UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

Autor: María del Rocio Carrillo Velázquez

INFORMÁTICA V (Análisis y diseño orientado a objetos)		Clave: 1565
Plan: 2005		Créditos: 8
Licenciatura: Informática		Semestre: 5º
Área: Informática (Desarrollo de sistemas)		Hrs. Asesoría: 2
Requisitos: Informática II (Estructura de datos estáticas y dinámicas en memoria principal)		Hrs. Por semana: 4
Tipo de asignatura:	Obligatoria (x)	Optativa ( )

Objetivo general de la asignatura

Al finalizar el curso, el alumno aprenderá a desarrollar sistemas utilizando metodologías para el análisis y diseño orientados a objetos.

Temario oficial (horas sugeridas 64)

1. Introducción (4h)
2. Metodologías orientadas a objetos (4h)
3. Planeación y elaboración (6h)
4. Análisis orientado a objetos (26h)
5. Diseño orientado a objetos (24h)

Introducción

Actualmente la producción de sistemas de software se enfocan a metodologías ágiles, que facilitan su desarrollo, restando tiempos, obteniendo productos de gran calidad, competitivos comparados con otros productos en el mercado. Uno de los paradigmas de vanguardia en la ingeniería de software es el Paradigma de la Orientación a Objetos; por ello este trabajo se dedica a proporcionar los elementos necesarios para realizar diseños de sistemas utilizando todas las facilidades y herramientas que permitan obtener software de gran calidad.

Este trabajo se encuentra organizado en cinco temas, en el primero (*Introducción)* se presenta el concepto del paradigma Orientado a Objetos, el ciclo de vida de los sistemas utilizando este paradigma y una breve descripción de otros paradigmas de desarrollo que existen.

En el segundo tema *(Metodologías Orientadas a Objetos)* se definen dos herramientas de modelado que son UML e IDEF4, utilizadas en el diseño y documentación de sistemas.

En el tercer tema *(Planeación y Elaboración)* se presentan algunas técnicas para recopilar requerimientos, y de ellos se podrán identificar los casos de uso, las interfaces con otros usuarios y con otros sistemas, las entradas y las salidas que debe tener el sistema y que cubre las necesidades del usuario.

En el cuarto tema *(Análisis Orientado a Objetos)* se presentan los elementos para estructurar y modelar de manera conceptual el nuevo sistema. Para ello se contará con las herramientas que permitan identificar a los objetos involucrados con sus responsabilidades y atributos, además de elaborar un diccionario del modelo, diagramas de secuencia del sistema y contratos.

También, en el quinto tema (*Diseño Orientado a Objetos*) está configurado de tal modo que se podrá: elaborar diagramas de colaboración; determinar la visibilidad de los objetos participantes en los diferentes casos de uso; elaborar diagramas de clases de diseño; diseñar la interfaz de usuario; elegir el lenguaje de programación a utilizar así como el manejador de base de datos y la plataforma en la que se ejecutará el sistema.

Por último, como el desarrollo de un sistema no solo lleva consigo las líneas de código, se podrá reconocer cuál debe ser el contenido de la documentación, con lo que quedarán cubiertos todos los requerimientos del usuario y se obtendrá el diseño global del nuevo software.

TEMA 1. INTRODUCCIÓN

Objetivo particular

El alumno conocerá el paradigma de la orientación a objetos, y el ciclo de vida de los sistemas haciendo uso de la orientación a objetos. Así mismo conocerá el proceso de desarrollo de software en sus diferentes modelos: iterativo, fases, prototipos, prototipo desechable o espiral, cascada, paralelo y ágil.

Temario detallado

1.1 El paradigma de la orientación a objetos

1.2 El ciclo de vida de los sistemas utilizando la orientación a objetos

1.3 Introducción al proceso de desarrollo de software

- 1.3.1 Iterativo
- 1.3.2 Fases
- 1.3.3 Prototipo
- 1.3.4 Prototipo desechable o espiral
- 1.3.5 Cascada
- 1.3.6 Paralelo
- 1.3.7 Ágil

Introducción

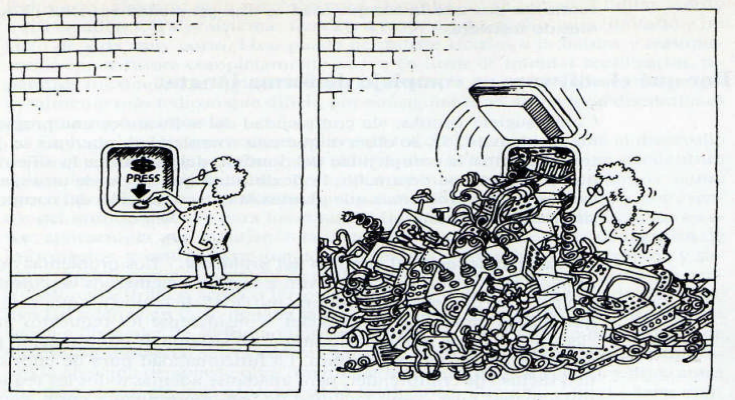
La programación Orientada a Objetos se considera uno de los paradigmas de vanguardia en lo que se refiere a la Ingeniería de Software, ya que facilita el desarrollo de sistemas, restando tiempo y permitiendo que se planteen los objetivos del software con mayor claridad. La importancia de la metodología de este paradigma surge de la necesidad de desarrollar software de gran “calidad” y de la gran competencia de contenidos que existe actualmente en el mercado.

Además los nuevos desarrollos requieren cumplir con las exigencias y expectativas de los usuarios, por lo que se debe contar con herramientas especializadas que proporcionen beneficios a los desarrolladores, mejorando la productividad y aumentando la confiabilidad en su producto.

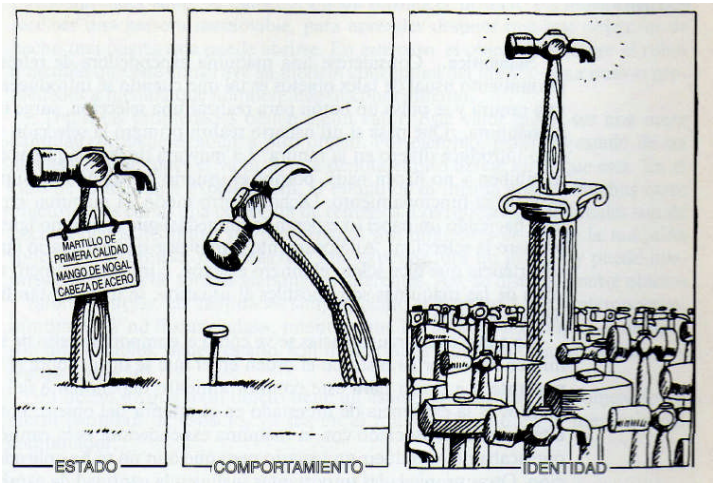
1.1 El paradigma de la orientación a objetos

Paradigma es “Un ejemplo que sirve de norma”, la definición de la palabra abarca un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento; dicho de otra manera, es una forma de ver el mundo, y es en este sentido que la programación orientada a objetos es un nuevo paradigma.

Esta técnica es una forma similar a como se expresa la gente de las cosas en la vida real. Y para entender mejor esta técnica se debe comenzar por los principales conceptos y principios que utiliza, por mencionar algunos: objeto, clase, abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia y persistencia. Y lo importante que es el hecho de conjugar todos estos elementos.

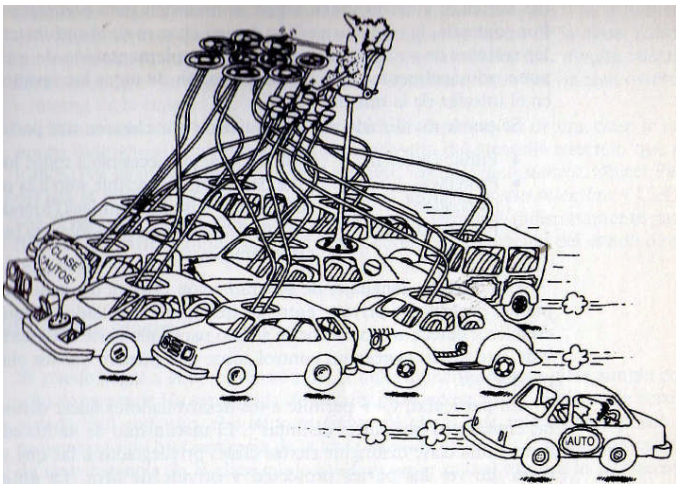


La tarea del equipo de desarrollo de software es ofrecer ilusión de simplicidad (Booch, 1996: 6)



Un objeto tiene estado, exhibe algún comportamiento bien definido, tiene una identidad única. (Booch, 1996: 97)

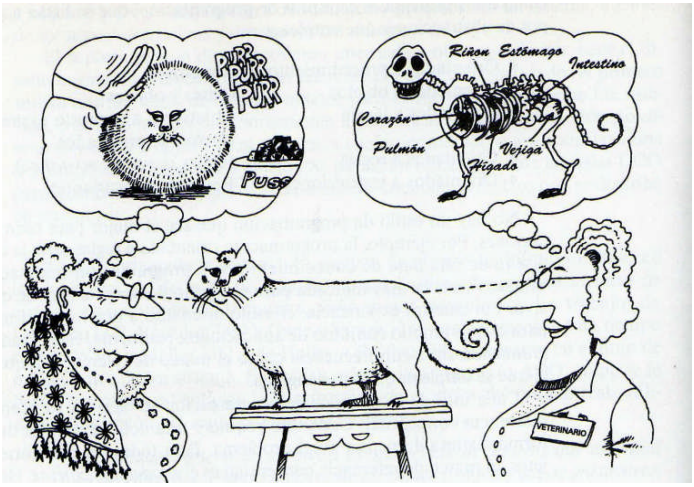
Un objeto es una entidad que consta de un conjunto de propiedades o atributos referenciado a datos y de comportamientos o funcionalidad llamados métodos, éstos, consecuentemente, reaccionan a sucesos o eventos. Se relacionan directamente con los objetos del mundo real que nos rodea; y se puede decir que un objeto es una instancia a una clase.



Una clase representa un conjunto de objetos que comparten una estructura común y un comportamiento común (Booch, 1996: 121)

Una clase es un conjunto de objetos que comparten una estructura y un comportamiento común.[1]. Sus dos componentes son:

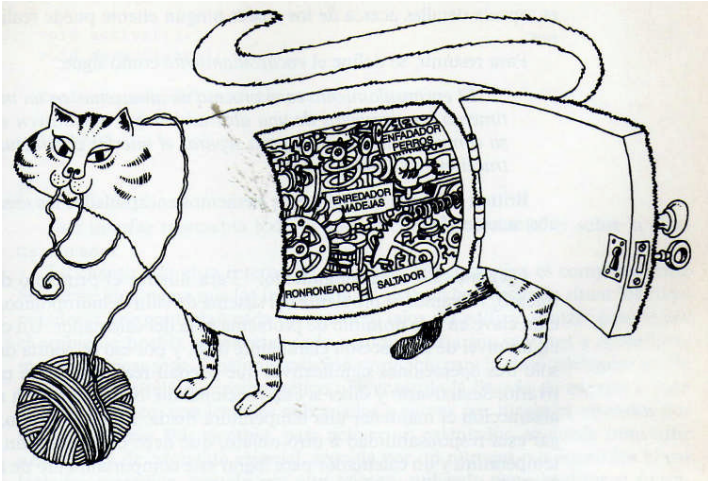
1) Atributos: que determinan una estructura de almacenamiento para cada objeto de la clase, por ejemplo si la clase es EMPLEADO sus atributos serán: domicilio, teléfono, edad, sexo, talla, peso, CURP, RFC, número de licencia, fecha de nacimiento, color favorito, etc.
2) Métodos: Son las operaciones aplicables a los objetos y único modo de acceder a los atributos, en la clase EMPLEADO sus métodos pueden ser: ObtenerEdad, GuardarDomicilio, LeerTalla, etc.



La abstracción se centra en las características esenciales de algún objeto, en relación con la perspectiva del observador
(Booch, 1996: 46)

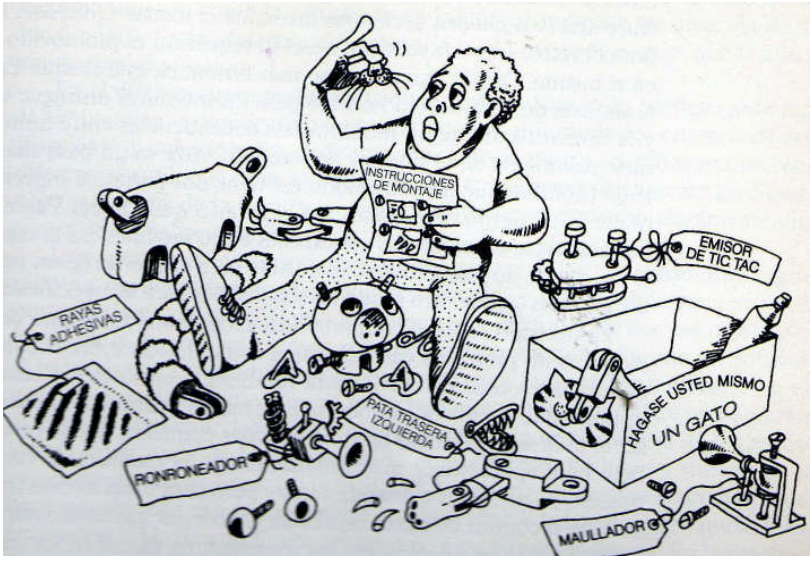
Una abstracción denota las características esenciales de un objeto, consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan, con el objeto de destacar de manera clara otros aspectos; se refiere a enfatizar el ¿qué hace? sobre el ¿cómo lo hace?

[1] Erick Téllez: "Complejidad del software", disponible en línea: http://ericktellez.iespana.es/resumen_para_uml.doc, recuperado el 17/09/09.



El encapsulamiento oculta los detalles de la implementación de un objeto. (Booch, 1996: 55)

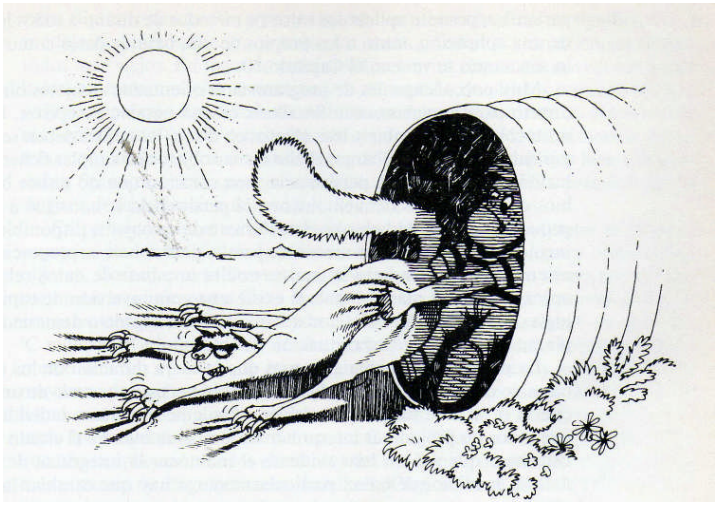
El encapsulamiento se realiza cuando son empaquetados métodos y atributos dentro de un objeto.



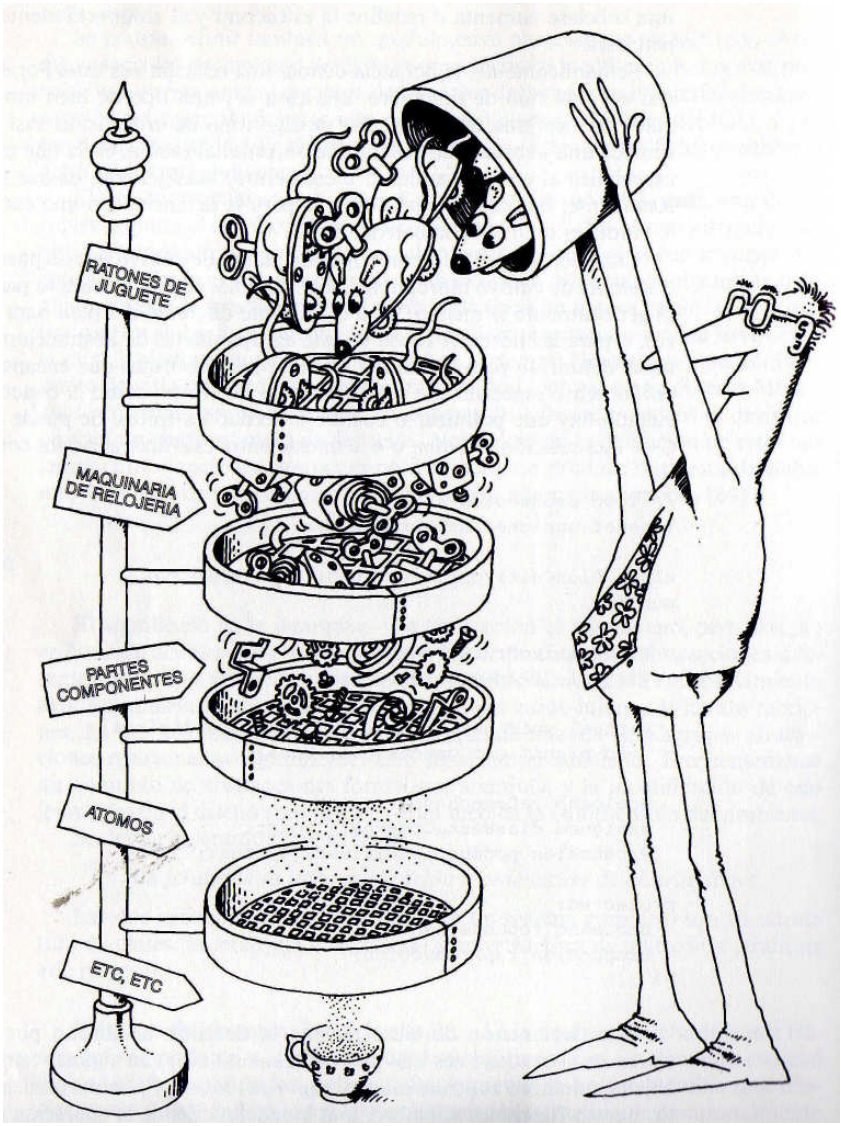
La modularidad empaqueta las abstracciones en unidades discretas. (Booch, 1996: 61)

Modularidad. Es una propiedad que permite dividir la aplicación en partes más pequeñas a las que llamamos módulos; mismos que deben estar tan independientes entre sí, como sea posible, es decir débilmente acoplados pero altamente cohesivos.

El hecho de dividir un programa en componentes individuales reduce su complejidad, con lo que nos facilitará a los desarrolladores crear fronteras, elaborar una buena documentación y sobre todo tener el control y comprensión del programa en general.

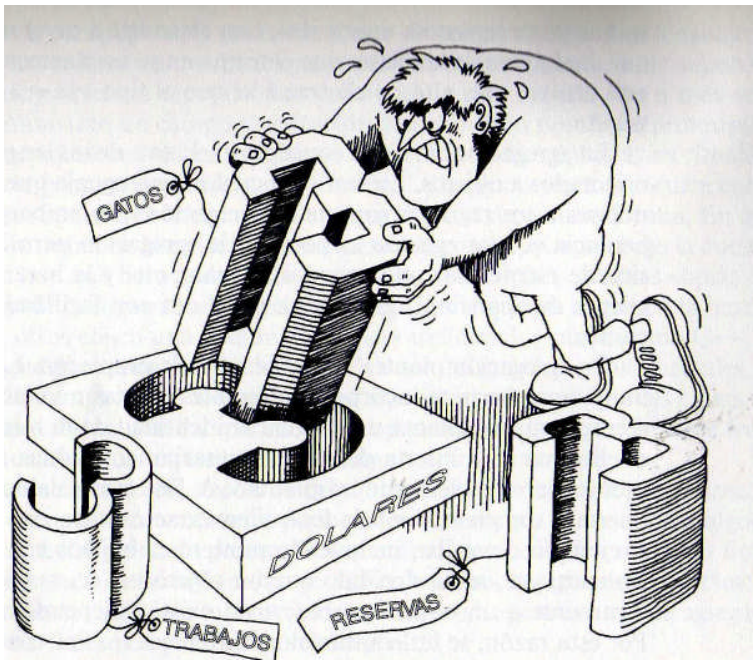


La persistencia conserva el estado de un objeto en el tiempo y en el espacio. (Booch, 1996: 85)



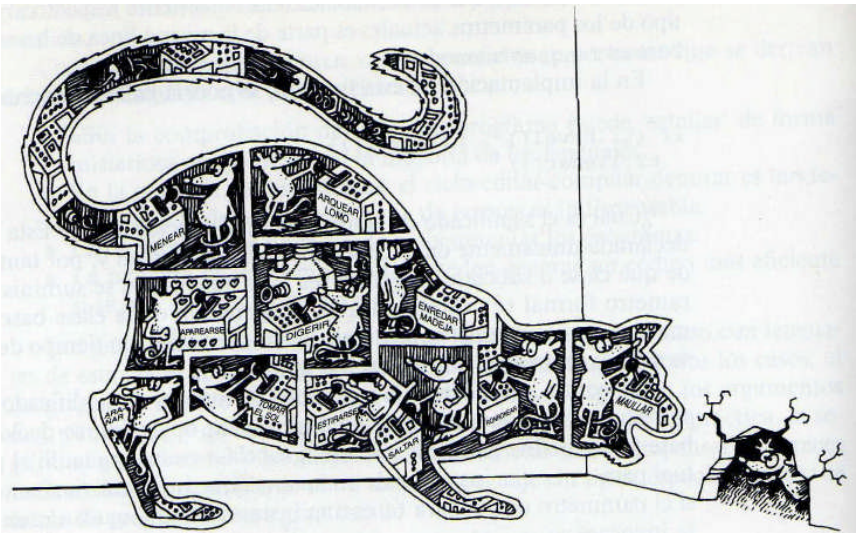
Las abstracciones forman una jerarquía. (Booch, 1996: 68)

Frecuentemente un conjunto de abstracciones forma una jerarquía, y la identificación de esas jerarquías en el diseño simplifica en gran medida la comprensión del problema que se va a implementar.



La comprobación estricta de tipos impide que se mezclen abstracciones. (Booch, 1996: 74)

Cabe mencionar que los objetos no son intercambiables en el caso de que sean de tipos distintos, por ejemplo no se pueden intercambiar el objeto AUTOS con el objeto LIBROS.



La concurrencia permite a diferentes objetos actuar al mismo tiempo. (Booch, 1996:pág. 82)

Así pues, conociendo los conceptos anteriores y aplicados al objetivo de desarrollar software de calidad Orientado a objetos, se deben realizar las siguientes actividades previas:

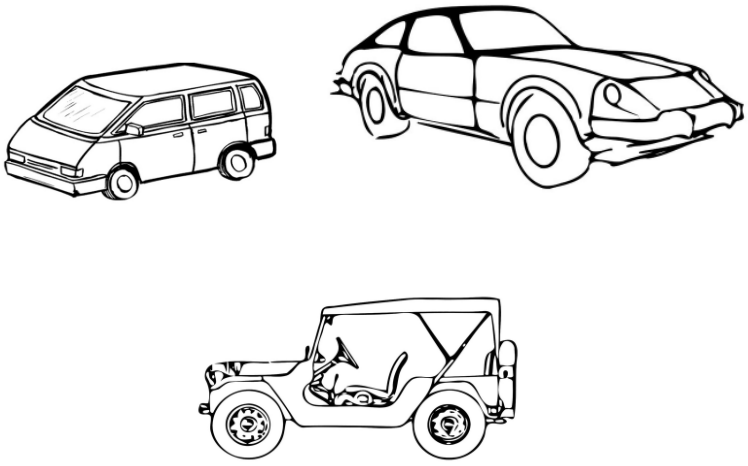
- Encontrar los objetos relevantes
- Encontrar las operaciones para los tipos de objetos
- Describir los tipos de objetos
- Encontrar relaciones entre objetos
- Utilizar los tipos de objetos y las relaciones para estructurar el software

Por Ejemplo:

Objeto: COCHE

Atributos

Métodos = Funciones que puede realizar	**Propiedades =** Tiene las características
•Ir •Parar •Girar a la derecha •Girar a la izquierda •Arrancar	•Color •Velocidad •Tamaño •Carburante



Objetos de la clase COCHE

Los objetos con estados similares y el mismo comportamiento se agrupan en **clases**.

1. 2 El ciclo de vida de los sistemas utilizando la orientación a objetos

El ciclo de vida de los sistemas orientados a objetos es una descripción de las etapas que deberán ser realizadas cuando se produce un proyecto de software. Este ciclo de vida es dividido en pequeñas etapas llamadas fases.

Fase: Análisis

PROPÓSITO	ACTIVIDADES
El propósito del análisis es proporcionar una descripción del problema. La descripción debe ser completa, consistente, legible y revisable por las diversas partes interesadas, y verificable frente a la realidad. Esto es, el propósito del análisis es proporcionar un modelo del comportamiento del sistema. El análisis debe dar lugar a una declaración de lo que hace el sistema, no de cómo lo hace.	1) El análisis de dominios busca identificar las clases y objetos que son comunes a un dominio de problema particular. 2) La planificación de escenarios es la actividad central del análisis. Los eventos para esta actividad son: • Identificar todos los puntos funcionales principales del sistema y, si es posible, reunirlos en grupos de comportamientos relacionados funcionalmente. • Realizar una narración de sucesos de un escenario, usando técnicas de casos de uso y análisis de comportamientos. • Documentar con diagramas de objetos • Elaborar un guión que muestre los eventos que disparan el escenario. • Documentar cualquier situación de suposiciones, restricciones. • Generar escenarios que ilustren comportamiento bajo condiciones excepcionales. • Elaborar diagramas de clases. • Actualizar el diccionario de datos en evolución.

Fase: Diseño

PROPÓSITO	ACTIVIDADES	PRODUCTOS
El propósito del diseño es crear una arquitectura para la implantación que va a desplegarse, y establecer las políticas tácticas comunes que deben utilizarse por parte de elementos dispares del sistema. Se recomienda que se inicie hasta que se tenga un modelo completo del comportamiento del sistema.	**1) Planificación arquitectónica** Los eventos para esta actividad son: • Considerar el agrupamiento de puntos funcionales partiendo de los productos del análisis, y asignar éstos a capas y particiones de la arquitectura. • Validar la arquitectura creando una versión ejecutable que satisfaga parcialmente la semántica de unos pocos escenarios interesantes del sistema, tal como se derivan del análisis. • Instrumentar esa arquitectura y evaluar sus puntos débiles y fuertes. **2) Diseño táctico** Un orden típico de los eventos para esta actividad es como sigue: • Enumerar las políticas comunes que deben seguir elementos dispares de la arquitectura, cuestiones independientes del dominio, como gestión de memoria, manejo de errores, etc. • Para cada política común, desarrolla un escenario que describa la semántica de esa política. Además, capturar su semántica en forma de un prototipo ejecutable que pueda instrumentarse y refinarse. • Documentar cada política y efectuar un recorrido parejo, para difundir su visión	1) una descripción de la arquitectura mediante diagramas de clases, objetos. 2) descripciones de políticas tácticas comunes.

	arquitectónica. **3) Planificación de versiones** Un orden típico de los eventos sería: • Dados los escenarios identificados durante el análisis, organizarlos en orden de comportamientos fundamentales a periféricos. Dando prioridad a los escenarios que mejor puedan completarse con un equipo que incluya personal experto del dominio, personal de análisis, de arquitectura y de control de calidad. • Asignar los puntos funcionales mencionados a una serie de versiones arquitectónicas cuyo producto final representa el sistema de producción. • Ajustar las metas y planes de esta corriente de versiones de forma que las fechas de entrega estén lo bastante separadas como para permitir un tiempo de desarrollo adecuado, y como para que las versiones estén sincronizadas con otras actividades de desarrollo, como documentación y pruebas de campo. • Comenzar la planificación de tareas, en la que se identifica una estructura de división de trabajo, y se identifican los recursos de desarrollo que son necesarios para conseguir cada versión arquitectónica.	

Fase: Implementación

PROPÓSITO	ACTIVIDADES	PRODUCTOS
El propósito de esta fase es aumentar y cambiar la implantación mediante refinamiento sucesivo, lo que conduce en última instancia al sistema en producción. Se refiere a intentar satisfacer una serie de restricciones que compiten, incluyendo funcionalidad, tiempo y espacio: siempre se está limitado por la restricción mayor.	1) La aplicación del microproceso. Análisis de requisitos para la siguiente versión, procede al diseño de una arquitectura y continúa con la invención de las clases y objetos necesarios para implantar este diseño. 2) La gestión de cambios. En reconocimiento a la naturaleza incremental e iterativa de los sistemas orientados a objetos, en espera de cambios durante su evolución.	El producto principal de la implementación es una corriente de versiones ejecutables que representan sucesivos refinamientos a la versión inicial de la arquitectura.

Fase: Mantenimiento

PROPÓSITO	ACTIVIDADES	PRODUCTOS
El mantenimiento es la actividad de gestionar la implantación postventa. Esta fase es en gran medida una continuación de la fase anterior, excepto en que la	Además de las actividades usuales de la evolución, el mantenimiento implica una actividad de planificación que da prioridad a las tareas de la lista-	Ya que el mantenimiento es en cierto sentido la evolución continuada de un sistema, sus productos son similares a los de la fase previa de implementación. Además, el mantenimiento implica gestionar una lista-guión de nuevas tareas. Inmediatamente, tras la entrega del sistema en producción, sus desarrolladores y usuarios finales ya tendrán probablemente una serie de mejoras o modificaciones que les gustaría realizar en

innovación arquitectónica es menos preocupante.	guión.	las versiones de producción siguientes, y que no se efectuaron en el producto inicial por otras razones. Además, a medida que hay más usuarios probando el sistema, se descubrirán nuevos errores y patrones de uso que el control de calidad no pudo anticipar.

Fase: Retiro

OBJETIVO
El objetivo de esta fase es preparar el entorno de retiro del software que cumplió su ciclo de vida para ponerlo fuera de servicio, esto ocurre cuando existe en el mercado productos que pueden sustituirlo, realizando las mismas actividades y necesidades del usuario, y que además lo considere obsoleto. Existen otros factores que influyen en esta decisión, que son condiciones socio-económicas del entorno, leyes, normas o influencias geopolíticas. Generalmente en esta fase no se obtienen resultados económicos; La actividad principal en esta fase es desmantelar el software.

1.3 Introducción al proceso de desarrollo de software

El proceso de desarrollo de software se refiere a un conjunto de actividades que comienza con la identificación de necesidades y termina con el retiro del software. Cuyo propósito primordial es la producción de software de calidad, que sea eficaz y eficiente, y que cumpla con los requisitos del cliente. Está formado por seis etapas bien definidas que son: obtención de requisitos de software, diseño, implementación, pruebas, implantación, mantener y mejorar. Si consideramos que el software que se obtiene al aplicar este proceso es un producto, podemos visualizar varios productos que se obtienen a lo largo del mismo, que pueden ser: documentos, manuales, diagramas, código fuente, objeto, etc.

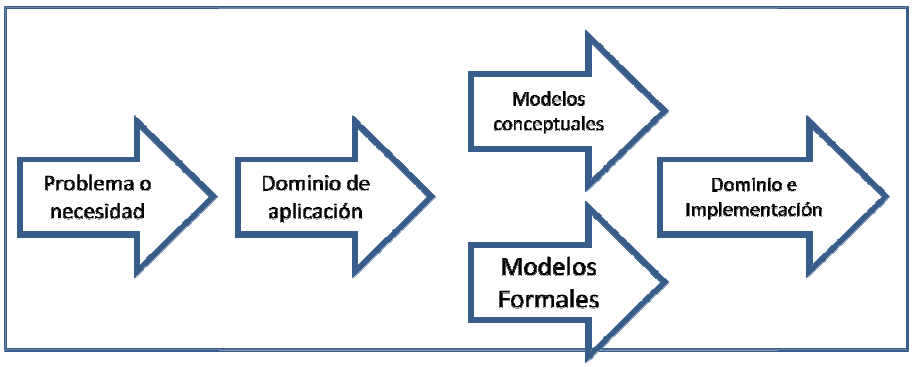


Figura 1.1. Proceso de desarrollo de Software

En sí el proceso software refleja cómo se usa la experiencia humana en la construcción del software y cómo se aplica en un dominio concreto.

En el siguiente cuadro se definen las principales actividades que deben realizarse en cada una de las fases del proceso de desarrollo de software, considera que pueden ser aplicadas a los diferentes modelos de desarrollo de software que existen y que conoceremos a continuación.

FASE	ACTIVIDADES GENERALES
Análisis de requisitos	1.- Obtención del conocimiento. 2.- Modelado Conceptual. 3.- Validación de Requisitos. 4.- Negociación de Requisitos. 5.- Documento de Especificación de requisitos. 6.- Gestión de Requisitos

Diseño del Software	1.- Diseño de Alto Nivel Diagrama Conceptual Definición de Casos de Uso de alto nivel Diagrama de Secuencia de Iteración con el sistema 2.- Diseño de la Base de Datos
Implementación y Pruebas	1.- Código Fuente 2.- Código Objeto 3.- Documento de resultado de las pruebas
Evolución / Mejora	1.- Productos del proceso de software modificados
Retiro	1.- Cronograma de retiro

1.3.1 Iterativo

En el proceso de desarrollo iterativo se elabora la arquitectura inicial completa del sistema, seguida de incrementos y versiones parciales del mismo, llamadas iteraciones. Cada iteración desarrolla su propio ciclo de vida y va agregando cierta funcionalidad o alguna mejora al sistema. Conforme se concluye cada etapa, se realizan pruebas de verificación y se va integrando a la versión anterior del sistema.

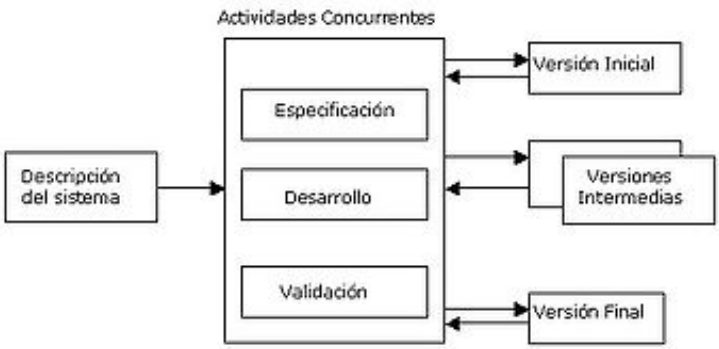


Figura 1.2. Proceso de desarrollo Iterativo e Incremental

Es recomendable efectuar una evaluación de la última versión del sistema con respecto a las versiones futuras. Las actividades en este modelo se dividen en procesos y subprocesos, por lo que es importante conocer perfectamente los requisitos iniciales del sistema.

En la siguiente figura se observa que existen ciertas actividades de desarrollo que son realizadas en cada incremento, y que se realizan en paralelo, así por ejemplo, se observa que mientras se realiza el diseño detalle del primer incremento ya se está realizando en análisis del segundo. Aquí observamos que un incremento no necesariamente se iniciará durante la fase de diseño del anterior, puede ser posterior e incluso antes, en cualquier tiempo de la etapa previa. Y que cada incremento concluye con la actividad de Operación y Mantenimiento, que es donde se entrega el producto de manera parcial al cliente.

El momento de inicio de cada incremento depende de varios factores como: el tipo de sistema; la independencia o dependencia entre incrementos, la capacidad y cantidad de personal involucrado en el desarrollo, etc.

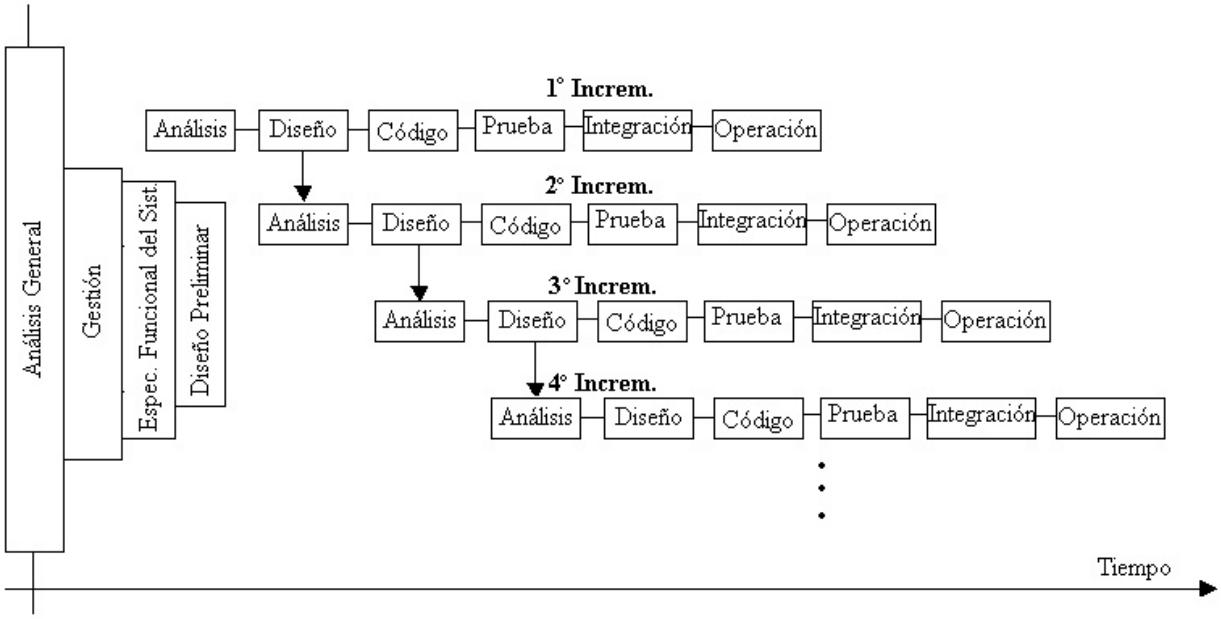


Figura 1.3. Proceso de desarrollo Iterativo e Incremental

1.3.2 Fases

Este modelo es considerado también el ciclo de vida del software, porque se realiza siguiendo una serie de fases o pasos de manera secuencial, es decir, la segunda no puede realizarse sin que se haya concluido la anterior y así sucesivamente con las posteriores.

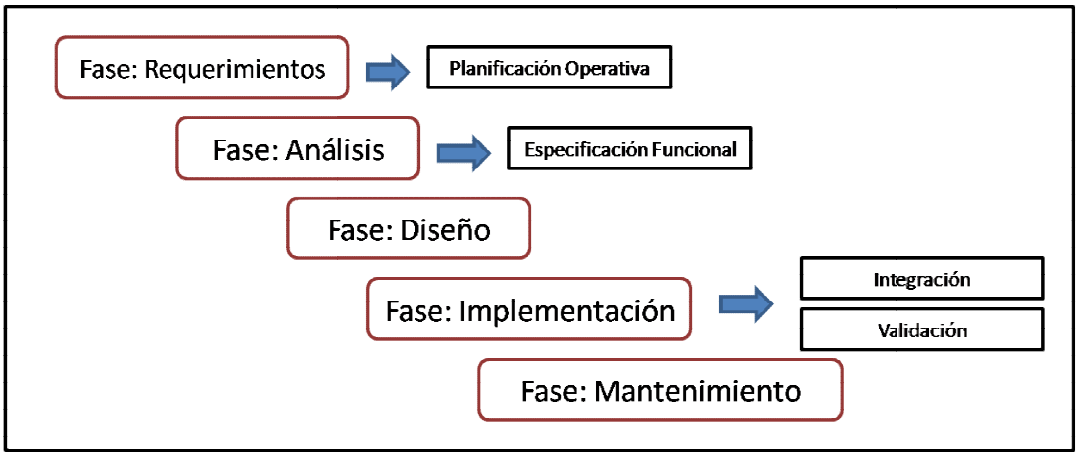


Figura 1.4. Proceso de desarrollo por Fases

En la figura se observa las principales fases de este modelo, así que se inicia con la fase de requerimientos en donde se define el problema por resolver, las necesidades del usuario que debe cubrir el software, las metas del proyecto, así como sus características de calidad y muy importante las restricciones materiales y humanas que existen para su realización. Esto lleva directamente a realizar una actividad que es la planificación operativa. En la Fase de Análisis se analizan los requerimientos y son representados en un documento de especificaciones, que es lo que “debe ser hecho”; es en esta etapa que permite entregar una visión sobre el proyecto de alto nivel, también permite hacer una planificación de los recursos sobre una escala de tiempo. Esta fase implica realizar una tarea que es la especificación funcional, en ella se define el software sobre el cual se desarrollará.

En la Fase de Diseño se describe cómo es que el sistema cubrirá todos los requerimientos, pudiendo detallar el proceso general en módulos o subsistemas, debidamente documentados.

En la Fase de Implementación se desarrolla y codifica en un lenguaje de programación, que dependiendo del tamaño del proyecto se distribuye a uno o más programadores o grupo de programadores, que se encargarán de implementarlo y que como objetivo primordial tienen que asegurar que todas las funciones estén correctamente implementadas dentro del sistema. Realizando dos tareas muy importantes: la unión de los diferentes módulos en un todo y las validaciones del proceso como un todo. Posteriormente se presenta al usuario y si es aceptado, se instala y se continúa con la última fase que es la de Mantenimiento; en ella se efectúan mejoras al sistema, adaptaciones por ejemplo a un nuevo ambiente, a nuevas definiciones, etc., perfecciones y acciones preventivas; y con esta fase se concluye el modelo.

1.3.3. Prototipo

Un prototipo es un modelo a escala de lo real, y partiendo de esta definición este modelo se basa en el desarrollo de prototipos, que son utilizados para mostrar al usuario un producto que contenga partes del funcionamiento del sistema, con ello se realizará una retroalimentación de manera periódica y servirá para ir modelando y ajustando a las necesidades del usuario. Comparando con una simple maqueta, éste si cuenta con archivos y datos reales e incluso cierta programación. Sirve también para confirmar que lo que se muestra y se está implementando es efectivamente lo que se necesita o bien puedan solicitar por comparación y se prepara una nueva versión del prototipo y se enseña otra vez.

Las principales ventajas de utilizar este modelo son, nos comenta J. Barranco (2001) que:

1. Facilita la comunicación entre cliente y desarrollador, permitiendo la obtención de sistemas ajustados a las necesidades reales.
2. Permite al cliente concretar sus necesidades.
3. Permite la obtención de resultados visibles del desarrollo en sus primeras etapas.
4. Disminuye el riesgo de error en el desarrollo del software.
5. Facilita la administración y gestión de cambios durante el desarrollo.
6. Aumenta la productividad del equipo de proyecto. (p. 42)

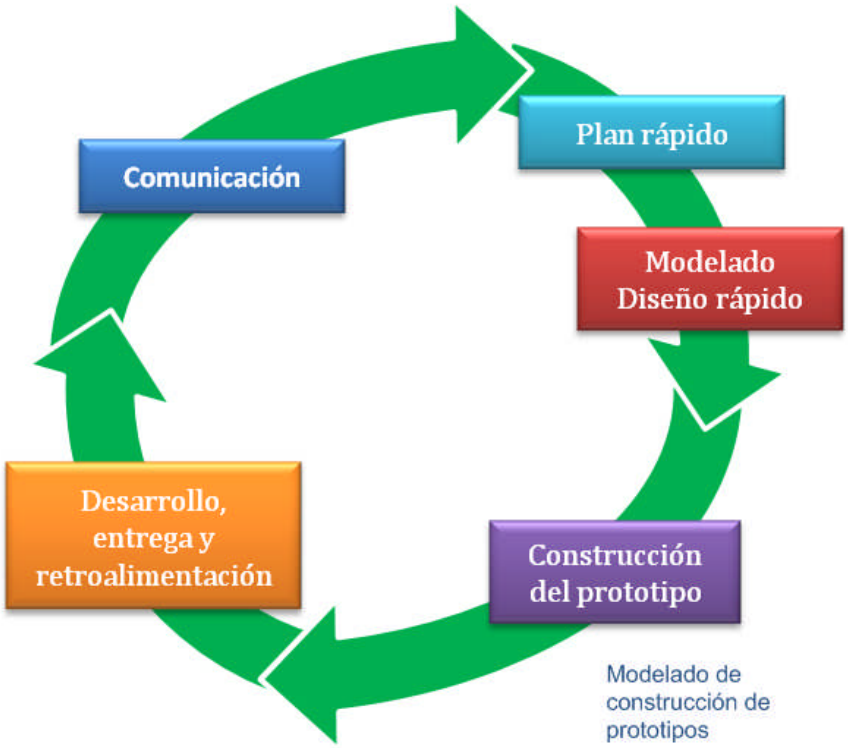


Figura 1.5. Proceso de desarrollo por construcción de Prototipos

Sin embargo, también existen motivos por los que puede resultar problemático el uso de este modelo, por ejemplo que el cliente ve funcionando una versión rápida del software y eso puede retrasar la aplicación de estándares de calidad, por las prisas de hacer que funcione; otro aspecto es que al hacer prototipos tal vez se utilicen herramientas que no sean apropiadas a los requerimientos, es decir un lenguaje de programación o un sistema operativo inapropiado.

1.3.4 Prototipo desechable o espiral

El modelo de desarrollo en espiral, también llamado prototipo desechable, se representa como una espiral, en lugar de una serie de actividades sucesivas secuenciales.

Figura 1.6. Proceso de desarrollo en Espiral

Consta de cuatro fases bien definidas: Determinación de los objetivos, análisis de riesgos, desarrollo y pruebas, y planificación; fases que se ejecutan de manera cíclica, es decir se inicia una por una y al concluir la última se vuelve a iniciar por la primera.

> En la fase de definición de objetivos se definen restricciones del proceso, se realiza un diseño detallado y se identifican riesgos, planeando estrategias alternativas a estos. En la segunda fase de evaluación y reducción de riesgos, se efectúa un análisis detallado de los riesgos identificados, y se pueden desarrollar prototipos para disminuirlos. En la fase de desarrollo y validación se desarrolla software con el modelo que se decide utilizar (cascada, evolutivo, etc.). En la cuarta y última fase se determina si es necesario continuar con otro ciclo, es decir se planea la siguiente fase del proyecto;
>
> Este modelo a diferencia de los otros toma en consideración explícitamente el riesgo, esta es una actividad importante en la administración del proyecto. (Letier, s/f, p. 10)

1.3.5. Cascada

En un modelo en cascada un proyecto progresa a través de una secuencia ordenada de pasos, es decir no se puede progresar con el siguiente hasta que no se haya concluido el anterior. Estos pasos o fases son:

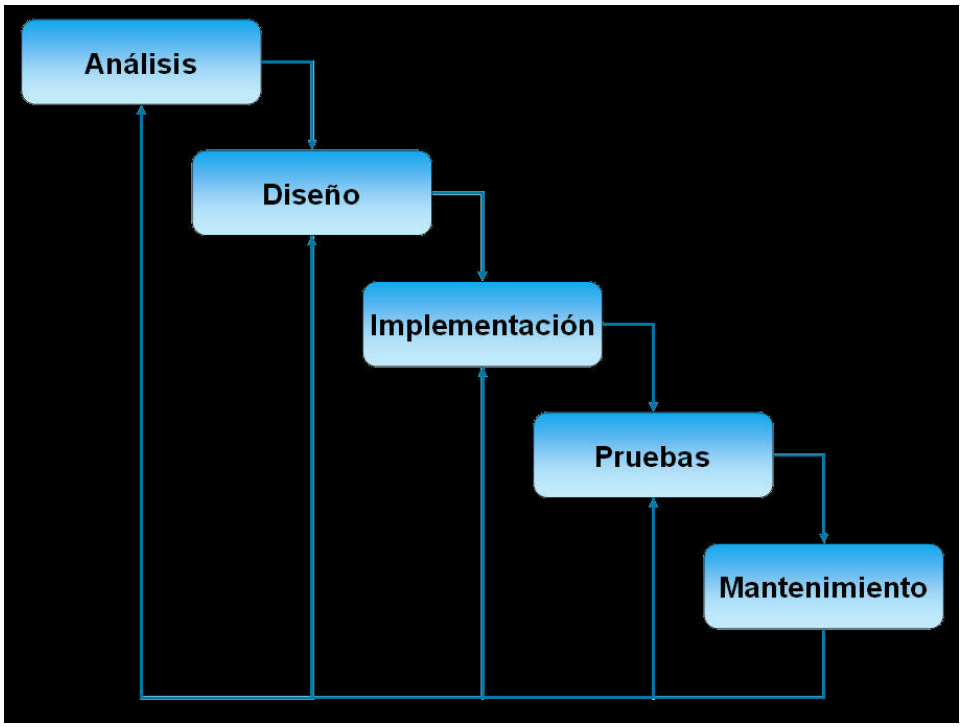


Figura 1.7. Proceso de desarrollo en Cascada

1.3.6. Paralelo

Este modelo de desarrollo de software, según Lambin (2003), se apoya en

> un equipo de proyecto, el cuál acelera el proceso de desarrollo ya que organiza el trabajo desde el principio hasta el final. Más que evolucionar la forma estructurada de una etapa a otra como lo hemos visto en modelos anteriores, esté resulta de la interacción de un equipo interfuncional, y surge de las interacciones entre los miembros del equipo.
>
> Este modelo se refiere a que varios integrantes según el rol que desempeñan, pueden realizar actividades, procesos, cambios, mejoras, etc. de manera simultánea o "paralela" a lo largo de la duración del proceso de desarrollo. (p. 543)

Este modelo tiene ciertas ventajas, las más importantes según Lambin (2003) son:

- Permite una mejor coordinación entre los participantes.
- Permite organizar simultáneamente diversas actividades, lo que lleva a que el proceso se realice más rápido.
- Cada actividad está mejor controlada, ya que determina directamente las actividades subsiguientes.
- Proporciona ganancias de tiempo considerables por el hecho del trabajo más intensivo y de la mejora de la coordinación.
- Este tipo de desarrollo en paralelo fomenta y mejora el trabajo en equipo. (p. 544)

1.3.7. Ágil

Los modelos de desarrollo de software ágil son modelos enfocados a desarrollos iterativos e incrementales, que requieren la participación constante del usuario durante el proceso de desarrollo, y eliminan la carga de trabajo durante el desarrollo con el objetivo de hacer los desarrollos más rápidos. Este tipo de modelos son llamados también metodologías livianas.

> El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar como mínimo una semana y máximo cuatro semanas; Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.[2]

[2] Wikipedia: "Desarrollo ágil de software", actualizado el 20/10/09; disponible en línea: http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software, recuperado el 24/10/09.

Los principios de desarrollo de software mediante métodos ágiles son:

Principio	Descripción
Participación del cliente	Los clientes deben estar fuertemente implicados en todo el proceso de desarrollo. Su papel es proporcionar y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del sistema.
Entrega incremental	El software se desarrolla en incrementos, donde el cliente especifica los requerimientos a incluir en cada incremento.
Personas, no procesos	Se deben reconocer y explotar las habilidades del equipo de desarrollo. Se les debe dejar desarrollar sus propias formas de trabajar, sin procesos formales, a los miembros del equipo.
Aceptar el cambio	Se debe contar con que los requerimientos del sistema cambian, por lo que el sistema se diseña para dar cabida a estos cambios.

Cuadro 1.1. Los principios de los métodos ágiles. (Sommerville, 2005)

Bibliografía del tema 1

Barranco de Areba, Jesús. (2001). *Metodología del análisis estructurado de sistemas*, Madrid, Universidad Pontificia Comillas de Madrid.

Booch, Grady. (1996). *Análisis y diseño orientado a objetos*, con aplicaciones, 2ª ed., México: Addison Wessley Longman.

Lambin, Jean Jacques. (2003). *Marketing estratégico*, Madrid, Escuela Superior de Gestión Comercial y Marketing

Larman, Craig. (1999). *Análisis y diseño orientado a objetos con UML*, México: Pearson.

Letelier Patricio. (s/f). Proceso de desarrollo de software, Universidad Politécnica de Valencia; Departamento de Sistemas Informáticas y Computación, disponible en línea: http://www.dsic.upv.es/asignaturas/facultad/lsi/doc/IntroduccionProcesoSW.doc, recuperado el 22/10/09.

Sommerville, Ian. (2005). *Ingeniería de Software*, 7ª edición, Madrid: Pearson Education

Actividades de aprendizaje

A.1.1. Realiza un cuadro comparativo entre los diferentes modelos de desarrollo de software orientado a objetos, indicando características y fases de cada una.

A.1.2. Realiza el análisis de requerimientos en un proceso de elaboración de cualquier prenda de vestir.

A.1.3. Define las operaciones o métodos y propiedades de las siguientes clases: *Empleado, Factura, Amigos, Estados de la República.*

Cuestionario de autoevaluación

1. ¿Qué es el paradigma de la orientación a objetos?
2. ¿Cuál es el propósito del análisis y diseño de sistemas?
3. ¿En qué difiere la programación orientada a objetos de la programación tradicional?
4. Define qué es un Objeto o Instancia.
5. ¿Cuál es el propósito y las actividades principales de la fase de análisis en el ciclo de vida de los sistemas orientados a objetos?
6. ¿Cómo es de manera general el desarrollo de software iterativo?
7. ¿En qué fase del ciclo de vida del software se realiza la codificación y se prueban unidades?
8. ¿Cómo es de manera general el desarrollo del software por “Prototipos”?
9. ¿Cuál es la peculiaridad del modelo en espiral y que lo hace diferente a los demás?
10. ¿Por qué son criticadas las metodologías ágiles o también llamadas livianas?

Examen de autoevaluación

Relaciona las columnas escribiendo dentro del paréntesis el número que una los conceptos con sus respectivas definiciones.

1. Es un ejemplo que sirve de norma, abarca un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento.	( )	a. Son las funciones y características que puede realizar y tener una clase.
2. Son los principales conceptos utilizados en la programación orientada a objetos.	( )	b. Son consideradas metodologías de desarrollo de software “Livianas”
3. Métodos y Propiedades.	( )	c. Paradigma
4. Son las fases que se llevan a cabo en el ciclo de vida del software.	( )	d. Con cada giro se construye un nuevo módulo del sistema, como una construcción sucesiva de versiones del software.
5. Consta de cuatro fases: determinación de objetivos, análisis de riesgos, desarrollo y pruebas, y planificación.	( )	e. Este modelo sirve como bloque de construcción para los demás modelos de ciclo de vida, y es a través de una secuencia simple de fases.
6. Mediante este modelo es factible reducir costos, reutilizar diseños, programas, módulos y datos.	( )	f. Objeto, clase, abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia y persistencia.
7. Modelo en espiral.	( )	g. Modelo de desarrollo de software Paralelo
8. Modelo en cascada.	( )	h. Análisis, diseño, implementación, mantenimiento y retiro.

9. Este modelo se refiere a que varios integrantes según el rol que desempeñan, pueden realizar actividades, procesos, cambios, mejoras, de manera simultánea.	( )	i. Modelo de desarrollo de software en Espiral.
10. Modelos de desarrollo de software ágil.	( )	j. Modelo de desarrollo de software por Prototipos.

TEMA 2. METODOLOGÍAS ORIENTADAS A OBJETOS

Objetivo particular

El alumno reconocerá las principales herramientas estándar de modelado orientadas a objetos: El Lenguaje Unificado de Modelado conocido como UML y la Definición de la integración para la modelización de las funciones nivel 4 conocida como IDEF4.

Temario detallado

2.1. UML

2.2. IDEF4

Introducción

Partiendo de que el objetivo de la ingeniería de software es desarrollar productos de calidad y que satisfagan las necesidades de los usuarios, se puede afirmar que cualquier empresa que puede desarrollar software de forma predecible y puntual con un uso eficiente y efectivo de los recursos, tanto humanos como de materiales, es una empresa sostenible. Y el modelado es una parte central de todas las actividades que conllevan a la producción de software de calidad, por lo que construimos modelos para:

- Comunicar la estructura deseada,
- Visualizar y controlar la arquitectura del sistema, y
- Comprender mejor el sistema; en la mayoría de los casos nos ayuda a descubrir oportunidades para simplificar o reutilizar elementos.

Por ello la importancia de conocer herramientas de modelado que nos permitan proporcionar desarrollos de calidad.

Las metodologías más aceptadas para el desarrollo Orientado a Objetos son aquellas que se basan en las técnicas de análisis y diseño; principalmente porque representan el mundo real formado por objetos, clases y sus relaciones, proporcionando un contexto amplio para comprender el comportamiento dinámico y funcional de los sistemas. Dentro de las herramientas que existen y proporcionan estas características se encuentran: UML e IDEF4.

Principales modelos propuestos

Existe un gran número de lenguajes de modelado de procesos, con sus respectivos modelos y herramientas asociadas, considerando la tradicional definición IPO (*Input Process Output*) y *Language Action* como las más relevantes. Se puede establecer una clasificación de los modelos de procesos en función de los formalismos que los sustentan.

Clasificación:	Descripción
Formalismo no ejecutables	Son aquellos en los que el modelado es informal y con el único fin de documentar o describir procesos. Como ejemplo existen : 1) Diagramas de Estado o Actividad de UML 2) Diagramas IDEF0 3) Diagramas de Flujo de Datos.
Basados en Redes	Son quizá los más utilizados debido a su fácil representación gráfica y comprensión por parte de usuarios no especializados, ya que asemejan formalismos no ejecutables pero con una semántica definida y sin ambigüedad. Como ejemplo: 1) Redes de Petri 2) Máquinas de Estado
Declarativos	Bajo este grupo englobamos aquellos formalismos que en vez de especificar imperativamente las actividades y sus dependencias, declaran restricciones sobre cómo debe realizarse el proceso. Generalmente se utiliza algún tipo de lógica (Kifer, 1996), permitiendo definir sistemas fácilmente modificables al poder añadir o quitar restricciones al modelo declarativo de proceso (Hull et al, 1999). En este grupo se incluirían también los modelos basados en reglas (Joeris y Herzog, 1998), disparos, o gestión del conocimiento en general.
Imperativos	Los procesos expresados en un lenguaje imperativo, ya sea específico de modelado de procesos o no, caerían en este grupo. Ejemplos de estos formalismos son los lenguajes de modelado de procesos (PML) como el propuesto por Keane (1994), el OCR propuesto en Alonso y Hagen, (1997) o extensiones a lenguajes clásicos como Ada o C.
Híbridos	Los grupos anteriores no son excluyentes. Pueden definirse sistemas en los que se haga uso de combinaciones de formalismos con el fin de

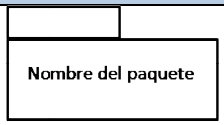
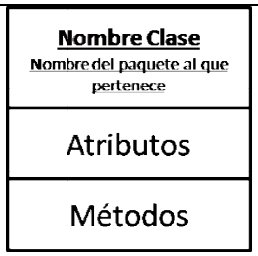
	aumentar su potencia (Kammer, 2000). Por ejemplo podría utilizarse algún tipo de red para modelar procesos, añadiendo algún método declarativo para expresar restricciones globales o formalismos no ejecutables como vistas para facilitar la comprensibilidad de los procesos.

Cuadro 2.1. Clasificación de los modelos de procesos (Martínez, Canós y García, 2001, pp. 18-19)

2.1. UML (Lenguaje Unificado de Modelado)

Este modelo fue creado por la necesidad de tener un único sistema para modelar y documentar sistemas de información y procesos de gestión, utilizando las técnicas de análisis y diseño orientado a objetos; con ello se unificó la semántica y las notaciones simbólicas aportando un estándar al mercado.

El lenguaje de modelado UML es un lenguaje gráfico que nos facilita visualizar, especificar, construir, documentar y describir el ciclo de vida completo del desarrollo orientado a objetos de un Modelo. Para desarrollar un modelo y representarlo con UML, es necesario conocer la representación de algunos conceptos:

Concepto	Descripción	Representación UML
Paquete	Permite organizar las clases de un modelo	
Clase	Es un conjunto de objetos que comparten estructura y comportamiento.	
Objeto	Es cualquier cosa, acerca de la cual almacenamos datos y métodos que manipulan y controlan dichos datos.	

Cuadro 2.2. Conceptos para desarrollar un modelo UML

UML incorpora nueve programas que permiten representar mediante diagramas un modelo desde diferentes perspectivas, como se muestra a continuación:

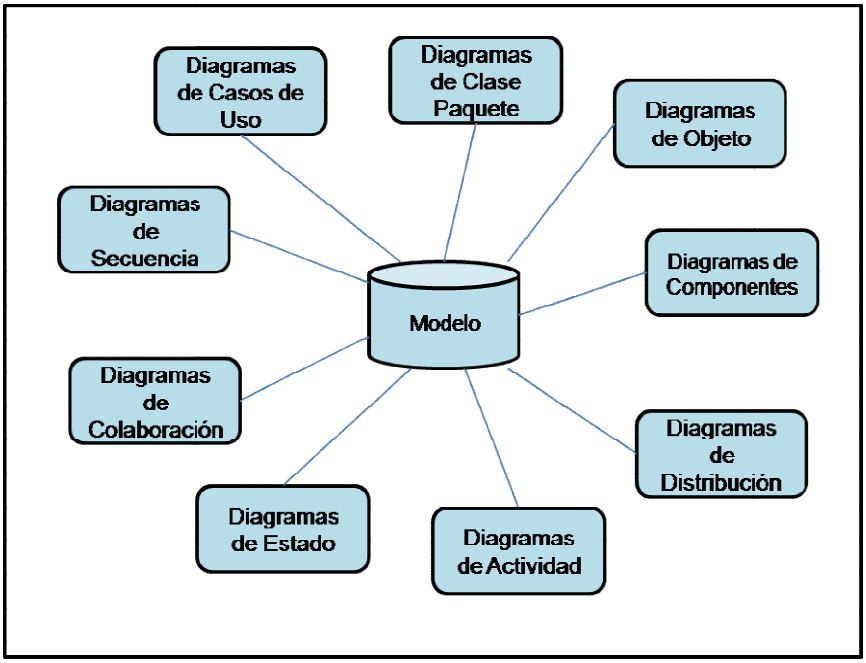


Figura 2.1. Los nueve diagramas que proporciona UML para representar un modelo mediante sus respectivos programas

Diagrama de	Representa
Clases	La vista estática de diseño y de proceso en términos de clases, relaciones, interfaces y colaboraciones.
Objetos	Los objetos y sus relaciones.
Actividades	El comportamiento de una operación en términos de acciones.
Caso de Uso	Las funciones del sistema visto por el usuario mediante un conjunto de casos de uso, actores y relaciones.
Colaboración	Una forma especial de los objetos, enlaces e interacciones entre ellos mediante el envío y recepción de mensajes.
Componentes	Los componentes físicos de una aplicación y sus relaciones.
Despliegue	Los despliegues de los componentes sobre los dispositivos materiales, nodos y relaciones.
Estados-Transiciones	El comportamiento de una clase en términos de estados, el estado de un objeto y las causas por las que puede cambiar de un estado a otro.
Secuencia	Temporalmente los objetos y sus interacciones.

Cuadro 2.3. Tipos de Diagramas que incorpora UML a través de sus programas

Algunos Ejemplos de estos diagramas son los siguientes:

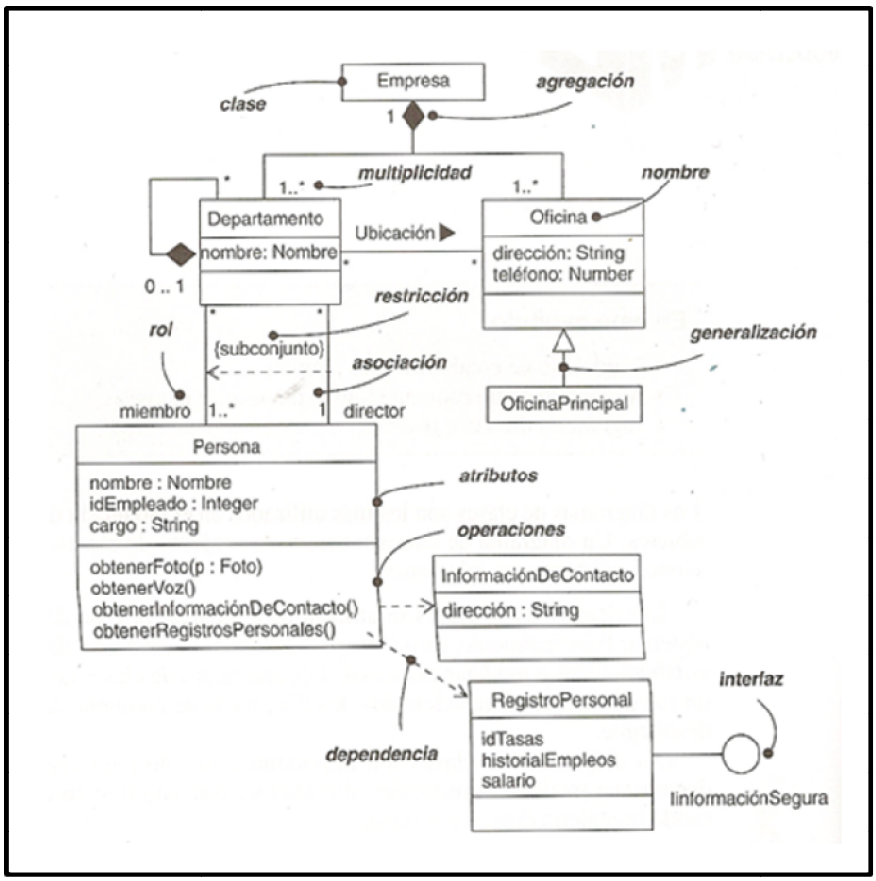


Figura 2.2. Diagrama de Clases
(Booch, y Rumbaugh 1999, p. 94)

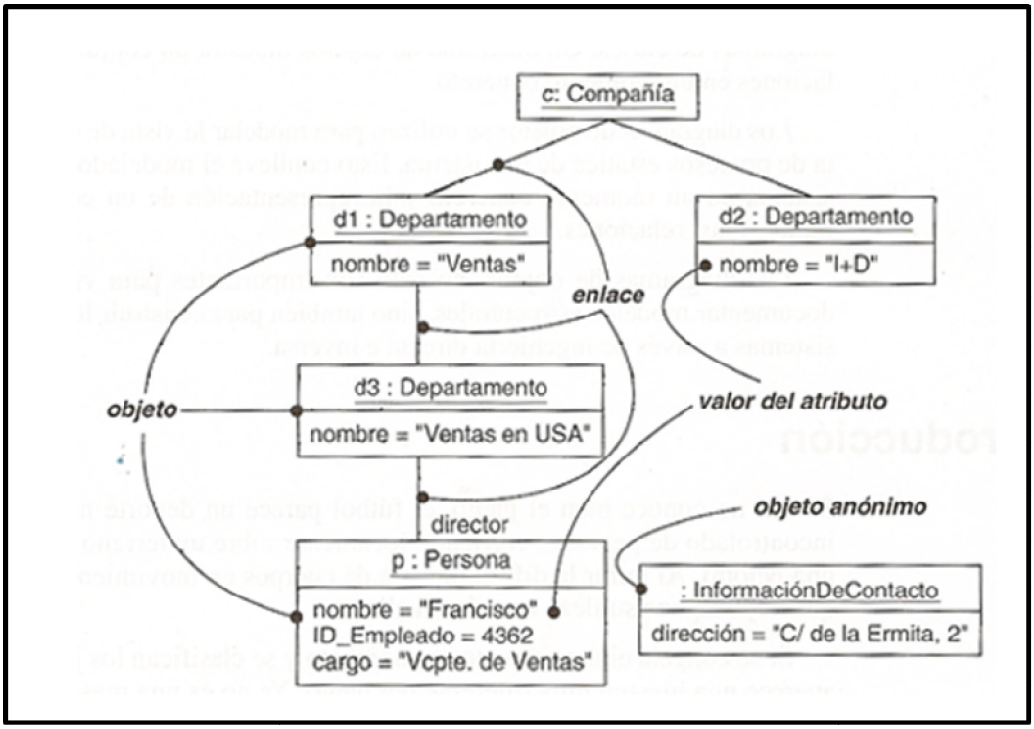


Figura 2.3. Diagrama de Objetos
(Booch-Rumbaugh 1999, p. 170)

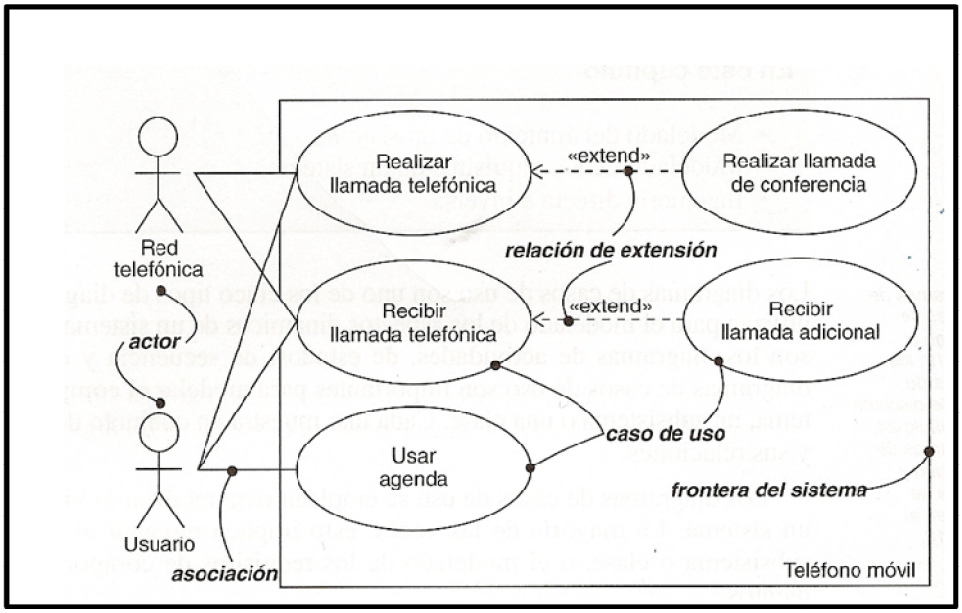


Figura 2.4. Diagrama de Casos de Uso
(Booch-Rumbaugh 1999, p. 204)

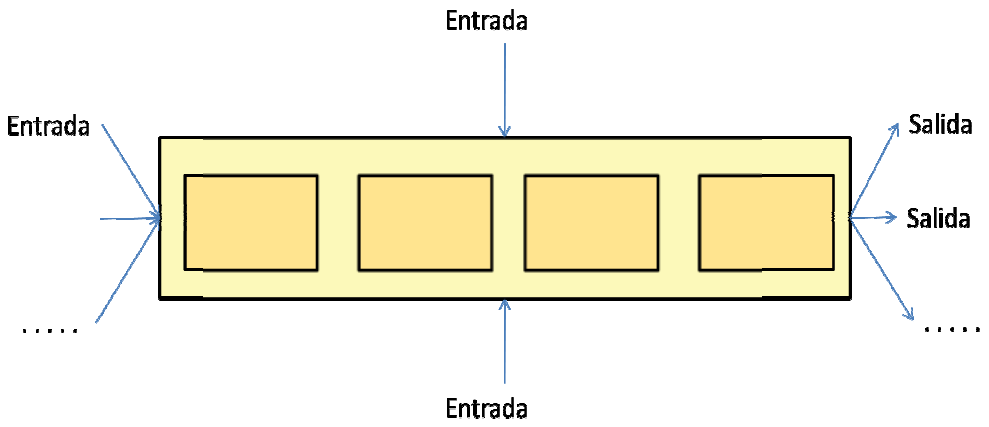
2.2. IDEF4 (Integration Definition For Function Modeling**)**

Definición de la integración para la modelización de las funciones Nivel 4

IDEF, toda una familia, consiste en un "conjunto de lenguajes de modelado dentro del ámbito de la ingeniería de sistemas, que cubren un amplio rango de necesidades, desde la captura y modelado de la información como son: IDEF0, IDEF1, IDEF1x, hasta el diseño de redes como IDEF14" (Chulvi y Sánchez, 2009, p. 1803).

A continuación muestro algunas características de IDEF0 hasta IDEF4:

IDEF0	Se trata de una representación global, de los procesos que desarrolla una organización en su conjunto.
IDEF1	Representa el proceso integral en su conjunto: Consiste en la representación global del proceso en su flujo por los diferentes proveedores de la organización y, por tanto, figurarán los distintos niveles asistenciales implicados en la atención.
IDEF2	Representa los subprocesos: Una vez diseñado el esquema gráfico del conjunto de actividades que componen el proceso (nivel 1), el segundo paso consiste en profundizar un poco más en estos grupos de actividades. Donde se trata de responder a la pregunta ¿qué hacemos? pero más detalladamente que en el nivel 1. La forma de representación de IDEF2 consiste, como norma general, en la presentación de una caja que recoge todas las actividades del subproceso de forma ligada y directamente relacionadas con los servicios facilitados a los usuarios.

	
IDEF3	Es un modelo de representación gráfica práctico en el que se desglosan las actividades de un proceso, a la vez que se muestra la persona que desarrolla la actividad. Es ya una representación gráfica *a* través del diagrama de flujos, y puede detallarse más mostrando, en la parte superior, los tiempos de ejecución o el lugar en el que se realiza la actividad o se entregará.
IDEF4	En él se realiza el diseño y modelado Orientado a Objetos.

Cuadro 2.4. Características de IDEF0 hasta IDEF4

"IDEF4 que se define como aquel modelo de ayuda en el diseño modular orientado a objetos, normalmente, de fácil manejo y reutilización. Apoya la transición de la aplicación y los requisitos para el diseño y la generación de código" (Chulvi y Sánchez, 2009, 8104). Además permite al diseñador hacer fácilmente un equilibrio entre la composición de una clase, la clase de herencia, la descomposición funcional, y un polimorfismo en el diseño.

IDEF4 es más que un gráfico de sintaxis, es un procedimiento que sigue los siguientes **pasos:**

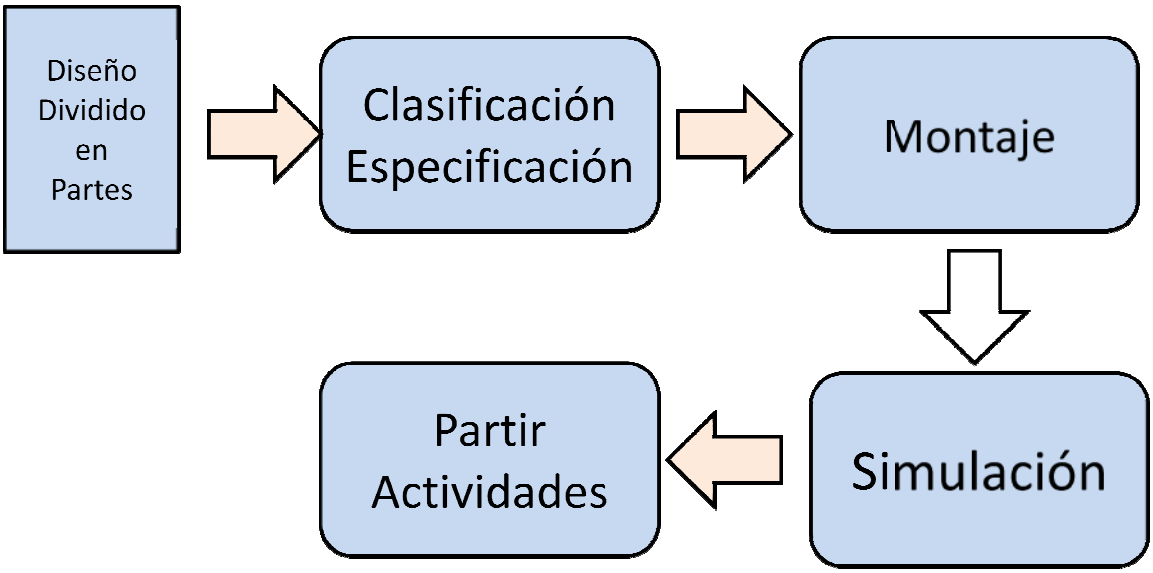


Figura 2.5. Pasos que sigue IDEF4

Primero, el diseño está dividido en objetos, para después clasificar/especificar las interfaces entre los objetos. Estos se especifican en el montaje de la actividad (dinámico, estático y modelos de comportamiento que detallan los diferentes aspectos de la interacción entre los objetos que se han desarrollado). Es importante la simulación de los mismos para poder variar los modelos existentes y simular hasta que el diseñador quede satisfecho.

Los conceptos con los que se define IDEF4 son:
dominios; características, artefactos y objetos; objetos instancia; clases, subclases y superclases; particiones; atributos; estados objeto; métodos; mensajes y polimorfismo; eventos; objetos ciclos de vida; clientes/servidores; relaciones y funciones; herencias; encapsulación y ocultación de la información. (Chulvi y Sánchez, 2009, 1804)

Organización del Modelo IDEF4

Conceptualmente IDEF4 modelo consta de 2 submodelos: La clase y el método submodelo, tal y como se observa en el siguiente diagrama:

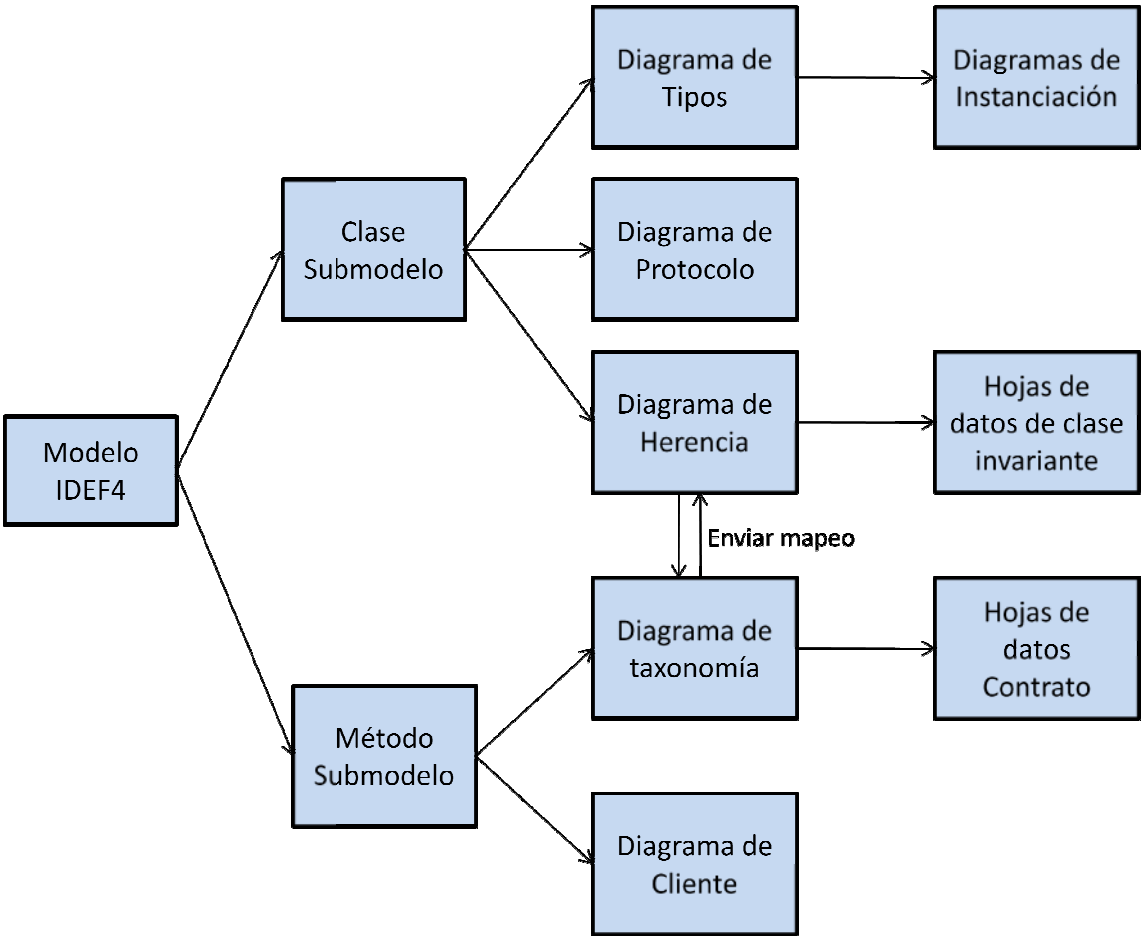


Figura 2.6: Organización del modelo IDEF4 (http://www.idef.com/IDEF4.html)

En el que podemos observar, que la clase submodelo se compone de:

1) Diagrama de Tipos
2) Diagrama de Protocolo
3) Diagramas de Herencia

y el Método submodelo que se compone de los siguientes tipos de diagramas:

1) Diagrama de Taxonomía
2) Diagrama de Cliente

Algunos Ejemplos de estos diagramas son los siguientes:

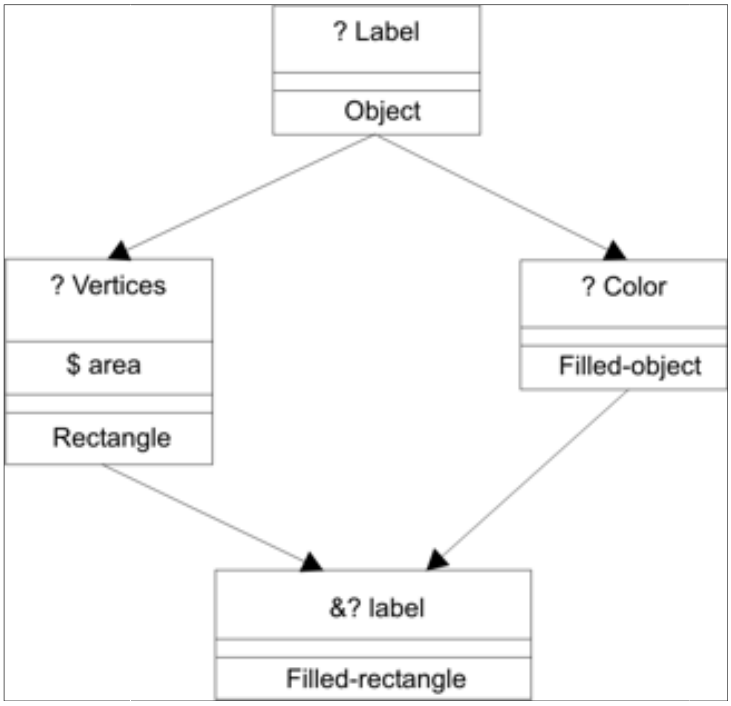


Figura 2.7. Diagrama de Herencia (http://www.idef.com/IDEF4.html)

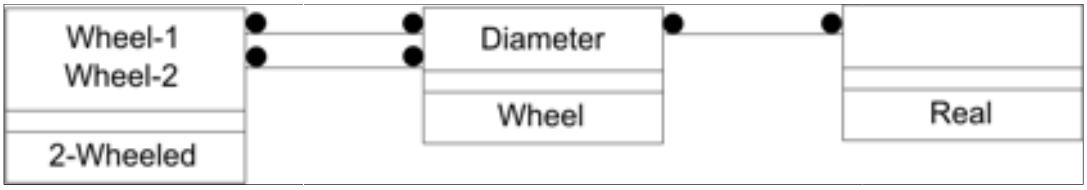


Figura 2.8. Diagrama de Tipo (http://www.idef.com/IDEF4.html)

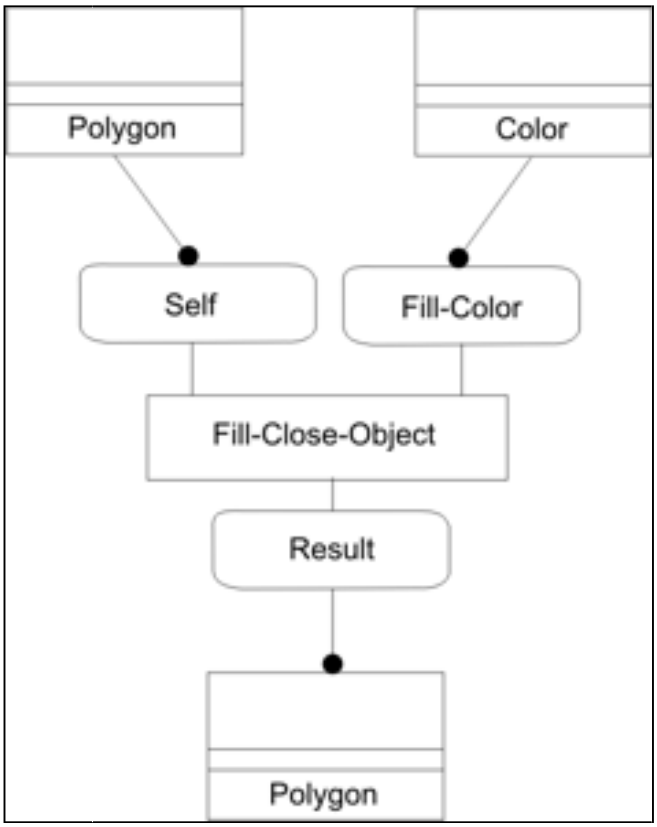


Figura 2.9. Diagrama de Protocolo (http://www.idef.com/IDEF4.html)

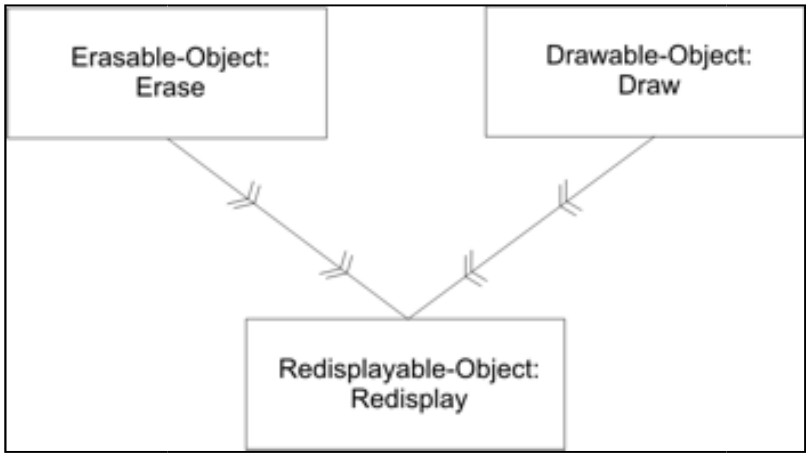


Figura 2.10. Diagrama de Clientes (http://www.idef.com/IDEF4.html)

Conclusiones

Existen diversas herramientas para modelar sistemas orientados a objetos, por lo que se debe conocerlas, así como sus terminologías para representar el conocimiento en función del campo en que nos desarrollemos realizando cierta actividad; No se trata pues de decir cuál es la mejor herramienta, sino de saber escoger la más adecuada cuando sea necesario.

En diversos trabajos se aprovecha esta adaptabilidad que tienen las herramientas, creando sus propias aproximaciones a estos lenguajes de modelado, aprovechando la simbología que ya ha sido estandarizada.

Bibliografía del tema 2

Booch Grady, James Rumbaugh, Ivar Jacobson. (1999). *El lenguaje Unificado de Modelado*, 2ª edición, Madrid, Addison Wessley.

IDEF4: http://www.idef.com/IDEF4.html

Chulvi, Vicente; María Sánchez Mora. (2009). *Modelo informal basado en idef4 para la representación de diseños basados en el esquema FBS,* XIII Congreso Internacional de Ingeniería de Proyectos, Badajoz, pp. 1802-1810, disponible en línea: http://aeipro.com/index.php/remository/congresos/congresos_badajoz2009/congresos_badajoz2009_08/MODELO-INFORMAL-BASADO-EN-IDEF4-PARA-LA-REPRESENTACION-DE-DISE%C3%91OS-BASADOS-EN-EL-ESQUEMA-FBS recuperado el 12/11/09

Martínez, Ángel, José Hilario Canós y Jesús Damián García-Consuegra. (Octubre 2010). Estudio del Estado del Arte en Modelado y Ejecución de Procesos Interorganizacionales, Albacete: Departamento de

Informática, Escuela Politécnica Superior, Universidad de Castilla-la Mancha, disponible en línea: http://www.info-ab.uclm.es/descargas/thecnicalreports/DIAB-01-06-21/diab-01-06-21.pdf

Actividades de Aprendizaje

A.2.1. Elabora un cuadro que contenga la clasificación más importante de los lenguajes de modelado, describiendo cada uno de ellos y mencionando al menos un ejemplo.

A.2.2. Representa en un diagrama a los 9 programas de que consta UML

A.2.3. Representa de manera gráfica la organización del modelo IDEF4

Cuestionario de Autoevaluación

1. Describe por qué es importante el uso de herramientas de modelado de sistemas orientados a objetos
2. Dentro de la clasificación de los lenguajes de modelado, IDEF0 en qué clasificación se encuentra y por qué
3. ¿Qué elementos incluye UML para definirse como lenguaje estándar de modelado de sistemas orientados a objetos en el mercado?
4. Menciona qué elementos básicos se deben conocer para desarrollar un modelo en UML
5. ¿Cuáles son los 9 diagramas que UML proporciona para el modelado de sistemas?
6. ¿Qué es lo que representan los siguientes diagramas en UML?
   Diagrama de Casos de Uso, Diagramas de Clases, Diagramas de Estados o transiciones.
7. ¿A qué nos referimos cuando hablamos de la familia IDEF?
8. Define IDEF4.
9. ¿Cuáles son los cuatro pasos que sigue IDEF4 para modelar sistemas?
10. Dentro de la definición de IDEF4 menciona de qué submodelos consta.

Examen de Autoevaluación

Relaciona las columnas escribiendo dentro del paréntesis el número que una los conceptos con sus respectivas definiciones.

1. Formalismos no ejecutables	( )	a. Diagramas de Casos de Uso
2. Permite organizar las clases de un modelo.	( )	b. Clases, Métodos, diagrama de protocolos, diagrama de taxonomía, diagrama de cliente
3. Pasos de IDEF4	( )	c. Paquete
4. Las funciones del sistema son vistos por el usuario mediante un conjunto de casos, actores y relaciones.	( )	d. IDEF0
5. Submodelos IDEF4	( )	e. Diagramas de Secuencia
6. Basados en Redes	( )	f. Diagramas de Colaboración
7. IDEF4	( )	g. Son los más utilizados debido a su fácil representación gráfica y compresión por parte de los usuarios no especializados.
8. Se trata de una representación global, de los procesos que desarrolla una organización en su conjunto.	( )	h. Son aquellos que el modelado es informal y su único fin es el de documentar o describir procesos.
9. Muestra temporalmente los objetos y sus interacciones.	( )	i. Clasificación especificación, montaje, simulación, partir actividades.
10. Representa una forma especial de los objetos, enlaces e interacciones entre ellos mediante un envío y recepción de mensajes.	( )	j. Definición de la integración para la modelización de las funciones nivel cuatro.

TEMA 3. PLANEACIÓN Y ELABORACIÓN

Objetivo particular

Al finalizar el tema el alumno identificará dentro del análisis de requerimientos los casos de uso, las interfaces con usuarios, otros sistemas, las entradas y salidas del sistema, así como la descripción de los procesos de negocio que realiza una empresa.

Temario detallado

3.1 Análisis de los requerimientos (Casos de uso)
- 3.1.1 Las interfaces con los usuarios
- 3.1.2 Las interfaces con otros sistemas
- 3.1.3 Las entradas y las salidas del sistema

3.2 Descripción de los procesos (procesos de negocios, lo que debe hacer una empresa)

Introducción

La elaboración y planificación de sistemas de información intentan identificar y establecer prioridades acerca de las tecnologías y las aplicaciones susceptibles de reportar el máximo beneficio a una empresa, es decir, la elaboración y planificación de sistemas de información indica la dirección correcta que se debe seguir en el desarrollo de un sistema, la forma correcta de proceder, la buena administración, los criterios de selección, etc.

3.1 Análisis de los requerimientos (Casos de uso)

Los requerimientos, también conocidos como requisitos, son una condición, capacidad o característica del software que necesita el usuario para solucionar un problema o bien para alcanzar un objetivo. El proceso Unificado Racional (RUP) fomenta un conjunto de buenas prácticas, una de las cuales es la gestión de los requerimientos.

Clasificación de los requerimientos

Estos pueden dividirse en requerimientos implícitos y explícitos y en funcionales y no funcionales, estos últimos es la clasificación que utiliza RUP.

Implícitos	Son requerimientos declarados por el cliente, usuario o identificados por el analista de sistemas.
Explícitos	Son aquellos requerimientos que se dan por obvios, tanto por el cliente como por el analista. Por ejemplo, un módulo de mantenimiento a catálogos, la alta, baja de información, etc.

Funcionales	Definen las funciones que el sistema realizará y describe las transformaciones que será necesario realizar para producir salidas.
No Funcionales	Tienen que ver con las características que podrían limitar de alguna forma al sistema, pudiendo ser estás: el rendimiento, las interfaces de usuario, la fiabilidad, el mantenimiento, etc. (ver, Zuñiga, 2009)

Cuadro 3.1. Clasificación de Requerimientos según RUP

Algunos de estos requisitos se denominan colectivamente **atributos de calidad** o **requisitos de calidad**, para obtener un software de calidad. Existen principalmente dos modelos de calidad de software generados por Hewlett-Packard **FURPS**, del acrónimo de las palabras **F**unctional, **U**sability, **R**eliability, **P**erformance, **S**upportability, y **FURPS+**.

Sigla	Tipo de Requerimiento		Descripción
F	**F**unctional	Funcional	Se refiere a las características, capacidades y seguridad del SW.
U	**U**sability	Facilidad de Uso	Se refiere a los factores humanos, ayudas y a la documentación
R	**R**eliability	Fiabilidad	Se refiere a la frecuencia de fallos, a la capacidad de recuperación de un fallo o al grado de previsión.
P	**P**erformance	Rendimiento	Se refiere a los tiempos de respuesta, a la productividad, a la precisión, a la disponibilidad, y al uso de los recursos.
S	**S**upportability	Soporte	Se refiere a la adaptabilidad, a la facilidad de mantenimiento, a la internacionalización y a la configuración.
+	Plus	Implementación	Se refiere a la limitación de los recursos, lenguajes, herramientas, hardware, etc.
		Interfaz	Se refiere a las restricciones impuestas para la interacción con otros sistemas externos.
		Operaciones	Se refiere a la gestión del sistema en su puesta en marcha, y proporciona algunas pautas administrativas.
		Empaquetamiento	Se refiere a la forma de distribución del Software
		Legales	Se refiere a la Licencia, derechos de autor, etc.

Cuadro 3.2. FURPS

El uso de estos factores de calidad como una lista permite comprobar que se cubran los requerimientos, de manera que con ello se reduzca el riesgo de no considerar alguna faceta importante en el nuevo sistema.

Técnicas de recopilación de requerimientos

En esta sección es importante mencionar que existen las siguientes técnicas de recopilación de requerimientos, que se utilizarán cuando el analista lo considere necesario cuando requiera el apoyo en el establecimiento de requisitos.

Técnica	Descripción
Entrevistas	Se basan en una serie de preguntas que se realiza a las personas que representen a todos los sectores críticos de la organización y que estén relacionadas con el sistema, y guiarán a los diseñadores e ingenieros de software en cómo serán las especificaciones que requieren su productos.
Talleres	Son discusiones que se llevan a cabo por el personal interesado en un ambiente controlado, con el propósito de analizar y definir los requisitos que durante las entrevistas no son detectados; en particular las implicaciones cruzadas.
Formularios	En lugar de una entrevista pueden llenar formularios, indicando los requerimientos.
Prototipos	Son aplicaciones que se realizan durante el desarrollo y reflejan el funcionamiento real de un proceso, y sirven para visualizar de manera global (sin detalles) cómo podría ser el sistema final. Con la elaboración de prototipos, el usuario puede dar opiniones acerca del proceso y corregir posibles desviaciones en el desarrollo antes de concluir la aplicación final.
Cuestionarios	Es un conjunto de preguntas que se deben contestar por escrito por una determinada población. Según el contenido de los cuestionarios podemos clasificarlos en Abiertos y cerrados, esto se refiere a que las respuestas están delimitadas o no.

Cuadro 3.3. Técnicas de recopilación de requerimientos

RUP (Rational Unified Process **– Proceso Unificado Racional)**

> RUP es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos y roles. Esto se refiere a los productos tangibles como: el modelo de casos de uso, el código fuente, etc. y al papel que desempeña una persona en un determinado momento, una persona puede desempeñar varios roles a lo largo de un proceso.[3]

Debido a que RUP es un proceso dinámico, se presenta en cuatro fases principalmente que son dividas a su vez en iteraciones. De manera gráfica se representa en dos ejes, el eje horizontal representa precisamente a las cuatro fases: Iniciación, Elaboración, Construcción y Transición que son divididas en iteraciones y en el eje vertical representamos las disciplinas principales del Proceso (Modelado de negocio, requisitos, análisis y diseño, implementación, pruebas, despliegue,) y Soporte (Configuraciones y gestión del cambio, Gestión del proyecto y Entorno) las cuales agrupan lógicamente las actividades por su naturaleza. Tal y como lo podemos observar en la siguiente figura:

[3] Wikipedia: "Proceso Unificado Racional", actualizado el 7/11/09, disponible en línea: http://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational, recuperado el 12/11/09. Consultado el 11/12/2009.

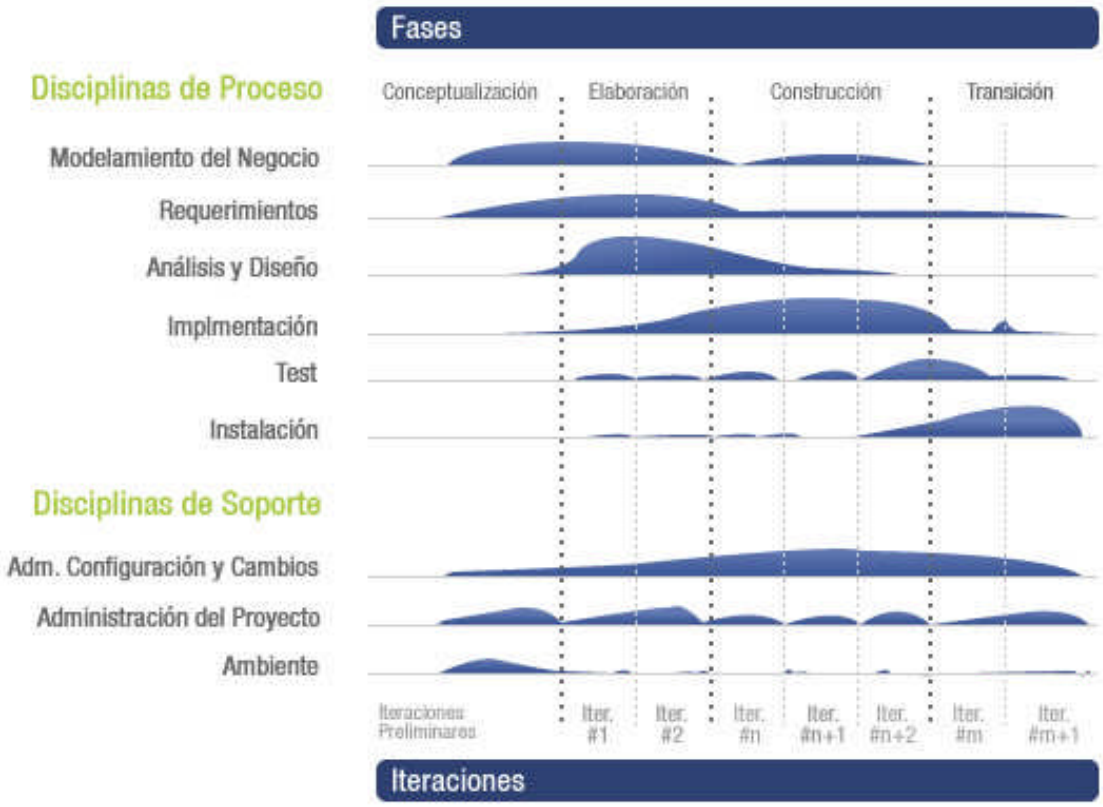


Figura 3.1. Estructura de RUP[4]

[4] Wikipedia: 'RUP' 26/06/08: http://es.wikipedia.org/wiki/Archivo:Rup_espanol.gif, recuperado el 27/05/10.

RUP en cada una de sus fases realiza una serie de artefactos o productos que sirven para comprender mejor tanto el análisis como el diseño del sistema. Estos productos son los siguientes:

Fase	Artefactos o Productos
Conceptualización	-Documento Visión -Especificación de Requerimientos
Elaboración	-Diagramas de Actividades -Diagramas de Caso de Uso
Construcción	**Vista Lógica** - Diagramas de Clases - Modelo Entidad-Relación
	Vista de Implementación - Diagramas de Secuencia - Diagramas de Estados - Diagramas de Colaboración
	Vista Conceptual - Modelo de dominio
	Vista Física - Mapa de Comportamiento a nivel Hardware

Cuadro 3.4. Productos RUP en cada Fase

Los requerimientos del proyecto en la parte de la vista deben estudiarse y recopilarse en una lista con sus características principales. Los otros requerimientos deben reflejarse en los diferentes casos de uso que están relacionados en el proceso global. El resultado de los requerimientos se verá reflejado en los diferentes artefactos o productos que hemos definido en el cuadro anterior.

Casos de Uso

Ningún sistema se encuentra aislado. Cualquier sistema interactúa con actores humanos o mecánicos que lo utilizan con algún objetivo y que esperan que el sistema funcione de manera predecible.

Un caso de uso especifica el comportamiento del sistema o de una parte del mismo y es una descripción de un conjunto de secuencias de acciones que ejecuta el sistema para producir un resultado observable de valor para un actor. Gráficamente un caso de uso se representa con una elipse y debe contar con un nombre, en la práctica los nombres de los casos de uso son expresiones verbales que describen algún comportamiento, por ejemplo: Hacer pedido, Validar usuario, etc.

Constituyen la base para establecer el comportamiento, la verificación *y* validación del sistema; mientras que la arquitectura es la base para la conceptualización, construcción, administración, evolución y la posibilidad de ofrecer una administración de versiones y mejoras increméntales de dicho sistema.

Figura 3.2. Un caso de uso reúne todos los elementos del software

Un caso de uso debe representar una interacción entre el software y el actor o actores. Los casos de uso nos ayudan a describir qué es lo que el sistema debe hacer, desde el punto de vista del usuario y no el cómo se debe hacer.

Pasos para desarrollar un caso de uso

Identificar Actores.- Los actores o mejor dicho los roles son los que uno o más usuarios del sistema llevan a cabo en algún momento del tiempo. También pueden ser otros sistemas con los que el sistema en modelado tiene interacción.

Identificar Metas o Roles.- Se refiere a los objetivos de manera general o responsabilidades, es decir todos los actores en el entorno por modelar tienen sus propias metas u objetivos, o en su defecto responsabilidades o acciones que desean realizar u obtener del sistema.

Obtener o identificar los casos de uso a partir de metas.- Las metas son muy importantes ya que a partir de su definición pasamos a realizarlas y se convierten en los casos de uso.

Especificar cada caso de uso.- Una vez identificados se especifican uno por uno, y es probable que durante ese proceso algunos casos de uso desaparezcan o se fusionen con otro, debido a ambigüedades detectadas o por olvido durante el proceso.

Es importante considerar que el modelado se trata precisamente de detectar esas ambigüedades, y que no quedará al cien por ciento a la primera, si no que permitirá hacerlo una y otra vez hasta lograr el diseño correcto, por eso hay que hacerlo en iteraciones o ciclos. (Softtlan, 2007)

Casos de uso: Plantilla de Especificación

La especificación de los casos de uso tiene varias partes o elementos, a lo que se le llama plantilla de Especificación la cual puede contener cualquiera de los siguientes elementos:

Elemento	Descripción
id	Clave o número de control del caso de uso
Nombre	Nombre del caso de uso
Autor	Nombre de quien lo elaboró
Fecha	Fecha de elaboración
Descripción	Detalle de lo que este caso de uso resuelve dependiendo de su principal objetivo
Actores	En esta sección se especifica a el actor o actores principales y a los secundarios o auxiliares que participan en este caso de uso. Se puede detallar su nombre y en qué otros casos de uso participa (si así lo desea).
Pre-Condiciones	Son las reglas o condiciones que deben cumplirse antes de que sea iniciado el caso de uso. Por ejemplo, que el usuario este dado de alta y conectado.
Post-Condiciones	Son las condiciones que se deben cumplir cuando se termine el caso de uso.
Flujo Principal	Es la secuencia de pasos del flujo principal, se puede utilizar solo texto enumerando el paso realizado por él o los actores.
Variaciones	Aquí se listan los pasos
Flujo Alternativo	Es una secuencia de pasos que son adicionales al flujo principal que nos indicará qué es lo que hace el sistema en los casos menos frecuentes e inesperados.

Cuadro 3.5. Elementos de una plantilla de Especificación de Casos de Uso

Por Ejemplo: ésta podría ser la plantilla del caso de uso *Escribir un mensaje en un foro de discusión.*

Nombre:	Crear mensaje
Autor:	Ing. Jesús Parra Chávez
Fecha:	25 de Octubre de 2009
Descripción: Permite Crear un mensaje dentro de un foro de discusión	
Actores: Usuario de Internet que ya esté conectado	
Precondiciones: El usuario debe haberse dado de alta	
Flujo Normal: 1.- El actor pulsa sobre el botón para crear un nuevo mensaje 2.- El sistema muestra una caja de texto para introducir el título del mensaje y una zona de mayor tamaño para introducir el cuerpo del mensaje 3.- El actor introduce el título del mensaje y el cuerpo del mismo 4.- El sistema comprueba la validez de los datos y lo almacena	
Flujo Alternativo: El sistema comprueba la validez de los datos; si los datos no son correctos, se avisa al usuario de ello permitiéndole que los corrija	
Poscondiciones: El mensaje ha sido almacenado correctamente en el sistema.	

Cuadro 3.6. Ejemplo Plantilla de Especificación del caso de uso
E*scribir un mensaje en un foro de discusión.*

Ésta podría ser la plantilla del caso de uso: *Alquilar una película*

Nombre:	Crear mensaje
Autor:	Ing. Jesús Parra Chávez
Fecha:	25 de Octubre de 2009.
Descripción: Permite Crear un mensaje dentro de un foro de discusión	
Actores: Usuario de Internet que ya esté conectado.	
Precondiciones: El usuario debe haberse dado de alta y conectado	
Flujo Normal	
Actor	Sistema
1. El cliente llega al mostrador con videos para alquilar (excepcionalmente videojuegos).	
2. El cliente presenta su tarjeta de socio y el empleado introduce su número de identificación en el sistema.	3. Presentar información sobre el cliente y el estado de alquileres. (normalmente sin ninguna película alquilada, y sin penalizaciones pendientes)
4. Para cada videojuego, el empleado graba en el sistema el número de identificación en el sistema.	5. Presenta una lista de los títulos alquilados, fechas de devolución, precio total del alquiler y cargos por retraso en la devolución.
6. El empleado informa al cliente la cantidad a pagar y le pide el pago.	
7. El cliente paga en efectivo o a crédito.	
8. El empleado registra y graba el pago en el sistema.	
	9. Autoriza el pago a crédito
	10. Genera el recibo de pago
11. El empleado entrega el recibo al cliente, que se va con los videojuegos / películas alquiladas.	
Flujo Alternativo:	
7. El cliente no tiene suficiente dinero en efectivo. Sugerir pago a crédito, cancelación de la transacción o eliminar artículos hasta que la cantidad resultante pueda ser pagada.	
8. El cliente tiene recargos por retraso sin pagar y no quiere abonarlos. Antes de hacer nuevos alquileres deben pagarse los recargos: paga todo o se cancela transacción.	

9. Autorización de pago a crédito denegada, por crédito insuficiente o por fallo del servicio de autorización: pedir pago en efectivo.

Cuadro 3.7. Ejemplo Plantilla de Especificación del caso de uso Alquilar una película

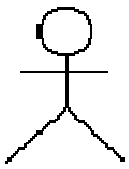
Diagramas de Casos de Uso

En el momento en que se cuenta con un gran número de casos de uso en un proyecto, no resulta fácil situarlos y relacionarlos, es entonces cuando se requiere tener una visión general del asunto, y representarlos mediante diagramas.

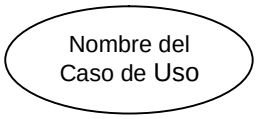
En UML los diagramas de casos de uso son uno de los cinco tipos de diagramas que se utilizan para el modelado de aspectos dinámicos de un sistema como observamos antes; veamos a continuación cómo se deben elaborar.

Elementos de un Diagrama de caso de uso

Actor. Se define como una entidad externa que participa en la secuencia de eventos del caso de uso, puede ser por ejemplo: una persona, un conjunto de personas, un sistema hardware, un reloj, un sistema software, etc. es representado por:



Casos de Uso. Es la especificación del caso de uso. Y se representa por una elipse que contiene el nombre del caso de uso.



Límites del Sistema. Se representa mediante un rectángulo que indica los límites de operación del sistema.

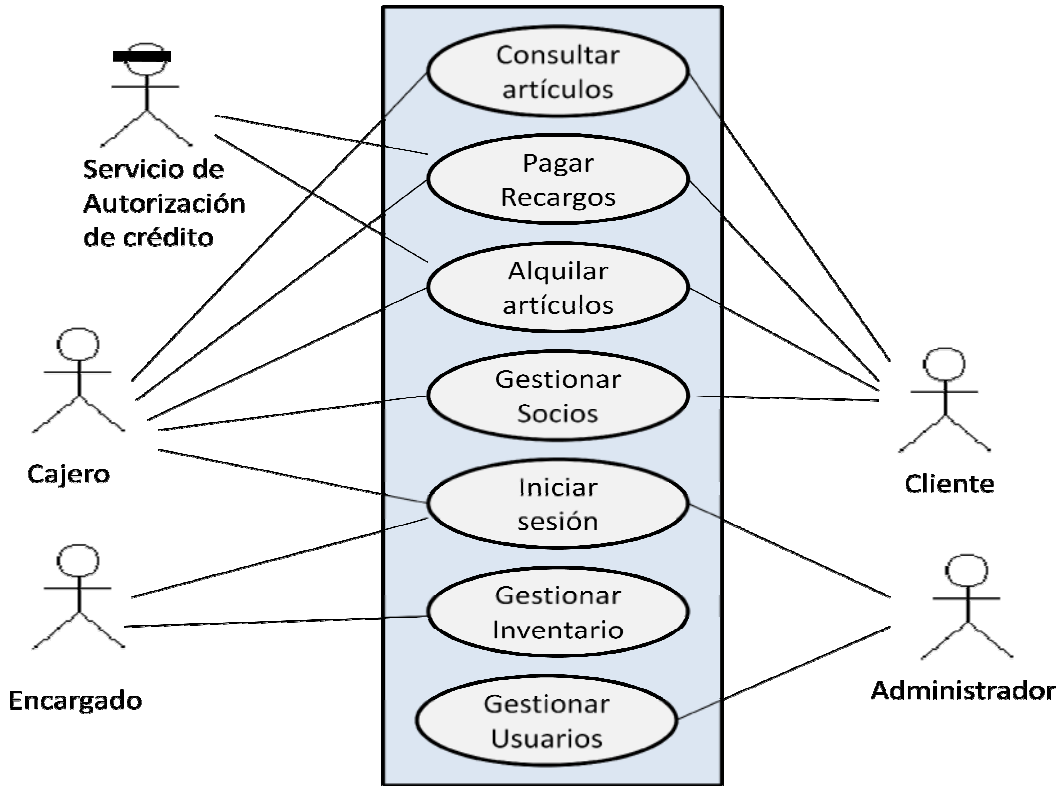


Figura 3.3. Elementos de un caso de uso

Relaciones de Dependencia, generalización, especialización y asociación
Se representan mediante líneas que unen casos de uso con actores.

Estas relaciones que existen entre los actores, pueden ser de generalización, cuando un actor es más general que otro, o si un caso de uso es una especialización de otro caso de uso. Otras relaciones que pueden darse son de Asociación, que pueden darse entre actores y casos de uso, en donde se ve la participación del actor en el caso de uso.

O bien relaciones con carácter de extensión, de inclusión o de generalización, que pueden darse entre diferentes casos de uso. Por ejemplo: el caso de uso inicial <<incluye>> el comportamiento del caso de uso final también llamados sub-caso, o El caso de uso final se puede <<extender>> con el comportamiento del caso de uso inicial en un punto concreto del primero. Tal y como lo podemos observar en el siguiente ejemplo:

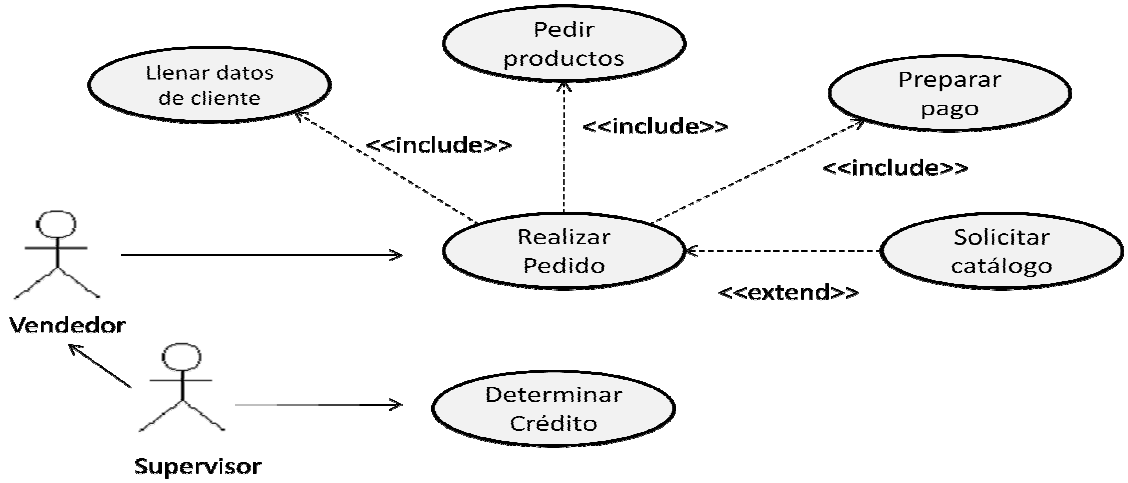


Figura 3.4. Relaciones de asociación

Interfaces

La interfaz es una abstracción completa de la implementación de una clase. La interfaz no define la implementación. Una clase puede implementar cualquier número de interfaces.

Una interfaz es una colección de operaciones que especifican un servicio de una clase o componente.

- Por lo tanto, una interfaz describe el comportamiento visible externamente de ese elemento.
- Una interfaz puede representar el comportamiento completo de una clase o componente o sólo una parte de este comportamiento.
- Una interfaz define un conjunto de especificaciones de operaciones (o

sea, sus signaturas), pero nunca sus implementaciones.

- Una interfaz raramente se encuentra asilada, más bien, suele estar conectada a la clase o componente que la realiza.

Algunos consejos para la elaboración de casos de uso

Una vez que hemos aprendido cómo es el proceso de elaboración y representación gráfica de los casos de uso, es conveniente tomar en cuenta las siguientes recomendaciones:

- Debe existir una comunicación real
- No complicar las cosas
- Tener en cuenta siempre a los interesados
- Hay que escribir el caso de uso
- Deben revisarse cuidadosamente con el usuario
- Deben escribir la integración del actor y el software
- Se pueden expresar por escrito los requisitos funcionales y no funcionales
- Expresar en los casos de uso el funcionamiento del sistema como un TODO (no de sus partes)
- Se puede priorizar a escala del 1 al 10 para desarrollar el sistema incrementalmente.
- Comienza el nombre de los eventos con un verbo, leer, escribir, guardar, almacenar, comprar, imprimir, etc.
- Comienza con la frase que debe especificar <Actor> lleva a cabo <evento>
- Todos los sistemas tienen un caso de uso Poner en marcha y Suspender

3.1.1 Las interfaces con los usuarios

Las interfaces básicas de usuario son aplicaciones que incluyen: menús, ventanas, teclado, ratón, etc. es decir, todos los canales por los cuales se permite la comunicación entre la computadora y el usuario. Por ello, una vez que se cuenta con todo un análisis previo debidamente documentado, se procede a la

elaboración de una interfaz para los usuarios, es decir la realización de un prototipo del sistema y se presenta al usuario.

Recordemos que un prototipo es una aplicación que refleja el funcionamiento real y sirve para visualizar de manera global cómo será el sistema final, además sirve para que el usuario dé opiniones y se corrijan posibles desviaciones.

Pero para ello, se debe realizar el diseño lógico de la interfaz de usuario así como un diseño físico.

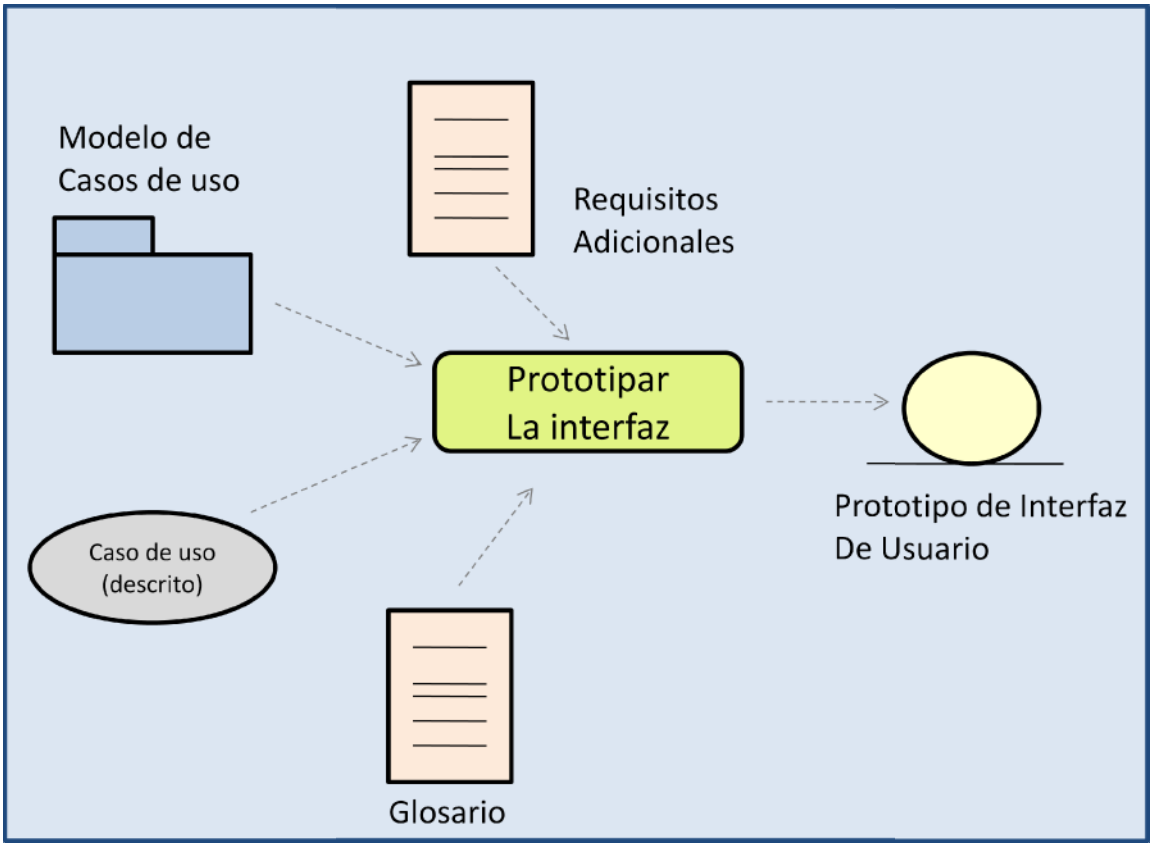


Figura 3.5. Elaboración de un prototipo de Interfaz de Usuario

Diseño lógico de la Interfaz de Usuario ó interfaz de Software

En el diseño lógico se decide qué es lo que se necesita de las interfaces de usuario para realizar los casos de uso para cada actor. El diseñador de interfaces

es el encargado de recorrer todos y cada uno de los casos de uso a que puede acceder un actor, e identificará de cada uno los elementos apropiados de la interfaz para cada caso de uso, así como las relaciones que hay entre estos, la apariencia de los elementos, el modo de manipularlos, las acciones que puede invocar, la información que debe proporcionar tanto el actor como el sistema, parámetros de entrada y parámetros de salida.

Diseño Físico de la Interfaz de Usuario ó interfaz de Hardware

En el diseño físico se desarrollan diferentes prototipos que muestran cómo pueden utilizar el sistema los actores para ejecutar los diferentes casos de uso definidos. Como resultado de esta actividad tendremos un conjunto de esquemas de interfaces de usuario y prototipos de interfaces que definirán la apariencia de esas interfaces cara a los actores más importantes de nuestros casos de uso.

Para lograr el diseño, se realizan esquemas de configuración y bosquejos para combinar los elementos de interfaces, como son: carpetas, ventanas, herramientas y controles. Se construyen finalmente prototipos ejecutables que permitan verificar a que cada actor pueda ejecutar el caso de uso que necesita, asegurando que le sea agradable y se trabaje de forma consistente.

3.1.2. Las interfaces con otros sistemas

Existen sistemas que para su funcionamiento requieren de la salida de datos que generan sistemas externos y que para obtenerlos es necesario proporcionarles entradas, o bien en sentido contrario, estos envíos se realizan mediante transacciones a través de cierta comunicación. En este caso es necesario diseñar, al igual que la interfaz de usuario, la interfaz con estos otros sistemas externos, estableciendo de inicio los diferentes tipos de entradas y salidas de datos, la especificación de cómo van a llevarse a cabo estas entradas, quiénes van a realizarlas, la seguridad de los datos, qué información se proporcionará, el tipo de

almacenamiento, tipo de comunicación, velocidad de respuesta, validación de respuestas en ambos sentidos, etc.

En la siguiente figura observamos el diálogo que existirá entre la interfaz del sistema con otro sistema de asignación. En la interfaz se muestra la petición de recursos que realiza nuestro sistema al sistema de asignación, mismo que devolverá la asignación solicitada, siempre y cuando esté disponible para realizarlo, o la negación a la petición, observa que el sistema de asignación libera los recursos que había dejado pendientes y los deja disponibles para futuras peticiones.

Figura 3.6. Interface con el sistema de Asignación
(Barranco, 2001: 322-24)

3.1.3 Las entradas y las salidas del sistema

Las entradas al sistema se refieren a los requisitos que necesita el sistema para empezar el procesamiento de la información. Como por ejemplo, datos como

usuario, NIP (número de identificación personal), una búsqueda, datos personales, etc.

Las salidas del sistema se refieren al resultado generado a partir de esos datos introducidos por el usuario en el inicio del caso de uso. Ejemplos de esto son informes, reportes, alertas, facturas, consultas en pantalla, etc.

Las entradas y salidas son mejor comprendidas con el típico ejemplo de un sistema, el cual tiene como actividades: las entradas, los procesos, las salidas, la retroalimentación y juntas trabajando bajo un mismo ambiente.

Figura 3.7. Actividades de un sistema de información

3.2. Descripción de los procesos (procesos de negocios, lo que debe hacer una empresa)

Un proceso de negocio es una actividad o conjunto de actividades que realiza una empresa para satisfacer las necesidades y expectativas de un cliente.

"Cada proceso de negocio tiene sus entradas, funciones y salidas. Las entradas son requisitos que deben tenerse antes de que una función pueda ser aplicada. Cuando una función es aplicada a las entradas de un método, tendremos ciertas salidas resultantes."[5] Con la implementación de los Casos de uso en los procesos de negocio de las empresas es más sencillo identificar las fallas que ocurren en dichos procesos, fallas que cuestan muy caro a la empresa, como prestigio, funcionalidad, clientes y dinero.

> Después de determinar los procesos del negocio de la organización bajo estudio, y de describir sus flujos de trabajo mediante diagramas de actividad, los casos de uso son identificados y estructurados a partir de las actividades de cada proceso, mientras que los conceptos que aparecen en el modelo conceptual se obtienen a partir de los datos que fluyen entre las actividades. Además, las reglas del negocio son identificadas e incluidas en un glosario, como parte de la especificación de datos y actividades. Un aspecto importante es el hecho de que el modelado conceptual y el de casos de uso deben realizarse en paralelo, haciendo más fácil la identificación y especificación de casos de uso adecuados. Tanto el modelado de casos de uso como el modelado conceptual forman parte de la fase de análisis de requisitos de un modelo de proceso completo. (García, 2007)

[5] Wikipedia: "Proceso de negocio", actualizado el 30/03/10, disponible en línea: http://es.wikipedia.org/wiki/Proceso_de_negocio, recuperado el 30/04/10.

Conclusiones

La planeación y elaboración dentro de la programación orientada a objetos destaca por su importancia, ya que en ella se realiza un análisis profundo de los requerimientos, y se plasma el qué debe hacer nuestro sistema en una visión global mediante los casos de uso, la definición de interfaces y la definición de entradas y salidas.

Esta visión global debe mostrar claramente las actividades que deben realizarse en el sistema sin desviaciones, vistas como una caja negra que tiene relaciones con los usuarios de manera interna, y debe mostrar las posibles relaciones con sistemas externos, las entradas y salidas necesarias, así como las interfaces que se necesitan para que sea posible su funcionamiento. Como resultado de estas actividades tendremos por completo el **qué** hará nuestro sistema.

Bibliografía del tema 3

Booch Grady; James Rumbaugh; Ivar Jacobson, (1999), *El lenguaje Unificado de Modelado*, 2ª edición, Madrid, Addison Wessley.

Barranco de Areba, Jesús, (2001), *Metodología del análisis estructurado de sistemas*. 2ª ed., Madrid, Universidad Pontificia de Comillas.

García Molina, Jesús, (2007), "De los procesos del negocio a los casos de uso", Universidad de Murcia; en *Técnica Administrativa*, Vol. 6, N° 4, octubre/diciembre, Bs. As. Disponible en línea: http://www.cyta.com.ar/ta0604/v6n4a1.htm#1, recuperado el 16/11/09.

Softtlan. (2007). "Tutorial de casos de uso", 1703/07, blog disponible en línea: http://softtlan.blogspot.com/2007/03/tutorial-de-casos-de-uso-paso.html, recuperado el 12/11/09.

Zuñiga Pakoz: “Procesos de la Ingeniería de Requerimientos”, Mitecnológico, disponible en línea: http://www.mitecnologico.com/Main/ProcesosDeLaIngenieriaDeRequerimientos, recuperado el 12/11/09

Actividades de Aprendizaje

A.3.1. Elabora una plantilla de especificación para el caso de uso: elaborar una ficha bibliográfica.

A.3.2. Elabora el diagrama de casos de uso: venta de productos en papelería.

A.3.3. Representa mediante un diagrama las posibles interfaces con otros sistemas que tendría el sistema de Facturación de una empresa.

A.3.4. Elabora un cuadro con las posibles entradas y salidas que tendría el sistema de Renta de películas.

A.3.5. Describe en un documento qué aspectos debe cuidar una empresa para lograr la satisfacción total de un cliente al que le brinda un servicio.

Cuestionario de Autoevaluación

1. ¿Qué es un requerimiento?
2. ¿Cuál es la clasificación de los requerimientos?
3. ¿Qué es FURPS?
4. ¿Cuáles son los parámetros de calidad que forman parte de FURPS+?
5. ¿Qué técnicas de recopilación de requerimientos existen?
6. ¿Qué es un caso de uso, para qué nos sirve?
7. ¿Define qué es una Plantilla de especificación y cuáles son sus elementos?
8. ¿Qué elementos debe tener un diagrama de casos de uso, a qué se refieren?
9. ¿Define qué es una Interfaz de Usuario?
10. Define qué otros elementos están involucrados durante la definición de la interfaz con otros sistemas.

Examen de Autoevaluación

Relaciona las columnas escribiendo dentro del paréntesis la letra que una los conceptos con sus respectivas definiciones.

( ) 1. Ejemplos de requerimientos No Funcionales	a) Alta al catálogo de Empleados, Respaldos, Pantalla de consulta, etc.
( ) 2. Es una aplicación que refleja el funcionamiento real y sirve para visualizar de manera global como será el sistema final, además sirve para que el usuario de opiniones y se corrijan posibles desviaciones.	b) Iniciación , Elaboración, Construcción y Transición
( ) 3. Es una actividad o conjunto de actividades que realiza una empresa para satisfacer las necesidades y expectativas de un cliente.	c) Entradas al Sistema
( ) 4. Las funciones del sistema son vistos por el usuario mediante un conjunto de casos, actores y relaciones.	d) FURPS, FURPS+
( ) 5. Paso para desarrollar un caso de uso, se refiere a los objetivos de manera general o responsabilidades que tienen todos los actores.	e) Mantenimiento, Fiabilidad.
( ) 6. Se refieren a los requisitos que necesita el sistema para empezar el procesamiento de la información.	f) Plantilla de especificación
( ) 7. Son modelos de calidad de software.	g) Diagramas de Casos de Uso
( ) 8. Ejemplos de requerimientos Implícitos	h) Proceso de Negocio
( ) 9. Fases que comprende el proceso unificado racional.	i) Identificar metas o roles
( ) 10. Se refiere a la especificación de los casos de uso, la cual tiene varias partes o elementos	j) Prototipo del sistema

TEMA 4. ANÁLISIS ORIENTADO A OBJETOS

Objetivo particular

El alumno reconocerá los principales elementos que debe contener el diseño de un sistema, que permitirá realizar el análisis de proyectos orientados a objetos.

Temario detallado

4.1. Análisis del dominio del problema (modelo conceptual)
4.2. Identificación de responsabilidades de los objetos
4.3. Asociaciones
4.4. Atributos
4.5. Diccionario del modelo
4.6. Diagrama de secuencia del sistema
4.7. Contratos

Introducción

El análisis es la descomposición del problema en sus partes integrantes, y permite especificar los requisitos del usuario, la estructura del sistema y su función. El resultado del análisis es un modelo del comportamiento del sistema, en el que se define el modelo conceptual de clases común para todos los involucrados en los requisitos, como son analistas y clientes.

Este modelo consiste en los objetos del dominio del problema con correspondencia directa en el área de la aplicación.

4.1. Análisis del dominio del problema (modelo conceptual)

Modelo de dominio, también llamado modelo conceptual, es el artefacto más importante que se crea durante el análisis Orientado a Objetos.

Utilizando la notación UML, un modelo de dominio se representa con un conjunto de diagramas de clases, en los que no se define ninguna operación. En esta actividad se identifica un conjunto rico de objetos o clases conceptuales, que representadas de manera visual, nos permitirán mostrar: los objetos de dominio, las asociaciones entre las clases y los atributos de las clases.

Elaboración del Modelo de Dominio

La elaboración del Modelo de Dominio se puede realizar en tres pasos:

1. Identificar las clases conceptuales
2. Dibujar las clases en un diagrama de clases
3. Añadir relaciones entre las clases conceptuales y definir sus atributos.

Identificar las clases conceptuales

Una “clase conceptual” es cualquier cosa que pertenezca al dominio. Por ejemplo: personas, máquinas, lugares así como elementos intangibles como: ventas, permisos, etc. Es decir, se realiza la identificación de las clases candidatas, explícitas o implícitas en los requisitos, referidas en la descripción del problema. Para ello se requiere extraer todos los sustantivos de la descripción del problema o de algún otro documento similar, teniendo en cuenta las siguientes consideraciones:

- Los sustantivos en la descripción del problema son los posibles candidatos a clases de objetos por lo que es conveniente subrayarlos.
- Identificar entidades físicas al igual que las conceptuales.
- No diferenciar entre clases y atributos.
- Añadir clases que puedan ser identificadas por nuestro conocimiento del área que no se hayan mencionado de manera explícita en la descripción del problema.

Por ejemplo

Dentro de la descripción de un problema que dice: “Un sistema de reservaciones que vende boletos para funciones de varios teatros”; cada función se presenta en un horario y se tienen tarifas de acuerdo con el asiento en el teatro; además los usuarios podrán realizar reservaciones a las funciones previamente.

Los **sustantivos subrayados** son: sistema, reservaciones, vende, boletos, funciones, teatros, horario, tarifas, asiento, usuarios.

Las **entidades físicas** son: boletos, teatros, asiento y las **entidades conceptuales** son: sistema, reservaciones, venta, funciones, horarios.

Con ellas se puede elaborar una lista inicial de clases candidatas, en ella se debe excluir a los sustantivos repetidos y se mantendrán en singular.

- ✓ Boleto
- ✓ Teatro
- ✓ Reservación
- ✓ Venta
- ✓ Función
- ✓ Tarifa
- ✓ Asiento
- ✓ Horario
- ✓ Usuario
- ✓ Sistema

En este punto existen varias consideraciones que se deben tomar en cuenta:

- Eliminar clases que pudieran ser más bien atributos,
- Eliminar clases redundantes o irrelevantes y
- Eliminar clases que pudieran ser operaciones o tengan que ver con aspectos de interfaces de usuario o que correspondan a los actores del sistema.

Dibujar las clases en un Diagrama de Clases

Se representa cada clase identificada en la lista, en un recuadro con su nombre como se muestra a continuación, recordando que el modelo de dominio sólo recogerá las clases relevantes para el problema específico que queremos entender.

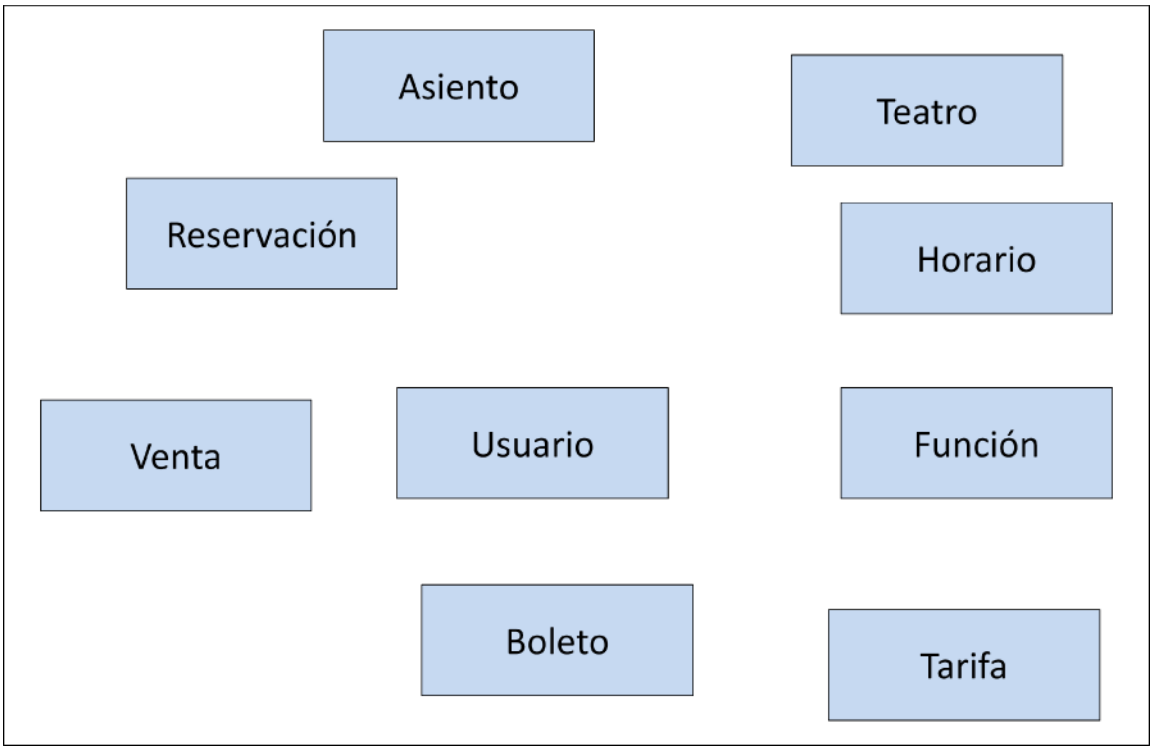


Figura 4.1. Diagrama de clases

Este diagrama de clases ayuda a identificar las responsabilidades y atributos de cada clase, las asociaciones entre ellas, la elaboración del diccionario del modelo y la realización del diagrama de secuencia.

A continuación se deben identificar las responsabilidades de los objetos definidos.

4.2. Identificación de responsabilidades de los objetos

Consiste en determinar cuáles son las responsabilidades u obligaciones de los objetos del diagrama de clases en respuesta a eventos externos que pueden ocurrir en el propio entorno de aplicación.

Estas responsabilidades se refieren a aquellos atributos del Saber y Hacer; sobre los atributos del Saber se encuentran los propios atributos del objeto, atributos de objetos asociados y datos que se puedan derivar; sobre los atributos del Hacer se encuentra: lo que hace el propio objeto, el iniciar una acción con otros objetos y el controlar y coordinar actividades en otros objetos.[6]

Durante el análisis orientado a objetos sólo se identificarán las responsabilidades, no se asignarán responsabilidades a los objetos, eso se realiza durante el diseño.

Siguiendo el ejemplo del sistema de reservaciones que vende boletos para funciones de varios teatros, veamos cuáles son las responsabilidades que se identifican en sus clases conceptuales:

Identificación de las responsabilidades de los objetos Clases del sistema venta de boletos para funciones de varios teatros
Una función contiene reservaciones
Una función tiene una hora de inicio
Una función tiene un teatro
Una función tiene un descanso
Una función tiene una tarifa
Una función tiene una tarifa por persona
Un teatro tiene asientos

[6] Véase, material disponible en línea: http://adimen.si.ehu.es/~rigau/teaching/EHU/ISHAS/Curs2007-2008/Apunts/IS.9.pdf, p. 30, consultado el 20/09/10.

Un usuario puede hacer reservaciones previo a la función Cada boleto tiene un precio de acuerdo con el día de la función, etc.

4.3. Asociaciones

El proceso es similar a la identificación de clases, sólo que en lugar de buscar sustantivos ahora se buscan frases que relacionen a los sustantivos de clases que ya hemos declarado. Tomando en cuenta que una asociación es una relación entre tipos (o más concretamente instancias de estos tipos) que indique alguna conexión significativa, interesante y útil, ya que ello permitirá mantener nuestro diagrama legible. Son relaciones del tipo "pertenece_a" o "está_asociado_con".

Cabe mencionar que este proceso ya no es necesario como parte del modelo de casos de uso, dado que las asociaciones y operaciones del sistema serán declaradas en la parte del diseño. Pero solamente lo ejemplificaremos de una manera sencilla dando seguimiento al sistema de venta de boletos para funciones de varios teatros que ya tenemos.

Es más importante identificar las clases conceptuales que identificar las asociaciones. Entonces, en lugar de iniciar por la descripción del problema o de los documentos de casos de uso, solo se identificarán las frases correspondientes al dominio del problema.

Asociaciones identificadas para relacionar clases de Reservaciones
Una función está_asociado_con boleto Una venta está_asociado_con boleto Un boleto está_asociado_con una tarifa Un teatro está_asociado_con asientos Una reservación está_asociado_con una función Una función está_asociado_con un horario

Un usuario está_asociado_con reservaciones etc.

Una vez identificadas las asociaciones entre clases, se representa en el diagrama de clases que ya se tenía, marcándolas con una línea directa entre ellas, como se muestra a continuación, nota que no es necesario poner el texto de la asociación.

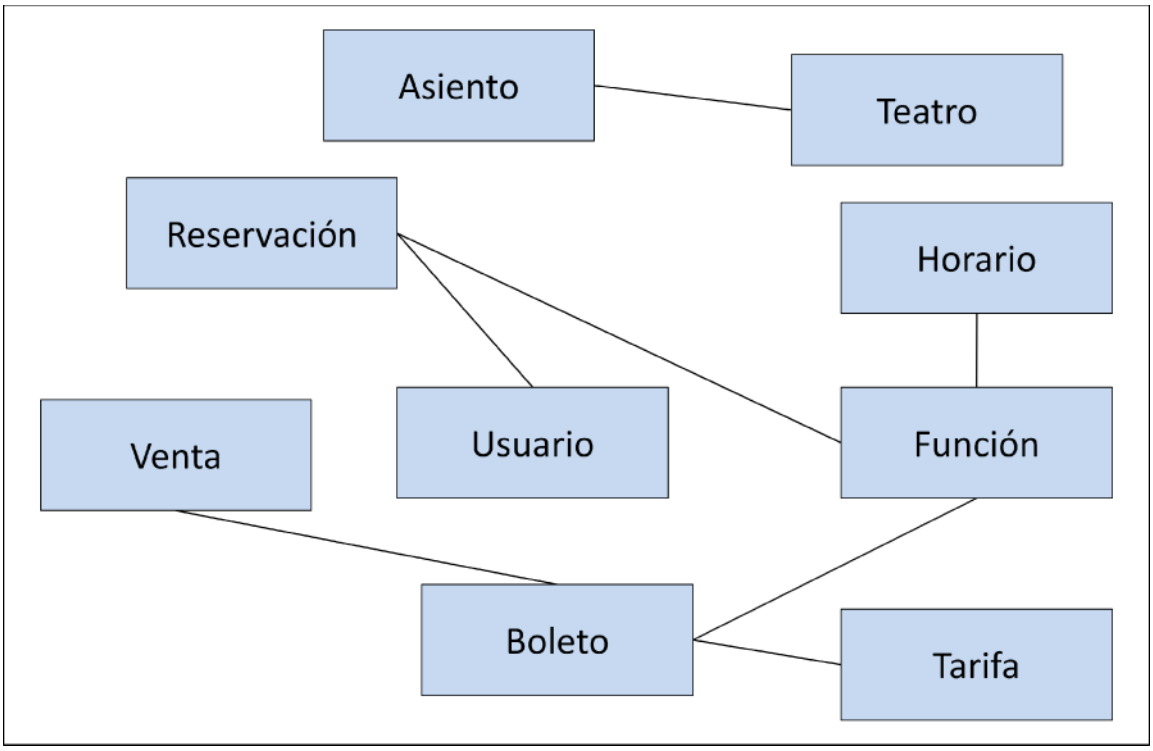


Figura 4.2. Diagrama de clases con asociaciones

En problemas más complejos donde el dominio es más difícil de analizar, se pueden realizar listas de asociaciones, clasificándolas como comunes o de alta prioridad.

Las asociaciones comunes contienen una relación de categorías habituales que normalmente merece la pena tomar en cuenta en los siguientes casos: A es una descripción de B, A es una línea de una transacción o importe de B, A es una subunidad organizativa de B.

Y las asociaciones de alta prioridad son aquellas que invariablemente son útiles y necesarias incluirlas en el modelo conceptual, como los siguientes casos: A es una parte lógica o física de B, A está contenida física o lógicamente en B, A se registra en B.

Los nombres de las asociaciones deberán tener el formato: nombre tipo- frase verbal, donde la frase verbal crea una secuencia legible con significado en el contexto real; y deberán tener una Dirección, normalmente es de derecha a izquierda o de arriba abajo.

A continuación se debe definir cuántas instancias de una clase A pueden asociarse con una instancia de una clase B, llamada Multiplicidad de la relación; Indicando el número de instancias de una clase asociada a otra, considerando que existen tres tipos de multiplicidad de la relación: uno a uno, uno a muchos o muchos a muchos y que deberá representarse con la siguiente notación, según UML:

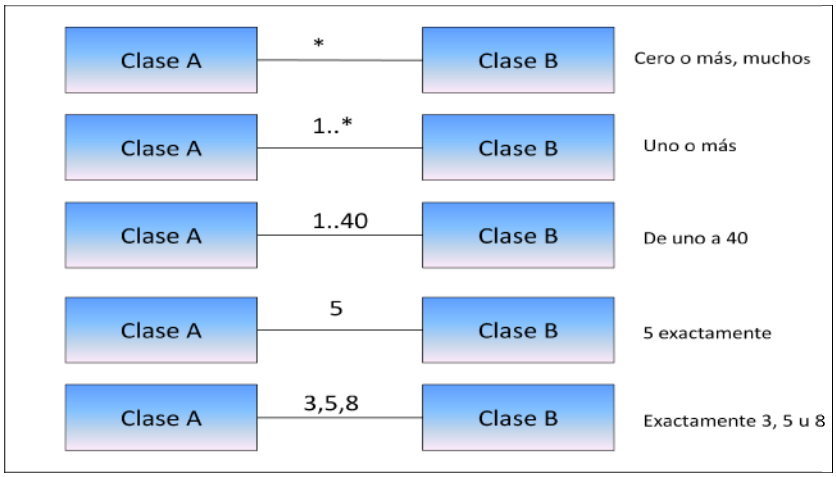


Figura 4.3. Representación de la Multiplicidad en Clases

Relaciones entre clases: dependencia, asociación y agregación

El enfoque de objetos permite tres tipos de relaciones entre elementos, todas ellas representadas en el diagrama de clases.

Relación de dependencia, o de **uso**, se da cuando los objetos de una clase utilizan los objetos de la segunda como argumento en alguna operación, o sus propios objetos utilizan alguna de las operaciones de la otra clase.

En UML está relación se representa con una flecha discontinua del elemento dependiente *A* hacia el elemento independiente *B* es decir, del que usa hacia el usado.

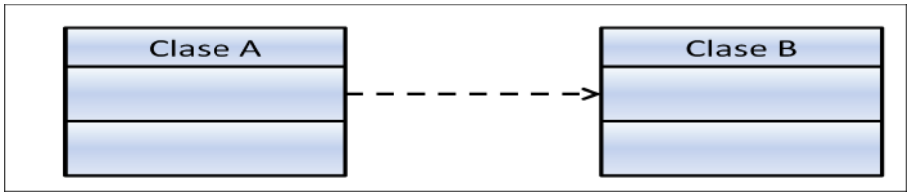


Figura 4.4. Relación de dependencia o uso UML

Relación de asociación se da cuando los objetos de una clase contienen atributos que son objetos de la segunda clase. Esta relación permite "navegar" de los objetos que contienen a los objetos contenidos y se puede presentar en dos modalidades: unidireccional y bidireccional. En la primera, al menos un atributo de los objetos de una clase pertenece a otra clase y en la segunda, los objetos de cada clase contienen al menos un atributo perteneciente a la otra clase.

Estas asociaciones se representan en UML con una flecha continua de la clase que contiene a la clase cuyos objetos son contenidos en el caso de la direccional, y en la bidireccional con una línea continua entre las clases asociadas.

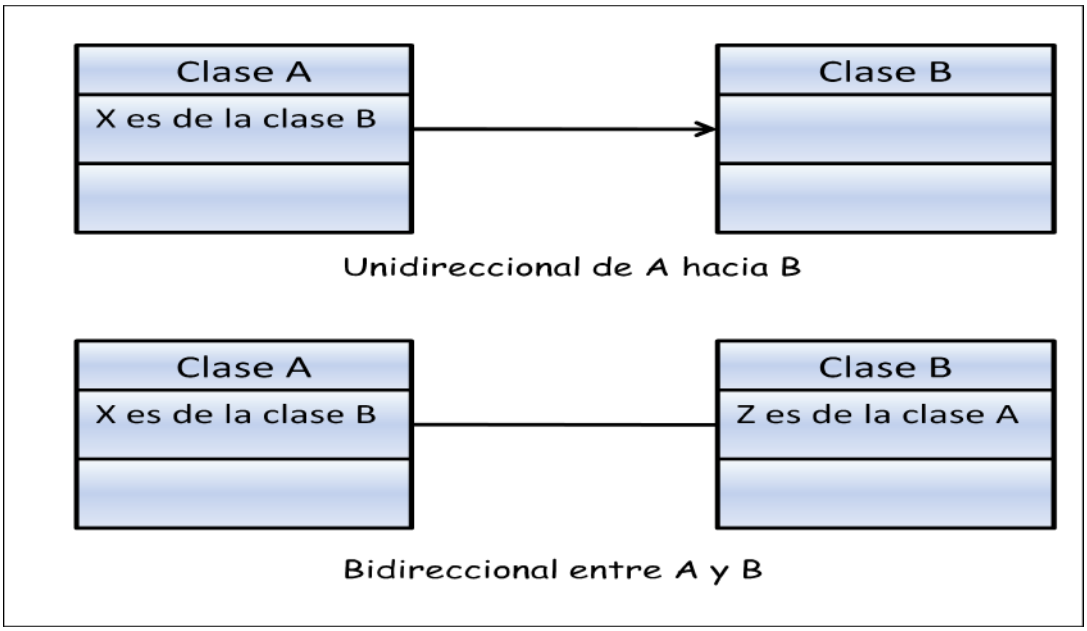


Figura 4.5. Relación de asociación UML

Relación de Agregación: se da cuando hay una relación "todo/parte" entre los objetos de las dos clases. En UML se representa utilizando una línea continua y un rombo en el lado del todo para representar una relación de agregación.

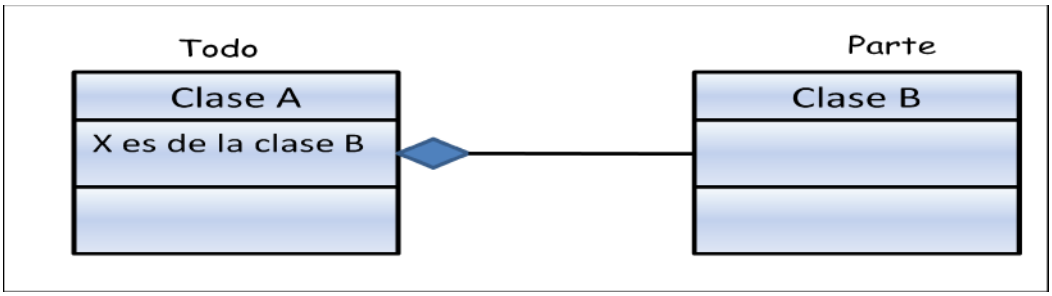


Figura 4.6. Relación de Agregación UML

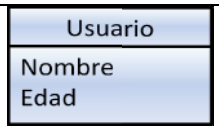
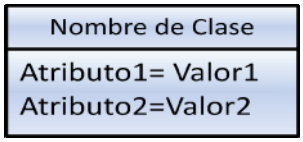
4.4. Atributos

Los atributos definen la estructura de una clase, es decir los valores de los datos para todos los objetos pertenecientes a una clase, por ejemplo: Nombre, teléfono, correo electrónico son *atributos* de la clase usuario. Pueden ser sustantivos: nombre, edad, teléfono, correo electrónico, etc. o adjetivos: Juanita, 24, 43 32 22 79, juanitaPerez@prodigy.net.mx.

Se debe definir un valor para cada uno de ellos, pueden ser iguales o distintos en los diferentes objetos, por ejemplo: el valor del atributo edad puede ser 24 para Juanita, Pedro y Juan, pero para Eduardo de 30.

Dentro de una clase los nombres de los atributos deben ser únicos.

Los atributos deben listarse en el diagrama de clases a continuación del nombre de la clase como una segunda sección, por ejemplo:

<table>
<tr><td>Clase con atributos</td><td> 
</td></tr>
<tr><td>Clase con atributos: nombre y edad para la entidad usuario</td><td> 
</td></tr>
<tr><td>Notación Extendida
 

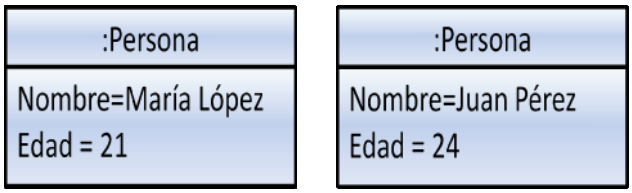
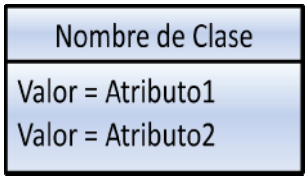
Ejemplos:
 
</td><td>Notación Compacta
 
</td></tr>
</table>

Figura 4.7. Representación de atributos en una clase

Nota la diferencia entre notación extendida y notación compacta, en la extendida se muestra cada valor en la misma clase, y en la compacta se presenta de manera genérica, que es la más utilizada.

El proceso de identificar los atributos tiene cierta complejidad, sin embargo puede resultar más fácil identificar nuestros propios atributos para las distintas clases identificadas en el dominio del problema del sistema de reservaciones como se muestra a continuación:

Atributos identificados para las clases del sistema de reservaciones que vende boletos para funciones de varios teatros

Clases	**Atributos**
Asiento	Id_teatro, fila, Letra
Teatro	Id_teatro, nombre, dirección, teléfono, tot_asient
Usuario	Id_usuario, Nombre, Dirección, Colonia, Ciudad, País, Código Postal, Teléfono, Fax, Email, Login, Password
Horario	Id_horar, Día, Hora
Función	Id_función, Título, clasificación, autor
Tarifa	Id_tarifa, Día, Precio
Reservación	Id_reservación, fecha_pag, asientos, importe, id_teatro
Venta	concepto, importe, iva, total, rfc, id_usuario

Una vez que se han definido los atributos, es necesario definir su tipo, pudiendo ser estos: public, private y protected. Todos ellos nos definen el grado de comunicación y visibilidad que tienen en el entorno.

Public indica que será visible tanto dentro como fuera de la clase.
Private indica que el atributo será accesible sólo desde dentro de la clase y

Protected indica que el atributo no será accesible desde fuera de la clase, pero sí podrá mostrarse por métodos de la clase además de las subclases que se deriven por herencia.

Al igual que los atributos los métodos también pueden clasificarse dentro de estos tres tipos.

Los atributos de las clases en el ejemplo del sistema de venta de boletos para funciones de varios teatros:

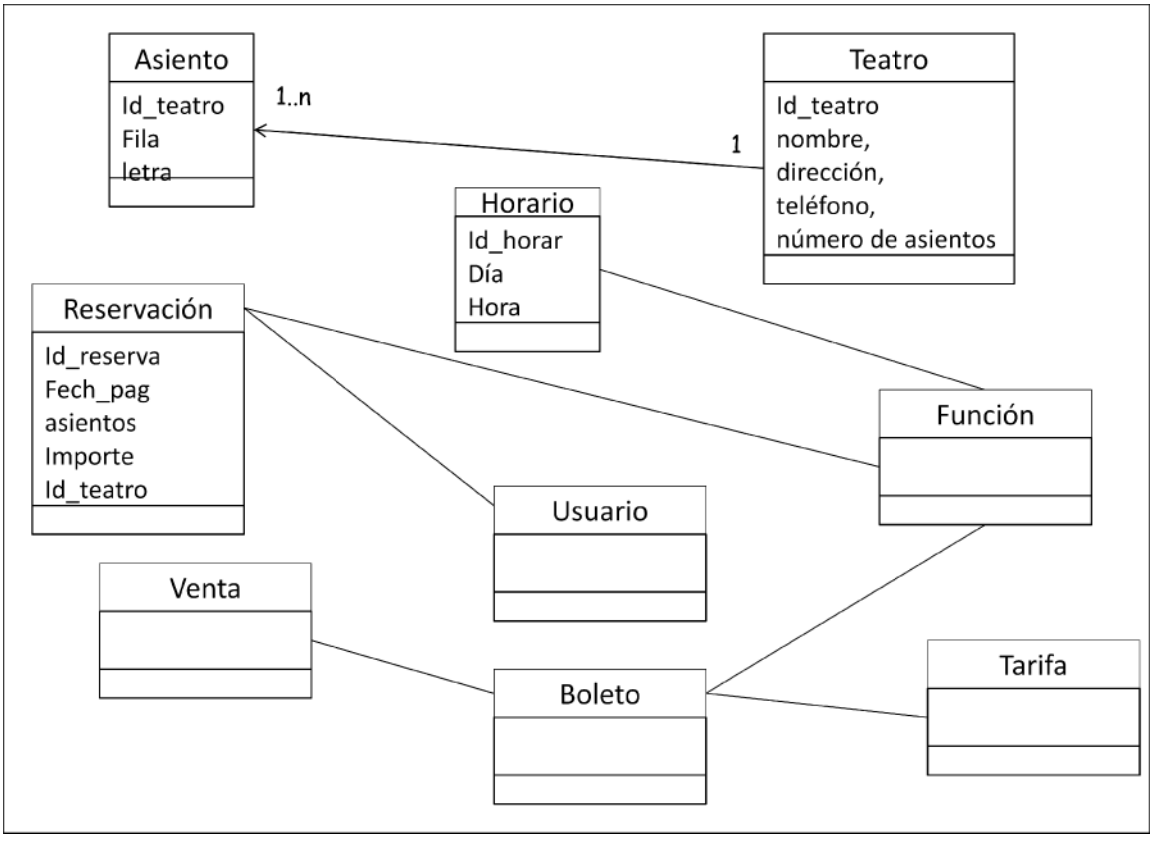


Figura 4.8. Diagrama de clases con atributos

4.5. Diccionario modelo

Es un listado organizado de todos los datos utilizados en el sistema, con definiciones rigurosas y precisas que permitan al analista y al usuario entender las diferentes entradas, salidas almacenamientos y cálculos intermedios. También llamado diccionario de datos.

El diccionario modelo describe de manera textual a las clases identificadas. Sirve como glosario de términos, y nos dan respuestas al ¿Qué?, ¿Cómo?, ¿Cuándo?, ¿Quién? y ¿Dónde? Se utiliza para un mejor manejo en sistemas muy grandes, para un significado de actividades y elementos, para documentar características del sistema, como almacén de información, para determinar algún cambio en el sistema, administrar identidades, localizar errores u omisiones.

En el ejemplo del sistema de reservaciones que vende boletos para funciones de varios teatros tenemos:

Clase	Atributos	Tipo	Req	Enlace	Descripción
Asiento	Id_teatro fila letra	Int(3) Int(3) VarChar(3)	Si Si Si	Teatro	Se denomina por medio de un número de fila y una letra. El asiento pertenece a un teatro.
Reservación	Id_reserva Fech_pag Asientos Importe Id_teatro	Int (5) Date Int (5) Int (9)	Si Si Si	Teatro Función	Para poder ver una función, es necesario contar con una reservación previa, la cual debe pagarse antes de una fecha límite, incluso el mismo día del evento. Una reservación puede hacerse para múltiples teatros y diferentes funciones. La reservación cuenta con una clave que corresponde a un récord de reservación particular.

Clase	Atributos	Tipo	Req	Enlace	Descripción
Usuario	Id_usuario Nombre Dirección Colonia Ciudad País Código Teléfono Fax Email Login password	Int (5) Varchar(50) Varchar(50) Varchar(50) Varchar(50) Varchar(50) Varchar(50) Varchar(20) Varchar(10) Varchar(15) Varchar(10) Varchar(6)	Si Si	Teatro Función	Para poder utilizar el sistema de reservaciones, el usuario debe estar registrado. El registro contiene información acerca del usuario como nombre, dirección, colonia, ciudad, país, código postal, teléfono de casa, teléfono de oficina, fax, email, login y password.
horario	Id_horar Día Hora	Int (5) Varchar(10) Time	Si Si Si		El horario de una función de teatro se determina por su hora de inicio durante los días que se presentará.
Teatro	Id_teatro Nombre Dirección Teléfono tot_asient	Int(5) Varchar(60) Varchar(50) Varchar(30)	Si Si Si SI		Es el lugar donde se presentan las funciones, y tiene un nombre, dirección y teléfono.
Función	Id_Función Título Clasif Autor	Int(5) Varchar(50) Varchar(50) Varchar(100)	Si Si Si		Una función tiene el título de la obra que se presenta, así como su autor y la clasificación, si es apta para menores de edad.
tarifa	Id_tarif Precio Día	Int (5) Varchar(50) Int(10)	Si Si Si		Una función puede tener diferentes tarifas según el día que se presente, sábados y domingos es más alto que entre semana; y dependiendo de qué tan cerca esté el lugar del escenario.

4.6. Diagrama de Secuencia del sistema

Muestra precisamente la secuencia de los eventos entre los actores y el sistema. También permite identificar las operaciones del sistema. La descripción de estos diagramas se realiza posterior a la descripción de los casos de uso.

Construcción de un diagrama de Secuencia del sistema
1. Dibujar una línea vertical que representa el sistema
2. Dibujar una línea para cada actor que interactúa directamente con el sistema
3. Del curso de eventos del caso de uso, identificar los eventos externos generados por los actores, mostrarlos en el diagrama.
Nota. Lo importante en el diagrama de secuencia es el orden en que los eventos ocurren y la dependencia entre ellos.

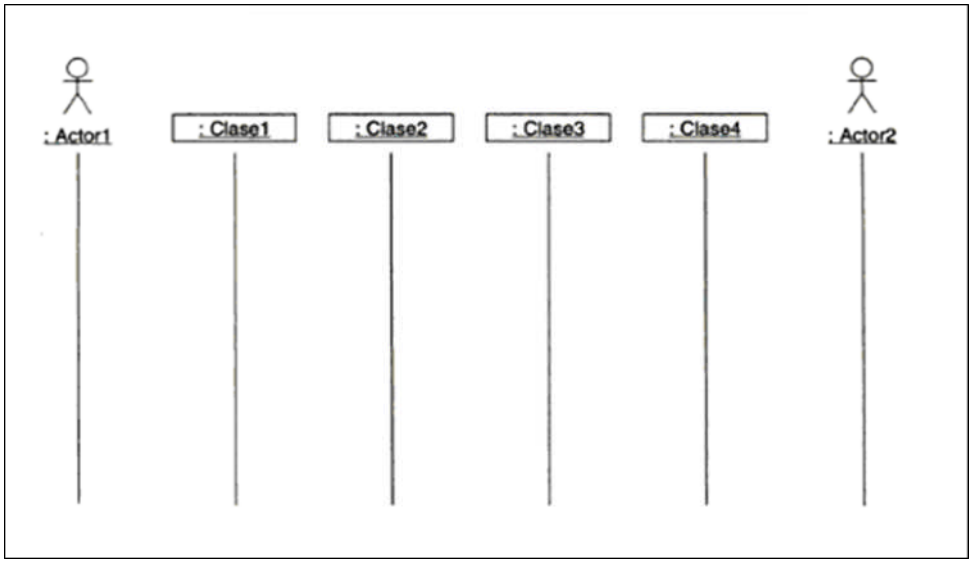


Figura 4.9. Diagrama de Secuencia, representación de clases y actores que intervienen. Weitzenfeld (2004, p. 276)

Se representa con barras gruesas a las actividades del objeto (a), con flechas los eventos (c); Un evento se representa con una flecha horizontal que comienza en la barra del objeto que lo envía y termina en la barra del objeto que lo recibe. En el siguiente ejemplo los eventos se ven numerados utilizando el número n seguido de dos puntos (n:)

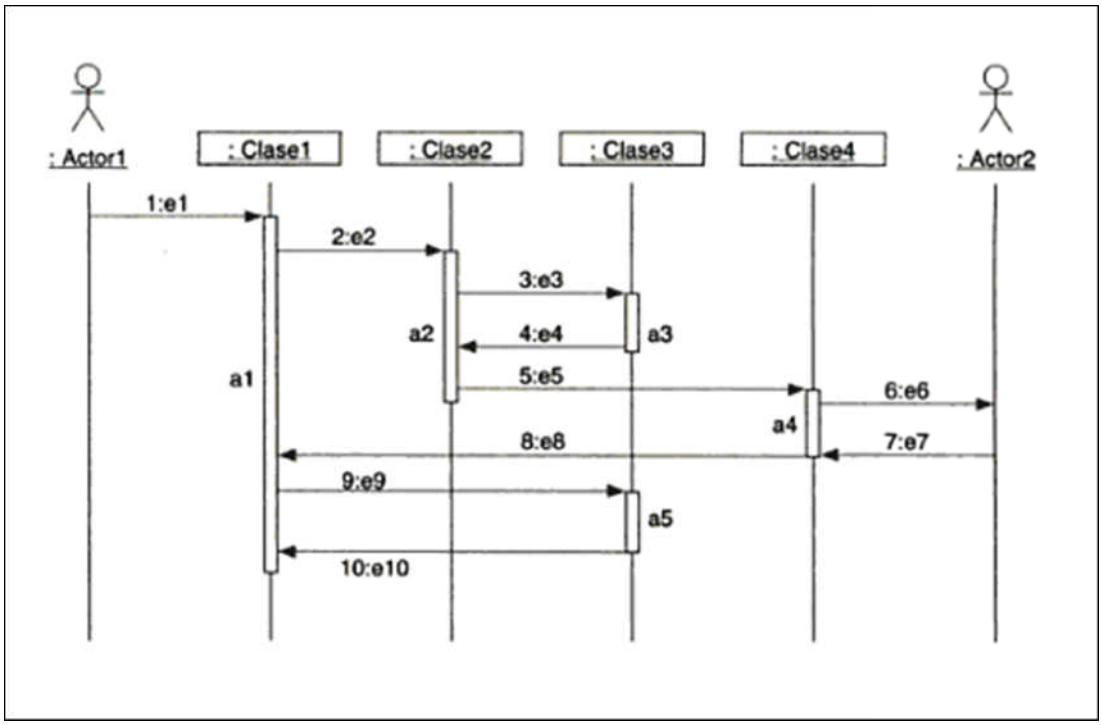


Figura 4.10. Diagrama de Secuencia, representación de clases y actores que intervienen. Weitzenfeld (2004, p. 278)

Las actividades inician por la llegada de eventos y el tiempo que duran es sólo relevante para resaltar eventos posteriores originados durante esa actividad. Podemos decir que en general, un diagrama de secuencia nos permite visualizar la fluidez de los eventos y la correspondencia de funcionalidad con la del caso de uso. Es importante que exista continuidad en los eventos. Finalmente recordemos que, los diagramas de secuencia sólo se utilizarán para describir los flujos principales y subflujos de cada caso de uso, esto nos ayudará a revisar la lógica y detectar posibles inconsistencias entre los casos de uso, los requisitos y la arquitectura de clases del modelo del análisis.

A continuación se presenta un ejemplo de un diagrama de secuencia para *Efectuar una reservación.*

En él podemos observar el flujo principal para efectuar una reservación que deberá incluir ciertas operaciones: validar asientos disponibles, mostrar asientos disponibles, validar horarios disponibles, obtener tarifas, registrar venta, registrar asiento ocupado, así como las clases y actores que intervienen en él.

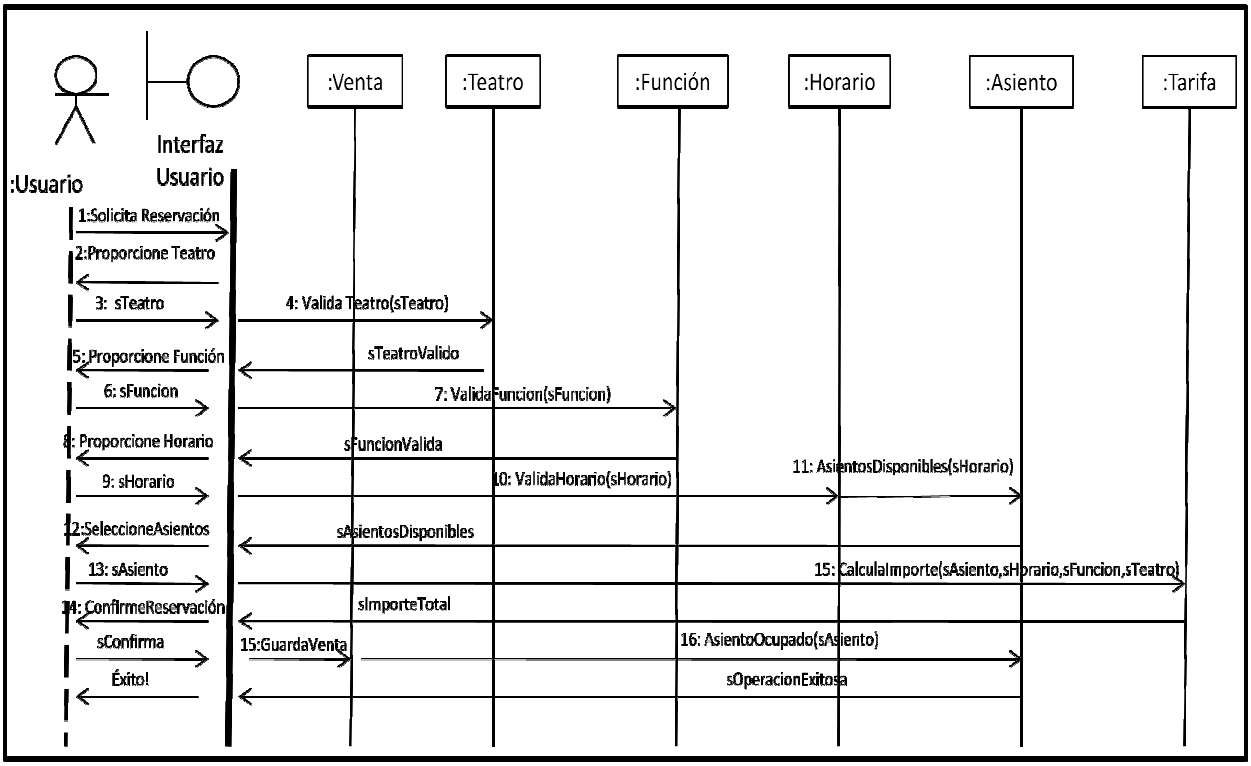
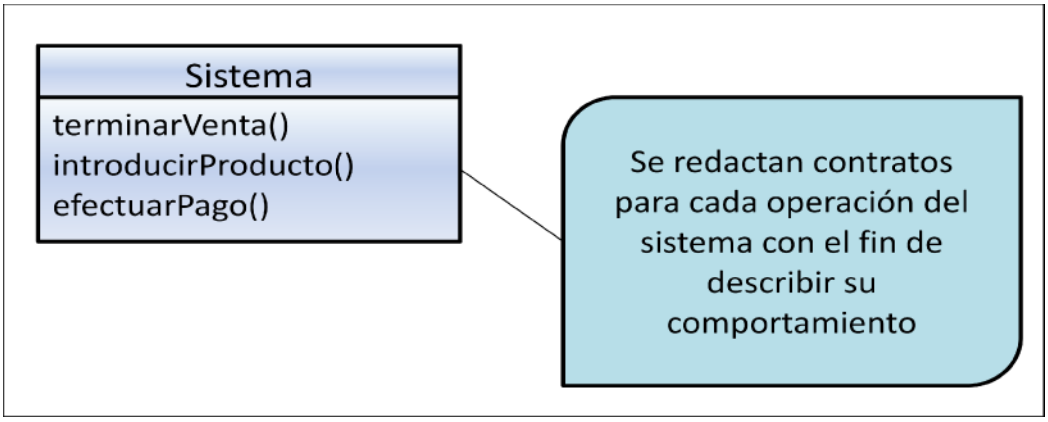


Fig. 4.11. Diagrama de Secuencia, representación de clases y actores que intervienen.

4.7. Contratos

Un contrato es un documento que describe lo que se propone cumplir o lograr una operación; se redactan de una manera declarativa, haciendo hincapié en lo que sucederá y no en el cómo sucederá. Se elaboran durante la fase de análisis.

- Describen el comportamiento del sistema informático según los cambios de estado de la base de información y según salidas que el sistema proporcione.
- Toda operación tiene un contrato. Por ejemplo:



Si el sistema tiene las operaciones: terminarVenta(), introducirProducto() y efectuarPago(), entonces tendremos tres contratos, uno para cada operación.

- El uso de contratos ayuda a eliminar las pretensiones asumidas por las partes involucradas; a eliminar la incertidumbre y dar mayor seguridad; a realizar simplicidad en los procesos de formación.

Elementos o Secciones del contrato	
Operación o Nombre	Se refiere al nombre de la operación y sus parámetros
Responsabilidades	Descripción informal de la función IR responsabilidades que debe cumplir la operación.
Tipo	Entorno de la operación (concepto, clase de software, interfaz)
Referencias	Números de referencia de las funciones del sistema, casos de uso, etc. que intervienen en la operación.
Notas	Notas de diseño, algoritmos, e información afín.
Excepciones	Descripción de la relación del sistema en situaciones no esperadas, casos inusuales.
Salida	Mensajes o registros que se envían fuera del sistema.
Poscondiciones	El estado del sistema después de la operación.
Precondiciones	Suposiciones acerca del estado del sistema antes de ejecutar la operación.

Para preparar un contrato en los casos de uso, es necesario identificar las operaciones del sistema a partir de los diagramas de secuencia, -ya vimos cómo se elaboran. Se debe elaborar un contrato por cada operación identificada, se recomienda iniciar con la redacción en la sección de responsabilidades y después se hace de manera formal. Luego, se completa la sección de poscondiciones, describiendo los cambios de estado en el modelo conceptual.

El siguiente es un ejemplo de cómo relacionar contratos con los principales artefactos en un caso de uso.

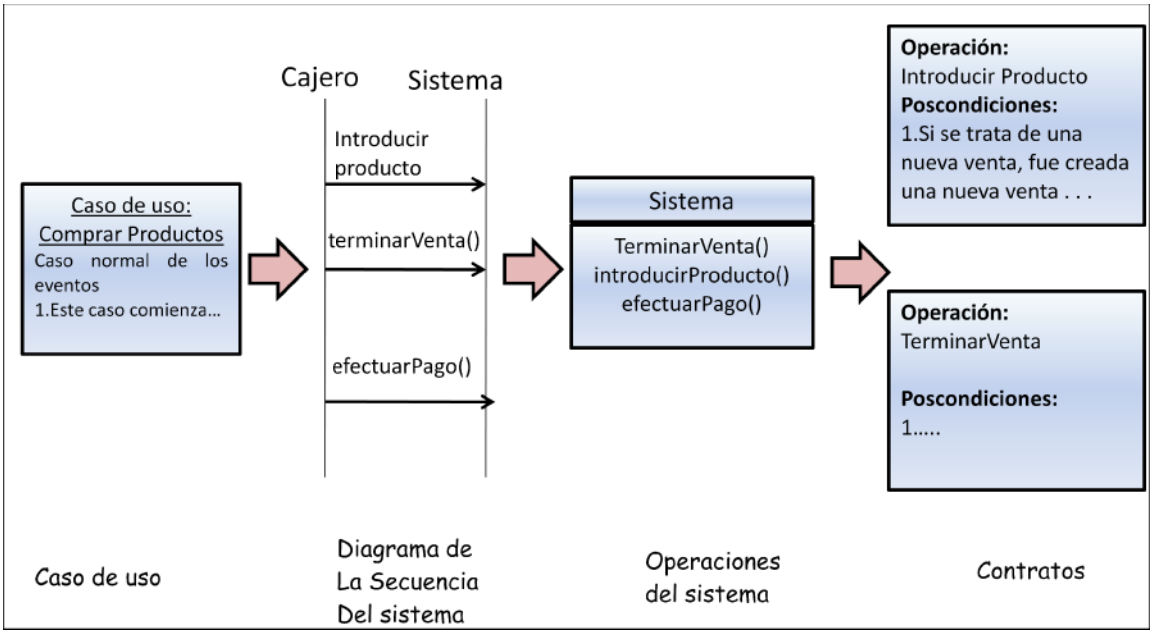


Figura 4.12. Relación Contratos con otros artefactos

El contrato para introducir producto puede ser:

Contrato Introducir Producto	
Nombre	introducirProducto (cup: número, cantidad: entero)
Responsabilidades	Registrar la venta del producto y agregarlo a la venta. Mostrar la descripción del producto y su precio.
Referencias	Funciones del sistema: R1.1, R1.3,
Cruzadas:	Casos de uso: comprar productos.
Tipo:	Sistema
Excepciones:	Si el cup no es válido, indicar que se ha incurrido en un error.
Salida:	
Precondiciones:	El sistema conoce el cup
Poscondiciones	Se creó la instancia VentasLineaDeProducto, se asoció VentasLineaDeProducto con la venta, etc.

Bibliografía del tema 4

Gómez, Cristina, *et al*. (2003). *Diseño de sistemas software en UML.* Barcelona: UPC

Graham, Ian. (1996). *Métodos orientados a Objetos,* 2ª ed., México: Addison Wessley Longman

Larman, Craig. (2006). *UML y patrones: Una introducción al análisis y Diseño Orientado a Objetos y al Proceso Unificado*, 2ª ed., México: Prentice Hall.

Teniente López, Ernest, Dolors Costal Costa, Sancho Samsó, M. Ribera. (2003). *Especificación de Sistemas Software en UML*, Barcelona: UPC

Weitzenfeld, Alfredo. (2004). *Ingeniería de software orientada a objetos con UML, Java e Internet*, México: Thomson International.

Actividades de aprendizaje

A.4.1 Identifica las clases conceptuales en la siguiente lista de requerimientos o definición del problema.

El sistema de reservaciones es un sistema que permite al usuario hacer consultas y reservaciones de vuelos además de poder comprar boletos aéreos en forma remota, sin necesidad de recurrir a un agente de viajes.
Se desea que el sistema sea accesible desde Internet.
El sistema debe presentar una pantalla principal un mensaje de bienvenida que describa los servicios ofrecidos.
El acceso debe ser mediante una clave y una contraseña (password).
La consulta de los vuelos debe poder hacerse de 3 maneras, por horarios, por tarifas o por estado del vuelo.
La consulta de tarifas debe mostrar los diferentes vuelos entre ciudades, dando prioridad al costo más bajo.
El pago en reservaciones puede hacerse con tarjeta de crédito.
Los boletos pueden enviarse al cliente, o deben estar listos para entrega en mostrador.

A.4.2 Realiza el diagrama de clases conceptuales, resultado del inciso anterior.

A.4.3 Define las responsabilidades de los objetos en el diagrama anterior.

A.4.4 Detecta las asociaciones entre objetos, marca cuáles son comunes y cuáles de alta prioridad.

A.4.5 Define los atributos para las clases conceptuales, del inciso anterior.

A.4.6 Elabora el diagrama de clases resultante en las actividades anteriores, que representa el modelo del dominio.

A.4.7 Elabora el diccionario modelo resultado de las actividades anteriores.

Cuestionario de autoevaluación

1. ¿A qué se refiere el análisis orientado a objetos?
2. ¿Cómo se representa un modelo de dominio, utilizando UML?
3. ¿Cuáles son los 3 pasos a seguir para elaborar el modelo de dominio?
4. ¿Cómo se identifican las responsabilidades de los objetos y qué consideraciones debo tomar para ello?
5. ¿Cómo se identifican las asociaciones entre objetos?
6. ¿Qué es el diccionario modelo, cuál es su objetivo?
7. ¿Qué representa el diagrama de secuencia y cuáles son sus objetivos?
8. ¿Qué elementos participan en la elaboración de los diagramas de secuencia?
9. ¿Qué es un contrato?
10. ¿Cuáles son los elementos o secciones del contrato?

Examen de autoevaluación

Relaciona las columnas escribiendo dentro del paréntesis el número que una los conceptos con sus respectivas definiciones.

1. Modelo de dominio en UML	a. Es un listado organizado de todos los datos utilizados en el sistema, con definiciones rigurosas y precisas.	( )
2. Asociaciones comunes	b. Se da cuando los objetos de una primera clase utiliza los objetos de la segunda como argumentos de alguna operación, o sus propios objetos utilizan alguna de las operaciones de otra clase.	( )
3. Diagramas de secuencia	c. Documento que describe lo que se propone cumplir o lograr en una operación.	( )
4. Relaciones de agregación	d. Es el artefacto más importante que se crea durante el análisis Orientado a Objetos y se representa con un conjunto de diagramas de clases.	( )
5. Diccionario Modelo	e. Es la representación gráfica de cualquier cosa que pertenece al dominio, así como sus relaciones y atributos.	( )
6. Asociaciones de Alta Prioridad	f. Contienen una relación de categorías habituales que, normalmente merece la pena tomar en cuenta en los siguientes casos: A es una descripción de B, A es una transacción de B, A es una subunidad de B.	( )
7. Contrato	g. Es la relación todo o parte entre objetos de dos clases.	( )
8. Diagrama de Clases	h. Definen la estructura de una clase, es decir los valores de los datos para todos los objetos pertenecientes a una clase.	( )
9. Relaciones de dependencia	i. Muestran cronológicamente los eventos entre los actores y el sistema.	( )
10. Atributos	j. Son inevitablemente útiles y es necesario incluirlos en el modelo conceptual, como los siguientes casos: A es una parte lógica o física de B.	( )

TEMA 5. DISEÑO ORIENTADO A OBJETOS

Objetivo particular

El alumno reconocerá la importancia de elaborar los diagramas de colaboración durante el diseño orientado a objetos, la forma de asignar mensajes, responsabilidades y la visibilidad a los objetos participantes en los diferentes casos de uso. También practicará el diseño de un sistema orientado a objetos elaborando los diagramas de clases de diseño, el propio diseño de la interfaz de usuario, la selección correcta de un lenguaje de programación, de un manejador de bases de datos, de la plataforma y la elaboración de la documentación del sistema.

Temario detallado

5.1. Diagramas de colaboración (muestra los mensajes entre los objetos de software)
5.2. Asignación de responsabilidades de los objetos
5.3. Determinación de la visibilidad entre objetos
5.4. Diagramas de clases del diseño
5.5. Diseño de la interfaz de usuario
5.6. Elección del lenguaje, manejador de bases de datos y plataforma
5.7. Documentación

Introducción

En la fase de diseño orientado a objetos se presta especial atención a la definición de los objetos de software y en cómo colaboran para satisfacer los requerimientos establecidos durante la planeación y la elaboración; para ello, se hará uso de los diagramas de colaboración y de los diagramas de clases de diseño. Posteriormente y para concluir el desarrollo de un sistema utilizando la

metodología de análisis y diseño orientado a objetos, se mostrarán los puntos más importantes a considerar para tomar decisiones que nos permitan elaborar el diseño de la interfaz, la elección del lenguaje de programación que se utilizará, el manejador de bases de datos adecuado, la plataforma a utilizar y la documentación.

5.1. Diagramas de Colaboración (muestra los mensajes entre los objetos de software)

Un diagrama de colaboración ilustra las interacciones entre los objetos en un formato de grafo o red, este tipo de diagramas destaca la organización estructural de los objetos que envían y reciben mensajes, muestra también las interacciones que existen entre los objetos; comparando con los diagramas de secuencia, observamos que contienen más o menos la misma información, pero a diferencia de estos no muestran la forma en que las operaciones se producen en el tiempo sino que centran su interés en las relaciones entre objetos y su topología.

Estos diagramas ayudan mucho en demostrar o explicar un proceso dentro de la lógica de un programa ya que permiten modelar los aspectos dinámicos de un sistema, es decir instancias concretas, clases, interfaces, componentes y nodos junto con los mensajes enviados entre ellos. Pueden ser utilizados también para visualizar, especificar, construir o documentar la dinámica de un grupo particular de objetos, o bien para modelar el flujo particular de un caso de uso.

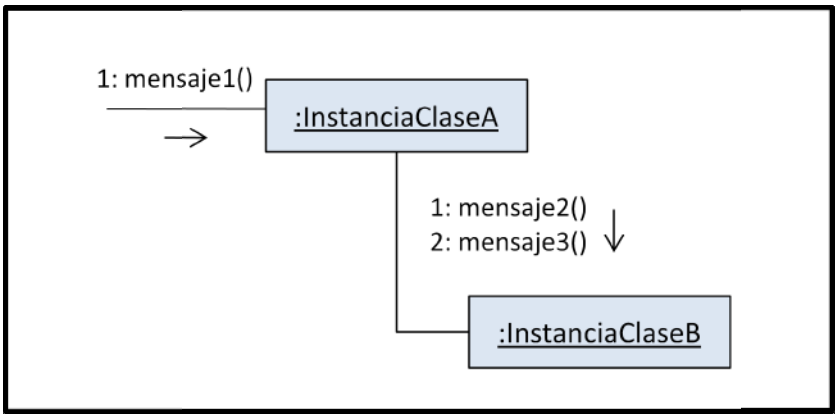


Figura 5.1. Diagrama de Colaboración o de Interacción

Un ejemplo muy sencillo de lo que es un diagrama de colaboración en donde observamos que las flechas representan los mensajes enviados de un objeto a otro, mostrando el nombre del mensaje y los parámetros de secuencia del mensaje, como puede observarse en la Figura 5.1.

Contenido

Los diagramas de interacción contienen: Objetos, Enlaces y Mensajes, y al igual que los demás diagramas pueden contener notas y restricciones.

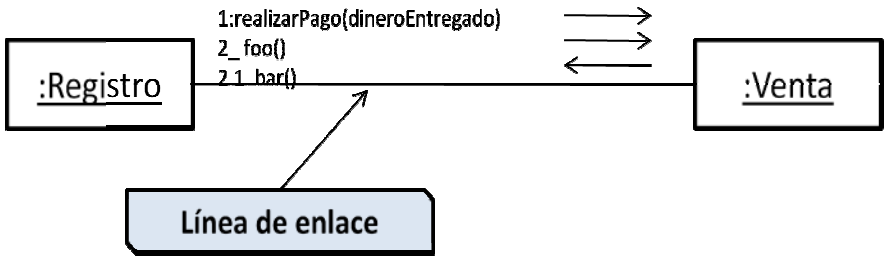
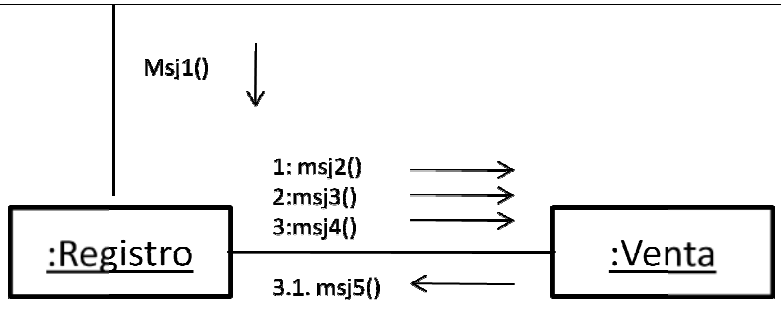
Elemento	Descripción
Enlaces	Es el camino de conexión entre dos objetos.
Mensajes	Son aquellos que se dan entre objetos con una expresión representados con una pequeña flecha que indica la dirección del mensaje. Cada mensaje se enumera según su secuencia y se representa mediante números. Los mensajes se pueden enviar desde un objeto a él mismo. (“*self*” o ”*this*”). Existen mensajes condicionales que se representan con corchetes cuadrados [ ], dentro la condición e indica que el mensaje se enviará únicamente si la condición es verdadera.

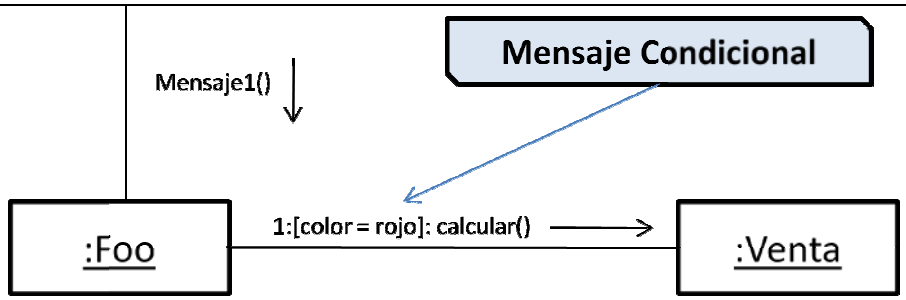
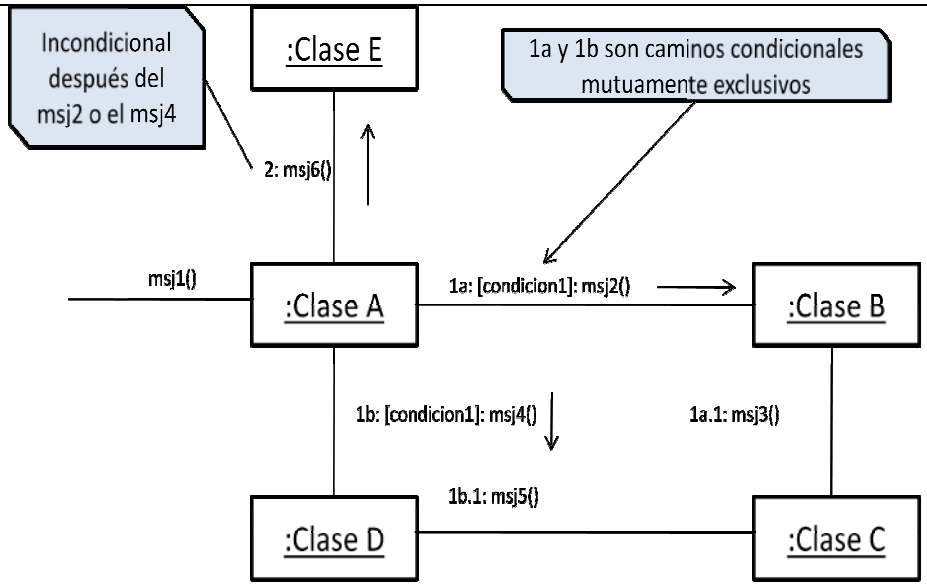
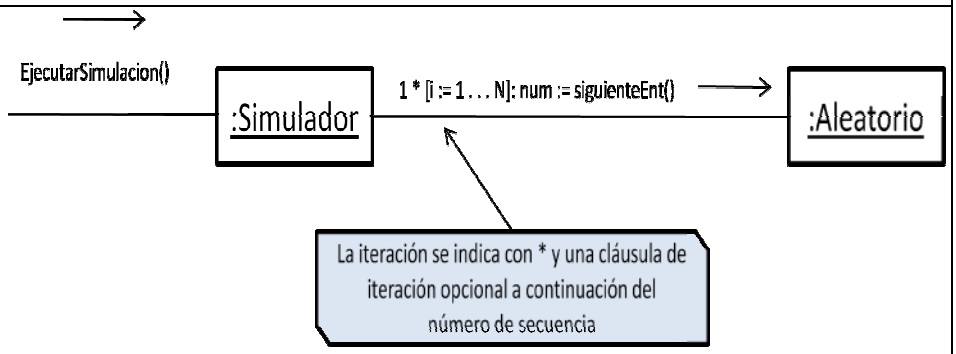
	Pueden existir mensajes de un objeto a una clase, entonces la clase será representada igual que un objeto con la diferencia de que su nombre no va subrayado como el objeto.
Objetos	Se representan en un recuadro con su nombre subrayado anteponiendo dos puntos, lo que indica una instanciación.

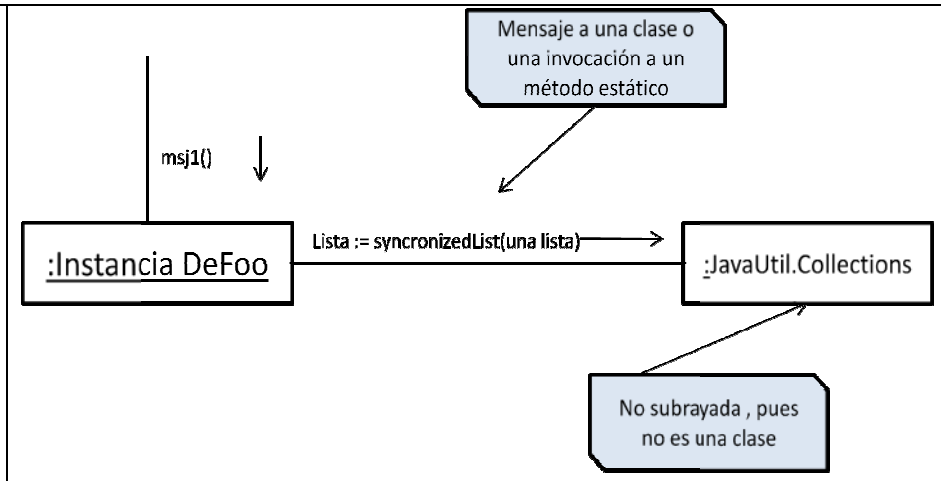
Cuadro. 5.1. Elementos en un diagrama de colaboración

La manera de construir un diagrama de colaboración es, primero, colocar los objetos que participan en la colaboración como nodos del grafo. A continuación se representan los enlaces que conectan esos objetos como arcos del grafo. Y por último, a estos enlaces se incluyen los mensajes que envían y reciben los objetos. Esto da una visión clara del flujo de control en el contexto de la organización estructural de los objetos que colaboran.

Veamos a continuación la representación en cada elemento:

Enlaces	
Mensajes	

Mensajes Condicionales	 Observe que el mensaje se envía solamente cuando la variable color es igual a “rojo”.
Caminos Condicionales Mutuamente Exclusivos	
Iteración o Bucle	

Mensaje a un objeto Clase	

Cuadro 5.2. Representación de cada elemento del diagrama

Si se elaborara el diagrama de colaboración del caso de uso: **realizar un pago**, sería el siguiente:

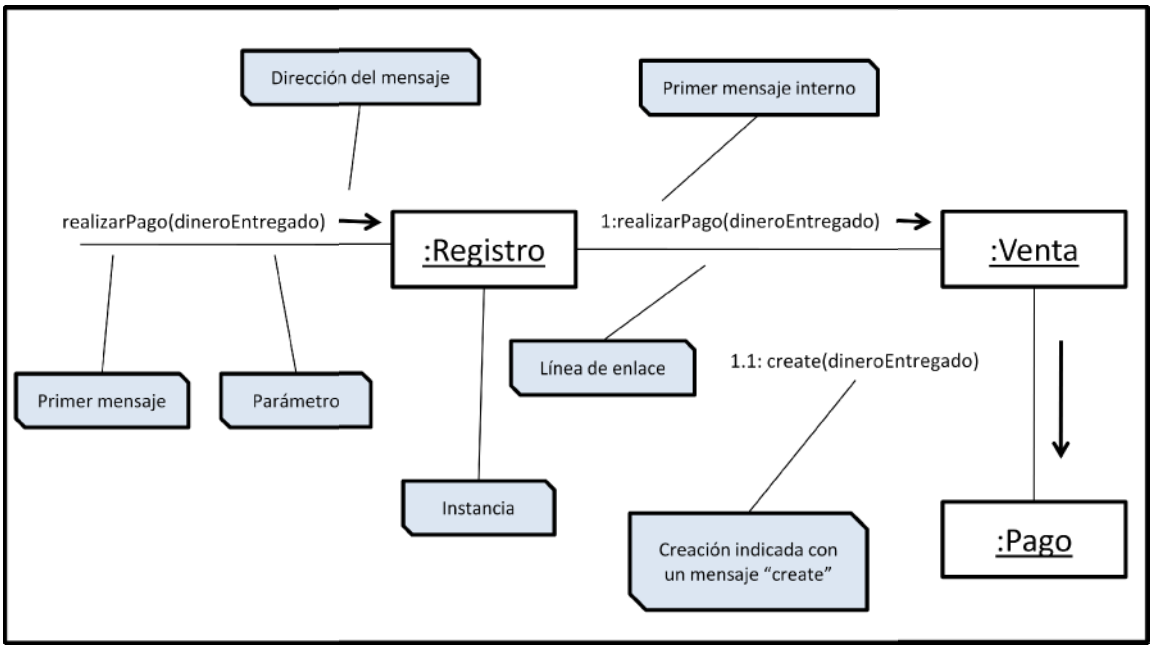


Figura. 5.2 Diagrama de Colaboración del caso de uso *Realizar un pago*

Al interpretarlo observaríamos lo siguiente:

Primero.- que se envía el mensaje *Realizar Pago* a una instancia de *Registro*, y no se identifica en este caso al emisor;

Segundo.- que la instancia *Registro* envía el mensaje *Realizar Pago* a una instancia de *Venta y*

Tercero.- La instancia *Venta* crea una instancia de *Pago.* También se observan claramente los mensajes, líneas de enlace, objetos y parámetros.

5.2. Asignación de responsabilidades de los objetos

A continuación se debe asignar al diagrama de colaboración las responsabilidades de los objetos, para ello se sabe que en el contexto donde se deben considerar estas responsabilidades siempre es durante la creación de estos diagramas. Para hacerlo existe un conjunto de patrones de asignación de responsabilidades de los objetos llamado **GRASP** de las siglas *General Responsability Assignment Software Pattern*, éstos constituyen un apoyo para la enseñanza y el entendimiento del diseño de objetos.

Un patrón es una descripción de un problema y su solución, es decir, es la pareja del problema y su solución, con un nombre aplicable a otros contextos y con la sugerencia respectiva en caso de utilizarse en situaciones nuevas.

El objetivo de los patrones intenta codificar el conocimiento, las expresiones y los principios ya existentes.

Los cinco patrones de GRASP son los siguientes:

Patrón GRASP	Descripción
Experto en Información	Expresa la intuición común de que los objetos hacen las cosas relacionadas con la información que tienen. El experto asigna una responsabilidad a la clase que tiene la información para realizarla. El cumplimiento de esta responsabilidad requiere de información dispersa por diferentes clases de objetos, por lo que será necesario el apoyo de varios expertos que colaboran en la tarea.
Creador	Asigna a la clase B la responsabilidad de crear una instancia de la clase A si se cumple uno o más de los casos: B agrega objetos a A, B contiene objetos de A, B registra instancias de objetos de A, B utiliza objetos de A, B tiene datos de inicialización que se pasarán al objeto A. Este patrón tiene la intención de encontrar un creador que necesite conectarse al objeto creado, favoreciendo al bajo acoplamiento.
Bajo Acoplamiento	El acoplamiento es una medida de la fuerza con que un elemento depende de otros elementos (clases, subsistemas, sistemas, etc.) puede ser su relación o su conocimiento. El objetivo es evitar que los cambios efectuados a una clase no tengan efectos en otras, facilite la comprensión de clases y facilite la reutilización de las mismas. La intención del patrón es manejar un bajo acoplamiento en las decisiones de diseño.

Alta Cohesión	La cohesión es una medida de la fuerza con la que se relacionan las responsabilidades de un elemento, un elemento que tiene responsabilidades altamente relacionadas y que no hace gran cantidad de trabajo, tiene alta cohesión. Una clase de baja cohesión es difícil de entender, de reutilizar, de mantener y son afectadas por los cambios. La intención del patrón es manejar una alta cohesión en las decisiones de diseño.
Controlador	Asigna la responsabilidad de recibir o de manejar un mensaje del sistema a una clase que representa el sistema en global, o un escenario de caso de uso en el que tiene lugar el evento del sistema. La intención del patrón es guiar para identificar los eventos del sistema.

Cuadro. 5.3. Patrones GRASP

Para ejemplificar tenemos los siguientes patrones expertos o parejas de problema/solución, considerados en el diagrama de colaboración del cuadro 5.3

Problema: ¿Quién es el responsable de calcular el total de la venta?
Solución: Debe examinarse el Modelo conceptual.

Problema: ¿Qué información se requiere para calcular el total?
Solución: Hay que conocer todas las instancias ventas Línea De Producto de una venta, la suma de sus subtotales.

Con estos elementos, se puede asignar responsabilidades a las clases involucradas como venta, ventaLíneadeProducto y EspercificacióndeProducto como sigue:

Clase	Responsabilidad
Venta	Conoce el total de la venta
VentasLíneadeProducto	Conoce el subtotal de la línea de producto.
EspecificacióndeProducto	Conoce el precio del producto.

Que visto en el diagrama de colaboración sería de la siguiente forma:

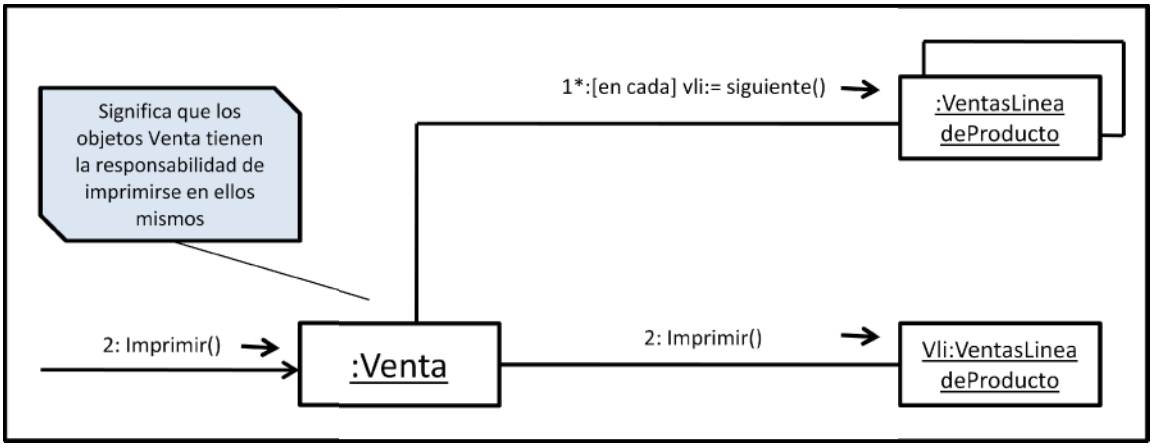


Figura. 5.3. Asignación de responsabilidades en diagrama de colaboración

5.3. Determinación de la visibilidad entre objetos

Visibilidad es la capacidad que puede tener un objeto de *ver* a otro, o de tener una referencia a otro. Es decir, para que un objeto pueda enviar un mensaje a otro receptor este debe ser visible al emisor, por ello se le asocia una referencia o un apuntador. Se debe considerar este aspecto durante el diseño ya que es necesario asegurar la visibilidad adecuada para soportar la interacción de mensajes.

Por ejemplo, el mensaje *getEspecificación* enviado desde un registro hacia un *CatalogodeProductos*, implica que la instancia *CatalogodeProductos* sea visible a la instancia registro, una manera de representarlo es la siguiente:

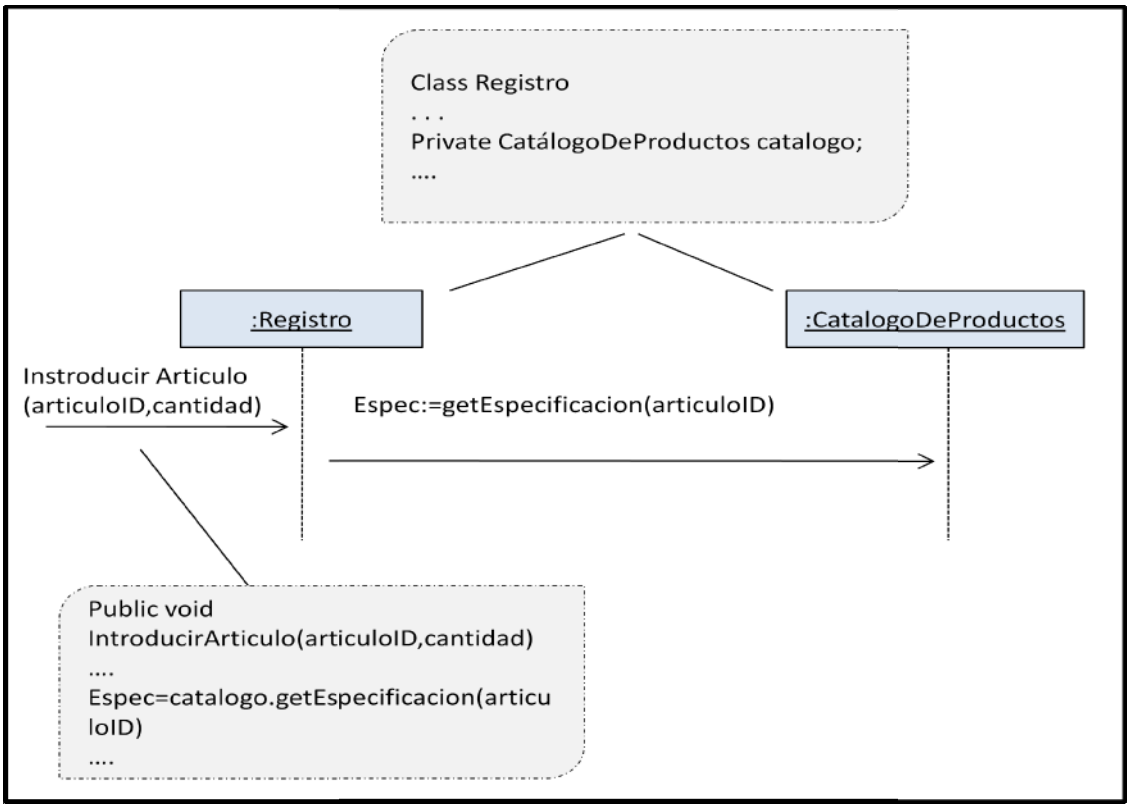


Figura. 5.4 Se requiere visibilidad desde el Registro al *CatálogoDeProducto*s

El cuadro superior define la clase en lenguaje Java para *Registro,* y de igual forma se define la clase *CatalogoDeProductos*, y el recuadro inferior define el método *introducirArticulo* en donde se observa que esté requiere ejecutar el método *getExpecificacion*, por lo que entonces, se dice que este objeto requiere ver el objeto Registro, por ello se realiza una instanciación a él, y con esto será visible donde será utilizado.

Si entonces, se entiende por visibilidad la capacidad de “ver” desde un objeto A o tener referencia a otro objeto B, existen cuatro formas de alcanzar esta visibilidad y son las siguientes:

Tipo de Visibilidad	Descripción
de atributo desde A a B	Existe cuando B es un atributo de A.
de parámetro desde A a B	Existe cuando B se pasa como parámetro a un método de A.
Local desde A a B	Existe cuando B se declara como un objeto local en un método de A.
Global de A a B	Existe cuando B es global de A

Cuadro 5.4. Cuatro formas de alcanzar la visibiliudad

UML incluye la siguiente notación para representar la visibilidad en un diagrama de colaboración.

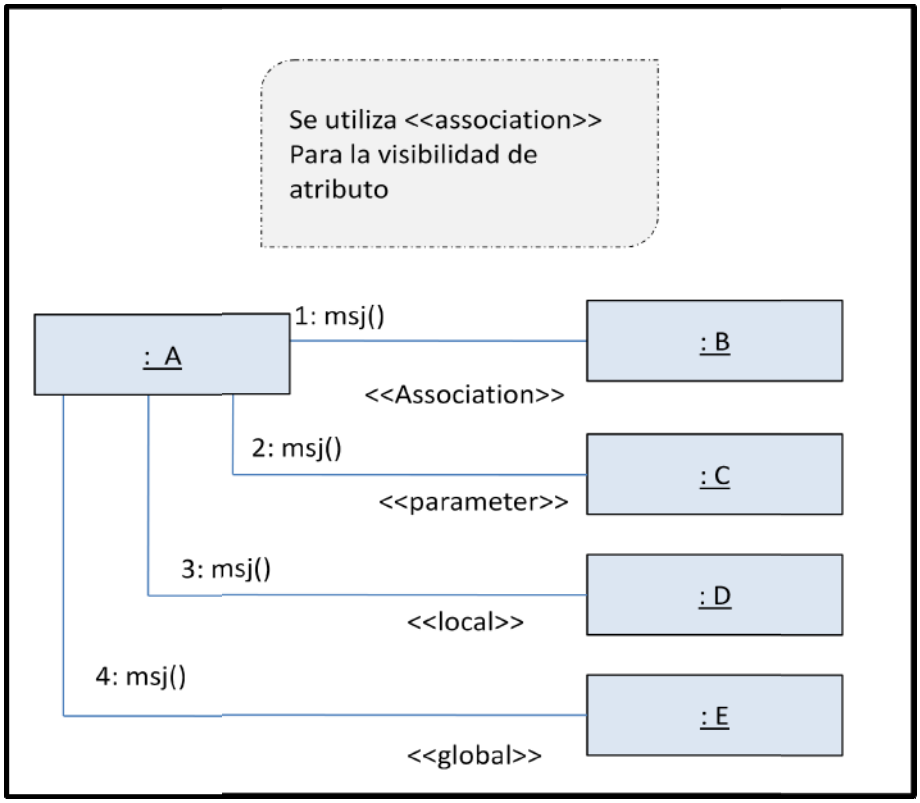


Figura 5.5 Implementación de los estereotipos para la visibilidad en diagramas de colaboración

5.4. Diagramas de clases del diseño

Además de representar mediante un conjunto de diagramas de clases el modelo de dominio, los diagramas de clases del diseño muestran también las asociaciones entre las clases y los atributos entre ellas, además de mostrar las definiciones de las clases de software en lugar de los conceptos del mundo real.

En el proceso unificado no se define de manera específica ningún artefacto denominado “diagrama de clases de diseño”; Sino que define en el modelo de diseño varios diagramas, entre ellos, diagramas de interacción, de paquetes y los de clase.

Este tipo de diagramas son llamados diagramas estáticos, por que muestran las diferentes clases que componen un sistema, cómo se relacionan las clases con otras, y de cada clase se muestran los métodos y atributos que las componen.

En estos diagramas se puede observar qué clases conocen a otras clases o son partes de ellas, sin mostrar los métodos que las invocan.

La elaboración de estos diagramas se realiza durante el diseño casi siempre de manera simultánea con los diagramas de interacción, ya que durante el diseño se van esbozando muchas clases, nombres de métodos y relaciones.

En estos diagramas se encuentran: clases, asociaciones, atributos, interfaces, métodos, información acerca del tipo de atributos, dependencias, etc.

El **primer paso** para elaborar un diagrama de clases es la identificación y representación de las clases de software, ellas pueden identificarse examinando los diagramas de colaboración y listando las clases (sustantivos) que en ellos se mencionan.

Recordemos que una clase define los atributos y los métodos de una serie de objetos y todos los objetos de esta clase (instancias de esa clase) tienen el mismo comportamiento y los mismos atributos.

El **segundo paso** es dibujar cada clase representándola en un rectángulo con su nombre por encabezado, y dos divisiones más para mostrar los atributos y sus métodos.

Los atributos se muestran con nombre y se puede indicar el tipo de dato, su valor inicial y si es + público, - privado o # protegido.

Las operaciones o métodos también se muestran con su nombre, estos resultan del análisis de los diagramas de colaboración, por ejemplo si se envía el mensaje *crearLineaDeVenta* a una instancia de la clase *Venta*, entonces en la clase *Venta* deberá existir un método llamado *crearLineaDeVenta.* Estas operaciones pueden mostrar sus parámetros y valores de retorno, al igual que los atributos se puede indicar si es operación + pública, - privada o # protegida.

Por ejemplo, tenemos el siguiente conjunto de clases: *Registro, Venta, EspecificacionDelProducto, CatalogoDeProductos, Tienda y LineaDeVentaPago* que fueron identificadas en cierto diagrama de colaboración, de donde también se identifican sus atributos y métodos.

La representación correspondiente en UML es la siguiente:

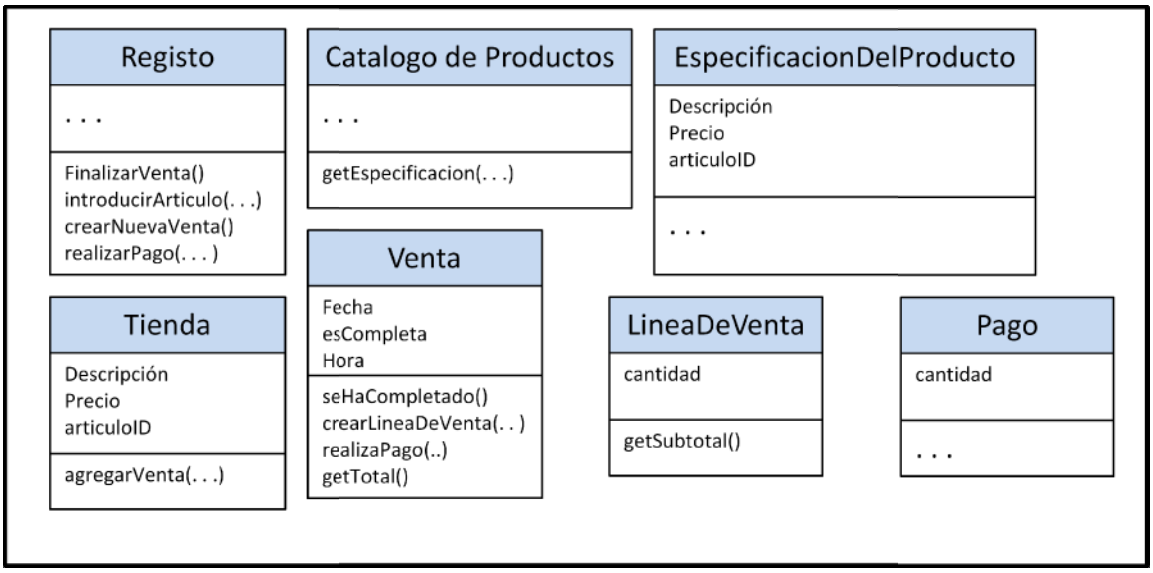


Figura 5.6. Representación UML de las clases identificadas en el diseño con sus atributos y métodos

Observa que en cada clase se tienen tres áreas: la primera es el nombre de la clase, la segunda las propiedades y tercera los métodos.

Finalmente el **tercer paso** es agregar las asociaciones entre clases y su navegabilidad. Recordemos que las asociaciones son mecanismos que permiten la comunicación entre las clases, llamadas también enlaces; estos pueden ser unidireccionales o bidireccionales indicándolo con una línea y su respectiva flecha, esto indicará que pueden intercambiar mensajes entre sí, o es uno de ellos solamente el que recibe información de otro. Representando la multiplicidad en un rango [mín . . máx], de valores positivos y con un asterisco * cuando se desea indicar el infinito en el lado máximo.

5.5 Diseño de la interfaz del usuario

La interfaz es la parte de hardware y software de un sistema informático que facilita al usuario el acceso a los recursos de la computadora, y es justo en este punto que los usuarios determinarán en gran medida su percepción e impresión de lo que será la aplicación.

Para llegar a construir la interfaz se desarrollan prototipos que ilustran cómo pueden utilizar el sistema los usuarios, para ejecutar los casos de uso definidos al inicio del proyecto. El desarrollo de prototipos facilita el desarrollo de sistemas interactivos ya que se llega a entablar una comunicación permanente de retroalimentación con el usuario, de aquí la importancia de este tema.

Una vez hecha la especificación, propuesto el diseño y que el código esté implementado, es muy fácil modificar la vista.

Para preparar interfaces de cara al usuario se disponen esquemas de la configuración de elementos de las interfaces, se bosquejan elementos adicionales necesarios para cambiar varios elementos de interfaces de usuario como son: ventanas, carpetas, herramientas, controles, etc. Y se constituyen prototipos ejecutables. Se realizan pruebas por cada actor que participa en los casos de uso y se efectúa una revisión y validación que puede hacerse de manera superficial y corregirse posteriormente durante el diseño, además debe verificarse que a cada actor le sea útil, le proporcione una vista agradable, tenga una forma consistente, y cumpla con los estándares como color, tamaño, botones, etc.

El diseño de esta interfaz de usuario implica conocer algunos aspectos tales como: Habilidades físicas y sensoriales (adaptación del entorno de trabajo), habilidades cognitivas (capacidad de razonamiento de acuerdo con el grado de experiencia en el proceso como en el uso de la computadora), diferencias de personalidad (tímidos, cautelosos, extrovertidos, etc.) y diferenciación cultural (lenguaje, expresiones, terminología, etc.).

Los principios del diseño para la interfaz de usuario son los siguientes:

Principio	Descripción
Uso equitativo	El diseño deberá ser usable y de un precio razonable.
Uso flexible	El diseño debe acomodarse a un rango de personas con distintos gustos y habilidades.
Uso simple e intuitivo	El diseño debe ser simple y fácil de entender.
Información perceptible	El diseño comunica la información efectivamente al usuario.
Tolerancia al error	El diseño debe minimizar posibles incidentes y evitar consecuencias adversas.
Esfuerzo físico mínimo	El diseño debe poderse usar confortablemente con un mínimo de esfuerzo.
Tamaño y espacio para poder aproximarse y usar el diseño	El diseño ha de tener un espacio y tamaño apropiados.

Cuadro 5.5. Principios del diseño para la interfaz de usuario.

En el diseño de la interfaz deben considerarse tres aspectos: el contenido, el formato y la interacción con el usuario. Existen ciertos elementos que deben existir en el diseño, pero que no necesariamente todos los tienen: Cuadros de diálogo, menús, pestañas de propiedad, barras de herramientas, asistentes, ventanas, entorno de escritorio, cursores, solo por mencionar algunos.

5.6. Elección del lenguaje, manejador de bases de datos y plataforma

Una vez hecho el diseño de la interfaz, se debe proceder a la elección del lenguaje de programación a utilizar, para ello hay que tomar en cuenta tres puntos importantes:

1) Que el lenguaje a seleccionar esté basado en objetos, en clases y sea capaz de manejar el concepto de herencia de clases.
2) Analizar la naturaleza de nuestra aplicación, el grado de dificultad del problema, el tipo de procesamiento que usará, definir si es por lotes o por líneas, la facilidad con que se le dará mantenimiento al programa y el tipo de problema, es decir identificar si es de aplicación administrativa, de negocios o científica;
3) Verificar el impacto que pueda tener nuestra aplicación, identificar si ¿encajará con otros estándares de sistemas, o resultará aceptable para el personal?, etc.

Una vez analizados y contemplados estos aspectos será más fácil seleccionar el lenguaje de programación más apropiado para desarrollar nuestra aplicación.

El siguiente cuadro muestra algunos aspectos importantes en diferentes lenguajes de programación orientados a objetos, con el fin de que sirva como una guía para seleccionar el mejor lenguaje para nuestros desarrollos. La manera de hacerlo es comparar los aspectos de cada uno contra la tarea que estamos realizando.

Lenguaje	Características
C++	-Se creó con el fin de extender el lenguaje de programación C, con mecanismos que permitan la manipulación de objetos, por lo que se considera desde el punto de vista de orientación a objetos como un lenguaje híbrido. -Es considerado un lenguaje multiparadigma ya que une la programación estructurada y programación orientada a objetos.

Visual Basic	-Está diseñado para generar de manera productiva aplicaciones con seguridad de tipos y orientados a objetos. -Lenguaje de programación llamado “visual” lo que hace referencia al método que se utiliza para crear la interfaz gráfica de usuario. -Se ha considerado como el lenguaje adecuado para principiantes en el mundo de la programación. -Ofrece una manera fácil y rápida de crear aplicaciones basadas en .NET framework. - Para ejecutar Visual Basic, necesitamos disponer de cierto hardware y software instalado en el equipo.
Java	-Lenguaje de programación simple. -Es un lenguaje multiplataforma, es decir, el mismo código Java que funciona en un sistema operativo, funcionará en cualquier otro sistema operativo que tenga instalada la máquina virtual java. -Funciona en redes computacionales Heterogéneas. -El lenguaje fue diseñado con las siguientes características: Simple, Robusto, Seguro, Sintaxis Familiar a C o C++, Portable, Independiente a la arquitectura, Multihilos, Interpretado, Dinámico, distribuido.
SmallTalk	-Considerado el primer lenguaje orientado a objetos. -Tipos dinámicos. -Interacciones entre objetos mediante el envío de mensajes. -Herencia simple y con raíz común. -Recolección de basura. -Compilación en tiempo de ejecución. -Múltiples implementaciones. -Su entorno o ambiente primordialmente es gráfico. -La forma de programar no es el ciclo típico de editar texto,

	compilar y ejecutar, sino que se manipula el entorno al que se llama Navegador del sistema. -La sintaxis tiende a ser minimalista, es decir, posee un grupo pequeño de palabras reservadas y declaraciones en comparación con la mayoría de los lenguajes populares. Las cinco palabras reservadas son: self, super, nil, true y false.
Power Builder	-Realizado por POWERSOFT. -Este sistema de desarrollo de aplicaciones para un ambiente cliente/servidor de Windows. -Soporta varias bases de datos, incluyendo DB2 y ORACLE, y además está empaquetada con la base de datos WATCON SQL. -Provee herramientas de programación visuales igual que el lenguaje de programación POWERSCRIPT. El soporte para Macintosh, Windows NT y UNIX.

Cuadro 5.6. Diferentes lenguajes de programación Orientados a Objetos

Ahora bien, para la elección de la base de datos depende de varios aspectos:
Para qué se va a utilizar, conocer los requerimientos comerciales, la población de usuarios, las habilidades del personal de soporte, del servidor de hardware que se usará, del presupuesto, la cantidad de datos a manejar, el tipo de aplicaciones por soportar, soporte de nuevas tecnologías y copias de seguridad.

El objetivo de poder **seleccionar la plataforma** es realizarlo de la forma más cercana a las especificaciones funcionales y técnicas, para esto se debe llevar a cabo las siguientes dos actividades: el **análisis y selección del hardware**, y el **análisis y selección del software**.

Respecto del hardware se debe seleccionar una máquina que permita cumplir con los requerimientos especificados: la velocidad de procesamiento, posibilidad de multitarea, seguridad ante caídas de la red, posibilidades de restauración, el tipo del sistema operativo a utilizar, la memoria en ram disponible, la capacidad de almacenamiento, el número de ranuras de conectividad, etc.

Y por su parte en software se debe atender dos aspectos, cuántas de las necesidades de información y requerimientos quedan cubiertos, y cuántas no, este dato nos ayudará a conocer el costo adicional necesario para cubrir el total de los requerimientos.

5.7. Documentación

El desarrollo de un sistema de software lleva consigo no sólo las líneas de código, sino que también debe proporcionar ciertos productos para que los usuarios tengan un mejor conocimiento sobre el proyecto. Al igual se busca dejar constancia de las decisiones tomadas en el análisis y diseño para el mantenimiento posterior del sistema.

Los productos de desarrollo orientado a objetos incluyen: diagramas de clases, diagramas de objetos, diagramas de módulos y diagramas de procesos. En conjunto sirven para remontarse a los requerimientos del sistema.

Los productos del proceso de diseño son un conjunto de documentación dirigida a clientes y usuarios en lenguaje natural, donde se describe qué es lo que el sistema hará, y en una segunda parte, se utiliza una documentación técnica que describe la estructura del sistema, datos y funciones.

En general la documentación esencial debe incluir: documentación de la arquitectura de alto nivel del sistema, documentación de las abstracciones y mecanismos de arquitectura y documentación de los escenarios que ilustran el funcionamiento práctico.

Debe contener descripciones de los componentes del sistema, la forma en que interactúan los usuarios con el sistema en donde se debe considerar lo siguiente:

- menús y otros formatos en pantalla,
- interfaces: teclas de función, descripciones de pantalla, esquemas de teclados y uso del ratón formatos de los reportes,
- entradas: información de datos como se dan, en qué formato y el medio donde son almacenados,
- salidas: dónde se envían los datos, cómo se les da formato y el medio donde son almacenados y procedimientos de archivo.

Si el sistema es distribuido, la configuración deberá detallarse presentando la topología de la red y posibles recomendaciones de integridad.

Finalmente se debe realizar un cruce del diseño frente a los requerimientos, para demostrar cómo el diseño ha sido efectivamente derivado de ellos.

Actividades de aprendizaje

A.5.1 Elabora el diagrama de colaboración del caso de uso “valida clave de usuario” para el acceso a cualquier sistema; para ello, realiza las siguientes dos actividades:

5.1.1 Realiza el diagrama colocando los objetos participantes en la colaboración (nodos), y los enlaces que conectan los objetos.

5.1.2. Representa los mensajes que envían y reciben los objetos, asigna responsabilidades y determina la visibilidad de los objetos.

A.5.2 Elabora un diagrama de clases para el sistema que administra las ventas de una empresa de Chocolates. Para ello, realiza las siguientes cuatro actividades:

5.2.1 Realizar una lista de posibles requerimientos en el diseño de un sistema que controla las ventas de una empresa.

5.2.2 Elaborar una lista de las posibles clases, identificándolas en la lista de requerimientos.

5.2.3 Dibujar las clases según UML con Nombre y encabezado, atributos y métodos.

5.2.4 Representa las asociaciones y los enlaces entre clases.

Cuestionario de autoevaluación

1. ¿Cuáles son las principales acciones a realizar durante el proceso de diseño orientado a objetos?
2. ¿Qué información permite representar un diagrama de colaboración durante el diseño de un sistema orientado a objetos?
3. ¿Cuál es el contenido de los diagramas de colaboración?
4. Explica cómo se construye un diagrama de colaboración
5. ¿Cómo se lleva a cabo el proceso de asignar responsabilidades a los objetos durante la elaboración de los diagramas de colaboración?
6. ¿Define que es la visibilidad de un objeto y por qué se considera importante?
7. Describe brevemente los tres pasos a seguir para elaborar un diagrama de clases del diseño.
8. Menciona algunos de los principios del diseño para la interfaz de usuario.
9. ¿Por qué es tan importante diseñar la interfaz del Usuario?
10. ¿Cuáles son las consideraciones más importantes para seleccionar el lenguaje de programación?
11. ¿Cuáles serían los aspectos más importantes en que se debe poner atención a la hora de elegir la base de datos a utilizar en el diseño de un sistema orientado a objetos?

Examen de autoevaluación

Relaciona las columnas escribiendo dentro del paréntesis el número que una los conceptos con sus respectivas definiciones

Concepto	Definición	
1. Diagrama de Colaboración	a. Colocar los objetos que participan como nodos del grafo, representar mediante arcos los enlaces entre objetos y por último incluir los mensajes que envían y reciben los objetos.	( )
2. Describe el proceso de Elaboración de un diagrama de Colaboración	b. Para seleccionarlo se debe tomar en cuenta: que esté basado en objetos, validar la naturaleza de la aplicación y verificar el impacto que puede tener nuestra aplicación.	( )
3. Elementos en un Diagrama de Colaboración	c. Ilustra las interacciones entre los objetos en un formato de grafo o red, en él se destaca la organización estructural de los objetos que envían y reciben mensajes.	( )
4. Cohesión	d. Clases, Asociaciones, atributos, interfaces, métodos, dependencias etc.	( )
5. Patrones GRASP	e. En ella se describen los aspectos de requerimientos, análisis, diseño, implementación y pruebas del sistema. Así como una descripción funcional de lo que puede hacer el sistema, cómo instalarlo, etc.	( )
6. Formas de alcanzar la visibilidad de un objeto	f. Experto en información, Creador, Bajo Acoplamiento, Alta Cohesión y Controlador.	( )

7. Documentación del sistema	g. Son principios útiles en su diseño: Uso Equitativo, Uso Flexible, Uso intuitivo, Tolerancia al error, esfuerzo físico mínimo, etc.	( )
8. Son elementos que encontramos en Diagramas de Clases del diseño	h. Medida de fuerza con la que se relacionan las responsabilidades de un elemento. En una clase que tiene baja esta medida, es difícil de entender, reutilizar, de mantener y son afectadas por los cambios. Por eso se requiere que en los desarrollos de sistemas esta sea Alta.	( )
9. Diseño de la interfaz de Usuario	i. Enlaces, Mensajes, objetos e iteraciones	( )
10. Elección del lenguaje de programación	j. de Atributo desde A a B, de parámetro desde A a B, Local desde A a B, Global desde A a B.	( )

Bibliografía básica

Barranco de Areba, Jesús. (2001). *Metodología del análisis estructurado de sistemas*, Madrid, Universidad Pontificia Comillas de Madrid.

Booch, Grady. (1996). *Análisis y diseño orientado a objetos*, con aplicaciones, 2ª ed., México: Addison Wessley Longman.

Booch Grady, James Rumbaugh, Ivar Jacobson. (1999). *El lenguaje Unificado de Modelado*, 2ª edición, Madrid, Addison Wessley.

Chulvi, Vicente; María Sánchez Mora. (2009). *Modelo informal basado en idef4 para la representación de diseños basados en el esquema FBS,* XIII Congreso Internacional de Ingeniería de Proyectos, Badajoz, pp. 1802-1810, disponible en línea: http://aeipro.com/index.php/remository/congresos/congresos_badajoz2009/congresos_badajoz2009_08/MODELO-INFORMAL-BASADO-EN-IDEF4-PARA-LA-REPRESENTACION-DE-DISE%C3%91OS-BASADOS-EN-EL-ESQUEMA-FBS recuperado el 12/11/09.

García Molina, Jesús, (2007), "De los procesos del negocio a los casos de uso", Universidad de Murcia; en *Técnica Administrativa*, Vol. 6, N° 4, octubre/diciembre, Bs. As. Disponible en línea: http://www.cyta.com.ar/ta0604/v6n4a1.htm#1, recuperado el 16/11/09.

Gómez, Cristina, *et al*. (2003). *Diseño de sistemas software en UML.* Barcelona: UPC.

Graham, Ian. (1996). *Métodos orientados a Objetos,* 2ª ed., México: Addison Wessley Longman.

IDEF4: http://www.idef.com/IDEF4.html

Lambin, Jean Jacques. (2003). *Marketing estratégico*, Madrid, Escuela Superior de Gestión Comercial y Marketing.

Larman, Craig. (1999). *Análisis y diseño orientado a objetos con UML*, México: Pearson.

_____ (2006). *UML y patrones: Una introducción al análisis y Diseño Orientado a Objetos y al Proceso Unificado*, 2ª ed., México: Prentice Hall.

Letelier Patricio. (s/f). Proceso de desarrollo de software, Universidad Politécnica de Valencia; Departamento de Sistemas Informáticas y Computación, disponible en línea: http://www.dsic.upv.es/asignaturas/facultad/lsi/doc/IntroduccionProcesoSW.doc, recuperado el 22/10/09.

Martínez, Ángel, José Hilario Canós y Jesús Damián García-Consuegra. (Octubre 2010). Estudio del Estado del Arte en Modelado y Ejecución de Procesos Interorganizacionales, Albacete: Departamento de Informática, Escuela Politécnica Superior, Universidad de Castilla-la Mancha, disponible en línea: http://www.info-ab.uclm.es/descargas/thecnicalreports/DIAB-01-06-21/diab-01-06-21.pdf

Softtlan. (2007). “Tutorial de casos de uso”, 1703/07, blog disponible en línea: http://softtlan.blogspot.com/2007/03/tutorial-de-casos-de-uso-paso.html, recuperado el 12/11/09.

Sommerville, Ian. (2005). *Ingeniería de Software*, 7ª edición, Madrid: Pearson Education.

Teniente López, Ernest, Dolors Costal Costa, Sancho Samsó, M. Ribera. (2003). *Especificación de Sistemas Software en UML*, Barcelona: UPC.

Weitzenfeld, Alfredo. (2004). *Ingeniería de software orientada a objetos con UML, Java e Internet*, México: Thomson International.

Zuñiga Pakoz: "Procesos de la Ingeniería de Requerimientos", Mitecnológico, disponible en línea: http://www.mitecnologico.com/Main/ProcesosDeLaIngenieriaDeRequerimientos, recuperado el 12/11/09.

RESPUESTAS A LOS EXÁMENES DE AUTOEVALUACIÓN

INFORMÁTICA V

	Tema 1	Tema 2	Tema 3	Tema 4	Tema 5
1.	c	h	e	d	c
2.	f	c	j	f	a
3.	a	i	h	i	i
4.	h	a	g	g	h
5.	i	b	i	a	f
6.	j	g	c	j	j
7.	d	j	d	c	e
8.	e	d	a	e	d
9.	g	e	b	b	g
10.	b	f	f	h	b