en un caso de uso aparte (que será incluido por un caso de uso base).

La relación de inclusión es básicamente un ejemplo de delegación: se toma un conjunto de responsabilidades del sistema y se capturan en un sitio (el caso de uso a incluir en otros), a continuación se permite que otras partes del sistema (otros casos de uso) incluyan la nueva agregación de responsabilidades, siempre que se necesite usar esa funcionalidad.

Formalización de la descripción de los casos de uso

En algunas ocasiones, como cuando tenemos un sistema en tiempo real complejo, los casos de uso pueden ser muy complejos, por eso puede llegar a ser necesaria una técnica de descripción más estructurada. La interacción entre los actores y los casos de uso puede transitar, por ejemplo, por tantos estados y tantas transiciones alternativas que es casi imposible mantener consistente la descripción textual de los casos de uso.

Entonces puede ser útil utilizar una técnica de modelado visual para describir los casos de uso. Pueden utilizarse los diagramas de estado de UML para describir los estados de los casos de uso y las transiciones entre esos estados.

d-Prototipar la interfaz del usuario

El objetivo es construir un prototipo de interfaz de usuario, hasta el momento el analista solo ha trabajado con diagramas de casos de uso, descripciones detalladas para cada caso de uso. Ahora necesitamos diseñar la interfaz de usuario que permita al usuario llevar a cabo los casos de uso de manera eficiente.

Lo haremos en varios pasos. Comenzamos con los casos de uso e intentamos discernir qué se necesita de las interfaces de usuario para habilitar los casos de uso para cada actor. Esto es, hacemos un diseño lógico (datos) de la interfaz de usuario. Después, creamos el diseño físico de la interfaz de usuario y desarrollamos prototipos que ilustren cómo pueden utilizar el sistema los usuarios para ejecutar los casos de uso.

El resultado final de esta actividad es un conjunto de esquemas de interfaces de usuario y prototipos de interfaces de usuario que especifican la apariencia de esas interfaces de cara a los actores más importantes.



Crear el diseño lógico de una interfaz de usuario

Cuando los actores interactúan con el sistema, utilizarán y manipularán elementos de interfaces de usuario que representan atributos (de los casos de uso). A menudo estos son términos del glosario (por ejemplo, *balance de cuenta*, *fecha de vencimiento* y *titular de*

cuenta). Los actores pueden experimentar estos elementos de las interfaces de usuario como iconos, listas de elementos u objetos en un mapa de 2D, y pueden manipularlos por selección, arrastre o hablando con ellos.

El diseñador de interfaces de usuario identifica y especifica estos elementos actor por actor, recorriendo todos los casos de uso a los que el actor puede acceder, e identificando los elementos apropiados de la interfaz de usuario para cada caso de uso.

Un único elemento de interfaz de usuario puede intervenir en muchos casos de uso, desempeñando un papel diferente en cada uno. Así, los elementos de la interfaz de usuario pueden diseñarse para jugar varios roles.

► ***¿Pueden haber varios prototipos, quizá uno para cada actor?***

El esfuerzo de prototipado debe ser proporcional al valor del retorno esperado.

La validación de interfaces de usuarios a través de revisiones de prototipos y esquemas, en los primeros momentos, puede prevenir muchos errores que serán mas caros de corregir después.

No olvidemos entonces:



Como accesorio de los casos de uso, los analistas deben especificar también cuál será la apariencia de la interfaz de usuario cuando se lleven a cabo los casos de uso. La mejor forma de desarrollar esta especificación de interfaz de usuario, es esbozar varias versiones que muestren la información que se transferirá, discutir los esbozos con los usuarios, y construir prototipos concretos para que los usuarios los prueben.

Para el caso de uso Pagar Factura ilustraremos cómo podemos encontrar los elementos de la interfaz de usuario que el actor necesita para trabajar sobre la pantalla, así como qué clase de guía podría necesitar el actor mientras trabaja.

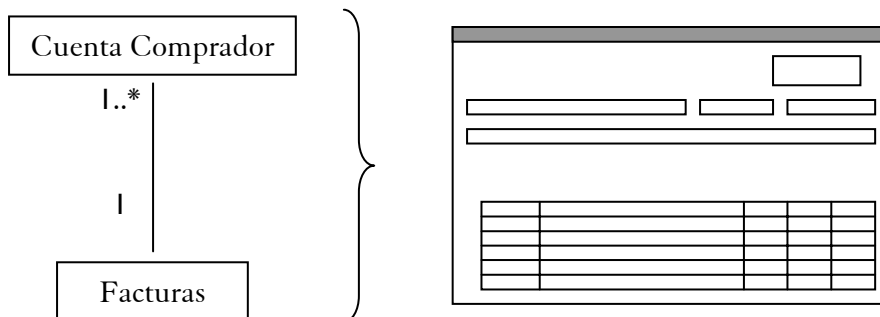
El actor trabajará seguramente con elementos de la interfaz de usuario como Facturas (identificadas a partir de una clase del dominio o de una entidad del negocio). Factura es por tanto un elemento de la interfaz de usuario como se ilustra en la Figura. Las facturas tienen una fecha de vencimiento, una cantidad a pagar y una cuenta destino. Todos estos atributos son necesarios para un actor, que debe decidir si paga o no la factura.

Además, como el actor está decidiendo qué facturas pagar, él o ella puede querer consultar la cantidad de dinero en su cuenta para evitar quedarse en números rojos. Esto es un ejemplo de la clase de guía o información que un actor necesita.

La interfaz de usuario debe pues mostrar las facturas según se planifican en el tiempo y cómo afectará el pago planificado de las facturas al saldo de la cuenta (como se indica en la asociación *cuenta comprador*; que se deriva de la asociación de la clase del dominio *comprador*. La cuenta es por eso otro aspecto de la interfaz de usuario. El saldo de la cuenta y su variación esperada en el tiempo a medida que se paguen las facturas.

Prototipo

El siguiente es un esquema, que relaciona, los datos guardados del sistema y un formato de interfaz, en los que se visualizan los mismos.

**e- Estructurar el modelo de casos de uso**

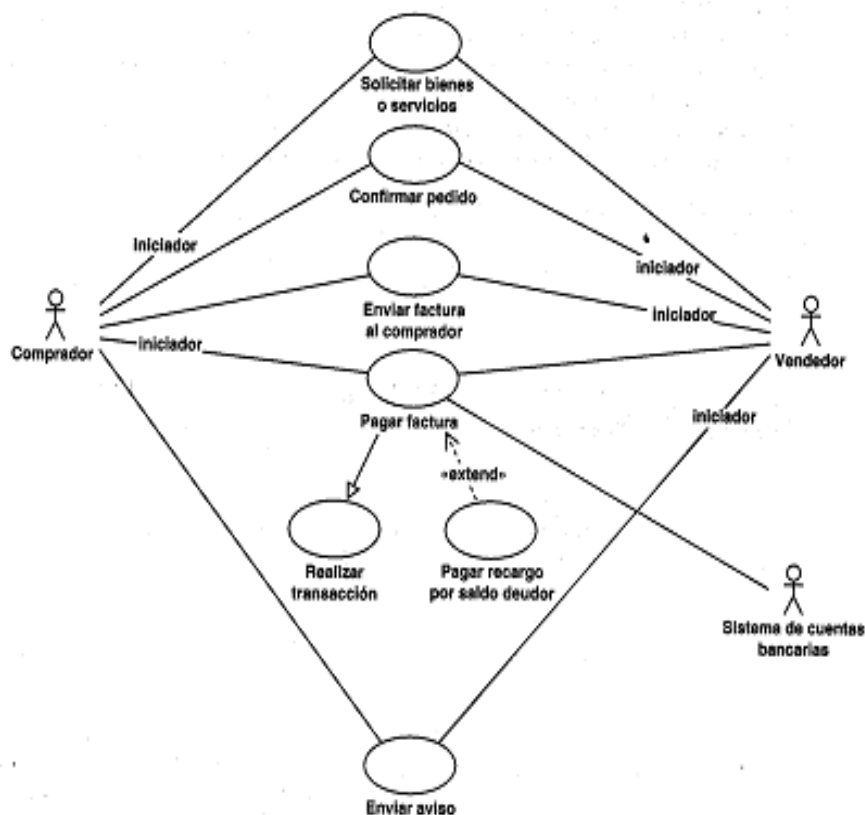
Antes de que tenga lugar esta actividad, el analista de sistemas ha identificado los actores y los casos de uso, dibujándolos en diagramas, y explicando el modelo de casos de uso como un todo. Los especificadores de casos de uso han desarrollado una descripción detallada de cada caso de uso. En este punto el analista de sistemas puede reestructurar el conjunto completo de casos de uso para hacer que el modelo sea más fácil de entender y de trabajar con él. El analista debe buscar comportamientos compartidos y extensiones.

Los resultados de la primera iteración a través de este flujo de trabajo consisten en una primera versión del modelo de casos de uso del SI, la descripción de los casos de uso y cualquier prototipo de interfaz de usuario asociado. Los resultados de cualquier iteración subsiguiente consistirán entonces en nuevas versiones de estos artefactos. Hay que recordar que todos los artefactos se completan y mejoran incrementalmente a través de las iteraciones.

Las distintas actividades en el modelado de casos de uso adoptan formas diferentes en diferentes fases del proyecto. Por ejemplo, cuando un analista de sistemas ejecuta la actividad de Encontrar Actores y Casos de Uso durante la **fase de inicio**, identificará muchos actores y casos de uso nuevos. Pero cuando la actividad se realice durante la fase de construcción, el analista hará sobre todo cambios secundarios en el conjunto de actores y casos de uso, tales como la creación de un diagrama de casos de uso que describa mejor el modelo de casos de uso desde una perspectiva en particular.

Podemos, por ejemplo, comenzar encontrando algunos casos de uso (la actividad de encontrar actores y casos de uso), después diseñar las interfaces de usuario (la actividad de Prototipar la Interfaz de Usuario), solamente para darnos cuenta de que necesitamos añadir un nuevo caso de uso (así que retrocederemos a la actividad de encontrar actores y casos de Uso, rompiendo la secuencia estricta marcada), y así sucesivamente. Una actividad puede ser retomada muchas veces, y cada una de éstas puede acarrear la ejecución de una sola fracción de la actividad.

Un diagrama del modelo de sistema de información, podría tener esta forma en las primeras iteraciones



4- Capturar requisitos no funcionales



Los requisitos no funcionales especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad, y fiabilidad .



La *fiabilidad* hace referencia a características como la disponibilidad, exactitud, tiempo medio entre fallos, defectos por miles de líneas de código, y defectos por clases.

Por último, algunos *requisitos no funcionales* son más genéricos y no pueden relacionarse con un caso de uso concreto o con una clase concreta del mundo real., Estos deberían gestionarse aparte en una lista de **requisitos adicionales**.

Síntesis :Modelos y diagramas de la Captura de Requisitos en el ciclo de vida.

		Modelos	Diagramas
Captura de Requisitos	Enunciar Requisitos Candidatos		Lista de características
	Comprender el Contexto	Modelo de dominio Modelo de Negocio	Diag. Clases de dominio Glosario de Términos Def. de actores del MN Diag. Caso de Uso de Negocio Descripción Caso Uso MN.
	Requisitos funcionales	Modelo CU Sistema de Información Prototipos de Interfaz	Def de actores del SI Diag. Caso Uso SI. Especificación de Caso de Uso de SI Prototipos
	Requisitos no Funcionales		Lista de requisitos adicionales

Cuadro resumen de las relaciones posibles

Pregunta	Extensión	Inclusión	Generalización
¿Cuál es la dirección de la Relación?	El use case adicional referencia al use case base.	El use case base referencia al use case adicional	El use case adicional (hijo) referencia al use case base (padre).
¿Tiene multiplicidad la relación?	Si, del lado del use case adicional.	No. Si se quiere incluir el mismo segmento más de una vez, es necesario que sea establecido en el use case base.	No.
¿Tiene una condición la relación?	Si.	No. Si se quiere expresar una condición en la inclusión es necesario decirlo explícitamente en el use case base.	No.
¿Es el use case adicional abstracto?	Frecuentemente si, pero no necesariamente.	Si.	Frecuentemente no, pero puede ser.
¿El use case base es modificado por el adicional?	La extensión implícitamente modifica el comportamiento del use case base	La inclusión modifica explícitamente el efecto del use case base.	Si el use case base (padre) es instanciado, no es afectado por el hijo. Para obtener los efectos de la adición, el use case adicional (hijo) debe ser instanciado.
¿Tiene que ser el use case base completo y significativo?	Si.	Junto con las adiciones, si.	Si es abstracto, no.
¿Tiene que ser el use case adicional completo y significativo?	No.	No.	Junto con el use case base (padre), si.
¿Puede el use case adicional acceder a atributos del use case base?	Si.	No. La inclusión es encapsulada y solo se "ve" a sí misma.	Si, por el mecanismo normal de herencia.
¿Puede el use case base acceder a atributos del use case adicional?	No. El use case base debe estar bien formado en ausencia de la adición.	No. El use case base solo conoce a cerca del efecto de la adición. La adición está encapsulada.	No. El use case base (padre) debe es este sentido estar bien formado en ausencia del adicional (hijo).

Autoevaluación

Resuelva las consignas que se presnetan para cada caso:

Caso práctico N°1

El hotel Resplandor, se encuentra ubicado en la ciudad de Santa Fe. El mismo aloja pasajeros de excursiones como clientes particulares que se hospedan por algunos días. Cada vez que un pasajero llega al hotel debe registrarse en la recepción, en donde completa una ficha de ingreso al hotel. Esa ficha contiene los datos de los pasajeros (nombre, apellido, estado civil, documento, domicilio, etc.). Una vez que se ha completado el tramite el conserje verifica las habitaciones que tiene disponibles y le asigna una, entregándole la llave de la misma.

Diariamente el pasajero solicita alguno de los servicios que se brindan en el hotel (lavandería, cafetería, etc.), para eso, este debe dirigirse al conserje, el cual registrará la solicitud del servicio que desea, y se lo asignará a la habitación correspondiente, emitiéndole al pasajero un recibo como constancia del mismo. Al finalizar el día, se ordenan y registran en una ficha realizada por habitación todos los servicios brindados junto con el correspondiente comprobante y se lo emite a contaduría.

Cuando el pasajero cancela su estadía, se emite un listado y la factura correspondiente, detallando los importes del alojamiento como así también el de los servicios prestados. Mensualmente se realiza un listado que se emite a Gerencia informando de la cantidad de habitaciones ocupadas, los importes cobrados y las anormalidades surgidas en ese período.

- 1- Derive los Casos de Uso del modelo de Negocio de la autoevaluación de la unidad II, al Modelo de Información
Seleccione un caso de Uso del modelo de Información y realice la especificación y su prototipo de Interfaz

Caso práctico N°2

En la Clínica XXX, los pacientes son atendidos de acuerdo al orden de llegada. Cuando el paciente ingresa, se le toman los datos con el objeto de buscar su historia clínica, en la cual el médico anotará: diagnóstico, terapia sugerida y análisis clínicos. En la clínica no se reciben mutuales, y el precio de cada consulta varía según la especialidad.

Cuando se cobra se emite un recibo para el paciente, y se registra el importe en un informe de ingreso a caja, que diariamente es enviado a Contaduría. Además del recibo, se le entrega al paciente una solicitud de consulta, que este le entregará al medico al ser atendido.

Todos los meses deben enviarse a Personal un reporte que contenga la cantidad de consultas atendidas para cada medico, para que le realicen la respectiva liquidación de honorarios.

- 1- Según su criterio, seleccione las herramientas necesarias y suficientes, para capturar en forma exitosa los requerimientos esenciales que se plantean en este caso práctico. Plantee alguna de ellas.

► Unidad IV

“Cómo construimos los modelos”

Objetivos

- Identificar los modelos y diagramas de trabajo
- Ejercitar los conocimientos adquiridos en las unidades anteriores.
- Resolver un caso práctico, en forma completa

Contenidos de la Unidad

¿Qué es entonces un modelo?.¿Por qué modelamos?

¿Las herramientas son esenciales en el proceso?

Diagramas.

Caso práctico: Enunciado

1. Modelado del Sistema de Negocio

1.3. Modelo de Use case del Sistema de Negocio (diagrama y descripciones de los use cases).

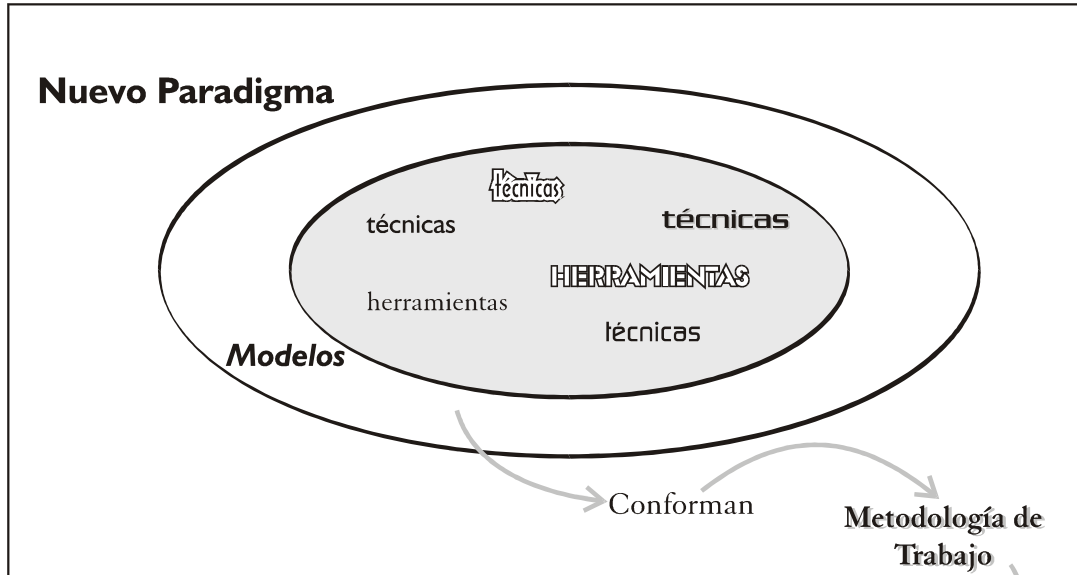
1.4. Modelo de Objetos del Dominio.

2. Modelado del Sistema de Información

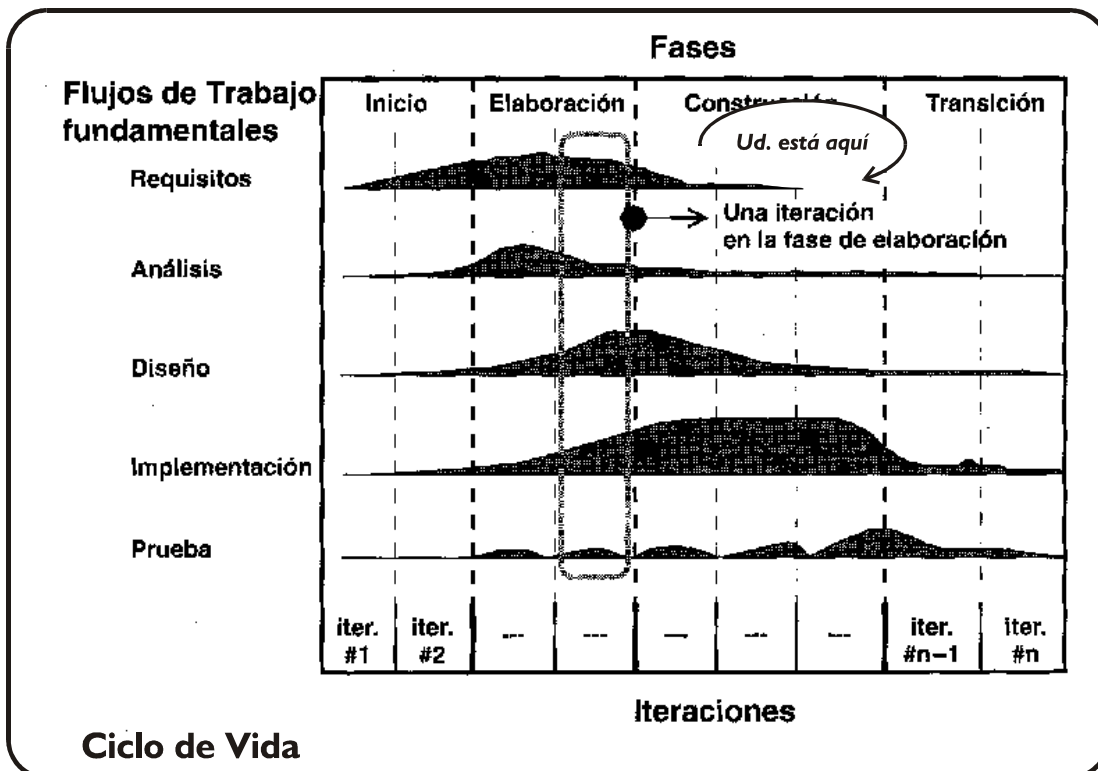
2.1. Modelo de use case (diagrama de use case y descripción de cada use case).

2.2. Descripción o Prototipo de Interfaz.

Mapa Conceptual



Proceso Unificado de Desarrollo de Software



Introducción

El modelado es una parte central de todas las actividades que conducen a la producción de buen software. Construimos modelos para comunicar la estructura deseada y el comportamiento de nuestro sistema. Construimos modelos para visualizar y controlar la arquitectura del sistema. Construimos modelos para comprender mejor el sistema que estamos construyendo, muchas veces descubriendo oportunidades para la simplificación y reutilización. Construimos modelos para controlar el riesgo.

¿Qué es, entonces un modelo?



Un modelo es una simplificación de la realidad

Un modelo proporciona los planos de un sistema. Los modelos pueden involucrar planos detallados, así como planos mas generales que ofrecen una visión global del sistema en consideración. Un buen modelo incluye aquellos elementos que tienen una gran influencia y omite aquellos elementos menores que no son relevantes para el nivel de abstracción dado.



Todo sistema puede ser descrito desde diferentes perspectivas utilizando diferentes modelos, y cada modelo es por tanto una abstracción semánticamente cerrada del sistema.

¿Por qué modelamos?

Hay una razón fundamental. *Construimos modelos para comprender mejor el sistema que estamos desarrollando.*

A través del modelado, conseguimos cuatro objetivos:

- 1- Los modelos nos ayudan a visualizar cómo es o queremos que sea un sistema.
2. Los modelos nos permiten especificar la estructura o el comportamiento de un sistema.
3. Los modelos nos proporcionan plantillas que nos guían en la construcción de un sistema.
4. Los modelos documentan las decisiones que hemos adoptado.

¿Construimos modelos de sistemas complejos porque no podemos comprender el sistema en su totalidad?



Hay límites a la capacidad humana de comprender la complejidad. A través del modelado, se reduce el problema que se está estudiando, centrándose sólo en un aspecto a la vez.

Decir que se debería modelar no significa que necesariamente se esté haciendo. De hecho, una serie de estudios sugieren que la mayoría de las organizaciones software hacen poco, o ningún, modelado formal. Si se representara el uso del modelado frente a la complejidad de un problema, se encontraría que cuanto, más sencillo es el proyecto, menos probable es que se haya utilizado un modelado formal.

Un desarrollador puede bosquejar una idea sobre una pizarra o un trozo de papel para visualizar una parte de un sistema, para trabajar a través de un escenario o el diseño de un mecanismo. No hay nada malo con ninguno de estos modelos. Si funcionan, deben utilizarse por todos los medios. Sin embargo, estos modelos *informales* son muchas veces ad hoc, y no proporcionan un lenguaje común que se pueda compartir fácilmente con otros. Así como existe un lenguaje común de planos para la industria de la construcción, un lenguaje común para la ingeniería eléctrica y un lenguaje común para el modelado matemático, también una empresa de software puede beneficiarse utilizando un lenguaje común para el modelado de software.

Cuanto más complejo sea un proyecto, más probable es que se fracase o no se construya el producto apropiado si no se hace nada de modelado. Todos los sistemas interesantes y útiles tienen una tendencia natural a hacerse más complejos con el paso del tiempo. Así que, aunque al inicio se pueda pensar que no es necesario modelar, cuando el sistema evolucione se lamentará esa decisión y entonces será demasiado tarde.

Modelado orientado a objetos

En el software hay varias formas de enfocar un modelo. Las dos formas más comunes son la perspectiva orientada a objetos y la perspectiva algorítmica.

La perspectiva algorítmica:

- ▶ La visión tradicional del desarrollo de software toma una perspectiva algorítmica. En este enfoque, el bloque principal de construcción de todo el software es el “procedimiento o función”. Esta visión conduce a los desarrolladores a centrarse en cuestiones de control y de descomposición de algoritmos grandes en otros más pequeños. No hay nada inherentemente malo en este punto de vista, salvo que tiende a producir sistemas frágiles. Cuando los requisitos cambian (lo harán!) y el sistema crece (lo hará!), los sistemas construidos con un enfoque algorítmico se vuelven muy difíciles de mantener.

La perspectiva orientada a Objetos

- ▶ La visión actual del desarrollo de software toma una perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas software es el objeto o clase. Si recordamos sencillamente, un objeto es una cosa, generalmente extraída del vocabulario del espacio del problema o del espacio de la solución; una clase es una descripción de un conjunto de objetos similares.

Todo objeto tiene:

- ▶ identidad (puede nombrarse o distinguirse de otra manera de otros objetos),
- ▶ estado (generalmente hay algunos datos asociados a él),
- ▶ y comportamiento (se le pueden hacer cosas al objeto, y él a su vez puede hacer cosas a otros objetos).

▶ **Por ejemplo**

Considérese una arquitectura sencilla de tres capas para un sistema de contabilidad, que involucre una interfaz de usuario, una capa intermedia y una base de datos. En la interfaz de usuario aparecerán objetos concretos tales como botones, menús y cuadros de diálogo. En la base de datos aparecerán objetos concretos, tales como tablas que representarán entidades del dominio del problema, incluyendo clientes, productos y otras. En la capa intermedia aparecerán objetos tales como transacciones y reglas de negocio, así como vistas de más alto nivel de las entidades del problema, tales como clientes, productos y pedidos.

Actualmente, el enfoque orientado a objetos forma parte de la tendencia principal para el desarrollo de software, simplemente porque ha demostrado ser válido en la construcción de sistemas en toda clase de dominios de problemas, abarcando todo el abanico de tamaños y complejidades. Más aún, la mayoría de los lenguajes actuales, sistemas operativos y herramientas son orientados a objetos de alguna manera, lo que ofrece más motivos para ver el mundo en términos de objetos.

¿Las herramientas son esenciales en el proceso?

Las herramientas soportan los procesos de desarrollo de software moderno. Hoy, es impensable desarrollar software sin utilizar un proceso soportado por herramientas.



El proceso y las herramientas vienen en el mismo paquete: las herramientas son esenciales en el proceso

Las herramientas influyen en el proceso



El proceso se ve influido fuertemente por herramientas. Las herramientas son buenas para automatizar procesos repetitivos, mantener las cosas estructuradas, gestionar grandes cantidades de información y para guiarnos a lo largo de un camino de desarrollo concreto

Sin un soporte por herramientas que automatice la consistencia a lo largo del ciclo de vida, será difícil mantener actualizados los modelos y la implementación. El desarrollo iterativo e incremental será más difícil, acabará siendo inconsistente, o requerirá una gran cantidad de trabajo manual para actualizar documentos y mantener así la consistencia.

El hecho de que existe un proceso es, por supuesto, el único motivo para necesitar herramientas. Las herramientas están ahí para automatizar tanto como sea posible del proceso.

La capacidad de automatizar un proceso depende de que tengamos una visión clara de qué casos de uso necesita cada usuario y qué artefactos necesita manejar. Un proceso automatizado proporciona un medio eficiente de permitir el trabajo concurrente del conjunto completo de los trabajadores, y proporciona una manera de comprobar la consistencia de todos los artefactos.

Las herramientas deben permitir a los trabajadores re utilizar tanto como sea posible; no deberían comenzar todo nuevo para cada alternativa probada.

En resumen, es esencial que las herramientas sean capaces tanto de soportar la automatización de actividades repetitivas como de gestionar la formación representada por la serie de modelos y artefactos, fomentando y soportando las actividades creativas que son el núcleo fundamental del desarrollo.

Diagramas

Cuando se modela algo, se crea una simplificación de la realidad para comprender mejor el sistema que se está desarrollando. Con UML, se construyen modelos a partir de bloques de construcción básicos, tales como clases, interfaces, colaboraciones, componentes, nodos, dependencias, generalizaciones y asociaciones.



Los diagramas son los medios para ver estos bloques de construcción. Un diagrama es una presentación gráfica de un conjunto de elementos, que la mayoría de las veces se dibuja como un grafo conexo de nodos (elementos) y arcos (relaciones).

Los diagramas se utilizan para visualizar un sistema desde diferentes perspectivas. Como ningún sistema puede ser comprendido completamente desde una única perspectiva, UML define varios diagramas que permiten centrarse en diferentes aspectos del sistema independientemente.

Los buenos diagramas hacen comprensible y accesible el sistema. La elección del conjunto adecuado de diagramas para modelar un sistema obliga a plantearse las cuestiones apropiadas sobre el sistema y ayuda a clarificar las implicaciones de las decisiones.

En el contexto del software hay cinco vistas complementarias que son las más importantes para visualizar, especificar, construir y documentar una arquitectura software:

1. La vista de casos de uso,
2. La vista de diseño,
3. La vista de procesos,
4. La vista de implementación y
5. La vista de despliegue.

Cada una de estas vistas involucra **modelado estructural** (modelado de cosas estáticas), así como **modelado de comportamiento** (modelado de cosas dinámicas).

Juntas, estas diferentes vistas capturan las decisiones más importantes sobre el sistema. Individualmente, cada una de estas vistas permite centrar la atención en una perspectiva del sistema para poder razonar con claridad sobre las decisiones.

Los diagramas se utilizarán de dos formas básicas:

- » para *especificar modelos* a partir de los cuales construir un sistema ejecutable (ingeniería directa) y
- » para *reconstruir modelos* a partir de partes de un sistema ejecutable (ingeniería inversa).

En cualquier caso, al igual que un arquitecto, se tenderá a construir los diagramas incrementalmente (elaborándolos uno a uno) e iterativamente (repitiendo el proceso de diseñar un poco, construir un poco).

Cuando se modelan sistemas reales, sea cual sea el dominio del problema, muchas veces se dibujan los mismos tipos de diagramas, porque representan vistas comunes de modelos comunes.

Normalmente, las partes estáticas de un sistema se representarán mediante uno de los cuatro diagramas siguientes:

- Diagramas de clases.
- Diagramas de componentes.
- Diagramas de despliegue.

A menudo se emplearán cinco diagramas adicionales para ver las partes dinámicas de un sistema:

1. Diagramas de casos de uso.
2. Diagramas de secuencia.
3. Diagramas de colaboración.
4. Diagramas de estados.
5. Diagramas de actividades.

Diagramas estructurales (ESTATICOS)

Los cuatro diagramas estructurales de UML existen para visualizar, especificar, construir y documentar los aspectos estáticos de un sistema. Se pueden ver los aspectos estáticos de un sistema como aquellos que representan su esqueleto y su andamiaje, ambos relativamente estables. Así como el aspecto estático de una casa incluye la existencia y ubicación de paredes, puertas, ventanas, tuberías, cables y conductos de ventilación, también los aspectos estáticos de un sistema software incluyen la existencia y ubicación de clases, interfaces, colaboraciones, componentes y nodos.

Diagramas de clases

Un *diagrama de clases* presenta un conjunto de clases, interfaces y colaboraciones, y las relaciones entre ellas. Los diagramas de clases son los diagramas más comunes en el modelado de sistemas orientados a objetos. Los diagramas de clases se utilizan para describir la vista de diseño estática de un sistema. Los diagramas de clases que incluyen clases activas se utilizan para cubrir la vista de procesos estática de un sistema.

Diagramas de componentes

Un *diagrama de componentes* muestra un conjunto de componentes y sus relaciones. Los diagramas de componentes se utilizan para describir la vista de implementación estática de un sistema. Los diagramas de componentes se relacionan con los diagramas de clases en que un componente normalmente se corresponde con una o más clases, interfaces o colaboraciones.

Diagramas de despliegue

Un *diagrama de despliegue* muestra un conjunto de nodos y sus relaciones. Los diagramas de despliegue se utilizan para describir la vista de despliegue estática de una arquitectura. Los diagramas de despliegue se relacionan con los diagramas de componentes en que un nodo normalmente incluye uno o más componentes.

Diagramas de comportamiento (DINAMICOS)

Los cinco diagramas de comportamiento de UML se emplean para visualizar, especificar, construir y documentar los aspectos dinámicos de un sistema. Se pueden ver los aspectos dinámicos de un sistema como aquellos que representan sus partes mutables. Así como los aspectos dinámicos de una casa incluyen flujos de aire y el tránsito entre las habitaciones, los aspectos dinámicos de un sistema software involucran cosas tales como el flujo de mensajes a lo largo del tiempo y el movimiento físico de componentes en una red.

Los diagramas de comportamiento de UML se organizan en líneas generales alrededor de las formas principales en que se puede modelar la dinámica de un sistema:

1. Diagramas de casos de uso. Organiza los comportamientos del sistema.
2. Diagramas de secuencia. Centrados en la ordenación temporal de los mensajes.
3. Diagramas de colaboración. Centrados en la organización estructural de los objetos que envían y reciben mensajes.
4. Diagramas de estados. Centrados en el estado cambiante de un sistema dirigido por eventos.
5. Diagramas de actividades. Centrados en el flujo de control de actividades.

► ***Un diagrama de casos de uso***

Representa un conjunto de casos de uso y actores (un tipo especial de clases) y sus relaciones. Los diagramas de casos de uso se utilizan para describir la vista de casos de uso estática de un sistema. Los diagramas de casos de uso son especialmente importantes para organizar y modelar el comportamiento de un sistema.

► ***Un diagrama de secuencia***

Es un diagrama de interacción que resalta la ordenación temporal de los mensajes. Un diagrama de secuencia presenta un conjunto de objetos y los mensajes enviados y recibidos por ellos. Los objetos suelen ser instancias con nombre o anónimas de clases, pero también pueden representar instancias de otros elementos, tales como colaboraciones, componentes y nodos. Los diagramas de secuencia se utilizan para describir la vista dinámica de un sistema.

► ***Diagrama de colaboración***

Es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensaje. Un diagrama de colaboración muestra un conjunto de objetos, enlaces entre esos objetos y mensajes enviados y recibidos por esos objetos. Los objetos normalmente son instancias con nombre o anónimas de clases, pero también pueden representar instancias de otros elementos, como colaboraciones, componentes y nodos. Los diagramas de colaboración se utilizan para describir la vista dinámica de un sistema.

► ***Un diagrama de estados***

Representa una máquina de estados, constituida por estados, transiciones, eventos y actividades. Los diagramas de estados se utilizan para describir la vista dinámica de un sistema. Son especialmente importantes para modelar el comportamiento de una interfaz, una clase o una colaboración. Los Diagramas de estados resaltan el comportamiento dirigido por eventos de un objeto, lo que es especialmente útil al modelar sistemas reactivos.

► ***Un Diagrama de actividades***

Muestra el flujo de actividades en un sistema. Una actividad muestra un conjunto de actividades, el flujo secuencial o ramificado de actividades, y los objetos que actúan y

sobre los que se actúa. Los diagramas de actividades se utilizan para ilustrar la vista dinámica de un sistema. Además, estos diagramas son especialmente importantes para modelar la función de un sistema, así como para resaltar el flujo de control entre objetos.

► ***Nota: Para Tener muy en Cuenta!!!***

Los dos diagramas que siguen son semánticamente equivalentes, como también sucede con los dos últimos, lo que significa que se puede modelar la dinámica de un sistema con un tipo de diagrama de comportamiento y luego transformarlo en otro tipo de diagrama sin pérdida de información. Esto permite razonar sobre diferentes aspectos de la dinámica del sistema. Por ejemplo, podría considerarse crear primero un diagrama de secuencia que ilustrase la ordenación temporal de los mensajes y luego transformarlo en un diagrama de colaboración para desarrollar las relaciones estructurales entre las clases cuyos objetos participan en la colaboración (también se puede pasar de los diagramas de colaboración a los diagramas de secuencia).

El nombre colectivo que se da a los diagramas de secuencia y los diagramas de colaboración es el de *diagramas de interacción*. Todos los diagramas de secuencia y los diagramas de colaboración son diagramas de interacción, y un diagrama de interacción es o bien un diagrama de secuencia o un diagrama de colaboración.

Igualmente, se podría considerar comenzar con un diagrama de estados para describir la respuesta del sistema ante los eventos externos y después convertirlo en un diagrama de actividades que se centre en el flujo de control (también se puede pasar de los diagramas de actividades a los diagramas de estados). La razón de que UML suministre estos diagramas semánticamente equivalentes es que el modelado de la dinámica de un sistema es difícil, ya menudo es necesario atacar los problemas complicados en exceso desde más de una perspectiva.

Enunciado y Resolución de un Caso Práctico



Es importante destacar que la resolución de este caso simulado, se llevara a cabo con algunos de los diagramas mencionados, no es necesario en el análisis y diseño de un sistema de Información, el uso de todos los recursos que nos ofrece PDUS. Cada diseñador debe elegir las herramientas , modelos , diagramas que considere convenientes para su equipo de trabajo.

La empresa Todo Viaje se dedica a brindar servicio de transporte diferencial puerta a puerta, ofreciendo comodidad, seguridad y puntualidad, para lograr satisfacer las necesidades de sus clientes. Además de realizar el envío de encomiendas, como un servicio adicional a sus clientes.

Es una empresa unipersonal, que nació hace ya 5 años en la ciudad de Río Segundo, en donde contaba con dos unidades, y el recorrido original era Río Segundo – Córdoba; dos

años después se amplió su recorrido hacia la ciudad de Pilar. Es así como la empresa incrementó su número de unidades llegando a tener una flota de 10 coches, en consecuencia también se amplió el número de empleados a 36, distribuidos en los diferentes departamentos incluyendo a los choferes, y estableciendo las distintas agencias que están funcionando actualmente. Finalmente, se radicó en Córdoba, estableciendo la administración, y dispuso cuatro agencias de ventas ubicadas en distintos lugares:

Agencia Córdoba Capital., Agencia Toledo, Agencia Río Segundo, Agencia Pilar.

La estructura que posee es la siguiente:

El mando superior es ejercido por un *Gerente General* encargado de la toma de decisiones estratégicas, define las políticas en la empresa, realiza la supervisión de las agencias y de todas las áreas a su cargo, haciendo conocer al personal las normas establecidas y responsabilidades pertinentes.

De él dependen los siguientes departamentos:

Administración de Personal: se ocupa de la liquidación de sueldos y jornales, de motivar al personal para que se sienta parte de la empresa, conocer sus problemas e intentar darles solución para que se sientan cómodos en su lugar de trabajo.

Departamento Servicios: conformado por los choferes encargados de realizar el transporte de pasajeros y rendir la venta de boletos efectuada a los pasajeros que suben durante el recorrido confeccionando la planilla de boletos vendidos.

Tesorería: es quien recibe las rendiciones de los choferes y de las agencias sobre la venta de boletos y envío de encomiendas. Paga al personal y maneja los fondos de la empresa. Además paga las compras efectuadas a los proveedores de los distintos insumos, y al taller mecánico por las reparaciones de las unidades.

Compras: realiza el trato con proveedores manteniendo actualizados sus datos, genera las órdenes de pedido según insumos necesarios y controla que sean entregados en tiempo y forma.

Departamento Ventas: controla el accionar de las agencias y realiza convenios con instituciones que solicitan el servicio de transporte para grupos o contingentes puntualmente, asignando unidades disponibles. Además asigna los distintos horarios a cubrir por los distintos choferes.

De ella dependen las agencias: Córdoba Capital, Toledo, Río Segundo, Pilar

Las funciones de todas las agencias son las siguientes: realizar las reservas y ventas de boletos, recibir las encomiendas a enviar y realizar su respectivo cobro, controlar el cumplimiento de los horarios de los choferes, realizar las solicitudes de insumos a compras, además de rendir a Tesorería los ingresos por los distintos conceptos de venta de boletos y envío de encomiendas.

Además cuenta con un Staff Contable que realiza el asesoramiento correspondiente y lleva los libros reglamentarios.

En lo que respecta al mantenimiento de las unidades, tienen un convenio firmado con el taller mecánico que realiza esta función. Cuando se ha producido un desperfecto de los coches se encarga de su reparación y entrega las unidades reparadas, remitiendo su correspondiente factura.

A continuación se describe el proceso a considerar para su estudio:

Trimestralmente se establecen los horarios a cubrir según lo autorizado por el ente regulador de transporte. Y se asigna mensualmente los choferes que cubrirán cada uno de los horarios y con qué unidades se harán.

En cada agencia el cliente efectúa la reserva personalmente o telefónicamente, en ningún caso se deja seña, y la tolerancia para presentarse y efectuar el viaje es de diez minutos previos al horario de partida, para ello se pide nombre, apellido y teléfono al cliente por cualquier novedad. Cuando el cliente se presenta a realizar el viaje, ya sea con reserva o no, se le efectúa el cobro correspondiente, aceptando como única forma de pago al contado, asignándole un número de asiento si se encuentra en una agencia en donde se inicia el recorrido, y sino se comunican por radio con el chofer de la unidad y se consulta la disponibilidad de asientos, de ser así se le reserva uno, comunicándole al chofer el hecho.

Durante el recorrido el chofer puede subir pasajeros siempre y cuando tenga lugar, respetando las reservas ya efectuadas hasta ese momento. Durante dicho recorrido y hasta el final va asentando en una planilla las ventas realizadas en el mismo. Al final de cada recorrido le rinde a Tesorería el dinero y la planilla con la venta de boletos. Cada agencia envía el detalle de cobros de sus ventas por ventanillas y de encomiendas, con el chofer que cubre cada horario del día, para que lo entregue junto a las planillas propias del chofer.

Cuando un coche se rompe, el chofer informa inmediatamente por radio a las respectivas agencias, de manera de enviar el refuerzo previsto para estos casos, y se comunican con el auxilio del taller mecánico para su reparación, el cual se encargará del problema, una vez que la unidad está en condiciones se comunica esto a Departamento Ventas para que sea considerada su puesta en circulación, y se entrega la factura que se remite a Tesorería para su posterior pago.

El encargado de Tesorería verifica cada mes las facturas pendientes de pago con el taller mecánico y emite los cheques para su correspondiente pago.

Cuando un cliente se presenta para enviar una encomienda, se registra el destino, se cobra lo que corresponde y se deja dicho paquete en espera hasta que pase el coche del próximo horario para su envío correspondiente. Cuando el chofer llega al final del recorrido entrega el paquete en la agencia para su posterior retiro.

En forma semanal la Tesorería efectúa un informe con los ingresos y egresos para rendir a Gerencia los movimientos realizados durante una semana.

El Gerente General ha observado algunos problemas en la empresa referidos a que las planillas de venta de boletos de las agencias y/o choferes no están bien confeccionadas, ya

sea con datos incompletos o erróneos, lo que no permite a Tesorería realizar un buen control de las rendiciones correspondientes. Además, la asignación de horarios no está bien realizada ya que ocurre a veces que un chofer tiene asignados horarios superpuestos, esto acarrea que se demoren en el cumplimiento de los horarios respectivos. A esto se agrega el hecho de que el Gerente no cuenta con información resumida y estadística de los viajes realizados, y de las rendiciones pertinentes, para una buena toma de decisiones y para conocer el accionar de la empresa.

Es por ello que desea contratar los servicios de un grupo de profesionales de sistemas de manera que le ofrezca una propuesta de solución a la situación planteada, además de estar interesado en automatizar los procesos de información previendo un presupuesto para la adquisición del equipamiento necesario.

Consideren que Uds. son ese grupo de profesionales, y teniendo en cuenta los requerimientos y problemas de información, plantee una propuesta de solución que incluya el desarrollo de los siguientes modelos:

1. Modelado del Sistema de Negocio
 - 1.5. Modelo de Use case del Sistema de Negocio (diagrama y descripciones de los use cases).
 - 1.6. Modelo de Objetos del Dominio.
2. Modelado del Sistema de Información
 - 2.1. Modelo de use case (diagrama de use case y descripción de cada use case).
 - 2.2. Descripción o Prototipo de Interfaz.

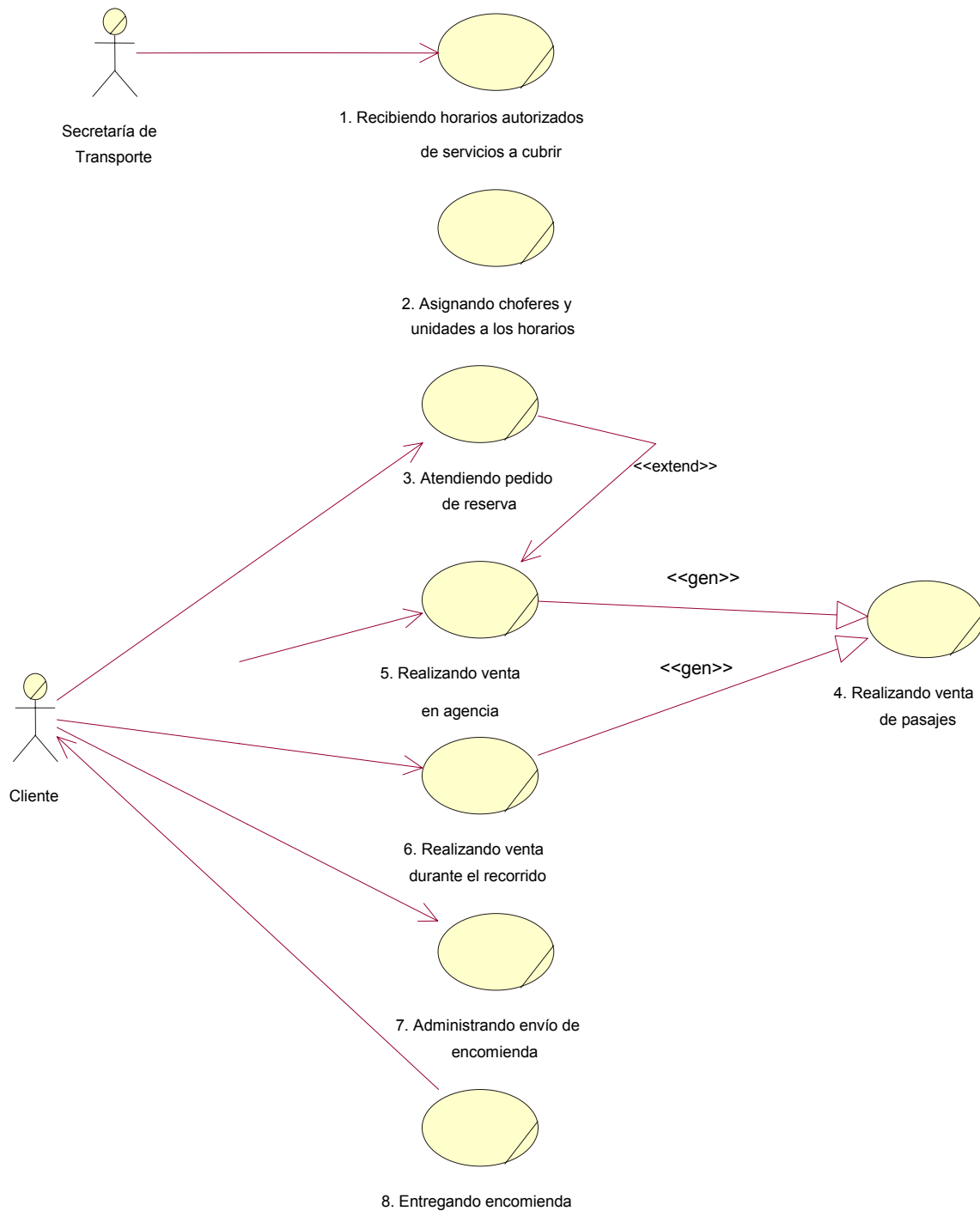
► ***Propuesta***

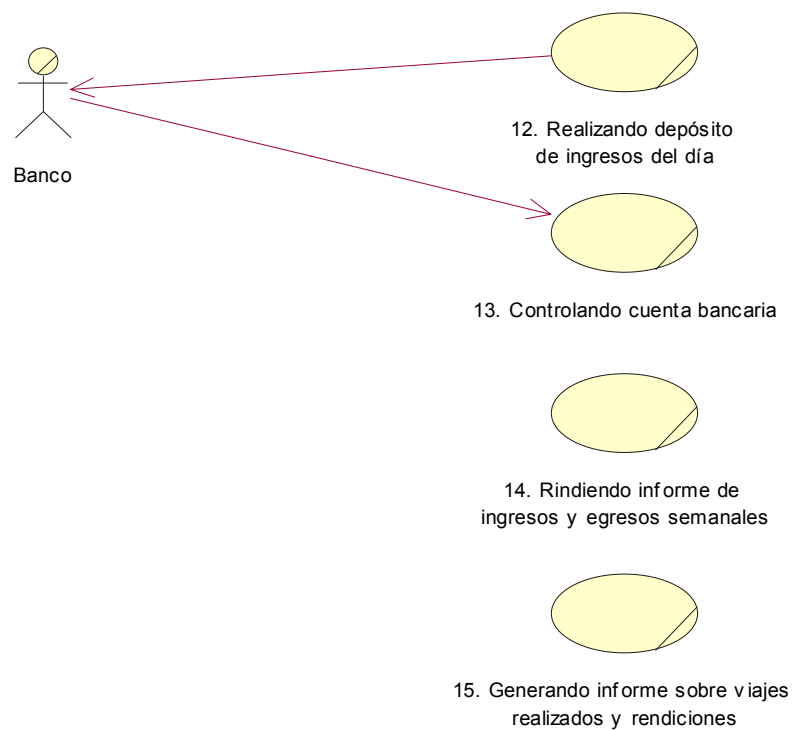
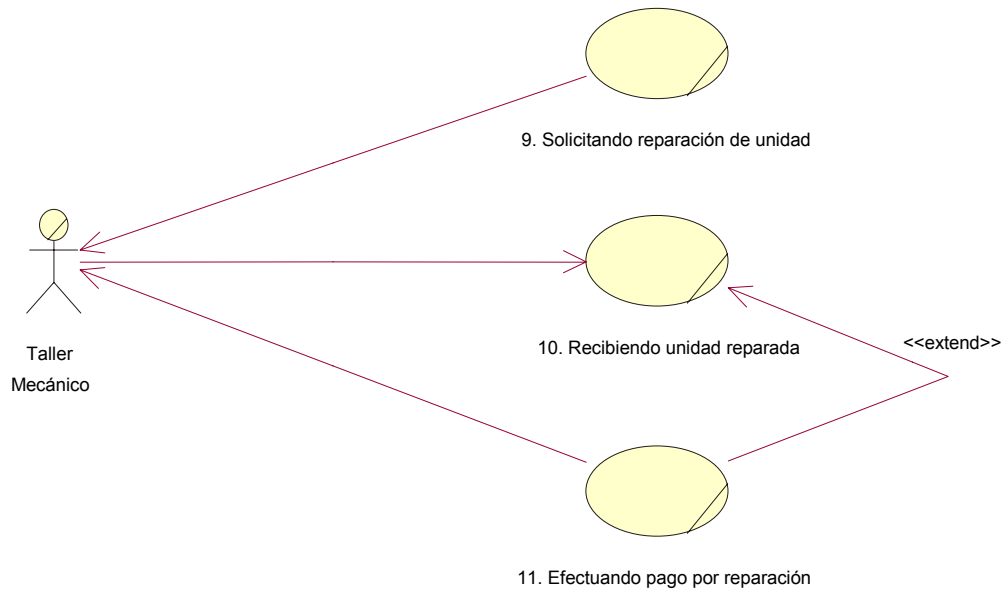
Nuestro sistema busca automatizar los procesos de venta, rendición de venta de pasajes y encomiendas, como así también la registración y generación de informes.

Es por ello que planteamos la instalación de una red entre las agencias, las cuales contarán con una terminal en línea y demás equipos necesarios para realizar la tarea requerida. Se contará además con un equipamiento mayor en la agencia de Córdoba Capital, donde serán los departamentos de Ventas, Tesorería y el Gerente General quienes tendrán acceso directo al sistema informático.

I. Modelado del sistema de Negocios

I.1. Modelo de Use-Case del Sistema de Negocios





Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Recibiendo horarios autorizados de servicios a cubrir	Nro. de Orden: 1
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Secretaría de Transporte	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Recibir y registrar los horarios de servicios autorizados por el Ente Regulador para cubrir los distintos recorridos con que cuenta la empresa.</i>		
Precondiciones:		
Post-Condiciones: <i>Se han registrado los nuevos horarios de los servicios de la empresa.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el empleado del Departamento Ventas recibe, en forma trimestral, los horarios autorizados por la Secretaría de Transporte. 2. El empleado del Departamento Ventas consulta los horarios de los servicios del último trimestre, verificando la existencia de cambios. 3. Se consultan cuáles son las unidades disponibles en el próximo trimestre. Estas unidades han sido previamente registradas por el empleado del Departamento Ventas. 4. De acuerdo a esto, se determinan los horarios del próximo trimestre. 5. El empleado registra los nuevos horarios de servicios a brindar en el próximo trimestre. 6. Se envía al Gerente General la notificación acerca de los horarios del próximo trimestre. Termina U-C		
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Asignando choferes y unidades a los horarios	Nro. de Orden: 2
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Actor Secundario:	
Tipo de Use Case:	<input type="checkbox"/> Concreto	<input checked="" type="checkbox"/> Abstracto
Objetivo: <i>Determinar, de forma mensual, cuáles serán los choferes y las unidades que serán asignadas a cada uno de los horarios.</i>		
Precondiciones:		
Post-Condiciones: <i>Se han asignado choferes y unidades a los servicios.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el empleado del Departamento Ventas, mensualmente, consulta los horarios de los servicios que brinda la empresa. 2. Se consultan los horarios disponibles de cada uno de los choferes de la empresa. Estos datos han sido registrados por el mismo empleado en su debido momento. 3. El empleado del Departamento Ventas consulta las unidades disponibles en el próximo mes. Los datos correspondientes a estas unidades han sido registrados con anterioridad por este mismo empleado. 4. Con los datos obtenidos, el empleado realiza la asignación de los choferes y las unidades a los distintos horarios. 5. Se registra, para cada horario, el chofer y la unidad asignada. 6. Se informa a los choferes los horarios de trabajo del próximo mes.		
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Atendiendo pedido de reserva	Nro. de Orden: 3
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Cliente	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Atender el pedido de reserva, y de ser necesario, llevar a cabo la misma.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha receptado y eventualmente realizado la reserva de el/los pasaje/s correspondientes de acuerdo a las preferencias del cliente.</i>		
Curso Normal	Alternativas	
<p>1. Comienza cuando un cliente realiza una solicitud de reserva, ya sea en forma personal o telefónicamente al empleado de la agencia.</p> <p>2. El empleado de agencia consulta al cliente el horario, fecha y recorrido para el cual desea realizar la reserva.</p> <p>3. Si es una agencia de inicio, el empleado de agencia consulta de inmediato la disponibilidad de pasajes para las condiciones establecidas.</p> <p>4. Si hay asientos disponibles, se solicita al cliente nombre, apellido y teléfono, además del lugar donde subirá al coche.</p> <p>5. Se registran los datos del cliente que realizó la reserva y los asientos del coche como reservados.</p> <p style="text-align: right;">Termina U-C</p>	<p>3.1 Si es una agencia intermedia, el empleado de agencia verifica si el coche de ese horario ya inició el recorrido.</p> <p>3.1.1. Si no se inició el recorrido, se verifica en la agencia la disponibilidad de asientos.</p> <p>3.1.2. Si ya se inició el recorrido, el empleado de agencia se comunica con el chofer y consulta si hay asientos disponibles.</p> <p>4.1. Si no hay asientos disponibles, ya sea en agencia o en recorrido, se informa al cliente de la situación. Termina U-C</p> <p>5.1. Si es una agencia intermedia y el recorrido ya se inició, se informa al chofer de la reserva, el destino y nombre del cliente, como así también el lugar donde subirá.</p> <p>5.2. El chofer registra la reserva en la planilla de ventas que posee.</p> <p style="text-align: right;">Termina U-C</p>	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende: Realizando venta en agencia		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Realizando venta de pasajes	Nro. de Orden: 4
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Actor Secundario:	
Tipo de Use Case:	<input type="checkbox"/> Concreto	<input checked="" type="checkbox"/> Abstracto
Objetivo: <i>Efectuar la venta de pasajes, y registrar la misma.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha producido y registrado la venta, a la vez que se ha entregado al cliente el comprobante de pago y el boleto correspondiente.</i>		
Curso Normal	Alternativas	
A. Comienza cuando el cliente se presenta para realizar el viaje. B. El empleado consulta al cliente si ha realizado alguna reserva. C. Si el cliente no cuenta con reserva, el empleado averigua acerca del destino en el cual desea realizar el viaje. D. Si hay asientos disponibles, el empleado consulta el precio del pasaje. E. El empleado realiza el cobro. F. Se entrega el comprobante de pago y el boleto al cliente. G. Se registra la venta.	C.1. Se verifican los datos personales del cliente. C.2. Se consulta cuál fue la reserva efectuada. D.1. De no haber asientos disponibles, se informa al cliente de la situación. Fin U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Realizando venta en agencia	Nro. de Orden: 5
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Cliente	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Realizar la venta de pasajes por ventanilla.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha vendido el/los pasaje/s y se ha efectuado el cobro por ventanilla.</i>		
Curso Normal	Alternativas	
1. = A 2. = B 3. = C 4. Se averigua el horario en el cual el cliente quiere viajar. 5. Si es una agencia de inicio, el empleado verifica la disponibilidad de asientos, consultando las ventas y reservas efectuadas. 6. = D 7. = E 8. = F 9. = G 10. Si es una agencia de inicio, el empleado asigna un número de asiento. 11. Si es una agencia de inicio, el empleado emite y entrega al chofer la planilla de venta de boletos y las reservas de asientos, a tener en cuenta antes de iniciar el recorrido. Termina U-C	3.1. Si el cliente cuenta con reserva, el empleado de agencia controla que el cliente haya concurrido con diez minutos de anticipación. 3.1.1. Si no concurrió en el horario establecido, se informa al cliente que ha perdido la reserva. 3.1.2. Se registra el asiento como disponible. 3.2. = C.1 3.3. = C.2 3.4. = E 3.5. = F Termina U-C 5.1. Si es una agencia intermedia, el empleado de agencia consulta si el coche de ese horario ya salió de la agencia anterior. 5.2. Si ya se inició el recorrido, el empleado de agencia se comunica con el chofer. 5.2.1. Si aun no se inició el recorrido, el empleado de agencia consulta si hay asientos disponibles. 5.3. Se consulta al chofer la disponibilidad de asientos. 5.4. El chofer consulta la planilla de ventas e informa al empleado si hay o no asientos desocupados. 6.1. = D.1 6.2. Si el cliente lo desea, se procede a realizar una reserva en un próximo horario. Se llama al U-C "Recibiendo pedido de reserva". Termina U-C 10.1. Si es una agencia intermedia, y el coche ya inició el recorrido, el empleado de agencia informa al chofer de la nueva reserva. 10.2. El chofer registra en la planilla el asiento como ocupado. Termina U-C	

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Realizando venta durante el recorrido	
Nro. de Orden:	6	
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Cliente	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Realizar la venta de pasajes a los clientes durante el recorrido.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha realizado la venta de pasajes durante el recorrido.</i>		
Curso Normal	Alternativas	
1. = A 2. = B 3. = C 4. El chofer del vehículo consulta si hay asientos disponibles, viendo en la planilla de venta y reservas los lugares ocupados. 5. = D 6. = E 7. El chofer registra en la planilla el cobro y el asiento que se ocupó 8. = F 9. El chofer comunica a las agencias intermedias próximas acerca de la venta del pasaje y qué asiento se ocupó. 10. = G 11. Al llegar a la agencia, el chofer entrega al empleado de la agencia el dinero y la planilla de ventas realizadas en el recorrido. 12. El empleado de la agencia verifica las ventas y controla el dinero entregado por el chofer. Termina U-C	3.1. Si el cliente tiene reserva, el chofer solicita los datos al cliente. 3.2. = C.1 3.3. = C.2 3.4. = E 3.5. = F Termina U-C 5.1. = D.1 5.2. Si el cliente lo desea, el chofer consulta al empleado de la agencia correspondiente si habrá asientos disponibles en un próximo horario. 5.3. De ser así, el chofer solicita los datos al cliente e informa a la agencia de la nueva reserva. Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización: Realizando Venta de Pasajes		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Administrando envío de encomienda	Nro. de Orden: 7
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Cliente	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Recibir, almacenar y realizar el traslado de la encomienda a destino.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha enviado la encomienda al destino correspondiente.</i>		
Curso Normal		Alternativas
1. Comienza cuando el cliente se presenta a la agencia para enviar una encomienda. 2. El empleado de la agencia consulta al cliente sobre cuál es el destino a donde desea enviar la encomienda. 3. El empleado de agencia recibe el paquete y pesa el mismo. 4. Se realiza el cálculo del precio del envío, de acuerdo al precio del kilómetro y el precio del kilogramo registrados anteriormente por el empleado del Departamento Ventas. 5. Se registra el destino de la encomienda, junto con los datos del destinatario y de quien lo envía. 6. Se realiza y registra el cobro. 7. Se entrega al cliente el comprobante de pago. 8. Se registra la agencia de destino en el paquete. 9. Se deja el paquete en espera hasta que pase el próximo coche. 10. Al llegar el coche a la agencia, el empleado entrega al chofer el listado de encomiendas (detallando destino, destinatario y remitente de la encomienda) y los paquetes correspondientes. 11. El chofer controla el listado y los paquetes que debe transportar. 12. El chofer traslada los paquetes de encomienda a destino, y llega a la agencia destino con el paquete de encomienda. 13. El chofer entrega al empleado de la agencia el paquete junto al listado de encomienda. 14. El empleado de la agencia verifica los datos del destinatario y encomienda, y guarda el paquete. 15. El empleado de la agencia registra que el paquete llegó a destino, entregando después el listado al chofer. Termina U-C		2.1. Si el destino no es cubierto por la empresa, se informa al cliente. Termina U-C
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Entregando encomienda a destinatario	Nro. de Orden: 8
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Cliente	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Entregar la encomienda a destinatario.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha entregado el paquete de encomienda al destinatario.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el cliente destinatario se presenta a la agencia para retirar la encomienda, el empleado de la agencia solicita los datos a la persona y el lugar de origen del envío. 2. Se verifica si el paquete se encuentra en la agencia. 3. Se entrega el paquete al cliente. 4. Se hace firmar el comprobante de entrega al cliente. 5. Se registra la entrega del paquete. Termina U-C	2.1. De no hallarse el paquete, se realiza una consulta a la agencia de origen. 2.2. Se informa al cliente acerca del estado del envío. Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Solicitando reparación de unidad	Nro. de Orden: 9
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Taller Mecánico	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Realizar, ante el desperfecto en la unidad, el aviso correspondiente a la agencia y solicitar la reparación al taller mecánico.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha asignado el refuerzo y se ha informado al auxilio del taller mecánico acerca de la rotura del coche.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el chofer informa al empleado de la agencia más cercana acerca de la rotura del coche. 2. El empleado de la agencia consulta las unidades y choferes disponibles. 3. El empleado de la agencia asigna la unidad de refuerzo y el chofer a cargo. 4. El empleado informa al chofer del refuerzo sobre la situación. 5. Se registran los datos de la unidad que sufrió el desperfecto y el lugar donde se produjo. 6. Se comunica al auxilio mecánico del taller acerca de la unidad rota. 7. El empleado de la agencia consulta los horarios de la unidad rota para así reasignar estos horarios a otra unidad. 8. Cuando la unidad de refuerzo llega al lugar, se realiza el traspaso de los pasajeros y las encomiendas, junto con los listados de encomiendas y planilla de ventas y reservas. Termina U-C	6.1. Si es la primera vez que se solicita el servicio del taller, se registran los datos del mismo.	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Recibiendo unidad reparada	Nro. de Orden: 10
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Taller Mecánico	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Recibir la unidad reparada y la factura correspondiente al servicio realizado por el taller.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha puesto a disposición de la empresa la unidad reparada.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el taller mecánico notifica al empleado del Departamento Ventas que la unidad se encuentra reparada. 2. El empleado del Departamento Ventas solicita al taller mecánico el envío de dicha unidad reparada. 3. El empleado de Ventas controla que la unidad ingrese a la empresa. 4. Se registra como disponible la unidad recibida y reparada. 5. El empleado de Ventas recibe la factura por la reparación realizada. 6. Se envía dicha factura al encargado de Tesorería. 7. El encargado de Tesorería registra los datos de la factura y su próxima fecha de pago. Termina U-C	7.1. Si la fecha corriente se corresponde con el final del mes, se llama al U-C "Efectuando pago por reparación" Termina U-C	
Asociaciones de Extensión: Efectuando pago por reparación		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Efectuando pago por reparación	Nro. de Orden: 11
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Taller Mecánico	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo:	<i>Realizar el pago correspondiente a las reparaciones efectuadas por el taller mecánico.</i>	
Precondiciones:		
Post-Condiciones:	<i>Se ha efectuado el pago pendiente, emitiendo el cheque correspondiente al taller mecánico.</i>	
Curso Normal	Alternativas	
1. Comienza cuando el encargado de Tesorería, al finalizar el mes, consulta las facturas pendientes de pago al taller mecánico. 2. El encargado de Tesorería consulta los datos del taller mecánico. 3. Se calcula el monto total por el cual se debe emitir el cheque. 4. Se emite el cheque al taller mecánico por el total de la deuda. 5. El encargado de Tesorería registra los datos del cheque. 6. Se entrega el cheque al taller mecánico. 7. Se actualiza el estado de la deuda con el taller. Termina U-C	1.1. Si no existen facturas pendientes de pago Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:	Recibiendo unidad reparada	
Use Case de Generalización:		

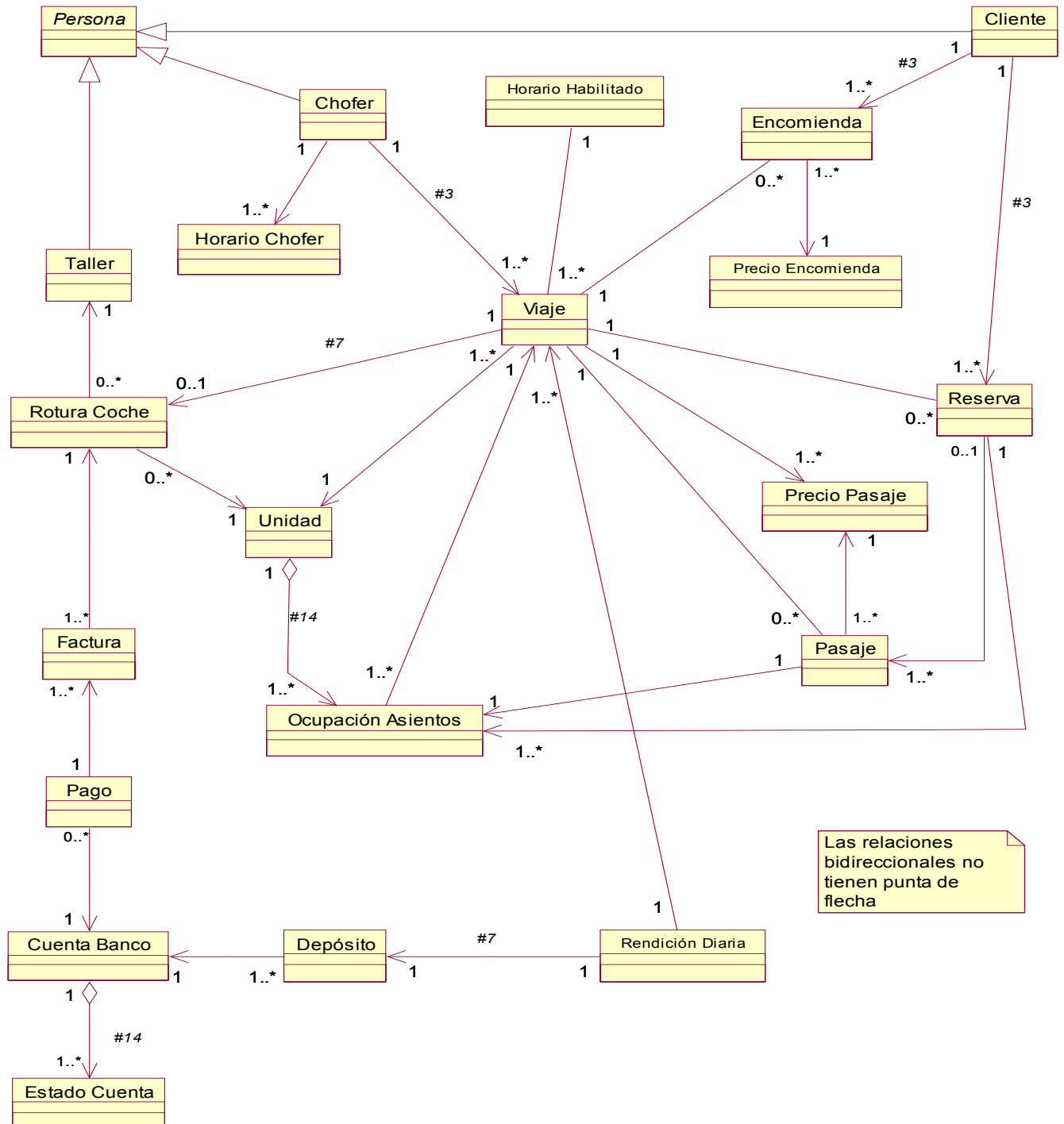
Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Realizando depósito de ingresos del día	Nro. de Orden: 12
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Banco	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Contabilizar el dinero que ingresó a cada agencia en el día y verificar si corresponde con lo registrado.</i>		
Precondiciones:		
Post-Condiciones: <i>Se han controlado los ingresos del día de cada agencia y se ha realizado el depósito correspondiente.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el empleado de agencia, al final del día, verifica los ingresos de dinero de la jornada. 2. El empleado de la agencia consulta los ingresos de dinero por venta de pasajes en ventanilla y durante el recorrido. 3. Se consulta el monto de dinero que ingresó por envío de encomiendas. 4. Se calcula el monto total de dinero que ingresó por todo concepto. 5. El empleado cuenta el dinero que posee en caja en ese momento. 6. Se controla si el dinero en caja coincide con el monto establecido. 7. Se confecciona la boleta de depósito correspondiente al monto determinado. 8. Se realiza el depósito de dinero que ingresó en el día. 9. Se registra monto, fecha y datos de la boleta de depósito. 10. El empleado de la agencia envía el comprobante de depósito al encargado de Tesorería. 11. El encargado de Tesorería registra la recepción de la boleta de depósito. Termina U-C	6.1. Si existe diferencia alguna entre los montos, se registra la misma.	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Controlando cuenta bancaria	Nro. de Orden: 13
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Banco	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Recibir y registrar el estado de cuenta, además de controlar los depósitos realizados por cada uno de las agencias.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha recibido el informe de estado de cuenta y efectuado el control de depósitos.</i>		
Curso Normal		Alternativas
1. Comienza cuando el encargado de Tesorería recibe, semanalmente, el estado de cuenta del banco. 2. El encargado de Tesorería consulta los montos diarios depositados por cada agencia. 3. Se verifica el estado de cuenta actual con los montos depositado diariamente y los pagos efectuados. 4. Se registra el estado de cuenta recibido. Termina U-C		2.1. Si es el primer estado de cuenta del mes, se consultan los pagos efectuados al taller en el mes anterior. 4.1. Si existe alguna diferencia, el encargado de Tesorería envía una notificación al Gerente General acerca de la situación. Termina U-C
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Rindiendo informe con ingresos y egresos semanales	Nro. de Orden: 14
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Actor Secundario:	
Tipo de Use Case:	<input type="checkbox"/> Concreto	<input checked="" type="checkbox"/> Abstracto
Objetivo: <i>Generar un informe acerca de los ingresos y egresos semanales de la empresa.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha emitido el informe de rendición de ingresos y egresos.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el Gerente General solicita, a Tesorería, el informe semanal de ingresos y egresos. 2. El encargado de Tesorería recibe la solicitud del informe. 3. El encargado de Tesorería consulta los ingresos y egresos que se han producido en la semana. 4. Se consulta el estado de la cuenta del banco. 5. Se confecciona el informe solicitado, adjuntando el informe de estado de cuenta actual. 6. Se envía el informe al Gerente General. Termina U-C		
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input checked="" type="checkbox"/> Negocio	<input type="checkbox"/> Sistema de Información
Nombre del Use Case	Generando informe sobre viajes realizados y rendiciones	Nro. de Orden: 15
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Actor Secundario:	
Tipo de Use Case:	<input type="checkbox"/> Concreto	<input checked="" type="checkbox"/> Abstracto
Objetivo: <i>Generar un informe mensual que contenga los datos acerca de las rendiciones efectuadas y los viajes realizados.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha emitido el informe mensual de viajes realizados y rendiciones al Gerente General.</i>		
Curso Normal	Alternativas	
1. Comienza cuando el Gerente General, de forma mensual, solicita el informe de rendiciones por viajes y encomiendas. 2. El encargado de Tesorería recibe la notificación. 3. El encargado de Tesorería consulta los datos de los viajes realizados en el último mes. 4. Se consultan cuáles fueron las ventas realizadas y las encomiendas recibidas. 5. Se consultan los ingresos recibidos por cada concepto en el último mes. 6. Se realizan los cálculos correspondientes y se confecciona el informe de viajes y rendiciones. 7. Se envía el informe de viajes y rendiciones al Gerente General. <p style="text-align: center;">Termina U-C</p>		
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

► 1.2 Modelo de Objetos del Dominio



Persona
nombre
apellido
dirección
teléfono

Unidad
númeroUnidad
modelo
patente
marca
cantidadAsientos
fechaCompra
Estado(reparación)
↑OcupaciónAsiento

OcupaciónAsiento
númeroAsiento
estadoAsiento
↑Viaje

Depósito
↑CuentaBanco
nroBoletaDepósito
fecha
importeDepositado

Encomienda
númeroEncomienda
origen
destino
peso
nombreDestinatario
fechaEnvío
monto
estado
fechaRecepción
horaRecepción
horaLlegada
fechaLlegada
fechaEntregaCliente
↑PrecioEncomienda
↑Viaje

Cliente
númeroDocumento
tipoDocumento
tipoCliente
↑Reserva
↑Encomienda

HorarioHabilitado
horarioPartida
horarioLlegada
origen
destino
díaSemana
fechaDesde
FechaHasta
↑Viaje

Reserva
númeroReserva
fechaReserva
lugarAscenso
cantidadPersonas
↑Viaje
↑OcupaciónAsiento
↑Pasaje

CuentaBanco
nombre
númeroSucursal
dirección
teléfono
númeroCuenta
↑EstadoCuenta

RoturaCoche
fechaDesperfecto
horaDesperfecto
lugarDesperfecto
fechaReparación
↑Taller
↑Unidad

Chofer
númeroDocumento
tipoDocumento
fechaNacimiento
fechaIngreso
fechaBaja
motivoBaja
↑HorarioChofer
↑Viaje

Pasaje
númeroPasaje
agencia
↑PrecioPasaje
↑OcupaciónAsiento
↑Viaje

PrecioEncomienda
codPrecio
kmDestino
precioKilómetro
precioKilogramo

Factura
númeroFactura
descripción
fecha
monto
estado
↑Pago
↑RoturaCoche

Estado Cuenta
fecha
saldoEsperado
saldoActual
observaciones

PrecioPasaje
localidadOrigen
localidadDestino
precioPasaje
porcentajeDescuento

Taller
razónSocial
númeroCUIT

Viaje
númeroViaje
fecha
refuerzo (S/N)
↑PrecioPasaje
↑Reserva
↑Pasaje
↑Unidad
↑Horario-habilitado
↑RoturaCoche
↑Encomienda

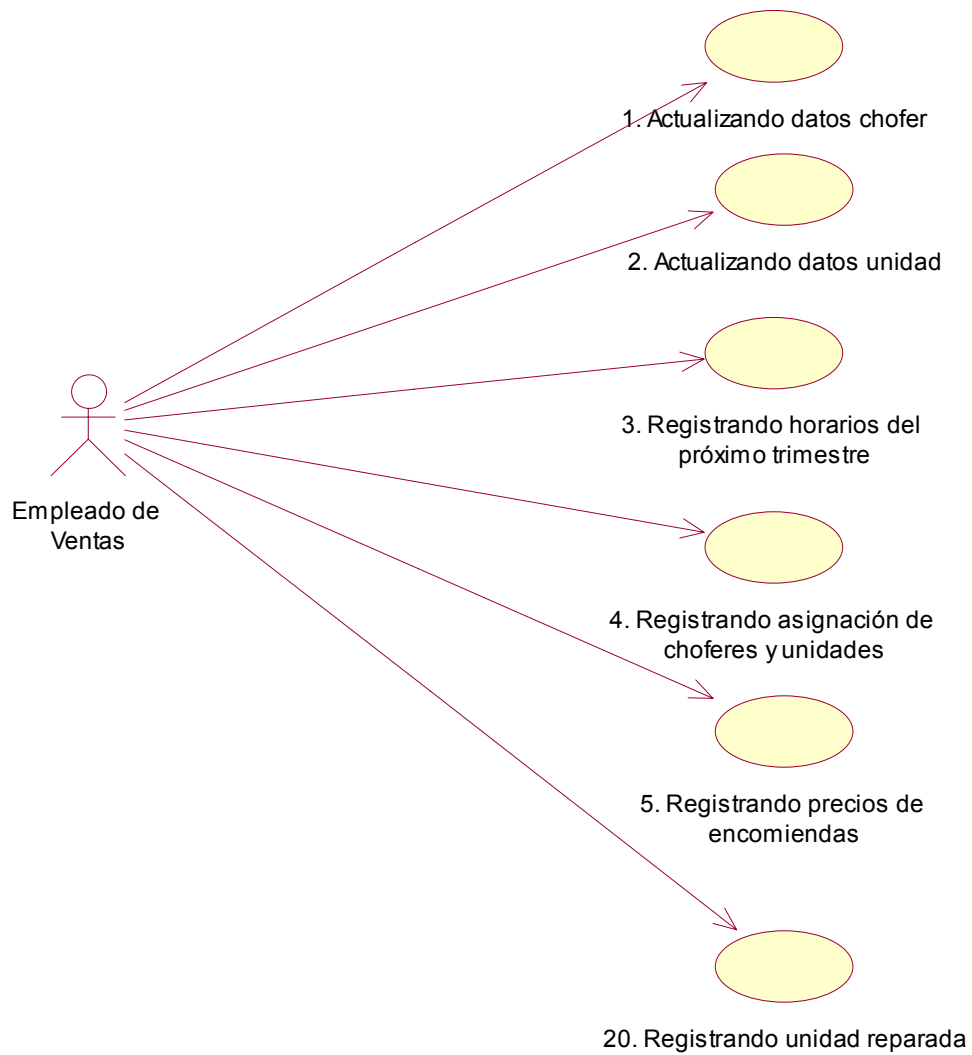
Rendición Diaria
fecha
montoCaja
montoEsperado
concepto
observaciones
↑Viaje
↑Depósito

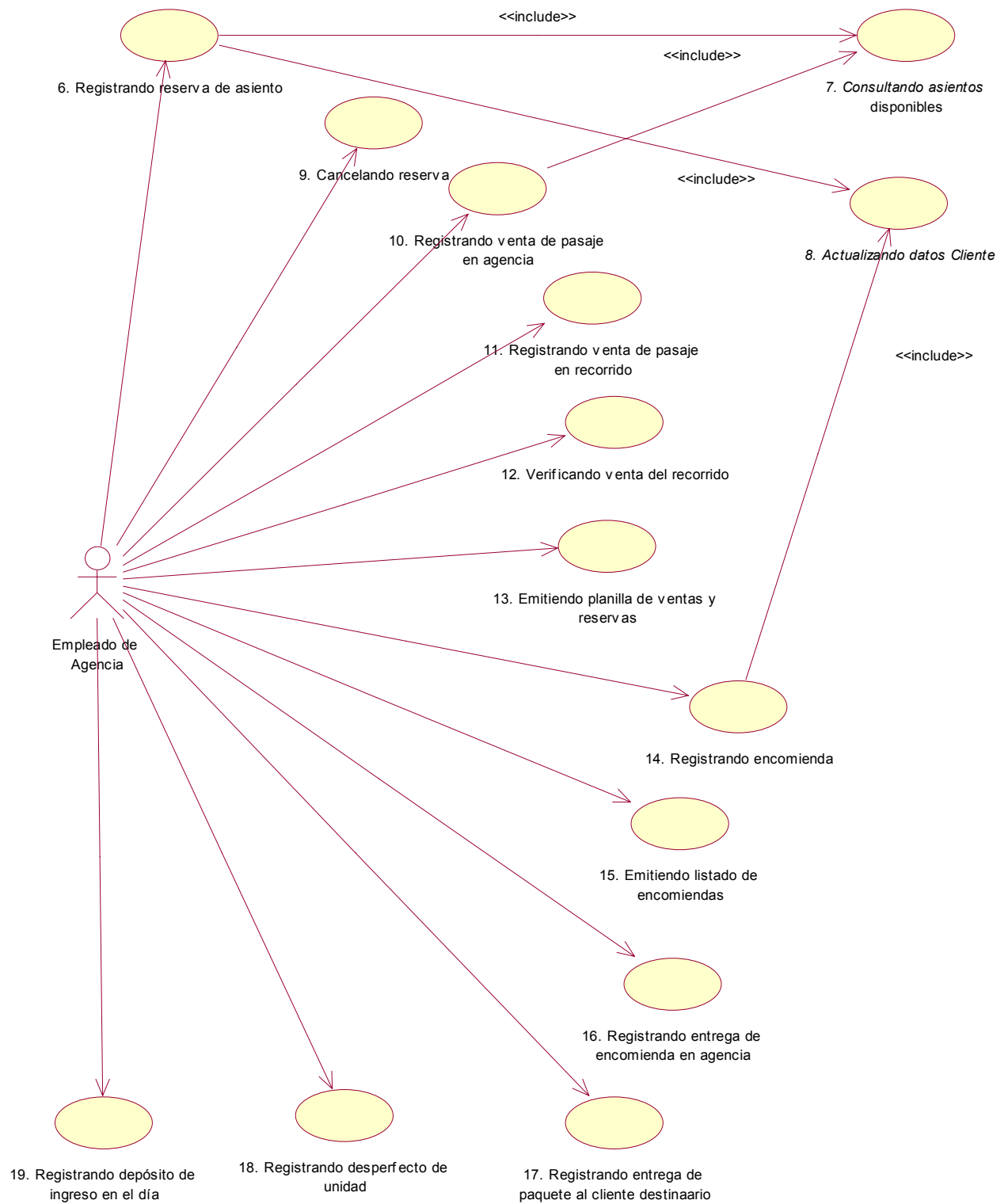
Pago
montoTotal
fechaPago
númeroCheque
fechaValidezCheque
fechaCobroCheque
nroReciboPago
↑CuentaBanco

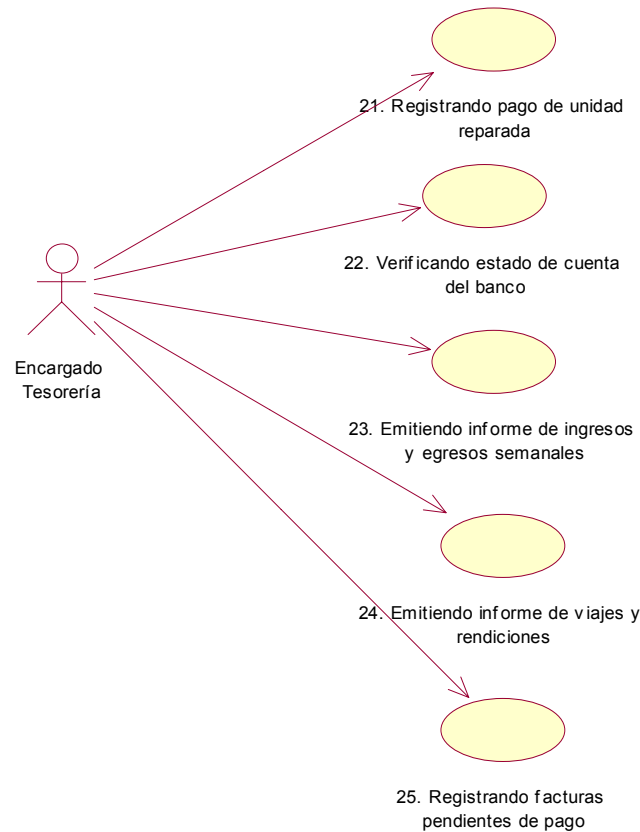
HorarioChofer
díaSemana
horaDesde
horaHasta
fechaDesde
fechaHasta

2. Modelado del sistema de Información

► 2.1 Modelo de Casos de Uso del Sistema de Información : Diagrama de Casos de Uso







► 2.2 Descripción de cada Caso de Uso y Prototipos de Interfaz

A partir de este punto se han seleccionado un grupo de Casos de Uso y las descripciones y prototipos de interfaz se desarrollarán sobre dicho grupo, a modo de ejemplo, y en el resto de los flujos de trabajo también se desarrollarán los distintos modelos teniendo como referencia este grupo de casos de uso.

Los Casos de Uso considerados son los siguientes:

- U-C 1: Actualizando datos del chofer
- U-C 6: Registrando reserva de asiento
- U-C 7: Consultando asientos disponibles
- U-C 8: Actualizando datos de cliente
- U-C 9: Cancelando reserva
- U-C 10: Registrando venta de pasaje en agencia
- U-C 11: Registrando venta de pasaje en recorrido
- U-C 12: Verificando venta en recorrido
- U-C 13: Emitiendo planilla de ventas y reservas
- U-C 24: Emitiendo informe de viajes y rendiciones

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case Actualizando datos chofer	Nro. de Orden: 1	
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal Empleado de Ventas	Actor Secundario:	
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Registrar o actualizar los datos de cada chofer de la empresa.</i>		
Curso Normal	Alternativas	
<p>1. Comienza cuando el empleado de Ventas necesita actualizar los datos de un chofer ingresando a la pantalla "Actualización de datos de Chofer".</p> <p>2. El empleado de Ventas desea ingresar los datos de un nuevo chofer al sistema, selecciona la opción Nuevo Chofer</p> <p>3. El empleado de Venta ingresa el apellido y nombre, dirección, teléfono, fecha de nacimiento y fecha de ingreso.</p> <p>4. Al momento de ingresar el tipo de documento se muestra una lista desplegable de opciones, de la cual se deberá seleccionar una de ellas.</p> <p>5. Se ingresa el número de documento, y en una grilla el día de la semana, el horario de entrada y el horario de salida que poseerá el chofer para cada día</p> <p>6. Se selecciona la opción Aceptar.</p> <p>7. Se actualiza el campo legajo luego de Aceptar.</p> <p>8. Se muestra un mensaje con el número de legajo asignado y la confirmación de la registración de los datos del chofer. Termina U-C</p>	<p>2.1. El empleado de Ventas necesita modificar los datos de un chofer, selecciona la opción Modificar.</p> <p>2.1.1. Se ingresa el número de legajo del chofer</p> <p>2.1.2. Se acepta el mismo, mostrando todos los datos personales del chofer.</p> <p>2.1.3. Se modifican los datos necesarios.</p> <p>2.1.4. Se selecciona la opción Aceptar.</p> <p>2.1.4.1. Si no se desea registrar los datos modificados se selecciona la opción Cancelar. FUC</p> <p>2.1.5. Se muestra la confirmación de la modificación .FUC</p> <p>2.2. El empleado de Ventas necesita dar de baja a un chofer del sistema, selecciona la opción Eliminar</p> <p>2.2.1. Se ingresa el número de legajo del chofer.</p> <p>2.2.2. Se acepta el mismo, mostrando los datos personales del chofer.</p> <p>2.2.3. Se ingresa la fecha de egreso y el motivo por el cual se produce la baja.</p> <p>2.2.4. Se selecciona la opción Aceptar.</p> <p>2.2.4.1. Si no se desea registrar los datos ingresados se selecciona la opción Cancelar. FUC</p> <p>2.2.5. Se muestra la confirmación de la baja, el chofer se considera inactivo. FUC</p> <p>6.1. Si no se desean registrar los datos del chofer, se selecciona la opción Cancelar.</p> <p>Termina U-C</p>	

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case Registrando reserva de asiento		Nro. de Orden: 6
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Empleado de Agencia	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Registrar la reserva de pasaje realizada por el cliente</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha llevado a cabo la registración de la reserva del asiento correspondiente y se ha asignado un número de reserva</i>		
Curso Normal		Alternativas
1. Comienza cuando el empleado de Agencia necesita registrar una reserva ingresando a la pantalla "Reserva de asiento". 2. Se selecciona la opción Consultar Asientos, se llama al use case "Consultando asientos disponibles". 3. Se actualizan los campos: origen , destino, fecha, hora, número(s) de asiento(s), precio del pasaje y precio total. 4. Se selecciona la opción Registrar cliente, se llama al use case "Registrando Cliente". 5. Se actualizan los campos nombre , apellido y teléfono del cliente que está realizando la reserva. 6. Se registra el lugar donde sube el cliente. 7. Se selecciona la opción Aceptar. En este momento se asigna un número de reserva, el cual aparece en el campo "Nº de Reserva". Termina U-C		7.1. Si no se desea registrar la reserva correspondiente, se selecciona la opción Cancelar. Termina U-C
Asociaciones de Extensión:		
Asociaciones de Inclusión: 7. Consultando datos disponibles 8. Registrando cliente		
Use Case donde se incluye:		
Use Case donde se extiende: 10. Registrando venta de pasaje		
Use Case de Generalización:		

Actualizacion de Datos de Chofer

Datos Chofer

Nro Legajo Fecha Nacimiento

Apellido Tipo Documento - Numero

Nombre Fecha Ingreso

Direccion Fecha Egreso

Telefono Motivo Egreso

Horario de Trabajo

	Dia	Desde	Hasta

Nuevo Chofer Modificar Eliminar Aceptar Cancelar

Reserva de Asiento

Datos Viaje

Origen Lugar Ascenso

Destino Nro Reserva

Fecha

Hora

Datos Pasajero

Apellido

Nombre

Telefono

Asientos

	Numero	Precio

Total

Consultar Asientos Registrar Cliente Aceptar Cancelar

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Consultando asientos disponibles	
Nro. de Orden:	7	
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Actor Secundario:	
Tipo de Use Case:	<input type="checkbox"/> Concreto	<input checked="" type="checkbox"/> Abstracto
Objetivo:	<i>Consultar la disponibilidad de asientos para las condiciones establecidas</i>	
Precondiciones:		
Post-Condiciones:	<i>Se ha consultado la disponibilidad de asientos del colectivo</i>	
Curso Normal	Alternativas	
1. Comienza cuando se necesita consultar la disponibilidad de asientos ingresando a la pantalla “Asientos Disponibles”. 2. Se selecciona de una lista el origen/destino. 3. Se selecciona de otra lista adjunta el día y horario para el cual se desea realizar la reserva. 4. Se selecciona la opción Consultar, llamando a la pantalla “Asientos”. 5. Se muestra la disposición de los asientos en el colectivo correspondiente a ese día y horario. 6. Se selecciona el o los asientos requeridos. 7. Se retorna a la pantalla “Asientos Disponibles” y se selecciona la opción Aceptar, inhabilitando así la disponibilidad del o los asientos, a la vez que se muestra un mensaje “El o los asientos asignados son ...” 8. Seleccionamos la opción Aceptar de la pantalla principal Termina U-C	6.1. De no haber asientos disponibles, se selecciona la opción Aceptar, con lo cual se muestra un mensaje de “Colectivo Lleno” Termina U-C 6.1.1. De no estar de acuerdo con la selección del o los asientos, se elige la opción Cancelar. Termina U-C 8.1. Si se desea no asignar los asientos consultados, se selecciona la opción Cancelar. Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:	6. Registrando reserva de asientos 10. Registrando venta de pasaje en agencia	
Use Case donde se extiende:		
Use Case de Generalización:		

Asientos Disponibles

Localidad

	Origen	Destino
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		

Horario

	Dia	Hora Partida	Hora Llegada
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			

Asientos

Datos Horario

Unidad

Dia

Horario

Datos Asientos

☐ 1 ☐ 5 ☐ 9 ☐ 13 ☐ 17 ☐ 21
☐ 2 ☐ 6 ☐ 10 ☐ 14 ☐ 18 ☐ 22
☐ 3 ☐ 7 ☐ 11 ☐ 15 ☐ 19 ☐ 23
☐ 4 ☐ 8 ☐ 12 ☐ 16 ☐ 20 ☐ 24

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case Actualizando datos Cliente	Nro. de Orden: 8	
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Actor Secundario:	
Tipo de Use Case:	<input type="checkbox"/> Concreto	<input checked="" type="checkbox"/> Abstracto
Objetivo: Registrar y actualizar los datos de los clientes		
Precondiciones:		
Post-Condiciones: Se han registrado y/o actualizado los datos del cliente		
Curso Normal	Alternativas	
<p>1. Comienza cuando es llamado por otro use case y se ingresa a la pantalla “ Actualización de Datos del Cliente”.</p> <p>2. Se ingresa tipo y número de documento del cliente.</p> <p>3. De no existir el cliente, se muestra un mensaje “El cliente no existe”, ante lo cual se permite el ingreso de los datos personales.</p> <p>4. Si se desea, se ingresa: apellido, nombre, teléfono dirección, tipo de documento seleccionando de una lista desplegable y número de documento.</p> <p>5. Se selecciona Aceptar, ante lo cual aparece un mensaje “Se ha registrado un nuevo cliente” Termina U-C</p>	<p>3. Se muestran los restantes datos del cliente.</p> <p>3.1. Si se desea actualizar algún dato se lo modifica y luego se selecciona la opción Actualizar con lo cual los datos modificados quedan registrados. Termina U-C</p> <p>4.1. Si no se desea registrar un nuevo cliente, se selecciona la opción Cancelar. Termina U-C</p> <p>5.1. Si no se desea registrar un nuevo cliente, se selecciona la opción Cancelar. Termina U-C</p>	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye: 6. Registrando reserva de asiento 14. Registrando encomienda		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Cancelando reserva	Nro. de Orden: 9
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Empleado de Agencia	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Registrar la cancelación de la reserva efectuada por el cliente</i>		
Precondiciones:.		
Post-Condiciones: <i>Se ha eliminado la reserva realizada por el cliente</i>		
Curso Normal	Alternativas	
<p>1. Comienza cuando el empleado de Agencia necesita registrar que se ha cancelado la reserva ingresando a la pantalla “Cancelación de reserva”.</p> <p>2. Se ingresa el número de reserva actualizándose los datos de la reserva: origen, destino, lugar de ascenso, número de asiento y precio como así también los datos del cliente que realizó la reserva: apellido, nombre y teléfono.</p> <p>3. Se selecciona la opción Cancelar exhibiéndose un mensaje “Desea eliminar la presente reserva?”</p> <p>4. Se selecciona la opción “Sí” dándole de baja a la reserva y liberándose el o los asiento/s que poseía la reserva.</p> <p style="text-align: center;">Termina U-C</p>	<p>4.1. Si se selecciona la opción “No”, no se elimina la reserva. Termina U-C</p> <p>4.2. Si no se desea continuar con la cancelación de reserva, se selecciona la opción Cancelar. Termina U-C</p>	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Actualizacion de Datos de Cliente

Datos Cliente

Tipo Documento Numero

Apellido

Nombre

Direccion

Telefono

Nuevo Cliente Modificar Eliminar Aceptar Cancelar

Cancelacion de Reserva

Datos Viaje

Nro Reserva

Fecha

Origen

Destino

Lugar Ascenso

Asientos

	Numero	Precio

Total

Datos Cliente

Apellido

Nombre

Telefono

Aceptar Cancelar

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Registrando venta de pasaje en agencia	Nro. de Orden: 10
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Empleado de Agencia	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Registrar la venta de pasajes efectuada en la agencia correspondiente</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha registrado la venta de pasajes en la agencia y se ha emitido el boleto</i>		
Curso Normal	Alternativas	
<p>1. Comienza cuando el empleado de Agencia necesita registrar la venta de una o más pasajes ingresando a la pantalla “Venta de pasaje en agencia”.</p> <p>2. Si el cliente no tiene reserva se selecciona la opción Consultar asientos, se llama al use case “Consultando asientos disponibles”.</p> <p>3. Si hay asientos disponibles, se actualiza el origen, destino, fecha y hora de viaje.</p> <p>4. Se asigna el precio y el número de boleto, número de viaje y unidad (en una lista).</p> <p>5. Se muestra el importe total.</p> <p>6. Se selecciona la opción Aceptar.</p> <p>7. Se emite el pasaje correspondiente, registrándose el pago.</p> <p>Termina U-C</p>	<p>2.1. Si el cliente tiene reserva se ingresa el tipo y número de documento del cliente, fecha y hora del viaje.</p> <p>2.2. Se selecciona la opción Reserva, la cual actualiza los campos nombre y apellido del cliente, origen y destino del viaje, número(s) de asiento(s), número(s) de pasaje(s), precio del pasaje, número de viaje y unidad (El número de asiento y número de pasaje se muestra en una lista).</p> <p>2.3. Se actualiza el campo importe total</p> <p>2.4. Se selecciona la opción Aceptar.</p> <p>2.5. Se emite el pasaje correspondiente</p> <p>Termina U-C</p> <p>3.1. Si no hay asientos disponibles, el empleado puede efectuar alguna reserva. Se llama al use case “Registrando reserva de asiento”.</p> <p>Termina U-C</p> <p>6.1. Si no se desea concretar la venta, se selecciona la opción Cancelar</p> <p>Termina U-C</p>	
Asociaciones de Extensión:	6. Registrando reserva de asiento	
Asociaciones de Inclusión:	7. Consultando asientos disponibles	
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Registrando venta de pasaje en recorrido	Nro. de Orden: 11
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Empleado de Agencia	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Registrar venta de pasaje en recorrido</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha registrado la venta del pasaje en recorrido</i>		
Curso Normal	Alternativas	
1. Comienza cuando el empleado de Agencia necesita registrar la venta de un pasaje efectuado durante el recorrido por el chofer ingresando a la pantalla "Venta de pasajes en recorrido". 2. Se ingresa el origen y destino y la fecha. 3. Se actualizan los datos del chofer: nombre, monto y número de unidad. 4. Se ingresa el o los números de asientos asignados, la hora y el importe cobrado. 5. Se selecciona la opción Aceptar. Termina U-C	5.1. Si no se desea registrar la venta del pasaje se selecciona la opción Cancelar . Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Venta de Pasaje en Agencia

Datos Cliente

Apellido Tipo Documento

Nombre Numero

Datos Viaje

Fecha Origen

Hora Destino

Unidad

Datos Pasaje

	Nro Asiento	Nro Pasaje	Precio

Total

Reserva Consultar Asientos Imprimir Aceptar Cancelar

Venta de Pasajes en Recorrido

Datos Chofer

Apellido

Nombre

Datos Viaje

Fecha Origen

Hora Destino

Unidad

Datos Pasaje

	Nro Asiento	Precio

Total

Registrar Aceptar Cancelar

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Verificando venta del recorrido	
Nro. de Orden:	12	
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Empleado de Agencia	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo:	<i>Verificar el monto de las ventas realizadas durante el recorrido del colectivo</i>	
Precondiciones:		
Post-Condiciones:	<i>Se ha verificado el monto de las ventas de pasajes en el recorrido</i>	
Curso Normal	Alternativas	
1. Comienza cuando el empleado de Agencia verifica la rendición de cuenta del chofer con la rendición en el sistema, ingresando a la pantalla "Control Venta en Recorrido". 2. Se ingresa el origen, destino, fecha y hora. 3. Se actualizan los campos: chofer, número de unidad, número de viaje y una grilla con los pasajes vendidos, el importe correspondiente a cada pasaje y el importe total. 4. Se ingresa el importe real entregado por el chofer. 5. Se selecciona de una lista desplegable el nombre del empleado de agencia que realiza el control. 6. Se selecciona la opción Aceptar. Termina U-C	5.1. Si se desea realizar algún comentario u observación, se completa el campo "Observaciones". 6.1. Si no se desea cargar la verificación al sistema, se selecciona la opción Cancelar. Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Emitiendo planilla de ventas y reservas	Nro. de Orden: 13
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Empleado de Agencia	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Generar una planilla con los asientos ocupados por ventas y reservas, anexando un listado de precios para cada viaje</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha emitido la planilla de ventas y reservas con los datos del viaje y un listado de precios</i>		
Curso Normal	Alternativas	
1. Comienza cuando el empleado de Agencia necesita emitir la planilla de ventas y reserva a cada chofer, ingresando a la pantalla "Emisión planilla de ventas y reservas". 2. Se ingresa : origen y destino del viaje, hora y la fecha. 3. Se actualizan los campos: número de legajo del chofer, nombre y apellido, número de unidad y número de viaje. 4. Se actualizan los datos de los pasajeros: número de asiento, estado y destino. 5. Se selecciona la opción Imprimir. 6. Se anexa en un extremo de la hoja, el listado de precios de los pasajes. Termina U-C	4.1. Para los asientos reservados se muestra lugar donde sube, número(s) de asiento(s), estado, precio unitario y total, y los datos del cliente. 5.1. Si no se desea emitir esta planilla, se selecciona la opción Cancelar . Termina U-C	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Verificando Venta en Recorrido

Datos Viaje

Fecha Origen

Nro Unidad Destino

Chofer

Pasaje

	Pasajes	Precio

Empleado

Total

Importe Real

Aceptar Cancelar

Emision Planilla de Ventas Reservas

Datos Viaje

Fecha Origen

Hora Destino

Datos Chofer

Nro Legajo Nro Unidad

Apellido Nro Viaje

Nombre

Ventas - Reservas

	Nro Asiento	Estado	Destino	Lugar Donde Sube	Precio	Cliente

Imprimir Cancelar

Nivel del Use Case	<input type="checkbox"/> Negocio	<input checked="" type="checkbox"/> Sistema de Información
Nombre del Use Case	Emitiendo informe de viajes y rendiciones	
Nro. de Orden:	24	
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media <input type="checkbox"/> Baja
Actor Principal	Encargado de Tesorería	Actor Secundario:
Tipo de Use Case:	<input checked="" type="checkbox"/> Concreto	<input type="checkbox"/> Abstracto
Objetivo: <i>Emitir informes estadísticos de viajes y rendiciones.</i>		
Precondiciones:		
Post-Condiciones: <i>Se ha emitido el informe estadístico de viajes y rendiciones.</i>		
Curso Normal	Alternativas	
<p>1. Comienza cuando el encargado de Tesorería necesita emitir un informe con los viajes y rendiciones realizadas, ingresando en la pantalla “Informes de viajes y rendiciones”.</p> <p>2. Se ingresan las fechas del periodo del cual se desea emitir el informe.</p> <p>3. Se abre una nueva ventana, la cual contiene los siguientes datos: n° de viaje, n° de unidad, nombre del chofer, origen, destino, horario, cantidad de pasajeros correspondientes a cada día del periodo establecido y el monto en concepto de pasajes y encomiendas.</p> <p>4. Si no se desea adjuntar este informe se selecciona la opción Informar. Se retorna a la ventana “Informes de Viajes y Rendiciones”.</p> <p>5. Se actualizan los campos en la ventana principal, fecha, N° de viaje, origen y destino, correspondientes a la mayor cantidad de pasajeros en un viaje.</p> <p>6. Con respecto a la mayor cantidad de encomiendas se actualizan los campos fecha, origen y destino.</p> <p>7. Se selecciona la opción Imprimir emitiéndose el informe con los datos que posee la ventana principal.</p> <p>8. Se selecciona la opción Aceptar.</p> <p style="text-align: center;">Termina U-C</p>	<p>4.1. Si se desea adjuntar el informe se selecciona la opción Imprimir.</p> <p>8.1. Si se desea cancelar la emisión del informe se selecciona la opción Cancelar.</p> <p style="text-align: center;">Termina U-C</p>	
Asociaciones de Extensión:		
Asociaciones de Inclusión:		
Use Case donde se incluye:		
Use Case donde se extiende:		
Use Case de Generalización:		

Informe Viajes - Rendiciones

Datos Chofer

Fecha Ingreso Fecha Egreso

Mayor Cantidad de Pasaje

Fecha
 Nro Viaje
 Origen
 Destino

Mayor Cantidad de Encomienda

Fecha
 Origen
 Destino

Imprimir Aceptar Cancelar

[illegible]

► Unidad V

“Introducción básica a los conceptos de Objetos”

Objetivos

- Apropiarse de los conceptos relacionados a un nuevo paradigma “Objetos”
- Relacionar los conceptos de objetos, herencia, polimorfismo, encapsulamiento con los flujos de trabajo y fases de la metodología PDUS.

Contenidos de la Unidad

Introducción básica a Objetos

 ¿que es un objeto?

 ¿que es un tipo objeto?

Métodos

Encapsulado

Mensajes

Blob

¿Que es una clase?

Herencia

 Herencia de clase

Polimorfismo

Análisis de la estructura de objetos

 Asociaciones de objetos

 Jerarquías de generalización

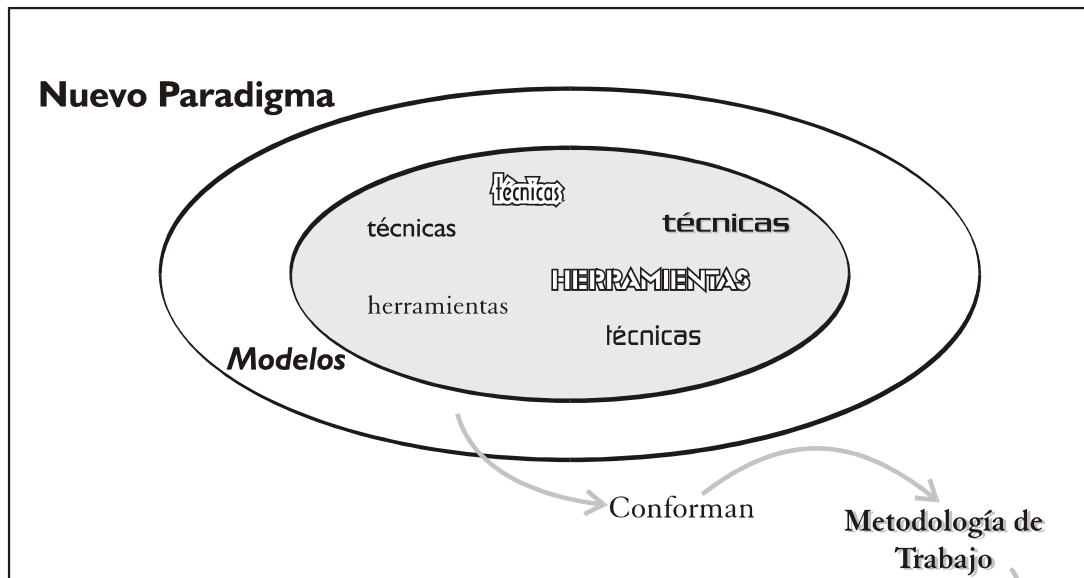
Bases de datos orientadas a objetos

Independencia de datos versus encapsulado

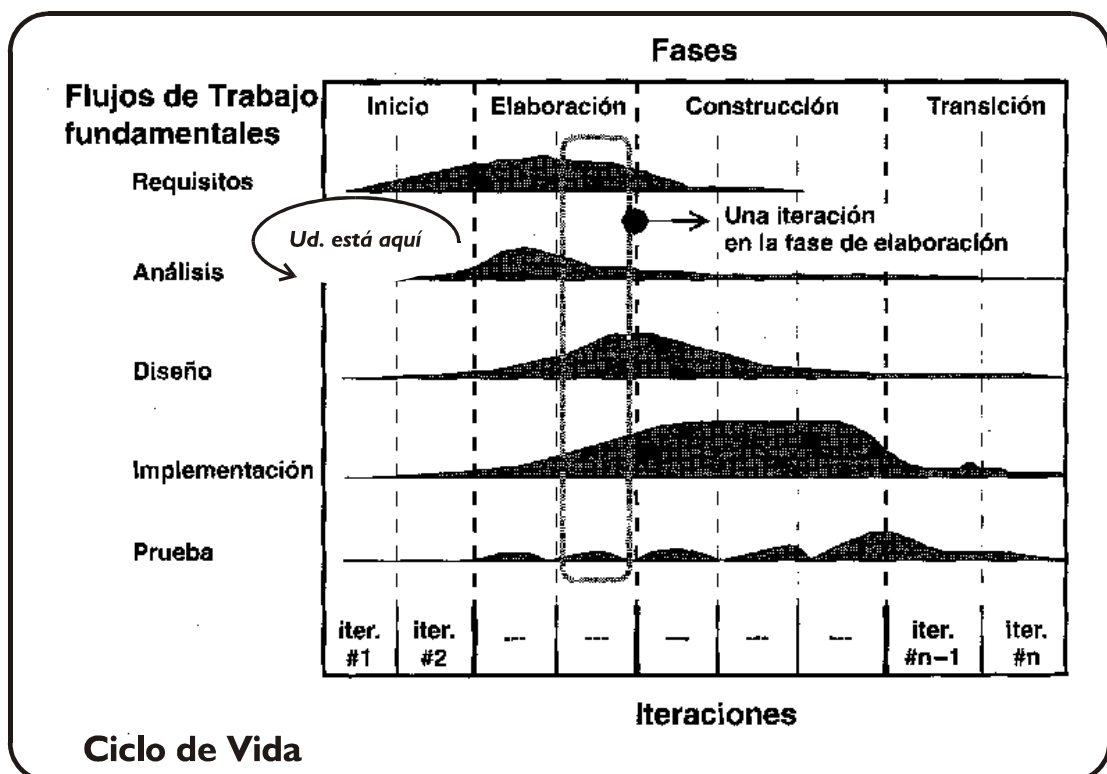
Rendimiento

Diferencias entre las bases de datos por relación y las bases de datos orientadas a objetos

Mapa Conceptual



Proceso Unificado de Desarrollo de Software



Introducción Básica

Algunas de las ideas fundamentales que subyacen en la tecnología orientada a objetos son las siguientes:

- ▶ objetos y clases (una clase es la implantación de un tipo de objeto)
- ▶ métodos
- ▶ solicitudes
- ▶ herencia
- ▶ encapsulado

Aunque estos conceptos (descritos en esta unidad) son la base del software orientado a objetos, son similares a los que forman la base de todos los seres vivientes.

¿Qué es un objeto?

Las personas nos formamos conceptos desde temprana edad. Cada concepto es una idea particular o una comprensión de nuestro mundo. Los conceptos adquiridos nos permiten sentir y razonar acerca de las cosas en el mundo. A estas cosas a las que se aplican nuestros conceptos se llaman *objetos*. Un objeto podría ser real o abstracto, como los ejemplos siguientes:

- ▶ Una factura
- ▶ Una organización
- ▶ Una figura en un programa de dibujo
- ▶ Una pantalla con la que interactúa un usuario
- ▶ Un campo o nodo de la pantalla de una herramienta CASE
- ▶ Un mecanismo en un dispositivo de robótica
- ▶ Todo un plano de ingeniería
- ▶ Un componente de un plano de ingeniería
- ▶ Un texto y fotografías utilizados en la plana de un periódico
- ▶ Un avión
- ▶ El vuelo de un avión
- ▶ Una reservación aérea
- ▶ Un ícono en la pantalla al que un usuario puede apuntar y "abrir"
- ▶ Un proceso para llenar un pedido
- ▶ El proceso para escribir esta línea

En el análisis y diseño orientados a objetos (OO), nos interesa el comportamiento del objeto. Si construimos software, los módulos de software OO se basan en los tipos de objetos. El software que implanta el objeto contiene estructuras de datos y operaciones *que* expresan dicho comportamiento. Las operaciones se codifican como *métodos*. La representación en software OO del objeto es entonces una colección de *tipos de datos* y *métodos*. En software OO:



Un *objeto* es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos.

¿Qué es un tipo objeto?

Los conceptos que conocemos se aplican a tipos determinados de objetos. Por ejemplo, “empleado” se aplica a los objetos que son personas empleadas por alguna organización. Algunas instancias de *empleado* podrían ser Juan Pérez, María Sánchez, etcétera. En el análisis orientado a objetos, estos conceptos se llaman tipos de objetos; las instancias se llaman objetos.



Un tipo de objeto es una categoría de objeto
Un objeto es una instancia de un tipo de objeto

El mundo de las bases de datos define *tipos de entes*, como cliente, empleado o parte. Existen muchas *instancias* de cada tipo de ente. **Por ejemplo**

Las *instancias* de empleado son Juan Pérez, María Sánchez, etcétera. Del mismo modo, el mundo orientado a objetos define *tipos de objetos* e *instancias* de tipos de objetos. Por ejemplo, un tipo de objeto podría ser factura y un objeto podría ser Factura #51783.

En general, almacenamos un registro por cada ente. *Objeto* se refiere a los datos y los métodos mediante los cuales se controla a los propios datos. En el mundo *OO*, la estructura de datos y los métodos de cada tipo de objeto se manejan juntos. No se puede tener acceso o control de la estructura de datos excepto mediante los métodos que forman parte del tipo de objeto.

Métodos



Los *métodos* especifican la forma en que se controlan los datos de un objeto.

Los métodos en un tipo de objeto *sólo* hacen referencia a las estructuras de datos de ese tipo de objeto. No deben tener acceso directo a las estructuras de datos de otros objetos. Para utilizar la estructura de datos de otro objeto, deben enviar un mensaje a éste. El tipo de objeto empaca juntos los tipos de datos y los métodos.

Un objeto es entonces una *cosa*, cuyas propiedades están representadas por tipos de datos y su comportamiento por métodos.

► **Por ejemplo**

Un método asociado con el tipo de objeto factura podría ser aquél que calcule el total de una factura. Otro podría transmitir la factura al cliente. Otro podría verificar de manera periódica si la factura ha sido pagada y, en caso contrario, añadir cierta tasa de interés.

Encapsulado



El empaque conjunto de datos y métodos se llama *encapsulado*. El objeto esconde sus datos de los demás objetos y permite el acceso a los datos mediante sus propios métodos. Esto recibe el nombre de *ocultamiento de la información*.

El encapsulado evita la corrupción de los datos de un objeto. Si todos los programas pudieran tener acceso a los datos en cualquier forma que quisieran los usuarios, los datos se podrían corromper o utilizar de mala manera. El encapsulado protege los datos del uso arbitrario y no pretendido.

El encapsulado oculta los detalles de su implantación interna a los usuarios de un objeto. Los usuarios se dan cuenta de las operaciones que puede solicitar del objeto, pero desconocen los detalles de cómo se lleva a cabo la operación. Todos los detalles específicos de los datos del objetos y la codificación de sus operaciones están fuera del alcance del usuario.

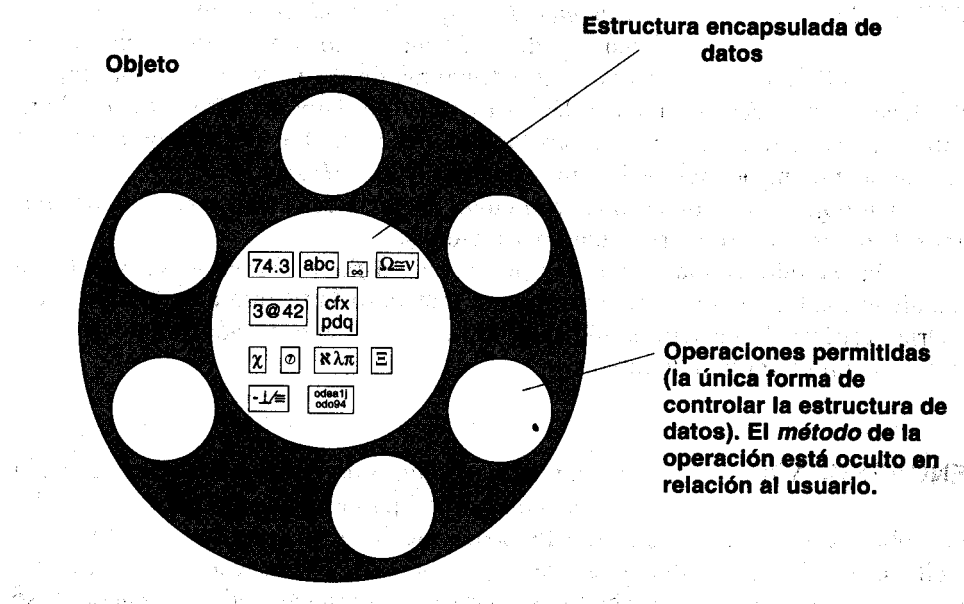


Encapsulado es el resultado (o acto) de ocultar los detalles de implantación de un objeto respecto de su usuario

El encapsulado es importante porque separa el comportamiento del objeto de su implantación. Esto permite la modificación de la implantación de un objeto sin que se tengan que modificar las aplicaciones que lo utilizan.

Es más fácil modificar los programas que utilizan el encapsulado porque se modifica al mismo tiempo un tipo de objeto. Si esto ocurre, sólo se afectan los métodos y las estructuras de datos asociados con dicho tipo de objeto; lo usual es que sólo se vean afectados *algunos* de los métodos y las estructuras de datos. El comportamiento del tipo objeto se puede modificar y probar de manera independiente a los demás tipos de objetos.

La figura ilustra un objeto. La estructura de datos en el centro sólo puede ser utilizada por los métodos del anillo exterior.



Cada objeto encapsula una estructura de datos y métodos. Una *estructura de datos* está en la parte central del objeto. El objeto es controlado por *métodos* que implantan las operaciones permitidas. La estructura de datos *sólo* puede ser utilizada por dichos métodos. A esta restricción de acceso se le llama *encapsulado*. El encapsulado evita la corrupción de los datos.

► Por ejemplo

Una Video casetera es un ejemplo de objeto. Tiene tipos específicos de comportamiento. Una Sony AH-8500 es un tipo de objeto y una Video casetera individual sería una instancia de dicho objeto. Todas las Video casetera del mismo tipo tienen los mismos métodos. Las Video casetera contienen muchos componentes complejos, la mayoría de los cuales también contienen otros componentes, pero usted no necesita conocerlos.

No se puede tener acceso a la electrónica de sus datos. (Tiene un letrero: PRECAUCION, NO ABRIR, RIESGO DE DESCARGA ELECTRICA.) Usted sólo puede utilizar los métodos ya determinados. El encapsulado evita la interferencia con los aspectos internos y oculta también la complejidad de los componentes. Usted sólo se preocupa del comportamiento de la Video casetera de la manera descrita en el manual.

Este tipo de objeto tiene muchos métodos, como tocar, grabar, cargar o descargar un casete, activar el temporizador (timer), realizar un doblaje de audio, así como las funciones del contador de cinta. Los datos del objeto no se pueden utilizar excepto por estos métodos. Los mensajes telefónicos no se pueden grabar en la Video casetera o utilizar su temporizador para hacer funcionar la cafetera.

Mensajes

Para que un objeto haga algo, le enviamos una *solicitud*. Esta hace que se produzca una operación. La operación ejecuta el método apropiado y, de manera opcional, produce una respuesta. El mensaje que constituye la solicitud contiene el nombre del objeto, el nombre de una operación y, a veces, un grupo de parámetros.

La programación orientada a objetos es una forma de diseño modular en la que, con frecuencia, el mundo se piensa en términos de objetos, operaciones, métodos y mensajes que se transfieren entre tales objetos.



Un mensaje es una *solicitud* para que se lleve a cabo la operación indicada y se produzca el resultado. En consecuencia, las implantaciones OO se refieren a los mensajes como solicitudes.



Una *solicitud* invoca una operación específica, con uno o más objetos como parámetros

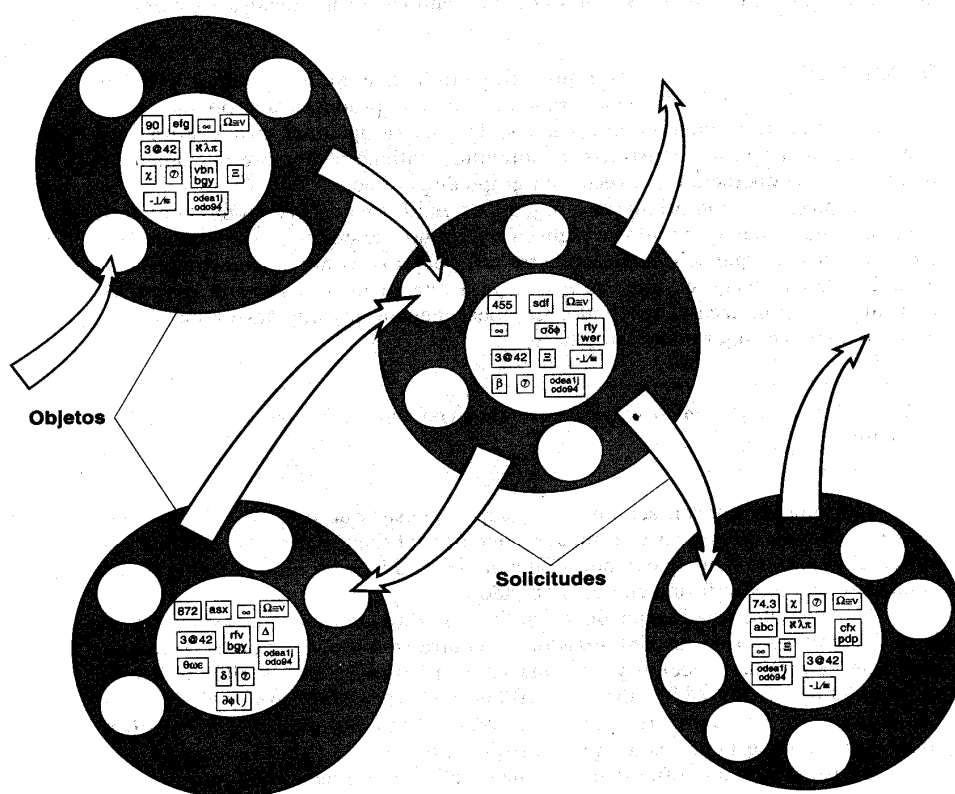
Los objetos pueden ser muy complejos, puesto que pueden contener muchos sub objetos, éstos a su vez pueden contener sub objetos. etc. La persona que utilice el objeto no tiene que conocer su complejidad interna, sino la forma de comunicarse con él y la forma en que responde.

► *Por ejemplo*

Usted se puede comunicar con la vídeo casetera al enviarle *solicitudes* por medio de un sintonizador de control remoto. Responde el aparato mediante determinada acción y presenta las respuestas en la pantalla. Todos los objetos del tipo Sony AH-8500 se controlan mediante el mismo tipo de *interfaz*. Las solicitudes del control remoto no se comunicarán con una JVC HR-S6600 VCR, puesto que ésta necesita un interfaz distinto.

► *Otro ejemplo*

El usuario de una oficina puede “hacer clic” en un icono que representa al objeto “directorio”. El objeto responde al desplegar en la pantalla una lista de nombres. El usuario puede utilizar distintas operaciones para la interacción con esta lista (pasar a una parte de la lista y apuntar hacia el nombre de cierta persona).



Representación de mensajes entre objetos

BLOB

Las computadoras actuales almacenan grandes cadenas de bits que representan imágenes, diagramas, enunciados o tal vez música o video.



Cada uno de esos objetos es conocido como BLOB (siglas en inglés de Binary Large Object, o sea, objeto binario de gran tamaño). Los BLOBs tienen métodos que permiten mostrarlos o utilizarlos.

Por ejemplo, se utilizan técnicas de compactación de modo que las imágenes se puedan almacenar en un número más pequeño de bits.

Los BLOBs disponen de métodos que permiten mostrarlos en pantallas distintas. Pueden tener métodos de seguridad, como el cifrado o descifrado. En el procesamiento de imágenes, éstas se pueden analizar de manera rápida con baja resolución y lentamente con resolución media, y presentarse después con alta resolución.

La creciente aceptación del procesamiento de imágenes aumentará la importancia de las bases de datos orientadas a objetos que puedan administrar los BLOBs de manera eficiente. Enviaremos solicitudes a los BLOBs para indicarles que se exhiban a sí mismos, se cifren a sí mismos, se ligen para su edición, etc. En una base de datos 00,

una *factura* se podría mantener en una forma gráfica hecha a mano junto con las instrucciones orales del cliente, así como un registro alfanumérico. La imagen gráfica es un objeto; las instrucciones orales son otro objeto; el registro alfanumérico es otra. Juntos forman un objeto único factura.

¿Qué es una clase?



El término *clase* se refiere a la implantación en software de un tipo de objeto.

El *tipo de objeto* es una noción de concepto. Especifica una familia de objetos sin estipular *la forma* en que se implanten. Los tipos de objetos se especifican durante el análisis OO. Sin embargo, al implantar los tipos de objetos, se utilizan otros términos.

En el lenguaje de programación Modular, los distintos tipos de objetos se implantan como *módulos*, en tanto que en Ada se utiliza la palabra *paquete*. En los lenguajes orientados a objetos, los tipos de objetos se implantan como *clases*.



Una *clase* es una implantación de un tipo de objeto. Especifica una estructura de datos y los métodos operativos permisibles que se aplican a cada uno de sus objetos.

La implantación de clases especifica la estructura de datos para cada uno de sus objetos.
Por ejemplo:

Una clase empleado incluiría datos de seguro social, puesto, salario, extensión telefónica, etc. Además, cada clase define un conjunto de operaciones permisibles que permiten el acceso y modificación de los datos del objeto. Una clase empleado podría incluir operaciones como contratar, promover y cambiar número de extensión para una clase. Los detalles del *método* operativo se especifican en la clase.

La figura del objeto que se mostró, dibuja un objeto como si el método fuera parte de un objeto. En realidad, el método se almacena una sola vez y todos los objetos de una clase dada, lo comparten.

La industria utiliza el término *clase* para hacer referencia a las implantaciones de los tipos de objetos. Se refiere a las *bibliotecas de clases* como los depósitos de las clases existentes que un analista o un diseñador podría utilizar.



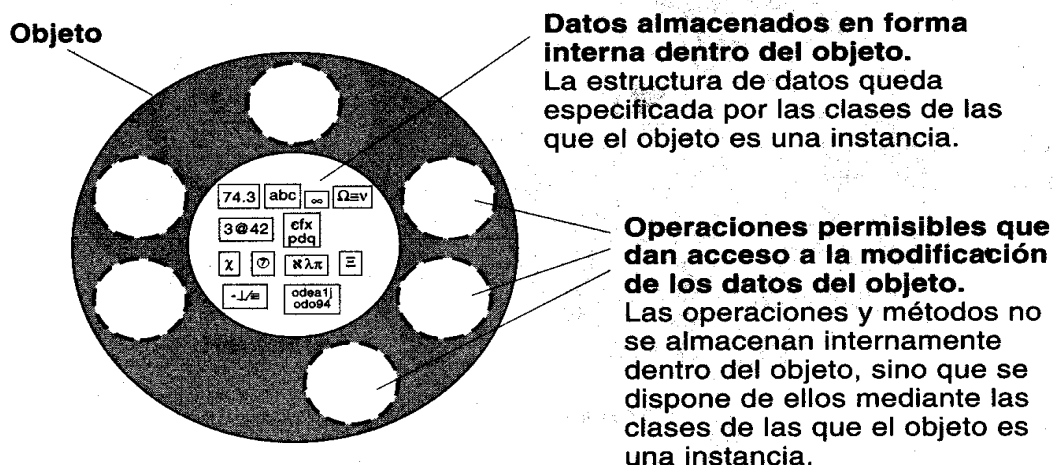
Clase es la implantación de un tipo de objeto. Especifica la estructura de datos, y los métodos operativos permitidos que se aplican a cada uno de sus objetos

La figura, mostró también, que los *datos* del objeto se almacenan dentro de él y se tiene acceso a ellos y se le modifica sólo mediante las operaciones permisibles. Esta restricción

al acceso se debe al *encapsulado*. El acceso o la actualización directa de los datos de un objeto por parte del usuario violaría el encapsulado.

Los usuarios observan el "comportamiento" del objeto en términos de las operaciones que se pueden aplicar a los objetos, así como los resultados de tales operaciones. Estas operaciones forman la *interfaz* del objeto con sus usuarios.

¿Cuál es la diferencia entre operación y método?



El encapsulado oculta la implantación de la estructura de datos y métodos de cada objeto. Los usuarios sólo conocen las operaciones permisibles del objeto, el *interfaz*.



Una operación es un proceso que se puede solicitar como unidad
Un método es la especificación de una operación.

Es decir la operación es el tipo de servicio solicitado y el método es su código de programación.

Herencia

Un tipo de objeto de alto nivel puede especializarse en tipos de objeto de bajo nivel. Un tipo de objeto puede tener subtipos. **Por ejemplo:**

El tipo de objeto persona puede tener subtipos civil y militar. Militar puede tener subtipos oficial y enrolado. Oficial puede tener subtipos teniente, capitán y mayor, y también subtipos como marino, oficial en servicio activo u oficial retirado. Existe una jerarquía de tipos, subtipos, sub subtipos, etcétera.

Una clase implanta el tipo de objeto. Una sub clase *hereda* propiedades de su clase padre; una sub-subclase hereda propiedades de las subclases; etc.



Una subclase puede *heredar* la estructura de datos y los métodos, o algunos de los métodos, de su superclase. También tiene sus métodos e incluso tipos de datos propios.

Herencia de clase

La generalización es una noción conceptual. La *herencia de clase* (que sólo se conoce como *herencia*) es una implantación de la generalización. La generalización establece que las propiedades de un tipo se *aplican* a sus subtipos.



La herencia de clase hace que la estructura de datos y operaciones sean disponibles para su reutilización por parte de sus sub clases.

La herencia de las operaciones de una superclase permite que las clases compartan el código (en lugar de volverlo a definir). La herencia de estructura de datos permite la reutilización de la estructura.

► Por ejemplo:

En la figura que encontramos a continuación, cuenta de cliente hereda las operaciones 1 y 2 de cuenta. Además de ésta, tiene sus propias operaciones 3 y 4. cuenta de cliente vencida también hereda las operaciones 1, 2, 3 y 4 de cuenta de cliente, además de tener la propia operación 6.

Herencia múltiple



En la *herencia múltiple*, una clase puede heredar estructuras de datos y operaciones de más de una superclase.

La primera figura muestra la *herencia simple*, mientras que la segunda figura muestra la *herencia múltiple*.

Completando el ejemplo

En el análisis de la estructura de objetos, el analista indicaría qué cuenta de cliente vencida tiene dos supertipos, que comparten un supertipo cuenta. En el diseño OO, esta jerarquía de generalización se implanta mediante la herencia. Cuenta de cliente vencida hereda las características de las. clases. cuenta de cliente y cuenta vencida. Por lo tanto, cuenta de cliente vencida dispone de las siguientes operaciones para reutilizar; 1, 2, 3, 4 y 6.

La *herencia de clase* implanta la jerarquía de generalización, y permite así que una clase comparta la estructura de datos y operaciones de otra clase.

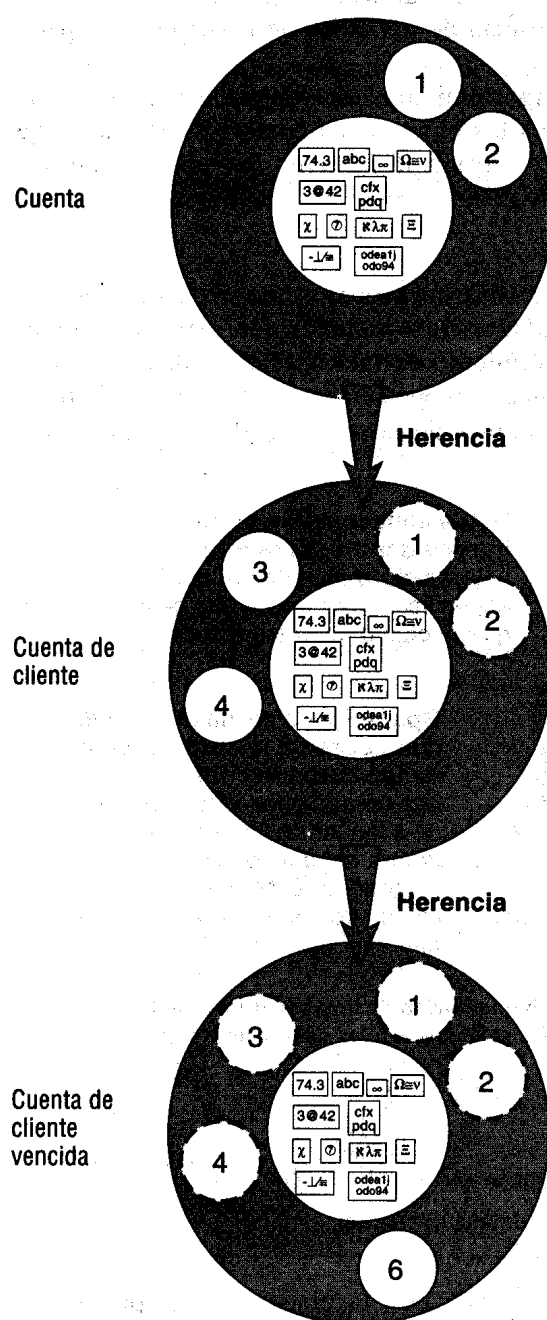


La herencia simple es aquella en la que una clase puede heredar la estructura de datos y operaciones de una superclase.

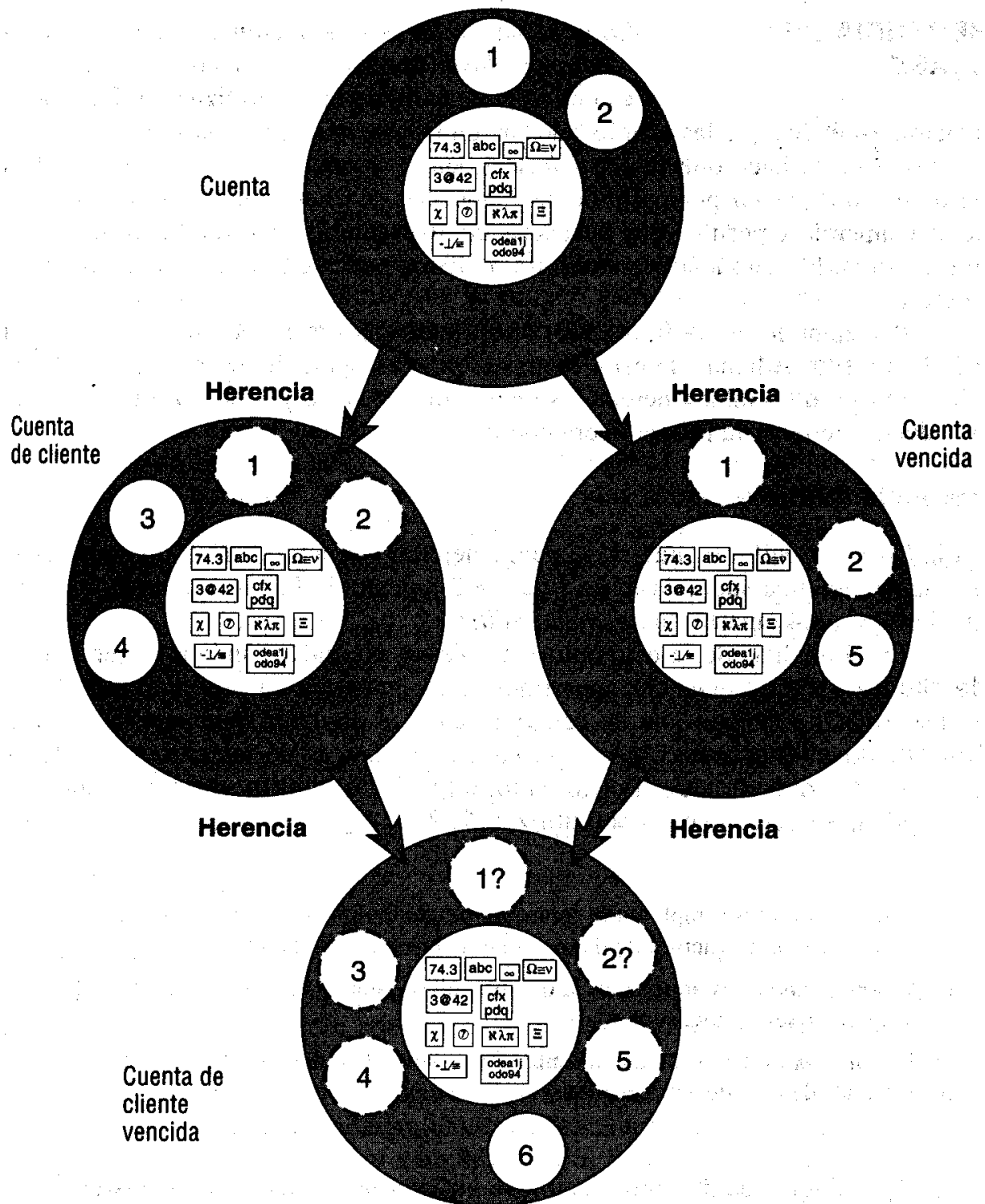


La *herencia múltiple* se da cuando una clase puede heredar la estructura de datos y operaciones de más de una superclase.

Herencia. La clase *CUENTA DE CLIENTE* hereda los métodos 1 y 2 de la clase *CUENTA*. *CUENTA DE CLIENTE* tiene sus propios métodos 3 y 4. La clase *CUENTA DE CLIENTE VENCIDA* hereda los métodos 1, 2, 3 y 4 de *CUENTA DE CLIENTE*, además de tener su propio método 6.



Las tres clases inferiores heredan los métodos de *CUENTA*. *CUENTA DE CLIENTE* vencida hereda los métodos de *CUENTA DE CLIENTE* y *CUENTA VENCIDA*. (Cuando se heredan operaciones con nombres idénticos de varias superclases, suelen surgir problemas.)



Polimorfismo

Uno de los objetivos principales de las técnicas OO es utilizar otra vez el código. Sin embargo, algunas de las operaciones requieren adaptación para resolver necesidades particulares.

Ejemplo, la clase empleado define una operación de retiro. En las implantaciones OO todas las sub clases empleado heredan esta operación en forma automática. Sin embargo, una organización puede tener distintos métodos, para retirar a un ejecutivo y a un empleado. En este caso, el método para el retiro de un ejecutivo *está por encima* del método para el retiro de los empleados general. Aun así, aunque los métodos sean distintos, llevan a cabo el mismo propósito operativo. Este fenómeno se conoce como *polimorfismo*. La palabra polimorfismo se aplica a una operación que adopta varias formas de implantación.

Percepción y Realidad

Con el software jugamos con la realidad. Podemos hacer que algo muestre cierto comportamiento cuando en realidad sea muy distinto. Podemos crear una interfaz sencilla para algo complejo y ocultar así la complejidad.

El término *transparente* se utiliza para indicar que algo parece no existir pero sí existe. Muchos de los complejos mecanismos relativos al almacenamiento o transmisión de datos quedan ocultos a los programadores para que éstos no tengan que entenderlos o incluso conocerlos. Los programadores pueden utilizar, por ejemplo, un registro *lógico*, donde algunos campos queden ocultos, al igual que las complejidades de la estructura física.

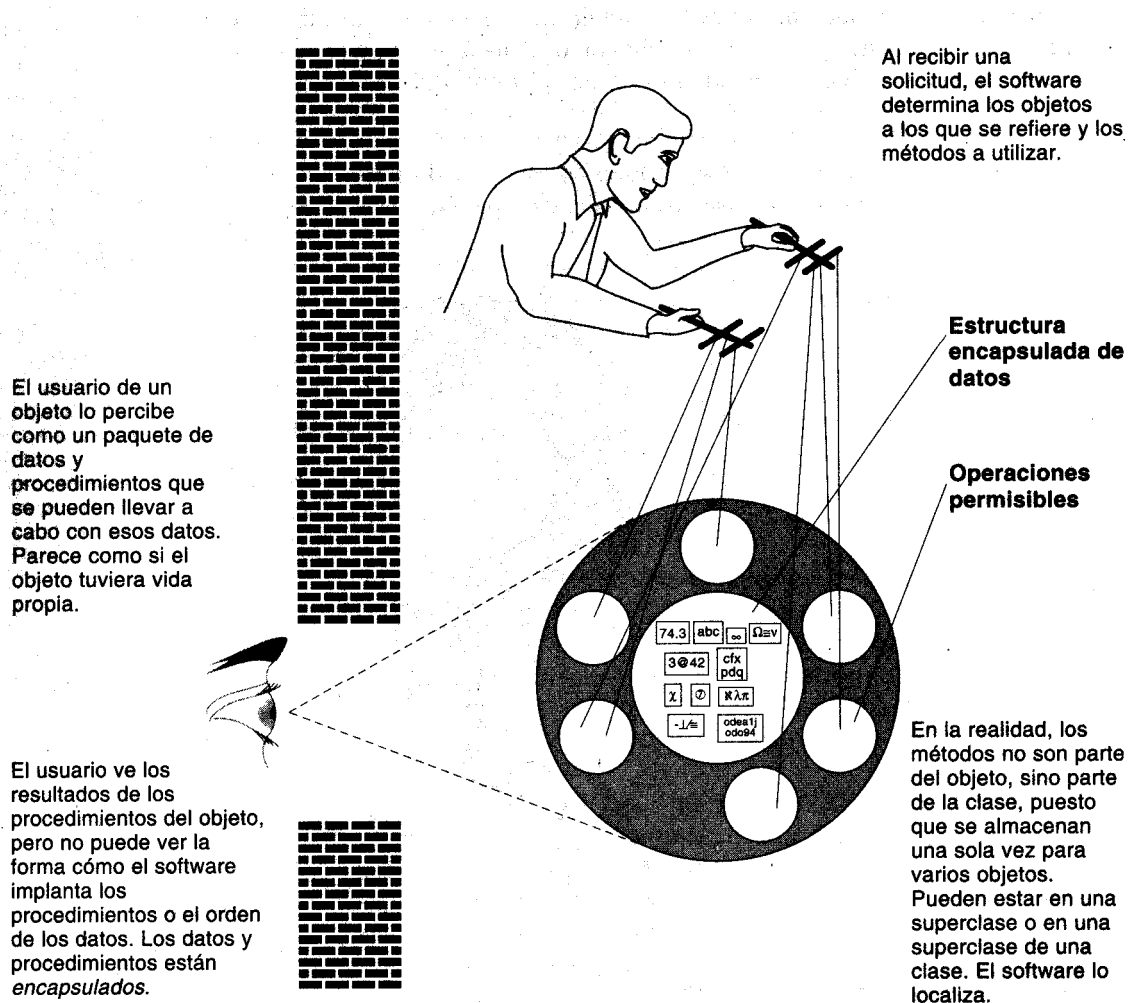
Por otra parte, en el contexto de las instalaciones para el almacenamiento de datos de computación, la palabra *virtual* se utiliza para indicar que cierto término *parece* existir para el programador o el usuario cuando, en realidad, no existe. Hablamos de almacenamiento virtual y circuitos virtuales.



“Algo virtual *parece* existir pero en realidad no existe”
“Algo transparente *parece* no existir pero sí existe.”

En las implantaciones OO se percibe un objeto como un paquete de datos y procedimientos que se pueden llevar a cabo con estos datos. Esto *encapsula* los datos y los procedimientos. La realidad es diferente. Los procedimientos no se almacenan en cada objeto, pues ello sería un desperdicio, puesto que habría que almacenar el mismo procedimiento muchas veces. En lugar de esto, el software examina cada solicitud referente a un objeto y tiene un mecanismo de selección para localizar el código a ejecutar. El *método* es parte de la *clase*, pero no parte del *objeto*. El método ni siquiera podría ser parte de la clase; pero podría ser parte de la clase de mayor nivel en la jerarquía de clases. El software puede localizarlo.

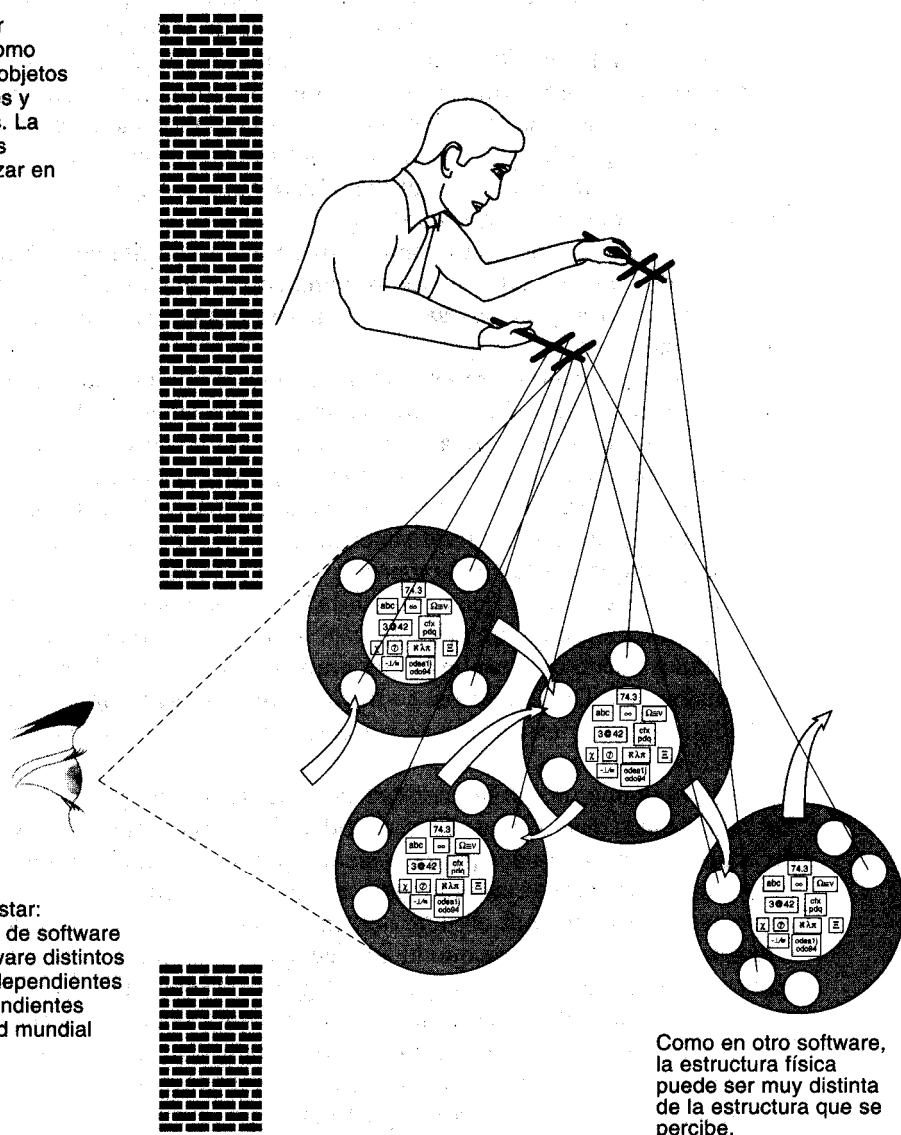
La figura muestra la diferencia entre cómo percibimos los objetos y cómo actúa el software en realidad. El uso de los mecanismos virtuales y



transparentes es la única vía para tener éxito en la construcción de las estructuras de la complejidad necesaria.

Los objetos se pueden comunicar con otros objetos: los existentes en un paquete de software o en diferentes paquetes, localizados en uno o varios procesadores, o bien distribuidos en una red mundial.

Percibimos el software orientado a objetos como algo que consiste en objetos que reciben solicitudes y devuelven respuestas. La solicitud especifica las operaciones por realizar en los objetos.



Análisis de la estructura de objetos

El análisis de la estructura de objetos (AEO) define las categorías de los objetos que percibimos y las formas en que los asociamos, presenta los modelos que se crean durante el análisis de la estructura de objetos. Estos mismos modelos del AEO son una guía para la definición de clases y sus estructuras de datos en el diseño de la estructura y Comportamiento del objeto.

El análisis de la estructura de objetos, tiene como objetivo primordial responder los siguientes cuestionamientos:

¿Qué son los tipos de objetos y cómo se asocian? La identificación de los objetos y sus asociaciones se representan mediante esquemas de objetos. Esta información guía al diseñador en la definición de clases y estructuras de datos.

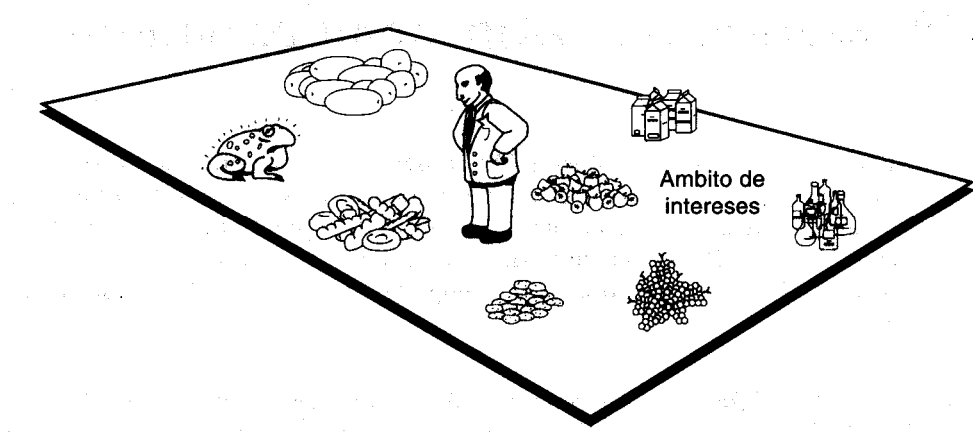
¿Cómo se organizan los tipos de objetos en supertipos y subtipos? Las jerarquías de generalización se pueden organizar en diagramas e indicar al diseñador las direcciones de herencia.

¿Cuál es la composición de los objetos complejos? Se pueden elaborar diagramas de jerarquías compuestas. La composición guía al diseñador en la definición de mecanismos que controlen adecuadamente a los objetos dentro de otros objetos.

Durante el análisis de la estructura de objetos, el equipo de análisis se preocupa más por identificar los tipos de objetos que por identificar los objetos individuales en un sistema, *los tipos de objetos son categorías de objetos*

Los tipos de objetos son importantes, puesto que crean los bloques conceptuales de construcción para el diseño de sistemas. En la programación orientada a objetos, estos bloques de construcción guían al diseñador en la definición de las clases y sus estructuras de datos. Además, los tipos de objetos son un índice, para los procesos del sistema.

Por ejemplo, actividades como contratar, retirar y despedir están ligadas íntimamente con el tipo de objeto empleado puesto que cambian su estado. En otras palabras, un objeto sólo debe ser controlado por medio de las funciones asociadas con su tipo. Así, las operaciones no se pueden definir de manera adecuada sin los tipos de objetos.

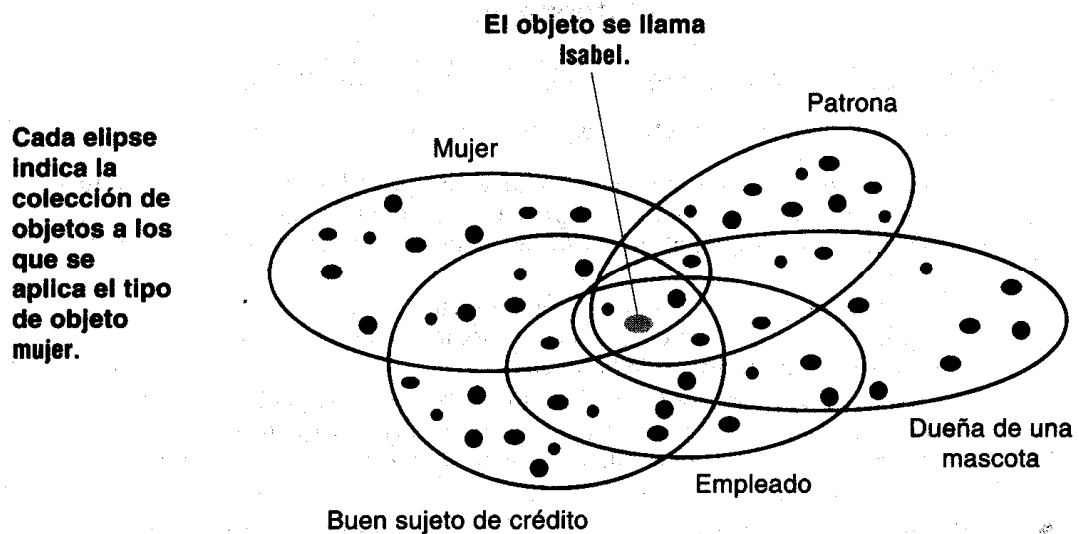


Nosotros elegimos la forma de dar categorías a nuestro mundo

Los tipos de objetos que definimos y utilizamos son variados, puesto que elegimos con base en la comprensión de nuestro mundo.

Aunque esto pudiera parecer muy arbitrario, así funciona nuestra mente, hecho un objeto se puede categorizar en más de una forma. **Por ejemplo**

Una persona puede considerar al objeto *Isabel* como mujer. Su jefe considera un empleado. El señor que le arregla el jardín la considera su patrona. La agencia local para el control de los animales la considera dueña de la mascota. La oficina de crédito informa que *Isabel* es una instancia del tipo objeto llamado de buen sujeto de crédito, etcétera.



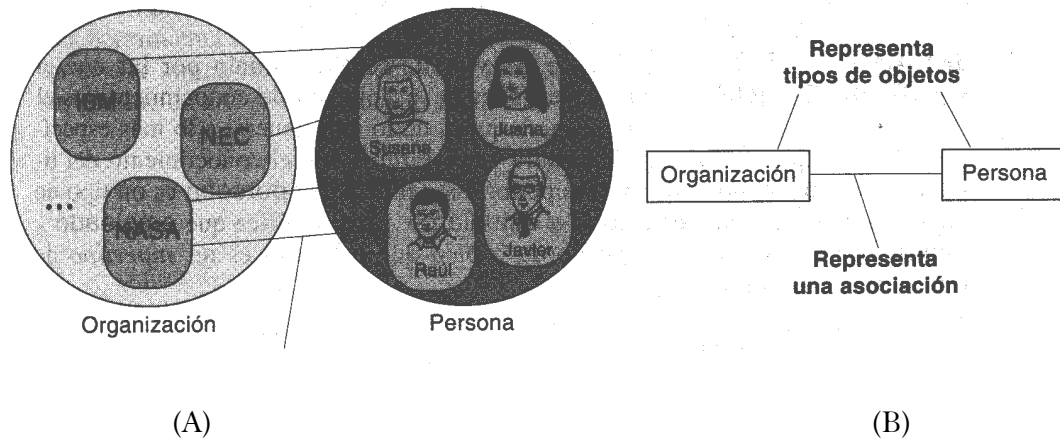
Otros ejemplos de tipos de objetos pueden ser

Concreto	Intangible	Papeles	Juicios
persona	tiempo	doctor	trabajo productivo
lápiz	calidad	paciente	salario alto
automóvil	compañía	propietario	buen ejemplo
Relación	Eventos	Visualización	Otros
matrimonio	venta	cadena	imagen
sociedad	compra	entero	señal
propiedad	falla del sistema	icono	gnomo

Asociaciones de objetos

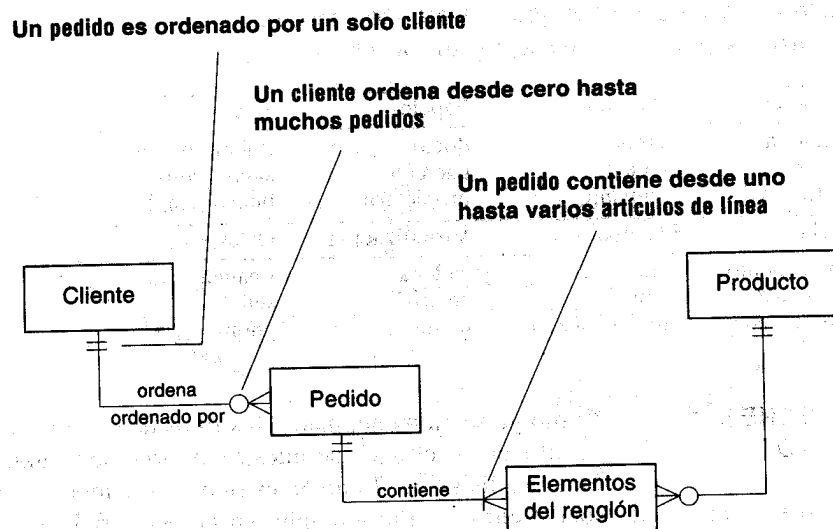
Como ya se ha mencionado, los tipos de objetos que clasifican los objetos de nuestro entorno son vitales en el análisis OO. También es importante modelar como los objetos se asocian entre sí. **Por ejemplo**

En la figura (A), los objetos organización, como IBM, NASA y NEC se asocian con los objetos *persona* Susana, Juana, Javier y Raúl. La figura (B) representa una forma de expresar la asociación entre estos dos tipos de objetos.



Asociación entre un objeto organización llamado NASA y un objeto llamado Raúl

Aunque la figura ilustra las asociaciones entre dos tipos de objetos, no se indica el *significado* de la asociación. Además del significado, tampoco se indica la cantidad de objetos con los que un objeto dado puede y debe asociarse. En el análisis: es útil nombrar de alguna forma a las asociaciones indicar la cantidad de objetos de un tipo dado que se deben asociar con los objetos de otro tipo, puesto que esto le da significado y aumenta la comprensión de la asociación



Jerarquías de generalización

Una de las vías de sentido común por las que el hombre organiza su volumen de conocimiento es el de las jerarquías, de lo más general a lo más específico.

Por ejemplo, la figura muestra una jerarquía con el conocimiento del tipo de objeto persona en la parte superior. Esto indica que persona es un tipo de objeto más general que empleado y estudiante. Esto significa que empleado, estudiante son *subtipos* de persona; o bien que persona es un *supertipo* de empleado y estudiante.



Una jerarquía de generalización de tipos de objetos que indica que persona es un supertipo de empleado y estudiante. Empleado es un supertipo de vendedor y gerente.

Todas las propiedades de un tipo de objeto también se aplican a sus subtipos. **Por ejemplo**

Sony fabrica muchas Vídeo casetera. Vídeo casetera es un tipo de objeto con determinadas características. AH-8500 es un subtipo de Vídeo casetera. Por lo tanto, AH-8500 tiene todas las propiedades de Vídeo casetera, además de las suyas propias. Además, Sony fabrica varios modelos de AH-8500; AH-8500-12 es uno de ellos. Por lo tanto, todas las propiedades de AH-8500 se aplican a AH-8500-12, aunque la AH-8500-12 también tiene algunas propiedades especiales.



La *generalización* es el resultado (o el acto) de distinguir un tipo de objeto como más general, o incluso, que es más que otro. Todo lo que se aplique a un tipo de objeto también se aplica a sus subtipos. Cada instancia de un tipo de objeto es también una instancia de sus supertipos.

Las jerarquías de generalización son importantes para el desarrollador por dos razones. La primera es que el uso de supertipos y subtipos proporciona una herramienta útil para describir el mundo del sistema de aplicación. La segunda es que indica las direcciones de herencia entre las clases en los lenguajes de programación.

Bases de datos orientadas a objetos

Una base de datos tradicional sólo almacena datos, sin procesarlos, de modo que resulten independientes de los procedimientos. Los datos son accesibles a diferentes usuarios, con diversos propósitos.

Por lo contrario, una base de datos orientada a objetos almacena objetos. Los datos se almacenan junto con los *métodos* que procesan dichos datos.

Después del uso constante de las bases tradicionales de datos, surge la necesidad de asociar ciertos procedimientos con los datos y activarlos cuando se tiene acceso a éstos. Esos procedimientos se utilizaron para controlar la integridad de los datos o su seguridad. A veces se calculaba un valor a partir de campos diferentes a los almacenados, procedimiento al que se conocía como colocación de *inteligencia* en la base de datos. Las técnicas de las bases inteligentes de datos fueron útiles para controlar la integridad en un ambiente cliente-despachador, puesto que nunca se tenía la certeza de la identidad del cliente que tenía acceso al despachador de una base de datos.



Las bases de datos orientadas a objetos toman la idea de las bases inteligentes de datos a su conclusión lógica. No se tiene acceso a dato alguno si no es a través de los métodos almacenados en la base de datos. Estos métodos están listos para entrar en acción al momento en que reciben una solicitud. Los datos de todos los objetos quedan entonces *encapsulados*. En general los datos son activos más que pasivos.

Tres enfoques de construcción de base de datos Orientados a Objetos

Las bases de datos se pueden construir mediante alguno de los tres enfoques.

- » El primero utiliza un sistema de administración convencional de una base de datos y añade un nivel para procesamientos de las solicitudes y métodos de almacenamiento 00. Este enfoque tiene un mérito: se puede utilizar el código actual altamente complejo de los sistemas de administración de bases de datos, de modo que una BDOO se implante más rápido sin partir de cero.
- » El segundo enfoque considera a la BDOO como una extensión de la tecnología de las bases de datos por relación. Se pueden añadir apuntadores a las tablas de relación para ligarlas con objetos binarios de gran tamaño (BLOB). Varios proveedores de BDR apoyan este enfoque.
- » El tercer enfoque reflexiona sobre la arquitectura de los sistemas de bases de datos y produce una nueva arquitectura optimizada, que cumple las necesidades de la tecnología 00. Los proveedores que utilizan este enfoque afirman que la tecnología de relación es un subconjunto de una capacidad más general. Las estructuras de datos de relación se podrían utilizar en los casos adecuados, pero lo usual es que para los objetos complejos sean mejores otras estructuras de datos; éstas permiten el acceso a los datos de un objeto complejo sin desplazar el mecanismo de acceso.

Los promotores de BDOO afirman que las BDOO no de relación son aproximadamente dos veces más rápidas que las bases de datos por relación para almacenar y recuperar la información compleja

Independencia de datos versus encapsulado

Uno de los objetivos principales de la tecnología tradicional de las bases de datos es la *independencia de los datos*. Las estructuras de datos deben ser independientes de los procesos que utilizan los datos.

Así, los datos se pueden utilizar en la forma que deseen los usuarios.

Por el contrario, uno de los principales objetivos de la tecnología orientada a objetos es el *encapsulado*. Esto significa que los datos *sólo* pueden ser utilizados con los métodos que forman parte de una clase.

La tecnología de las bases tradicionales de datos, está diseñada para soportar procesos sujetos a una modificación infinita. Por ello, es necesaria la independencia de los datos. La BDOO soporta clases, algunas de las cuales rara vez son modificadas. Los cambios provienen de clases ligadas entre sí de varias formas. Las estructuras de datos en la BDOO deben estar fuertemente optimizadas para soportar la clase en la que están encapsuladas. Así, los objetivos en las bases de datos por relación y las bases de datos orientadas a objetos son fundamentalmente diferentes.

Al final de la unidad, se presenta un cuadro comparativo de BD tradicional y BDOO.

Rendimiento

Las BDOO superan en mucho el rendimiento de las BDR para las aplicaciones de gran conexión entre datos. Esta es una de las principales razones por las que las aplicaciones que comparten esta característica utilizan las BDOO en la actualidad. Otras razones son:

Las BDOO utilizan diversas estructuras de almacenamiento. En una BDR, son difíciles el almacenamiento y el acceso a los datos que no se pueden expresar con facilidad en forma tabular. Por ejemplo, las aplicaciones multimedia requieren el almacenamiento de grandes flujos de datos, que representan datos de video y audio digitalizados. Por lo tanto, las BDR no pueden ofrecer una administración eficiente del almacenamiento para estas áreas de aplicación. El modelo de almacenamiento de las BDOO es ilimitado, puesto que el sistema es extendible por naturaleza. Las BDOO pueden ofrecer diferentes mecanismos de almacenamiento para distintos tipos de datos.

Evación de la redundancia

Con las BDR, los datos se normalizan para evitar la redundancia y las anomalías provocadas por la redundancia de los datos. Esto evita la redundancia en los datos pero no el código de la aplicación.

La tecnología OO utiliza la *herencia* para reducir el desarrollo redundante de los *métodos*. También crea clases diseñadas para utilizarse otra vez en muchas aplicaciones. El encapsulado y la herencia reducen la cantidad de código redundante, así como los datos redundantes, de dos maneras: mediante la herencia y la reutilización de las clases. Esto también reduce el costo del desarrollo y el mantenimiento.

Con la tecnología de las bases tradicionales de datos, el procesamiento común se reproduce a través de distintas aplicaciones, lo que provoca una menor productividad y mayores costos de mantenimiento. De manera más reciente, las BDR han introducido el concepto de reglas de activación o procedimientos que puede ejecutar la base de datos. El uso de estas capacidades para colocar el procesamiento común dentro de la base de datos es un avance. Sin embargo, sigue separado el procesamiento de aquellas entidades naturalmente asociadas con el procesamiento. Por ejemplo, uno podría escribir una regla de activación que impusiera el comportamiento donde el salario de un empleado despedido fuese \$0. Sin embargo, el proceso de despido no se puede asociar con la tabla empleado. En una BDOO, este proceso es directo, ya que la clase empleado definiría la solicitud despedido e impondría la restricción de salario \$0.

Diferencias entre las bases de datos por relación y las bases de datos orientadas a objetos

Las bases de datos orientadas a objetos y las bases de datos por relación tienen objetivos y características fundamentalmente diferentes. En ciertos ambientes de computación, predominan los objetivos de las bases clásicas de datos por relación.

En estos casos se prefieren las BDR. En otros ambientes, las BDR no cubren de manera adecuada las necesidades

Bases de datos por relación

Objetivo principal: independencia de los datos

Únicamente datos. En general, la base de datos sólo almacena precisamente datos.

Datos compartidos. Los datos pueden compartirse entre varios procesos. Los datos están diseñados para cualquier tipo de uso.

Datos pasivos. Los datos son pasivos. Se pueden activar en forma automática ciertas operaciones limitadas al utilizarlos.

Cambio constante. Los procesos que utilizan los datos cambian de manera constante.

Independencia de datos. Los datos pueden reorganizarse físicamente sin afectar su forma de uso.

Sencillez. Los usuarios perciben los datos como columnas, hileras y tablas.

Tablas independientes. Cada relación (tabla) es independiente. Los comandos JOIN relacionan los datos en las diversas tablas.

Bases de datos orientadas a objetos

Objetivo principal: encapsulado.

Datos más métodos. La base de datos almacena datos y métodos.

Encapsulado. Los datos sólo pueden utilizarse por métodos de clases. Los datos están diseñados para su uso exclusivo por métodos particulares.

Objetos activos. Los objetos son activos. Las solicitudes hacen que los objetos ejecuten sus métodos. Algunos de estos métodos pueden ser muy complejos; por ejemplo, aquellos que utilicen reglas y una máquina de inferencias.

Clases diseñadas para su reutilización. Las clases diseñadas para una alta reutilización rara vez se modifican.

Independencia de la clase. Las clases se pueden reorganizar sin afectar su forma de uso.

Complejidad. Las estructuras de datos pueden ser complejas. Los usuarios no están conscientes de la complejidad por el encapsulado.

Datos ligados entre sí. Los datos pueden estar ligados entre sí, de modo que los métodos de la clase logren un mejor rendimiento. Las tablas son sólo una de las estructuras de datos que se pueden utilizar. Los objetos binarios de gran tamaño (BLOB) se

Datos no redundantes. La normalización de los datos se efectúa para ayudar a eliminar la redundancia en los datos. (No ayuda en el caso de la redundancia en el desarrollo de la aplicación.)

SQL. El lenguaje SQL se utiliza para el manejo de las tablas.

Rendimiento. El rendimiento es una preocupación en el caso de las estructuras de datos altamente complejas.

Distinto modelo conceptual. El modelo de estructura de datos y acceso representado por tablas y comandos JOIN es distinto del modelo para análisis, diseño y programación. El diseño debe traducirse en tablas de relación y acceso al estilo SQL.

utilizan para sonido, imágenes, video y grandes flujos de bits sin estructura.

Métodos no redundantes. Con el encapsulado y la herencia se obtienen datos y métodos no redundantes. La herencia ayuda a reducir la redundancia en los métodos, mientras que la reutilización de las clases ayuda a reducir la redundancia general en el rendimiento.

Solicitudes OO. Las solicitudes provocan la ejecución de los métodos. Se pueden utilizar diversos métodos.

Optimización de clases. Los datos de un objeto se pueden ligar entre sí y almacenar juntos, de modo que se pueda tener acceso a ellos desde una posición del mecanismo de acceso. Las BDOO ofrecen un rendimiento mucho mejor que las bases de datos por relación, para ciertas aplicaciones con datos complejos.

Modelo conceptual consistente. Los modelos utilizados para análisis, diseño, programación y acceso y estructura de la base de datos son similares. Los conceptos de la aplicación se representan de manera directa mediante clases en la BDOO. Cuanto más complejas sean la aplicación y su base de datos, más tiempo y dinero se ahorrarán en el desarrollo de aplicaciones.

Modelo de examen

INSTITUCION CERVANTES

Apellido y Nombre: Matrícula:

- ▶ Lea con detenimiento y analice el siguiente enunciado, antes de comenzar a desarrollar su examen.
- ▶ Justifique todas sus posiciones.
- ▶ Este examen corresponde a la parte práctica. Si Ud. aprueba la misma, pasará a una segunda instancia oral o escrita, sobre los aspectos teóricos de la asignatura.

MUTUAL XXX

La mutual tiene por cometido, el de ofrecer una serie de servicios a sus asociados. Para ello la entidad toma contactos con distintos proveedores, que luego de una evaluación de los mismos celebran un contrato que obliga a vender sus productos a los asociados, ofreciendo las mejores condiciones de comercialización, tanto en calidad como en financiación. El cobro de lo consumido por los asociados, se efectuará a la Mutual, presentando los originales de cada orden de compra.

Para ello el mecanismo de recaudación es el siguiente:

El empleado municipal se asocia presentando su DNI y mandamiento, completa una solicitud y a partir de allí, se le cobra una cuota por pertenecer a la mutual, que se descuenta de sus haberes cada fin de mes, (utilice los servicios o no), a ello se le suma los importes de los consumos efectuados (valor de cada cuota o el total si fuese de un solo pago), vía proveedores, previa emisión de una orden de compra en la mutual.

Con lo descontado efectivamente de los salarios, la mutual intenta cubrir sus costos operativos y abonar a los proveedores.

A partir del control de los descuentos se actualiza los estados de cuenta de cada socio y se conforma en caso de ser necesario un registro de morosos.

Requerimientos del usuario

- » Ejercer un control adecuado de la emisión de ordenes de compra.
- » Conocer la información precisa de los estados de cuenta de socios y proveedores.
- » Conocer el rendimiento de la prestaciones de los proveedores
- » Liquidar correctamente los descuentos en los haberes de los socios.

Actividades a desarrollar:

1. Esboce el dimensionamiento que Ud. ofrecería (determine objetivos y alcances).
2. Desarrolle el modelo de negocios del sistema, e identifique los actores.
3. Según su criterio que diagrama de estos:
Diagrama de clases del dominio, Diagrama de Caso de uso Sistema Información selecciona, para completar el estudio de la propuesta, y ¿porqué lo elige?. fundamente su respuesta en forma sintética
4. Diseñe un prototipo de pantalla, especificando de donde obtiene la información.

Índice

Carta al alumno.....	1
Fundamentación	1
Objetivos Generales.....	2
Mapa Conceptual.....	3
Contenidos	4
Orientaciones Metodológicas	5
Formas de evaluación.....	6
Material Soporte de Información de todas las unidades.....	6
Iconos de Referencia.....	7
Evaluación Diagnóstica	8
 UNIDAD I	
“Introducción a la Tecnología de Objetos” Proceso Unificado de Desarrollo de Software “Ciclo de vida”	9
Objetivos.....	9
Contenidos de la Unidad	9
Tecnologías orientada a objetos	10
Componentes reusables = Standard	13
El paradigma de la integración	13
Los factores que influyeron en el cambio de paradigma	14
Ingeniería de software (Un Concepto de actualidad).....	16
Qué es una metodología.....	17
El Proceso Unificado de desarrollo de Software.....	18
El Proceso Unificado, dirigido por casos de uso	20
¿Por qué casos de Uso?	21
Para capturar los requisitos	21
Para dirigir el proceso	22
Para idear la arquitectura y más... ..	22
El Proceso Unificado está centrado en la arquitectura	23
Por qué es necesaria la arquitectura.....	23
Comprensión del sistema	24
Organización del desarrollo.....	24
Fomento de la reutilización	24
Evolución del sistema.....	25
El Proceso Unificado es iterativo e incremental.....	25
Relación entre los tres aspectos del Proceso de desarrollo Unificado	26
Ciclo de Vida del Proceso Unificado	27
Fases dentro de un ciclo	27
Iteraciones.....	30
Ciclos de desarrollo.....	30
Flujos de trabajo del proceso.....	30
Modelos.....	31
UML.....	31
Resumen de la unidad... ..	32
PDUS	34
Flujos de trabajo del proceso.....	34
Autoevaluación	36

UNIDAD II

Captura de Requisitos: “Enumerar los requisitos candidatos”, “Comprender el contexto del sistema”.	37
Objetivos	37
Contenidos de la Unidad	37
Mapa Conceptual	38
Introducción	39
Visión general de la captura de requisitos	39
1- Enumerar los requisitos candidatos	40
2- Comprender el contexto del sistema	40
Modelo de Caso de uso del Negocio	44
¿Están los Caso de uso del negocio relacionados siempre con actores?	45
Vistas del negocio	46
Nombre de un Caso de uso	46
Estructura de un Caso de uso	46
Actor	47
Diagrama de Caso de uso en el modelo de UC de Negocio	47
Autoevaluación	50

UNIDAD III

Captura de Requisitos: “Capturar requisitos funcionales”, “Capturar requisitos no funcionales”.	53
Objetivos	53
Contenidos de la Unidad	53
Mapa Conceptual	54
Introducción	55
¿Qué es la captura de requisitos?	55
¿Por qué la captura de requisitos es complicada?	55
¿Cómo se consigue?	56
¿Cuál es la finalidad?	56
3- Capturar requisitos funcionales	56
a- Encontrar actores y casos de uso - Describir brevemente cada caso	57
b- Priorizar casos de uso	59
c- Detallar un caso de uso	59
d- Prototipar la interfaz del usuario	65
e- Estructurar el modelo de casos de uso	67
4- Capturar requisitos no funcionales	68
Síntesis :Modelos y diagramas de la Captura de Requisitos en el ciclo de vida	69
Cuadro resumen de las relaciones posibles	70
Autoevaluación	71

UNIDAD IV

“Cómo construimos los modelos”	73
Objetivos	73
Contenidos de la Unidad	73
Mapa Conceptual	74
Introducción	75
¿Qué es, entonces un modelo?	75
¿Por qué modelamos?	75

¿Construimos modelos de sistemas complejos porque no podemos comprender el sistema en su totalidad?	75
Modelado orientado a objetos	76
¿Las herramientas son esenciales en el proceso?.....	77
Las herramientas influyen en el proceso.....	78
Diagramas	78
Diagramas estructurales (ESTATICOS)	80
Diagramas de clases.....	80
Diagramas de componentes.....	80
Diagramas de despliegue	80
Diagramas de comportamiento (DINAMICOS)	80
Enunciado y Resolución de un Caso Práctico.....	82
1. Modelado del sistema de Negocios	86
I. I. Modelo de Use-Case del Sistema de Negocios	86
2. Modelado del sistema de Información	105

UNIDAD V

“Introducción básica a los conceptos de Objetos”	125
Objetivos.....	125
Contenidos de la Unidad	125
Mapa Conceptual.....	126
Introducción Básica.....	127
¿Qué es un objeto?.....	127
¿Qué es un tipo objeto?	128
Métodos.....	128
Encapsulado	129
Mensajes	131
BLOB	132
¿Qué es una clase?.....	133
¿Cuál es la diferencia entre operación y método?	134
Herencia	134
Herencia de clase	135
Herencia múltiple.....	135
Polimorfismo.....	138
Percepción y Realidad.....	138
Análisis de la estructura de objetos	140
Asociaciones de objetos	142
Jerarquías de generalización	143
Bases de datos orientadas a objetos.....	144
Tres enfoques de construcción de base de datos Orientados a Objetos	145
Independencia de datos versus encapsulado.....	145
Rendimiento.....	146
Evasión de la redundancia.....	146
Diferencias entre las bases de datos por relación y las bases de datos orientadas a objetos	147
Modelo de examen.....	149

Área Informática

guía de actividades

Sistemas III

Material elaborado por
Ing. Cecilia Savi

INSTITUCION CERVANTES



Guía de Actividades

En esta Guía se presentan distintas actividades, entre ellas, un caso práctico que Ud. deberá resolver individualmente.

El trabajo con el caso práctico tiene por objetivos que Ud. pueda:

- ▶ Resolver problemas en la temática de los sistemas de información.
- ▶ Capacitarse para el estudio y análisis de un sistema de información mediante la utilización de una metodología.
- ▶ Aprender a documentar una carpeta de Anteproyecto y Análisis de un Proyecto.

Características del trabajo:

Presentación de un enunciado, de un caso práctico simulado para trabajar los siguientes contenidos:

Primer instancia (Se trabajan contenidos que corresponden a la unidad II)

Repaso y revisión de los conceptos del Informe Preliminar (aplicados al caso), para lograr consistencia e integración con la nueva etapa de Análisis.

Construcción de cada uno de los modelos vistos, en la captura de requisitos: Modelo de Dominio y Modelo de Negocio.

Segunda Instancia (Se trabajan contenidos que corresponden a la unidad III)

Construcción de los modelos, de la Captura de Requisitos: Diagrama de Casos de Uso de SI, Descripción de los Casos de Uso y Prototipos de Interfaz.

Tiempo de desarrollo: A partir del primer mes/encuentro se comenzará el trabajo paulatinamente, realizando muestras del avance parcial, en cada encuentro, para la revisión por parte del docente. La fecha final de entrega la determina el docente a cargo de curso.

Actividad I

Busque algún ejemplo, en el que se pueda distinguir claramente, que está aplicando una técnica.

tutor métodos y técnicas 

Actividad 2

Identifique, según el concepto de CASO de USO, qué párrafos en este texto responden a una descripción de caso de uso del negocio.

tutor caso de usos

El Centro de Educación y Nutrición, está en la calle 9 de Julio, en el centro de la ciudad. Nace luego de que dos Licenciadas en Nutrición, realizaran un estudio de mercado sobre distintos planes de salud existentes en el área de nutrición que se ofrece en la ciudad de Córdoba.. Este centro se dedica, a brindar novedosos planes de salud, entre los cuales se encuentra el programa de control de crecimiento del niño “ Crecer”. El objetivo del plan crecer, es vigilar el normal crecimiento del niño, a fin de detectar tempranamente posibles desviaciones, para corregirlas a tiempo. El programa no intenta sustituir el control pediátrico periódico, simplemente lo complementa.

La organización, intenta brindar un servicio educativo y asistencial ambulatorio privado en el área de nutrición, con carácter especializado y diferente a todo lo que ya existe en el mercado.

Como estrategias, se han pensado y creado algunos paquetes de servicios, con bajo costo, que son ofrecidas en las guarderías, bajo el nombre de club crecer, la implementación de un Web-site, en las que se pudiesen realizar consultas, por medio de e-mails o por Chat con especialistas, test de preguntas indicativas, sobre la conveniencia de visitar el centro etc.

tutor ▶

Métodos y técnicas

Sería bueno en primera instancia visualizar el esquema que está al comienzo de cada unidad y luego recordar las diferencias conceptuales, entre metodología, modelos y técnicas (Unidad I).

Por último, tomar un ejemplo cotidiano (hacer una torta, arreglar un auto) e identificar la técnica aplicada. Esto les ayudará a comprender mejor otros sistemas de mayor complejidad que se analizan luego.

tutor ▶

Casos de usos

Para descubrir los procesos del negocio, le recomiendo la lectura de los objetivos principales que debe cumplir un use case del negocio, la relaciones con los actores del negocio, y las vistas internas y externas del negocio. Estos temas los puede revisar en la unidad II.

Caso práctico

Actividad 3

Con la ayuda de los tutores (referencias) trabaje los diagramas indicados, construyendo la documentación correspondiente a la Captura de Requisitos.

Caso práctico

tutor : practico unidad IV

Una empresa de telefonía celular que se esta por radicar en nuestra provincia ha diseñado un proceso de venta de modo de adquirir una gran cartera de clientes.

Para las ventas ha contratado una gran cantidad de promotores a través de su área de personal, indicándole cuales son las características que se requieren. Una vez que todo el personal esta contratada (información remitida desde personal a ventas) se le asignan sectores de nuestra ciudad a los cuales los promotores deberán concurrir con el fin de ofrecer el servicio de telefonía celular. Al final de cada día el personal deja las planillas de las personas que afiliaron al servicio en ventas.

Con las planillas de los afiliados se carga la base de datos de clientes y se realiza mensualmente la emisión de los cedulones de cobro y se los remiten a los clientes a través de una empresa de correo privado.

Para pagar los clientes se dirigen a algún banco habilitado a tal fin y realizan el abono que corresponda. Luego el día 11 de cada mes el banco remite un listado con todas las personas que realizaron el pago. Estos datos sirven para ver cuales han sido los clientes que no han pagado. A estos clientes se le envía una citación para que se dirija a la sección finanzas a fin de aclarar su situación.

Mensualmente esta área, ventas, realiza un informe estadístico de ventas que es remitido a la gerencia de la empresa a fin de mantenerla informada de todo lo que sucede.

tutor m. de negocios

tutor m. de dominio

tutor m. de Sistema de Informaçion

tutor ▶ Empresa de Colectivos

Teniendo como referencia el caso practico Unidad IV, completamente desarrollado, lea nuevamente y observe la secuencia de pasos, que es conveniente seguir para desarrollar cada uno de los modelos que corresponden a la Captura de requisitos .

tutor ▶	M. de Negocios Diagrama de Casos de Uso del Mod. Negocios.
----------------	--


tutor ▶	M. de Dominio Diagrama de clases
----------------	--


tutor ▶	M. de Sistema de Información Diagrama Casos de Uso de SI
----------------	--

Actividad 4

Plantee las **herramientas que falta** utilizar en el caso anterior, para una captura de Requisitos completa y desarrolle las que Ud. crea necesarias.

tutor m. de negocios 

tutor m. de dominio 

tutor m. de Sistema de Información 

tutor ▶	M. de Negocios Especificación de procesos del negocio.
----------------	--

tutor ▶	M. de Dominio Listado de clases y atributos
----------------	---

tutor ▶	M. de Sistema de Información Plantillas de proceso – prototipos
----------------	---



Es necesario que considere la importancia de estas herramientas, y se cuestione si son complementarias o no, a las ya vistas.

Actividad 5

Diseñe por lo menos tres prototipos de interfaz para la Empresa de telefonía, que satisfaga los requerimientos del usuario.

tutor prototipo 

tutor ▶

Prototipo

El siguiente es un esquema, que relaciona, los datos guardados del sistema y un formato de interfaz, en los que se visualizan los mismos.

