Universidad Nacional Autónoma de México
Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia

Licenciatura en Informática

Informática IV. Análisis y Diseño Orientado a Objetos

Apunte electrónico

SUAyED

COLABORADORES

DIRECTOR DE LA FCA
Dr. Juan Alberto Adam Siade

SECRETARIO GENERAL
L.C. y E.F. Leonel Sebastián Chavarría

COORDINACIÓN GENERAL
Mtra. Gabriela Montero Montiel
Jefe de la División SUAyED-FCA-UNAM

COORDINACIÓN ACADÉMICA
Mtro. Francisco Hernández Mendoza
FCA-UNAM

AUTORES
Mtro. Hugo Díaz García
Mtro. Rene Montesano Brand

DISEÑO INSTRUCCIONAL
Mtro. Mario Gilberto Ramírez Varela

CORRECCIÓN DE ESTILO
Mtro. Carlos Rodolfo Rodríguez de Alba

DISEÑO DE PORTADAS
L.CG. Ricardo Alberto Báez Caballero
Mtra. Marlene Olga Ramírez Chavero
L.DP. Ethel Alejandra Butrón Gutiérrez

DISEÑO EDITORIAL
Mtra. Marlene Olga Ramírez Chavero

OBJETIVO GENERAL

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

TEMARIO OFICIAL
(64 horas)

		Horas
1. Introducción		12
2. Metodologías orientadas a objetos		14
3. Planeación y elaboración		12
4. Análisis orientado a objetos		14
5. Diseño orientado a objetos		12
	Total	**64**

INTRODUCCIÓN

A lo largo de los años ha cambiado la forma de realizar el análisis y diseño de sistemas, hasta lo que hoy son los paradigmas y metodologías orientadas a objetos que nos permiten modelar un problema y entender, a partir de diferentes diseños de vistas, los procesos y actividades que se realizan en una empresa. En este apunte se desarrollan los temas que te permitirán entender y realizar el análisis y diseño orientado a objetos.

En la primera unidad, *Introducción*, abarcaremos los conceptos y principios fundamentales para entender el paradigma *orientado a objetos*; de la misma manera se conocerá el ciclo de vida de un sistema, las etapas para construirlo y el proceso de desarrollo del software.

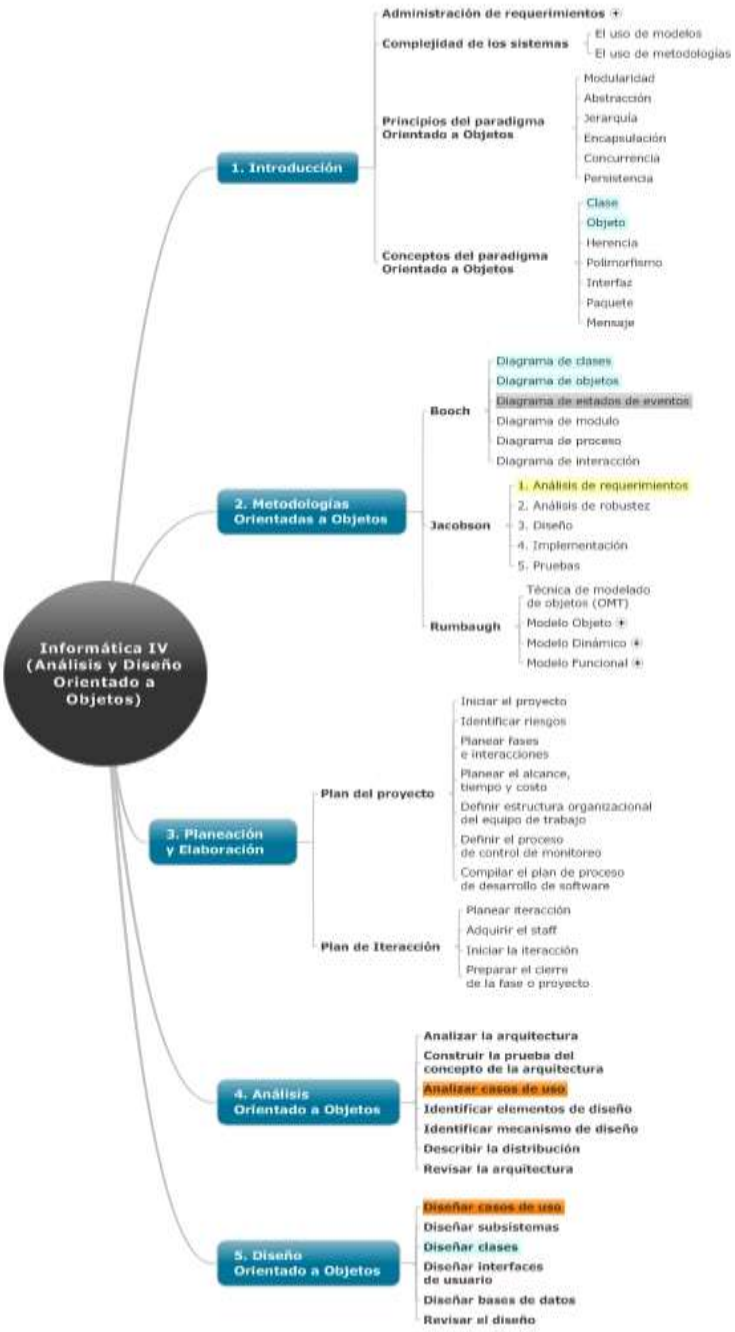
En la segunda unidad, *Metodologías orientada a objetos*, se describen las aportaciones metodológicas de quienes dieron origen a UML, los famosos tres amigos. De igual forma se verá el método de Grady Booch, a través de sus 4 modelos y 6 anotaciones, Ivar Jacobson con el método de Objectory y, finalmente, la técnica de modelado de objetos (OMT) de James Rumbaught.

En la tercera unidad, *Planeación y elaboración*, se estudiará la clasificación y cualidades de los requerimientos, fuentes de obtención de los mismos, la administración de los requerimientos, y cómo se realiza la especificación de un requerimiento a través de la descripción de un caso de uso, conociendo los rubros que lo componen.

En la cuarta unidad, *Análisis orientado a objetos*, se estudia la definición de la arquitectura para el dominio o contexto del negocio, cómo generar los diagramas de colaboración a partir del listado de pasos de un caso de uso, cómo entender los subsistemas y desarrollar los diagramas de clases.

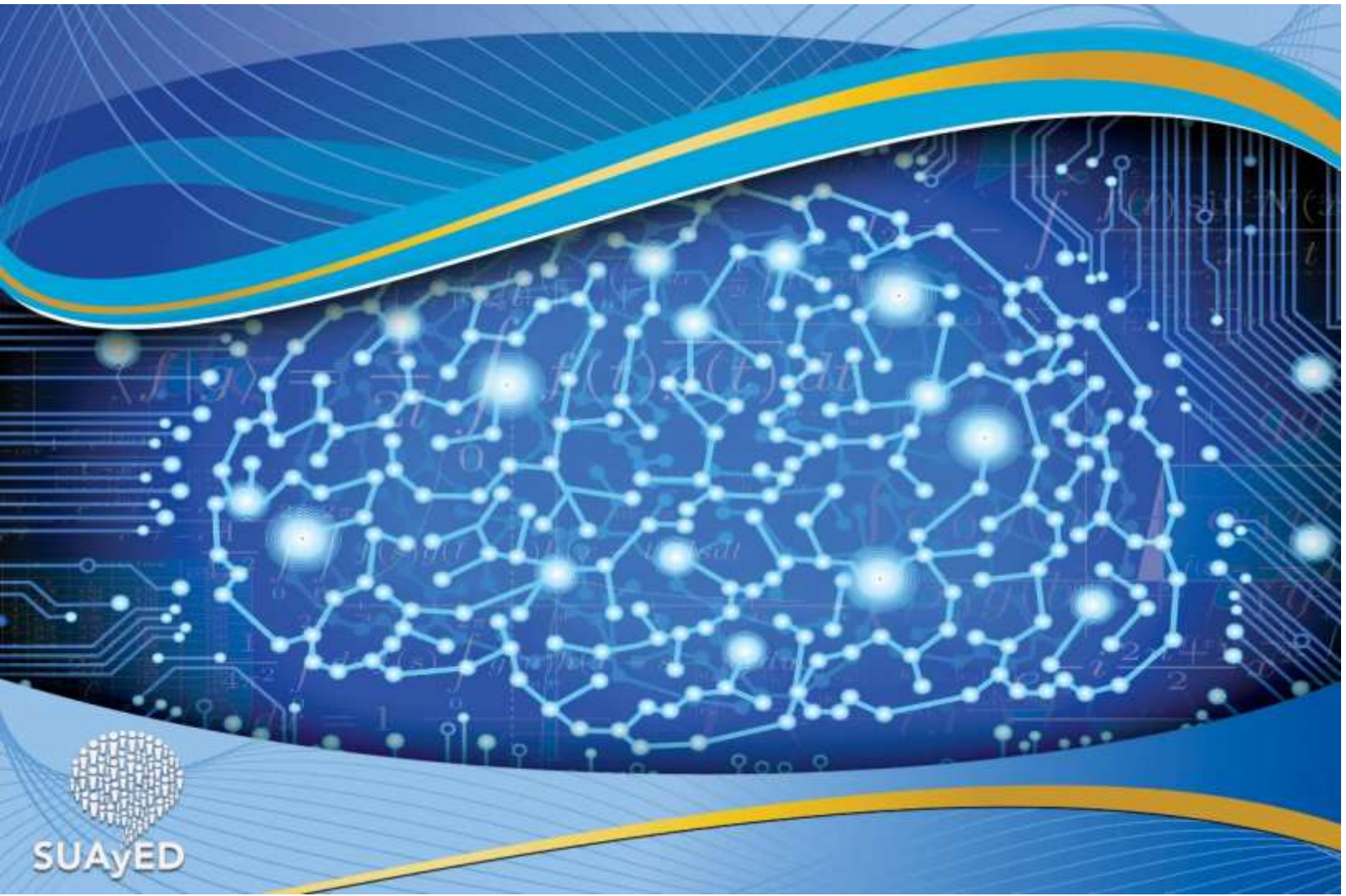
En la última unidad, *Diseño orientado a objetos*, se estudian los componentes y diseños de interfaz de usuario, cómo se crean los diferentes diagramas (tales como el diagrama de actividades, diagrama de estado y diagrama de secuencia), cómo se unifican las clases y la forma de pasar de una diagrama de clases a un modelo de base de datos.

ESTRUCTURA CONCEPTUAL



Unidad 1.

Introducción



OBJETIVO PARTICULAR

Analizar los conceptos y principios que conforman el paradigma orientado a objetos.

TEMARIO DETALLADO
(4 horas)

1. Introducción

- 1.1. Administración de requerimientos
- 1.2. Complejidad de los sistemas
- 1.3. Principios del paradigma orientado a objetos
  - 1.3.1. Abstracción
  - 1.3.2. Modularidad
  - 1.3.3. Jerarquía
  - 1.3.4. Encapsulación
  - 1.3.5. Concurrencia
  - 1.3.6. Persistencia
- 1.4. Conceptos del paradigma orientado a objetos
  - 1.4.1. Clase
  - 1.4.2. Objeto
  - 1.4.3. Herencia
  - 1.4.4. Polimorfismo
  - 1.4.5. Interfaz
  - 1.4.6. Paquete
  - 1.4.7. Mensaje

INTRODUCCIÓN

En la construcción de sistemas de información, cuando llegan peticiones nuevas es necesario identificar los requerimientos de los clientes que se deben cumplir, para ello es necesario llevar un control a través de la administración de los requerimientos, de tal forma que se puedan identificar las personas involucradas, los requerimientos encontrados y llevar un seguimiento de los cambios realizados. Por esto es necesario algún documento y/o software que lleve la administración de requerimientos.

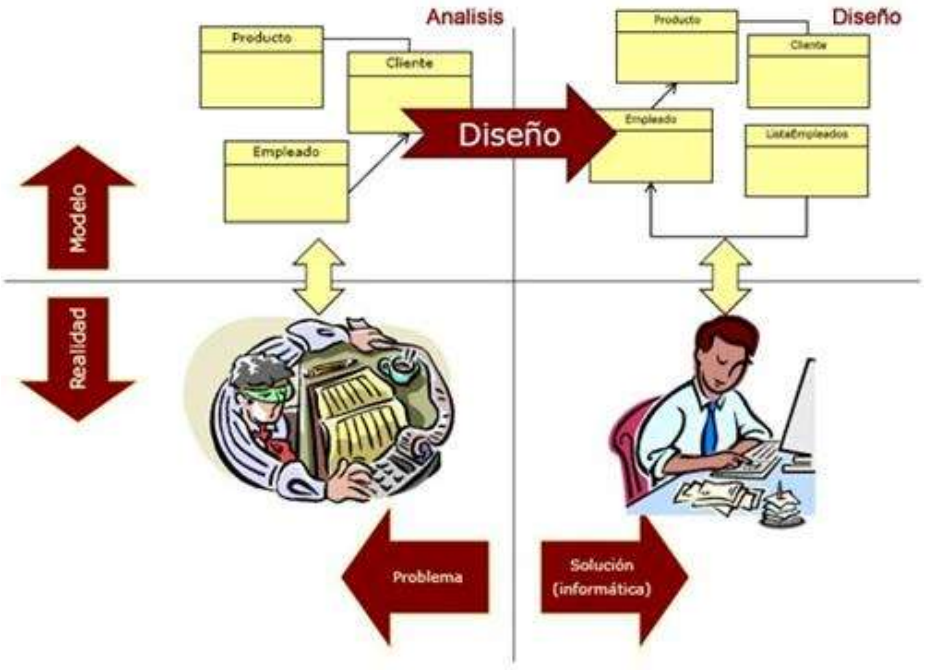
Todos los proyectos tienen cierto nivel de complejidad, mas debido a las limitaciones del cerebro humano es imposible recordar todas las reglas y/o restricciones que se deben de contemplar, por lo tanto, es necesario ayudarse de modelos y metodologías para plasmar todo de forma clara y precisa siguiendo un método que genere los productos (documentos, diagramas, etc.) inteligibles para todos.

Dicho que la capacidad del cerebro humano es limitada para retener gran cantidad de información, a diferencia de una computadora, y por la complejidad en los sistemas, la metodología debe plasmar de modo claro y conciso los resultados generados.

Cuando se analizan y diseñan los sistemas desde la perspectiva de la orientación a objetos, primero se deben conocer y entender los conceptos y principios fundamentales del paradigma orientado a objetos, con la finalidad de realizar la abstracción del mundo real a través de identificar las clases y objetos.

Las clases tienen una jerarquía que va desde clases generales hasta clases especializadas. Los objetos se comunican por medio de mensajes para realizar tareas en común, y no deben permitir el acceso a datos importantes, solo en casos específicos que convenga dar a conocer. De tal manera que corresponde diseñar modularmente las clases, es decir, pensando en la función básica que se obliga realizar un análisis y diseño orientado a objetos.

Figura de Análisis y Diseño de Sistemas



Fuente: http://www.sharesucre.com/manuales/analisis-y-diseno-de-sistemas-parte-i

1.1. Administración de requerimientos

En los proyectos para el desarrollo de sistemas y de software, desde que se inicia el proyecto se identifican ciertos requerimientos, pero con el paso del tiempo y durante el diseño o desarrollo del sistema pueden existir cambios en ellos, sea por la evolución de las necesidades del cliente, por la evolución de la tecnología o por los estándares internacionales. Debido a estas causas, se vuelve indispensable tener un control adecuado de los requerimientos.

Pero ¿qué entendemos por requisito? De acuerdo con la IEEE (830-990), en términos de ingeniería de software se define de la siguiente manera: “Un requisito de software es la capacidad que debe alcanzar o poseer un sistema o componente de un sistema para satisfacer un contrato estándar, especificación u otro documento.”

La guía SWEBOK (SWEBOK, 2013) denomina un requisito de software como la propiedad que debe ser exhibida por el software desarrollado o adaptado para resolver un problema en particular. En este sentido, un requisito/requerimiento presenta una necesidad de un cliente con respecto a un sistema, que le ayudará a resolver un problema en particular, y cuya identificación ayuda en el análisis y diseño del sistema.

De acuerdo con Sommerville (2011: 112-114), la administración de requerimientos se refiere: *al proceso de comprender y controlar los cambios de requerimientos del sistema*, que va desde la definición de los mismos hasta su clasificación y validación para su uso, a fin de minimizar fallas en el sistema por falta de organización/planeación.

Por ello, el Instituto de Ingenieros del Software (SEI, 2013) a fin de ayudar a las organizaciones a desarrollar y mantener productos y servicios de calidad, generó un documento con la integración de modelos de madurez de capacidades**,** *Capability*

Maturity Model Integration, (CMMI, 2013), consistente en un modelo centrado en la aplicación de las buenas prácticas a seguir para desarrollar productos y servicios de calidad, que cumplan con las necesidades de los clientes y los usuarios finales.

Los cinco niveles de madurez de CMMI.

Fuente: http://www.webesfera.com/webcms/index.php?menu=40

En este documento, basado en el nivel de **madurez dos** para la Gestión de Requisitos (REQM) (CMMI, 2010), se indica que vía los requisitos se gestionan y se identifican las inconsistencias con los planes y productos de trabajo del proyecto. Para ello se definen cinco prácticas a realizar:

1. Comprender los requisitos

Desarrollar una comprensión del significado de los requisitos con los proveedores de los requisitos.

Según el estándar IEEE 830-1990 (IEEE, 1998), será importante generar un documento de especificación de requerimientos del software (SRS) basados en una plantilla. Los rubros son los siguientes:

1. Introducción
   1.1. Propósito
   1.2. Alcance
   1.3. Definición, siglas y abreviaturas
   1.4. Referencias
   1.5. Visión general
2. Descripción de conjunto
   2.1. Perspectiva del producto
   2.2. Funcionalidades
   2.3. Características de usuario
   2.4. Restricciones
   2.5. Asunciones y dependencias
3. Requisitos especiales

Esta plantilla de los rubros puede cambiar de acuerdo a las necesidades adicionales y al nivel de detalle que se necesite llegar con el cliente. La biblioteca de IBM tiene una plantilla (IBM, plantilla SRS, 2010). Por su parte, OpenUP de eclipse también tiene su propio formato para la especificación de requerimientos (OpenUP, s.f.).

2. Obtener el compromiso sobre los requisitos

Los requisitos evolucionan a lo largo del proyecto.

A medida que los requisitos evolucionan esta práctica asegura que los participantes del proyecto se comprometen con los requisitos actuales y aprobados, y con los cambios resultantes en los planes, actividades y productos de trabajo del proyecto.

Tabla con la identificación y clasificación de requerimientos (Sandoval, M., y García, M. 2008).

Requirements:	Prioridad	Estado	Dificultad	Asignado a
CUS1: Facturar Entrega Pedido	Baja	Propuesto	Media	
CUS2: Cobro Clientes	Baja	Propuesto	Media	
CUS3: Compra a Proveedores	Media	Aprobado	Media	
CUS4: Confeccionar Catálogo	Media	Aprobado	Media	
CUS5: Consultar Pedidos no Atendidos	Alta	Validado	Baja	José A. Mocholí, Eduardo Bueno
CUS6: Control Estadísticas	Media	Aprobado	Media	
CUS7: Consultar Catálogo	Media	Aprobado	Media	
CUS8: Entrevista Trabajo	Baja	Propuesto	Baja	
CUS9: Gestión Nóminas	Media	Aprobado	Media	
CUS10: Gestión de Personal	Baja	Propuesto	Media	
CUS11: Gestión de Regiones	Media	Aprobado	Media	
CUS12: Otorgar Incentivos	Baja	Propuesto	Baja	
CUS13: Política de Ventas	Baja	Propuesto	Baja	
CUS14: Reabastecer Almacén	Media	Aprobado	Media	
CUS15: Realizar Oferta	Media	Aprobado	Media	
CUS16: Redistribución de Personal	Media	Aprobado	Media	
CUS17: Atender Pedido	Alta	Incorporado	Media	José Antonio Mocholí
CUS18: Cancelar Pedido Atendido	Alta	Incorporado	Media	Eduardo Bueno Medina
CUS19: Consultar Pedidos a Enviar	Alta	Incorporado	Baja	German Mira Rico
CUS20: Elaborar Pedido	Alta	Validado	Alta	Germán Mira y José A. Mocholí
CUS21: Elaborar Pedido On-line	Media	Aprobado	Alta	
CUS22: Gestión de Clientes	Media	Aprobado	Media	
CUS23: Incidencia Pedido	Media	Incorporado	Media	César López Rodríguez
CUS24: Introducir Recibos	Baja	Aprobado	Media	
CUS25: Pasar Pedido a Envío	Alta	Incorporado	Media	Germán Mira Rico
CUS26: Realizar Envio	Media	Aprobado	Media	
CUS27: Reposición de Stock	Media	Aprobado	Media	
<Click here to create a requirement>	Media	Aprobado	Media	

Fuente: La trazabilidad en el proceso de requerimientos de software

3. Gestionar los cambios a los requisitos

A medida que cambian las necesidades y avanza el trabajo, es posible que se tengan que hacer cambios a los requisitos existentes.

La administración de cambios es esencial porque nos ayuda a conocer quién solicita los cambios, quién o quiénes los atenderán y qué impacto tiene en el costo de implementación. Para Somerville (2011: 114) hay tres etapas principales en este proceso:

I. **Análisis del problema y especificación del cambio**. El proceso comienza con la identificación de un problema en los requerimientos o, en ocasiones, con una propuesta de cambio específico.

II. **Análisis del cambio y estimación del costo**. El efecto del cambio propuesto se valora usando información de seguimiento y conocimiento general de los requerimientos del sistema.

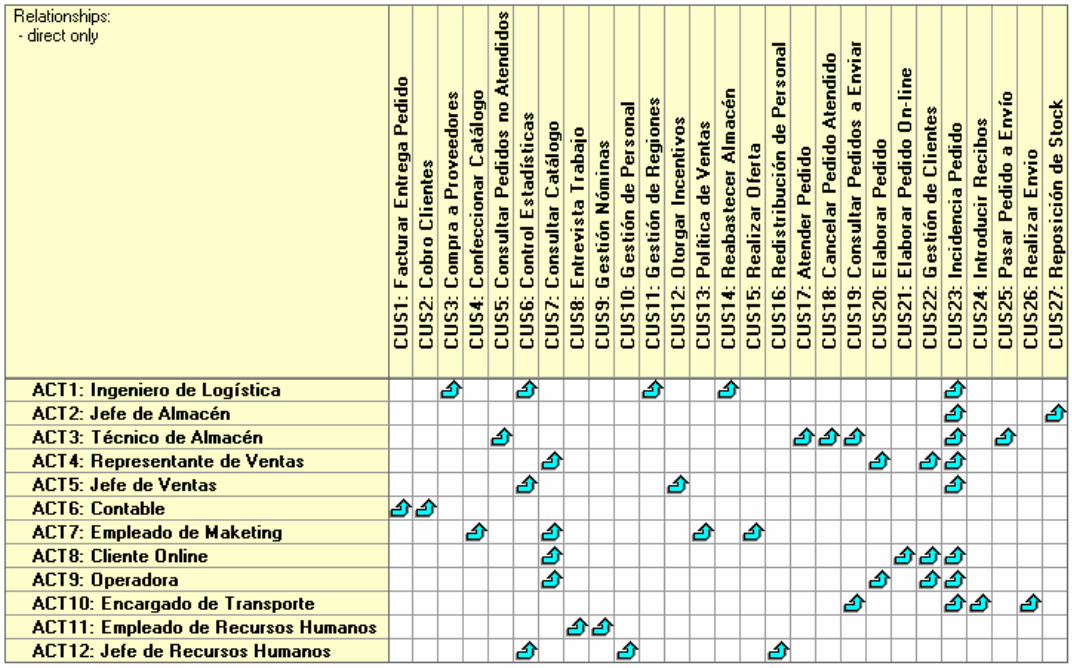
III. **Implementación del cambio**. Se modifican de requerimientos y donde sea necesario, el diseño y la implementación del sistema.

4. Mantener la trazabilidad bidireccional de los requisitos.

La intención de esta práctica específica es mantener la trazabilidad bidireccional de los requisitos.

Cuando se gestionan bien los requisitos, se puede establecer la trazabilidad desde un requisito fuente hasta sus requisitos de más bajo nivel, y desde estos requisitos de más bajo nivel de vuelta hasta sus requisitos fuente. La trazabilidad, en pocas palabras, es identificar el origen o persona que lo solicita y, si hay cambios, quién los solicito, por lo que se dice que deben ser rastreables los requisitos. ¿Quién los propuso?, ¿cuál es la necesidad y relación entre los requisitos?, es decir, ¿cuáles son las dependencias?

Figura de la matriz de trazabilidad, casos de uso y actores (Sandoval, M., y García, M. 2008).

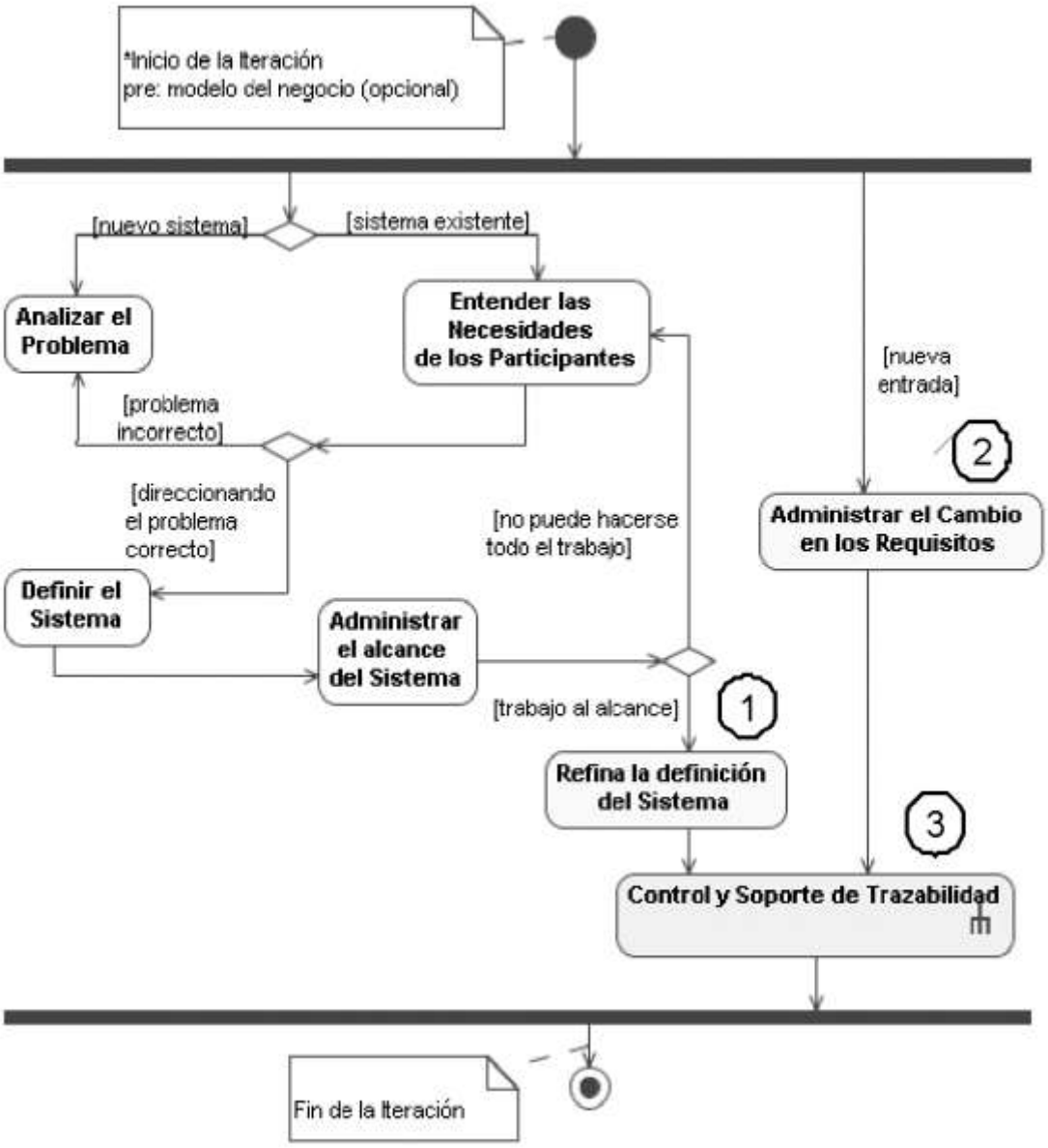


Fuente: La trazabilidad en el proceso de requerimientos de software

5. Asegurar el alineamiento entre el trabajo del proyecto y los requisitos.

Esta práctica encuentra las inconsistencias entre los requisitos, los planes del proyecto y los productos de trabajo, e inicia acciones correctivas para resolverlas. Para Yamal Chamud (2007: 67) ayuda a identificar a los involucrados en nuestros proyectos y concilia sus expectativas, facilita la misión por cumplir y alinea los esfuerzos del equipo.

Diagrama de flujo de trabajo de la disciplina de requisitos (Tabares *et al*., 2007)



Fuente: Revista EIA

Dada la importancia de la correcta gestión de los requisitos de software, es común encontrar en la actualidad varias herramientas que nos ayuden a esta actividad. Dentro de las opciones que podemos encontrar en el mercado están PEquisitePRO, DOORS,

IrqA, MKS Integrity suite, OSRMT (Open Source Requirements Management Tool), RTM Workshop o Caliber RM (Brooks, s.f).

1.2. Complejidad de los sistemas

Los sistemas grandes son demasiados complejos para que una sola persona, por las limitaciones de la mente humana, sea capaz de retener con precisión todos los elementos y características que lo conforman.

La complejidad expresa el grado en que un sistema o componente tiene un diseño o implementación difícil de entender o verificar. En 1972 Edsger W. Dijkstra (Edsger Wybe Dijkstra) señaló que se debe reflexionar en la estructura de las abstracciones necesarias para hacer frente conceptualmente a la complejidad de lo que se está diseñando.

Se puede hablar de dos factores principales que causan esta complejidad (Weitzenfield, 2008: 13):

La complejidad del problema	La complejidad de la solución
Tiene que ver con la funcionalidad que el sistema debe brindar. Cuando mayor sea el número de requerimientos o funcionalidad ofrecida por una aplicación, mayor será el tamaño del sistema, creando sistemas más difíciles de comprender y desarrollar.	Tiene que ver con el diseño del sistema, el cual debe satisfacer la funcionalidad del problema.

Se consideran dos factores relacionados con la complejidad de un sistema: uno estático y otro dinámico.

1. El factor estático corresponde a la funcionalidad que un sistema de software debe ofrecer al ser inicialmente desarrollado.

2. El factor dinámico corresponde a la funcionalidad que varía con el tiempo, en otras palabras, con los posibles cambios en el sistema. Estos cambios pueden ser considerables y, en muchos casos, son la causa de los retrasos y cancelaciones de los proyectos.

El uso de modelos

Cuando nos encontramos con sistemas con una complejidad alta, es necesario manejar modelos, pero ¿qué se entiende por modelo?

Un modelo (Rumbaugh, Jacobson y Booch, 2000: 11-12) *es una representación de algo; generalmente contiene una semántica y una notación y puede adoptar varios formatos que incluyen textos y gráficos*.

Los modelos permiten captar y enumerar los requisitos y el dominio de conocimiento, de forma que todos los implicados puedan entenderlos y estar de acuerdo con ellos. Un modelo de un sistema grande permite ocuparse de la complejidad que es difícil de tratar.

Un modelo puede:

- Abstraer a un nivel que sea comprensible.
- Determinar el impacto potencial de un cambio antes de que se haga, explorando dependencias en el sistema.
- Mostrar cómo reestructurar un sistema y reducir riesgos.

Niveles de los modelos

Los modelos adquieren diversas formas para diferentes propósitos y aparecen en diferentes niveles de abstracción. La cantidad de detalle del modelo debe adaptarse a uno de los siguientes propósitos (Rumbaugh, Jacobson y Booch, 2000: 13-14):

Guías al proceso de pensamiento.

Los modelos de alto nivel construidos al principio de un proyecto, sirven para enfocar el proceso del pensamiento de los participantes y destacar determinadas opciones. Capturan requisitos y representan un punto de partida hacia un diseño del sistema. Los primeros modelos ayudan a los autores a explorar las opciones posibles antes de converger en un concepto de sistema. Conforme progresa el diseño, los primeros modelos son sustituidos por otros modelos más exactos.

Especificaciones abstractas de la estructura esencial de un sistema.

Los modelos en el análisis o las etapas preliminares del diseño se centran en los conceptos y mecanismo claves del probable sistema. El propósito de los modelos abstractos es conseguir que los aspectos de alto nivel estén correctos, antes de abordar los detalles. Estos modelos se piensan para evolucionar en modelos finales.

Especificaciones completas de sistema final.

En esta clase de modelo debe incluir las construcciones para empaquetar el modelo, para la comprensión de la persona y para la conveniencia de la computadora. Estas no son las características de la aplicación misma. En realidad son características del proceso de construcción.

Ejemplos de sistemas típicos o posibles.

Algunos bien elegidos pueden facilitar el entendimiento a las personas, y pueden validar las especificaciones e implementación del sistema. Los ejemplos se deben de utilizar con cierto cuidado.

Descripciones completas o parciales de sistemas.

Un modelo puede ser una descripción completa de un solo sistema, sin referencias externas. Tales modelos tienen conexiones que se deben enlazar a otros modelos en un sistema completo. Como las piezas tienen coherencia y significado, pueden ser combinadas en otras piezas de varias maneras para producir sistemas muy diversos.

¿Qué hay en un modelo?

Los modelos tienen dos aspectos importantes (Rumbaugh, Jacobson y Booch: 15): Información (semántica) y presentación visual (notación).

El **aspecto semántico** explica el significado de cada símbolo y cómo éste debe ser interpretado, ya sea por sí mismo o en el contexto con otros símbolos (Hans-Erik, 2004: 11). Otro elemento en el que se apoya es la sintaxis, el cual se compara al lenguaje natural, en donde es importante para saber cómo decir correctamente las cosas y cómo usar diferentes palabras “símbolos” juntos para conformar una sentencia y conocer cómo pueden ser combinadas dentro del lenguaje de modelado.

La **presentación visual, notación,** es el material gráfico que se ve en los modelos; es la sintaxis del lenguaje de modelado (Martin Fowler, 1999: 5). No agregan significado, más bien ayudan en la presentación, organización y distribución de los elementos dentro del modelo de una manera más comprensible. Por ejemplo, en un diagrama de clases se puede representar la clase, los tipos de asociación y la multiplicidad de los elementos gráficos y símbolos en su conjunto constituyen el modelo.

El uso de una metodología

El término metodología, cuyas forma su raíces griegas son *methodos* “método” y *logos* “estudio o razonamiento”. En otras palabras, es una colección de métodos, prácticas y reglas usados para trabajar en algún campo o disciplina.

Una metodología puede englobar un conjunto de métodos (de análisis, diseño, programación, etc.) que en su conjunto abarcan el ciclo de sistemas. Por lo tanto, una metodología en el desarrollo de sistemas complejos nos ayuda al definir los pasos a seguir para plasmar las ideas, implementarlas y darle un mantenimiento de modo sistemático para administrar un proyecto y llevarlo a cabo con altas posibilidades de éxito, cumpliendo con el objetivo final.

Una metodología de desarrollo de sistemas nos ayuda a manejar la complejidad, ya que es una estrategia que lleva un proceso estandarizado de desarrollo de sistemas que incluye un conjunto de actividades, métodos, modelos, prácticas y herramientas automatizadas que se usan para construir un sistema.

1.3. Principios del paradigma orientado a objetos

El término *paradigma* significa ejemplo o modelo, es una forma de entender y representar una realidad. Se basa en un conjunto de teorías, estándares y métodos, que juntos representan un modo de organizar el pensamiento, es decir, el modo de “ver” el mundo. Cada nuevo paradigma responde a una necesidad real de nuevos modos de afrontar problemas.

En el ambiente de la programación, existen diferentes paradigmas, de los cuales resaltan:

A) **Paradigma funcional**, en el que el lenguaje describe los procesos, por ejemplo LISP, Haskell, ML.

B) Paradigma lógico, como Prolog.

C) Paradigma imperativo (o procedural[1]), tal como C, Pascal.

D) Paradigma orientado a objetos (el cual nos ocupará en éste tema).

El paradigma de *Programación Orientada a Objetos* (POO) se enfoca en la identificación de entidades, su estructura, clasificación y comportamiento dentro del sistema por desarrollar o en el existente. Teniendo esto presente, tras hacer un modelado de un sistema utilizando este paradigma, el analista deberá identificar objetos y clases, de tal forma que se puedan crear relaciones, que reflejarán el trabajo entre los objetos con el fin de generar un modelo de objetos representativos de la realidad del problema o modelo del negocio.

Cuando se realiza el análisis y diseño orientado a objetos se deben de seguir los principios básicos con la finalidad de que nuestro modelado esté correctamente definido bajo el paradigma orientado a objetos, (tales como abstracción, encapsulación, modularidad, jerarquización, herencia, concurrencia y persistencia, que veremos en un momento más).

El *paradigma orientado a objeto* se basa en cuatro principios que constituyen la base de todo desarrollo orientado a objetos. Estos son: la abstracción, la modularidad, la jerarquía y el encapsulamiento. Pero también hay otros elementos que son importantes (aunque no fundamentales) en la POO: concurrencia y la persistencia.

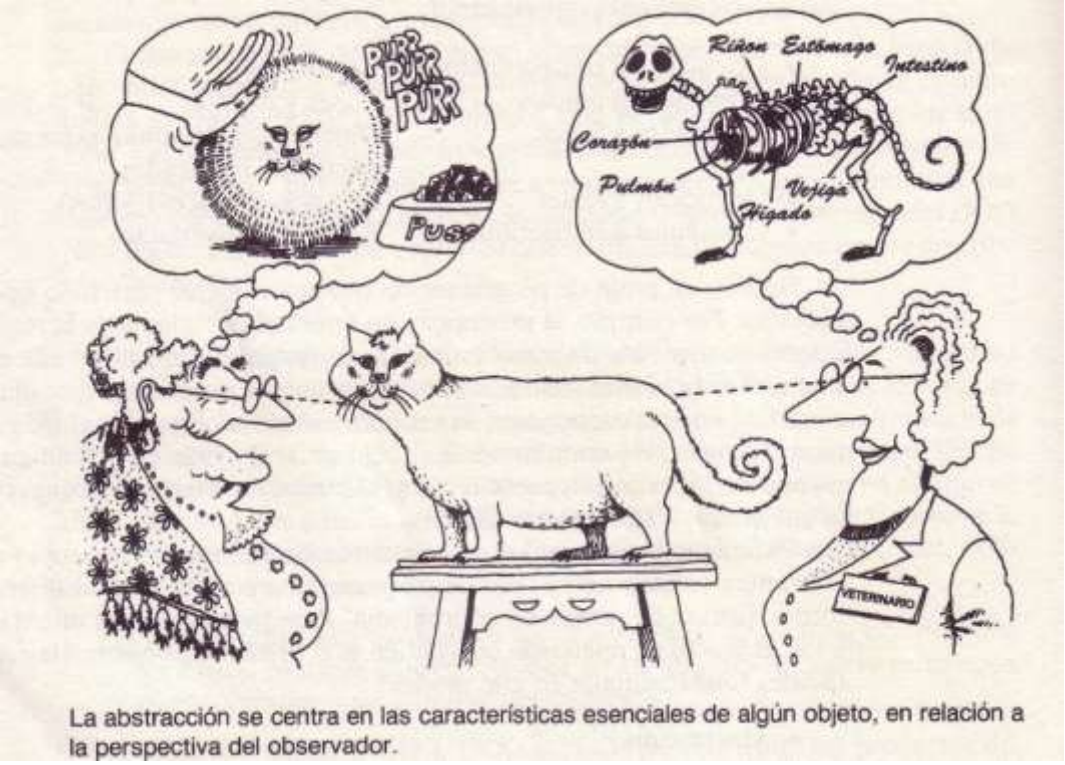
1.3.1 Abstracción

La abstracción (Booch, 1996: 46-47), *denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objeto*. La abstracción no ayuda a

[1] Se dice *procedural* porque se usan procedimientos o funciones en esos lenguajes.

representar de forma concisa una entidad que la distingue de otras; desde una perspectiva, en el análisis y diseño orientado a objetos, realizamos abstracciones a partir de un planteamiento de la vida real para crear los modelos que lo representan, que posteriormente se pasará en programa orientado a objetos para construir el sistema. Por lo tanto, desde el punto de vista de la orientación de objetos, la abstracción es la capacidad humana de identificar los objetos y clases de objetos que conforman el sistema y que permite, además, manejar la complejidad.

Figura del concepto de abstracción



Fuente: Análisis y diseño orientado a objetos con aplicaciones (Grady Booch, *et al*: 45).

Para construir sistemas complejos, el desarrollador debe abstraer distintas vistas del sistema, construir modelos utilizando notaciones precisas, verificar que los modelos plasmen los requisitos del sistema y añadir gradualmente los detalles para trasformar los modelos en una aplicación. La clasificación por categorización clásica (agrupar elementos con propiedades similares), agrupamiento conceptual (agrupar entidades que

compartan significado conceptual, es decir para qué sirven) y teoría de prototipos, pueden ayudar mucho.

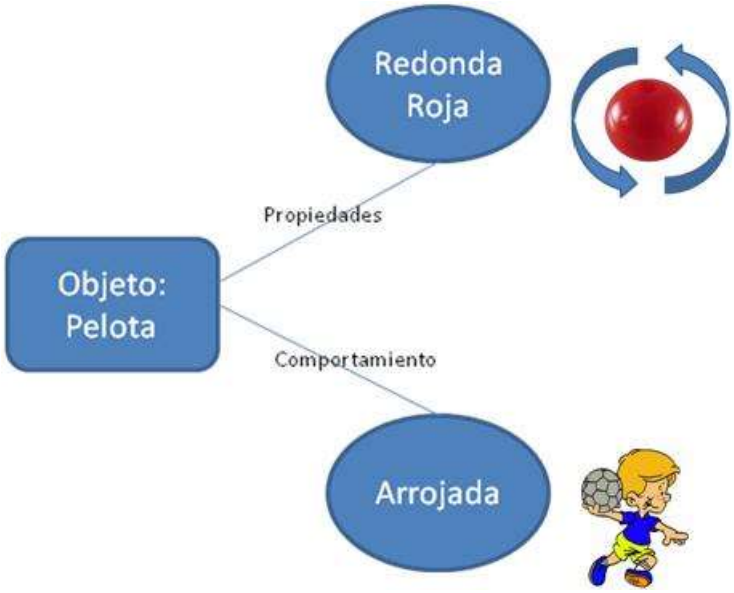
También es adecuado recordar que no debe pretenderse realizar una sola abstracción, lo mejor es realizar varias, y en cada una de ellas plasmar una parte del problema. Mediante la abstracción podemos identificar todos los elementos de un objeto, tales como su identidad (propiedades), sus estados (los valores de las propiedades), y comportamiento (los métodos que realiza).

Por ejemplo:

Una pelota redonda de color rojo es arrojada por los niños.

Realizando la abstracción, tenemos los objetos *pelota* y *niño*, de pelota tenemos como características *redonda y roja* y como comportamiento *es arrojada*.

Figura con la abstracción de niño que arroja una pelota.



Fuente: Elaboración propia

1.3.2 Modularidad

Un sistema complejo es más fácil de resolver si se divide en un conjunto de elementos (Pressman, 2010: 85). La modularidad nos ayuda a comprender mejor un negocio, ya que es mejor entender por partes como funciona y después ensamblar todo.

Cuando se realiza la modularidad, debe ser eficazmente, es decir, se debe ser altamente cohesivo en su función o restrictivo en su contenido y debe tener poco acoplamiento con respecto a fuentes de datos y otros elementos internos o externos.

Por ejemplo:
Una tienda de ropa crea diseños, compra insumos, vende su ropa, realiza composturas y devoluciones, etc. Todos estos procesos son módulos dentro de un sistema de una tienda de ropa.

Figura: Módulos de un sistema de una tienda de ropa.

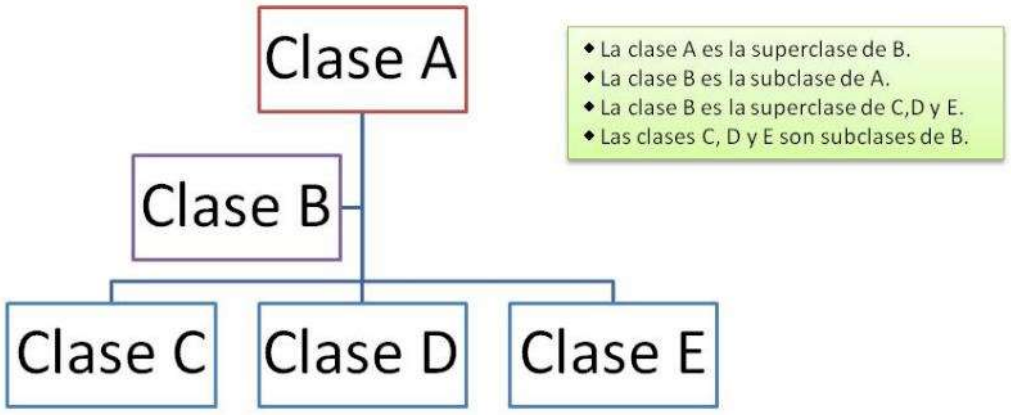
Fuente: Elaboración propia

1.3.3 Jerarquía

La jerarquía *es una clasificación u ordenación de abstracciones* (Booch, 1996: 66). La jerarquización consiste en agrupar clases que se obtuvieron de abstracciones realizadas y se agrupan de acuerdo a las funciones que realizan. La relación jerárquica se da cuando una clase hija (subclase) hereda de una clase padre (superclase).

Básicamente, la jerarquía define un tipo de relación entre las clases, que indica que puede tomar cierto comportamiento definido en una o más clases, la jerarquía entre las clases se puede diseñar como si fuera un organigrama.

Diagrama de jerarquía de clases



Fuente: http://www.ciberaula.com/articulo/tecnologia_orientada_objetos

Un conjunto de abstracciones a menudo forman una jerarquía, y mediante la identificación de esta jerarquía el diseño se simplifica ayudando a la comprensión del problema. A partir de la jerarquía de las clases se puede realizar la herencia. El concepto de herencia se explicará más abajo en conceptos.

1.3.4 Encapsulamiento

El encapsulamiento *consiste en separar los aspectos externos de un objeto, los cuales son accesibles por otros objetos, los detalles de implementación interno del objeto se*

ocultan de otros objetos (Rumbaugh, 1991: 7). Es importante ocultar los detalles de implementación de un objeto y que solo ciertas operaciones puedan ser visibles y utilizadas por el usuario. La encapsulación es como una caja negra que esconde los detalles para que no puedan ser vistos ni alterados y sólo permite acceder a ellos de forma controlada y a su nivel de acceso.

Por ejemplo:
La clase *persona* tiene como privado el atributo nombre, solo los métodos *set* y *get* de la propia clase puede acceder a su valor. Si otra clase quisiera acceder a él no podría por ser privado, de esta manera se protege y oculta la implementación, aplicando el encapsulamiento.

Figura que muestra el encapsulamiento

Fuente: Elaboración propia

En el siguiente código se muestra el encapsulamiento cuando son privados el atributo nombre y los métodos setNombre() y getNombre(), al ser privados están ocultando su uso a otras clases.

```
class Persona {
        private String nombre;
        private void setNombre(String nombre) {
                this.nombre = nombre;
        }
        private void getNombre() {
                return nombre;
        }
}
```

1.3.5 Concurrencia

Para cierto tipo de problemas, un sistema puede tener varios manejos de eventos simultáneamente. Esto puede involucrar mucha capacidad de cómputo de un solo procesador. En estos casos se debe considerar el uso distribuido de implementaciones que sean multitarea, por lo tanto la concurrencia es *la ocurrencia de dos o más lugares de ejecución durante el mismo intervalo de tiempo* (Booch, Jacobson y Rumbaugh, 2006: 500). Un sistema que tiene concurrencia debe poder manejar varios hilos en tiempo de ejecución como si fuera múltiple en un solo CPU.

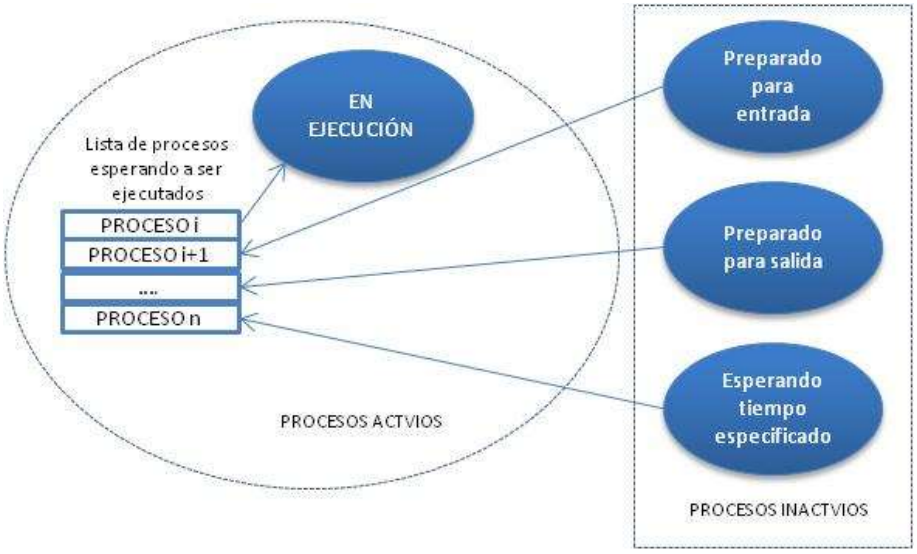


Figura con los procesos (tareas) concurrentes.

El objeto es un concepto que unifica dos puntos de vista diferentes: cada objeto (extraído de una abstracción del mundo real) puede representar un hilo de control separado (una abstracción del proceso), como se muestra en la figura de arriba, estos objetos se denominan activos.

En un sistema basado en un diseño orientado a objetos, se puede conceptualizar el mundo como consistente en un conjunto de objetos cooperativos, algunos de los cuales están activos y, por lo tanto, sirven como centros de actividad independiente. Dada esta concepción, la concurrencia es la propiedad que distingue un objeto activo de uno que no está activo.

Una de las realidades acerca de la concurrencia en un sistema, es considerar cómo los objetos activos pueden sincronizar sus actividades con los otros y con los objetos que son puramente secuenciales. Por ejemplo, si dos objetos activos intentan enviar mensajes a un tercer objeto, hay que estar seguro de utilizar algún medio de exclusión mutua, de manera que el estado del objeto sobre el que actúa no esté dañado cuando los objetos activos intenten actualizar su estado simultáneamente.

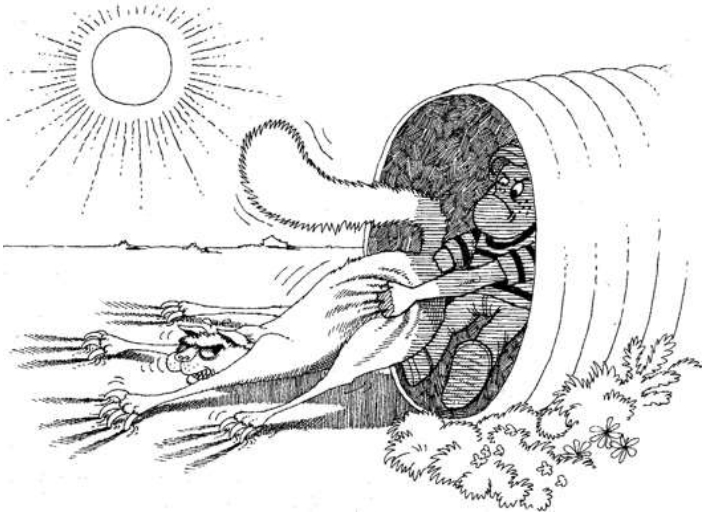
1.3.6 Persistencia

Un objeto, desde el punto de vista de software, toma cierta cantidad de espacio en memoria y existe una cantidad de tiempo. Por lo que un objeto en ocasiones debe tener la capacidad de existir sin algún cambio durante un tiempo. De aquí la importancia de que los datos se conserven sin alteración alguna durante las diferentes ejecuciones del programa.

La persistencia *es la propiedad de un objeto por lo que su existencia trasciende el tiempo (es decir, el objeto continua existiendo después de que su creador deja de existir) y/o el espacio (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado)*. (Booch *et al*., 2007: 71)

La persistencia ofrece algo más que el tiempo de vida de los datos. En bases de datos orientadas a objetos, no sólo el estado de un objeto debe persistir, sino también su clase debe superar cualquier cambio en el programa, de modo que cada programa interprete el estado guardado de la misma manera. Esto claramente hace que sea difícil mantener la integridad de una base de datos a medida que crece, especialmente si hay que cambiar la clase de un objeto después de que ya existe y se realizaron operaciones con éste.

Figura de persistencia que guarda el estado de un objeto.



Fuente: Booch, Jacobson y Rumbaugh, 2006: 70.

En la mayoría de los sistemas, un objeto, una vez creado, consume la misma memoria física hasta que deja de existir. Sin embargo, para los sistemas que se ejecutan en un sistema distribuido de procesadores, a veces tenemos que estar preocupados con la persistencia a través del espacio. En tales sistemas, es útil pensar en objetos que pueden moverse de una máquina a otra y que incluso pueden tener diferentes representaciones en diferentes máquinas, de allí la importancia de la persistencia.

En resumen la persistencia es la propiedad de un objeto a través del cual su existencia trasciende en el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, cuando se mueve el objeto de ubicación del espacio de direcciones en la que fue creado originalmente).

1.4 Conceptos del paradigma orientado a objetos

En el análisis y diseño orientado a objetos, existen algunos conceptos que primero se deben entender, para comprender la forma en que se maneja a partir de su asociación con el mundo real.

En el mundo, generalmente, encontramos personas, animales y cosas, todas ellas desde la prospectiva de la orientación a objetos se consideran clases de objetos y estos objetos tienen *estado* (definen cómo es, es decir, sus características) y *comportamiento* (las acciones que realiza), los cuales lo definen como un ente al que se conoce como *objeto*. Los objetos para poder comunicarse (interactuar) lo deben hacer por medio de mensajes, de esta manera saben que deben de responder con base en la petición recibida.

Una clase define cómo es el objeto en sí y existen 2 tipos de clases, las clases generales (superclases) y las subclases (especializadas), que van de lo general a lo particular, de aquí que exista una jerarquía entre ellas; a partir de ésta se puede saber de qué clase se puede realizar herencia. La herencia, como en la vida real, adquiere sus rasgos (atributos) físicos e incluso su forma de ser (comportamiento), esta cualidad también da origen al polimorfismo, que permite realizar operaciones de diferentes formas dependiendo del objeto que las realiza.

Las clases generalmente se agrupan de acuerdo al tipo de operación que realizan, a partir de ello se requiere una organización y ésta se realiza por medio de paquetes. También en la vida seguimos reglas de comportamiento (operación) y también de comunicación; para definir estas reglas necesarias en la operación se utilizan las interfaces.

1.4.1 Clase

Una clase *describe un grupo de objetos con propiedades similares (atributos), comportamiento común (operaciones), relaciones comunes con otros objetos y semántica común* (Rumbaugh *et. al*., 1991: 22). Desde la perspectiva del análisis y diseño orientación a objetos, en una clase se definen los ***atributos*** cómo las características que describen a la clase y las ***operaciones***, como las acciones que puede realizar. Desde la perspectiva de la programación orientada a objetos las características están representadas en ***variables*** y las operaciones en los ***métodos***.

Por ejemplo, piensa que te piden describir a un empleado. ¿Qué características puede tener?, Un empleado tiene un nombre, una edad, un tipo de sexo, tiene un número de seguro social y pertenece a un departamento, etc. Y ¿qué operaciones puede realizar o se le pueden solicitar?, se le puede pedir desde su nombre, apellidos, teléfono, su número de seguro social y también puede registrar su entrada o vender un producto, etc. En el siguiente diagrama se muestra algunos de ellos.

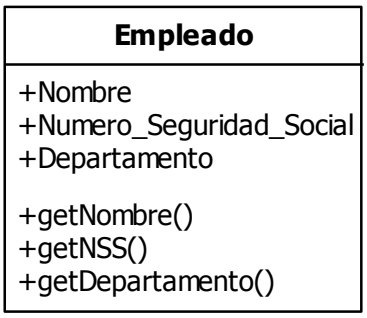


Diagrama de la clase empleado

Fuente: Booch, Jacobson y Rumbaugh, 1991: 21.

Atributos

Un atributo es un valor de un dato en poder de los objetos en una clase (Rumbaugh, 1991: 23). Los *atributos* definen las características que pueden tener un clase y éste es almacenado en variables.

Por ejemplo, del diagrama anterior, un empleado tiene como atributos:

- Nombre
- Número de seguro social
- Departamento (Al que pertenece)

Operaciones

Una operación *es una función o transformación que puede ser aplicada a o realizada por objetos en una clase* (Rumbaugh *et a*l., 1991: 25). Las *operaciones* representan el comportamiento que puede realizar. En la clase se encuentran definidos en métodos o funciones.

Usando el ejemplo anterior las operaciones (métodos) son:

- obtenerNSS()
- registrarEntrada()
- venderProducto()

En la siguiente tabla se muestran tres clases con algunos atributos y métodos que puede tener.

Clase	Atributos	Métodos
Persona	Color de piel Altura Género	Ver Escuchar Hablar
Televisión	Pulgadas Tipo Marca	Prender Apagar Emitir audio
Empleado	Nombre Seguro social Departamento	Obtener NSS Registrar Entrada Vender Producto

Fuente: Booch, Jacobson y Rumbaugh, 2006: 70.

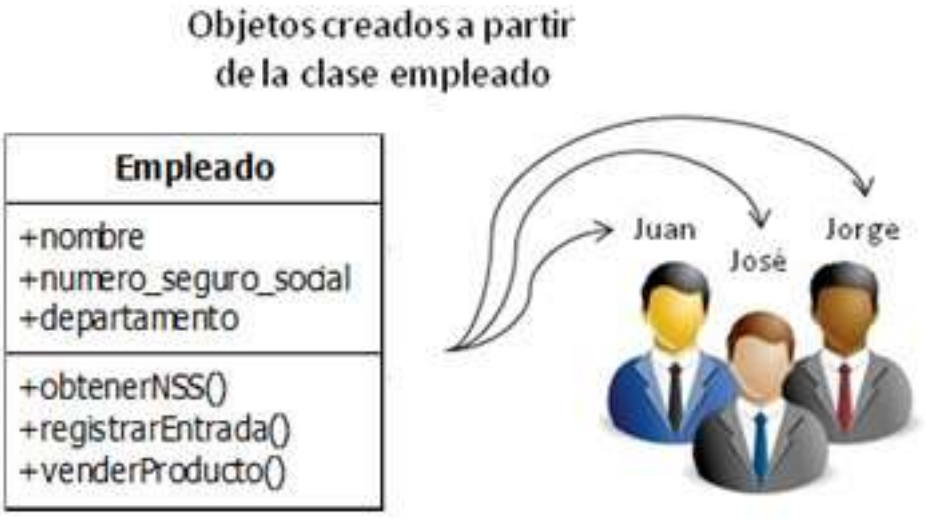
Se dice que una clase es un modelo base o plantilla por que a partir de ella se crean objetos.

1.4.2 Objeto

Un objeto *es una entidad tangible que exhibe algún comportamiento bien definido* (Booch *et al.*, 2007: 76), se le considera un objeto como la instancia de una clase debido a que es una copia que se hace a partir de una clase y para identificarlo se le da un nombre. Los objetos son abstracciones que forman parte del dominio del problema o del negocio, el cual representa un ente de la vida real. Los objetos por ser un ente de la vida real tienen *estado* y *comportamiento*, que se toman a partir de la definición de la clase.

Por ejemplo, a partir de la clase Empleado podemos crear los objetos Juan, José y Jorge, cada uno va a contener valores propios.

Figura de la clase Empleado base de los objetos: Juan, José y Jorge



Fuente: Booch, Jacobson y Rumbaugh, 2006: 76.

Cuando creamos objetos además de darles un nombre de objeto estos almacenan valores en las propiedades (generando un estado) y al utilizar las operaciones (se crea un comportamiento) del objeto. Por ejemplo, a partir de la clase empleado, se crean los siguientes objetos:

Objetos **(creados a partir de la *clase* Empleado)**		
Estado		
Juan (Nombre)	**José** (Nombre)	**Jorge** (Nombre)
1001 (Número de Seguro Social)	**1012** (Número de Seguro Social)	**1101** (Número de Seguro Social)
Finanzas (Departamento)	**Contabilidad** (Departamento)	**Recursos Humanos** (Departamento)

Comportamiento

1001 (obtenerNSS)	**1012** (obtenerNSS)	**1101** (obtenerNSS)
09:02:13 10/10/2010 (registrarEntrada)	**08:59:35 10/10/2010** (registrarEntrada)	**09:08:01 10/10/2010** (registrarEntrada)
Camisa (venderProducto)	**Chamarra** (venderProducto)	**Pantalón** (venderProducto)

Fuente: Booch, Jacobson y Rumbaugh, 2006: 77.

Un objeto almacena toda en sus atributos y establece u obtiene información a través de sus métodos, cada objeto es una instancia distinta a otras y en cuanto a su estructura, es decir la clase, es la misma. Del ejemplo anterior, los tres “Juan, José y Jorge” objetos se crean a partir de la clase Empleado. En conclusión, un objeto es la instancia de una clase que puede almacenar valores y una clase se conforma de atributos y métodos que la definen.

1.4.3 Herencia

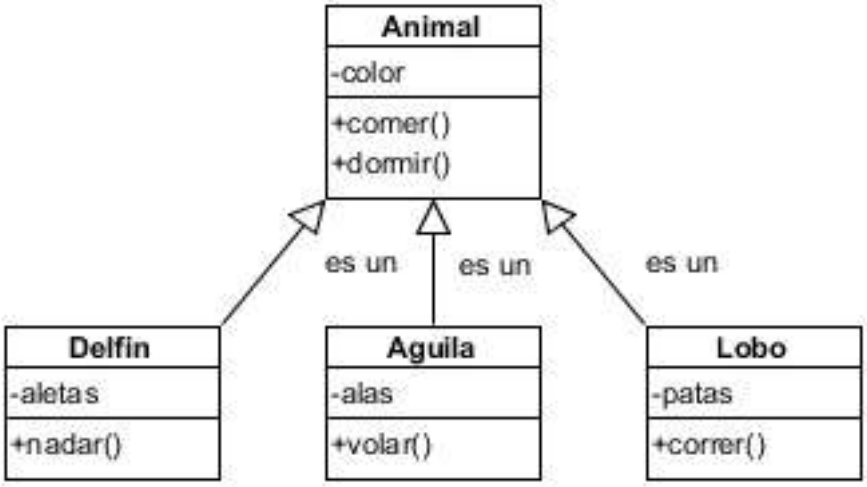
La herencia *es una técnica de reutilización en el cual una clase hija hereda todos los atributos y operaciones de una clase padre* (Bruegge *et al.*, 2002: 519). La herencia se realiza a partir de una subclase o clase hija que retoma la lo que tiene definido la superclase o clase padre, además puede agregar nuevos atributos y operaciones para hacerla una clase más especializada.

Cuando se aplica el concepto de jerarquía en las clases, aparecen otros, como la herencia simple (cuando una subclase hereda los métodos, estructura y comportamiento de una superclase), herencia múltiple (cuando una sub-clase hereda los métodos, estructura y comportamiento de varias superclases) y agregación (el concepto de herencia pero visto en sentido inverso; lo que significa que una subclase es parte de ésta,

es decir agregada a una súperclase), conceptos que tienen mucho que ver con los tipos de relaciones que hay entre las clases.

Por ejemplo, una clase superclase Animal tiene tres subclases: Delfín, Águila y Lobo, éstas heredan de animal el atributo de color y las operaciones de comer y dormir, las subclases incluyen sus características, propios atributos y operaciones como nadar para delfín, volar para águila y correr para lobo. Utilizando la herencia, el delfín, el águila y el lobo pueden comer y dormir debido a la herencia.

Figura con la representación de la herencia entre clases.



Fuente: Bruegge *et al.*, 2002: 519.

Así se simplifican los diseños y se evita la duplicación de código al no tener que volver a codificar métodos ya implementados en otra clase.

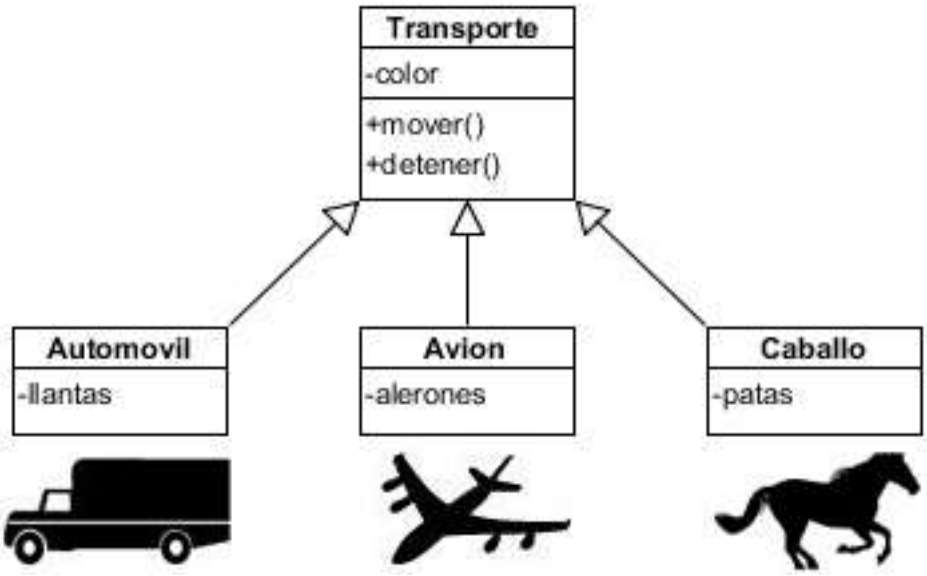
1.4.4 Polimorfismo

La Real Academia española define polimorfismo como “la cualidad de lo que tiene o puede tener distintas formas” (www.rea.es, 2013). Desde la perspectiva de la orientación de objetos, el polimorfismo se puede aplicar en dos sentidos: uno es cuando a partir de

una clase se crean diferentes objetos y, el segundo, es cuando una operación puede tener diferente comportamiento (implementación) en cada una de las instancias.

Por ejemplo, de la clase transporte se pueden crear diferentes objetos Coche, Cohete y Caballo; los tres objetos son un medio de transporte y además estos avanzan y frenan de diferente manera.

Diagrama ejemplificando el polimorfismo a partir de un objeto



Fuente: Bruegge *et al*., 2002: 519

El polimorfismo es útil para extender la funcionalidad del sistema al definir nuevas clases especializadas y, por otra parte, sirve para definir un estándar en el nombre de los métodos o funciones que todos pueden recordar.

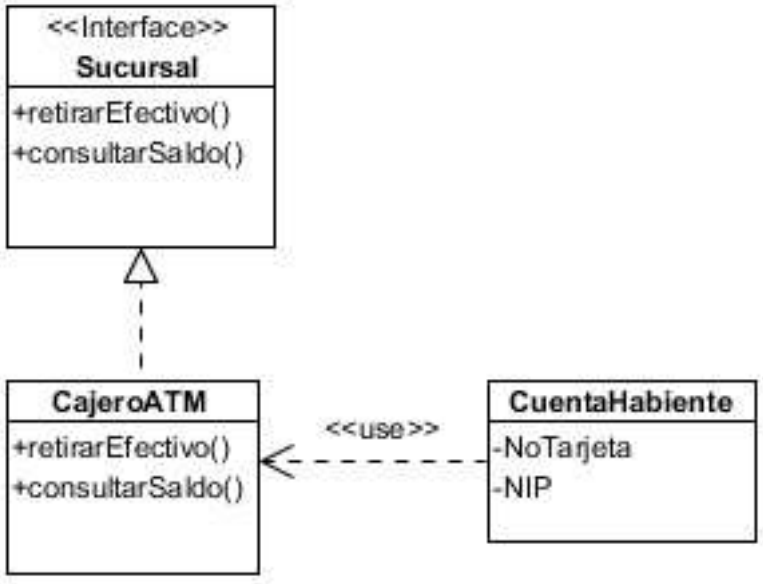
1.4.5 Interfaz

Una interfaz es *una colección de operaciones que se usan para especificar un servicio de una clase o de un componente* (Booch, Jacobson y Rumbaugh, 2006: 163). La interfaz de una clase captura sólo su vista exterior, que abarca nuestra abstracción del comportamiento común a todas las instancias de la clase. En el lenguaje de programación

Java, una interface se considera un contrato en donde la clase que la implementa está obligada a contener todos los métodos.

También al definir en una arquitectura se le llaman interfaces a los puntos o nodos que fungen como conexión entre dos componentes para entrar en comunicación. Y de forma más general también una interface es aquel medio con el que el usuario puede interactuar con el sistema o aplicación.

Figura de la interface sucursal que implementa el cajero ATM para el cuentahabiente.



Fuente: Bruegge *et al*., 2002.

En la práctica, esto significa que cada clase debe tener dos partes: una interfaz y una implementación. La interfaz de una clase captura sólo su vista exterior, que englobe nuestra abstracción del comportamiento común a todas las instancias de la clase. La implementación de una clase comprende la representación de la abstracción, así como los mecanismos necesarios para conseguir el comportamiento deseado. La interfaz de una clase es el único lugar en el que afirman todos las acciones que un cliente puede hacer sobre las instancias de la clase definidas, de esta forma la aplicación encapsula detalles que ningún cliente debe conocer.

1.4.6 Paquete

Los paquetes de servicios *constituyen un elemento esencial para las actividades de diseño e implementación, las cuales ayudaran en la estructura de los modelos de diseño e implementación en términos de subsistemas de servicios* (Ivar Jacobson, http: //blog.ivarjacobson-com/soa, 2004). Los paquetes son una forma organizacional ya que generalmente son directorios que contienen un conjunto de clases y/o subpaquetes, agrupados por su funcionalidad o por el tipo de servicio que prestan. Nosotros podemos tener varios paquetes en una aplicación.

Para todos los problemas, el desarrollador debe decidir en donde declarar cada clase y objeto, es decir, en que paquete debe estar. Para cualquier tipo de negocio, por muy grande o pequeño que sea el software a construir, la mejor solución es agrupar clases relacionadas lógicamente en el mismo módulo.

Por ejemplo, en una tienda que vende productos o servicios, se tiene contemplado un paquete con las clases que se encargan de los clientes (para da de alta clientes y registro de su domicilio etc.), en otro paquete las clases que se encargan de las ventas, otro paquete para manejar los pagos de las ventas realizadas, otro paquete que lleve la facturación electrónica al cliente y, además, otro paquete que se encargue de la seguridad tanto de las operaciones como la autentificación de los clientes, emisión cifrada de los pagos, etc., este acopio de paquetes tienen una relación que trabaja en conjunto para que funcione el sistema. En resumen, cada paquete debe cumplir con un servicio o función en particular, para que todo quede de forma especializada y se pueda reutilizar.

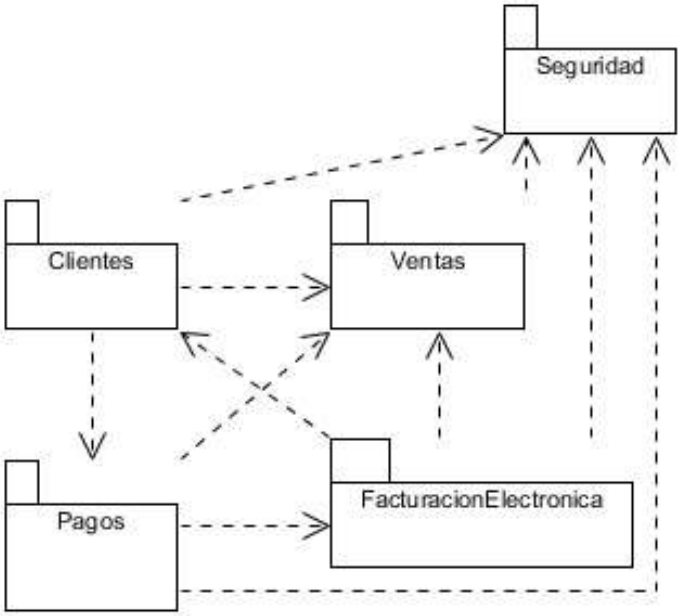


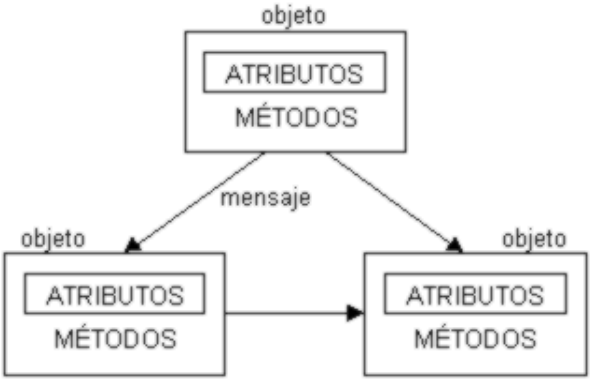
Figura de un diagrama de paquetes.

1.4.7 Mensaje

Los objetos en un sistema se comunican con otros objetos y para ello utilizan los mensajes. Un mensaje típicamente es una llamada hacia una operación que un objeto invoca en otro objeto (Erikson Hans-Erick, *et al*., 2004: 145p). Debido a esto los objetos pueden ser activados mediante la recepción de mensajes. Un mensaje realiza una petición para que un objeto sea ejecutando con una función en particular. La técnica de enviar mensajes se conoce como *paso de mensajes*. En casi todos los casos, el programador sabe qué tipo de objetos y de dato se espera que reciba como argumento(s) de un mensaje y también conoce el tipo de objeto de retorno.

Por ejemplo, consideremos que una aplicación se ejecuta en un conjunto distribuido de procesadores y utiliza un mecanismo de pase de mensajes para coordinar las actividades de los diferentes programas. El servidor de aplicaciones debe conocer qué objetos se crean y en que ámbito existen para que pueda realizar el envío de mensajes entre los objetos involucrados.

Figura del envío de mensajes entre objetos



Fuente: http://compuupc2009.blogspot.mx

La comunicación entre los objetos y los efectos de tal comunicación muestran la dinámica del sistema, esto es, cómo los objetos a través de la comunicación trabajan colaborativamente y como los objetos en el sistema cambian de estado durante el ciclo de vida del sistema.

RESUMEN

En esta unidad identificamos en qué consiste la administración de requerimientos, su importancia (comprenderlos, obtener el compromiso sobre éstos, gestionar los cambios, mantener la trazabilidad bidireccional, asegurar el alineamiento entre el trabajo del proyecto y los requisitos), cómo se lleva el seguimiento de los requisitos y cómo manejar los cambios en ellos a fin de aclarar con los clientes y/o usuarios las modificaciones en la construcción del sistema.

Los modelos y las metodologías nos ayudan a comprender, mediante la realización de diagramas (empleando la semántica y la notación correspondiente), todos los detalles del sistema, de esta manera la abstracción de la realidad se plasma en modelos que todos los involucrados puedan entender y conocer todos los escenarios.

Asimismo, abordamos los principios y conceptos del paradigma orientado a objetos. Una clase es una plantilla o molde que a partir de ella se puede crear diferentes objetos, cuando de una clase como figura creamos diferentes objetos como cuadrado, circulo, rectángulo esto es polimorfismo. Un objeto posee un estado (almacenado en atributos o variables) y un comportamiento (representado en métodos u operaciones).

La forma en que se comunican los objetos es por medio de mensajes lo que permite enviar información por medio de parámetros para obtener un resultado. Las clases pueden estar agrupadas por su funcionalidad en paquetes, al estar organizadas tienen un jerarquía entre la clase padre y las clases hijas; asimismo, también pueden adquirir estado y comportamiento de otra clase por medio de la herencia.

El encapsulamiento nos ayuda a ocultar el acceso a la implementación de otros objetos que no necesitan conocer. El manejo de la concurrencia permite atender varias tareas al

mismo tiempo y la persistencia en los objetos permite conservar su estado y/o comportamiento durante el tiempo sin afectaciones.

Es importante resaltar los beneficios que se presentan con este paradigma:

- ✓ Cambia la organización del programa, ya que se presenta en clases (datos + operaciones sobre datos).
- ✓ Cambia el concepto de ejecución de programa (a través del paso de mensajes).
- ✓ Cambia el concepto de dato (pasivo) por el de objeto (activo), ya que cada objeto es una especie de máquina funcional.
- ✓ Los mecanismos de encapsulación facilitan la comprensión del programa y permite, por ejemplo, la generación automática de documentación.
- ✓ Permite reutilizar clases por medio de la herencia y darle una clasificación de acuerdo a su propósito.

El comprender los principios y conceptos ayudan a realizar un análisis y diseño orientado a objetos correctamente y con facilidad.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Sommerville (2011)	4	112-114
Weitzenfeld (2008)	4	82
Sánchez (2012)	4	139 - 141
Software Engineering Institute	2a Parte	429-436
Rumbaugh, Jacobson y Booch (2000)	2	11-16
Fowler (1999)	1	5-7
Hans-Erik (2004)	1	11
Rumbaugh, *et al*. (1991)	1	7
Rumbaugh, Jacobson y Booch	23	358
Bruegge Bernd y Dutoit Allen H. (2002)	Apéndice B	519

Sommerville, I. (2011). *Ingeniería de software* (9ª ed.). México: Pearson Addison-Wesley.

Weitzenfeld, A. (2008). *Ingeniería de software orientada a objetos con UML, Java e Internet.* México: Cengage Learning.

Sánchez, S. *et al*. (2012). *Ingeniería del software: Un enfoque desde la guía SWEBOK.* Barcelona: Alfaomega.

Rumbaugh J., Jaconson, I. y Booch G. (2000*). El lenguaje unificado de modelado. Manual de referencia.* España: Addison-Wesley.

Fowler Martin con Kendall Scott (1999). *UML gota a gota*. México: Pearson Addison-Wesley.

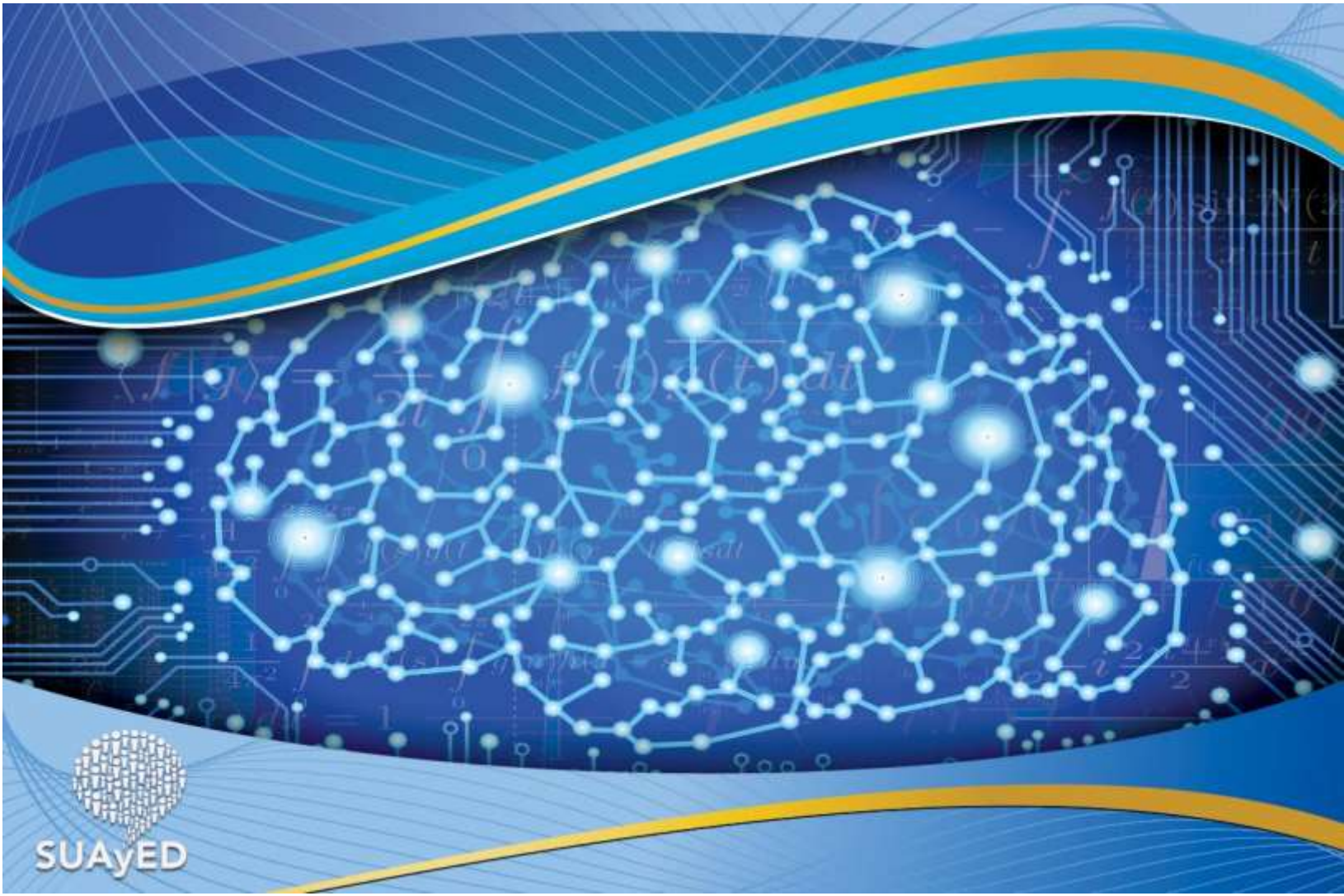
Hans-Erik Erikson *et al*. (2004). *UML 2 Toolkit*. E.U.A: Wiley Publishing Inc.

Rumbaugh Et Al. (1991). *Diseño y modelado orientado a objetos*. E.U.A: Prentice-Hall.

Bruegge Bernd y Dutoit Allen H. (2002). *Ingeniería de software orientada a objetos*. México: Prentice Hall.

Unidad 2.

Metodologías orientadas a objetos



OBJETIVO PARTICULAR

Identificar las actividades de los modelos representativos del análisis y diseño orientado a objetos.

TEMARIO DETALLADO
(14 horas)

2. Metodologías orientadas a objetos

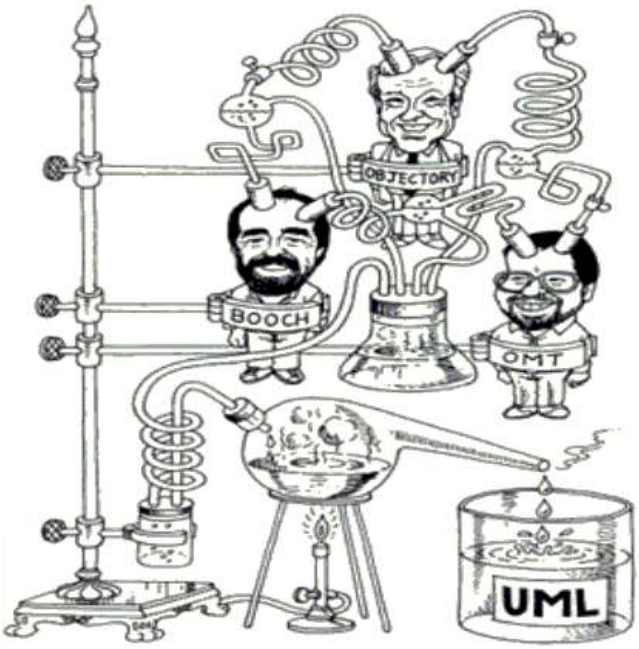
2.1. Booch
2.2. Jacobson (Objectory)
2.3. Rumbaugh (OMT-Técnica de Modelado de Objetos)

INTRODUCCIÓN

Los lenguajes de modelado orientados a objetos aparecieron a mediados de 1970 y continuaron durante 1980, los programadores y administradores de proyectos intentaron con diferentes técnicas de análisis y diseño, pero lo único que se tenía eran metodologías que conceptualizaban todo desde un punto de vista estructurado y nada visto como objeto.

Así es como la *orientación a objetos* emergió inicialmente a través de un lenguaje de programación conocido como *Simula*, el cual no fue popular sino hasta después de 1980, junto con el surgimiento de lenguajes de programación Como C++ y Smalltalk. Cuando la programación orientada a objetos llegó a ser un éxito, hecho que cambio la forma de ver y plantear los problemas, hubo la necesidad de alguna metodología que siguiera el paradigma orientado a objetos. Más de 50 métodos y lenguajes de modelado aparecieron en 1994 y siguieron creciendo.

A mediados de 1990, los lenguajes de modelado empezaron a incorporar las mejores prácticas de lenguajes anteriores, pero solo pocos fueron bien conceptualizados y estructurados. Entre estos lenguajes de modelado orientado a objetos se incluyen los de Grady Booch enfocándose sobre el diseño y construcción de fases y proyectos, la aportación de Ivar Jacobson con Objectory empleando los casos de uso y James Rumbaugh con la técnica de modelado de objetos (OMT y OMT2), el cual está orientado al análisis y el modelado de datos para los sistemas de información.



Caricatura de los tres Amigos pioneros del lenguaje del lenguaje del modelado unificado
Fuente: Quatrani, 2010.

El uso de una metodología para el análisis y diseño es muy útil; en esta unidad se verán los principales exponentes y pioneros con sus respectivas aportaciones, quienes posteriormente dieron origen al Lenguaje de Modelado Unificado (UML), el cual se empleará en las siguientes unidades.

2. Metodologías Orientadas a Objetos

A continuación abordaremos las metodologías también conocidas como lenguajes de modelado orientado a objetos de sus principales autores.

Booch (1996) define la noción a partir de que un sistema se analiza desde diferentes puntos de vista, donde se describe cada vista por un número de diagramas de modelo. El método también contiene un proceso por el cual se analiza el sistema, tanto desde el aspecto macro y micro de desarrollo, y se basa en un proceso basado en modelos altamente incremental e iterativo, para describir el sistema.

Por otro lado, están los métodos OOSE/Objectory ambos creados por Ivar Jacobson. El método OOSE era la propia visión de Jacobson de un método orientado a objetos. El método Objectory fue adaptado para la ingeniería de negocios donde las ideas se plasman para modelar y mejorar los procesos de negocio. El método Objectory se utilizó para la construcción de una serie de sistemas de telecomunicaciones y sistemas financieros para empresas de Wall Street. Ambos métodos se basan en casos de uso, que luego son aplicadas en todas las fases del desarrollo.

La técnica de modelado de objetos (OMT) se desarrolló en General Electric por James Rumbaugh y es un proceso sencillo para la realización de las pruebas basadas en una especificación de requerimientos. El sistema se describe mediante un número de modelos, incluyendo: el modelo de objetos, el modelo dinámico y el modelo funcional, que se complementan entre sí para dar la descripción completa del sistema. El método OMT también contiene instrucciones prácticas para el diseño del sistema, teniendo en cuenta la concurrencia y la asignación a las bases de datos relacionales.

2.1 Metodología Booch

El diseño orientado a objetos *es un método que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estáticos y dinámicos del sistema que se diseña* (Booch, 1996: 43). Durante la realización del diseño se busca definir la estructura de las clases, que Booch denomina diseño lógico, y también construir la arquitectura que denomina diseño físico.

El análisis orientado a objetos *es un método de análisis que examina los requerimientos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema* (Booch, 1996: 44). A partir de la perspectiva que se tiene de la vida real y con base en los requerimientos se identifican las clases y los objetos para construir el sistema.

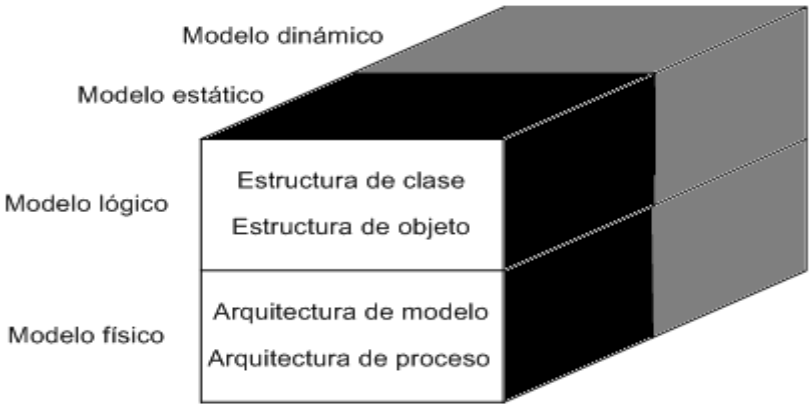


Figura con los modelos de desarrollo orientado a objetos
Fuente: Booch (1996: 200)

Es necesario el uso de una notación estándar que haga posible que un analista o desarrollador pueda describir un sistema y la definición de la arquitectura. En general una notación es muy útil ya que sigue de forma consistente para plasmar los conceptos e

ideas de esta manera todo el equipo conoce todos los aspectos y detalles desde diferentes vistas. En la metodología de Booch, *en cada dimensión se define una serie de diagramas que denotan una vista de los modelos de un sistema* (Booch, 1996: 202).

Modelo lógico versus modelo físico

En la vista lógica se busca definir todas las abstracciones y entidades clave que son parte de la composición del sistema y/o son parte del dominio del problema, para esto se pueden utilizar los diagramas de clases. Dentro del modelo físico se debe definir tanto el software y hardware del sistema.

Durante el análisis (Booch, 1996: 203), señala que debemos tener en mente las siguientes preguntas:

- ¿Cuál es el comportamiento deseado del sistema?
- ¿Cuáles son las funciones y responsabilidades de los objetos que llevan a cabo este comportamiento?

Para el modelo lógico, se busca definir el comportamiento del sistema a través de diagramas de objetos mostrando así sus roles y responsabilidades. Así se puede conocer el comportamiento del sistema para cada escenario.

Durante el diseño, (Booch, 1996: 203-204), se deben hacer las siguientes preguntas y responder haciendo los diagramas correspondientes para definir la arquitectura del sistema:

Pregunta	**Diagramas estáticos**
¿Qué clases existen y cómo se relacionan esas clases?	*Diagrama de clases*
¿Qué mecanismos se utilizan para regular cómo los objetos colaboran?	*Diagrama de objetos*
¿Dónde debe cada clase y objeto ser declarado?	*Diagrama de módulos*
¿En qué procesador debe asignarse un proceso y por qué procesador dado, cómo sus múltiples procesos se programaran?	*Diagrama de procesos*

Modelo estático versus el modelo dinámico

Los diagramas estáticos representan solo una parte del sistema pero debido a la dinámica de los sistemas los objetos realizan operaciones que a su vez disparan otros eventos sobre otros objetos, *para representar la dinámica se utiliza los siguientes diagramas* (Booch, 1996: 204):

- Diagramas de transición de estados.
- Diagramas de interacción.

Los diagramas de transición de estados nos muestran el comportamiento de las instancias de clase, es decir, de los objetos y los eventos que ocurren, como se van presentando y resultado que tiene al pasar a otro estado.

Los diagramas de interacción nos ayudan mostrando en una línea de tiempo los eventos con los objetos como evalúan para enviar un mensaje a otro objeto con las respuestas que se envía a su vez a otro, hasta terminar de representar todo un escenario.

Diagrama de clases

Un diagrama de clases se utiliza para mostrar la existencia de entidades y sus relaciones en la vista lógica de un sistema. *Un diagrama de clases únicamente representa una vista de la estructura de un sistema* (Booch, 1996: 206).

Figura que representa una clase

Fuente: Booch, 1996: 206.

En la metodología de Booch una clase se representa en un diagrama de clases con el símbolo de la forma de una nube. A una clase debe dársele un nombre, en caso de que el nombre sea muy largo puede ser abreviado. Cada nombre de clase debe ser único en su categoría de clase e irrepetible en el contexto del dominio del problema.

En una clase solo se muestran los atributos y operaciones necesarios ya que a veces no se representa de forma específica. Se puede mostrar solo el nombre de la clase o también con los atributos y operaciones.

Atributos

Usando la siguiente sintaxis independiente del lenguaje, *un atributo puede tener un nombre, una clase, o ambos, y opcionalmente un valor por defecto* (Booch, 1996: 207).

- ✓ A Solo el nombre del atributo
- ✓ :C Solo el nombre de la clase
- ✓ A:C Nombre del atributo y de la clase
- ✓ A:C = E Nombre del atributo, clase y expresión por defecto.

Operaciones

Una operación denota algún servicio proporcionado por la clase. *Las operaciones se distinguen de los atributos añadiendo paréntesis* (Booch, 1996: 207):

- ✓ N( ) Solo el nombre de la operación.
- ✓ R N(A) Valor de retorno R, nombre de la operación N y argumentos (A) "si hubieran".

Clase abstracta

Una clase abstracta es una clase que solo muestra la definición de los métodos. Booch define un adorno para representar una clase con una letra A dentro de un rectángulo, para indicar que es una clase abstracta, como a continuación se presenta.

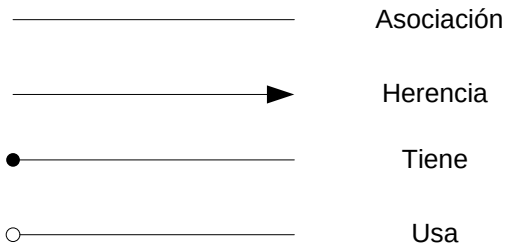
Figura que representa una clase abstracta.

Fuente: Booch, 1996: 207.

Relaciones entre clases

Las clases por el hecho de colaborar con otras clases es necesario indicar el tipo de relación que tienen. *Las relaciones esenciales entre las clases son de asociación, herencia, posesión "tiene" y utilización "usa"* (Booch, 1996: 208). A las relaciones se puede incluir un texto que describe el nombre de la relación. Una clase también puede tener una relación a sí misma, que se conoce como reflexiva.

Figura con los iconos para representar las relaciones entre las clases



Fuente: Booch, 1996: 207.

La cardinalidad entre clases

La cardinalidad se indica en el extremo de una asociación e *indica el número de enlaces de cada instancia de la clase de origen y las instancias de la clase destino* (Booch, 1996: 208).

Sintaxis para representar la cardinalidad de las asociaciones

- 1 En específico uno
- N Muchos
- 0..N Cero a muchos
- 1..N Uno a muchos
- 0..1 Cero o Uno
- 3..7 Un rango en especifico
- 1..3,7 Un rango en específico o un número exacto

Fuente: Booch, 1996: 208.

Después de definir la relación general entre dos clases, el siguiente paso es definir el tipo de relación, que puede ser de herencia, posesión o utilización.

Tipos de relaciones

La herencia define *una relación de generalización y especialización, esta expresa una relación "es un"* (Booch, 1996: 208), es decir, una subclase es una superclase, por

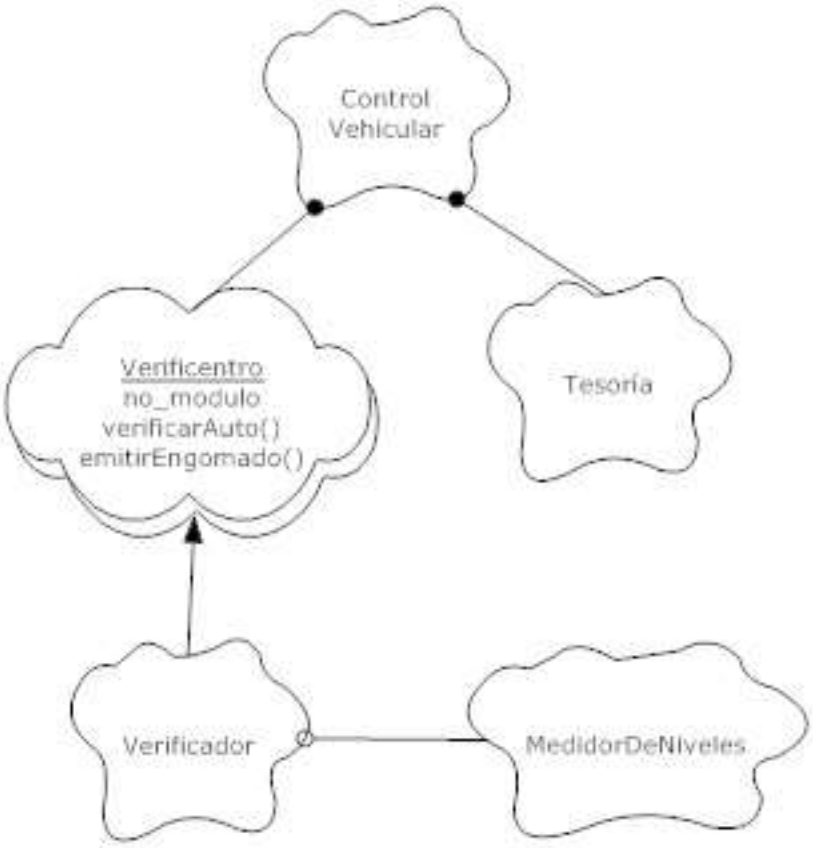
ejemplo: Un tigre es un animal. En la metodología de Booch la herencia se presenta con una asociación de una flecha con punta, en donde la punta de flecha va a la superclase y el otro extremo a la subclase. Algo importante es que las relaciones de herencia no tienen cardinalidad.

La relación *de agregación indica un todo o parte y expresa "tiene un"* (Booch, 1996: 208), en la metodología de Booch se denota como una asociación con un círculo relleno. La clase en el otro extremo es la parte cuyo contenido es agregado a la otra clase, por ejemplo: Un automóvil tiene un motor. La agregación puede ser reflexiva, es decir, puede asociarse a sí misma.

La relación *de utilización indica una relación de "usa"* (Booch, 1996: 208) y se representa en la metodología de Booch la asociación con un círculo sin relleno. Esta relación expresa que una clase depende de otra para brindar ciertos servicios o que tienen firmas de operaciones cuyo valor de retorno o argumentos son instancias de la clase que las provee, por ejemplo: Un cliente usa una tarjeta bancaria.

Para ejemplificar el uso de todos los tipos de relaciones entre clases, pensemos en un sistema de control vehicular, del cual se describe el manejo para la verificación de los automóviles. En este modelo de clases, la clase ***Verificentro***, la cual tiene en su definición un atributo no_modulo junto con las operaciones de ***verificarAuto*** y ***emitirEngomado***. Existe una asociación de todo/parte del ControlVehicular con Vereficentro y Tesorería. Además, el equipo ***Verificador*** es una clase de Verificentro, cada equipo verificador tiene un ***MedidorDeNiveles***.

Diagrama de clases del control vehicular



Fuente: Booch, 1996.

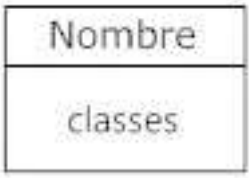
Este diagrama también muestra que la clase ***ControlVehicular*** es una agregación, cuyas clases contenidas son ***Verificentro*** y ***Tesorería***. El ***Verificador*** es una subclase de ***Verificentro***. La clase Verificador usa la clase ***MedidorDeNiveles***.

Categoría de clases

Después de realizar las abstracciones aparecen una serie de clases que se deben de categorizar de acuerdo a su función en grupos de clases que en sí mismas se cohesionan, pero que están débilmente acopladas con respecto a otros grupos de clases. Por lo tanto, para representar estos grupos se definen las categorías de clase.

Las clases y categorías de clase pueden aparecer en el mismo diagrama. Lo más común para representar la arquitectura lógica de alto nivel de nuestro sistema, por medio de diagramas de clases que sólo contienen las categorías de clase. En pocas palabras, una categoría de clase es una agregación que contiene otras clases. Cada clase en el sistema tiene que estar en una categoría. Cada categoría de clase representa una encapsulación.

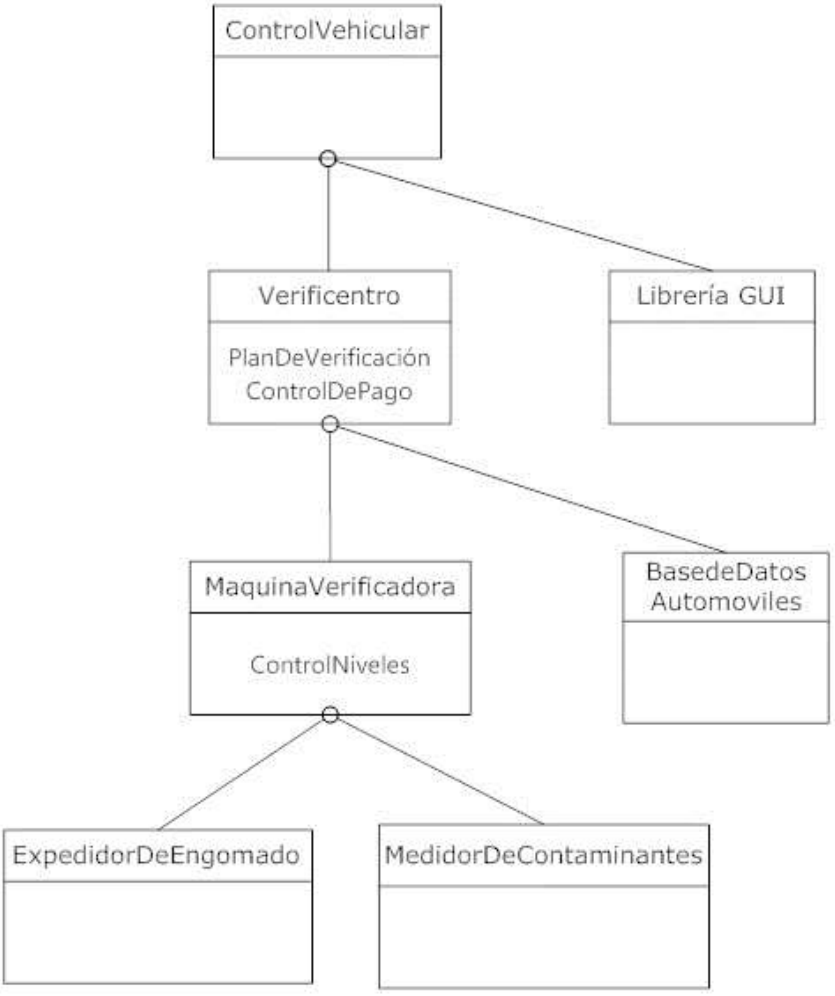
Figura que representa una categoría de clases.



Fuente: Booch, 1996: 210.

Durante el análisis y diseño arquitectónico, esta distinción es muy importante, porque nos permite especificar una clara separación de las funcionalidades de las clases exportadas que proporcionan ciertos servicios desde la categoría de clase y las clases que implementan estos servicios (Booch, 1996: 211).

Diagrama de clases por categorías de alto nivel del sistema de verificentro.



Fuente: Booch, 1996.

Los diagramas de clases pueden contener categorías de clases que representan de forma agrupada la clases y muestra la arquitectura superior de nuestro sistema. A estos diagramas se les considera de alto nivel porque muestran las capas en que se divide le sistema.

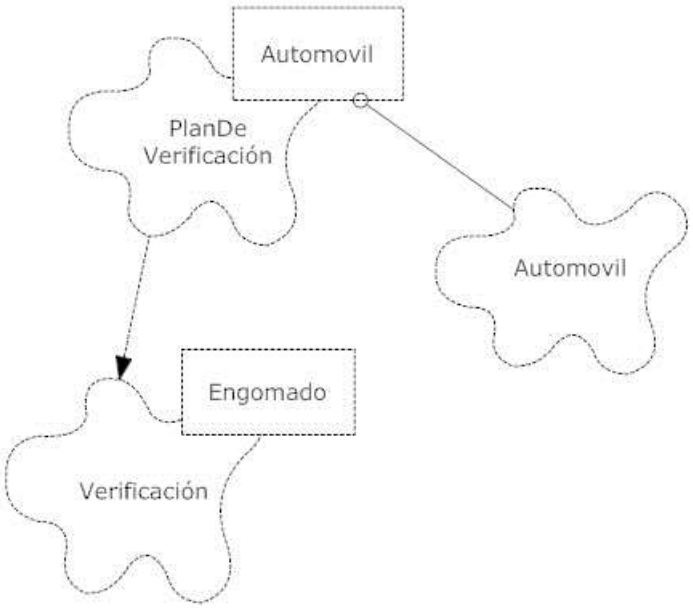
Las capas representan agrupaciones de categorías de clase, del mismo modo que las categorías de clases representan agrupaciones de clases. Un uso común de capas, es aislar las capas superiores de los detalles de la capa inferior (Booch, 1996: 212).

La categoría de clase llamada **ControlVehicular** es general, lo que indica que sus servicios están disponibles para todas las categorías de la clase. La categoría de clase **Verificentro** expone dos de sus clases **PlanDeVerificacion** y **ControlDePago**.

Clases parametrizadas

También existe una forma de representar los parámetros que reciben las clases, a éstas se les conocen como clases parametrizadas. La forma de representar la relación entre una clase parametrizada de su clase de instancia es por medio de una línea discontinua que va hacia la clase parametrizada, de acuerdo a la notación de Booch.

Diagrama con clases parametrizadas



Fuente: Booch, 1996.

A partir de las clases concretas se crean diferentes instancias de las clases y a partir de los parámetros que reciben se crean diferentes tipos de objetos, actuando de forma polimórfica y, finalmente, los diferentes argumentos son los que permiten tener un comportamiento diferente.

Diagrama de transición de estados

Un diagrama de transición de estado *se utiliza para mostrar los estados de una clase dada, los eventos que causan una transición de un estado a otro y las acciones que resultan de un cambio de estado* (Booch, 1996: 229).

Los diagramas de transición de estados no ayudan a ver el cambio de estado y éste depende los eventos que disparan el cambio, no todas las clases tienen un cambio de estado, por lo tanto, solo es útil para aquellas que sufren un cambio, dado un evento. Otra característica es que este diagrama sirve para ver el modelo dinámico de una clase o también para ver el estado de todo un sistema.

Durante el análisis, se utiliza diagramas de transición de estado para indicar el comportamiento dinámico del sistema. Durante el diseño, se utiliza diagramas de transición de estado para capturar el comportamiento dinámico de las clases individuales o de colaboración de clases (Booch, 1996: 230).

Existen dos elementos importantes en este diagrama uno son los estados y otro las transiciones de estado.

Estados

El estado de un objeto muestra el comportamiento de un objeto, dado un evento, y este provoca un cambio durante un tiempo hasta que sucede otro. Por ejemplo, una computadora al encenderla se inicia, se encuentra en el estado de inicio, después la computadora hace una revisión del hardware y software, si hubiera un fallo de hardware pasa a dar de baja y si tiene problemas de algún tipo iniciará con fallos. Una vez que el sistema operativo levanta, se dice que está en estado de *operando*, por falta de inactividad o baja de energía puede entrar al estado de *suspendido* y, finalmente, cuando el usuario decide apagar la computadora entra al estado *dar de baja*.

El estado de un objeto abarca todas sus propiedades (usualmente estáticas), junto con los valores actuales (usualmente dinámico) de cada una de estas propiedades (Booch, 1996: 230). Recordando que las propiedades son todos los atributos de un objeto así como también las relaciones que tiene con otros objetos. La figura de abajo muestra el icono que se usa en la metodología de Booch para representar un estado.

Figura para definir el estado de un objeto

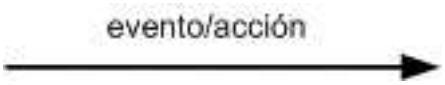
Fuente: Booch, 1996: 231.

El nombre del estado debe ser único dentro del o dominio de negocio. Los estados a su vez pueden tener estados otros estados de forma anidada.

Transiciones de estado

Un evento es algún suceso que pueda causar un cambio de estado en un sistema. Este cambio de estado se denomina transición de estado (Booch, 1996: 231). La transición va de un estado A hacia un estado B y un estado puede tener una transición a sí mismo (lo que no es común), lo más utilizado es tener muchas transiciones a diferentes estados y otra regla es que la transición también se considera única desde el mismo estado.

Icono de transición de estado



Fuente: Booch, 1996: 231.

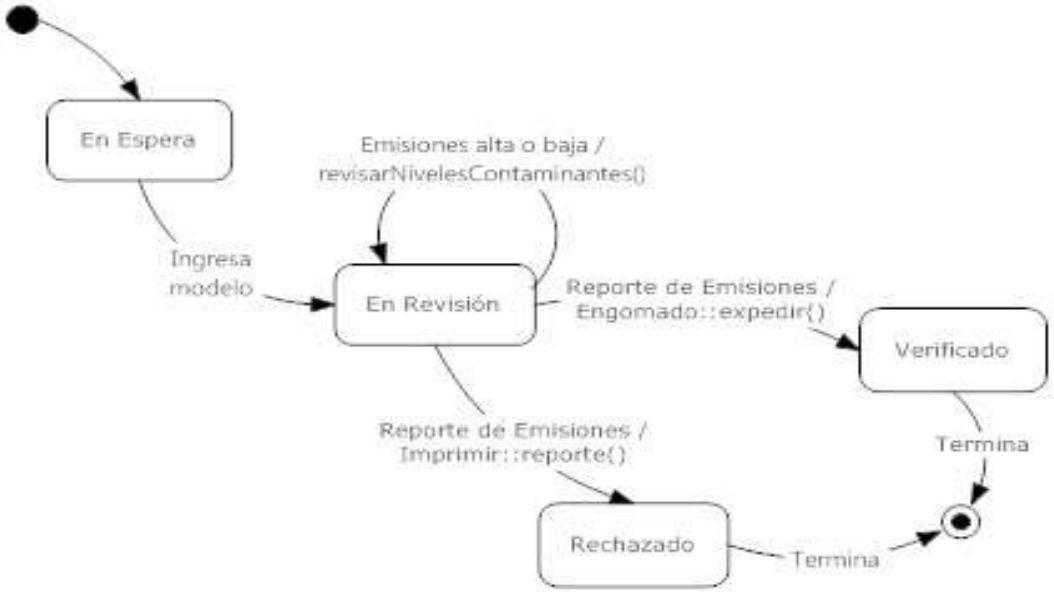
Una acción indica la realización de una operación, lo cual indica que se dispara un método y que no implica un tiempo de realización. En cambio, una actividad sí lleva un tiempo de realización.

Con base en esta metodología, una acción puede ser escrita usando la siguiente sintaxis, aplicada al sistema manejado:

- verfiicador.iniciar() - Una operación
- errorDelMedidorDeNiveles - Activación de un evento
- autorizarVerificacion - Inicia alguna actividad
- recahzarVerificacion - Termina algina actividad

El nombre tanto de la operación como del evento, debe ser descriptivo en cuanto a la función que realiza dentro del sistema.

Diagrama de transición de estados del VerifiCentro de Automóviles



Fuente: Booch, 1996.

En un diagrama de estado se establece el inicio con círculo con relleno, a partir de éste se muestra la transición hacia el primer estado y, posteriormente, en la transición, la acción que ocasiona el cambio de estado; puede haber situaciones en donde un estado

regresa a otro o es recursivo, finalmente, ya en los últimos estados, puede pasar a un estado de termino, indicado con el círculo con contorno blanco o puede regresarse a otro estado inicial dependiendo del comportamiento del objeto.

Acciones, condiciones de transición y estados anidados

Las acciones pueden estar asociadas con estados (Booch, 1996: 234). En esta metodología en un diagrama de transición de estados una acción se puede realizar en la entrada al estado o al salir del estado y esto se indica empleando la sintaxis siguiente:

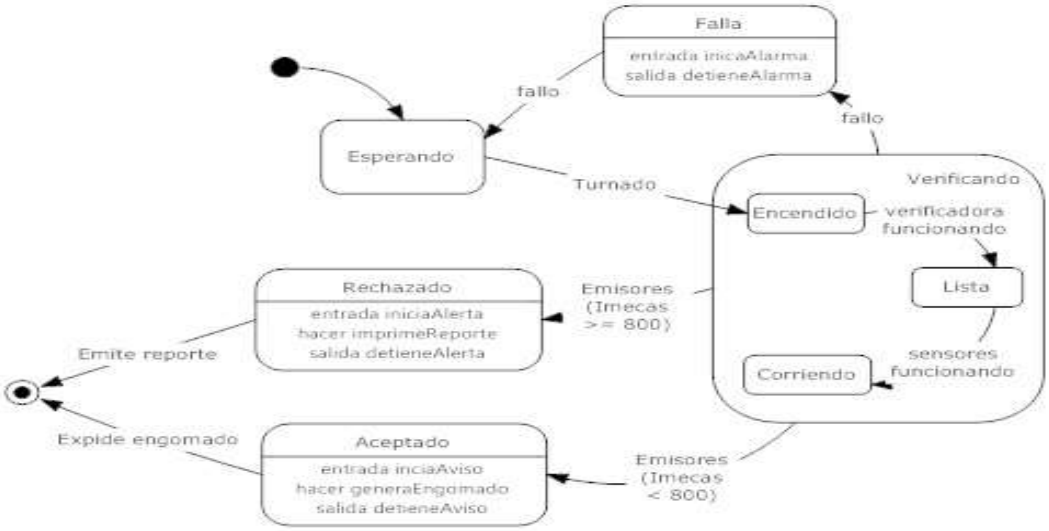
- entrada acelerarAutomovil – Realizar una actividad al entrar.
- salida detenerAutomovil - Realiza una operación al salir.

En las transiciones de estado, *es posible especificar cualquier acción tras las palabras clave de la entrada y salida* (Booch, 1996: 235).

Las actividades en esta metodología se realizan mientras se encuentra en el estado y se emplea, por ejemplo, la sintaxis siguiente:

- hacer Verificar - Lleva a cabo esta actividad.

Diagrama de estados con acciones, condiciones y un estado anidado



Fuente: Booch, 1996.

En este diagrama, cuando entra al estado de Rechazado, inicia la alerta avisando que ha sido rechazado de la verificación, durante el estado imprime el reporte con los niveles que detallan el rechazo y finalmente cuando abandona el estado detiene la alarma del rechazo. En el estado de fallo al entrar y salir, inicia y detiene la alarma, respectivamente. Durante el estado de Verificando tiene una serie de subestados o estados anidados por los cuales pasa antes de pasar a Aceptado o Rechazado.

En una transición de estado condicional se representa en una expresión booleana, se coloca dentro de paréntesis; por ejemplo, cuando revisa si los emisores son superiores o inferiores para pasar al siguiente estado.

“En general, una transición de estado dado tiene un evento y/o una condición. Se permite también que una transición de estado no tenga eventos asociados. Si la transición de estado es condicional, entonces la transición se dispara sólo en el caso de que la expresión se evalúe como verdadera” (Booch, 1996: 235).

Diagrama de objetos

Un diagrama de objetos “se utiliza para demostrar la existencia de los objetos y sus relaciones en el diseño lógico de un sistema” (Booch, 1996: 239). Los diagramas de objetos nos ayudan a determinar las relaciones que pueden darse entre los objetos (instancias de clases), sin importar el nombre de los objetos que están involucrados, modela en sí como trabajan los objetos dentro del sistema.

En el análisis, “se utiliza el diagrama de objetos para indicar la semántica de los escenarios principales y secundarios que proporcionan una traza de comportamiento del sistema. En el diseño, se utiliza el diagrama de objetos para ilustrar la semántica de los mecanismos en el diseño lógico de un sistema” (Booch, 1996: 239). El diagrama de objetos tiene dos elementos importantes, los objetos y las relaciones entre éstos.

Los objetos

Los objetos en esta metodología se representan con el icono en forma de una nube dentro de ella se le da un nombre y con una línea horizontal se separa de los atributos que describen al objeto.

Icono que representa un objeto

Fuente: Booch, 1996: 239.

El nombre del objeto puede seguir la siguiente sintaxis:

- Objeto — Solo el nombre del objeto. Por ejemplo: (Carlos)
- :Clase — Solo el nombre de clase. Por ejemplo: (:Persona)
- Objeto:Clase — Nombre de objeto y clase. Por ejemplo: (Carlos:Persona)

Los diagramas de objetos definidos en el nivel más alto del sistema tienen un uso global y también puede haber diagramas de objeto definidos en categorías individuales y así tener el alcance correspondiente. En un diagrama de objetos se le define solo el nombre de clase y sin nombre de objeto se considera anónimo.

El nombre dado para la clase de un objeto debe ser el de la clase verdadera (o cualquiera de sus superclases) en el ámbito del diagrama utilizado para instanciar el objeto, incluso si dichas clases resultan ser abstractas (Booch, 1996: 240). Generalmente, se piensa en un nombre de objeto que representa la realidad de las personas, cosas o cualquier otro elemento involucrado.

Las relaciones entre objetos

Los objetos se relacionan con otros objetos y es por esto que existe un tipo de asociación entre las clases. Los tipos de relaciones que existen esta metodología son: La relación normal, la relación es un que indica “herencia” y la relación es parte de la cual se indica como “tiene”, es posible que algunas las clases tenga una referencia a sí misma.

“La existencia de una asociación entre dos clases denota por tanto una vía de comunicación (es decir, un enlace) entre instancias de las clases, por la que un objeto puede enviar mensajes a otro” (Booch, 1996, p. 241). Y es por medio de mensajes que también se realizan operaciones necesarias entre los objetos para la funcionalidad deseada del sistema.

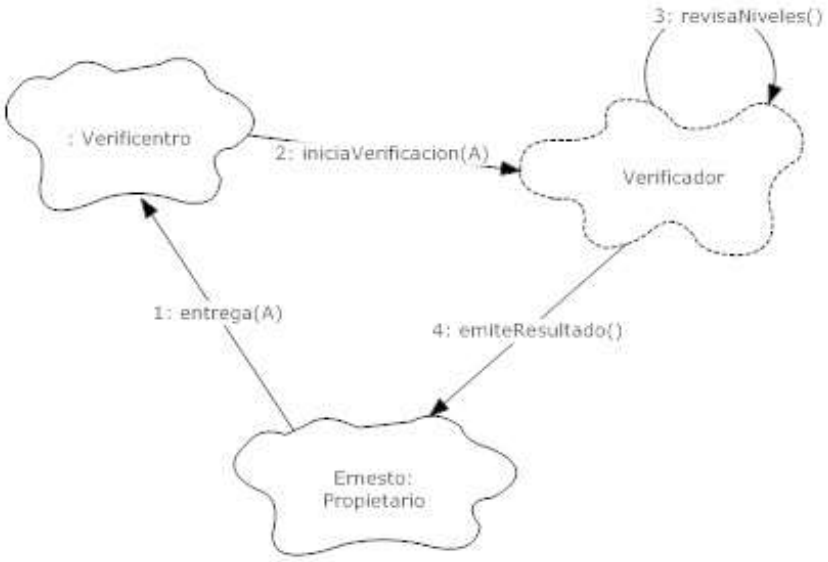
Una invocación de una operación independientemente del lenguaje de programación puede incluir parámetros o no y responder a la firma que coincida en prototipo de las operaciones pueden ser (Booch, 1996, p. 242):

- N( ) Solo el nombre de la operación

- R N(argumentos) Objeto de retorno, nombre, y argumentos actuales.

En un diagrama de objetos, para conocer mejor la secuencia de los mensajes se les numera, así se ordenan los mensajes, cuya secuencia inicia con 1, luego, 2, 3 y así sucesivamente cuantos se necesiten.

Diagrama de objetos del sistema de verificentro



Fuente: Booch, 1996.

Este ejemplo del diagrama muestra el escenario que sigue durante la ejecución de una función del sistema, en donde un propietario de su auto lo entrega en el verificentro, se inicia la verificación y el verificador revisa los niveles, al final le entrega el resultado de la verificación. En este sentido, durante el diseño, nos ayuda a saber cómo se da el orden de la comunicación entre los objetos involucrados.

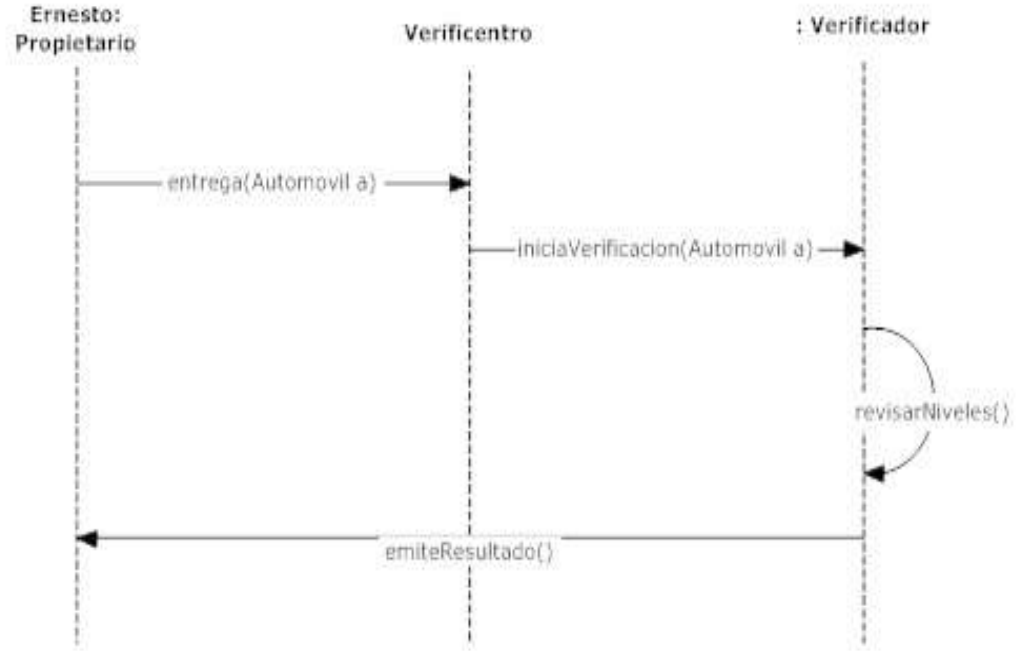
Diagramas de interacción

Un diagrama de interacción “se usa para realizar una traza de la ejecución de un escenario en el mismo contexto que el diagrama de objetos. Realmente, en gran medida, un diagrama de interacción es simplemente otra forma de representar un diagrama de objetos” (Booch, 1996: 248). El cambio que presenta el diagrama de interacción es la forma de visualizar los objetos y que los mensajes se muestren en forma ascendente mostrando con claridad cómo se van realizando. No contiene iconos nuevos, más bien aprovecha los existentes y solo se les incorporan datos adicionales en el flujo.

En este diagrama los objetos van en la parte superior, cada objeto por debajo lleva con una línea vertical punteada y con una flecha de izquierda a derecha se indica el mensaje, es decir, la función o método, que indica hacia que objeto va y en ocasiones un objeto también puede dar respuesta a las peticiones por lo que puede enviar un mensaje de regreso, estos se indican con una flecha de derecha a izquierda.

“El orden se indica por la posición vertical, el primer mensaje se muestra en la parte superior del diagrama, y el último mensaje se muestra en la parte inferior. Como resultado de ello, es innecesario el uso de números de secuencia” (Booch, 1996, p. 249). Estos diagramas cuando se crean en primera instancia se crean a partir de los eventos y no tomando en cuenta las operaciones, ya que están se disparan por la serie de eventos que se presentan, conforme se va avanzando se realizan los ajustes necesarios para mejorarlos.

Diagrama de interacción del sistema del verificentro



Fuente: Booch, 1996.

Los diagramas de interacción tienden a centrarse en los acontecimientos en lugar de las operaciones, ya que los eventos ayudan a definir los límites de un sistema en desarrollo.

Diagramas de módulo

Un diagrama de módulo “se utiliza para mostrar la asignación de clases y objetos a módulos en el diseño físico de un sistema. Un solo diagrama de módulo representa una vista de la estructura de los módulos de un sistema” (Booch, 1996: 250). Muestra por niveles la arquitectura, es decir, los elementos físicos del sistema. Existen dos elementos que se utilizan en diagrama de módulos y son los siguientes:

Los módulos

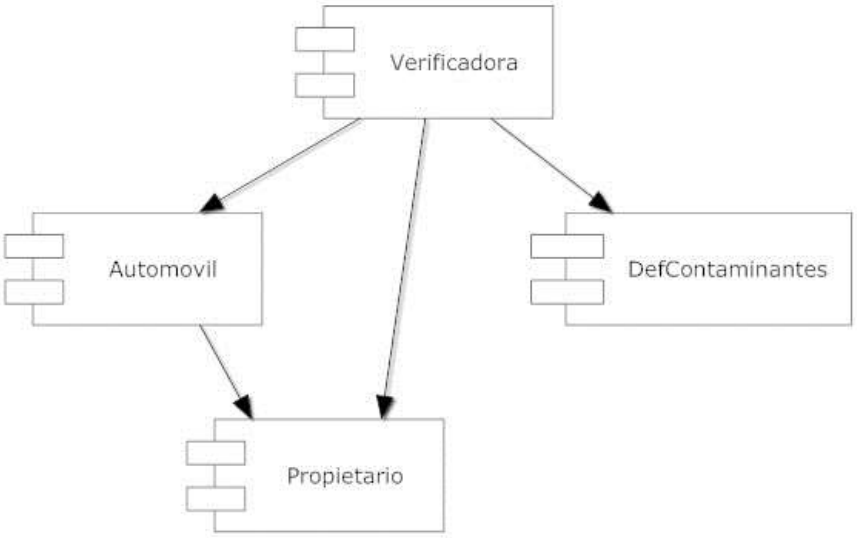
A cada módulo se la da un nombre y este representa un archivo físico dentro de un directorio, el nombre debe de ser único en el contexto del sistema. Un módulo representa

en sí la definición de una clase independientemente del lenguaje y debido a esto se le puede indicar el nombre del archivo fuente y su extensión.

Dependencias

“La única relación que puede tener entre dos módulos es una dependencia de compilación, representada por una línea dirigida el módulo en el que tiene la dependencia” (Booch, 1996, p. 251). Una dependencia en el ámbito de un sistema de archivos y en un sistema indica que existen archivos que emplean otros archivos y que deben estar compilados para que puedan operar.

Diagrama de modulo del sistema del verificentro



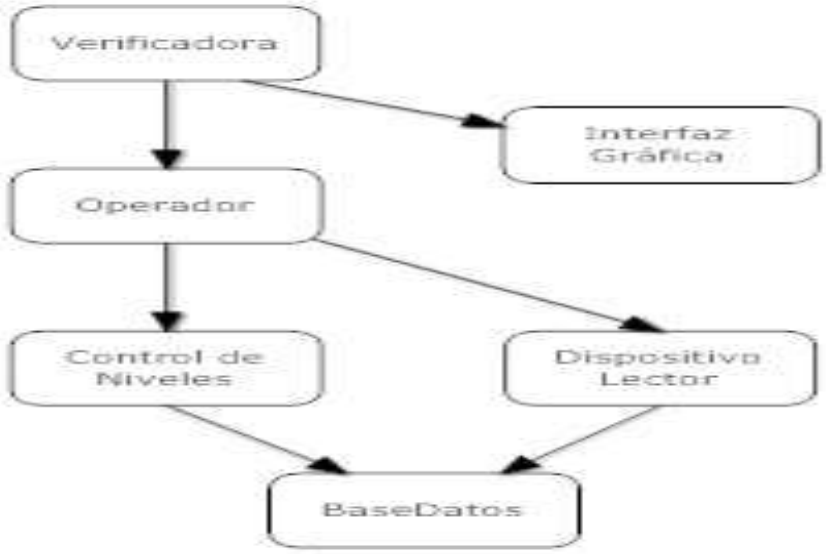
Fuente: Booch, 1996.

El diagrama anterior muestra cuatro módulos. Un módulo es DefContaminantes que es una especificación y proporcionan tipos y constantes. Los otros tres módulos muestran la relación entre ellos y contienen las definiciones generales del sistema. En este caso, el cuerpo de verificadora depende de la especificación de definición de contaminantes, que a su vez depende de la especificación del propietario y del automóvil para realizar la verificación del automóvil.

Subsistema

Los subsistemas *sirven para dividir el modelo físico de un sistema. Un subsistema es un agregado que contiene otros módulos y otros subsistemas* (Booch, 1996: 253). Se puede dar una restricción de acceso entre los módulos dependiendo del nivel de acceso que se quiera dar y también con este diagrama se logra comprender cómo va a ser la arquitectura y como va a ser su estructura jerárquica.

Diagrama de módulo de alto nivel del sistema del verificentro.



Fuente: Booch, 1996.

Otros módulos pueden ser parte de la implementación del subsistema, lo que significa que no están destinados a ser utilizados por cualquier otro módulo fuera del subsistema. Por convención, todos los módulos en un subsistema se consideran públicos, salvo que se defina de otra manera.

En la práctica, un diagrama de módulo de alto nivel contiene subsistemas en el más alto nivel de abstracción. A través de este esquema un desarrollador llega a entender la arquitectura general de un sistema físico.

Diagramas de procesos

Un diagrama de proceso *se utiliza para mostrar la asignación de procesos a procesadores en el diseño físico de un sistema. Un diagrama de procesos único representa una visión de la estructura de procesos de un sistema* (Booch, 1996: 255). Este diagrama es útil para diseñar el arreglo físico de los procesadores y dispositivos que sirven dentro de la tecnología usada en la ejecución del sistema. Los empleados en este diagrama son los procesadores, los dispositivos y sus conexiones.

Los procesadores

Un procesador *es un fragmento de hardware capaz de ejecutar programas* (Booch, 1996: 255). A cada procesador se le debe dar un nombre y éste representa un elemento de hardware.

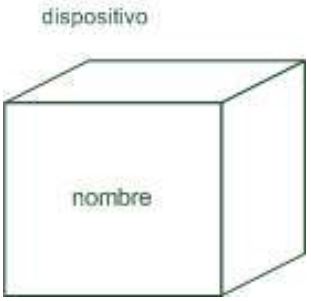
Icono de procesador

Fuente: Booch, 1996, p. 255.

Los dispositivos

Un dispositivo *es un fragmento de hardware incapaz de ejecutar programas* (Booch, 1996: 255). También a los dispositivos se les debe dar nombre.

Icono de dispositivo



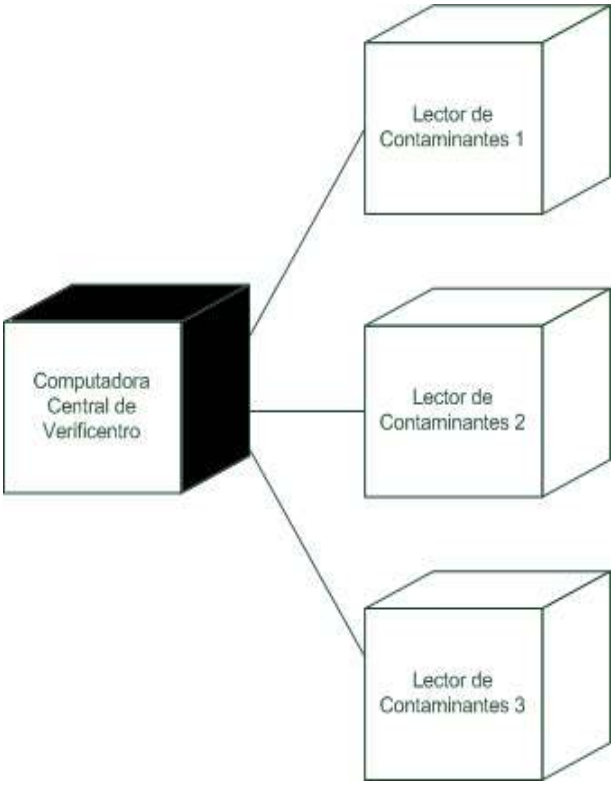
Fuente: Booch, 1996: 255.

Las conexiones

Mediante una línea dirigida, se puede indicar la conexión entre un dispositivo y un procesador, un procesador a procesador, o de dispositivo a otro dispositivo (Booch, 1996: 255). Las conexiones pueden representar desde una conexión por cable o también sin cable (wireless) como las comunicaciones por satélite. La conexión indica bidireccional por defecto, pero si es unidireccional con una punta de flecha también puede dársele un nombre a la conexión.

Por ejemplo, dentro del sistema del Verificentro se utilizan varios dispositivos en cada unidad verificadora, que son los sensores de niveles, y éstos los envían a la computadora central con la lectura de cada automóvil.

Diagrama de procesos del sistema del verificentro



Fuente: Booch, 1996.

La configuración de hardware de un sistema a veces es compleja y puede implicar jerarquías complejas de procesadores y dispositivos. En algunos casos quizá sea necesario representar grupos de procesadores, dispositivos y sus conexiones de forma anidada, dependiendo de la composición que refleje su arquitectura.

2.2 Jacobson (Objectory)

Objectory es una palabra compuesta de “Object Factory” que significa *fábrica de objetos*, creada por la compañía Objectory System, fundada en 1987 por Jacobson en Suecia. Es una extensión del método Ericsson. En el desarrollo de software se incorporan los casos de uso. En éste, el desarrollo o flujos de trabajo se representan mediante una serie de seis modelos (Modelado de caso de uso, Modelado de dominio de objetos, Análisis de robustez, Diseño, Implementación y Pruebas) diferentes durante sus cinco fases (Análisis de requerimientos, Análisis de robustez, Diseño, Implementación y Pruebas), siendo el orden de las tareas de izquierda a derecha. El sistema de desarrollo es un proceso iterativo donde todo es apto al cambio, e incluso algunas tareas pueden cambiar de orden.

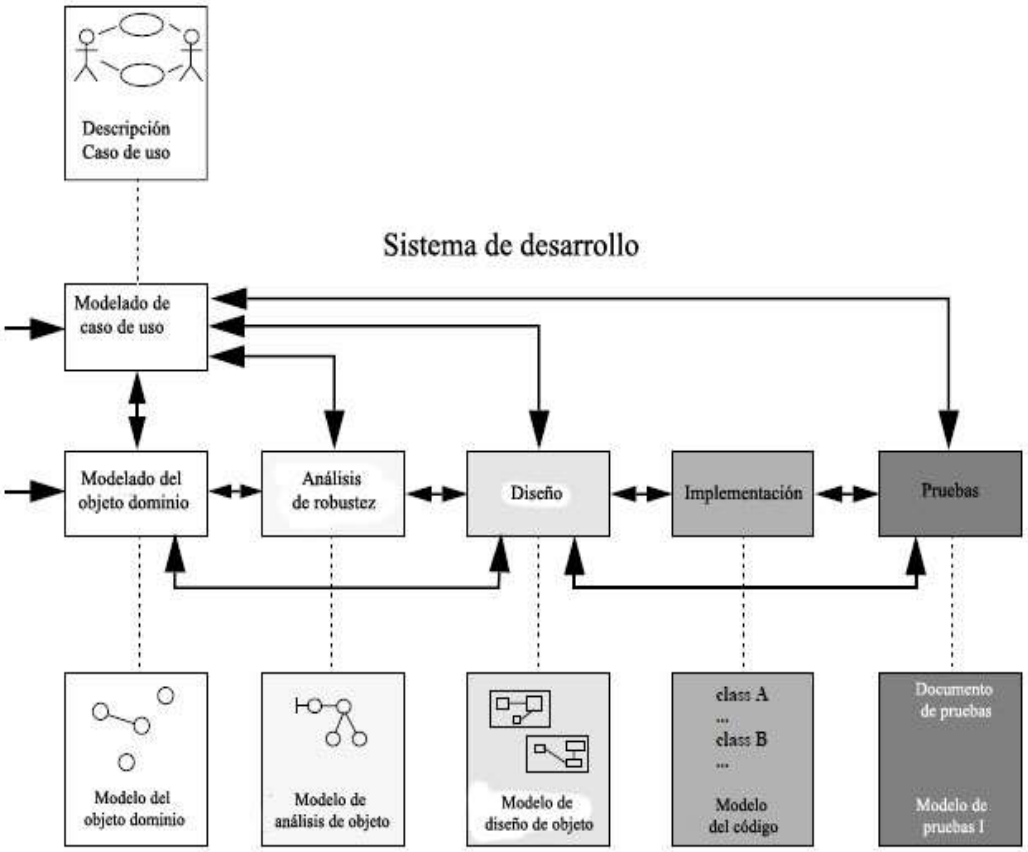


Figura. Procesos en Objectory y sus conexiones.

En la **especificación de requerimientos**, las dos descripciones importantes a crear son: el modelo de casos de uso y el modelo del dominio del objeto. Estas son fundamentales porque describen el entorno y la funcionalidad que se espera del sistema.

En el modelo de análisis se crea un análisis de robustez, que permite cambios posteriores. Objectory es clasificado como un método de diseño, pero se comienza con el modelo de diseño. El modelo de diseño se basa en el modelo de análisis e identifica los objetos y sus propiedades, tales como atributos y métodos. La relación entre objetos debe ser examinada cuidadosamente. Es importante considerar el entorno del sistema para ser implementado con sus limitaciones y posibilidades. El modelo del diseño es transferido frecuentemente a una herramienta de programación y pasa a un código fuente, y esto es el modelo de implementación. La última fase son las pruebas. El código

escrito debe funcionar. Entre otras cosas, se utiliza el modelo de casos de uso de prueba para verificar que el sistema desarrollado funcione bien.

FASES

1. Análisis de requerimientos

La primera transformación es desde la especificación de requerimientos a un modelo de requerimientos. El modelo de requerimientos debe especificar todas las funcionalidades del sistema. Esta consiste de:

- **Un modelo de caso de uso**.

  Explícitamente se describe cada caso de uso en orden para que sea claro cómo los usuarios y el sistema van a interactuar unos con otros, y cómo el sistema actúa en cada caso. Es importante observar que un actor no es idéntico a un usuario. Un usuario puede tener varios roles, por lo tanto estos roles deben de ser clasificados en actores. Los casos de uso deben tener nombres descriptivos, que definan un proceso. Se deben definir las relaciones entre el actor y el caso de uso y establecer la asociación entre éstos.

  Por ejemplo: Cuando un cliente o una arrendadora de autos va a comprar un automóvil, debe comprar el seguro con alguna aseguradora, posteriormente realizar el pago del automóvil y, al final, la agencia le entrega el automóvil.

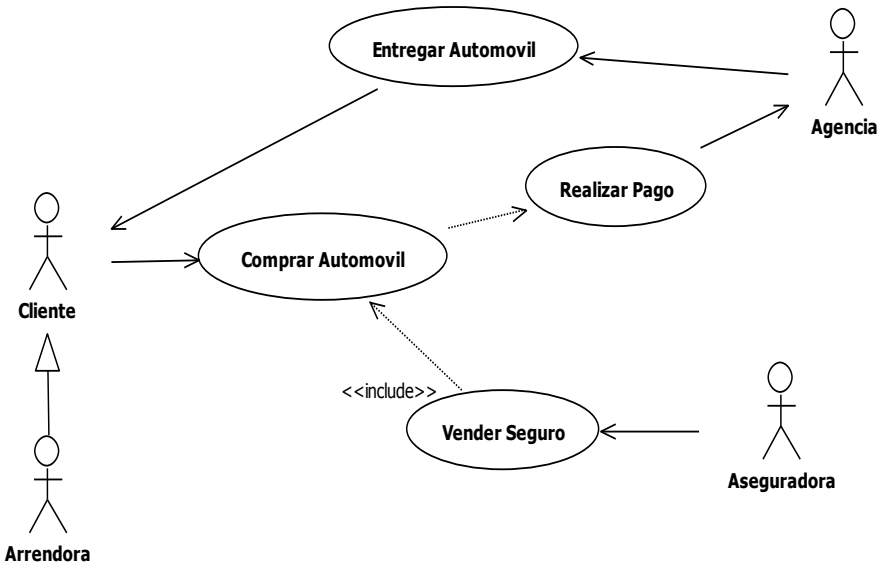


Figura. Empleo de asociaciones en un diagrama de caso de uso.

Un cliente puede ser una persona física o una persona moral, como una arrendadora, por lo tanto, un *cliente* es una generalización.

- **Una descripción de interface.**

  La interface debe de dar una correcta imagen del sistema y la lógica de operación. Un método bueno es tratar con la propuesta de interface a través de prototipos.

- **Un modelo de dominio de problema.**

  Desarrollar una vista lógica del sistema mediante objetos para entender los conceptos del dominio del problema, es decir, objetos que son parte del entorno de la aplicación y que el sistema debe manejar. Para identificar los objetos podemos hacer las siguientes preguntas:

  ¿Cuál es la lista de conceptos?
  ¿Qué cosas naturalmente ocurren que el sistema debe conocer?
  ¿Qué es lo que el sistema debe de recordar?
  ¿Qué se necesita conocer de los usuarios y sus características?

Por ejemplo, del siguiente enunciado: Los clientes de video club rentan películas, de éste podemos encontrar los objetos: cliente, videoclub y película.

Todos los conceptos en Objectory tienen un símbolo que se utiliza para hacer correlaciones, de esta forma es más fácil de ver y entender. Aquí se presenta una figura con los símbolos más comunes.

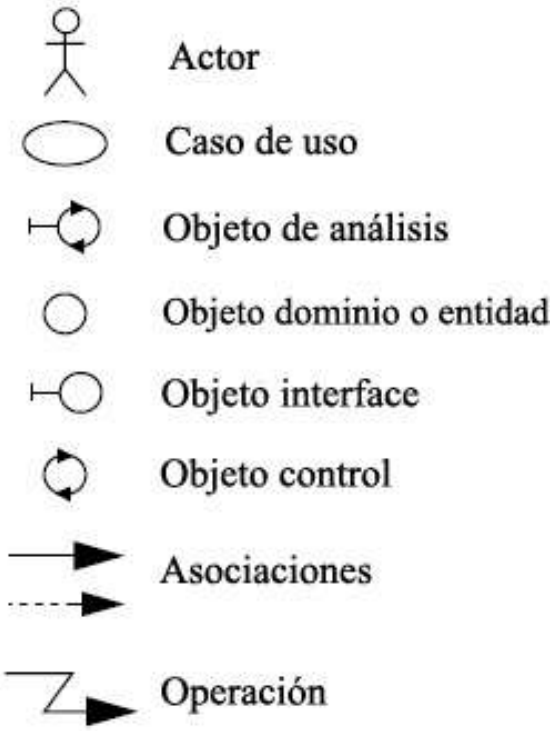


Figura. Símbolos más comunes empleados en Objectory.

2. Análisis de robustez

El objetivo es estructurar el sistema, independientemente del entorno de ejecución. Se utiliza un modelo lógico en un entorno ideal. Se espera que el modelo sea robusto, fácil de mantener y modificar. El requisito de la modificación es importante, ya que un pequeño cambio no debería obligar a la modificación de gran parte del sistema. El procedimiento normal es pasar el caso de uso, uno por uno, y dividir el comportamiento del sistema entre los objetos de interfaz, de entidad y de control.

El comportamiento no necesariamente tiene que ser dividido en operaciones, una descripción general de cada función o responsabilidad de los objetos es suficiente. Las relaciones entre los objetos, es decir, las asociaciones, también deben ser descritas.

Figura. Objetos que separan la funcionalidad del sistema.

Un ejemplo de la notación del diagrama de robustez, suponiendo que un cliente quiere consultar la disponibilidad de un cuarto en un hotel, el modelo sería el siguiente:

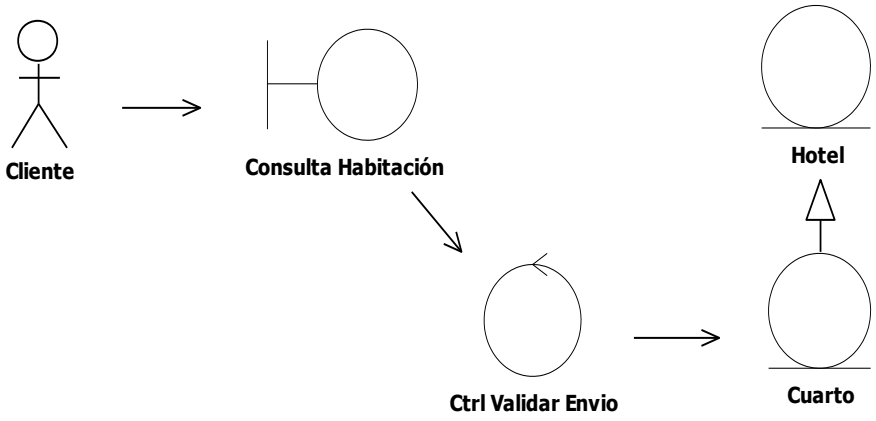


Diagrama de robustez "Un cliente consultando la disponibilidad de un cuarto"

Las acciones de los actores, a través de los objetos de la interfaz, se traducen en hechos en el sistema, y los eventos (que son interesantes para el actor) deben ser traducidos y representados en forma adecuada. Los objetos-entidad realizarán la administración de la información que debe mantenerse en el sistema durante un tiempo mayor, normalmente abarca más las limitaciones de un caso de uso y también deben capturar el comportamiento que existe, naturalmente. Los objetos controladores tienen la

responsabilidad de realizar el comportamiento adecuado y comunicar los objetos de interfaz con los de entidad.

3. Diseño

La fase de diseño consiste en traducir el modelo de análisis directamente en el modelo de diseño. Cada objeto en el modelo de análisis comienza en un bloque. Un bloque debe ser equivalente al concepto del módulo del lenguaje de programación que el sistema va a aplicar El bloque se convierte en una abstracción del sistema a implementar. Es la transformación de los objetos de análisis directamente en un bloque de subvenciones con canales trazados entre los modelos. Si uno desea hacer cambios en el análisis, es fácil ver qué cambios hay que hacer en el diseño e implementación. El objetivo es cambiar lo menos posible después de esta transición. Los cambios del modelo de diseño ideal que se puede hacer son:

- Agregar nuevos bloques.
- Remover bloques.
- Cambiar bloques (división o fusión).
- Cambiar asociaciones entre bloques.

Es una buena idea agregar nuevos bloques para encapsular el medio ambiente. Uno puede verse obligado a quitar bloques en función del entorno de ejecución, pero en general se debe hacer en el modelo de análisis. Es común que las asociaciones entre los bloques sean alterados por el entorno de aplicación, por ejemplo la sincronización y la comunicación. Para asegurarse de que la complejidad del sistema se encuentra bajo control, durante el desarrollo, es posible introducir un paquete o paquetes, si se desea una estructura jerárquica.

Para refinar el modelo de diseño, se hace mediante el estudio de cómo los bloques se comunican entre sí durante la ejecución, esto se presenta en un diagrama de interacción. Los bloques extremos entre sí desencadenan una secuencia de eventos en el bloque de

recepción. Los diagramas de interacción son, como muchos otros, la base de información de los casos de uso. El caso de uso *describe una secuencia de estímulos entre los diferentes bloques*. Cuando todos los casos han sido revisados, se ha hecho una descripción completa de la comunicación externa de los bloques y se han descrito sus interfaces.

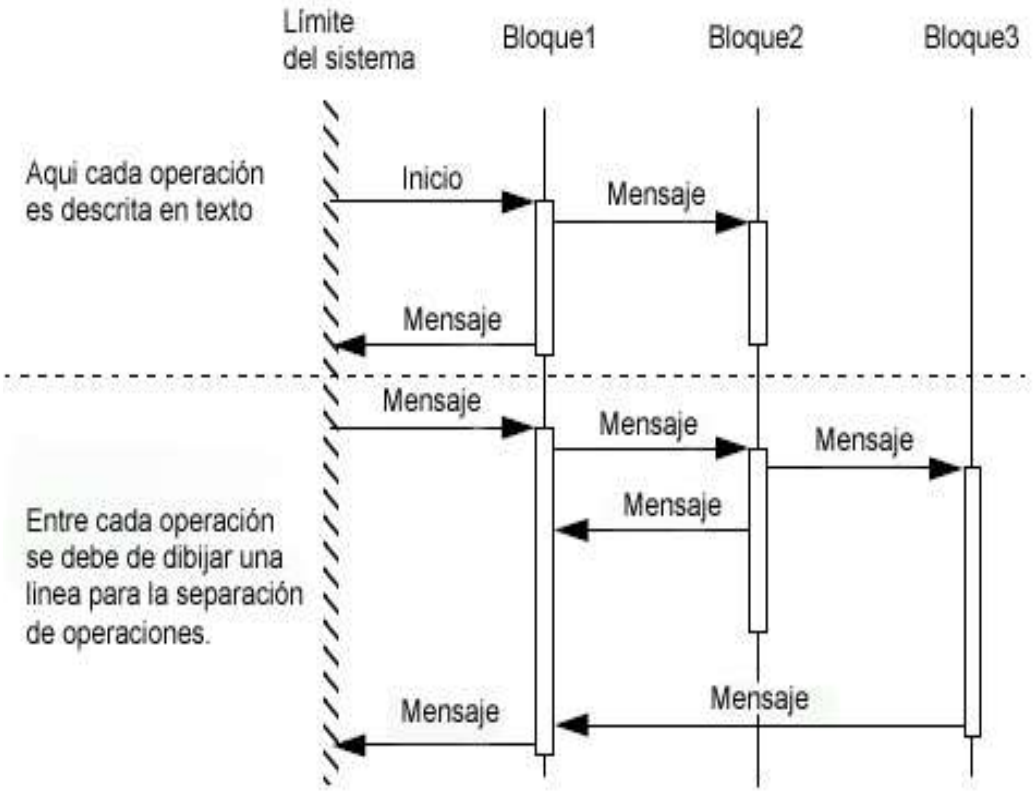


Figura. Ejemplo de un diagrama de interacción.

4. Implementación

La implementación se inicia cuando las interfaces de los bloques se establecen. Se empieza con la implementación de los bloques en la parte superior de la jerarquía de herencia. De la interacción de diagramas se pueden encontrar los métodos que se necesitan y es posible determinar las interfaces fijas entre los bloques y llevar a cabo el diseño de bloques en paralelo.

- **Diagramas de estado.**

  Sirve como previo a la codificación real. Esta es una manera de describir la funcionalidad final y crear un mejor entendimiento sin tener el código fuente. En el diagrama a partir de los mensajes que se reciben se decide qué acciones se toman.

- **Codificación.**

  Finalmente los diagramas de estado se convierten al código en el lenguaje de programación elegido.

5. Pruebas

La última parte del ciclo de desarrollo son las pruebas, se corrigen y revisan la mayoría de los errores que son comunes durante el desarrollo del sistema. El cliente debe aprobar el producto final. Durante la revisión y la verificación, los casos de uso se utilizan una vez más para controlar, como guía de lo que el sistema debe realizar y lo que se espera de éste.

Al depurar el modelo implementado, se pueden dividir los errores en tres diferentes categorías de acuerdo con la IEEE 729.

- **Falla** - Surge cuando un programa no es capaz de realizar lo que se espera.
- **Falta** - Reside en una acción que se necesita y no se incluyó. Una falta puede provocar una falla.
- **Error** - Un error en el código se origina en una decisión humana inesperada.

Las pruebas se dividen en dos categorías:

Pruebas unitarias

Las pruebas unitarias son la forma más simple de verificación cuando se prueba una clase, un bloque o un paquete de servicios. Las pruebas unitarias a menudo se dividen en dos partes: pruebas de caja blanca y pruebas de caja negra **.**

Pruebas de caja blanca	Prueba de caja negra
La prueba de caja blanca se aplica sobre el código. El objetivo es revisar las condiciones y ciclos para ver si están contenidas todas las reglas y operaciones. Este tipo de prueba lleva mucho tiempo, pero a menudo se descubren errores en el código del programa.	La prueba de caja negra se aplica sobre el funcionamiento y no cómo se lleva a cabo. Este debe ser muy parecido a lo que puede darse en la vida real.

Pruebas integrales

Las pruebas integrales se realizan en equipo; previamente se define el objetivo a evaluar, las partes o funciones, se les identifican en el programa y se especifican las condiciones a evaluar; al final, se revisan los errores encontrados. Los cuatro pasos a realizar son los siguientes:

a. Planificación	Las pruebas de integración se deben realizar en el entorno operativo del sistema. También tiene que decidir qué nivel de la prueba se hará en adelante, y que directrices o guías se deben utilizar.
b. Identificación	Se decide qué se va a probar y qué recursos se van a necesitar; ésta también se divide en partes más pequeñas, éstas pueden ser módulos del software o bloques en específico como procedimientos o funciones en el código.
c. Especificación	Aquí las condiciones de ensayo, las pruebas de usuario y equipos de prueba necesarios, se especifican. La especificación se utiliza para planificar y llevar a cabo la prueba.

d. Realización	Durante las pruebas, los errores que se producen se detallan. Los errores son reportados y corregidos. Si es necesario, la prueba se ejecutará nuevamente.

Resultado

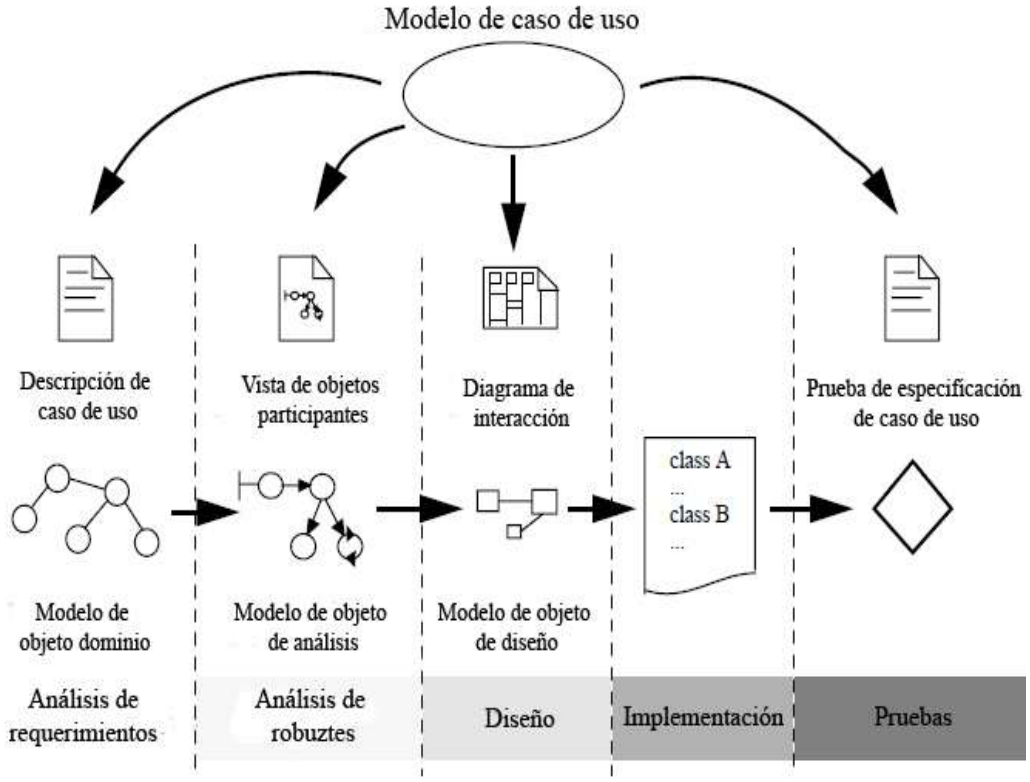


Figura. Resultado de cada fase en el proceso es presentado.

Durante los años 1995 a 1997, el proceso Objectory se desarrolla en Rational Software Corporation, pues esta empresa fue comprada a Objectory AB, y sale al mercado el Proceso Objectory de Rational (Rational Objectory Process, ROP), posteriormente en UML, sentando las bases para realizar los casos de uso.

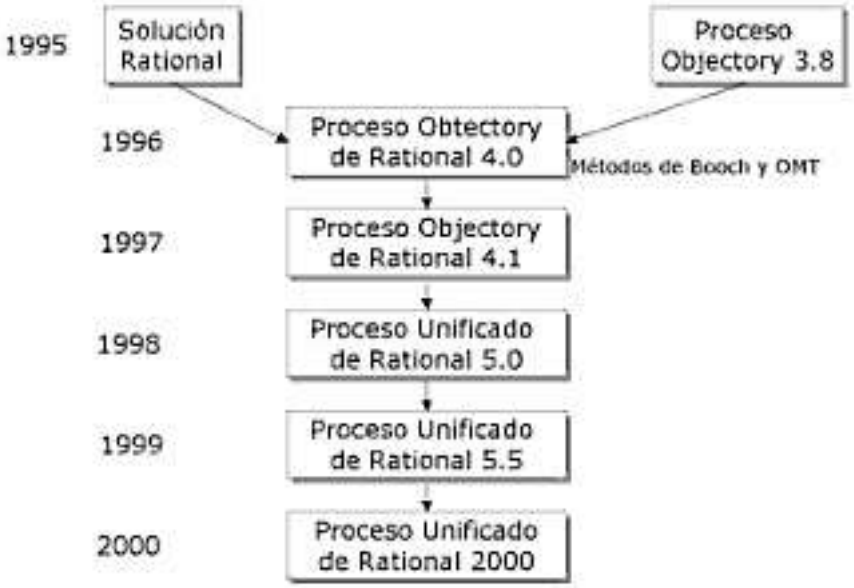


Figura. Evolución de Objectory hacia el proceso unificado de Rational.

2.3 Rumbaugh (técnica de modelado de objetos)

Es una metodología para el desarrollo, creada por James Rumbaugh, Michel Blaha, Willian Premerlani, Frederick Eddy y William Lorensen en 1991. Esta es una metodología para el desarrollo orientada a objetos y contiene una notación para representar los objetos, a ésta se le conoce como Técnica de Modelado de Objetos (OMT). La metodología *consiste en construir un modelo de un dominio de aplicación y entonces agregar los detalles de implementación para esto durante el diseño del sistema* (Rumbaugh **et al**., 1991: 5).

Etapas de OMT

Cada etapa del proceso se transforma de una entrada hacia una salida, realizando diferentes niveles de abstracción que finalmente representa la solución del problema. El proceso de desarrollo completo de OMT consiste de cuatro fases: análisis, diseño de sistema, diseño de objeto e implementación del software.

- **Análisis**. Entender y modelar la aplicación y su dominio en términos del modelo de objetos, del modelo dinámico y del modelo funcional.
- **Diseño del sistema**. Determina la arquitectura general del sistema en términos de subsistemas, las tareas concurrentes y de almacenamiento de datos.
- **Diseño de objeto**. Ampliar, mejorar y optimizar el modelo de análisis. Durante esta etapa hay un cambio de conceptos de aplicación a los conceptos de computación.

- **Codificar**. Poner en práctica las clases de objetos en un lenguaje de programación.
- **Pruebas**. Las pruebas son incrementales en las clases de objetos, en cualquier etapa, tanto en lo probado como en lo no probado. La prueba se basa en los escenarios desarrollados en el marco del proceso de modelo dinámico.

Los modelos

En el modelo de objetos se describen los aspectos estáticos, estructural y los datos de un sistema, describiendo los objetos y sus interrelaciones. En el modelo dinámico se describen los aspectos temporales, de comportamiento y control de un sistema, captura los cambios que ocurren en los estados de distintos objetos con los acontecimientos que puedan ocurrir en el sistema. En el modelo funcional se describe los aspectos transformacionales de datos y funcionales de un sistema, describe el flujo de datos y los cambios que se producen en los datos en todo el sistema. Así que cada sistema tiene estos tres aspectos. Cada modelo describe un aspecto del sistema, pero contiene referencias a los otros modelos.

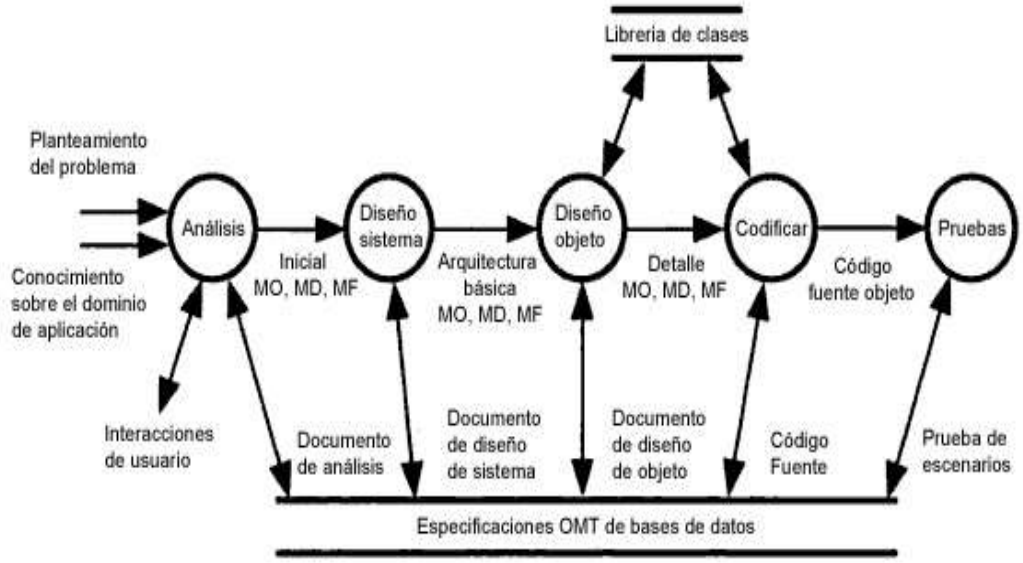


Figura de entrada y salida del proceso de desarrollo usando OMT.

La mayor parte de la modelización se realiza en la fase de análisis y diseño. En ambas fases, se desarrollan los tres modelos básicos. Modelo de objeto, modelo dinámico y el modelo funcional.

Modelo objeto

El modelo de objetos es el más importante, ya que describe la estructura de los objetos en el sistema, su identidad, sus relaciones con otros objetos, sus atributos y sus operaciones. El modelo de objeto muestra la vista principal de cómo el mundo real en el que interactúa el sistema se divide y la descomposición total del sistema. El modelo de objetos proporciona el marco en el que los otros modelos se colocan.

El modelo de objetos se representa gráficamente con un diagrama de objetos. El diagrama de objetos contiene clases interconectadas por líneas de asociación. Cada clase representa un conjunto de objetos individuales. Las líneas indican la relación de asociación entre las clases. Cada línea de asociación representa un conjunto de enlaces de objeto de una clase al objeto de otra clase.

Clases y objetos

Las clases se construyen sobre la base de la abstracción, donde se observa un conjunto de objetos similares y sus características comunes se enumeran. De todas ellas, las características de interés para el sistema bajo observación se toman y la definición de la clase se hace. Los atributos de ningún interés para el sistema se quedan fuera. Esto se conoce como abstracción.

En cada concepto del dominio del negocio en el mundo real es importante que en la aplicación deba ser modelado como una clase de objeto. Las clases se organizan en jerarquías que comparten una estructura común y el comportamiento y están asociadas con otras clases. Esto da lugar al concepto de herencia.

En OMT, las clases están representadas por una caja rectangular que puede ser dividido en tres partes. La parte superior contiene el nombre de la clase, en medio contiene una lista de los atributos y en la parte inferior contiene una lista de las operaciones.

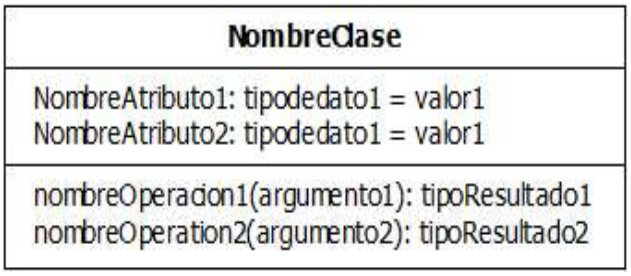


Figura de una clase

Un atributo es un valor de datos en poder de los objetos de una clase. Cada atributo tiene un valor para cada instancia de objeto. Este valor debe ser un valor puro, no un objeto. Los atributos se enumeran en la segunda parte de la clase. Los atributos pueden o no ser mostrados, sino que depende del nivel de detalle deseado. Cada nombre de atributo puede ser seguido por los detalles opcionales tales como el tipo y el valor predeterminado.

Una operación es una función o transformación que puede ser aplicada a/o por los objetos de una clase. Las operaciones se listan en la tercera parte de la clase. Las operaciones pueden o no ser mostradas, sino que depende del nivel de detalle deseado. Cada operación puede ser seguida por los detalles opcionales tales como lista de argumentos y tipo de resultado. El nombre y el tipo de cada argumento se pueden especificar. Una lista de argumentos vacía entre paréntesis indica explícitamente que no hay argumentos. Todos los objetos de una clase comparten las mismas operaciones. Cada operación tiene un objeto de destino como argumento implícito. Una operación puede tener argumentos, además de su objeto destino, que parametrizan la operación.

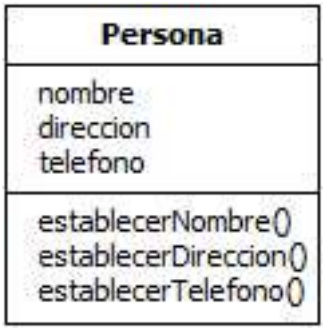


Figura de la clase *persona*

La figura muestra la clase persona con nombre, dirección y teléfono, que son sus atributos, y establecer Nombre, Dirección y Teléfono son las operaciones.

Un objeto es un concepto, abstracción o cosa con límites bien definidos y significado para el problema en cuestión. Un objeto tiene las siguientes cuatro características principales:

- Identificación única
- Conjunto de atributos
- Conjunto de estados
- Conjunto de operaciones (comportamiento)

La identificación única significa que cada objeto tiene un nombre único por el que se identifica en el sistema. El conjunto de atributos se refiere a que cada objeto tiene un conjunto de propiedades. El conjunto de estados se refiere a los valores de los atributos de un objeto que constituyen el estado del objeto. Cada objeto tendrá un número de estados, pero en un momento dado puede estar en uno de esos estados. El conjunto de operaciones se refiere a las acciones visibles que un objeto puede realizar. Cuando se realiza una operación, el estado del objeto puede cambiar.

En otras palabras, un objeto es una instancia de una clase. Una instancia de un objeto es un objeto particular que surge desde una clase. Por ejemplo, una clase *libro* puede tener diferentes tipos de objetos, tales como física, química y matemáticas.

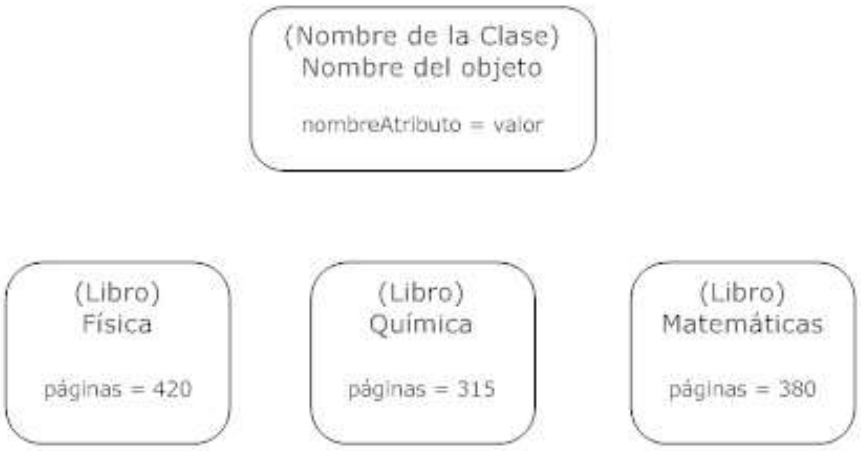


Figura de tres objetos de la clase libro

El concepto de objeto es derivado, porque a partir de una clase surgen uno o varios objetos.

Enlaces y asociaciones

Un enlace es una conexión física o conceptual entre instancias de objetos. En OMT un enlace se representa por una línea etiquetada con su nombre.

Diagrama que muestra el enlace entre dos objetos

Por ejemplo, una clase persona y una clase libro, aquí si una persona *Juan* lee un libro de *física*, aquí *leer* es el enlace entre los objetos.

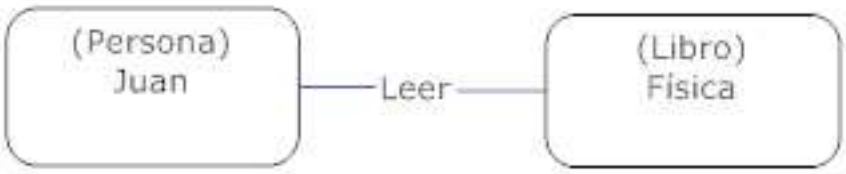


Diagrama del objeto Juan lee un libro de física

Una asociación describe una conexión física o conceptual entre instancias de objetos. En OMT el enlace es representado por una línea etiquetada con su nombre. Las asociaciones son bidireccionales por naturaleza.

Multiplicidad. Ésta especifica cuantas instancias de una clase pueden referirse a una sola instancia de una clase asociada. La multiplicidad, en otras palabras, indica el número de objetos relacionados.

Hay terminadores de línea especial para indicar ciertos valores de multiplicidad. Una bola sólida es el símbolo de "muchos", es decir, cero, uno o más. Una bola hueca indica "opcional", que significa cero o uno. La multiplicidad se indica con símbolos especiales en los extremos de líneas de asociación. En el caso más general, la multiplicidad se puede especificar con un número o un conjunto de intervalos. Si no se especifica símbolo de multiplicidad, significa una asociación uno-a-uno. Las reglas de multiplicidad se resumen a continuación:

- Línea sin ninguna bola indica uno-a-uno.
- Bola hueca indica cero o uno.
- Bola sólida indica cero, uno o más.
- Los números escritos en bola sólida tal como 1, 2, 6 indica 1 o 2 o 6.
- Los números escritos en bola sólida tal como 1 +, indica 1 o más, 2 +, indica 2 o más, etc.

Una asociación unaria es con la misma clase. Por ejemplo, una persona le enseña a una o más personas.

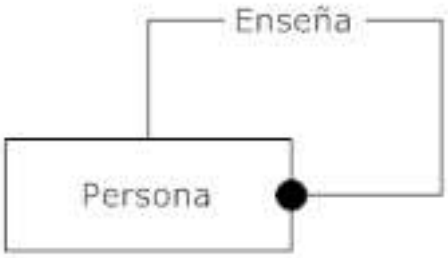


Diagrama con la asociación unaria

Una asociación binaria, por ejemplo, es una “Persona se sienta en una silla”. Así la multiplicidad de esta asociación es uno a uno.

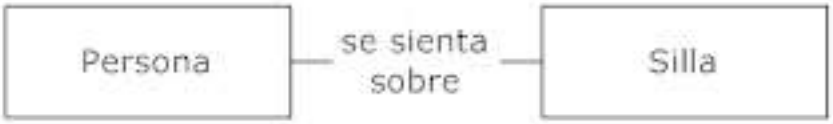


Diagrama con la asociación binaria uno a uno.

Ejemplo de una asociación binaria muchos a muchos es "suministros de proveedores de piezas". Un proveedor puede suministrar muchas partes o una parte puede ser suministrada por muchos proveedores. Así multiplicidad de la asociación es de muchos a muchos.

Diagrama con la asociación binaria de muchos a muchos

La asociación binaria opcional es una "persona posee un pasaporte". O bien una persona puede tener un pasaporte o no tener pasaporte pero un pasaporte puede ser de una persona. Así la multiplicidad de esta asociación es uno a una opcional.

Diagrama con la asociación binaria opcional entre tres clases

La asociación ternaria es una asociación entre tres clases. El símbolo en OMT para la relación ternaria y de las asociaciones es un diamante con líneas que conectan a las clases relacionadas. Por ejemplo, un programador escribe en un lenguaje código para el proyecto.

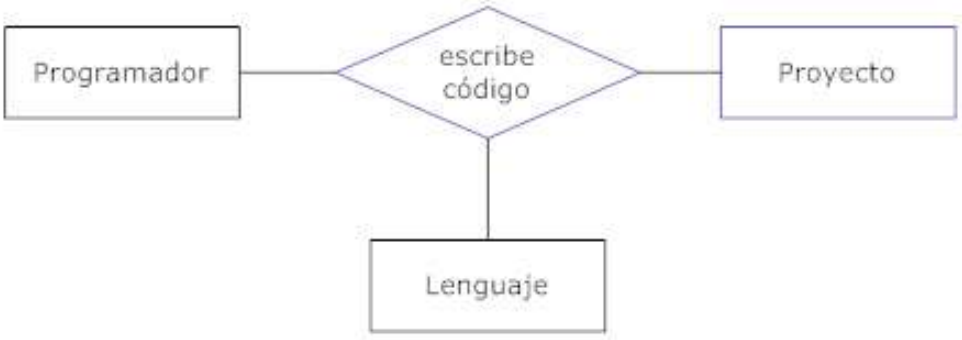


Diagrama con la asociación ternaria entre las clases

Un calificador es un atributo asociación. Una asociación cualificada relaciona dos clases de objetos. El calificador se dibuja como una pequeña caja en el extremo de la línea de asociación, cerca de la clase a la que califica. El rectángulo calificador es parte de la asociación, no de la clase. El calificador distingue entre el conjunto de objetos en el extremo de "muchos" de una asociación. Una asociación cualificada también se puede considerar una forma de asociación ternaria.

Por ejemplo, una persona puede ser un objeto relacionado con el objeto banco, un atributo de esta asociación es el número de cuenta (numeroDeCuenta). El número de cuenta es el calificador de esta asociación.

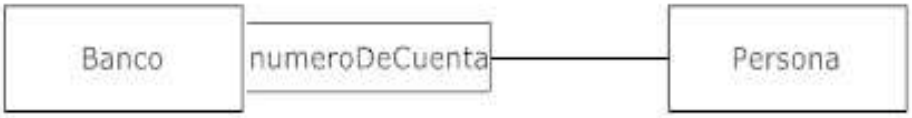


Diagrama con un calificador entre el objeto banco y una persona

La agregación es otra relación entre las clases. Es una forma de acoplado en la asociación, indica que es "parte de un todo" o "es parte de", en la semántica de la relación en donde los objetos representan algo que se asocia a un objeto que representa todo el conjunto. Las agregaciones se dibujan como asociaciones, excepto un pequeño diamante hueco que indica el final de montaje de la relación. La clase opuesta al lado del diamante es parte de la clase en el lado de diamante. Por ejemplo, los jugadores son parte de un equipo.

Diagrama con la relación de agregación. Los jugadores son parte del equipo

Herencia

La herencia es una manera de formar nuevas clases utilizando clases que ya han sido definidas. La herencia ayuda a reutilizar código existente con poca o ninguna modificación. Las nuevas clases, conocidas como clases derivadas o clases hijas o subclases, heredan los atributos y el comportamiento de las clases preexistentes, que se conocen como clases base o clases para padres o superclases. La relación de herencia de clases de sub y súper da lugar a una jerarquía.

La herencia es un tipo de relación que significa "es-un" entre dos clases. Por ejemplo, un estudiante es una persona, una silla es un mueble; un loro es un pájaro, etc., en todos éstos, la primera clase (es decir, Estudiante, Silla, Loro) heredan las propiedades de la segunda clase (Persona, Mueble, Pájaro), respectivamente.

Ejemplo de una herencia: Administrador es un empleado. La clase *administrador* hereda las características de la clase *empleado*.

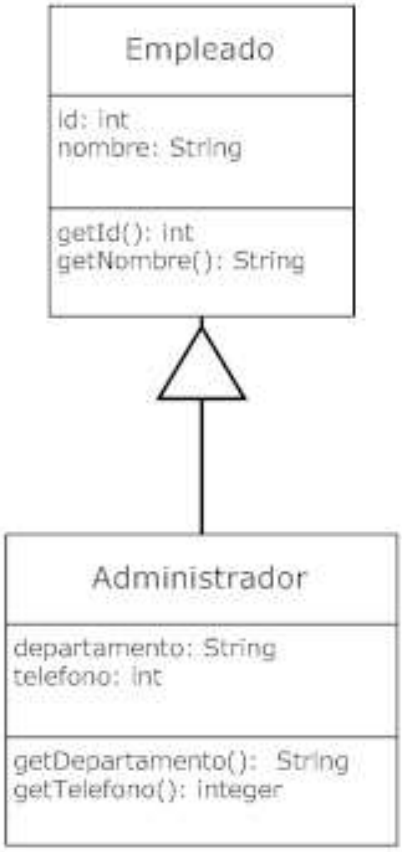


Diagrama de la subclase administrador que hereda de la clase empleado

Se debe tomar en cuenta un enfoque, en donde todas las clases que son producidas en este proceso de diseño son parte del mismo dominio del negocio o contexto, es decir, que describen las cosas claramente usando sólo una terminología.

Modelo dinámico

El modelo dinámico describe aquellos aspectos del sistema que cambian con el tiempo. Se utiliza para especificar y aplicar aspectos de control del sistema. Representa los estados, transiciones, eventos y acciones. Los principales conceptos son estados, transiciones entre estados y eventos para desencadenar transiciones. Las acciones

pueden ser modeladas como algo que ocurre dentro de los estados. La generalización y agregación (concurrencia) son relaciones predefinidas. Los resultados de un modelo dinámico son escenarios, utilizando diagramas de traza de eventos y diagramas de estado.

El modelo dinámico da seguimiento a los eventos, el cual incluye diagramas que describen los escenarios. Un evento es un estímulo externo de un objeto a otro, lo cual ocurre en un punto particular en el tiempo. Un evento es una transmisión unidireccional de información de un objeto a otro. Un escenario es una secuencia de eventos que se producen durante una ejecución particular de un sistema. Cada ejecución básica del sistema debe ser representado como un escenario.

El modelo dinámico se representa gráficamente mediante diagramas de estado. Un estado corresponde al intervalo entre dos eventos recibidos por un objeto y describe el "valor" del objeto de ese período de tiempo. Un estado es una abstracción de los valores de los atributos de un objeto y enlaces, donde grupos de valores se agrupan en un estado de acuerdo con las propiedades que afectan el comportamiento general del objeto. Cada diagrama de estado muestra las secuencias de estado y evento permitidos en un sistema para la clase de un objeto. Los diagramas de estado se refieren también a otros modelos: acciones que corresponden a funciones en el modelo funcional, y eventos que corresponden a operaciones sobre los objetos en el modelo de objetos.

Escenario

Un escenario es una secuencia de eventos que se producen durante una ejecución particular de un sistema. Cada ejecución básica del sistema debe ser representado como un escenario. El alcance de escenario puede variar. Se pueden incluir todos los eventos en el sistema o puede incluir sólo los sucesos generados por ciertos objetos. Un escenario puede ser escrito como una lista de instrucciones de texto.

Por ejemplo, un escenario para utilizar un cajero ATM para retirar dinero. Cada evento transmite información desde un objeto a otro. Inicia el evento "el cajero automático pide

al usuario que inserte una tarjeta" transmite una señal desde el cajero automático para el usuario. El próximo evento es "el usuario inserta una tarjeta de dinero en efectivo". El próximo evento es "el ATM acepta la tarjeta y lee su número de serie." Y así sucesivamente.

El escenario normal de ATM	
1	El cajero ATM pregunta al usuario para insertar su tarjeta. El usuario inserta una tarjeta.
2	El cajero ATM acepta la tarjeta y lee su número de serie.
3	El cajero ATM solicita su contraseña. El usuario introduce su contraseña.
4	El cajero verifica el número de serie y la contraseña en la sucursal. La sucursal revisa con el banco y notifica al cajero automático su aceptación.
5	El cajero ATM le solicita el monto. El usuario introduce la cantidad.
6	El cajero procesa la solicitud y expide la cantidad necesaria de dinero.

Diagrama de traza de eventos

La limitación del escenario es que no está claro cuántos objetos están afectados y qué objeto genera un evento y el objeto que recibe un evento. Para superar esta limitación, un diagrama de traza de eventos se emplea. En el diagrama de traza de eventos, la secuencia de los eventos y los objetos que intercambian los eventos pueden ser vistos. El diagrama muestra cada objeto con una línea vertical y cada evento como una flecha horizontal del objeto emisor al objeto receptor. El tiempo va de arriba hacia abajo. La separación entre las flechas horizontales no lleva información.

En este diagrama, hay cuatro objetos - Usuario, ATM, Sucursal y Banco – que están involucrados, los cuales se muestran con cuatro líneas verticales. Usuario genera un evento "tarjeta insertada", que se muestra por la flecha horizontal de usuario ATM. Eso significa que la fuente del evento es el objeto usuario y el destino del evento es ATM. En respuesta a este evento, el cajero automático genera la "solicitud de contraseña" evento hacia el Usuario. El espacio entre estas dos flechas es insignificante, pero el evento

"tarjeta insertada" se produce antes del evento "solicitud de contraseña" y así sucesivamente.

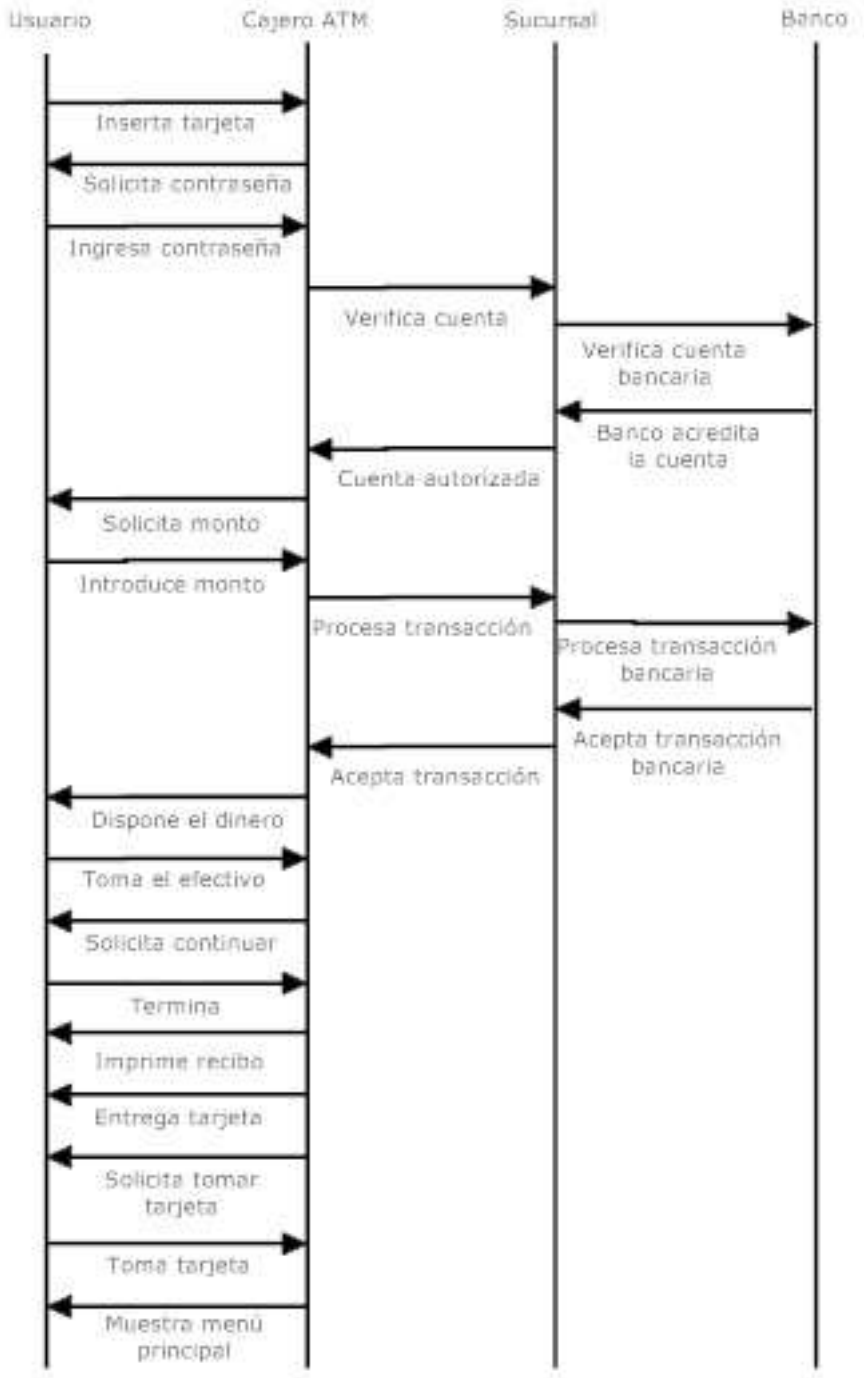


Diagrama de traza de eventos del cajero ATM

Máquina de estados

Una máquina de estados muestra un comportamiento que especifica una secuencia de estados de un objeto visitas durante su tiempo de vida en respuesta a eventos, junto con sus respuestas a esos eventos. Existen diferentes conceptos relacionados con la máquina de estados, estos son el estado, el evento, la transición, la acción y la actividad.

Estado

Un estado corresponde al intervalo entre dos eventos recibidos por un objeto y describe el "valor" del objeto en ese momento del tiempo. Un estado es una abstracción de los valores de los atributos de un objeto y enlaces, donde los valores se agrupan en un estado de acuerdo con las propiedades que afectan el comportamiento general del objeto. Por ejemplo, la pila está vacía o pila está llena son dos diferentes estados de la pila.

Un subestado es un estado que está anidado en otro estado. Un estado que tiene subestados se llama un estado compuesto. Un estado que no tiene subestados se llama un estado simple. Se pueden anidar subestados hasta cualquier nivel.

Evento

En la máquina de estados, un evento es la aparición de un estímulo que puede desencadenar una transición de estado. En otras palabras, un evento es algo que sucede en un punto en el tiempo. Un evento no tiene duración. Un estímulo individual de un objeto a otro es un evento. Por ejemplo: oprimir un botón del ratón; el avión sale de un aeropuerto, son ejemplos de eventos.

Transición

Una transición es una relación entre dos estados, el cual indica que un objeto en el primer estado permanece en éste y cuando un conjunto especifico de eventos y condiciones se satisfacen entra en el segundo estado y así sucesivamente. La transición puede ser autotransición, esta es una transición cuya fuente y estados de destino son iguales. Si la transición es hacia un estado compuesto, la máquina de estado anidada debe tener un estado inicial.

Actividad

La actividad es una operación que toma tiempo en completarse y se asocia con un estado. La actividad incluye operaciones continuas tales como mostrar una imagen en una pantalla de televisión, así como operaciones secuenciales que terminan por sí mismos después de un intervalo, como el cierre de una válvula o la realización de un cálculo.

Un estado puede controlar una actividad continua, tales como hacer sonar una campana que persiste hasta que termina un evento que provoca la transición del estado. Actividad comienza en la entrada al estado y se detiene al salir. Un estado puede también controlar una actividad secuencial, tal como, un robot que mueve una parte para avanzar, hasta que se completa o hasta que es interrumpido por un evento que lo hace terminar prematuramente.

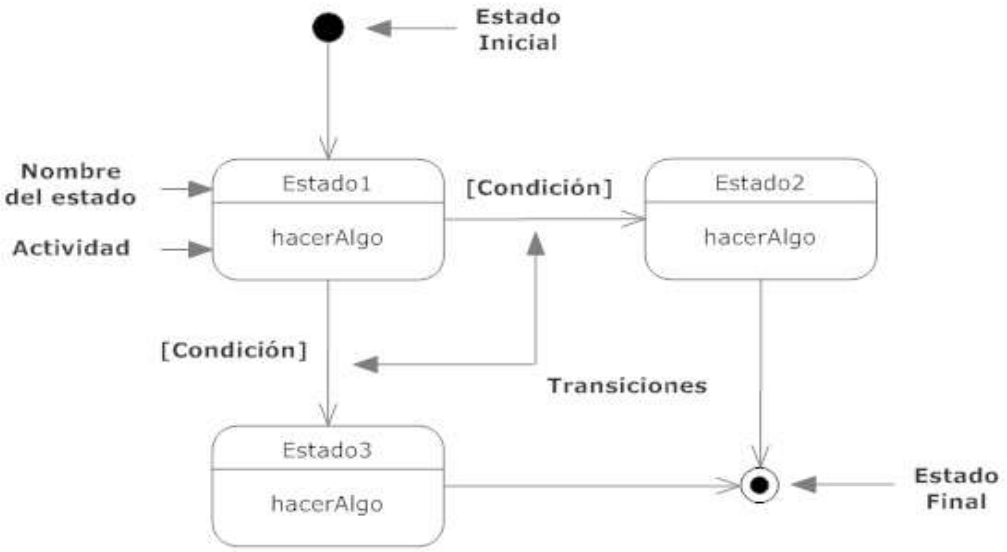
Diagrama de estados

El diagrama de estado se utiliza para describir el comportamiento de un sistema. Los diagramas de estado describen todos los estados posibles de un objeto como ocurren por los eventos. Cada diagrama suele representar los objetos de una sola clase y un seguimiento de los diferentes estados de los objetos a través del sistema.

Un cambio de estado es provocado por un evento y a este se le denomina transición. La transición se dibuja como una flecha desde el *estado* receptor al *estado* de destino. Un diagrama de estados es grafo cuyos nodos son estados y cuyos arcos dirigidos son

transiciones etiquetadas con los nombres del evento. El diagrama de estado especifica la secuencia de estado causado por una secuencia de eventos. Sólo utiliza los diagramas de estado para las clases donde es necesario entender el comportamiento del objeto a través de todo el sistema. No todas las clases requerirán un diagrama de estado.

Los elementos básicos del diagrama son cuadros con esquinas redondeadas que representan el estado del objeto y las flechas que indican la transición al siguiente estado. La sección de actividades dentro del símbolo del estado que muestra las actividades que el objeto va a realizar mientras se encuentre en ese estado.



Elementos del diagrama de estados

Aplicando este diagrama de estado al cajero ATM, cuando el cliente introduce su tarjeta se verifica que sea válida su cuenta, si no es válida se cancela el servicio y termina con la entrega de su cuenta, en caso de ser válida se le entrega el efectivo y su tarjeta y al final termina.

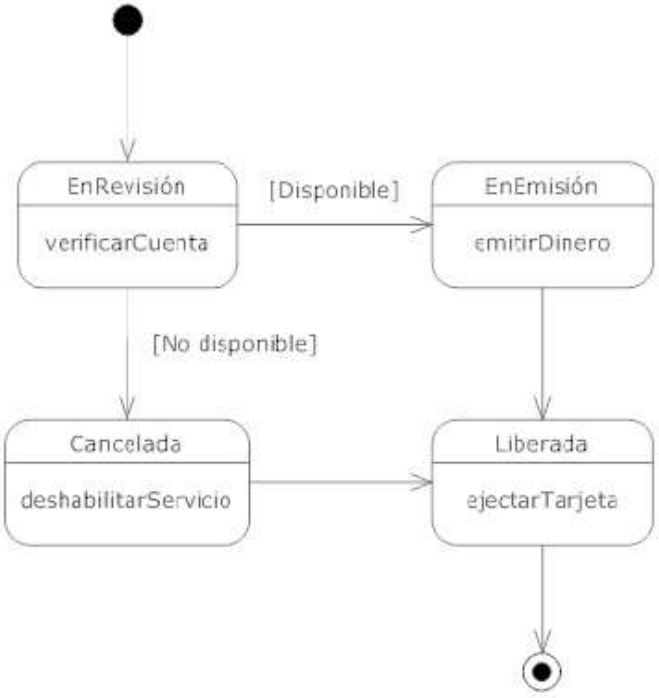


Diagrama de estados del cajero ATM

El diagrama de la máquina de estados modela el comportamiento de un solo objeto, especificando la secuencia de eventos que un objeto atraviesa durante su vida en respuesta a eventos.

Modelo funcional

El modelo funcional describe las operaciones y especifica los aspectos del sistema concernientes a las transformaciones de valores −funciones, asignaciones, restricciones y dependencias funcionales. El modelo funcional capta lo que hace el sistema, independientemente de cómo o cuándo se hace.

El modelo funcional se representa gráficamente con varios diagramas de flujo de datos (DFD), que muestran el flujo de los valores de las entradas externas, a través de operaciones y almacenes internos de datos, hacia las salidas externas. Los diagramas de flujo de datos muestran las dependencias entre los valores y el cálculo de los valores de salida de los valores de entrada y funciones. Las funciones se llaman como acciones en el modelo dinámico y se muestran como las operaciones de los objetos del modelo de

objetos. El diagrama de flujo de datos debe adherirse a la notación OMT y explotar las capacidades de la OMT, tales como anidaciones, flujos de control y restricciones.

Diagrama de flujo de datos (DFD)

Los diagramas de flujo de datos se utilizan durante el análisis del problema. No se limitan al análisis de problemas para la especificación de software de requisitos. DFD son muy útiles en la comprensión de un sistema y pueden ser utilizados eficazmente durante el análisis.

Un DFD muestra el flujo de datos a través de un sistema. Se considera un sistema como una función que transforma los insumos en productos deseados. Cualquier sistema complejo no puede transformar en un "solo paso" y, por lo tanto, los datos se someterán a una serie de transformaciones antes de que se convierta en la salida. El DFD tiene como objetivo captar las transformaciones que tienen lugar dentro de un sistema con datos de entrada que finalmente cambian por datos nuevos producidos de salida. El agente que realiza la transformación de los datos a partir de un estado a otro se denomina proceso. Así un DFD muestra el movimiento de datos a través de la transformación o proceso diferente en el sistema.

Los DFD son básicamente de dos tipos: las físicas y lógicas. Los DFD físicos se utilizan en la fase de análisis para estudiar el funcionamiento del sistema actual. Los DFD lógicos se utilizan en la fase de diseño para representar el flujo de datos en un sistema propuesto.

En pocas palabras, un diagrama de flujo de datos es un gráfico que muestra el flujo de valores desde los objetos con datos fuente a través de procesos que los transforman a sus destinos en otros objetos. Los diagramas de flujo de datos se componen de cuatro símbolos básicos: *entidades externas, procesos, almacén de datos* y *el flujo de datos.*

- El símbolo *entidad externa* representa las fuentes de datos al sistema o destinos de datos del sistema.

- El símbolo de *proceso* representa una actividad que transforma o manipula los datos.
- El símbolo del *almacén de datos* representa los datos que no se están moviendo (datos retardados en reposo).
- El símbolo de *flujo de datos* representa el movimiento de datos.

Cualquier sistema puede representarse en cualquier nivel de detalle de estos cuatro símbolos.

Entidad externa

El símbolo entidad externa representa las fuentes de datos al sistema o destinos de datos del sistema. Ellos determinan los límites del sistema. Ellos son externos al sistema que está siendo estudiado. Pueden representar otro sistema o subsistema. Se representan mediante un símbolo de rectángulo y se denominan con el nombre adecuado.

Figura que representa una entidad externa.

Algunos autores los llaman *actores*, ya que son los objetos activos que impulsan el diagrama de flujo de datos mediante la producción o consumo de los valores. Los actores están unidos a las entradas y salidas de un diagrama de flujo de datos.

Procesos

Los procesos son acciones o trabajos realizados en los flujos de datos de entrada para producir flujos de datos de salida. Estos datos muestran la transformación o cambio. Los datos entran en un proceso “transformado en” de algún modo. Por lo tanto, todos los procesos deben tener entradas y salidas. En algunos casos, las entradas o salidas de datos sólo se mostrarán a niveles más detallados de los diagramas. Cada proceso está

siempre "corriendo" y listo para aceptar datos. Las funciones principales de los procesos son cálculos y toma de decisiones. Cada proceso puede tener un tiempo radicalmente diferente: anual, semanal, diario, etc. Un proceso se representa por un círculo, como se muestra a continuación.

Figura que representa un proceso.

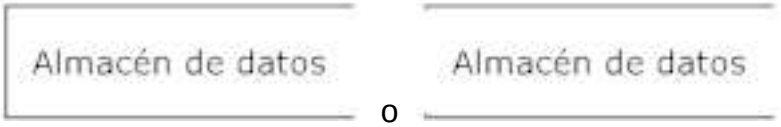
A cada proceso se le da un nombre. A los procesos se les denomina con un verbo cuidadosamente elegido y con el objeto del verbo. No hay sujetos o personas. El nombre no debe incluir la palabra "proceso". Cada proceso debe representar una función o acción. Si hay una "y" en el nombre, es probable que el proceso tenga más de una función. Por ejemplo: *actualizar los clientes y crear orden*. Los procesos están numerados en el diagrama como identificador del orden en que ocurren. Los niveles de detalle se muestran con notación decimal. Por ejemplo: el proceso de nivel superior sería el “proceso 4”; el siguiente nivel sería “4.1”, es decir, el proceso detalle, y así sucesivamente. Los procesos generalmente deben realizarse de arriba a abajo y de izquierda a derecha.

Almacenamiento de datos

Los almacenes de datos son repositorio de datos para que estén temporalmente o permanentemente registrados en el sistema. Son un vínculo común entre los datos y modelos de procesos. Sólo los procesos pueden conectar con almacenes de datos.

Puede haber dos o más sistemas que comparten un almacén de datos. Esto puede ocurrir en el caso de que un sistema realice una actualización al almacén de datos, mientras que

otro sistema sólo tiene acceso a la consulta de datos. Los almacenes de datos son representados por rectángulo abierto o dos líneas paralelas, como se muestra a continuación.



Figuras que representan el almacén de datos.

A los almacenes de datos se les da un nombre apropiado de acuerdo al contexto, no debe incluir la palabra "archivo", los nombres deben usarse en plural, pues describen la colección de datos. Como los clientes, los pedidos y loa productos. Estos pueden ser duplicados y pueden almacenar datos para su uso posterior. No generan ninguna operación en sí mismos, pero pueden responder a la solicitud. Es por eso que son objetos pasivos en un diagrama de flujo de datos.

Flujo de datos

El flujo de datos representa la entrada o salida de datos hacia o desde un proceso, almacén de datos o una entidad (actor). Los datos de flujo de datos, no son de control. Representan los datos mínimos esenciales que el proceso necesita. Utilizando sólo los datos mínimos esenciales reduce la dependencia entre procesos. Los flujos de datos deben comenzar y/o terminar en un proceso.

A los flujos de datos siempre se les da un nombre. Éste no debe incluir la palabra "datos" y debe ser un nombre único. Los nombres deben ser algo sustantivo como identificación. Por ejemplo: marcas, orden de pago, quejas, etc. Un flujo de datos se representa mediante una flecha, como se muestra a continuación.

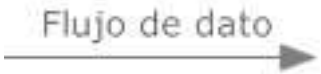


Figura que muestra la fecha del flujo de dato

Una flecha entre el origen y el destino de los datos representa el valor de un flujo de datos. La flecha debe etiquetarse con la descripción de los datos. La flecha de entrada indica el almacenamiento de datos hacia un almacén de datos y la flecha de salida indica el acceso de datos del almacén de datos.

Control de flujo

El control de flujo es un valor booleano en el DFD que determina si un proceso se evalúa o no. El flujo de control no es un valor de entrada para el proceso. Se representa mediante una línea punteada; a partir de un proceso de origen, el valor booleano evalúa el proceso para controlar su realización. En el siguiente diagrama DFD muestra un retiro de una cuenta bancaria. El cliente proporciona una contraseña y una cantidad. La actualización (el retiro), puede ocurrir únicamente cuando la contraseña es correcta, que se muestra con el control de flujo en el diagrama.

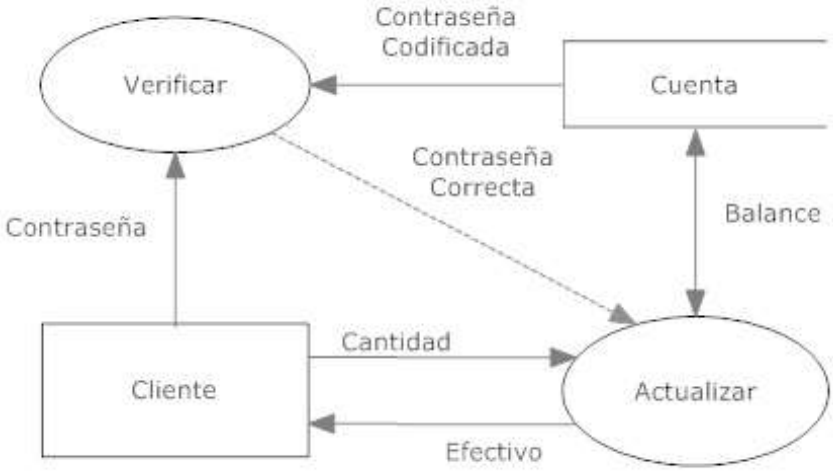


Diagrama de flujo de datos, plasmando el control del flujo.

A continuación se presentan el conjunto de DFD elaborado para un modelo general de un sistema de pedidos de publicaciones; éste es un diagrama de primer nivel.

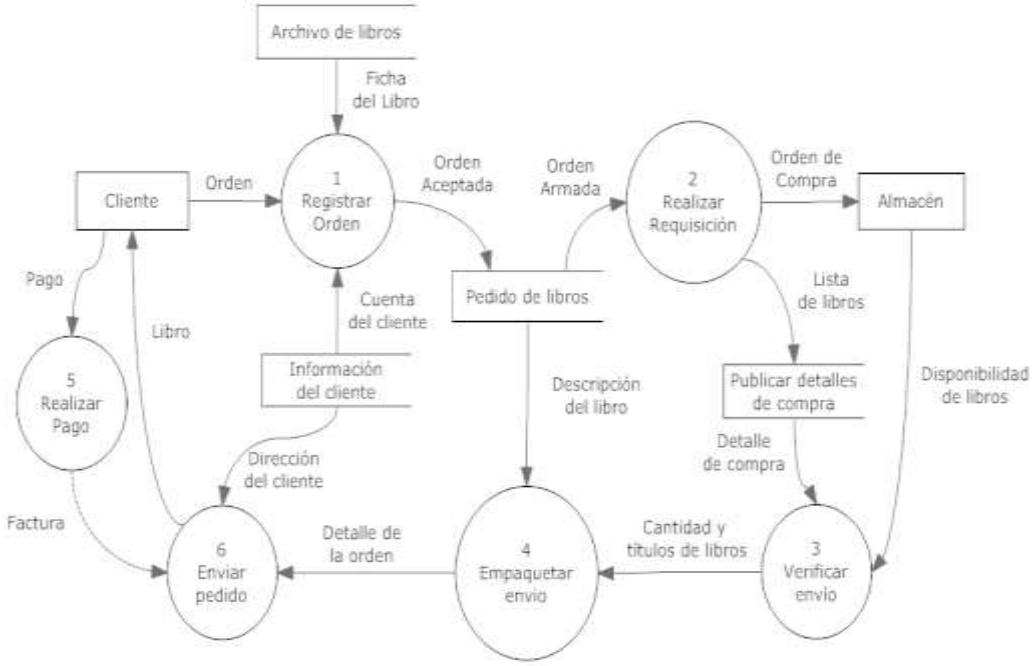


Diagrama de flujo de datos de primer nivel.

Cuando se necesita un mayor nivel de detalle se debe “romper” un proceso para describir con precisión que ocurre dentro de éste. Para esto se romperá el proceso de “Realizar el pago” y ver qué subpasos se realizan.

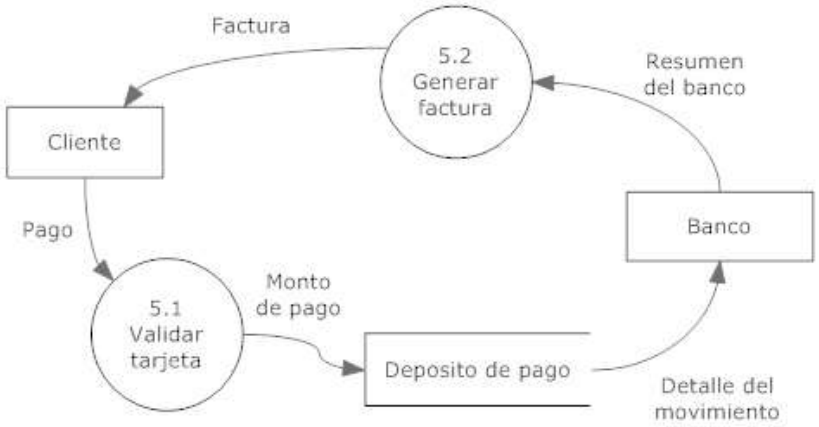


Diagrama de flujo de datos de segundo nivel

Los diagramas de flujo de datos pueden ser de varios niveles dependiendo del nivel de detalle que se requiere modelar. Los DFD, sin embargo, se consideran más su uso desde el punto de vista estructurado y, por lo tanto, ya no se ocupan para modelar sistemas desde el punto de vista de la orientación a objetos.

RESUMEN

La metodología de Grady Booch se enfoca en fases para la construcción de un proyecto orientado a objetos, éste se divide en 4 modelos: El *modelo lógico*, que sirve para describir la existencia de las abstracciones y elementos clave que forman el dominio del problema o que definen la arquitectura del sistema, este modelo incluye la realización del diagrama de clases y objetos. El *modelo físico* describe el software y hardware concreto de composición del contexto sistema, e incluye la realización del diagrama de módulo y el diagrama de procesos. Los diagramas de clases, objetos de módulo y procesos describen, todos, el modelo estático y para el modelo dinámico se utilizan el diagrama de transición de estados y de interacción.

Ivar Jacobson desarrolló Objectory, la cual emplea tres fases: La fase de *análisis de requerimientos*, que define el dominio del problema y el modelo de los casos de uso; la fase de *análisis de robustez*, que emplea el diseño de objetos de acuerdo a su responsabilidad dividiéndolos en objetos de interfaz, de entidad y de control; la fase de *diseño,* donde se realiza un diagrama de interacción que muestra la comunicación entre los objetos; la fase de implementación, en que se realiza el diagrama de estado y la codificación y, finalmente, la fase de *pruebas*, en donde se validan los casos de uso y se comprueba si cumplen con la funcionalidad requerida.

James Rumbaugh con la técnica de modelado de objetos (OMT), define tres modelos. En el *modelo de objeto* muestra la vista principal de los objetos que definen el mundo real en el que interactúan (sus relaciones con otros objetos, sus atributos y sus operaciones) y la descomposición total del sistema. En el *modelo dinámico* muestra los estados, transiciones, eventos y acciones, que en sí todo lo cambia durante el tiempo; y el *modelo funcional*, que representa las operaciones o “transacciones” de los procesos y los cambios que sufren los datos por medio de diagramas de flujo de datos.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Booch, Rumbaugh y Jacobson (2000)	3	21-34
Booch (1994)	5	167 - 226
Flowler, Martin y Scott Kendall (1999)	3	49 – 59
Rumbaugh, James (1996)	6	257 - 309
Jacobson, Ivar (2000)	3	27 - 53

Booch, G., Rumbaugh, J. y Jacobson, I (2000). *El lenguaje unificado de modelado UML 1.3.* Madrid: Pearson Addison-Wesley Iberoamericana.

Booch, G. (1994). *Análisis y diseño orientado a objetos con aplicaciones* (2ª Ed). California, EU: Addison-Wesley.

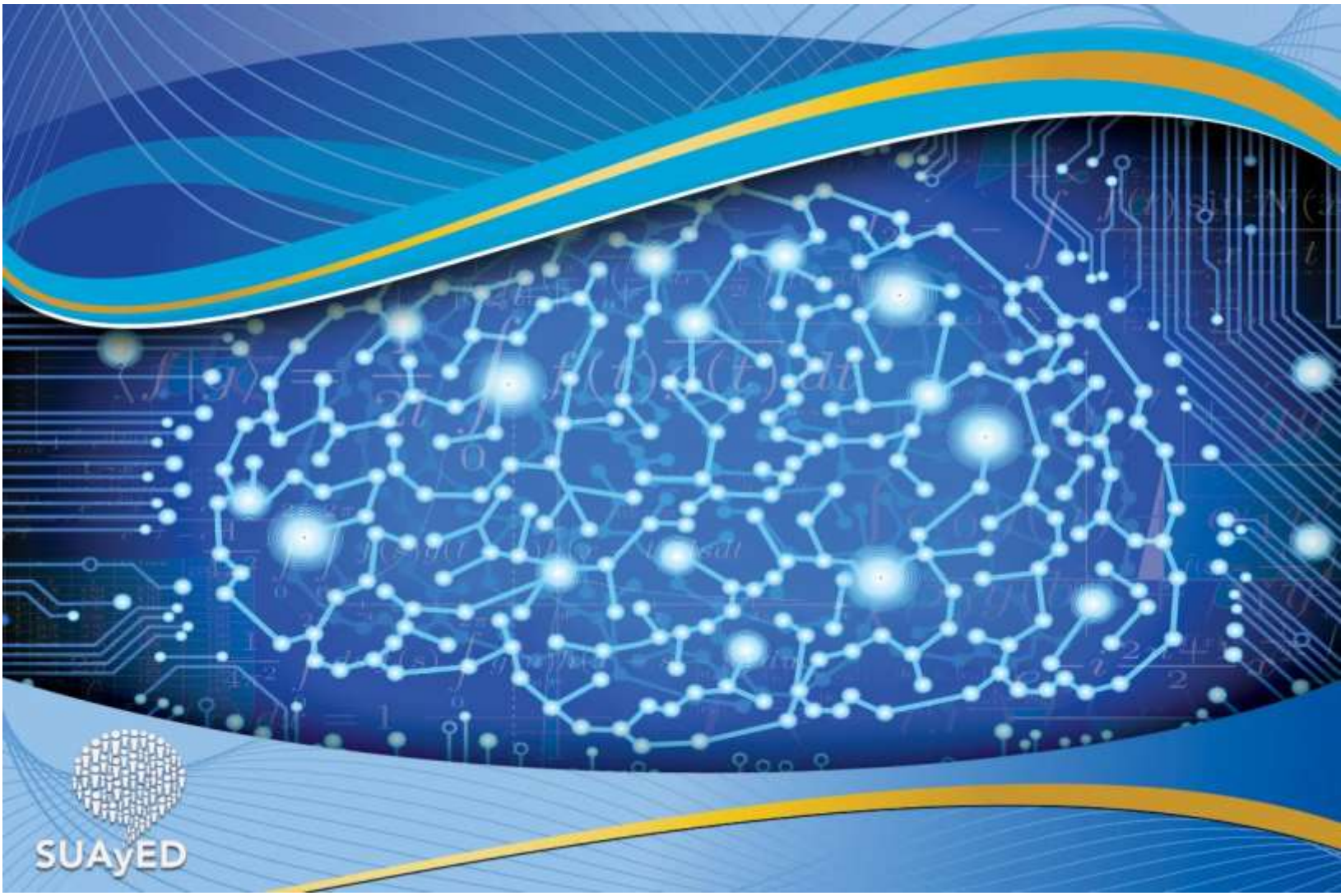
Flowler, M. y Scott, K. (1999). *UML gota a gota.* Massachusetts, EU: Addison-Wesley Longman.

Rumbaugh, J. (1996). *OMT Insights.* New York, E.U: SIGS Books.

Jacobson, I. (2000). *The Road to the Unified Software Development Process.* E.U. SIGS Books.

Unidad 3.

Planeación y elaboración



OBJETIVO PARTICULAR

Elaborar el plan de análisis y diseño de un sistema utilizando los casos de uso.

TEMARIO DETALLADO

(12 horas)

3. Planeación y elaboración

- 3.1. Delimitación del alcance del sistema
- 3.2. Clasificación de los requerimientos
- 3.3. Cualidades de los requerimientos
- 3.4. Administración de requerimientos
  - 3.4.1. La ciberjusticia
  - 3.4.2. La validez del documento electrónico
  - 3.4.3. El espacio electromagnético y las señales electrónicas en relación con la soberanía nacional
  - 3.4.4. La propiedad intelectual
- 3.5. Modelado de casos de uso
  - 3.5.1. Especificación de casos de uso

INTRODUCCIÓN

Una de las primeras tareas en el desarrollo de un sistema de información, es identificar las características y funciones que debe tener, para ello es necesario que los desarrolladores se entrevisten con los clientes y que interactúen con aquéllos que serán los usuarios del sistema para conocer sus características, formas de trabajo, debilidades, ambiente laboral, etc.

Todo lo anterior es parte de un proceso que nos ayudará a delimitar el alcance que tendrá el sistema, que a su vez nos ayudará a identificar los requerimientos necesarios para poder planificar las actividades de diseño y aplicación del sistema informático.

A lo largo de la presente unidad, estudiaremos algunos de los aspectos principales de la administración de los requerimientos de un sistema, comenzando por su clasificación, que nos ayudará a separar los requerimientos para una mejor identificación e implementación, la revisión de sus cualidades principales, que nos auxiliará para conocer cómo debemos conformar un buen requerimiento, hasta su administración, con lo que se documentará y dará seguimiento a los requerimientos a lo largo del desarrollo del sistema.

3.1 Delimitación del Alcance del Sistema

La delimitación del alcance de un sistema de información, es una tarea que utiliza como punto inicial a los procesos identificados a partir del levantamiento de requerimientos, durante esta tarea se identifican todos los procesos que pertenecen al sistema y las entidades externas que interactúan con él proporcionando datos de entrada o recibiéndolos.

Dentro del paradigma orientado a objetos, es necesario conocer antes que nada el contexto del negocio, es decir, la forma como se trabajan de forma cotidiana las operaciones que serán realizadas por el sistema; al realizar esta tarea, los desarrolladores tendrán la posibilidad de conocer más a detalle todos los procesos que involucra cada tarea, lo que facilitará la especificación de los requerimientos del sistema y, por consecuencia, su análisis y modelado.

Durante este proceso, es recomendable que se desarrolle un glosario de la terminología técnica empleada en la organización para la cual se desarrollará el sistema, dado que servirá para que los involucrados en el desarrollo del mismo comprendan con mayor facilidad los procesos que van a ser integrados al sistema de información.

3.2 Clasificación de los requerimientos

Los requerimientos de los sistemas de información pueden ser divididos en dos clasificaciones básicas:

Requerimientos funcionales

Los requerimientos funcionales ayudan a definir las funciones del sistema, es decir, son aquellos requerimientos que ayudan a identificar las entradas y salidas de información que conforman al sistema en sí y los procesos que trabajan con dicha información; en resumen, definen lo que el software debe o no hacer.

Existe una clasificación más amplia de los tipos de requerimientos funcionales, que pueden ser consultados en la unidad 1 subtema 4 de los apuntes de informática II de mi autoría.

Requerimientos no funcionales

Los requerimientos no funcionales, se refieren a la forma en que el sistema deberá de comportarse en su entorno, es decir, nos ayudan a revisar aspectos como son el rendimiento del sistema, la carga de trabajo, su disponibilidad, etc.

Sommerville (2005: 125) indica que los requerimientos no funcionales, como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino que están orientados más hacia las necesidades que surgen del usuario.

Dentro de las buenas prácticas de la ingeniería de software, se sugiere emplear el estándar ISO/IEC 9126, ahora englobado en el proyecto SQuaRE, para el desarrollo de

la norma ISO 25000**,** que define un modelo independiente de la tecnología para caracterizar la calidad de software y considera las características (Norma ISO/IEC 9126, 2014) siguientes:

- *Funcionalidad.*
- *Confiabilidad.*
- *Amabilidad.*[2]
- *Eficiencia*.
- *Capacidad de mantenimiento*.
- *Compatibilidad*.
- *Portabilidad*
- *Productividad*.
- *Seguridad.*
- *Satisfacción.*

La descripción detallada de las características antes mencionadas se aborda en detalle en la unidad 1 en el subtema 4 de los apuntes de Informática II.

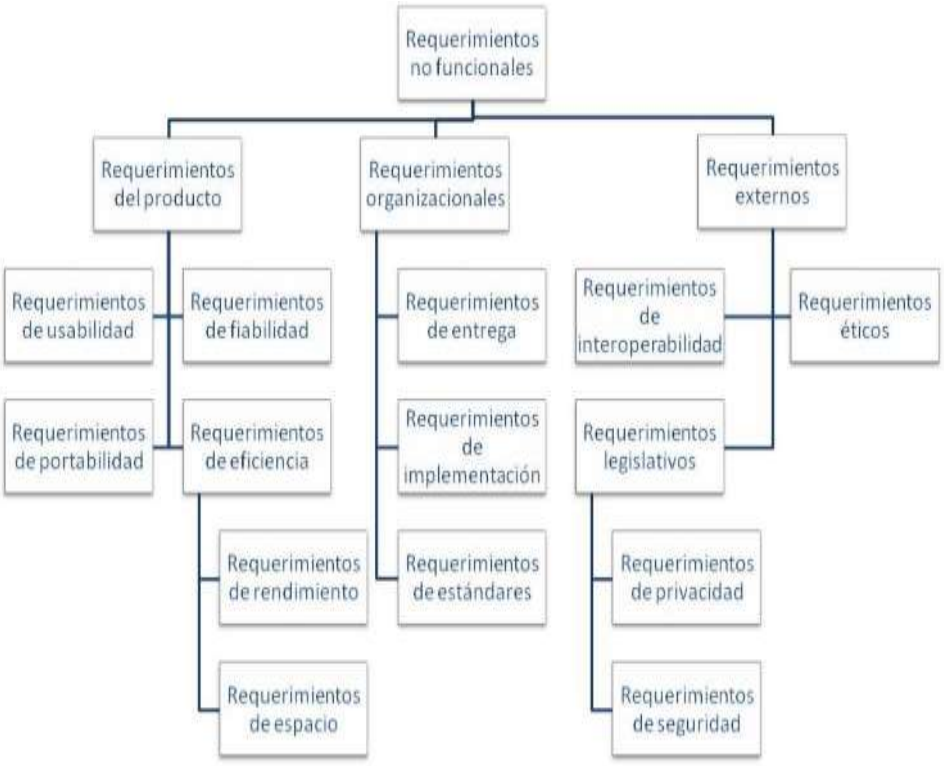
Los requerimientos no funcionales, pueden ser catalogados en 3 categorías principales:

- *Requerimiento del producto*. Son aquellos que hacen referencia al comportamiento del sistema, como, por ejemplo, la rapidez, la memoria empleada, espacio de disco duro, etc.
- *Requerimientos organizacionales*. Este tipo de requerimientos hace referencia principalmente a las normas, políticas y procesos de la organización a la que se le desarrolla el sistema, lo anterior con el objetivo de ser coherentes con su enfoque organizacional.

[2] Algunos autores le denominan “usabilidad”.

- *Requerimientos externos*. Hacen referencia a todos los factores del medio ambiente de trabajo que tendrá el sistema: interactuación con otros sistemas o con los usuarios, las legislaciones vigentes, etc.

La figura siguiente muestra de forma general los tipos de requerimientos no funcionales de acuerdo a Sommerville.



Tipos de requerimientos no funcionales

Fuente: Sommerville, 2005: 126.

3.3 Cualidades de los Requerimientos

La administración de requerimientos es fundamental para dar un buen seguimiento al sistema a desarrollar, así mismo, ayuda a clasificarlos y a verificarlos en instancias posteriores en el ciclo de vida del sistema.

La definición de atributos de los requerimientos es indispensable para poder identificar funciones o tareas específicas del sistema, en el caso del paradigma orientado a objetos, estos atributos nos ayudarán a definir objetos y clases que serán necesarios para construir al sistema.

Entre los atributos principales de los requerimientos podemos encontrar:

- ✓*Disponibilidad*.
- ✓*Integridad conceptual*.
- ✓*Flexibilidad*.
- ✓*Interoperabilidad*.
- ✓*Capacidad de mantenimiento.*
- ✓*Capacidad de administración*.
- ✓*Rendimiento*.
- ✓*Confiabilidad*.
- ✓*Escalabilidad*.
- ✓*Seguridad*.
- ✓*Riesgo*.
- ✓*Prioridad*.

La definición a detalle de los atributos anteriores puede ser consultada en la unidad 1, subtema 6.2 de los apuntes digitales de informática II.

3.4 Administración de Requerimientos

Un plan de administración de requerimientos debe contar con las siguientes características:

1. Identificar los requerimientos formales (no los documentos).

2. Identificar los tipos de documentos (cada tipo de documento maneja de forma predeterminada un tipo de requerimiento formal).

3. Por cada uno de los requerimientos formales identificar atributos:

- Prioridad.
- Riesgo.
- Dificultad
- Etc.

4. Realizar matriz de trazabilidad, para saber cómo rastrear los cambios.

Un plan de administración de requerimientos debe ayudar a responder a las siguientes preguntas:

- ¿Cómo deben usarse las herramientas de gestión de requerimientos?
- ¿Qué tipos de requerimientos serán trazados en el proyecto?
- ¿Cuáles son los atributos de estos requerimientos?
- ¿Dónde se crearán los requerimientos en una base de datos o solo en documentos?
- ¿En qué requerimientos necesitamos implementar trazabilidad?
- ¿Qué documentos se requieren?
- ¿Qué requerimientos y documentos serán utilizados como contrato con un proveedor?
- ¿Qué metodología será utilizada?

- ¿El cliente necesita algún documento específico para cumplir con su proceso de desarrollo?
- ¿Cómo se implementará la gestión de cambios?
- ¿Qué proceso garantizará que todos los requerimientos serán implementados y comprobados?
- ¿Qué requerimientos u opiniones necesitamos para generar informes? (Gestión de requerimientos VI, 2011)

Para mayor detalle en los puntos anteriores, favor de revisar la unidad 1, tema 6 de los apuntes de Informática II.

3.4.1 Identificación

Identificar los requerimientos es el primer paso que se da al analizar la información que se recaba al definir el tipo de sistema a ser desarrollado, lo anterior permite a los desarrolladores identificar necesidades tanto de la organización como del sistema, así como lo que se desea que el sistema realice.

Para identificar los requerimientos de un sistema, es posible emplear diversos métodos y técnicas que involucren a todos los que tendrán contacto con el sistema, sean desarrolladores o no.

Algunos de los métodos de recopilación de requerimientos empleados son, por ejemplo, las entrevistas, método Delphi, diagramas de Ishikawa, etc. Para mayor detalle sobre estos métodos y otros más, consulte la unidad 2 de los apuntes de Informática II

3.4.2 Jerarquización

Cuando se desarrolla un sistema de información, normalmente no es posible incorporar todos los requerimientos identificados, es por ello que, para poder lograr un sistema que

cubra la mayor parte de los requerimientos de los clientes, es necesario realizar una jerarquización de ellos.

Para poder jerarquizar los requerimientos de un sistema, es necesario identificar aquellos requerimientos que son esenciales para el sistema, lo cual implica una tarea de análisis a profundidad de todos los requerimientos determinados y establecer criterios sobre los cuales serán jerarquizados.

Martínez (s. f), menciona que una buena estrategia para realizar la priorización de requerimientos se basa en tres aspectos principales:

- “Selección de los criterios definidos para ordenar requerimientos. Pueden ser criterios de negocios como necesidades de los usuarios, costo; o bien técnicos como factibilidad, recursos existentes, etc.

- Determinación de un ordenamiento de acuerdo a criterios específicos para uno o más participantes

- Composición de un orden final combinando el punto anterior con varios participantes”.

3.4.3 Especificación

Cuando se habla de la especificación de requerimientos, dentro de la ingeniería de software se sugiere emplear el estándar IEEE 830 que se enfoca a la especificación de requerimientos de software.

El objetivo del estándar IEEE 830 es especificar los requerimientos del software a desarrollar y establecer un acuerdo documentado entre el cliente y los desarrolladores de sistemas, sobre el sistema a construir.

La plantilla del documento sugerido por el estándar IEEE830 también conocido como especificaciones de requerimiento de software o SRS, contiene las siguientes secciones:

1. Introducción.
2. Descripción general.
3. Requerimientos específicos.
4. Apéndices.

Para mayor detalle sobre cada una de las secciones que componen el documento IEEE830 SRS y la forma de redacción del mismo, consulta la unidad 3 subtema 1 de los apuntes de informática II.

En la siguiente hipervínculo encontrarás un ejemplo de un documento SRS desarrollado por Liliana Machuca de la Universidad del Valle, Santiago de Cali, Colombia.

3.4.4 Validación

La validación de los requerimientos trata de verificar que todo lo que se ha solicitado que realice el sistema y su medio ambiente de trabajo, ha sido entendido tanto por los desarrolladores como por los clientes.

La validación de requerimientos puede emplear varios métodos para verificar que el modelo de desarrollo a emplear sea el adecuado, entre estos se encuentra:

La *revisión de requerimientos,* que es uno de los métodos de validación más eficiente; por medio de la técnica de revisión es posible:

- Descubrir inconsistencias y defectos en los requisitos.

- Disminuir los costos en la fase de desarrollo del sistema alrededor de 30%
- Reducir los tiempos de la fase de pruebas hasta en 50%.

El proceso de revisión de requerimientos se ejecuta a través de reuniones entre los clientes y desarrolladores, donde se verifica que los requerimientos identificados sean los adecuados para la construcción del sistema.

Las *reuniones de revisión* siguen seis etapas principales:

1. Establecimiento del plan de revisión.
2. Distribución de documentos para revisión.
3. Preparación de la reunión.
4. Realización de la reunión.
5. Identificación de defectos.
6. Correcciones de documentos.

Las etapas antes mencionadas así como el resultado de las mismas se abordan a detalle en la unidad 4 subtema 2 de los apuntes de Informática II.

3.5 Modelado de Casos de Uso

Los diagramas de caso de uso son documentos que ayudan a los desarrolladores a entender la funcionalidad del sistema desde el punto de vista de los usuarios. Dentro de varias metodologías de desarrollo, incluyendo la orientada a objetos, los diagramas de caso de uso son la fuente principal de información en la identificación de requerimientos de un sistema de información.

La principal ventaja del empleo de diagramas de caso de uso, es su facilidad de interpretación, pues al detallar las funciones del sistema a manera de escenarios, es posible leer las interacciones y funciones con mayor claridad y detalle.

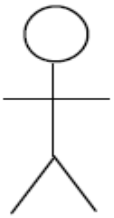
3.5.1 Especificación De Casos De Uso

Los elementos principales que integran un diagrama de caso de uso son los siguientes:

Actores

Los actores son la representación gráfica de los diversos tipos de usuarios del sistema, concibiendo que un usuario es toda aquella entidad externa que tiene interacción con el sistema, sean personas, departamentos de una empresa, otros sistemas de información, etc.

El símbolo para representar a los actores es el siguiente:

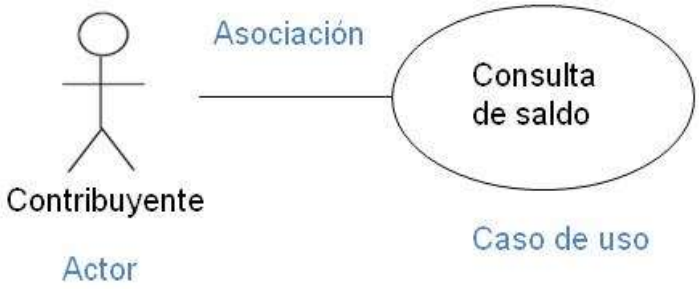


Caso de uso

El caso de uso representa de forma gráfica una tarea o función que el usuario puede realizar apoyándose en el sistema a desarrollar. Los casos de uso se representan habitualmente mediante un texto que describe la actividad, encerrado en un óvalo.

Asociaciones

Las asociaciones son líneas rectas que relacionan a cada caso de uso con los actores que realizan dicha tarea. Entonces, para cada usuario existirá al menos una asociación con una acción o caso de uso, todo dependiendo de la forma en cómo se interactúe con el sistema.



Escenarios

Los escenarios son la representación de una interacción entre un usuario y el sistema. Dentro de la fase de diseño del sistema pueden surgir múltiples escenarios que describan la forma en que diferentes usuarios interactúen con el sistema.

EJEMPLOS

Escenario 1	Escenario 2
Cliente consultando su saldo en un cajero automático.	Capturista ingresando datos personales para su registro en una base de datos.

Al especificar un caso de uso, se busca realizar una descripción de cada una de las partes definidas en él, para lograr establecer una descripción a detalle del mismo.

Para mayor información sobre los casos de uso consulta la unidad 3 subtema 4 de los apuntes de informática II.

RESUMEN

Una de las fases más importantes en el desarrollo de los sistemas de información es la de análisis, en ésta se planifica y se mide el alcance del sistema de información que se operará y de los requerimientos, que son identificados a partir de las entrevistas con el cliente y el uso de diversos métodos de determinación.

La buena identificación de requerimientos es fundamental para poder establecer una base de desarrollo de sistemas de información sólida y coherente, recordando que los requerimientos ayudan a definir las funciones que tendrá el sistema y la forma de realizar sus procesos.

Los requerimientos pueden ser clasificados en dos formas principales. Los funcionales que definen las características internas del sistema como son sus funciones, procesos, entradas y salidas, y los no funcionales, que describen el entorno de trabajo y las exigencias que tendrá el sistema.

Existen diversos métodos que nos permiten identificar requerimientos, como son las entrevistas, cuestionarios, etc., que auxilian a los desarrolladores a identificar procesos que no suelen ser expresados correctamente por parte de los clientes.

Debido a su importancia, los requerimientos deben ser documentados de forma adecuada, permitiendo darles seguimiento a lo largo de todo el proceso de construcción del sistema, registrar los cambios generados e identificar en qué partes del sistema fueron incorporados cada uno de ellos.

Para poder documentar correctamente los requerimientos es deseable emplear un formato estándar desarrollado por la IEEE, conocido como el documento de especificación de requerimientos y registrado con el estándar IEEE-830.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Bruegge (2001)	4	120-131
Pressman (2002)	1	17-25
Sommerville (2001)	1	24-35

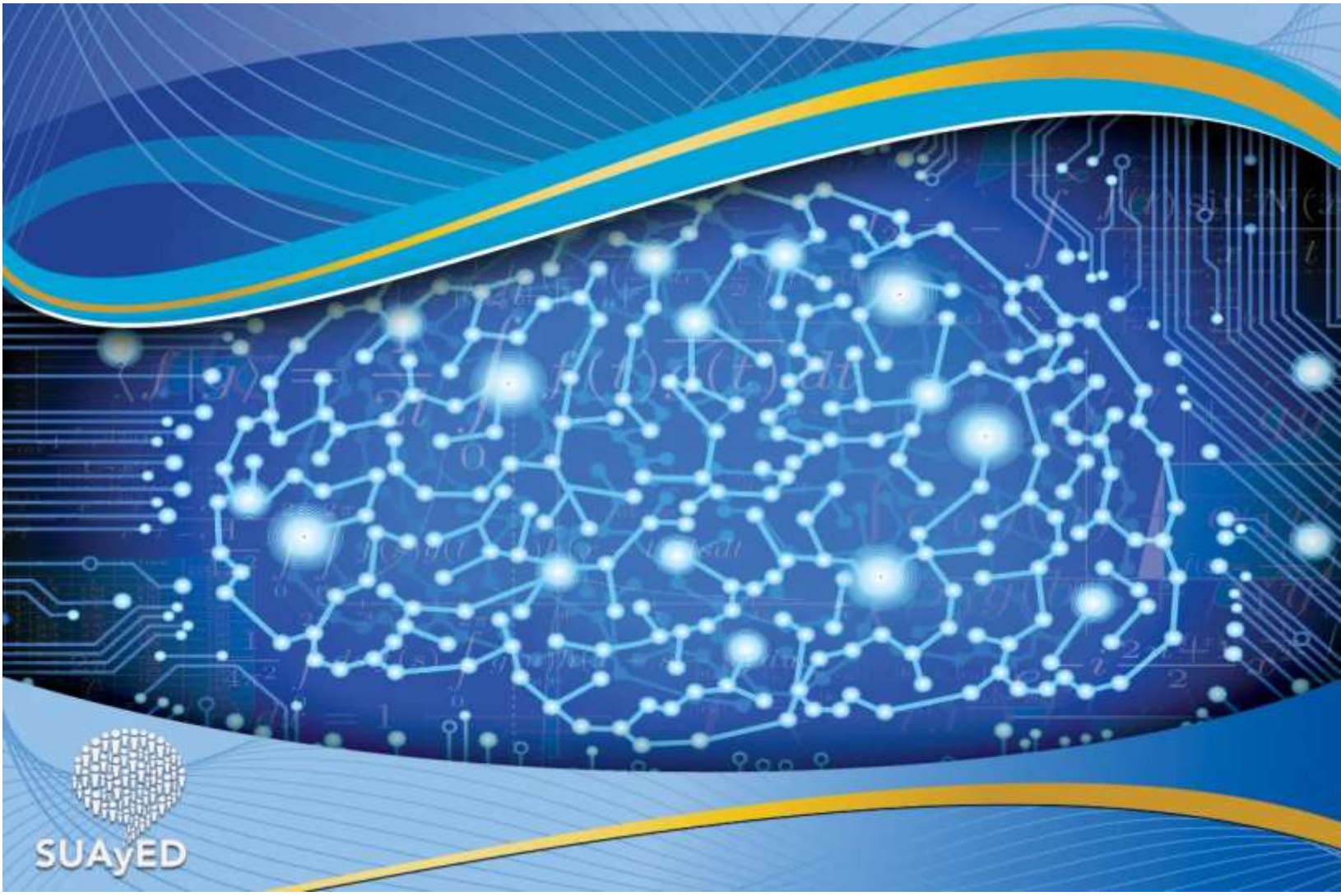
Bruegge, B. (2001). *Ingeniería de software orientada a objetos.* México: Prentice Hall.

Pressman, R. S. (2002). *Ingeniería de software* (5ª. ed.). México: Mc. Graw-Hill.

Sommerville, I. (2001). *Ingeniería de software (*6a. ed.). México: Addison Wesley.

Unidad 4

Análisis orientado a objetos



OBJETIVO ESPECÍFICO

Definir la arquitectura candidata del sistema propuesto.

TEMARIO DETALLADO
(16 horas)

4. Análisis orientado a objetos

4.1. Definir la arquitectura candidata
4.1.1. Creación de la realización de casos de uso con diagramas de colaboración
4.1.2. Definición de los subsistemas con diagramas de clases
4.1.3. Identificación de los mecanismos de análisis con diagramas de clases
4.2. Analizar el comportamiento del sistema
4.2.1. Identificación del comportamiento de clases y subsistemas con diagramas de clases
4.2.2. Identificación de interfaces de subsistemas con diagramas de clases

INTRODUCCIÓN

El objetivo de la fase de análisis en cualquier paradigma de programación tiene como objetivo establecer las características principales del software a desarrollar, identificar sus funciones, entradas, salidas, etc., con el fin de poder modelar el sistema para proceder con su desarrollo.

En el caso del paradigma orientado a objetos, el análisis pretende abstraer la información recabada en los requerimientos para obtener los objetos, clases y sus interacciones, y a partir de ello, a través de una serie de herramientas como los diagramas de casos de uso y de clases, será posible determinar las funciones y relaciones principales de cada objeto y del sistema en general.

En esta unidad revisaremos a detalle la forma de cómo se emplean los diagramas de caso de uso, los cuales auxilian a determinar las características principales del sistema y, a partir de ello, poder definir los objetos, clases y responsabilidades de los mismos.

Posteriormente, mediante el empleo de los diagramas de clases determinaremos los subsistemas o funciones del sistema a desarrollar, así como de las interfaces de las cuales se compondrá.

4. Análisis Orientado a Objetos

Todo proceso de desarrollo de software, como el paradigma estructurado u orientado a objetos, debe de partir de un análisis, como lo marca el ciclo de vida de los sistemas la fase de análisis, que permite a los ingenieros de software identificar los requerimientos del sistema, y, a partir de ellos, definir el comportamiento general del sistema, sus funciones, sus entradas, salidas, etc.

En el caso del análisis orientado a objetos, de acuerdo con Pressman (2002: 362), se aplican cinco principios básicos:

1. Modelo del dominio de la información.
2. Descripción de la función.
3. Representación del comportamiento del modelo.
4. División del comportamiento de los modelos de datos, funcional y de comportamiento.
5. Representación del problema a través de los modelos iniciales y de la implementación a través de los modelos finales".

En otras palabras, es necesario entender bien la problemática para poder describir las funciones que deberá tener el software a construir. Además, debemos describir su comportamiento y la forma en que deberá procesar los datos para establecer los modelos finales del sistema, que serán los que se implementarán.

En el análisis orientado a objetos se debe comenzar por identificar los objetos, clases y atributos que compondrán al sistema a desarrollar, esto se lleva a cabo a través del uso de varias herramientas, tales como los casos de uso y los diagramas de clases, donde

será posible establecer el comportamiento del sistema y su interacción. En seguida, veremos la forma como se emplean dichas herramientas.

4.1. Definir la Arquitectura Candidata

Existen varias metodologías de desarrollo orientadas a objetos (Pressman 2002: 362 y 363), entre las principales encontramos:

El método de Booch. Este método se basa en un enfoque de desarrollo evolutivo, que define dos niveles principales de desarrollo: el *micro*, que delimita un conjunto de tareas de análisis cuyo objetivo es identificar las clases, los objetos y las semánticas aplicadas a ellos, precisando la forma en que se relacionan y el modelo de análisis; el *macro*, que emplea el modelo de análisis resultante del micro nivel para comenzar el desarrollo del software, una vez terminado el primer diseño se vuelve al nivel micro para comenzar a analizar los resultados de la primera evolución y así se continúa hasta terminar el desarrollo del software.

El método de Rumbaugh. Está metodología se basa en la creación de tres modelos derivados de las actividades de análisis, que son:

- El modelo de objetos, donde se definen los objetos, clases, relaciones y jerarquías.
- El modelo dinámico, donde se representa el comportamiento del sistema y de los objetos que lo integran.
- El modelo funcional, que reproduce el flujo de los datos a través del sistema.

El método de Jacobson. Este método centra el proceso de análisis en el empleo de casos de uso que describen la forma en que el usuario interactúa con el sistema para poder determinar los objetos, clases, relaciones y jerarquías.

El método Coad y Yourdon. Esta metodología sigue los siguientes pasos para realizar su proceso de análisis:

- Identificar objetos, usando el criterio de «qué buscar».
- Definir una estructura de generalización-especificación.
- Definir una estructura de todo-parte.
- Identificar temas (representaciones de componentes de subsistemas).
- Definir atributos.
- Definir servicios.

El método de Wirfs-Brock. A diferencia de los otros modelos, este método propone el seguimiento de un proceso continuo que une el análisis y el diseño del sistema comenzando con la valoración de los requerimientos del cliente. Las fases del análisis del método Wirfs-Brock se listan a continuación:

- Evaluar la especificación del cliente.
- Usar un análisis gramatical para extraer clases candidatas de la especificación.
- Agrupar las clases en un intento de determinar superclases.
- Definir responsabilidades para cada clase.
- Asignar responsabilidades a cada clase.
- Identificar relaciones entre clases.
- Definir colaboraciones entre clases basándose en sus responsabilidades.
- Construir representaciones jerárquicas de clases para mostrar relaciones de herencia.
- Construir un grafo de colaboraciones para el sistema.

Aunque existen métodos diferentes para realizar el análisis y construir un modelo de software, Pressman (2002) señala que hay pasos genéricos que todo desarrollador debe seguir al momento de realizar un análisis orientado a objetos, estos pasos son:

- Obtener los requisitos del cliente para el sistema.
- Identificar escenarios o casos de uso.
- Seleccionar clases y objetos usando los requisitos básicos como guías.
- Identificar atributos y operaciones para cada objeto del sistema.
- Definir estructuras y jerarquías que organicen las clases.
- Construir un modelo objeto-relación.
- Construir un modelo objeto-comportamiento.
- Revisar el modelo de análisis orientado a objetos con relación a los casos de uso.

4.1.1 Creación de la Realización de Casos de Uso con Diagramas de Colaboración

El análisis orientado a objetos debe comenzar con un enfoque de análisis basado en el punto de vista de los usuarios, es decir, por un enfoque que permita al analista comprender el funcionamiento del sistema desde el punto de vista de cómo será empleado.

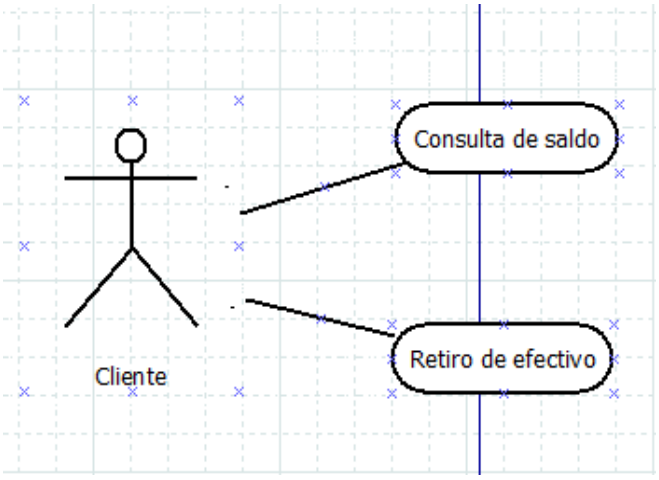
Una de las herramientas más útiles para poder lograr lo antes mencionado, son los diagramas de casos de uso, estos diagramas permiten al analista modelar el sistema desde la perspectiva del usuario, estos diagramas se deben crear desde la fase de obtención de requerimientos, tal como se vio en la unidad anterior.

Al crear casos de uso, se deben cumplir (Pressman, 2002: 362 y 367) los siguientes objetivos:

- Identificar y definir los requerimientos funcionales y operativos del sistema, desarrollando un escenario que describa el uso del sistema desde el punto de vista del usuario final, proporcionando una descripción clara y libre de ambigüedades de la forma en que el usuario final empleará el sistema y viceversa.
- Proporcionar un punto base para realizar la validación de las pruebas del sistema.

Los casos de uso son el primer elemento empleado en el análisis del sistema, empleando UML es posible crear un diagrama de casos de uso, que es una representación visual de los casos que se vayan identificando durante el proceso de análisis. Los casos de uso pueden ser desarrollados con diversos niveles de abstracción, como vimos en la unidad anterior, los casos contienen actores que representan a los usuarios u otros sistemas que interactúan con el sistema.

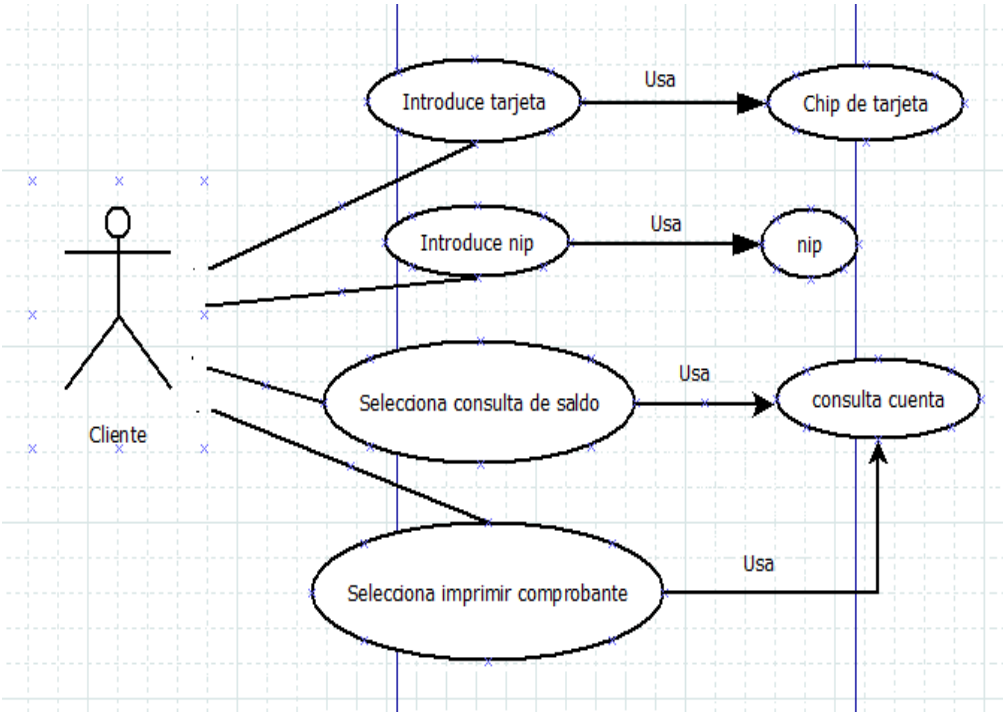
Para poder ilustrar esto, tomemos como ejemplo un cajero automático, el actor es el usuario del cajero, los óvalos representan los casos de uso, es decir, las operaciones que puede realizar el usuario en el cajero.



Escenario de casos de uso de un cajero automático

Fuente: Pressman, 2002: 368.

Este caso de uso inicial, puede dividirse en otros casos más detallados, permitiendo a los desarrolladores identificar las diversas acciones que se pueden realizar en cada operación.



Escenario de casos de uso del cajero con mayor nivel de detalle

Fuente: Pressman, 2002, p. 368.

Una vez que son desarrollados los diagramas de caso de uso, debemos proceder a la identificación de las clases, sus responsabilidades y colaboraciones, para ello es menester emplear otra herramienta que nos ayude a realizar esta tarea: el modelo de clases-responsabilidades-colaboraciones (CRC).

De acuerdo con Roger Pressman (2002: 368), el CRC se describe de la siguiente manera:

> Un modelo CRC es realmente una colección de tarjetas índice estándar que representan clases. Las tarjetas están divididas en tres secciones. A lo largo de la cabecera de la tarjeta usted escribe el nombre de la clase. En el cuerpo se listan las responsabilidades de la clase a la izquierda y a la derecha los colaboradores.

El objetivo de emplear el modelo CRC es hacer una representación organizada de las clases, dentro del modelo CRC las responsabilidades son los atributos y operaciones que pertenecen a la clase, o, en otras palabras, es todo aquello que puede hacer o hacerse con la clase. Finalmente, las colaboraciones son todas aquellas clases que proveen a la clase de información para que estas puedan llevar a cabo sus responsabilidades, o en palabras más simples, una colaboración no es otra cosa que una solicitud de información a otra clase o la ejecución de alguna de sus operaciones.

Algunas de las consideraciones que debemos de seguir para identificar clases (Pressman, 2002: 368-369) son las siguientes:

Los potenciales objetos se pueden manifestar de diversas formas, pueden ser entidades externas, sucesos, lugares, estructuras, etc. Para poder identificar estos objetos es necesario realizar un análisis gramatical de la problemática que deseamos resolver, durante la narrativa de la problemática los nombres se transforman en objetos potenciales, estos objetos deben tener las siguientes características:

1. *Retener información:* Los objetos deben poder almacenar la información concerniente a ellos mismos para que el objeto pruebe ser útil en el análisis del sistema.
2. *Servicios necesarios:* Los objetos deben tener operaciones asociadas que permitan la modificación de sus atributos.
3. *Múltiples atributos:* Los objetos deben poder contar con múltiples operaciones que permitan su flexibilidad de manejo durante el análisis, los objetos con un solo atributo también son útiles, pero por lo general este atributo corresponderá al de otro objeto cuando se realice un análisis más profundo.
4. *Atributos comunes:* Cuando se define una clase, los atributos identificados a los objetos asociados a ella deben de poder ser aplicables a la clase y a todas las veces que se repita el objeto.

5. *Operaciones comunes:* Al igual que en el punto anterior, todas las operaciones identificadas para el objeto que defina a la clase deben de poder ser aplicadas a la clase y a todas las ocurrencias del objeto.
6. *Requisitos esenciales:* las entidades externas que proporcionan información esencial para la operación del sistema deben ser definidas como objetos en el modelado de requerimientos.

Al definir una clase, es necesario que esta cumpla con las seis características antes mencionadas.

Pasando ahora a lo que son las responsabilidades, todas las clases tienen atributos estables, representados comúnmente a través de información que debe de almacenarse para poder ejecutar las funciones del sistema, por ende, los atributos pueden ser identificados a partir de un análisis del alcance y del contexto del sistema y de la comprensión de la naturaleza de la clase que se está definiendo. Las operaciones, que también forman parte de las responsabilidades de una clase, asimismo son identificadas a partir de un análisis gramatical del problema, en donde los verbos son los candidatos principales a convertirse en operaciones.

Finalmente, las colaboraciones, de acuerdo con Wirfs-Brock, pueden definirse de la siguiente manera:

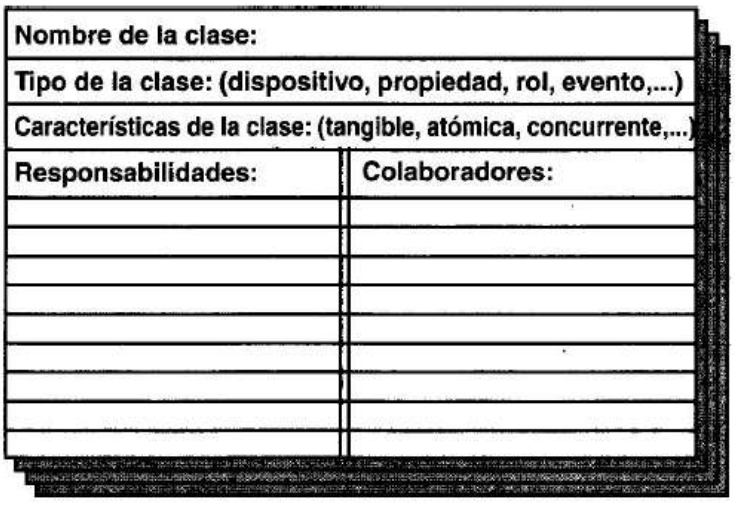
> “Las colaboraciones representan solicitudes de un cliente a un servidor en el cumplimiento de una responsabilidad del cliente. Una colaboración es la realización de un contrato entre el cliente y el servidor. Decimos que un objeto colabora con otro, si para ejecutar una responsabilidad necesita enviar cualquier mensaje al otro objeto. Una colaboración simple fluye en una dirección, representando una solicitud del cliente al servidor. Desde el punto de vista del cliente, cada una de sus colaboraciones está asociada con una responsabilidad particular implementada por el servidor” (Pressman, 2002: 370).

De manera general podemos decir que las colaboraciones son relaciones entre las diferentes clases del sistema, donde es posible que varias clases trabajen en conjunto para poder realizar alguna función del sistema o satisfacer algún requerimiento.

Las colaboraciones se identifican analizando las responsabilidades de cada clase y determinando si éstas pueden satisfacer dichas responsabilidades o requieren de otras clases para urdirlo en forma colaborativa.

Retomando nuestro ejemplo del cajero automático, para que el objeto cajero pueda validar el NIP del usuario es necesario que interactué con el objeto registro de clientes, para ello el objeto cajero establece la responsabilidad validar- NIP, si el NIP es correcto el atributo del objeto cajero debe cambiar a verdadero, de lo contrario el atributo será falso y no podrá continuar con sus otras responsabilidades, en este caso el objeto cajero establece una colaboración con el objeto registro de clientes.

La figura siguiente es un ejemplo de una tarjeta índice CRC que puede emplearse para documentar una clase.



Nombre de la clase:	
Tipo de la clase: (dispositivo, propiedad, rol, evento,...)	
Características de la clase: (tangible, atómica, concurrente,...)	
Responsabilidades:	**Colaboradores:**

Ejemplo de tarjeta CRS

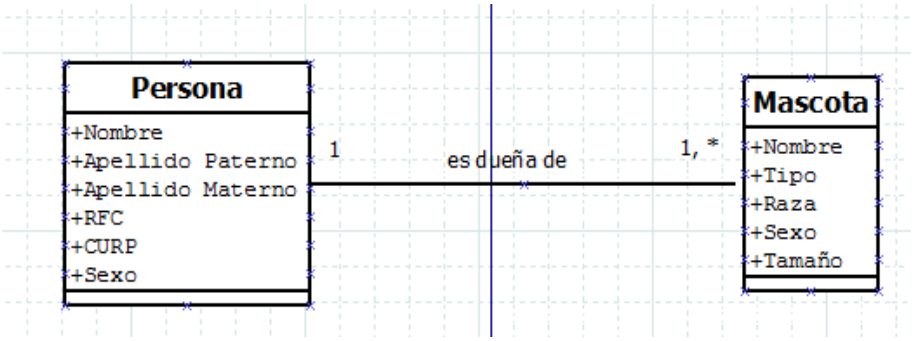
Fuente: Pressman, 2002: 370.

4.1.2 Definición de los subsistemas con diagramas de clases

Los subsistemas, en el análisis orientado a objetos, se definen como un subconjunto de clases colaborativas entre sí para realizar un conjunto de responsabilidades asociadas. Los subsistemas, de manera general, pueden verse como un objeto que tiene sus propias responsabilidades y colaboradores externos, donde cada subsistema establece una o varias solicitudes que pueden ser realizadas con cada uno de sus colaboradores externos, a estas solicitudes también se les denomina *contratos*.

Una herramienta muy útil para representar a los subsistemas es mediante un diagrama de clases, este diagrama se compone de clases y relaciones entre ellas, donde los tipos de relaciones pueden ser:

De asociación. Se trata de una conexión entre dos o más clases, generalmente este tipo de relaciones son de tipo semántico, como por ejemplo:



Ejemplo de relación de asociación

Fuente: Pressman, 2002: 383.

El ejemplo puede leerse así: una persona es dueña de una (1) o más (*) mascotas, que es lo que se denota en la línea de relación entre clases.

De dependencia. Es una relación entre clases donde una es dependiente de la otra, o sea, un cambio en la clase independiente conlleva a un cambio en la dependiente.

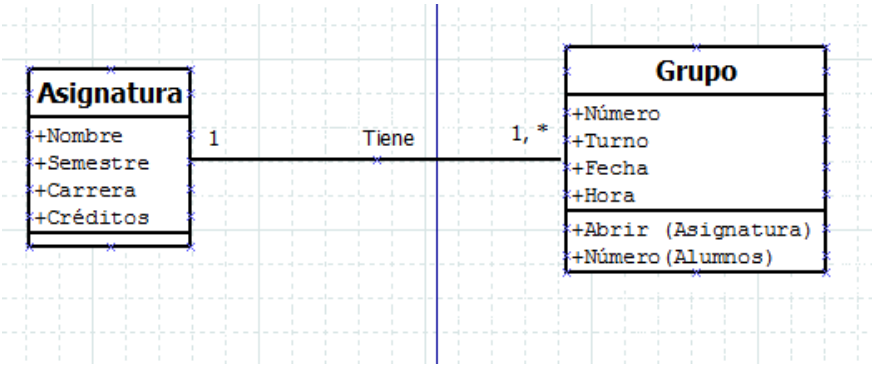
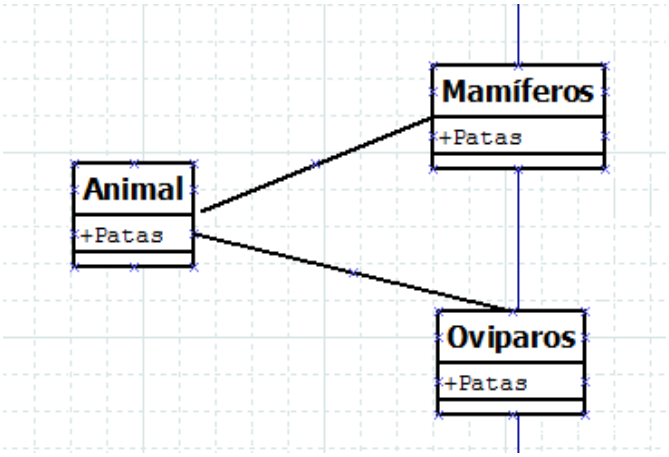


Diagrama de clases con relación de dependencia

Fuente: Pressman, 2002: 383.

En el ejemplo anterior, la clase grupo depende de la existencia de la asignatura y de la clase alumnos (que no está representada), la dependencia puede observarse en las operaciones de la clase grupo, ya que para que pueda realizarlas depende de las otras clases.

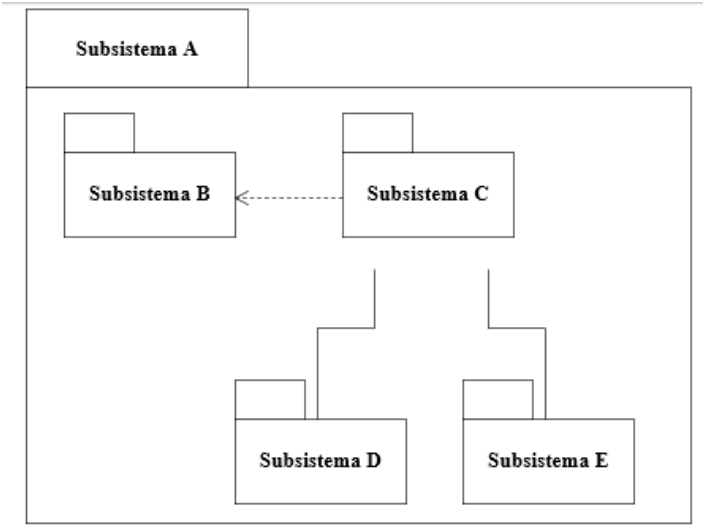
De generalización. Este es un tipo de relación entre clases asociada con la herencia, donde la clase más general (superclase) se relaciona con otra más específica (subclase) a través de los atributos que le son heredados de la superclase a la subclase.



Fuente: Pressman, 2002: 383.

En el ejemplo anterior, la superclase *Animal* se relaciona con las subclases *Mamífero* y *Ovíparo* y hereda el atributo *Patas*.

Los subsistemas o paquetes son representados en los diagramas de clases por un rectángulo grande acompañado de otro más pequeño en la esquina superior izquierda; dentro del rectángulo mayor se muestra su contenido (clases y responsabilidades), de no ser así, sólo es necesario colocar el nombre del subsistema o paquete tal como se muestra abajo:



Ejemplo de representación de subsistemas en un diagrama de clases

Fuente: Otero, s/a.

4.1.3. Identificación de los mecanismos de análisis con diagramas de clases

Una de las herramientas útiles para definir los mecanismos de análisis a través de las clases en el análisis orientado a objetos, es el modelo de clases de análisis, que es una representación abstracta de una o varias clases y subsistemas que componen al sistema en desarrollo.

Los casos de uso se realizan empleando clases de análisis (UPC, 2008), presentan las siguientes características:

- “Está centrado en los requisitos funcionales;
- Su comportamiento está definido mediante responsabilidades (normalmente no definen una interfaz en términos de operaciones);
- Define atributos que son conceptuales y reconocibles en el dominio del problema;
- Participa en relaciones;
- Encaja en uno de estos tres estereotipos básicos: de interfaz, de control, de entidad”.

Al crear el modelo de clases de análisis (UPC, 2008), se deben seguir los pasos siguientes

- “Identificar los casos de uso definidos y descritos para esa iteración.
- Sugerir qué clasificadores y asociaciones se necesitan para hacer realidad el caso de uso.
- Cada clasificador juega uno o varios roles en una realización de caso de uso.
- Cada rol de clasificador especifica las responsabilidades y atributos que debe tener para participar en hacer realidad un caso de uso”.

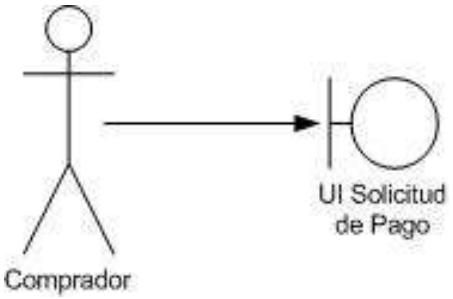
Para cada clase de análisis (UPC, 2008), se consideran estos rasgos:

- Responsabilidades
- Atributos
- Relaciones
- Requisitos especiales

En las clases de análisis (UPC, 2008) se identifican tres tipos básicos:

Clase de interfaz: Empleada para representar las interacciones entre el sistema y los usuarios, por ejemplo la captura de datos, la solicitud de información, la ejecución de alguna función, etc.

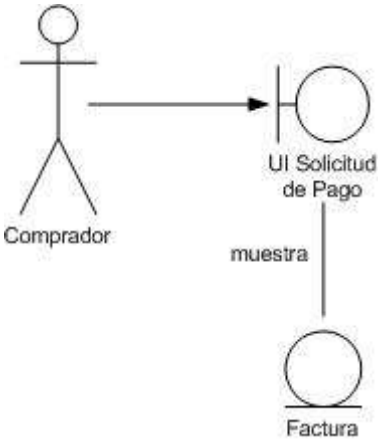
Normalmente este tipo de clases representa las interfaces del sistema a través de las cuales los usuarios interactúan con él, como por ejemplo ventanas, formularios, reportes, etc.



La Interfaz *UI solicitud de pago* se usa para cubrir la interacción entre el actor *comprador* y el caso de uso *pagar factura*

Fuente: UPC, 2008.

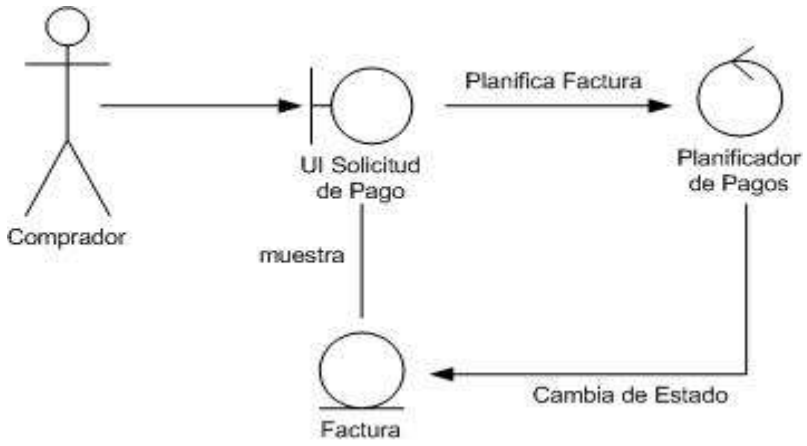
Clase entidad: Se usa para representar la información proveniente de personas, objetos o sucesos, este tipo de información generalmente tiende a ser duradera y sin cambios; por ejemplo, la información de un reporte de ventas resultado de una consulta al módulo de ventas.



La *clase de entidad factura* y su relación con la Interfaz *UI solicitud de pago*

Fuente: UPC, 2008.

Clase de control. Empleada para representar transacciones, operaciones complejas, secuencias y control de otros objetos.



La *clase* de *control planificador de pagos* y sus relaciones con las *clases de interfaz y de entidad.* Fuente: UPC, 2008.

4.2 Analizar el Comportamiento del Sistema

Los modelos anteriores (casos de uso, diagramas de clases y CRC) muestran el aspecto estático del sistema a modelar, por lo que la siguiente fase de diseño consiste en analizar su parte dinámica, es decir, la forma como el sistema ha de comportarse, para ello Pressman (2002: 374) señala los siguientes puntos a considerar:

- "Evaluar todos los casos de uso para comprender totalmente la secuencia de interacción dentro del sistema.
- Identificar sucesos que dirigen la secuencia de interacción y comprender cómo estos sucesos se relacionan con objetos específicos.

- Crear una traza de sucesos para cada caso de uso.
- Construir un diagrama de transición de estados para el sistema.
- Revisar el modelo objeto-comportamiento para verificar exactitud y consistencia".

En resumen, para poder evaluar el comportamiento del sistema es necesario volver a revisar los casos de uso diseñados a partir de los requerimientos iniciales, con el objetivo de identificar en esta ocasión las acciones realizadas para poder entender la forma como el sistema deberá responder a cada interacción del usuario. A partir de ello, comenzaremos por definir un diagrama de transiciones que nos ayudará a modelar las respuestas de los objetos en el momento de realizar las interacciones; este diagrama también es conocido como diagrama de secuencias. Finalmente, se procede a revisar nuevamente este diagrama para verificar que las interacciones han sido modeladas de forma correcta.

En el siguiente punto comenzaremos con el proceso de modelado de comportamiento de un sistema con el enfoque orientado a objetos.

4.2.1 Identificación del comportamiento de clases y subsistemas con diagramas de clases

Comencemos por recordar que el objetivo de los diagramas de caso de uso es representar la interacción de los actores (usuarios, otros sistemas, etc.) con el sistema a desarrollar (caso de uso), estos diagramas también nos permiten identificar el comportamiento del sistema; en general, cuando un actor realiza algún intercambio de información con el sistema dentro del diagrama de caso de uso se produce un suceso, el suceso en sí no es la información, sino el hecho de que se realice o no dicho intercambio, para comenzar con el análisis de comportamiento es necesario analizar los casos de uso desde el punto de vista de los sucesos o intercambios de información.

Retomemos el ejemplo del cajero automático, en el caso de uso el actor insertará su tarjeta en el cajero y el cajero validará el chip de la tarjeta en su base de datos; si la tarjeta no es válida, el cajero le pedirá al actor volver a insertar su tarjeta; si la tarjeta es válida, el cajero le solicitará al actor su NIP; el actor escribirá en el teclado su NIP de cuatro dígitos y el sistema validará el NIP con los almacenados en el sistema; si el NIP fuere incorrecto el sistema solicitará al actor volver a teclearlo; si es correcto, el sistema mostrará el menú de operaciones que puede realizar el actor.

En el texto anterior, se describen los sucesos o intercambios de información; cabe señalar que es necesario que se identifique a un actor con cada suceso; la información que se intercambia entre el actor y el sistema en cada suceso debe registrarse y deberán indicarse otras condiciones o restricciones que puedan presentarse. Cuando el actor ingresa su NIP se presenta un suceso, donde el actor intercambia información con el objeto cajero, el suceso puede identificarse como *ingreso de NIP;* aunque la información de los cuatro dígitos del NIP es importante, no es esencial para el análisis de comportamiento, sino solamente el suceso.

Dentro de los sucesos identificados es importante reconocer aquéllos que realizan un cambio o un control en el flujo de las acciones a través de la interacción, por ejemplo, el introducir el NIP es un flujo de información, que por sí solo no cambia el flujo de las acciones; en cambio, la respuesta del cajero al validar el NIP sí es un suceso que cambia el flujo, pues si fuere negativo, se realiza una acción, y si fuere positivo, otra.

Una vez identificados todos los sucesos, es necesario asociarlos a los objetos o clases que intervienen en el diagrama.

A continuación, es necesario representar los estados de los objetos y clases a partir de los sucesos.

Los estados en el enfoque orientado a objetos, tienen dos características:

1. El estado del objeto al momento en que el sistema ejecuta sus funciones.
2. El estado del sistema cuando se ejecutan sus funciones desde un punto de vista exterior.

En todo caso, los estados toman dos características básicas: la pasiva y la activa.

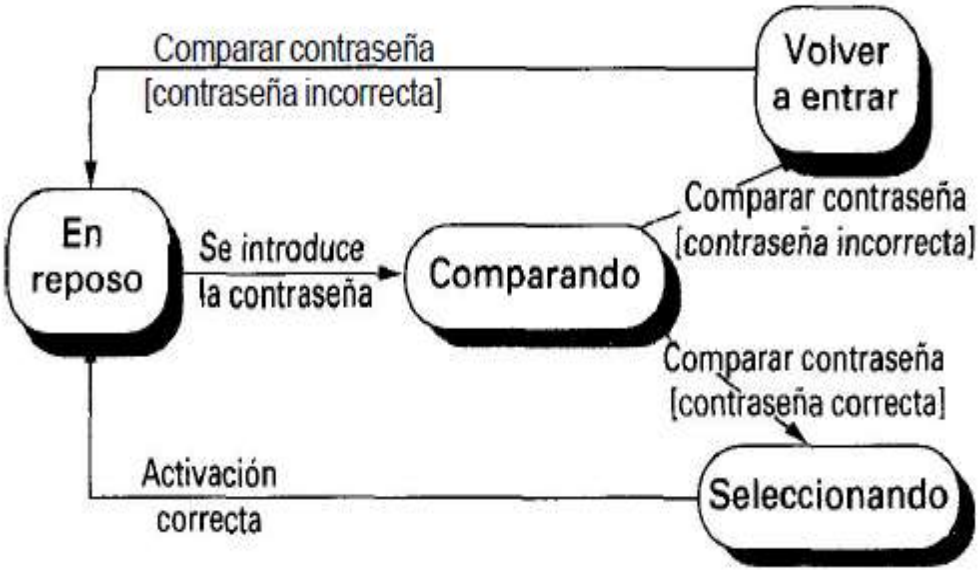
El estado pasivo no es otra cosa que el estado del objeto en el momento actual, en otras palabras, son las condiciones que presenta en el momento en que se le observa.

El estado activo se refiere a cuando el objeto comienza a realizar algún proceso, es decir, cuando comienzan a presentarse los sucesos.

Para que un objeto cambie de estado debe ocurrir un suceso que permita ese cambio, por ejemplo, el cajero realizará la validación del NIP hasta que el actor introduzca el mismo, la validación del NIP es un cambio en el estado del objeto cajero que estaba pasivo.

Un componente básico del modelo de comportamiento de un sistema es la creación de diagramas simples de los estados activos de los objetos y de los sucesos que provocan esos cambios, esto es, de pasivos a activos.

La siguiente figura muestra un ejemplo de un diagrama de estados.



Ejemplo de diagrama de estados

Fuente: Pressman, 2002: 376.

En el diagrama, las flechas representan los cambios de estados de un objeto a otro, las etiquetas identifican al suceso que dispara el cambio de estado, dentro de los diagramas de estado es posible identificar los estados que están asociados a una condición de tipo “si ocurre, entonces x”, de lo contrario y, como se ve en el estado *comparando* de la figura anterior.

El siguiente paso es la representación de la sucesión de acciones entre cada objeto o clase, lo cual veremos en el punto siguiente.

4.2.2 Identificación de interfaces de subsistemas con diagramas de clases

Una vez identificados, a partir de los casos de uso, los sucesos y estados de los objetos y clases, es necesario modelar su comportamiento para poder determinar qué entradas y salidas recibirá cada objeto, para ello emplearemos el diagrama de secuencia.

Un diagrama de secuencia representa la interacción que existe entre los diversos objetos de un sistema, permite a los analistas describir gráficamente la forma en que interactúan un grupo de objetos dentro de un sistema a través del paso del tiempo.

Los diagramas de secuencia trabajan como complemento de los diagramas de caso de uso, mientras que el caso de uso modela las interacciones desde el punto de vista del usuario y del negocio, los diagramas de secuencia permiten analizar detalles que permitirán la implementación del escenario en cuestión en el sistema. Cabe señalar que es necesario realizar un modelado de cada método de las clases involucradas.

Los conceptos necesarios para poder elaborar un diagrama de secuencia (UPC, 2008) son los siguientes:

- "Objetos: Se representan como rectángulos que contienen el nombre del objeto y el de su clase en el formato nombreObjeto: nombreClase.
- Línea de vida de un objeto (*lifeline*): La línea de vida de un objeto representa la vida del objeto durante la interacción. Se representa como una línea vertical punteada con un rectángulo de encabezado que representa el objeto y con rectángulos a través de la línea principal que denotan la ejecución de métodos (activación).
- Activación: Muestra el período en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto.
- Mensaje: El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje, hacia el objeto que lo ejecuta.
- Tiempos de transición: En un entorno de objetos concurrentes o de demoras en la recepción de mensajes, es útil agregar nombres a los tiempos de salida y llegada de mensajes".

Para poder elaborar un diagrama de secuencia se siguen los siguientes pasos:

1. Revisar los casos de uso para identificar los objetos necesarios para implementar el escenario representado en él.

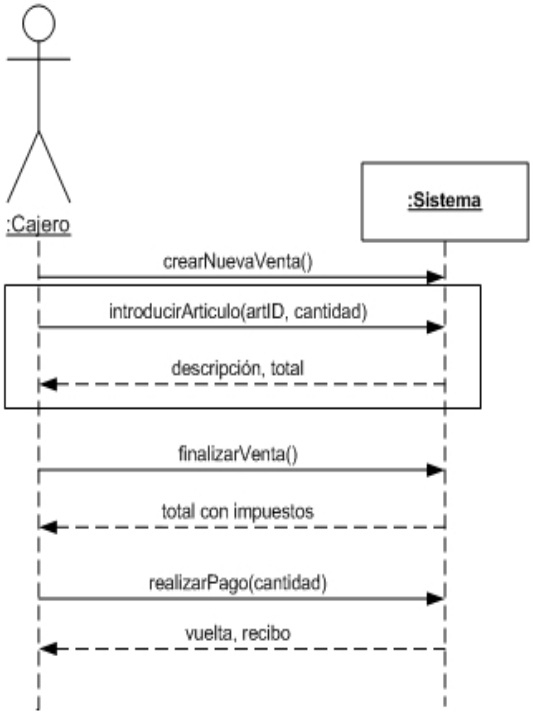
2. Los objetos serán representados en el diagrama con líneas verticales discontinuas, mientras que los mensajes se representarán con flechas horizontales.

3. Los mensajes deberán de estar representados de forma cronológica desde la parte superior a la inferior en el diagrama, la forma de distribuir los objetos queda a criterio del analista.

4. En el diseño inicial los diseñadores emplean la palabra "suceso o negocio" en la etiqueta de cada mensaje, una vez terminado el diseño preliminar se coloca el método llamado *perteneciente a la clase modelada.*

La siguiente figura muestra un ejemplo de un diagrama de secuencia.

Caso de uso *Procesar Venta*:



1. El Cliente llega a la caja.
2. El Cajero inicia una nueva venta.
3. El Cajero inserta el identificador del artículo.
4. El Sistema registra la línea de venta y presenta la descripción del artículo, precio y suma parcial.
5. El Cajero repite los pasos 3 y 4 hasta que se indique.
6. El Sistema muestra el total con los impuestos calculados.
7. El Cajero le dice al cliente el total y le pide que le pague.
8. El Cliente paga y el Sistema gestiona el pago.

Ejemplo de diagrama de secuencia a partir de un caso de uso

Fuente: UPC, 2008.

Una vez realizado el modelado de comportamiento a través de los diagramas de estado y de secuencia, el analista puede identificar con facilidad las interfaces necesarias a ser implementadas para la interacción de cada clase u objeto.

RESUMEN

Una de las fases más importantes en el desarrollo de los sistemas de información es la de análisis, dentro de ella se realiza la planificación y se mide el alcance de un sistema de información. Así mismo, a través del análisis podemos identificar las funciones del sistema, los datos de entrada y salida que deberá tener, además de ayudar a definir el entorno en el cual operará.

Dentro del análisis orientado a objetos, el objetivo principal es identificar los objetos y clases que serán empleados en el desarrollo del sistema para posteriormente modelar su comportamiento en cada función del sistema. Una de las herramientas más empleadas por los analistas orientados a objetos son los diagramas de caso de uso en una primera instancia para identificar a los objetos y las clases asociados a esos mismos objetos.

Los casos de uso se emplean de forma general para modelar la interacción de los usuarios con el sistema desde el punto de vista de los mismos usuarios, es decir, se describe la forma en cómo el usuario ve al sistema e interactúa con él. Y es a través de dichas interacciones que el analista procede a la identificación de los objetos y las clases, que una vez hecho lo anterior procede a realizar un modelo o diagrama de clases, responsabilidades y colaboraciones (CRC), donde se modela la forma en que las diversas clases se relacionan entre ellas y se identifica el tipo de relación existente.

Después de que se haya elaborado el modelo CRC se procede a volver a revisar los escenarios de los diagramas de casos de uso para identificar los sucesos, es decir, los puntos donde se realiza la transferencia de información entre los objetos y los usuarios, lo anterior permite al analista realizar un modelado de los estados de los objetos y a partir

de ello elaborar un diagrama de estados para comprender la forma en que el sistema deberá de realizar sus funciones.

Adicionalmente, a los diagramas de estados, los diagramas de secuencia son empelados para modelar las interacciones de las clases y objetos a través del tiempo, modelando la secuencia de mensajes que son generados en un escenario ayudando a modelar los métodos que serán empelados por cada objeto o clase.

Las herramientas como los casos de uso, diagramas de clases, modelos CRC, diagramas de estado y diagramas de secuencia, permiten a los analistas y desarrolladores comprender de forma detallada el comportamiento del sistema que va a ser construido y, en consecuencia, a definir de mejor forma los objetos y clases que serán empleados en la construcción y a determinar cuáles de estas clases y objetos podrán ser reutilizados en los diferentes módulos que conformarán al sistema.

BIBLIOGRAFÍA

SUGERIDA

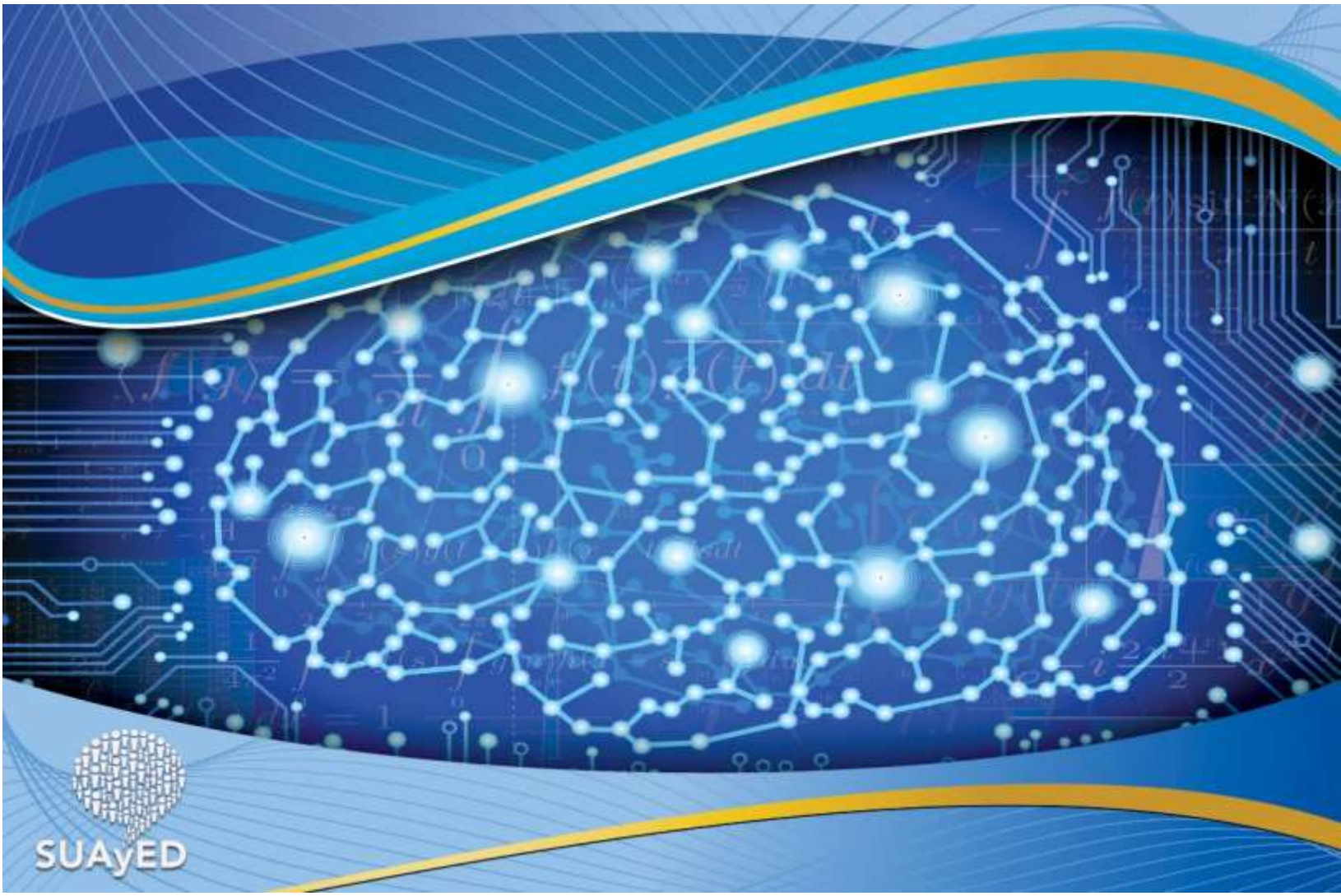
Autor	Capítulo	Páginas
Pressman	21	361-378
Sommerville	8	164-169

Pressman, R. S. (2002). *Ingeniería de software* (5ª. Ed.) México: Mc Graw-Hill.

Sommerville, I. (2001). *Ingeniería de software (*6a. Ed.) México: Addison Wesley.

Unidad 5.

Diseño orientado a objetos



OBJETIVO ESPECÍFICO

Especificar el comportamiento de los componentes del sistema y la persistencia de los datos.

TEMARIO DETALLADO
(8 horas)

5. Diseño orientado a objetos

- 5.1. Diseñar componentes
  - 5.1.1. Diseño de interfaces de usuario
  - 5.1.2. Refinación de flujos de eventos con diagramas de actividades y con diagramas de estados
  - 5.1.3. Unificación de clases y subsistemas con diagramas de clases
- 5.2. Diseñar base de datos
  - 5.2.1. Diseño de clases de persistencia de datos con diagramas de clases

INTRODUCCIÓN

La fase de diseño en cualquier paradigma de programación trata del modelado en sí del sistema y en desarrollar una representación conceptual de lo que será el sistema final.

Durante la fase de diseño, los desarrolladores identifican los flujos de información y las operaciones que el sistema debe realizar para poder representarlo de forma descriptiva en uno o varios diagramas. Dichos diagramas pueden variar dependiendo del paradigma empleado, pero en el diseño orientado a objetos podemos encontrar los diagramas de colaboración, diagramas de secuencias y de componentes.

A lo largo de la presente unidad veremos la forma en que se emplean dichos diagramas para el diseño de los sistemas orientados a objetos.

5. Diseño Orientado a Objetos

El diseño orientado a objetos proporciona a los desarrolladores de software un anteproyecto para la construcción de un sistema o software, a diferencia de los enfoques de diseño tradicionales el diseño orientado a objetos proporciona diferentes modelos de modularidad, es decir, organiza al sistema en subsistemas o módulos que pueden ser construidos de forma individual hasta completar el sistema completo.

Dentro del diseño orientado a objetos, los datos y las operaciones realizadas con ellos son encapsulados en objetos, que son los bloques básicos de este tipo de diseño. Adicionalmente, debe describirse la forma como se organizan los datos y debe detallar las operaciones relacionadas con ellos.

De acuerdo con Pressman (202: 380), es posible definir el diseño orientado a objetos en cuatro capas:

- **La capa subsistema.** Esta capa contiene a todos los subsistemas definidos en la fase de análisis, lo que permite que el software, o el sistema a desarrollar, implemente los requerimientos definidos por el cliente y dé soporte a los mismos.

- **La capa de clases y objetos.** Contiene cada una de las clases identificadas durante el análisis organizadas de forma jerárquica, lo que permite al desarrollador crear el sistema empleando generalizaciones de clases y objetos y especificaciones más detalladas.

- **La capa de mensajes.** Esta capa pormenoriza los detalles de la forma en que los objetos se comunican entre ellos y establecen las colaboraciones, lo que permite al desarrollador establecer las interfaces internas y externas del sistema.

- **La capa de responsabilidades.** Esta capa contiene los algoritmos y estructuras de datos necesarios para definir los atributos y operaciones asociadas a cada objeto.

Cada una de las capas requiere de un correcto análisis de requerimientos, comenzando con el diseño de los subsistemas que son definidos a partir de los requerimientos del cliente (definidos a través de los casos de uso) y del comportamiento descrito por ellos. La capa de clases y objetos son obtenidos del modelo CRC estudiado en la unidad anterior, que define las responsabilidades y colaboraciones de cada objeto. La capa de mensajes se define a partir del modelo objeto-relacional y, finalmente, la capa de responsabilidades se define a partir de las operaciones y colaboraciones también definidas en el modelo CRC.

Pressman (2002: 383) nos dice que para poder realizar un diseño adecuado orientado a objetos se deben seguir las siguientes etapas:

1. “Describir cada subsistema y asignar a procesadores y tareas.
2. Elegir una estrategia para implementar la administración de datos, soporte de interfaz y administración de tareas.
3. Diseñar un mecanismo de control para el sistema apropiado.
4. Diseñar objetos creando una representación *procedural* para cada operación, y estructuras de datos para los atributos de clase.
5. Diseñar mensajes, usando la colaboración entre objetos y relaciones.
6. Crear el modelo de mensajería.
7. Revisar el modelo de diseño y renovarlo cada vez que se requiera”.

5.1 Diseñar Componentes

El diseño de componentes se basa en el diseño de los objetos, es decir, en la descripción de los objetos del sistema y su interacción entre ellos, dentro de esta etapa se detallan las estructuras de datos, los atributos y operaciones de cada objeto.

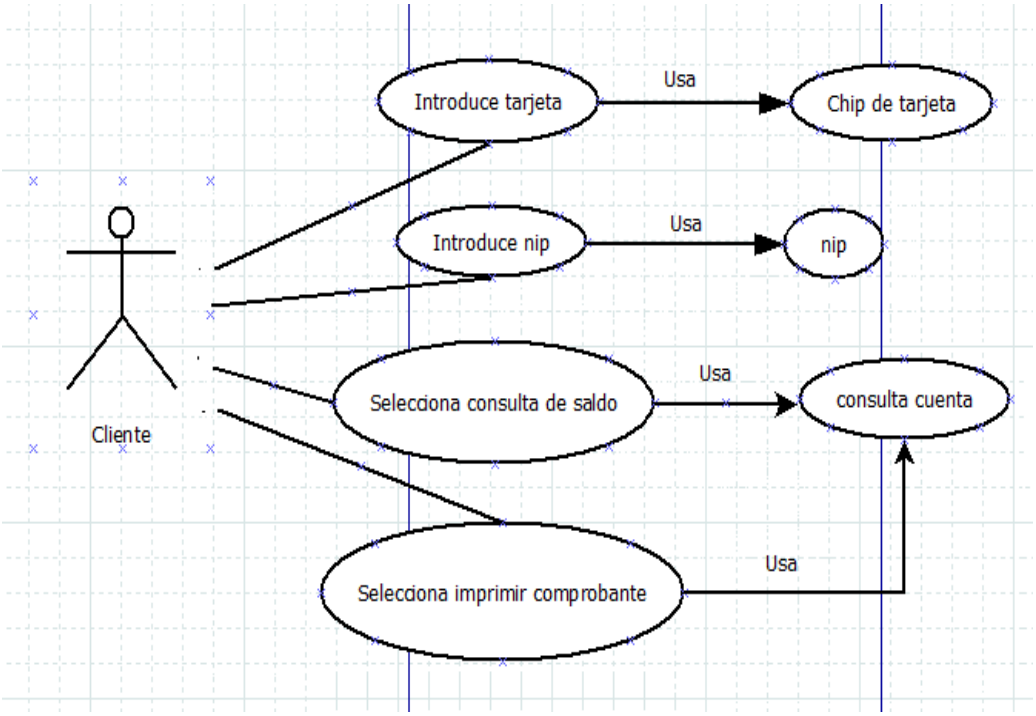
El diseño de objetos puede tomar cualquiera de los siguientes caminos:

1. *Una descripción de protocolo*, que define cada mensaje que el objeto recibe y las operaciones que realiza con dicho mensaje, en otras palabras, precisa la interfaz del objeto.
2. *Una descripción de implementación,* que describe la forma de implementar cada una de las operaciones realizadas por un objeto cuando recibe un mensaje (Pressman, 2002: 388).

5.1.1 Diseño de interfaces de usuario

Las interfaces de usuario en sí representan los subsistemas de mayor importancia dentro del sistema, ya que es a través de ellas que los usuarios podrán interactuar con él y podrán desencadenarse las acciones internas del mismo.

Dentro del diseño orientado a objetos, los diagramas de casos de uso son los primeros referentes para identificar dichas interfaces. En los diagramas de caso de uso los usuarios se simbolizan por medio de los actores, los cuales representan los roles que los usuarios juegan en la interacción con el sistema, y dependiendo de dicho rol se comienza a definir el diseño de la interfaz del usuario.



Escenario de casos de uso del cajero con mayor nivel de detalle

Fuente: Pressman, 2002: 383.

En la figura anterior podemos observar que el actor (cliente) requiere de una interfaz que le permita validar su tarjeta, su NIP y realizar una consulta de saldo. Estas acciones permiten al desarrollador identificar las opciones que debe tener la interfaz.

Una vez que se definen los actores y las situaciones en que cada actor interactúa con el sistema se debe proceder a identificar las operaciones que intervienen en dicha interacción. Una vez identificadas se jerarquizan y ordenan de acuerdo a su importancia, lo que al final definirá el menú de opciones de la interfaz.

De esta forma, siguiendo con el ejemplo, podemos ver que una de las pantallas principales debe ser la de ingreso de NIP y posteriormente la pantalla de operaciones que el cliente podrá realizar, comenzando con la de consulta de saldo.

En la mayoría de los lenguajes de programación orientados a objetos ya existen clases que representan a estas pantallas, por lo que los desarrolladores pueden ahorrar tiempo empleando dichas clases.

5.1.2 Refinación de flujos de eventos con diagramas de actividades y con diagramas de estados

Una vez que son identificados los objetos es necesario comenzar a refinar las funciones y las interacciones de cada uno de ellos, algunas de las herramientas en el diseño orientado a objetos que nos ayudan a realizar lo anterior son los diagramas de actividad y los diagramas de estado.

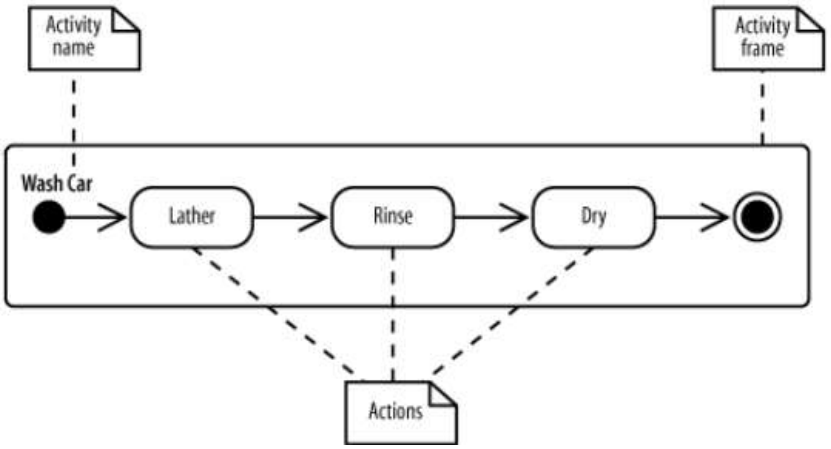
Los diagramas de actividad describen la forma en que el sistema realiza sus funciones, modelando el comportamiento del sistema, describiendo las funciones y transacciones del sistema enfocándose en la forma en que cada una se lleva a cabo.

Los diagramas de actividad heredan características de los diagramas de flujo de datos y de los diagramas de estado empleados en el diseño de sistemas tradicionales, pero permitiendo profundizar de mejor manera en las funciones que realiza cada objeto.

El diagrama de actividad tiene los siguientes elementos:

Acción. Es un paso de un proceso o función, que debe ser ejecutado hasta ser completado.

Actividad. Se trata de un conjunto de acciones que permiten realizar un proceso o función completa.



Ejemplo de diagrama de actividades con acciones y actividades

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

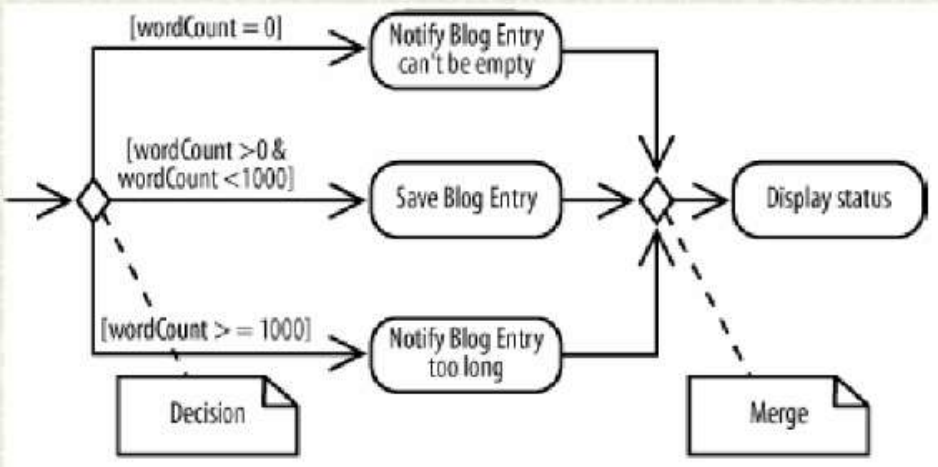
En la figura anterior podemos ver la forma como se representa a la actividad lavar carro, donde las acciones necesarias para realizar dichas actividades son tres: enjabonar, enjuagar y secar. Las acciones se representan por pequeños óvalos, mientras que el comienzo y final de una actividad se representa por los puntos simples y con un punto con una circunferencia exterior; la actividad completa se encierra en un rectángulo.

Decisiones. Representan puntos de una secuencia de acciones dentro de una actividad donde es posible tomar varios caminos posibles para realizar dicha acción. Las decisiones deben ser libres de ambigüedades y lo más concretas posibles.

Ramas. Representan diversas opciones que pueden ser tomadas para poder realizar una actividad.

Convergencia. Punto donde cada una de las ramas generadas a partir de una decisión coincide para continuar en un flujo de acciones comunes a dichas ramas.

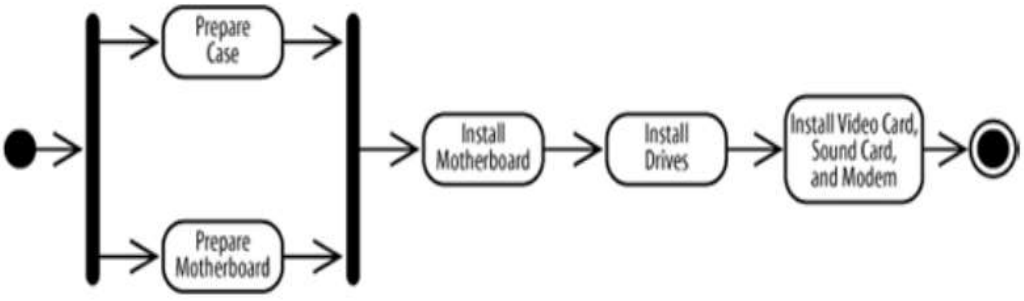
En la figura siguiente se ejemplifican estos elementos.



Ejemplo de ramificación en un diagrama de actividad

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

Fork y Joint. Representan líneas de acciones que pueden ser ejecutadas de forma paralela o concurrente, es decir, simbolizan acciones que se ejecutan al mismo tiempo y tienen como salida una serie de acciones comunes.

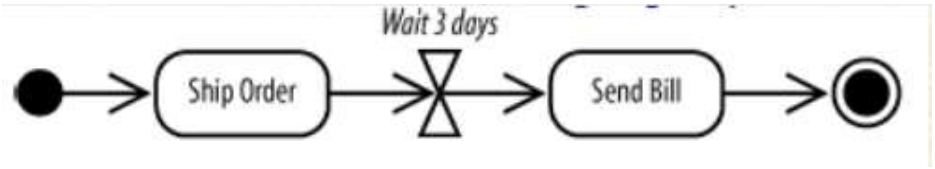


Ejemplo de Fork y Joint.

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

En la figura anterior se representa un ejemplo del Fork y Join, donde, en la actividad de ensamblado de una PC, las acciones *prepare case* y *prepare motherboard* se realizan de forma simultánea (Fork), siendo la acción consecuente de ambas la instalación de la motherboard (Join).

Eventos de tiempo. Los eventos de tiempo son representaciones de pausas, esperas o retrasos en una actividad, que en los diagramas se representa empleando un símbolo semejante a un reloj de arena para indicar estos eventos.

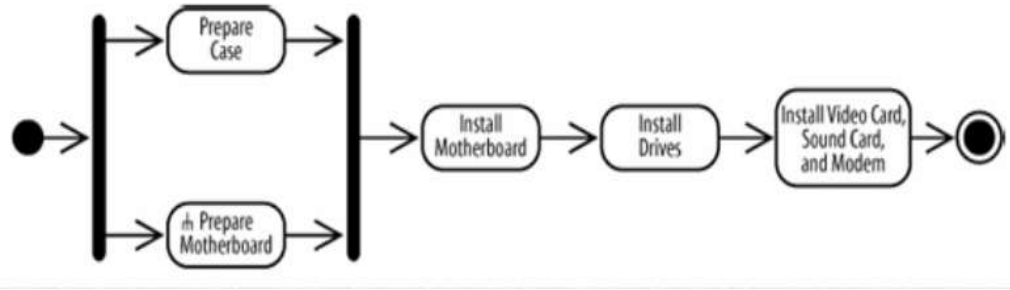


Ejemplo de un evento de tiempo

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

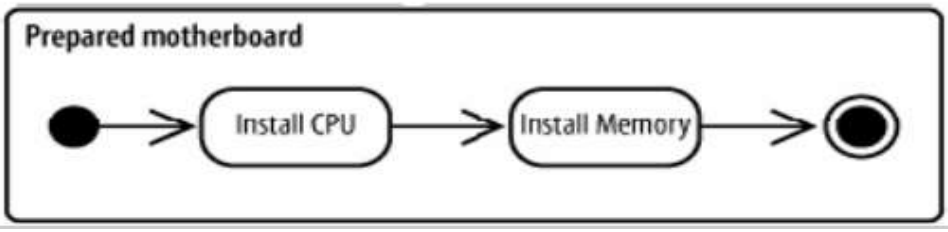
En la figura anterior podemos ver que la actividad de enviar un pedido comienza con la acción enviar orden, posteriormente se tiene una espera de tres días y finalmente se realiza la actividad de envío de la cuenta, el evento de espera se ve representado con el reloj de arena.

Llamado a otras actividades. Dentro de un diagrama de actividades es posible indicar dentro de una acción cuando es necesario realizar otra actividad que es complementaria, esto se representa colocando un símbolo de tridente dentro de la acción.



Ejemplo de un llamado a una actividad complementaria

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

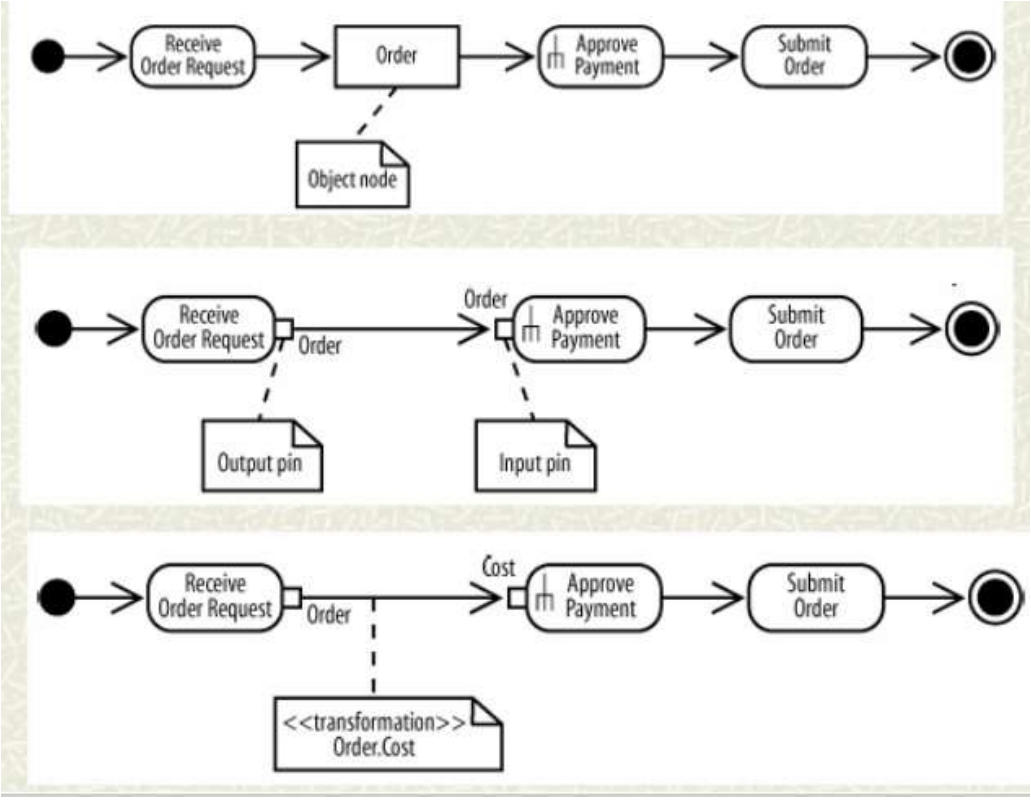


Actividad complementaria llamada en el ejemplo anterior

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

En el ejemplo del ensamblado de la PC, la acción preparar *motherboard* llama a una actividad complementaria que lleva el nombre de la acción, en esta actividad se realizan las acciones de instalar el CPU y la memoria.

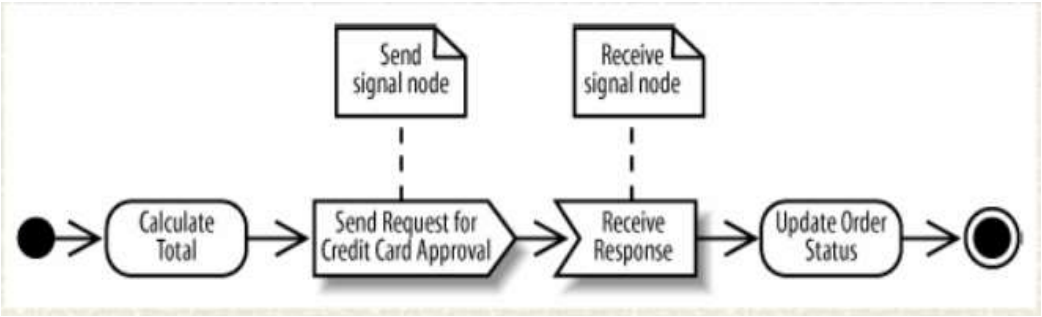
Objetos. Los objetos en los diagramas de actividad se pueden representar de varias formas: la primera, representando a los objetos como parte del flujo de actividades donde los datos forman parte de dicho flujo y del objeto. La segunda, representando al objeto como un pequeño cuadro pegado a una acción, representan a los objetos como entrada o salida de datos para dicha acción.



Representación de objetos en un diagrama de actividad

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

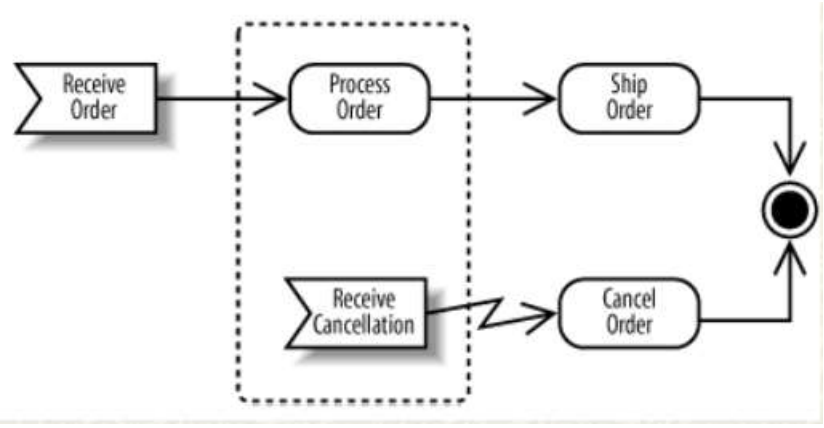
Envío y recepción de señales. Representan entradas o salidas de datos de la actividad hacia entidades externas que pueden ser otros procesos o funciones u otros sistemas. En el diagrama de estados, las acciones que envían señales se presentan con un rectángulo con una esquina doblada en la parte superior izquierda.



Ejemplo de acciones con envío y recepción de datos. Fuente: *ídem*.

En la figura anterior que representa a la actividad de pago de un pedido, donde las acciones de aprobación de tarjeta de crédito envía una señal hacia una entidad externa para aprobarla y la siguiente acción recibe la respuesta de dicha entidad.

Interrupción de actividades. Existen algunas actividades que pueden interrumpir el flujo de las acciones que la conforman dependiendo de ciertas condiciones, generalmente debido a la recepción de alguna señal que propicie dicha cancelación. Las cancelaciones se representan por medio de flechas con líneas en zigzag.

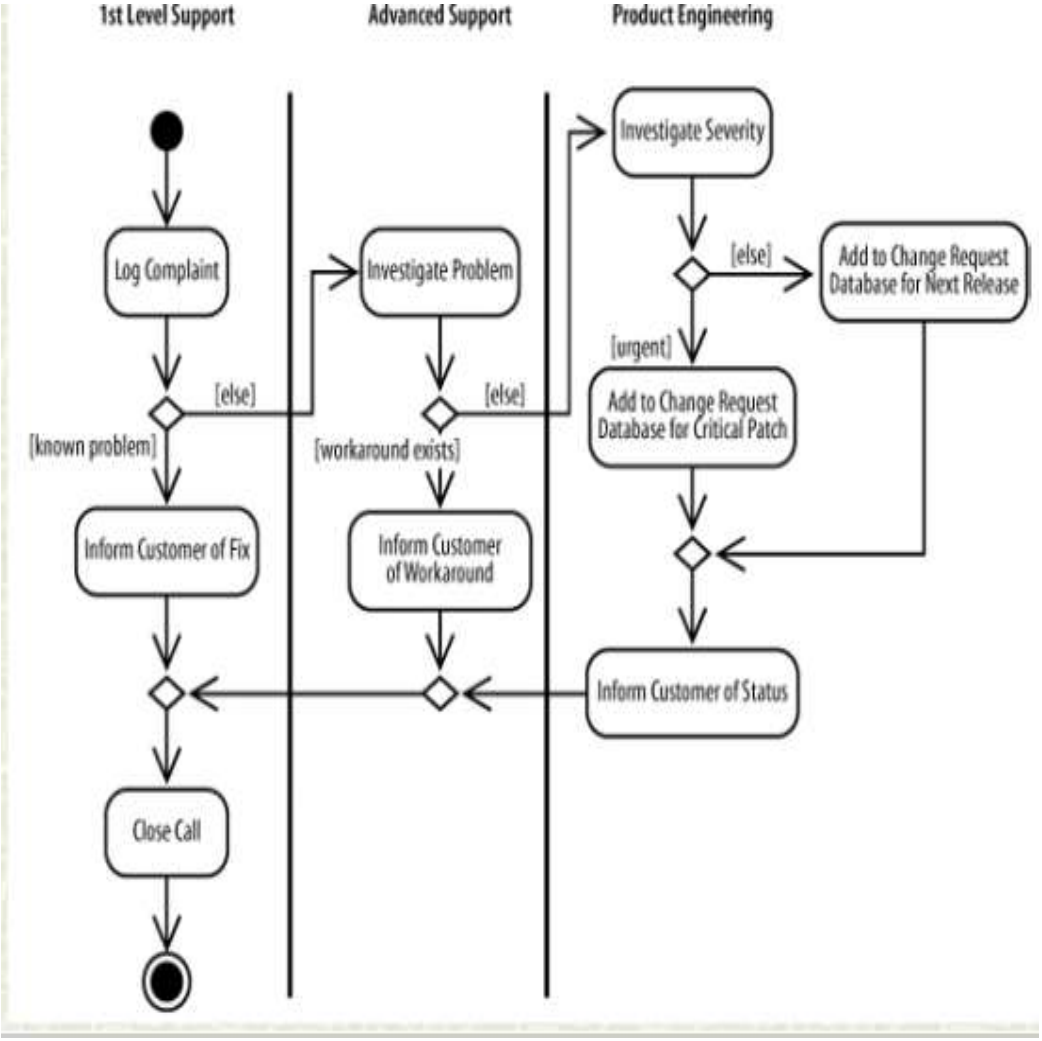


Ejemplo de cancelación de actividad

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

En la figura anterior la actividad de proceso de una orden de compra se ve interrumpida por la acción de recepción de una señal de cancelación del pedido.

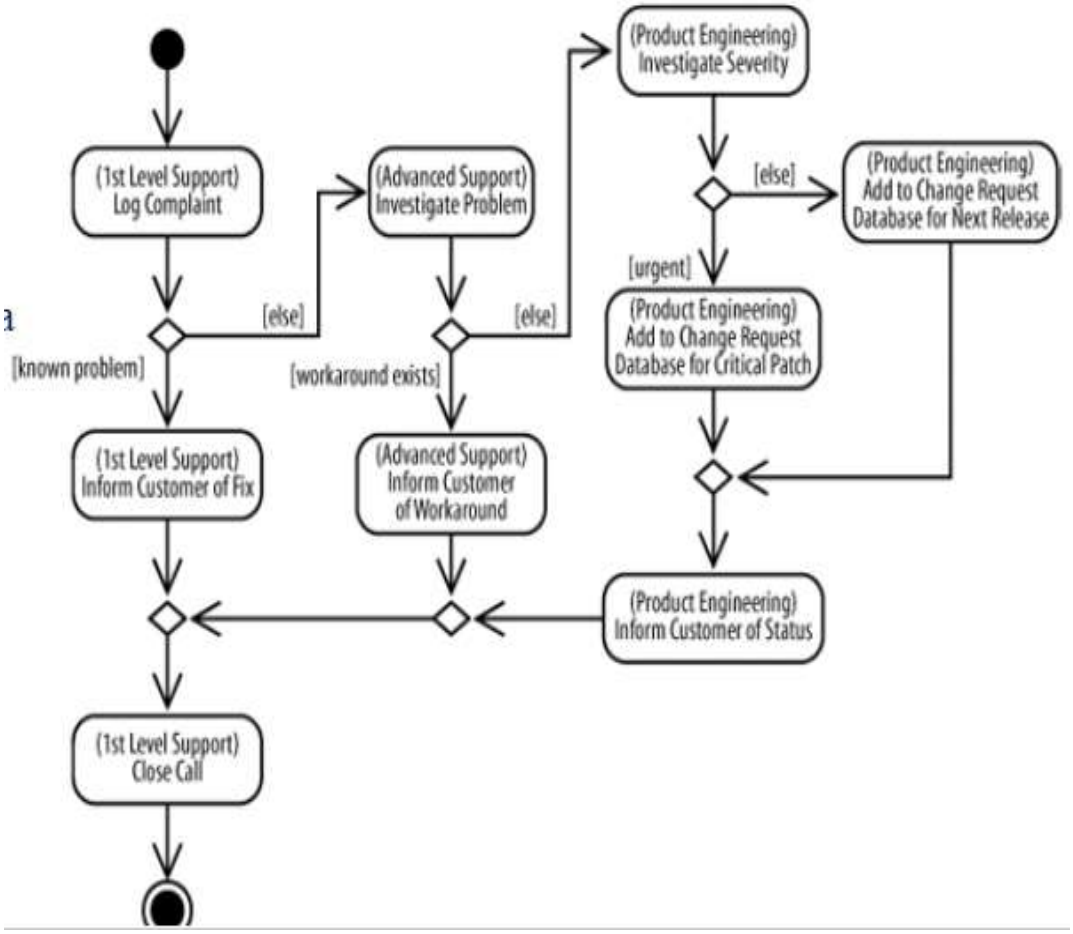
Particiones. Las particiones son líneas verticales paralelas que representan diversos procesos involucrados en una actividad.



Ejemplo de uso de particiones en la representación de procesos

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

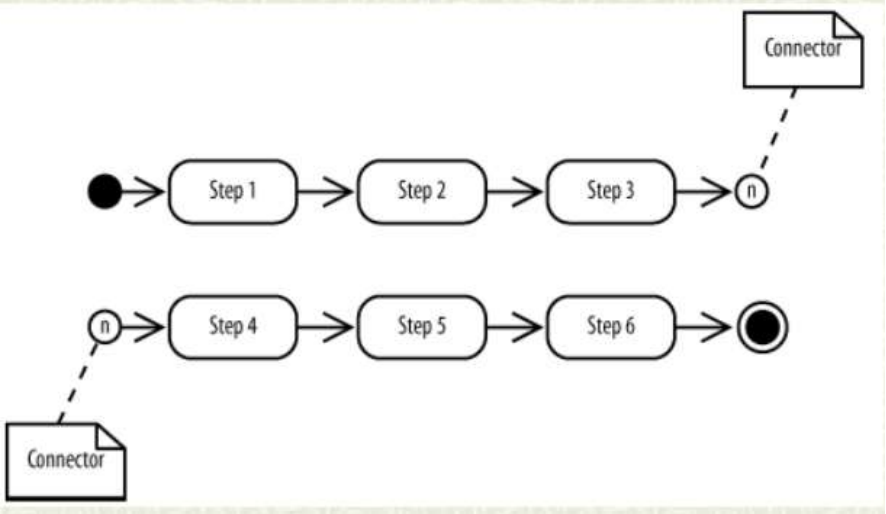
Otra forma de representar los procesos responsables de cada acción es mediante etiquetas dentro de cada acción con el nombre de cada proceso entre paréntesis.



Ejemplo empleando etiquetas para identificar procesos

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

Conectores. Un conector es empleado cuando los diagramas de actividad son muy grandes y requieren de varias páginas para poder representar las actividades y acciones que lo componen. Los conectores se representan por medio de pequeños círculos con un identificador en su interior que ayuda a su seguimiento.



Ejemplo de uso de conectores

Fuente: Drake (s. f.). *Diagramas de actividad y diagramas de estados.*

Los diagramas de actividad ayudan a los desarrolladores a entender con mayor detalle cada función del sistema y, por tanto, a modelarlo de mejor manera para su correcta implementación.

La otra herramienta empleada en el diseño orientado a objetos es el diagrama de estados, del cual hablamos en la unidad anterior.

Recordemos rápidamente las características de los estados en el enfoque orientado a objetos (Pressman, 2002: 376).

3. El estado del objeto al momento en que el sistema ejecuta sus funciones.
4. El estado del sistema cuando se ejecutan sus funciones desde un punto de vista exterior.

Los diagramas de estado al representar las acciones en cada objeto y los eventos que ejecutan cada acción ayudan a los desarrolladores a entender y especificar cada una de las operaciones asociadas al conjunto de objetos y los eventos que ejecutan cada operación.

5.1.3 Unificación de clases y subsistemas con diagramas de clases

Recordemos lo que es un subsistema: “es un subconjunto de clases que colaboran entre sí para poder realizar un conjunto de responsabilidades asociadas” (Pressman, 2002: 372). En este sentido, las clases y subsistemas están relacionados de forma íntima, por así decirlo.

En el proceso de diseño orientado a objetos los subsistemas agrupan a las clases de acuerdo a su función para poder realizar las funciones del subsistema asociado, pero adicionalmente es necesario clasificar a los subsistemas de forma que realicen sus funciones de forma priorizada o jerárquica, lo que permite al desarrollador organizar al sistema de una forma ascendente o descendente para mejorar su comprensión.

Cabe señalar, que los subsistemas también pueden ser conformados de otros subsistemas, por lo que es posible definir subsistemas de bajo nivel o de servicio que proporcionan funciones opcionales para el subsistema que los agrupa.

Cuando se diseña un sistema de forma ascendente se comienza por la agrupación de las clases en subsistemas perfectamente definidos y posteriormente en la agrupación de los subsistemas en otros subsistemas, si es necesario, hasta conformar el sistema completo.

En el enfoque descendente se comienza por la definición de los subsistemas de mayor nivel, desglosando cada subsistema superior en inferiores, hasta llegar a definir las clases a partir de los subsistemas de bajo nivel.

El siguiente es un ejemplo de cómo se separa un sistema de cajero automático en tres subsistemas:

Ej.: Para el sistema de CA se agrupan las clases en tres subsistemas:

- <<subsystem>> Interfaz del CA: agrupa todas las clases que proporcionan la interfaz gráfica del CA:
  - Lector de tarjetas
  - Dispositivo de visualización
  - Teclado
  - Alimentador de la salida
  - Sensor de la salida
  - Contador de efectivo
  - **Gestor de cliente**
- <<subsystem>> Gestión de transacciones
  - **Gestión de Transacciones**
  - <<service subsystem>> Gestión de retirada de efectivo
    - Retirada de efectivo
- <<subsystem>> Gestión de cuentas
  - Clase Persistente
  - **Gestor de Cuentas**
  - Cuenta

Ejemplo de creación de subsistemas.

Fuente: Torrossi, (s. f.). *El Proceso unificado de desarrollo de software.*

En el ejemplo podemos observar que cada subsistema agrupa a varios objetos o clases, lo cual facilita su representación en un diagrama de clases para su asociación.

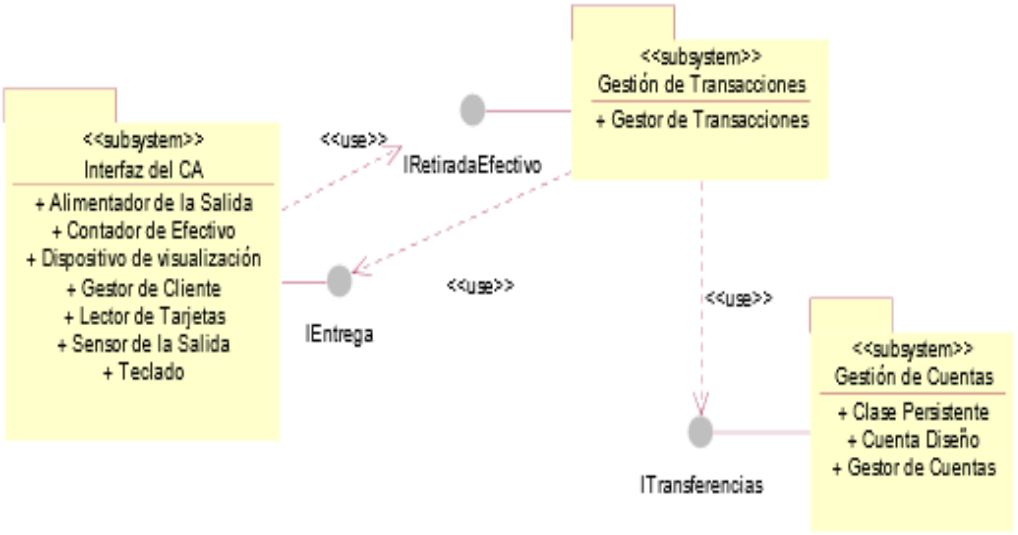


Diagrama de clases representando el ejemplo del cajero automático.

Fuente: Torrossi, (s. f.). *El Proceso unificado de desarrollo de software.*

5.2 Diseñar Base De Datos

Dentro del paradigma orientado a objetos el diseño de base de datos ha encontrado un híbrido entre las bases de datos relacionales y las bases de datos orientadas a objetos, este nuevo modelo se denomina objeto-relacional y es el más empleado en la actualidad para el diseño de bases de datos de sistemas orientados a objetos.

La idea principal de las bases de datos objeto-relacional es permitir a los usuarios crear sus propios tipos de datos (objetos), así como definir los métodos (operaciones) que pueden ser realizados sobre esos datos, lo que admite la implementación de funciones que se adecuen más a las necesidades de los usuarios y también facilita el rehusó de dichas funciones cuando sea necesario.

Los sistemas manejadores de base de datos objeto-relacional permiten el manejo de datos complejos de una forma más sencilla para los usuarios, consintiendo también almacenar parte del sistema diseñado directamente en la base de datos, liberando de carga de trabajo al sistema principal haciendo más ágiles las operaciones.

Otra de las ventajas que ofrecen las bases de datos objeto-relacional es que permiten migrar la información de bases de datos relacionales sin mayores problemas, ya que el modelo en sí hereda características del modelo relacional.

Uno de los sistemas administradores de bases de datos objeto-relacional más empleado en la actualidad es Oracle 8.

Oracle es un sistema manejador de base de datos desarrollado por *Oracle Corporation*, (Oracle, s. f.). La versión de Oracle 8 permite a los diseñadores de base de datos crear y

administrar bases de datos relaciones y objeto-relacional, lo que lo convierte en uno de los sistemas manejadores de base de datos más robustos en el mercado.

5.2.1 Diseño de clases de persistencia de datos con diagramas de clases.

Cuando se diseñan bases de datos orientadas a objetos los usuarios pueden definir sus propios tipos de datos (en el modelo orientado a objetos estos datos son los objetos y clases). Derivados de los diagramas de casos de uso, **secuencia, actividad, estados y de Clases-Responsabilidades-Colaboraciones** (CRC) vistos en la unidad anterior, se identifican las clases que son más empleadas en el sistema o las que denominaremos críticas, estas clases son las que en un momento determinado se desea modelar y almacenar en una base de datos.

En las bases de datos objeto-relacional, un tipo de dato (clase), debe definir el comportamiento y la estructura común para un conjunto de datos (objetos) que serán empleados en el sistema a desarrollar (recordemos que las clases son las estructuras que permiten generar objetos que heredan sus características, lo mismo pasa en las bases de datos, donde lo que se almacena es la clase que nos permitirá crear nuevos objetos).

En las bases de datos objeto-relacional los tipos de datos definen objetos, y los objetos son una representación de una entidad del mundo real que deben tener los siguientes elementos:

- **Nombre**. Que es el identificador del objeto.
- **Atributos**. Que definen la estructura y los tipos de datos del objeto.
- **Métodos**. Que definen las operaciones o funciones que pueden ser realizadas con el objeto.

Estos tipos de datos servirán como la base o plantilla generar nuevos objetos con las características del tipo de dato, lo que denominamos herencia.

DEFINICIÓN ORIENTADA A OBJETOS

```
define type Direccion_T:
tuple [calle:string,
       ciudad:string,
       prov:string,
       codpos:string]

define class Cliente_T
type tuple [clinum: integer,
       clinomb:string,
       direccion:Direccion_T,
       telefono: string,
       fecha-nac:date]
operations edad():integer
```

DEFINICIÓN EN ORACLE

```
CREATE TYPE direccion_t AS OBJECT (
       calle VARCHAR2(200),
       ciudad VARCHAR2(200),
       prov CHAR(2),
       codpos VARCHAR2(20));

CREATE TYPE cliente_t AS OBJECT (
       clinum NUMBER,
       clinomb VARCHAR2(200),
       direccion direccion_t,
       telefono VARCHAR2(20),
       fecha_nac DATE,
MEMBER FUNCTION edad RETURN NUMBER,
PRAGMA
RESTRICT_REFERENCES(edad,WNDS));
```

Ejemplo de definición de datos.

Fuente: Bases de Datos Objeto-Relacionales, (s. f.).

En el ejemplo anterior podemos observar la forma que se declara una clase en un lenguaje orientado a objetos y la forma como se hace en una base de datos objeto-relacional (en este caso Oracle).

La especificación de los métodos asociada a cada objeto se debe realizar al momento de finir el objeto mismo. En el ejemplo anterior, en el caso del lenguaje orientado a objetos, vemos directamente la declaración *operations,* mientras que en el caso de la base de datos se define en la parte de *PRAGMA RESTRICT_REFERENCES*, que le indica a la base de datos que se trata de uno de los métodos (operación) que puede ser realizada con el objeto (dato) que se está creando, restringiendo su uso para evitar la manipulación de la base de datos por parte de los métodos o las variables externas.

Como ya mencionamos, las clases son los generadores de nuevos objetos de ese mismo tipo, así en el caso de las bases de datos objeto-relacional los tipos de objetos definidos por los usuarios permiten la creación de nuevos objetos del mismo tipo, para ello solo es

necesario crear el tipo de dato y ejecutarlo pasando los parámetros asociados, con lo que crearemos objetos que heredaran el tipo de dato y sus métodos.

```
direccion_t('Avenida Sagunto', 'Puzol', 'Valencia', 'E-23523')
cliente_t( 2347,
        'Juan Pérez Ruíz',
        direccion_t('Calle Eo', 'Onda', 'Castellón',
                    '34568'),
            '696-779789',
            12/12/1981)
```

Ejemplo de creación de un objeto a partir de un tipo de dato de usuario.
Fuente: Bases de Datos Objeto-Relacionales, (s. f.).

En el ejemplo anterior, se toma el tipo de dato creado en el primer modelo y se pasan los valores que definirán al nuevo objeto. Si realizamos la analogía con las bases de datos relacionales sería el equivalente de la inserción de un registro en una tabla, con la diferencia de que el objeto puede ser manipulado de manera más eficiente a través de sus métodos, lo que no permite el registro de una tabla.

Una de las ventajas de las bases de datos objeto-relacional sobre las bases de datos relacionales es que una vez que se define un tipo de dato por parte del usuario, dicho dato puede ser empleado para crear otros tipos de datos, como es el caso de tablas que almacenen objetos del mismo tipo o para definir los atributos de una tabla.

En el caso de la tabla “objetos”, se trata de una tabla de la base de datos que en lugar de almacenar registros simples como en el caso de las bases de datos relacionales, puede almacenar en cada registro un objeto del mismo tipo.

Las tablas que tienen atributos o campos de tipo objeto son tablas simples, pero tienen la posibilidad de almacenar datos complejos dentro de sus campos, con la diferencia de que estos atributos no son vistos como objetos.

La siguiente imagen nos muestra un ejemplo de cada una de estas tablas.

```
CREATE TABLE clientes_año_tab OF cliente_t
        (clinum PRIMARY KEY);
CREATE TABLE clientes_antiguos_tab (
        año NUMBER,
        cliente cliente_t);
```

Ejemplo de tablas objeto y tablas con campos tipo objeto.

Fuente: Bases de Datos Objeto-Relacionales, (s. f.).

La primera tabla es del tipo que almacena objetos, mientras que la segunda es del tipo de dato complejo.

Los tipos de datos definidos por los usuarios son los derivados de las clases y objetos identificados y definidos en la fase de análisis y diseño orientados a objetos.

RESUMEN

En el enfoque de programación orientado a objetos existen diversas herramientas que permiten a los desarrolladores realizar el análisis y diseño de un sistema, los diagramas de casos de uso son la base de todo el diseño orientado a objetos. A través de ellos podemos definir las interfaces que deberá llevar el sistema y definir su comportamiento general, que a partir de ahí, se continúa con la identificación de las clases y los objetos que compondrán al sistema a desarrollar, al igual que sus relaciones y la forma en que colaboran entre sí.

El diseño orientado a objetos tienen el objetivo de establecer un marco de trabajo o anteproyecto que sirva como la base del desarrollo subsecuente del sistema, siempre trabajando bajo un enfoque iterativo, donde las fases de análisis y diseño son refinadas en cada repetición para ayudar a refinar los conceptos obtenidos de la fase de análisis. En el diseño orientado a objetos se emplean diagramas de actividades que permiten a los desarrolladores identificar los procesos o funciones específicas del sistema, clasificarlas y asociarlas con los objetos ya definidos, de tal forma que el diagrama nos da una idea más a detalle de las funciones del sistema en general.

Otra herramienta de utilidad son los diagramas de estado, donde es posible identificar las operaciones realizadas en cada uno de los objetos y los eventos que permiten realizar dichas operaciones, ya que de esta manera es posible determinar las estructuras de datos y las funciones que serán asociadas a cada objeto.

Finalmente, es posible emplear bases de datos objeto-relacional para complementar a los sistemas, porque estas bases de datos en particular heredan características de aquellas que son relacionales y orientadas a objetos, dando mayor flexibilidad a los

desarrolladores en crear objetos (tipos de datos) que podrán ser empleados en los sistemas y que se encuentren almacenados y ejecutados dentro del mismo servidor de la base de dato, liberando carga de trabajo por parte del sistema e induciéndolo a ser más eficiente.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Pressman (2002)	22	379-407
Sommerville (2001)	14	285-304

Pressman, R. S. (2002). *Ingeniería de software* (5ª. ed.) México: Mc. Graw-Hill.

Sommerville, I. (2001). *Ingeniería de software (*6a. ed.) México: Addison Wesley.

Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia